

Libraries Guide

ISE 8.1i



Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

Copyright © 1995-2005 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

About This Guide

The Libraries Guide is part of the ISE documentation collection.

Guide Contents

This guide contains the following:

- Discussion of the [Xilinx Unified Libraries](#)
- [Slice Count](#) information for FPGAs
- Design elements associated with the following architectures are described in this guide.
 - ◆ Spartan™-II
 - ◆ Spartan™-IIE
 - ◆ Spartan™-3
 - ◆ Virtex™
 - ◆ Virtex™-E
 - ◆ Virtex™-II
 - ◆ Virtex™-II Pro
 - ◆ Virtex™-II Pro X
 - ◆ XC9500™
 - ◆ XC9500XV™
 - ◆ XC9500XL™
 - ◆ CoolRunner™ XPLA3
 - ◆ CoolRunner™-II
- A listing of the various [Functional Categories](#) of design elements
- Individual sections for each of the [Design Elements](#).

Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/literature>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support>.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild <i>design_name</i>
Helvetica bold	Commands that you select from a menu	File →Open
	Keyboard shortcuts	Ctrl+C
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	ngdbuild <i>design_name</i>
	References to other manuals	See the <i>Development System Reference Guide</i> for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7:0] , they are required.	ngdbuild [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = {on off}
Vertical bar	Separates items in a list of choices	lowpwr = {on off}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	allow block <i>block_name loc1 loc2 ... locn;</i>

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section " Additional Resources " for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the <i>Virtex-II Handbook</i> .
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Functional Categories

The functional categories list the available design elements in each category along with a brief description of each element that is supported under each Xilinx architecture.

Design Elements

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

The following information is provided for each library element, where applicable:

- Name of the element
- Graphic symbol (only in the schematic version of the guide)
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table
- VHDL and Verilog instantiation and inference code (only in the HDL version of the guide)

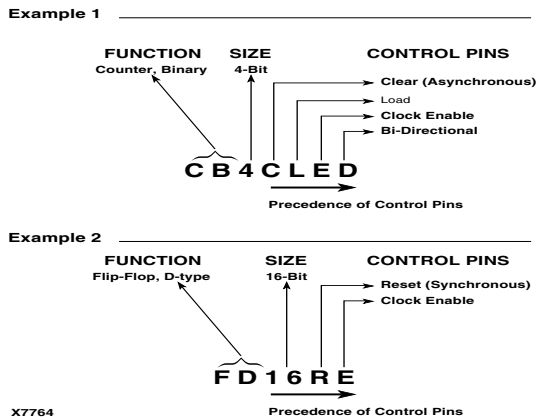
Schematic Examples

Schematics are included for each device library, if the implementation differs. These schematics are illustrated only in the schematic user version of the libraries guide for each Xilinx architecture.

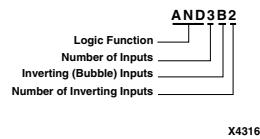
Design elements with bussed or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic -- generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

Naming Conventions

Examples of the general naming conventions for the unified library elements are shown in the following figures.



Naming Conventions



Combinatorial Naming Conventions

Attributes and Constraints

The terms attribute and constraint have been used interchangeably by some in the engineering community, while others ascribe different meanings to these terms. In addition, language constructs use the terms attribute and directive in similar yet different senses. For the purpose of clarification, the Xilinx documentation refers to the terms attributes and constraints as defined below.

About This Guide

Guide Contents	3
Additional Resources	3
Conventions	4
Functional Categories	5
Design Elements	5
Schematic Examples	5
Naming Conventions	5
Attributes and Constraints	6

Xilinx Unified Libraries

Overview	17
Applicable Architectures	18
Functional Categories	18
Design Elements	18
Schematic Examples	18
Naming Conventions	19
Attributes and Constraints	19
Carry Logic	19
Flip-Flop, Counter, and Register Performance	20

Slice Count

About Configurable Logic Blocks (CLBs)	23
Slice Count for FPGA Components	23

Functional Categories

Arithmetic Functions	38
Buffers	39
Comparators	40
Counters	40
Decoders	46
Edge Decoders	47
Flip-Flops	47
General	51
Input Latches	54
Input/Output Flip-Flops	55
Input/Output Functions	57
Latches	60
Logic Primitives	61
Map Elements	67
Memory Elements	67
Multiplexers	69

Shifters	70
Shift Registers	70

Design Elements

ACC1	75
ACC4, 8, 16	79
ADD1	87
ADD4, 8, 16	89
ADSU1	95
ADSU4, 8, 16	99
AND2-9	105
AND12, 16	111
BRLSHFT4, 8	115
BSCAN_SPARTAN2	117
BSCAN_SPARTAN3	119
BSCAN_VIRTEX	121
BSCAN_VIRTEX2	123
BUF	125
BUF4, 8, 16	127
BUFCF	129
BUFE, 4, 8, 16	131
BUFG	135
BUFGCE	137
BUFGCE_1	139
BUFGDLL	141
BUFGMUX	143
BUFGMUX_1	145
BUFGP	147
BUFGSR	149
BUFGTS	151
BUFT, 4, 8, 16	153
CAPTURE_SPARTAN2	157
CAPTURE_SPARTAN3	159
CAPTURE_VIRTEX	161
CAPTURE_VIRTEX2	163
CB2CE, CB4CE, CB8CE, CB16CE	165
CB2CLE, CB4CLE, CB8CLE, CB16CLE	169
CB2CLED, CB4CLED, CB8CLED, CB16CLED	173
CB2RE, CB4RE, CB8RE, CB16RE	177
CB2RLE, CB4RLE, CB8RLE, CB16RLE	181
CB2X1, CB4X1, CB8X1, CB16X1	183
CB2X2, CB4X2, CB8X2, CB16X2	187

CBD2CE, CBD4CE, CBD8CE, CBD16CE	191
CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE	193
CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED	197
CBD2RE, CBD4RE, CBD8RE, CBD16RE	201
CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE	205
CBD2X1, CBD4X1, CBD8X1, CBD16X1	209
CBD2X2, CBD4X2, CBD8X2, CBD16X2	211
CC8CE, CC16CE	215
CC8CLE, CC16CLE	219
CC8CLED, CC16CLED	223
CC8RE, CC16RE	227
CD4CE	231
CD4CLE	235
CD4RE	239
CD4RLE	243
CDD4CE	247
CDD4CLE	249
CDD4RE	251
CDD4RLE	253
CJ4CE, CJ5CE, CJ8CE	255
CJ4RE, CJ5RE, CJ8RE	257
CJD4CE, CJD5CE, CJD8CE	259
CJD4RE, CJD5RE, CJD8RE	261
CLK_DIV2,4,6,8,10,12,14,16	263
CLK_DIV2,4,6,8,10,12,14,16R	267
CLK_DIV2,4,6,8,10,12,14,16RSD	273
CLK_DIV2,4,6,8,10,12,14,16SD	281
CLKDLL	289
CLKDLLE	293
CLKDLLHF	297
COMP2, 4, 8, 16	301
COMPM2, 4, 8, 16	303
COMPMC8, 16	307
CR8CE, CR16CE	311
CRD8CE, CRD16CE	315
D2_4E	317
D3_8E	319
D4_16E	321
DCM	323
DEC_CC4, 8, 16	331
DECODE4, 8, 16	333
DECODE32, 64	335

FD	337
FD_1	339
FD4, 8, 16	341
FD4CE, FD8CE, FD16CE	343
FD4RE, FD8RE, FD16RE	345
FDC	347
FDC_1	349
FDCE	351
FDCE_1	353
FDCP	355
FDCP_1	357
FDCPE	359
FDCPE_1	363
FDD	365
FDD4,8,16	367
FDD4CE, FDD8CE, FDD16CE	369
FDD4RE, FDD8RE, FDD16RE	371
FDDC	373
FDDCE	375
FDDCP	377
FDDCPE	379
FDDP	381
FDDPE	383
FDDR	385
FDDRCPE	387
FDDRE	389
FDDRRSE	391
FDDRS	393
FDDRSE	395
FDDS	397
FDDSE	399
FDDSR	401
FDDSRSE	403
FDE	405
FDE_1	407
FDP	409
FDP_1	413
FDPE	415
FDPE_1	419
FDR	421
FDR_1	423
FDRE	425

FDRE_1	429
FDRS	431
FDRS_1	435
FDRSE	437
FDRSE_1	441
FDS	443
FDS_1	445
FDSE	447
FDSE_1	451
FDSR	453
FDSRE	455
FJKC	457
FJKCE	459
FJKCP	461
FJKCPE	463
FJKP	465
FJKPE	467
FJKRSE	469
FJKSRE	471
FMAP	473
FTC	475
FTCE	479
FTCLE	483
FTCLEX	485
FTCP	487
FTCPE	489
FTCPLE	491
FTDCE	493
FTDCLE	495
FTDCLEX	497
FTDCP	499
FTDRSE	501
FTDRSLE	503
FTP	505
FTPE	509
FTPLE	511
FTRSE	513
FTRSLE	515
FTSRE	517
FTSRLE	519
GND	521
GT_AURORA_n	523

GT_CUSTOM	527
GT_ETHERNET_n	529
GT_FIBRE_CHAN_n	533
GT_INFINIBAND_n	537
GT_XAUI_n	541
GT10_10GE_n	545
GT10_10GFC_n	549
GT10_AURORA_n	553
GT10_AURORAX_n	557
GT10_CUSTOM	561
GT10_INFINIBAND_n	563
GT10_OC48_n	567
GT10_OC192_n	571
GT10_PCI_EXPRESS_n	575
GT10_XAUI_n	579
IBUF, 4, 8, 16	583
IBUFDS	589
IBUFG	591
IBUFGDS	593
ICAP_VIRTEX2	595
IFD, 4, 8, 16	597
IFD_1	601
IFDDRCPE	603
IFDDRRSE	605
IFDI	607
IFDI_1	609
IFDX, 4, 8, 16	611
IFDX_1	615
IFDXI	617
IFDXI_1	619
ILD, 4, 8, 16	621
ILD_1	625
ILDI	627
ILDI_1	629
ILDX, 4, 8, 16	631
ILDX_1	635
ILDXI	637
ILDXI_1	639
INV, 4, 8, 16	641
IOBUF	643
IOBUFDS	645
IOPAD, 4, 8, 16	647

IPAD, 4, 8, 16	649
JTAGPPC	651
KEEPER	653
LD	655
LD4, 8, 16	659
LD_1	661
LDC	663
LDC_1	665
LDCE	667
LDCE_1	669
LD4CE, LD8CE, LD16CE	671
LDCP	673
LDCP_1	675
LDCPE	677
LDCPE_1	679
LDE	681
LDE_1	683
LDG	685
LDG4, 8, 16	687
LDP	689
LDP_1	691
LDPE	693
LDPE_1	695
LUT1, 2, 3, 4	697
LUT1_D, LUT2_D, LUT3_D, LUT4_D	703
LUT1_L, LUT2_L, LUT3_L, LUT4_L	709
M2_1	715
M2_1B1	717
M2_1B2	719
M2_1E	721
M4_1E	723
M8_1E	725
M16_1E	727
MULT_AND	729
MULT18X18	731
MULT18X18S	733
MUXCY	735
MUXCY_D	737
MUXCY_L	739
MUXF5	741
MUXF5_D	743
MUXF5_L	745

MUXF6	747
MUXF6_D	749
MUXF6_L	751
MUXF7	753
MUXF7_D	755
MUXF7_L	757
MUXF8	759
MUXF8_D	761
MUXF8_L	763
NAND2-9	765
NAND12, 16	769
NOR2-9	773
NOR12, 16	777
OBUF, 4, 8, 16	781
OBUFDS	785
OBUE, 4, 8, 16	787
OBUFT, 4, 8, 16	791
OBUFTDS	795
OFD, 4, 8, 16	797
OFD_1	803
OFDDRCPE	805
OFDDRRSE	807
OFDDRTCPE	809
OFDDRTRSE	813
OFDE, 4, 8, 16	815
OFDE_1	819
OFDI	821
OFDI_1	823
OFDT, 4, 8, 16	825
OFDT_1	829
OFDX, 4, 8, 16	831
OFDX_1	835
OFDXI	837
OFDXI_1	839
OPAD, 4, 8, 16	841
OR2-9	843
OR12, 16	847
ORCY	851
PPC405	853
PULLDOWN	865
PULLUP	867
RAM16X1D	869

RAM16X1D_1	873
RAM16X1S	877
RAM16X1S_1	879
RAM16X2D	881
RAM16X2S	883
RAM16X4D	887
RAM16X4S	889
RAM16X8D	893
RAM16X8S	895
RAM32X1D	899
RAM32X1D_1	903
RAM32X1S	907
RAM32X1S_1	909
RAM32X2S	911
RAM32X4S	915
RAM32X8S	919
RAM64X1D	923
RAM64X1D_1	927
RAM64X1S	931
RAM64X1S_1	935
RAM64X2S	939
RAM128X1S	943
RAM128X1S_1	947
RAMB4_Sm_Sn	951
RAMB4_Sn	969
RAMB16_Sm_Sn	975
RAMB16_Sn	1005
ROC	1027
ROCBUF	1029
ROM16X1	1031
ROM32X1	1033
ROM64X1	1035
ROM128X1	1037
ROM256X1	1039
SOP3-4	1041
SR4CE, SR8CE, SR16CE	1043
SR4CLE, SR8CLE, SR16CLE	1045
SR4CLED, SR8CLED, SR16CLED	1047
SR4RE, SR8RE, SR16RE	1051
SR4RLE, SR8RLE, SR16RLE	1053
SR4RLED, SR8RLED, SR16RLED	1055
SRD4CE, SRD8CE, SRD16CE	1059

SRD4CLE, SRD8CLE, SRD16CLE	1061
SRD4CLED, SRD8CLED, SRD16CLED	1063
SRD4RE, SRD8RE, SRD16RE	1067
SRD4RLE, SRD8RLE, SRD16RLE	1069
SRD4RLED, SRD8RLED, SRD16RLED	1071
SRL16	1075
SRL16_1	1079
SRL16E	1081
SRL16E_1	1083
SRLC16	1085
SRLC16_1	1087
SRLC16E	1089
SRLC16E_1	1091
STARTBUF_architecture	1093
STARTUP_SPARTAN2	1095
STARTUP_SPARTAN3	1097
STARTUP_VIRTEX	1099
STARTUP_VIRTEX2	1101
TOC	1103
TOCBUF	1105
UPAD	1107
VCC	1109
XNOR2-9	1111
XOR2-9	1117
XORCY	1123
XORCY_D	1125
XORCY_L	1127

Xilinx Unified Libraries

This chapter describes the Unified Libraries and the applicable device architectures for each library. It also briefly discusses the contents of the other chapters, the general naming conventions, and performance issues.

This chapter consists of the following major sections.

- “Overview”
- “Applicable Architectures”
- “Functional Categories”
- “Design Elements”
- “Schematic Examples”
- “Naming Conventions”
- “Attributes and Constraints”
- “Carry Logic”
- “Flip-Flop, Counter, and Register Performance”
- “Unconnected Pins”

Overview

Xilinx maintains software libraries with thousands of functional design elements (primitives and macros) for different device architectures. New functional elements are assembled with each release of development system software. The catalog of design elements is known as the Unified Libraries. Many of the elements in these libraries can be used in multiple Xilinx device architectures. This “unified” approach means that you can use your circuit design created with “unified” library elements across all current Xilinx device architectures that recognize the element you are using.

Elements that exist in multiple architectures look and function the same, but their implementations might differ to make them more efficient for a particular architecture. A separate library still exists for each architecture (or architectural group) and common symbols are duplicated in each one, which is necessary for simulation (especially board level) where timing depends on a particular architecture.

If you have active designs that were created with former Xilinx library primitives or macros, you may need to change references to the design elements that you were using to reflect the Unified Libraries elements.

The *Libraries Guide* describes the primitive and macro logic elements available in the Unified Libraries for the Xilinx FPGA and CPLD devices. Common logic functions can be implemented with these elements and more complex functions can be built by

combining macros and primitives. Several hundred design elements (primitives and macros) are available across multiple device architectures, providing a common base for programmable logic designs.

This libraries guide provides a functional selection guide and describes the design elements.

Applicable Architectures

Design elements for the Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex -II, Virtex-II Pro, Virtex-II Pro X, XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II libraries are included in the Xilinx Unified Libraries. Each library supports specific device architectures. For detailed information on the architectural families referenced below and the devices in each, see the current version of *The Programmable Logic Data Sheets* (an online version is available from the Xilinx web site, <http://support.xilinx.com>).

Functional Categories

The functional categories list the available elements in each category along with a brief description of each element and an applicability table identifying which libraries (Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex -II, Virtex-II Pro, Virtex-II Pro X, XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II) contain the element.

Design Elements

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

The following information is provided for each library element, where applicable:

- Graphic symbol
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table
- Schematic for macros
- VHDL and Verilog instantiation and inference code
- Commonly used constraints

Schematic Examples

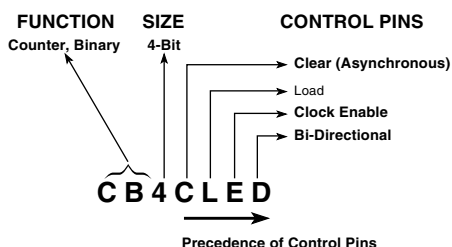
Schematics are included for each library if the implementation differs.

Design elements with bussed or multiple I/O pins (2-, 4-, 8-, 16-bit versions) typically include just one schematic -- generally the 8-bit version. When only one schematic is included, implementation of the smaller and larger elements differs only in the number of sections. In cases where an 8-bit version is very large, an appropriate smaller element serves as the schematic example.

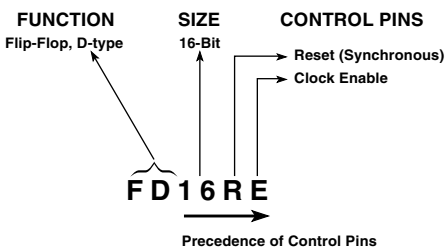
Naming Conventions

Examples of the general naming conventions for the unified library elements are shown in the following figures.

Example 1

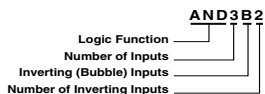


Example 2



X7764

Naming Conventions



X4316

Combinatorial Naming Conventions

Attributes and Constraints

Attributes and constraints are instructions placed on components or nets to indicate their placement, implementation, naming, directionality, etc.. The *Constraints Guide* provides information on all attributes and constraints.

Carry Logic

The Spartan-II, Spartan-IIE, Virtex, and Virtex-II architectures include dedicated carry logic components.

Spartan-II, Spartan-IIE, Virtex, and Virtex-E

Carry Logic for Spartan-II, Spartan-IIE, Virtex, and Virtex-E is a simple structure associated with each look-up table. The design entry library contains the following dedicated carry logic primitives: MULT_AND, MUXCY, MUXCY_D, MUXCY_L, XORCY, XORCY_D, and XORCY_L. The function performed is determined by their

connectivity and the contents of the look-up table. For an example of how to use carry logic, see “[CC8CE](#), [CC16CE](#)”.

For detailed information on Carry Logic in Virtex and Spartan-II, see *The Programmable Logic Data Sheets* available on the Xilinx web site, <http://support.xilinx.com>.

Virtex-II, Virtex-II Pro, Virtex-II Pro X, and Spartan-3

The dedicated carry logic primitives for Virtex-II, Virtex-II Pro, Virtex-II Pro X, and Spartan-3 are MULT_AND, MUXCY, MUXCY_D, MUXCY_L, XORCY, XORCY_D, and XORCY_L.

ORCY can only be used exclusively with Virtex-II, Virtex-II Pro, and Virtex-II Pro X.

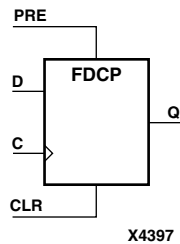
For detailed information on Carry Logic in Virtex-II, Virtex-II Pro, Virtex-II Pro X, and Spartan-3, see *The Programmable Logic Data Sheets* available on the Xilinx web site, <http://support.xilinx.com>.

Flip-Flop, Counter, and Register Performance

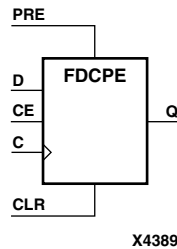
All counter, register, and storage functions derived from the flip-flops are available in the Configurable Logic Blocks (CLBs).

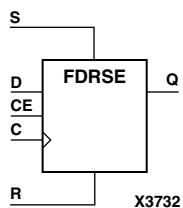
The D flip-flop is the basic building block for all architectures. Differences occur from the availability of asynchronous Clear (CLR) and Preset (PRE) inputs, and the source of the synchronous control signals, such as Clock Enable (CE), Clock (C), Load enable (L), synchronous Reset (R), and synchronous Set (S). The basic flip-flop configuration for each architecture follows.

The basic XC9000 flip-flops have both Clear and Preset inputs.



Virtex and Spartan-II have two basic flip-flop types. One has both Clear and Preset inputs and one has both asynchronous and synchronous control functions.





The asynchronous and synchronous control functions, when used, have a priority that is consistent across all devices and architectures. These inputs can be either active-High or active-Low as defined by the macro. The priority, from highest to lowest, is as follows.

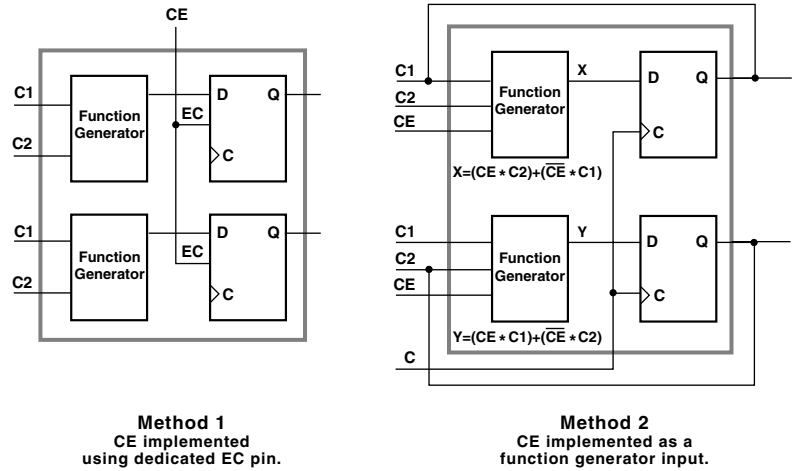
- Asynchronous Clear (CLR)
- Asynchronous Preset (PRE)
- Synchronous Set (S)
- Synchronous Reset (R)
- Clock Enable (CE)

Note: The asynchronous CLR and PRE inputs, by definition, have priority over all the synchronous control and clock inputs.

For FPGA families, the Clock Enable (CE) function is implemented using two different methods in the Xilinx Unified Libraries; both are shown in the following figure.

- In method 1, CE is implemented by connecting the CE pin of the macro directly to the dedicated Enable Clock (EC) pin of the internal Configurable Logic Block (CLB) flip-flop. This allows one CE per CLB. CE takes precedence over the L, S, and R inputs. All flip-flops with asynchronous clear or preset use this method.
- In method 2, CE is implemented using function generator logic. This allows two CEs per CLB. CE has the same priority as the L, S, and R inputs. All flip-flops with synchronous set or reset use this method.

The method used in a particular macro is indicated by the inclusion of asynchronous clear, asynchronous preset, synchronous set, or synchronous reset in the macro 's description.



Clock Enable Implementation Methods

Unconnected Pins

Xilinx recommends that you *always* connect input pins in your designs. This ensures that front end simulation functionally matches back end timing simulation. If an input pin is left unconnected, mapper errors may result.

If an output pin is left unconnected in your design, the corresponding function is trimmed. If the component has only one output, the entire component is trimmed. If the component has multiple outputs, the portion that drives the output is trimmed. As an example of the latter case, if the overflow pin (OFL) in an adder macro is unconnected, the logic that generates that term is trimmed, but the rest of the adder is retained (assuming all of the sum outputs are connected).

Slice Count

This chapter contains the following sections.

- [About Configurable Logic Blocks \(CLBs\)](#)
- [Slice Count for FPGA Components](#)

About Configurable Logic Blocks (CLBs)

Configurable Logic Blocks (CLBs) implement most of the logic in an FPGA.

Each Virtex, Virtex-E and Spartan-II, Spartan-IIE CLB contains two slices. Each Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLB contains four slices. In the following table, the numbers for Spartan-II, Spartan-IIE, Virtex, Virtex-E, Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X are the number of slices required to implement the component.

The Slice Count table lists FPGA design elements in alphanumeric order with the number of CLBs or slices needed for their implementation in each applicable library.

Note: This information is for reference only. The actual count could vary, depending upon the switch settings of the implementation tools; for example, the effort level in PAR (Place and Route) or usage of the components with other components.

The asterisk for the RAM16X1D and RAM16X1D_1 in the Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X columns indicates that these design elements consume 1/2 of two slices.

The double asterisks for design elements indicate that these primitives cannot be used by themselves. However, there is only one available per slice.

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
ACC4	5	6	5	6
ACC8	9	10	9	10
ACC16	17	18	17	18
ADD4	3	3	3	3
ADD8	5	5	5	5
ADD16	9	9	9	9
ADSU4	3	3	3	3
ADSU8	5	5	5	5

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
ADSU16	9	9	9	9
AND2	1	1	1	1
AND3	1	1	1	1
AND4	1	1	1	1
AND5	1	1	1	1
AND6	1	1	1	1
AND7	1	1	1	1
AND8	2	2	2	2
AND9	2	2	2	2
AND12	2	2	2	2
AND16	2	2	2	2
BRLSHIFT4	8	4	8	4
BRLSHIFT8	12	12	12	12
BSCAN_SPARTAN2	-	-	-	-
BSCAN_VIRTEX	-	-	-	-
BSCAN_VIRTEX2	-	-	-	-
BUF	-	-	-	-
BUF4	-	-	-	-
BUF8	-	-	-	-
BUF16	-	-	-	-
BUFCF	-	-	-	-
BUFE	-	-	-	-
BUFE4	-	-	-	-
BUFE8	-	-	-	-
BUFE16	-	-	-	-
BUFG	-	-	-	-
BUFGCE	-	-	-	-
BUFGCE_1	-	-	-	-
BUFGDLL	-	-	-	-
BUFGMUX	-	-	-	-
BUFGMUX_1	-	-	-	-
BUFGP	-	-	-	-
BUFT	-	-	-	-
BUFT4	-	-	-	-
BUFT8	-	-	-	-
BUFT16	-	-	-	-
CAPTURE_SPARTAN2	-	-	-	-

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
CAPTURE_VIRTEX	-	-	-	-
CAPTURE_VIRTEX2	-	-	-	-
CB2CE	2	3	2	3
CB2CLE	3	3	3	3
CB2CLED	3	3	3	3
CB2RE	2	3	2	3
CB4CE	3	4	3	4
CB4CLE	5	5	5	5
CB4CLED	6	7	6	7
CB4RE	3	4	3	4
CB8CE	6	7	6	7
CB8CLE	9	10	9	10
CB8CLED	12	12	12	12
CB8RE	6	7	6	7
CB16CE	13	14	13	14
CB16CLE	18	19	18	19
CB16CLED	24	25	24	25
CB16RE	13	14	13	14
CC8CE	8	5	8	5
CC8CLE	9	9	9	9
CC8CLED	9	17	9	17
CC8RE	9	9	9	9
CC16CE	16	9	16	9
CC16CLE	17	17	17	17
CC16CLED	17	33	17	33
CC16RE	17	17	17	17
CD4CE	3	4	3	4
CD4CLE	5	5	5	5
CD4RE	3	4	3	4
CD4RLE	7	7	7	7
CJ4CE	2	4	2	4
CJ4RE	2	4	2	4
CJ5CE	3	5	3	5
CJ5RE	3	5	3	5
CJ8CE	4	4	4	4
CJ8RE	4	4	4	4
CLKDLL	-	-	-	-

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
CLKDLLE	-	-	-	-
CLKDLLHF	-	-	-	-
COMP2	1	1	1	1
COMP4	2	2	2	2
COMP8	3	4	3	4
COMP16	6	9	6	9
COMPM2	1	2	1	2
COMPM4	5	5	5	5
COMPM8	11	13	11	13
COMPM16	24	32	24	32
COMP8CE	8	8	8	8
COMP16CE	16	16	16	16
CR8CE	8	8	8	8
CR16CE	16	16	16	16
D2_4E	2	2	2	2
D3_8E	4	4	4	4
D4_16E	16	16	16	16
DCM	-	-	-	-
DEC_CC4	1	1	1	1
DEC_CC8	1	1	1	1
DEC_CC16	2	2	2	2
DECODE4	1	1	1	1
DECODE8	2	2	2	2
DECODE16	2	2	2	2
DECODE32	4	4	4	4
DECODE64	8	8	8	8
FD	1	1	1	1
FD_1	1	1	1	1
FD4CE	2	4	2	4
FD4RE	2	4	2	4
FD8CE	4	4	4	4
FD8RE	4	4	4	4
FD16CE	8	8	8	8
FD16RE	8	8	8	8
FDC	1	1	1	1
FDC_1	1	1	1	1
FDCE	1	1	1	1

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
FDCE_1	1	1	1	1
FDCP	1	1	1	1
FDCP_1	1	1	1	1
FDCPE	1	1	1	1
FDCPE_1	1	1	1	1
FDDRCPE	-	-	-	-
FDDRRSE	-	-	-	-
FDE	1	1	1	1
FDE_1	1	1	1	1
FDP	1	1	1	1
FDP_1	1	1	1	1
FDPE	1	1	1	1
FDPE_1	1	1	1	1
FDR	1	1	1	1
FDR_1	1	1	1	1
FDRE	1	1	1	1
FDRE_1	1	1	1	1
FDRS	1	1	1	1
FDRS_1	1	1	1	1
FDRSE	1	1	1	1
FDRSE_1	1	1	1	1
FDS	1	1	1	1
FDS_1	1	1	1	1
FDSE	1	1	1	1
FDSE_1	1	1	1	1
FJKC	1	1	1	1
FJKCE	1	1	1	1
FJKP	1	1	1	1
FJKPE	1	1	1	1
FJKRSE	1	1	1	1
FJKSRE	1	1	1	1
FMAP	-	-	-	-
FTC	1	1	1	1
FTCE	1	1	1	1
FTCLE	1	1	1	1
FTCLEX	1	1	1	1
FTP	1	1	1	1

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
FTPE	1	1	1	1
FTPLE	1	1	1	1
FTRSE	1	1	1	1
FTRSLE	2	1	2	1
FTSRE	1	1	1	1
FTSRLE	2	2	2	2
GND	-	-	-	-
GT_AURORA_n	-	-	-	-
GT_CUSTOM_n	-	-	-	-
GT_ETHERNET_n	-	-	-	-
GT_FIBRE_CHAN_n	-	-	-	-
GT_INFINIBAND_n	-	-	-	-
GT_XAUI_n	-	-	-	-
GT10_AURORA_n	-	-	-	-
GT10_AURORAX_n	-	-	-	-
GT10_CUSTOM_n	-	-	-	-
GT10_INFINIBAND_n	-	-	-	-
GT10_XAUI_n	-	-	-	-
GT10_10GE_n	-	-	-	-
GT10_10GFC_n	-	-	-	-
GT10_OC48_n	-	-	-	-
GT10_OC192_n	-	-	-	-
IBUF	-	-	-	-
IBUF4	-	-	-	-
IBUF8	-	-	-	-
IBUF16	-	-	-	-
IBUFDS	-	-	-	-
IBUFG	-	-	-	-
IBUFGDS	-	-	-	-
ICAP_VIRTEX2	-	-	-	-
IFD	-	-	-	-
IFD_1	-	-	-	-
IFD4	-	-	-	-
IFD8	-	-	-	-
IFD16	-	-	-	-
IFDDRCPE	-	-	-	-
IFDDRRSE	-	-	-	-

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
IFDI	-	-	-	-
IFDI_1	-	-	-	-
IFDX	-	-	-	-
IFDX4	-	-	-	-
IFDX8	-	-	-	-
IFDX16	-	-	-	-
IFDX_1	-	-	-	-
IFDXI	-	-	-	-
IFDXI_1	-	-	-	-
ILD	1	1	1	1
ILD_1	1	1	1	1
ILD4	2	2	2	2
ILD8	4	4	4	4
ILD16	8	8	8	8
ILDI	-	-	-	-
ILDI_1	-	-	-	-
ILDY	-	-	-	-
ILDY4	-	-	-	-
ILDY8	-	-	-	-
ILDY16	-	-	-	-
ILDY_1	-	-	-	-
ILDYI	-	-	-	-
ILDYI_1	-	-	-	-
INV	1	1	1	1
INV4	1	1	1	1
INV8	1	1	1	1
INV16	1	1	1	1
IOBUF	-	-	-	-
IOPAD	-	-	-	-
IOPAD4	-	-	-	-
IOPAD8	-	-	-	-
IOPAD16	-	-	-	-
IPAD	-	-	-	-
IPAD4	-	-	-	-
IPAD8	-	-	-	-
IPAD16	-	-	-	-
JTAGPPC	-	-	-	-

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
KEEPER	-	-	-	-
LD	1	1	1	1
LD_1	1	1	1	1
LD4	2	4	2	4
LD8	4	4	4	4
LD16	8	8	8	8
LD4CE	2	4	2	4
LD8CE	4	4	4	4
LD16CE	8	8	8	8
LDC	1	1	1	1
LDC_1	1	1	1	1
LDCE	1	1	1	1
LDCE_1	1	1	1	1
LDCP	1	1	1	1
LDCP_1	1	1	1	1
LDCPE	1	1	1	1
LDCPE_1	1	1	1	1
LDE	1	1	1	1
LDE_1	1	1	1	1
LDP	1	1	1	1
LDP_1	1	1	1	1
LDPE	1	1	1	1
LDPE_1	1	1	1	1
LUT1	1	1	1	1
LUT2	1	1	1	1
LUT3	1	1	1	1
LUT4	1	1	1	1
LUT1_D	1	1	1	1
LUT2_D	1	1	1	1
LUT3_D	1	1	1	1
LUT4_D	1	1	1	1
LUT1_L	1	1	1	1
LUT2_L	1	1	1	1
LUT3_L	1	1	1	1
LUT4_L	1	1	1	1
M2_1	1	1	1	1
M2_1B1	1	1	1	1

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
M2_1B2	1	1	1	1
M2_1E	1	1	1	1
M4_1E	1	1	1	1
M8_1E	2	2	2	2
M16_1E	5	5	5	5
MULT_AND **	-	-	-	-
MULT18X18	-	-	-	-
MULT18X18S	-	-	-	-
MUXCY **	-	-	-	-
MUXCY_D **	-	-	-	-
MUXCY_L **	-	-	-	-
MUXF5 **	-	-	-	-
MUXF5_D **	-	-	-	-
MUXF5_L **	-	-	-	-
MUXF6 **	-	-	-	-
MUXF6_D **	-	-	-	-
MUXF6_L **	-	-	-	-
MUXF7 **	-	-	-	-
MUXF7_D **	-	-	-	-
MUXF7_L **	-	-	-	-
MUXF8 **	-	-	-	-
MUXF8_D **	-	-	-	-
MUXF8_L **	-	-	-	-
NAND2	1	1	1	1
NAND3	1	1	1	1
NAND4	1	1	1	1
NAND5	1	1	1	1
NAND6	1	1	1	1
NAND7	1	1	1	1
NAND8	2	2	2	2
NAND9	2	2	2	2
NAND12	2	2	2	2
NAND16	2	2	2	2
NOR2	1	1	1	1
NOR3	1	1	1	1
NOR4	1	1	1	1
NOR5	1	1	1	1

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
NOR6	1	1	1	1
NOR7	1	1	1	1
NOR8	2	2	2	2
NOR9	2	2	2	2
NOR12	2	2	2	2
NOR16	2	2	2	2
OBUF	-	-	-	-
OBUF4	-	-	-	-
OBUF8	-	-	-	-
OBUF16	-	-	-	-
OBUFDS	-	-	-	-
OBUFE	-	-	-	-
OBUFE4	-	-	-	-
OBUFE8	-	-	-	-
OBUFE16	-	-	-	-
OBUFT	-	-	-	-
OBUFT4	-	-	-	-
OBUFT8	-	-	-	-
OBUFT16	-	-	-	-
OBUFTDS	-	-	-	-
OFD	-	-	-	-
OFD_1	-	-	-	-
OFD4	-	-	-	-
OFD8	-	-	-	-
OFD16	-	-	-	-
OFDDRCPE	-	-	-	-
OFDDRRSE	-	-	-	-
OFDDRTCPE	-	-	-	-
OFDDRTRSE	-	-	-	-
OFDE	-	-	-	-
OFDE_1	-	-	-	-
OFDE4	-	-	-	-
OFDE8	-	-	-	-
OFDE16	-	-	-	-
OFDI	-	-	-	-
OFDI_I	-	-	-	-
OFDT	-	-	-	-

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
OFDT_1	-	-	-	-
OFDT4	-	-	-	-
OFDT8	-	-	-	-
OFDT16	-	-	-	-
OFDX	-	-	-	-
OFDX4	-	-	-	-
OFDX8	-	-	-	-
OFDX16	-	-	-	-
OFDX_1	-	-	-	-
OFDXI	-	-	-	-
OFDXI_I	-	-	-	-
OPAD	-	-	-	-
OPAD4	-	-	-	-
OPAD8	-	-	-	-
OPAD16	-	-	-	-
OR2	1	1	1	1
OR3	1	1	1	1
OR4	1	1	1	1
OR5	1	1	1	1
OR6	1	1	1	1
OR7	1	1	1	1
OR8	2	2	2	2
OR9	2	2	2	2
OR12	2	2	2	2
OR16	2	2	2	2
ORCY **	-	-	-	-
PPC405	-	-	-	-
PULLDOWN	-	-	-	-
PULLUP	-	-	-	-
RAM16X1D	1	2*	1	2*
RAM16X1D_1	1	2*	1	2*
RAM16X1S	1	1	1	1
RAM16X1S_1	1	1	1	1
RAM16X2D	2	4	2	4
RAM16X2S	2	2	2	2
RAM16X4D	4	8	4	8
RAM16X4S	4	3	4	3

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-II-E	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
RAM16X8D	8	16	8	16
RAM16X8S	8	5	8	5
RAM32X1D	-	2	-	2
RAM32X1D_1	-	2	-	2
RAM32X1S	1	1	1	1
RAM32X1S_1	1	1	1	1
RAM32X2S	2	2	2	2
RAM32X4S	8	3	8	3
RAM32X8S	-	6	-	6
RAM64X1D	-	-	-	4
RAM64X1D_1	-	4	-	4
RAM64X1S	-	2	-	2
RAM64X1S_1	-	2	-	2
RAM64X2S	-	4	-	4
RAM128X1S	-	4	-	4
RAM128X1S_1	-	4	-	4
RAMB4_Sn	-	-	-	-
RAMB4_Sm_Sn	-	-	-	-
RAMB16_Sn	-	-	-	-
RAMB16_Sm_Sn	-	-	-	-
ROM16X1	1	1	1	1
ROM32X1	1	1	1	1
ROM64X1	2	2	2	2
ROM128X1	-	4	-	4
ROM256X1	-	8	-	8
SOP3	1	1	1	1
SOP4	1	1	1	1
SR4CE	2	4	2	4
SR4CLE	3	3	3	3
SR4CLED	5	5	5	5
SR4RE	2	4	2	4
SR4RLE	3	3	3	3
SR4RLED	5	5	5	5
SR8CE	4	4	4	4
SR8CLE	5	5	5	5
SR8CLED	9	9	9	9
SR8RE	4	4	4	4

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
SR8RLE	5	5	5	5
SR8RLED	9	9	9	9
SR16CE	8	8	8	8
SR16CLE	9	9	9	9
SR16CLED	17	17	17	17
SR16RE	8	8	8	8
SR16RLE	9	9	9	9
SR16RLED	17	17	17	17
SRL16	1	1	1	1
SRL16_1	1	1	1	1
SRL16E	1	1	1	1
SRL16E_1	1	1	1	1
SRLC16	-	1	-	1
SRLC16_1	-	1	-	1
SRLC16E	-	1	-	1
SRLC16E_1	-	1	-	1
STARTUP_SPARTAN2	-	-	-	-
STARTUP_VIRTEX	-	-	-	-
STARTUP_VIRTEX2	-	-	-	-
UPAD	-	-	-	-
VCC	-	-	-	-
XNOR2	1	1	1	1
XNOR3	1	1	1	1
XNOR4	1	1	1	1
XNOR5	1	1	1	1
XNOR6	1	1	1	1
XNOR7	1	1	1	1
XNOR8	2	2	2	2
XNOR9	2	2	2	2
XOR2	1	1	1	1
XOR3	1	1	1	1
XOR4	1	1	1	1
XOR5	1	1	1	1
XOR6	1	1	1	1
XOR7	1	1	1	1
XOR8	2	2	2	2
XOR9	2	2	2	2

Slice Count for FPGA Components

Design Element	Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X
Name	Number of Slices to Implement			
XORCY **	-	-	-	-
XORCY_D **	-	-	-	-
XORCY_L **	-	-	-	-

* The RAM16X1D and RAM16X1D_1 consume 1/2 of two slices.

** These primitives cannot be used by themselves. However, there is only one available per slice.

Functional Categories

This section categories, by function, the logic elements that are described in detail in the “Design Elements” sections. Each category is briefly described. Tables under each category identify all the available elements for the function and indicate which architectures are supported by each.

Arithmetic Functions	Flip-Flops	Logic Primitives
Buffers	General	Map Elements
Comparators	Input Latches	Memory Elements
Counters	Input/Output Flip-Flops	Multiplexers
Decoders	Input/Output Functions	Shifters
Edge Decoders	Latches	Shift Registers

Elements are listed in alphanumeric order under each category.

The Xilinx libraries contain three types of elements.

- Primitives are basic logical elements such as AND2 and OR2 gates.
- Soft macros are schematics made by combining primitives and sometimes other soft macros.
- Relationally placed macros (RPMs) are soft macros that contain relative location constraint (RLOC) information, carry logic symbols, and FMAP symbols, where appropriate.

The last item mentioned above, RPMs, applies only to FPGA families.

The relationally placed macro (RPM) library uses RLOC constraints to define the order and structure of the underlying design primitives. Because these macros are built upon standard schematic parts, they do not have to be translated before simulation. The components that are implemented as RPMs are listed in the “[Slice Count](#)” section.

Designs created with RPMs can be functionally simulated. RPMs can, but need not, include all the following elements.

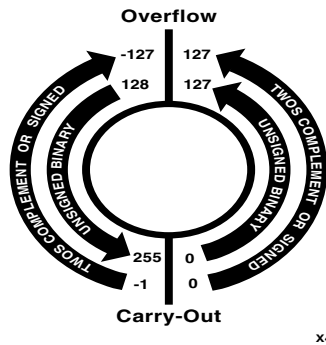
- Relative location (RLOC) constraints to provide placement structure. They allow positioning of elements relative to each other.
- Carry logic primitive symbols.

The RPM library offers the functionality and precision of the hard macro library with added flexibility. You can optimize RPMs and merge other logic within them. The elements in the RPM library allow you to access carry logic easily and to control mapping and block placement. Because RPMs are a superset of ordinary macros, you can design them in the normal design entry environment. They can include any

primitive logic. The macro logic is fully visible to you and can be easily back-annotated with timing information.

Arithmetic Functions

There are three types of arithmetic functions: accumulators (ACC), adders (ADD), and adder/subtractors (ADSU). With an ADSU, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.



X4720

ADSU Carry-Out and Overflow Boundaries

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
ACC1	1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset	No	No	No	No	Primitive	Primitive	Primitive
ACC4	4-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ACC8	8-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ACC16	16-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ADD1	1-Bit Full Adder with Carry-In and Carry-Out	No	No	No	No	Primitive	Primitive	Primitive
ADD4	4-Bit Cascadable Full Adder with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ADD8	8-Bit Cascadable Full Adder with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ADD16	16-Bit Cascadable Full Adder with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ADSU1	1-Bit Cascadable Adder/Subtractor with Carry-In, Carry-Out	No	No	No	No	Primitive	Primitive	Primitive
ADSU4	4-Bit Cascadable Adder/Subtractor with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
ADSU8	8-Bit Cascadable Adder/Subtractor with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
ADSU16	16-Bit Cascadable Adder/Subtractor with Carry-In, Carry-Out, and Overflow	Macro	Macro	Macro	Macro	Macro	Macro	Macro
MULT18X18	18 x 18 Signed Multiplier	No	Primitive	No	Primitive	No	No	No
MULT18X18S	18 x 18 Signed Multiplier -- Registered Version	No	Primitive	No	Primitive	No	No	No

Buffers

The buffers in this section route high fanout signals, 3-state signals, and clocks inside a PLD device. The [“Input/Output Functions”](#) section covers off-chip interfaces.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500 /XV/XL	CR XPLA3	CR-II
BUF	General Purpose Buffer	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
BUF4	4-Bit General Purpose Buffer	Macro	No	No	No	Primitive	Primitive	Primitive
BUF8	8-Bit General Purpose Buffer	Macro	No	No	No	Primitive	Primitive	Primitive
BUF16	16-Bit General Purpose Buffer	Macro	No	No	No	Primitive	Primitive	Primitive
BUFCF	Fast Connect Buffer	Primitive	Primitive	Primitive	Primitive	No	No	No
BUFE	Internal 3-State Buffer with Active High Enable	Primitive	No	Primitive	Primitive	Primitive ^a	No	No
BUFE4	Internal 3-State Buffer with Active High Enable	Primitive	No	Macro	Macro	Primitive	No	No
BUFE8	Internal 3-State Buffer with Active High Enable	Primitive	No	Macro	Macro	Primitive	No	No
BUFE16	Internal 3-State Buffer with Active High Enable	Primitive	No	Macro	Macro	Primitive	No	No
BUFG	Global Clock Buffer	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
BUFGCE	Global Clock MUX with Clock Enable and Output State 0	No	Primitive	No	Primitive	No	No	No
BUFGCE_1	Global Clock MUX Buffer with Clock Enable and Output State 1	No	Primitive	No	Primitive	No	No	No
BUFGDLL	Clock Delay Locked Loop Buffer	Primitive	Primitive	Primitive	Primitive	No	No	No
BUFGMUX	Global Clock MUX Buffer with Output State 0	No	Primitive	No	Primitive	No	No	No
BUFGMUX_1	Global Clock MUX with Output State 1	No	Primitive	No	Primitive	No	No	No
BUFGP	Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)	Primitive	Primitive	Primitive	Primitive	No	No	No
BUFGSR	Global Set/Reset Input Buffer	No	No	No	No	Primitive	Primitive	Primitive
BUFGTS	Global 3-State Input Buffer	No	No	No	No	Primitive	Primitive	Primitive
BUFT	Internal 3-State Buffer with Active-Low Enable	Primitive	No	Primitive	Primitive	Primitive ^b	No	No
BUFT4	Internal 3-State Buffer with Active-Low Enable	Macro	No	Macro	Macro	Primitive	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500 /XV/XL	CR XPLA3	CR-II
BUFT8	Internal 3-State Buffer with Active-Low Enable	Macro	No	Macro	Macro	Primitive ^c	No	No
BUFT16	Internal 3-State Buffer with Active-Low Enable	Macro	No	Macro	Macro	Primitive	No	No

- a. Not supported for XC9500XL and XC9500XV devices.
- b. Not supported for XC9500XL and XC9500XV devices.
- c. Not supported for XC9500XL and XC9500XV devices.

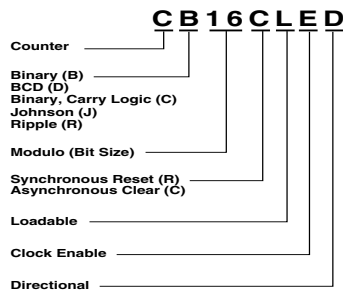
Comparators

Following is a list of comparators.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
COMP2	2-Bit Identity Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP4	4-Bit Identity Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP8	8-Bit Identity Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP16	16-Bit Identity Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP2M	2-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP4M	4-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP8M	8-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP16M	16-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
COMP8MC	8-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	No	No	No
COMP16MC	16-Bit Magnitude Comparator	Macro	Macro	Macro	Macro	No	No	No

Counters

There are six types of counters with various synchronous and asynchronous inputs. The name of the counter defines the modulo or bit size, the counter type, and which control functions are included. The counter naming convention is shown in the following figure:

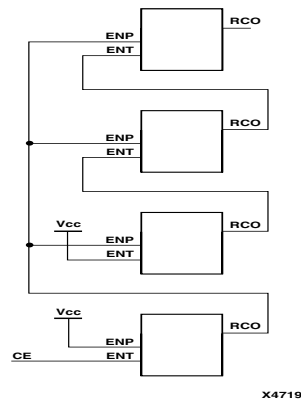


X4577

Counter Naming Convention

A carry-lookahead design accommodates large counters without extra gating. On TTL 7400-type counters with trickle clock enable (ENT), parallel clock enable (ENP), and ripple carry-out (RCO), both the ENT and ENP inputs must be High to count. ENT is

propagated forward to enable RCO, which produces a High output with the approximate duration of the QA output. The following figure illustrates a carry-lookahead design.



Carry-Lookahead Design

The RCO output of the first stage of the ripple carry is connected to the ENP input of the second stage and all subsequent stages. The RCO output of the second stage and all subsequent stages is connected to the ENT input of the next stage. The ENT of the second stage is always enabled/tied to VCC. CE is always connected to the ENT input of the first stage. This cascading method allows the first stage of the ripple carry to be built as a prescaler. In other words, the first stage is built to count very fast.

Note: For counters, do not use TC (or any other gated signal) as a clock. Possible glitches may not always allow for a proper setup time when using gated signals.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CB2CE	2-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB4CE	4-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB8CE	8-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB16CE	16-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB2CLE	2-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB4CLE	4-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB8CLE	8-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB16CLE	16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CB2CLED	2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB4CLED	2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB8CLED	2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB16CLED	2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear	No	No	No	No	Primitive	Primitive	Primitive
CB2RE	2-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB4RE	4-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB8RE	8-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB16RE	16-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CB2RLE	2-Bit Loadable Cascadable Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB4RLE	4-Bit Loadable Cascadable Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB8RLE	8-Bit Loadable Cascadable Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB16RLE	16-Bit Loadable Cascadable Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB2X1	2-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	Macro	Primitive	Primitive	Primitive
CB4X1	4-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	Macro	Primitive	Primitive	Primitive
CB8X1	8-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	Macro	Primitive	Primitive	Primitive
CB16X1	16-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	Macro	Primitive	Primitive	Primitive
CB2X2	2-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB4X2	4-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CB8X2	8-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CB16X2	16-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Synchronous Reset	No	No	No	Macro	Primitive	Primitive	Primitive
CBD2CE	2-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD4CE	4-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD8CE	8-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD16CE	16-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD2CLE	2-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD4CLE	4-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD8CLE	8-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD16CLE	16-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD2CLEd	2-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD4CLEd	4-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD8CLEd	8-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD16CLEd	16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD2RE	2-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD4RE	4-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive

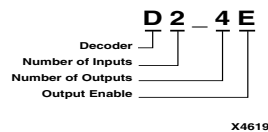
Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
CBD8RE	8-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD16RE	16-Bit Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD2RLE	2-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD4RLE	4-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD8RLE	8-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD16RLE	16-Bit Loadable Cascadable Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD2X1	2-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD4X1	4-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD8X1	8-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD16X1	16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CBD2X2	2-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD4X2	4-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD8X2	8-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CBD16X2	16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CC8CE	8-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No
CC16CE	16-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No
CC8CLE	8-Bit Loadable Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CC16CLE	16-Bit Loadable Cascadable Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No
CC8CLED	8-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No
CC16CLED	16-Bit Loadable Cascadable Bidirectional Binary Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	No	No	No
CC8RE	8-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	No	No	No
CC16RE	16-Bit Cascadable Binary Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	No	No	No
CD4CE	4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CD4CLE	4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CD4RE	4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CD4RLE	4-Bit Cascadable BCD Counter with Clock Enable And Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CDD4CE	4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CDD4CLE	4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CDD4RE	4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CDD4RLE	4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CD4RE	4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CD4RLE	4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ4CE	4-Bit Johnson Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ5CE	5-Bit Johnson Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ8CE	8-Bit Johnson Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ4RE	4-Bit Johnson Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ5RE	5-Bit Johnson Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CJ8RE	8-Bit Johnson Counter with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
CJD4CE	4-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CJD5CE	5-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CJD8CE	8-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CJD4RE	4-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CJD5RE	5-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CJD8RE	8-Bit Dual Edge Triggered Johnson Counter with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
CR8CE	8-Bit Negative-Edge Binary Ripple Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CR16CE	16-Bit Negative-Edge Binary Ripple Counter with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
CRD8CE	8-Bit Dual-Edge Triggered Binary Ripple Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
CRD16CE	16-Bit Dual-Edge Triggered Binary Ripple Counter with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive

Decoders

Decoder names, shown in the following figure, indicate the number of inputs and outputs and whether or not an enable is available. Decoders with an enable can be used as multiplexers.



Decoder Naming Convention

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
D2_4E	2- to 4-Line Decoder/Demultiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
D3_8E	3- to 8-Line Decoder/Demultiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
D4_16E	4- to 16-Line Decoder/Demultiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
DEC_CC4	4-Bit Active Low Decoder	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
DEC_CC8	8-Bit Active Low Decoder	Macro	Macro	Macro	Macro	No	No	No
DEC_CC16	16-Bit Active Low Decoder	Macro	Macro	Macro	Macro	No	No	No

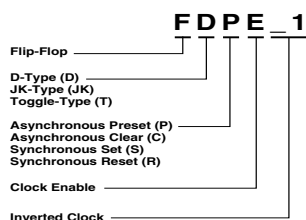
Edge Decoders

Edge decoders are open-drain wired AND gates that are available in different bit sizes.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
DECODE4	4-Bit Active-Low Decoder	Macro	Macro	Macro	Macro	No	No	No
DECODE8	8-Bit Active-Low Decoder	Macro	Macro	Macro	Macro	No	No	No
DECODE16	16-Bit Active-Low Decoder	Macro	Macro	Macro	Macro	No	No	No
DECODE32	32-Bit Active-Low Decoder	Macro	Macro	Macro	Macro	No	No	No
DECODE64	64-Bit Active-Low Decoder	Macro	Macro	Macro	Macro	No	No	No

Flip-Flops

There are three types of flip-flops (D, J-K, toggle) with various synchronous and asynchronous inputs. Some are available with inverted clock inputs and/or the ability to set in response to global set/reset rather than reset. The naming convention shown in the following figure provides a description for each flip-flop. D-type flip-flops are available in multiples of up to 16 in one macro.



X4579

Flip-Flop Naming Convention

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FD	D Flip-Flop	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FD_1	D Flip-Flop with Negative-Edge Clock	Primitive	Primitive	Primitive	Primitive	No	No	No
FD4	Multiple D Flip-Flop	No	No	No	No	Primitive	Primitive	Primitive
FD8	Multiple D Flip-Flop	No	No	No	No	Primitive	Primitive	Primitive
FD16	Multiple D Flip-Flop	No	No	No	No	Primitive	Primitive	Primitive
FD4CE	4-Bit Data Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FD8CE	8-Bit Data Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FD16CE	16-Bit Data Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FD4RE	4-Bit Data Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FD8RE	8-Bit Data Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FD16RE	16-Bit Data Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FDC	D Flip-Flop with Asynchronous Clear	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDC_1	D Flip-Flop with Negative-Edge Clock and Asynchronous Clear	Primitive	Primitive	Primitive	Primitive	No	No	No
FDCE	D Flip-Flop with Clock Enable and Asynchronous Clear	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDCE_1	D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear	Primitive	Primitive	Primitive	Primitive	No	No	No
FDCP	D Flip-Flop with Asynchronous Preset and Clear	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDCP_1	D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear	Primitive	Primitive	Primitive	Primitive	No	No	No
FDCPE	D Flip-Flop with Clock Enable and Asynchronous Preset and Clear	Primitive	Primitive	Primitive	Primitive	Macro	Macro	Primitive
FDCPE_1	D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear	Primitive	Primitive	Primitive	Primitive	No	No	No
FDD	Dual Edge Triggered D Flip-Flop	No	No	No	No	No	No	Primitive
FDD4	Multiple Dual Edge Triggered D Flip-Flop	No	No	No	No	No	No	Primitive
FDD8	Multiple Dual Edge Triggered D Flip-Flop	No	No	No	No	No	No	Primitive
FDD16	Multiple Dual Edge Triggered D Flip-Flop	No	No	No	No	No	No	Primitive
FDD4CE	4-Bit Dual Edge Triggered Data Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FDD8CE	8-Bit Dual Edge Triggered Data Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FDD16CE	16-Bit Dual Edge Triggered Data Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FDD4RE	4-Bit Dual Edge Triggered Data Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
FDD8RE	8-Bit Dual Edge Triggered Data Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
FDD16RE	16-Bit Dual Edge Triggered Data Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
FDDC	D Dual Edge Triggered Flip-Flop with Asynchronous Clear	No	No	No	No	No	No	Primitive
FDDCE	Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FDDCP	Dual Edge Triggered D Flip-Flop with Asynchronous Preset and Clear	No	No	No	No	No	No	Primitive
FDDCPE	Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset and Clear	No	No	No	No	No	No	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FDDP	Dual Edge Triggered D Flip-Flop with Asynchronous Preset	No	No	No	No	No	No	Primitive
FDDPE	Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset	No	No	No	No	No	No	Primitive
FDDR	Dual Edge Triggered D Flip-Flop with Synchronous Reset	No	No	No	No	No	No	Primitive
FDDRCPE	Dual Data Rate D Flip-Flop with Clock Enable and Asynchronous Preset and Clear	No	Primitive	No	Primitive	No	No	No
FDDRE	Dual Edge Triggered D Flip-Flop with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
FDDRSE	Dual Data Rate D Flip-Flop with Clock Enable and Synchronous Reset and Set	No	Primitive	No	Primitive	No	No	No
FDDRS	Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set	No	No	No	No	No	No	Primitive
FDDRSE	Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set and Clock Enable	No	No	No	No	No	No	Primitive
FDDS	Dual Edge Triggered D Flip-Flop with Synchronous Set	No	No	No	No	No	No	Primitive
FDDSE	D Flip-Flop with Clock Enable and Synchronous Set	No	No	No	No	No	No	Primitive
FDDSR	Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset	No	No	No	No	No	No	Primitive
FDDSRE	Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset and Clock Enable	No	No	No	No	No	No	Primitive
FDE	D Flip-Flop with Clock Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
FDE_1	D Flip-Flop with Negative-Edge Clock and Clock Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
FDP	D Flip-Flop with Asynchronous Preset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDP_1	D Flip-Flop with Negative-Edge Clock and Asynchronous Preset	Primitive	Primitive	Primitive	Primitive	No	No	No
FDPE	D Flip-Flop with Clock Enable and Asynchronous Preset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDPE_1	D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset	Primitive	Primitive	Primitive	Primitive	No	No	No
FDR	D Flip-Flop with Synchronous Reset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDR_1	D Flip-Flop with Negative-Edge Clock and Synchronous Reset	Primitive	Primitive	Primitive	Primitive	No	No	No
FDRE	D Flip-Flop with Clock Enable and Synchronous Reset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDRE_1	D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset	Primitive	Primitive	Primitive	Primitive	No	No	No
FDRS	D Flip-Flop with Synchronous Reset and Set	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDRS_1	D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set	Primitive	Primitive	Primitive	Primitive	No	No	No
FDRSE	D Flip-Flop with Synchronous Reset and Set and Clock Enable	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FDRSE_1	D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
FDS	D Flip-Flop with Synchronous Set	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDS_1	D Flip-Flop with Negative-Edge Clock and Synchronous Set	Primitive	Primitive	Primitive	Primitive	No	No	No
FDSE	D Flip-Flop with Clock Enable and Synchronous Set	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
FDSE_1	D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set	Primitive	Primitive	Primitive	No	No	No	No
FDSR	D Flip-Flop with Synchronous Set and Reset	No	No	No	No	Primitive	Primitive	Primitive
FDSRE	D Flip-Flop with Synchronous Set and Reset and Clock Enable	No	No	No	No	Primitive	Primitive	Primitive
FJKC	J-K Flip-Flop with Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FJKCE	J-K Flip-Flop with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FJKCP	J-K Flip-Flop with Asynchronous Clear and Preset	No	No	No	No	Primitive	Primitive	Primitive
FJKCPE	J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable	No	No	No	No	Primitive	Primitive	Primitive
FJKP	J-K Flip-Flop with Asynchronous Preset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FJKPE	J-K Flip-Flop with Clock Enable and Asynchronous Preset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FJKRSE	J-K Flip-Flop with Clock Enable and Synchronous Reset and Set	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FJKSRE	J-K Flip-Flop with Clock Enable and Synchronous Set and Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTC	Toggle Flip-Flop with Toggle Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTCE	Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTCLE	Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTCLEX	Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTCP	Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset	No	No	No	No	Primitive	Primitive	Primitive
FTCPE	Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset	No	No	No	No	Primitive	Primitive	Primitive
FTCPLE	Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset	No	No	No	No	Primitive	Primitive	Primitive
FTDCE	Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FTDCLE	Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FTDCLEX	Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
FTDCP	Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset	No	No	No	No	Primitive	Primitive	Primitive
FTDRSE	Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set	No	No	No	No	No	No	Primitive
FTDRSLE	Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTP	Toggle Flip-Flop with Toggle Enable and Asynchronous Preset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTPE	Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTPLE	Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTRSE	Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTRSLE	Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTSRE	Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
FTSRLE	Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

General

General elements include FPGA configuration functions, oscillators, boundary scan logic, and other functions not classified in other sections.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
BSCAN_SPARTAN2	Spartan-II Boundary Scan Logic Control Circuit	Primitive ^a	No	No	No	No	No	No
BSCAN_SPARTAN3	Spartan-3 Boundary Scan Logic Control Circuit	No	Primitive	No	No	No	No	No
BSCAN_VIRTEX	Virtex Boundary Scan Logic Control Circuit	Primitive ^b	No	Primitive	No	No	No	No
BSCAN_VIRTEX2	Virtex2 Boundary Scan Logic Control Circuit	No	No	No	Primitive	No	No	No
CAPTURE_SPARTAN2	Spartan-II Register State Capture for Bitstream Readback	Primitive	No	No	No	No	No	No
CAPTURE_SPARTAN3	Spartan-3 Register State Capture for Bitstream Readback	No	Primitive	No	No	No	No	No
CAPTURE_VIRTEX	Virtex Register State Capture for Bitstream Readback	No	No	Primitive	No	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan- 3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CAPTURE_VIRTEX2	Virtex-II Register State Capture for Bitstream Readback	No	No	No	Primitive	No	No	No
CLK_DIV2	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV4	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV6	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV8	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV10	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV12	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV14	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV16	Global Clock Divider	No	No	No	No	No	No	Primitive
CLK_DIV2R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV4R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV6R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV8R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV10R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV12R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV14R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV16R	Global Clock Divider with Synchronous Reset	No	No	No	No	No	No	Primitive
CLK_DIV2RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV4RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV6RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV8RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV10RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV12RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV14RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV16RSD	Global Clock Divider with Synchronous Reset and Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV2SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV4SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV6SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive

Design Element	Description	Spartan-II, IIE	Spartan- 3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/X L	CR XPLA3	CR-II
CLK_DIV8SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV10SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV12SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV14SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLK_DIV16SD	Global Clock Divider with Start Delay	No	No	No	No	No	No	Primitive
CLKDLL	Clock Delay Locked Loop	Primitive	No	Primitive	No	No	No	No
CLKDLLE	Clock Delay Locked Loop with Expanded Output	Primitive ^c	No	Primitive ^d	No	No	No	No
CLKDLLHF	High Frequency Clock Delay Locked Loop	Primitive	No	Primitive ^e	No	No	No	No
DCM	Digital Clock Manager	No	Primitive	No	Primitive	No	No	No
GND	Ground-Connection Signal Tag	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
ICAP_VIRTEX2	User Interface to Virtex-II Internal Configuration Access Port	No	No	No	Primitive	No	No	No
JTAGPPC	JTAG Primitive for the Power PC	No	No	No	Primitive ^f	No	No	No
KEEPER	KEEPER Symbol	Primitive	Primitive	Primitive	Primitive	Primitive ^g	No	Primitive
LUT1	1-Bit Look-Up Table with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT2	2-Bit Look-Up Table with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT3	3-Bit Look-Up Table with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT4	4-Bit Look-Up Table with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT1_D	1-Bit Look-Up Table with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT2_D	2-Bit Look-Up Table with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT3_D	3-Bit Look-Up Table with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT4_D	4-Bit Look-Up Table with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT1_L	1-Bit Look-Up Table with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT2_L	2-Bit Look-Up Table with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT3_L	3-Bit Look-Up Table with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
LUT4_L	4-Bit Look-Up Table with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
PPC405	Primitive for the Power PC Core	No	No	No	Primitive ^h	No	No	No
PULLDOWN	Resistor to GND for Input Pads	Primitive	Primitive	Primitive	Primitive	No	No	No
PULLUP	Resistor to VCC for Input PADS, Open-Drain, and 3-State Outputs	Primitive	Primitive	Primitive	Primitive	No	Primitive	Primitive
ROC	Reset On Configuration	Primitive	Primitive	Primitive	Primitive	No	No	No
STARTBUF_architecture	VHDL Simulation of FPGA Designs	Primitive	Primitive	Primitive	Primitive	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
STARTUP_SPARTAN2	Spartan-II User Interface to Global Clock, Reset, and 3-State Controls	Primitive ⁱ	No	No	No	No	No	No
STARTUP_SPARTAN3	Spartan-3 User Interface to Global Clock, Reset, and 3-State Controls	No	Primitive	No	No	No	No	No
STARTUP_VIRTEX	Virtex User Interface to Global Clock, Reset, and 3-State Controls	Primitive ^j	No	Primitive	No	No	No	No
STARTUP_VIRTEX2	Virtex-II User Interface to Global Clock, Reset, and 3-State Controls	No	No	No	Primitive	No	No	No
TOC	Three-State On Configuration	Primitive	Primitive	Primitive	Primitive	No	No	No
TOCBUF	Three-State On Configuration Buffer	Primitive	Primitive	Primitive	Primitive	No	No	No
VCC	VCC-Connection Signal Tag	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

- a. Primitive is supported for Spartan-II, but not for Spartan-IIE, which is supported by BSCAN_VIRTEX
- b. Primitive is supported for Spartan-IIE, but not for Spartan-II, which is supported by BSCAN_SPARTAN2.
- c. Supported for Spartan-IIE and Virtex-E devices only.
- d. Supported for Spartan-IIE and Virtex-E devices only.
- e. For Virtex E, use CLKDLLHF in HF mode. In LF mode, both the separate CLKDLL and CLKDLL primitive can be used.
- f. Supported for Virtex-II Pro and Virtex-II Pro X only.
- g. Supported for only XC9500XL and XC9500XV.
- h. Not supported for Virtex-II. Supported for Virtex-II Pro and Virtex-II Pro X only.
- i. The Primitive in the field marked Spartan IIE is supported only for Spartan-II but not for Spartan-IIE, the latter of which is supported by STARTUP_VIRTEX.
- j. The Primitive in the Spartan IIE field is supported for Spartan-IIE, but not for Spartan-II, which is supported by STARTUP_SPARTAN2.

Input Latches

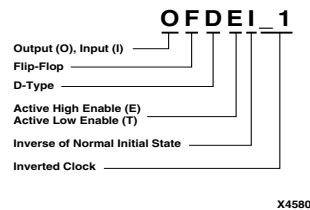
Single and multiple input latches can hold transient data entering a chip. Input latches use the same naming convention as I/O flip-flops.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
ILD	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD4	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD8	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD16	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD_1	Transparent Input Data Latch with Inverted Gate	Macro	Macro	Macro	Macro	No	No	No
ILDI	Transparent Input Data Latch (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
ILDI_1	Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
ILDx	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
ILD _{X4}	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD _{X8}	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD _{X16}	Transparent Input Data Latch	Macro	Macro	Macro	Macro	No	No	No
ILD _{X_1}	Transparent Input Data Latch with Inverted Gate	Macro	Macro	Macro	Macro	No	No	No
ILD _{XI}	Transparent Input Data Latch (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
ILD _{XI_1}	Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No

Input/Output Flip-Flops

Input/Output flip-flops are configured in IOBs. They include flip-flops whose outputs are enabled by 3-state buffers, flip-flops that can be set upon global set/reset rather than reset, and flip-flops with inverted clock inputs. The naming convention specifies each flip-flop function and is illustrated in the following figure.



Input/Output Flip-Flop Naming Convention

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
IFD	Single- and Multiple-Input D Flip-Flop	Macro	Macro	Macro	Macro	No	No	No
IFD ₁	Input D Flip-Flop with Inverted Clock	Macro	Macro	Macro	Macro	No	No	No
IFD ₄	Single- and Multiple-Input D Flip-Flop	Macro	Macro	Macro	Macro	No	No	No
IFD ₈	Single- and Multiple-Input D Flip-Flop	Macro	Macro	Macro	Macro	No	No	No
IFD ₁₆	Single- and Multiple-Input D Flip-Flop	Macro	Macro	Macro	Macro	No	No	No
IFDDRCPE	Dual Data Rate Input D Flip-Flop with Clock Enable and Asynchronous Preset and Clear	No	Primitive	No	Primitive	No	No	No
IFDDRRSE	Dual Data Rate Input D Flip-Flop with Synchronous Reset and Set and Clock Enable	No	Primitive	No	Primitive	No	No	No
IFDI	Input D Flip-Flop (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
IFDI ₁	Input D Flip-Flop with Inverted Clock (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
IFDX	Single- and Multiple-Input D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
IFDX_1	Input D Flip-Flop with Inverted Clock and Clock Enable	Macro	Macro	Macro	Macro	No	No	No
IFDX4	Single- and Multiple-Input D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
IFDX8	Single- and Multiple-Input D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
IFDX16	Single- and Multiple-Input D Flip-Flops with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
IFDXI	Input D Flip-Flop with Clock Enable (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
ILDXL_1	Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
OFD	Single- and Multiple-Output D Flip-Flops	Macro	Macro	Macro	Macro	No	No	No
OFD4	Single- and Multiple-Output D Flip-Flops	Macro	Macro	Macro	Macro	No	No	No
OFD8	Single- and Multiple-Output D Flip-Flops	Macro	Macro	Macro	Macro	No	No	No
OFD16	Single- and Multiple-Output D Flip-Flops	Macro	Macro	Macro	Macro	No	No	No
OFD_1	Output D Flip-Flop with Inverted Clock	Macro	Macro	Macro	Macro	No	No	No
OFDDRCPE	Dual Data Rate Output D Flip-Flop with Clock Enable and Asynchronous Preset and Clear	No	Primitive	No	Primitive	No	No	No
OFDDRRSE	Dual Data Rate Output D Flip-Flop with Synchronous Reset and Set and Clock Enable	No	Primitive	No	Primitive	No	No	No
OFDDRTCPE	Dual Data Rate D Flip-Flop with Active-Low 3-State Output Buffer, Clock Enable, and Asynchronous Preset and Clear	No	Primitive	No	Primitive	No	No	No
OFDDRTRSE	Dual Data Rate D Flip-Flop with Active -Low 3-State Output Buffer, Synchronous Reset and Set, and Clock Enable	No	Primitive	No	Primitive	No	No	No
OFDE	D Flip-Flop with Active-High Enable Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDE4	D Flip-Flop with Active-High Enable Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDE8	D Flip-Flop with Active-High Enable Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDE16	D Flip-Flop with Active-High Enable Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDE_1	D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock	Macro	Macro	Macro	Macro	No	No	No
OFDI	Output D Flip-Flop (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
OFDI_1	Output D Flip-Flop with Inverted Clock (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
OFDI	Single and Multiple D Flip-Flop with Active-Low 3-State Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDI4	Single and Multiple D Flip-Flop with Active-Low 3-State Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDI8	Single and Multiple D Flip-Flop with Active-Low 3-State Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDI16	Single and Multiple D Flip-Flop with Active-Low 3-State Output Buffers	Macro	Macro	Macro	Macro	No	No	No
OFDI_1	D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock	Macro	Macro	Macro	Macro	No	No	No
OFDX	Single- and Multiple-Output D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
OFDX4	Single- and Multiple-Output D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
OFDX8	Single- and Multiple-Output D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
OFDX16	Single- and Multiple-Output D Flip-Flop with Clock Enable	Macro	Macro	Macro	Macro	No	No	No
OFDX_1	Output D Flip-Flop with Inverted Clock and Clock Enable	Macro	Macro	Macro	Macro	No	No	No
OFDXI	Output D Flip-Flop with Clock Enable (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No
OFDXI_1	Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)	Macro	Macro	Macro	Macro	No	No	No

Input/Output Functions

Input/Output Block (IOB) resources are configured into various I/O primitives and macros for convenience, such as output buffers (s) and output buffers with an enable (OBUFES). Pads used to connect the circuit to PLD device pins are also included.

Virtex, Virtex-E, Spartan-II, Spartan-IIE, Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X have multiple variants (Primitives) to choose from for each SelectIO buffer. The I/O interface for each variant corresponds to a specific I/O standard, and those standards are detailed in Xilinx hardware documentation.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
GT_AURORA_n	Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT_CUSTOM	Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT_ETHERNET_n	Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
GT_FIBRE_CHAN_n	Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT_INFINIBAND_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT_XAUL_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_10GE_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_10GFC_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_AURORA_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_AURORAX_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_CUSTOM	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_OC48_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_OC192_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
GT10_PCI_EXPRESS_n	10-Gigabit Transceiver for High-Speed I/O	No	No	No	Primitive	No	No	No
IBUF	Single- and Multiple-Input Buffer	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IBUF4	Single- and Multiple-Input Buffer	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
IBUF8	Single- and Multiple-Input Buffer	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
IBUF16	Single- and Multiple-Input Buffer	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
IBUFDS	Differential Signaling Input Buffer with Selectable I/O Interface	No	Primitive	No	Primitive	No	No	No
IBUFG	Dedicated Input Buffer with Selectable I/O Interface	Primitive	Primitive	Primitive	Primitive	No	No	No
IBUFGDS	Dedicated Differential Signaling Input Buffer with Selectable I/O Interface	No	Primitive	No	Primitive	No	No	No
IOBUF	Bi-Directional Buffer with Selectable I/O Interface (multiple primitives)	Primitive	Primitive	Primitive	Primitive	No	No	No
IOBUFGDS	3-State Differential Signaling I/O Buffer with Active Low Output Enable	No	Primitive	No	Primitive	No	No	No
IOPAD	Single- and Multiple-Input/Output Pad	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IOPAD4	Single- and Multiple-Input/Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
IOPAD8	Single- and Multiple-Input/Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
IOPAD16	Single- and Multiple-Input/Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
IPAD	Single- and Multiple-Input Pad	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
IPAD4	Single- and Multiple-Input Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
IPAD8	Single- and Multiple-Input Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
IPAD16	Single- and Multiple-Input Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUF	Single- and Multiple-Output Buffer	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUF4	Single- and Multiple-Output Buffer	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUF8	Single- and Multiple-Output Buffer	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUF16	Single- and Multiple-Output Buffer	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFDS	Differential Signaling Output Buffer with Selectable I/O Interface	No	Primitive	No	Primitive	No	No	No
OBUE	3-State Output Buffers with Active-High Output Enable	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
OBUE4	3-State Output Buffers with Active-High Output Enable	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
OBUE8	3-State Output Buffers with Active-High Output Enable	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
OBUE16	3-State Output Buffers with Active-High Output Enable	Macro	No	Macro	Macro	Primitive	Primitive	Primitive
OBUFT	Single and Multiple 3-State Output Buffer with Active Low Output Enable	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OBUFT4	Single and Multiple 3-State Output Buffer with Active Low Output Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFT8	Single and Multiple 3-State Output Buffer with Active Low Output Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFT16	Single and Multiple 3-State Output Buffer with Active Low Output Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OBUFTDS	3-State Output Buffer with Differential Signaling, Active-Low Output Enable, and Selectable I/O Interface	No	Primitive	No	Primitive	No	No	No
OPAD	Single- and Multiple-Output Pad	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OPAD4	Multiple-Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OPAD8	Multiple-Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OPAD16	Multiple-Output Pad	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
UPAD	Connects the I/O Node of an IOB to the Internal PLD Circuit	Primitive	Primitive	Primitive	Primitive	No	No	No

Latches

Latches (LD) are available for all architectures.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
LD	Transparent Data Latch	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
LD_1	Transparent Data Latch with Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LD4	Multiple Transparent Data Latch	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
LD8	Multiple Transparent Data Latch	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
LD16	Multiple Transparent Data Latch	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
LDC	Transparent Data Latch with Asynchronous Clear	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
LDC_1	Transparent Data Latch with Asynchronous Clear and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LD4CE	Transparent Data Latch with Asynchronous Clear and Gate Enable	Macro	Macro	Macro	Macro	No	No	No
LD8CE	Transparent Data Latch with Asynchronous Clear and Gate Enable	Macro	Macro	Macro	Macro	No	No	No
LD16CE	Transparent Data Latch with Asynchronous Clear and Gate Enable	Macro	Macro	Macro	Macro	No	No	No
LDCE	Transparent Data Latch with Asynchronous Clear and Gate Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
LDCE_1	Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LDCP	Transparent Data Latch with Asynchronous Clear and Preset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
LDCP_1	Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LDCPE	Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
LDCPE_1	Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LDE	Transparent Data Latch with Gate Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
LDE_1	Transparent Data Latch with Gate Enable and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LDG	Transparent Datagate Latch	No	No	No	No	No	No	Primitive
LDG4	Multiple Transparent Datagate Latch	No	No	No	No	No	No	Primitive
LDG8	Multiple Transparent Datagate Latch	No	No	No	No	No	No	Primitive
LDG16	Multiple Transparent Datagate Latch	No	No	No	No	No	No	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
LDP	Transparent Data Latch with Asynchronous Preset	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
LDP_1	Transparent Data Latch with Asynchronous Preset and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No
LDPE	Transparent Data Latch with Asynchronous Preset and Gate Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
LDPE_1	Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate	Primitive	Primitive	Primitive	Primitive	No	No	No

Logic Primitives

Combinatorial logic gates that implement the basic Boolean functions are available in all architectures with up to five inputs in all combinations of inverted and non-inverted inputs, and with six to nine inputs non-inverted.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
AND2	2-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND2B1	2-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND2B2	2-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND3	3-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND3B1	3-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND3B2	3-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND3B3	3-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND4	4-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND4B1	4-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND4B2	4-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND4B3	4-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND4B4	4-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
AND5B1	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5B2	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5B3	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5B4	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND5B5	5-Input AND Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
AND6	6-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND7	7-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND8	8-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND9	9-Input AND Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
AND12	12- Input AND Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
AND16	16- Input AND Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
INV	Single and Multiple Inverters	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
INV4	Single and Multiple Inverters	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
INV8	Single and Multiple Inverters	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
INV16	Single and Multiple Inverters	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
MULT_AND	Fast Multiplier AND	Primitive	Primitive	Primitive	Primitive	No	No	No
NAND2	2-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND2B1	2-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND2B2	2-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND3	3-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND3B1	3-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND3B2	3-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND3B3	3-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND4	4-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
NAND4B1	4-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND4B2	4-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND4B3	4-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND4B4	4-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5B1	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5B2	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5B3	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5B4	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND5B5	5-Input NAND Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NAND6	6-Input NAND Gate with Inverted and Non-Inverted Inputs.	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NAND7	7-Input NAND Gate with Inverted and Non-Inverted Inputs.	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NAND8	8-Input NAND Gate with Inverted and Non-Inverted Inputs.	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NAND9	9-Input NAND Gate with Inverted and Non-Inverted Inputs.	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NAND12	12- Input NAND Gate with Non-Inverted Inputs.	Macro	Macro	Macro	Macro	No	No	No
NAND16	16- Input NAND Gate with Non-Inverted Inputs.	Macro	Macro	Macro	Macro	No	No	No
NOR2	2- Input NOR Gate with Inverted and Non-Inverted Inputs.	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR2B1	2- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR2B2	2- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR3	3- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR3B1	3- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
NOR3B2	3- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR3B3	3- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR4	4- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR4B1	4- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR4B2	4- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR4B3	4- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR4B4	4- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5B1	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5B2	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5B3	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5B4	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR5B5	5- Input NOR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
NOR6	6- Input NOR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NOR7	7- Input NOR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NOR8	8- Input NOR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NOR9	9- Input NOR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
NOR12	12-Input NOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
NOR16	16-Input NOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
OR2	2-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR2B1	2-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
OR2B2	2-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR3	3-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR3B1	3-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR3B2	3-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR3B3	3-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR4	4-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR4B1	4-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR4B2	4-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR4B3	4-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5B1	5-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5B2	12-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5B3	5-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5B4	5-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR5B5	5-Input OR Gate with Inverted and Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
OR6	6-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OR7	6-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OR8	8-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OR9	9-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
OR12	12-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
OR16	16-Input OR Gate with Inverted and Non-Inverted Inputs	Macro	Macro	Macro	Macro	No	No	No
ORCY	OR with Carry Logic	No	No	No	Primitive	No	No	No
SOP3	Sum of Products	Macro	Macro	Macro	Macro	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
SOP3B1A	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP3B1B	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP3B2A	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP3B2B	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP3B3	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4B3	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4B4	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4B1	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4B2A	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
SOP4B2B	Sum of Products	Macro	Macro	Macro	Macro	No	No	No
XNOR2	2-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR3	3-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR4	4-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR5	5-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR6	6-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR7	7-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR8	8-Input XNOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XNOR9	9-Input XNOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XOR2	2-Input XOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XOR3	3-Input XOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XOR4	4-Input XOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR5	5-Input XOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR6	6-Input XOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XOR7	7-Input XOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XOR8	8-Input XOR Gate with Non-Inverted Inputs	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive	Primitive
XOR9	9-Input XOR Gate with Non-Inverted Inputs	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
XORCY	XOR for Carry Logic with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
XORCY_D	XOR for Carry Logic with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
XORCY_L	XOR for Carry Logic with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No

Map Elements

Map elements are used in conjunction with logic symbols to constrain the logic to particular CLBs or particular F function generators.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
FMAP	F Function Generator Partitioning Control Symbol	Primitive	Primitive	Primitive	Primitive	No	No	No

Memory Elements

In the Virtex, Virtex-E, Spartan-II, and Spartan-IIE, Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X architectures, a number of static RAMs are defined as primitives. These 16- or 32-word RAMs are 1, 2, 4, and 8 bits wide.

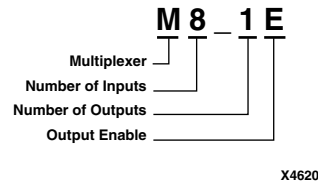
The Virtex, Virtex-E, Spartan-II, and Spartan-IIE, Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X architectures have dedicated blocks of on-chip 4096-bit single-port and dual-port synchronous RAM. Each port is configured to a specific data width. There are five single-port block RAM primitives and 30 dual-port block RAM primitives.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
RAM16X1D	16-Deep by 1-Wide Static Dual Port Synchronous RAM	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM16X1D_1	16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM16X1S	16-Deep by 1-Wide Static Synchronous RAM	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM16X1S_1	16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM16X2D	16-Deep by 2-Wide Static Dual Port Synchronous RAM	Macro	No	Macro	Macro	No	No	No
RAM16X2S	16-Deep by 2-Wide Static Synchronous RAM	Macro	Primitive	Macro	Primitive	No	No	No
RAM16X4D	16-Deep by 4-Wide Static Dual Port Synchronous RAM	Macro	No	Macro	Macro	No	No	No
RAM16X4S	16-Deep by 4-Wide Static Synchronous RAM	Macro	Primitive	Macro	Primitive	No	No	No
RAM16X8D	16-Deep by 8-Wide Static Dual Port Synchronous RAM	Macro	No	Macro	Macro	No	No	No
RAM16X8S	16-Deep by 8-Wide Static Synchronous RAM	Macro	No	Macro	Primitive	No	No	No
RAM32X1D	32-Deep by 1-Wide Static Dual Static Port Synchronous RAM	No	No	No	Primitive	No	No	No
RAM32X1D_1	32-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock	No	No	No	Primitive	No	No	No
RAM32X1S	32-Deep by 1-Wide Static Synchronous RAM	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM32X1S_1	32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock	Primitive	Primitive	Primitive	Primitive	No	No	No
RAM32X2S	32-Deep by 2-Wide Static Synchronous RAM	Macro	Primitive	Macro	Primitive	No	No	No
RAM32X4S	32-Deep by 4-Wide Static Synchronous RAM	Macro	No	Macro	Primitive	No	No	No

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
RAM32X8S	32-Deep by 8-Wide Static Synchronous RAM	Macro	No	Macro	Primitive	No	No	No
RAM64X1D	64-Deep by 1-Wide Dual Port Static Synchronous RAM	No	No	No	Primitive	No	No	No
RAM64X1D_1	64-Deep by 1-Wide Dual Port Static Synchronous RAM with Negative-Edge Clock	No	No	No	Primitive	No	No	No
RAM64X1S	64-Deep by 1-Wide Static Synchronous RAM	No	Primitive	No	Primitive	No	No	No
RAM64X1S_1	64-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock	No	Primitive	No	Primitive	No	No	No
RAM64X2S	64-Deep by 2-Wide Static Synchronous RAM	No	No	No	Primitive	No	No	No
RAM128X1S	128-Deep by 1-Wide Static Synchronous RAM	No	No	No	Primitive	No	No	No
RAM128X1S_1	128-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock	No	No	No	Primitive	No	No	No
RAMB4_Sm_Sn	4096-Bit Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 8, or 16 Bits	Primitive	No	Primitive	No	No	No	No
RAMB4_Sn	4096-Bit Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits	Primitive	No	Primitive	No	No	No	No
RAMB16_Sm_Sn	16384-Bit Data Memory and 2048-Bit Parity Memory, Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 9, 18, or 36 Bits	No	Primitive	No	Primitive	No	No	No
RAMB16_Sn	16384-Bit Data Memory and 2048-Bit Parity Memory, Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 9, 18, or 36 Bits	No	Primitive	No	Primitive	No	No	No
ROC	Reset On Configuration	Primitive	Primitive	Primitive	Primitive	No	No	No
ROCBUF	Reset On Configuration Buffer	Primitive	Primitive	Primitive	Primitive	No	No	No
ROM16X1	16-Deep by 1-Wide ROM	Primitive	Primitive	Primitive	Primitive	No	No	No
ROM32X1	32-Deep by 1-Wide ROM	Primitive	Primitive	Primitive	Primitive	No	No	No
ROM64X1	64-Deep by 1-Wide ROM	No	Primitive	No	Primitive	No	No	No
ROM128X1	128-Deep by 1-Wide ROM	No	Primitive	No	Primitive	No	No	No
ROM256X1	256-Deep by 1-Wide ROM	No	Primitive	No	Primitive	No	No	No

Multiplexers

The multiplexer naming convention shown in the following figure indicates the number of inputs and outputs and whether or not an enable is available.



Multiplexer Naming Convention

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
M2_1	2-to-1 Multiplexer	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M2_1B1	2-to-1 Multiplexer with D0 Inverted	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M2_1B2	2-to-1 Multiplexer with D0 and D1 Inverted	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M2_1E	2-to-1 Multiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M4_1E	4-to-1 Multiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M8_1E	8-to-1 Multiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
M16_1E	16-to-1 Multiplexer with Enable	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
MUXCY	2-to-1 Multiplexer for Carry Logic with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXCY_D	2-to-1 Multiplexer for Carry Logic with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXCY_L	2-to-1 Multiplexer for Carry Logic with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF5	2-to-1 Lookup Table Multiplexer with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF5_D	2-to-1 Lookup Table Multiplexer with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF5_L	2-to-1 Lookup Table Multiplexer with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF6	2-to-1 Lookup Table Multiplexer with General Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF6_D	2-to-1 Lookup Table Multiplexer with Dual Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF6_L	2-to-1 Lookup Table Multiplexer with Local Output	Primitive	Primitive	Primitive	Primitive	No	No	No
MUXF7	2-to-1 Lookup Table Multiplexer with General Output	No	Primitive	No	Primitive	No	No	No
MUXF7_D	2-to-1 Lookup Table Multiplexer with Dual Output	No	Primitive	No	Primitive	No	No	No
MUXF7_L	2-to-1 Lookup Table Multiplexer with Local Output	No	Primitive	No	Primitive	No	No	No
MUXF8	2-to-1 Lookup Table Multiplexer with General Output	No	Primitive	No	Primitive	No	No	No
MUXF8_D	2-to-1 Lookup Table Multiplexer with Dual Output	No	Primitive	No	Primitive	No	No	No
MUXF8_L	2-to-1 Lookup Table Multiplexer with Local Output	No	Primitive	No	Primitive	No	No	No

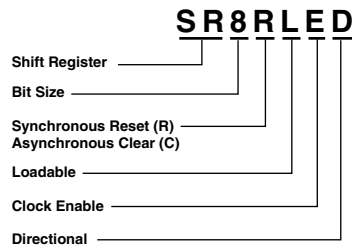
Shifters

Shifters are barrel shifters (BRLSHFT) of four and eight bits.

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
BRLSHFT4	4-Bit Barrel Shifter	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
BRLSHFT8	8-Bit Barrel Shifter	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Shift Registers

Shift registers are available in a variety of sizes and capabilities. The naming convention shown in the following figure illustrates available features.



X4578

Shift Register Naming Convention

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
SR4CE	4-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR8CE	8-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16CE	16-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR4CLE	4-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR8CLE	8-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16CLE	16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR4CED	4-Bit Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR8CED	8-Bit Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16CED	16-Bit Shift Register with Clock Enable and Asynchronous Clear	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR4RE	4-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
SR8RE	8-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16RE	16-Bit Serial-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR4RLE	4-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR8RLE	8-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16RLE	16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR4RLED	4-Bit Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR8RLED	8-Bit Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SR16RLED	16-Bit Shift Register with Clock Enable and Synchronous Reset	Macro	Macro	Macro	Macro	Primitive	Primitive	Primitive
SRD4CE	4-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD8CE	8-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD16CE	16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD4CLE	4-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD8CLE	8-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD16CLE	16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD4CLED	4-Bit Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD8CLED	8-Bit Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD16CLED	16-Bit Dual Edge Triggered Shift Register with Clock Enable and Asynchronous Clear	No	No	No	No	No	No	Primitive
SRD4RE	4-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive

Design Element	Description	Spartan-II, IIE	Spartan-3	Virtex, E	Virtex II, Pro, Pro X	XC9500/XV/XL	CR XPLA3	CR-II
SRD8RE	8-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD16RE	16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD4RLE	4-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD8RLE	8-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD16RLE	16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD4RLED	4-Bit Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD8RLED	8-Bit Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRD16RLED	16-Bit Dual Edge Triggered Shift Register with Clock Enable and Synchronous Reset	No	No	No	No	No	No	Primitive
SRL16	16-Bit Shift Register Look-Up Table (LUT)	Primitive	Primitive	Primitive	Primitive	No	No	No
SRL16_1	16-Bit Shift Register Look-Up Table (LUT) with Negative-Edge Clock	Primitive	Primitive	Primitive	Primitive	No	No	No
SRL16E	16-Bit Shift Register Look-Up Table (LUT) with Clock Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
SRL16E_1	16-Bit Shift Register Look-Up Table (LUT) with Negative-Edge Clock and Clock Enable	Primitive	Primitive	Primitive	Primitive	No	No	No
SRLC16	16-Bit Shift Register Look-Up Table (LUT) with Carry	No	Primitive	No	Primitive	No	No	No
SRLC16_1	16-Bit Shift Register Look-Up Table (LUT) with Carry and Negative-Edge Clock	No	Primitive	No	Primitive	No	No	No
SRLC16E	16-Bit Shift Register Look-Up Table (LUT) with Carry and Clock Enable	No	Primitive	No	Primitive	No	No	No
SRLC16E_1	16-Bit Shift Register Look-Up Table (LUT) with Carry, Negative-Edge Clock, and Clock Enable	No	Primitive	No	Primitive	No	No	No

Design Elements

The remaining sections in this book describe each design element that can be used with the supported architectures.

Design elements are organized in alphanumeric order, with all numeric suffixes in ascending order. For example, FDR precedes FDRS, and ADD4 precedes ADD8, which precedes ADD16.

The following information is provided for each library element, where applicable

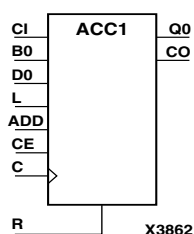
- Graphic symbol
- Applicability table (with primitive versus macro identification)
- Functional description
- Truth table
- Schematic for macros
- VHDL and Verilog instantiation and inference code
- Commonly used constraints

ACC1

1-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset

Architectures Supported

ACC1	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



ACC1 can add or subtract a 1-bit unsigned-binary word to or from the contents of a 1-bit data register and store the results in the register. The register can be loaded with a 1-bit word. The synchronous reset (R) has priority over all other inputs and, when High, causes the output to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low. The accumulator is asynchronously cleared, outputs Low, when power is applied. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Load

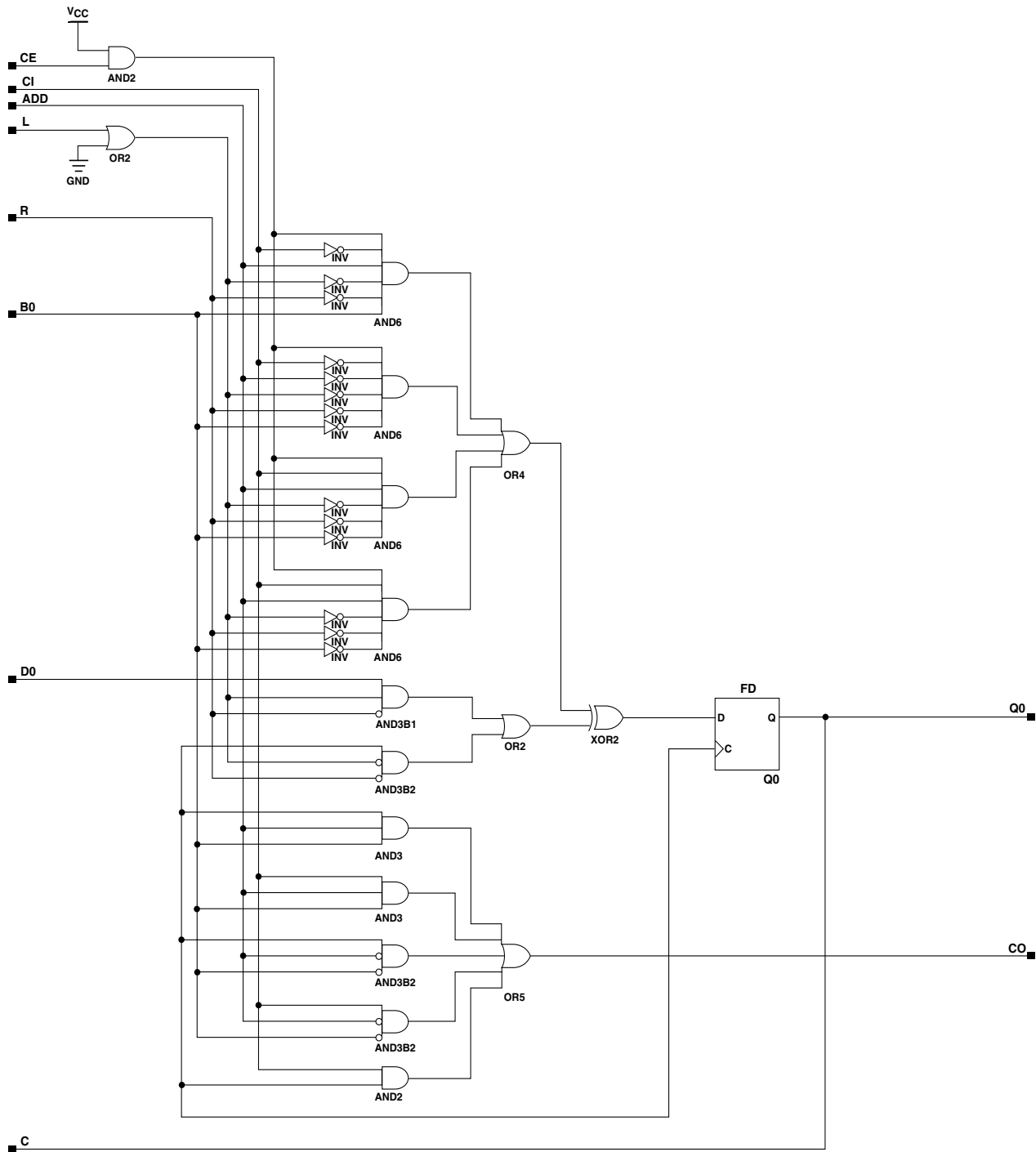
When the load input (L) is High, CE is ignored and the data on the input D0 is loaded into the 1-bit register during the Low-to-High clock (C) transition.

Add

When control inputs ADD and CE are both High, the accumulator adds a 1-bit word (B0) and carry-in (CI) to the contents of the 1-bit register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In add mode, CO acts as a carry-out, and CO and CI are active-High.

Subtract

When ADD is Low and CE is High, the 1-bit word B0 and CI are subtracted from the contents of the register. The result is stored in the register and appears on output Q0 during the Low-to-High clock transition. The carry-out (CO) is not registered synchronously with the data output. CO always reflects the accumulation of input B0 and the contents of the register, which allows cascading of ACC1s by connecting CO of one stage to CI of the next stage. In subtract mode, CO acts as a borrow, and CO and CI are active-Low.



X7688

ACC1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

ACC is schematic and inference only-- not instantiated.

VHDL Inference Code

Following is some "basic" code for inference of the ACC modules.

```
architecture Behavioral of acc1 is
begin
  process(C, R)
  begin
    if (R = '1') then
      Q <= (others => '0');
    elsif (C'event and C = '1') then
      if (L = '1') then
        Q <= D;
      elsif (CE = '1') then
        if (ADD = '1') then
          Q <= Q + B;
        else
          Q <= Q - B;
        end if;
      end if;
    end if;
  end process;
end Behavioral;
```

Verilog Inference Code

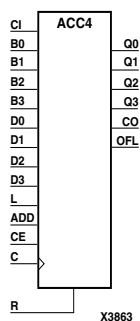
```
always @ (posedge C)
begin
  if (R)
    Q <= 0;
  else if (L)
    Q <= D;
  else if (CE)
    if (ADD)
      Q <= Q + B;
    else
      Q <= Q - B;
    end if;
end
```


ACC4, 8, 16

4-, 8-, 16-Bit Loadable Cascadable Accumulators with Carry-In, Carry-Out, and Synchronous Reset

Architectures Supported

ACC4, ACC8, ACC16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



ACC4, ACC8, ACC16 can add or subtract a 4-, 8-, 16-bit unsigned-binary, respectively or twos-complement word to or from the contents of a 4-, 8-, 16-bit data register and store the results in the register. The register can be loaded with the 4-, 8-, 16-bit word.

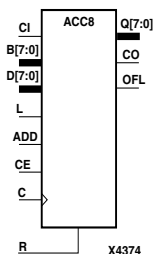
The synchronous reset (R) has priority over all other inputs, and when High, causes all outputs to go to logic level zero during the Low-to-High clock (C) transition. Clock (C) transitions are ignored when clock enable (CE) is Low.

The accumulator is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

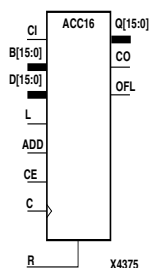
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



Load

When the load input (L) is High, CE is ignored and the data on the D inputs is loaded into the register during the Low-to-High clock (C) transition. ACC4 loads the data on inputs D3 – D0 into the 4-bit register. ACC8 loads the data on D7 – D0 into the 8-bit register. ACC16 loads the data on inputs D15 – D0 into the 16-bit register.



Unsigned Binary Versus Twos Complement

ACC4, ACC8, ACC16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned

binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

Unsigned Binary Operation

For unsigned binary operation, ACC4 can represent numbers between 0 and 15, inclusive; ACC8 between 0 and 255, inclusive; and ACC16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder/subtractor. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds. The carry-out (CO) is not registered synchronously with the data outputs. CO always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register. This allows cascading of ACC4s, ACC8s, or ACC16s by connecting CO of one stage to CI of the next stage. An unsigned binary “overflow” that is always active-High can be generated by gating the ADD signal and CO as follows.

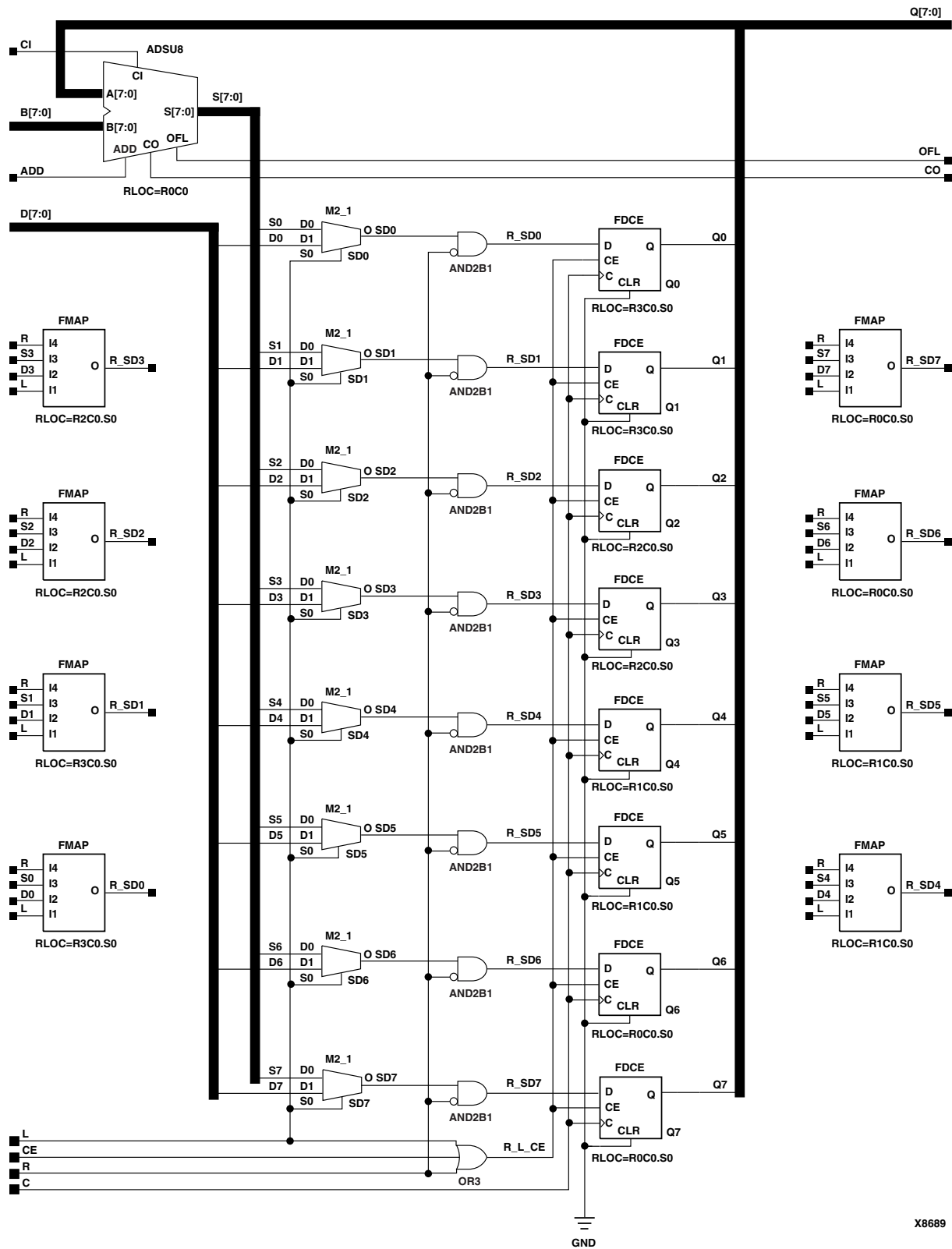
$$\text{unsigned overflow} = \text{CO XOR ADD}$$

Ignore OFL in unsigned binary operation.

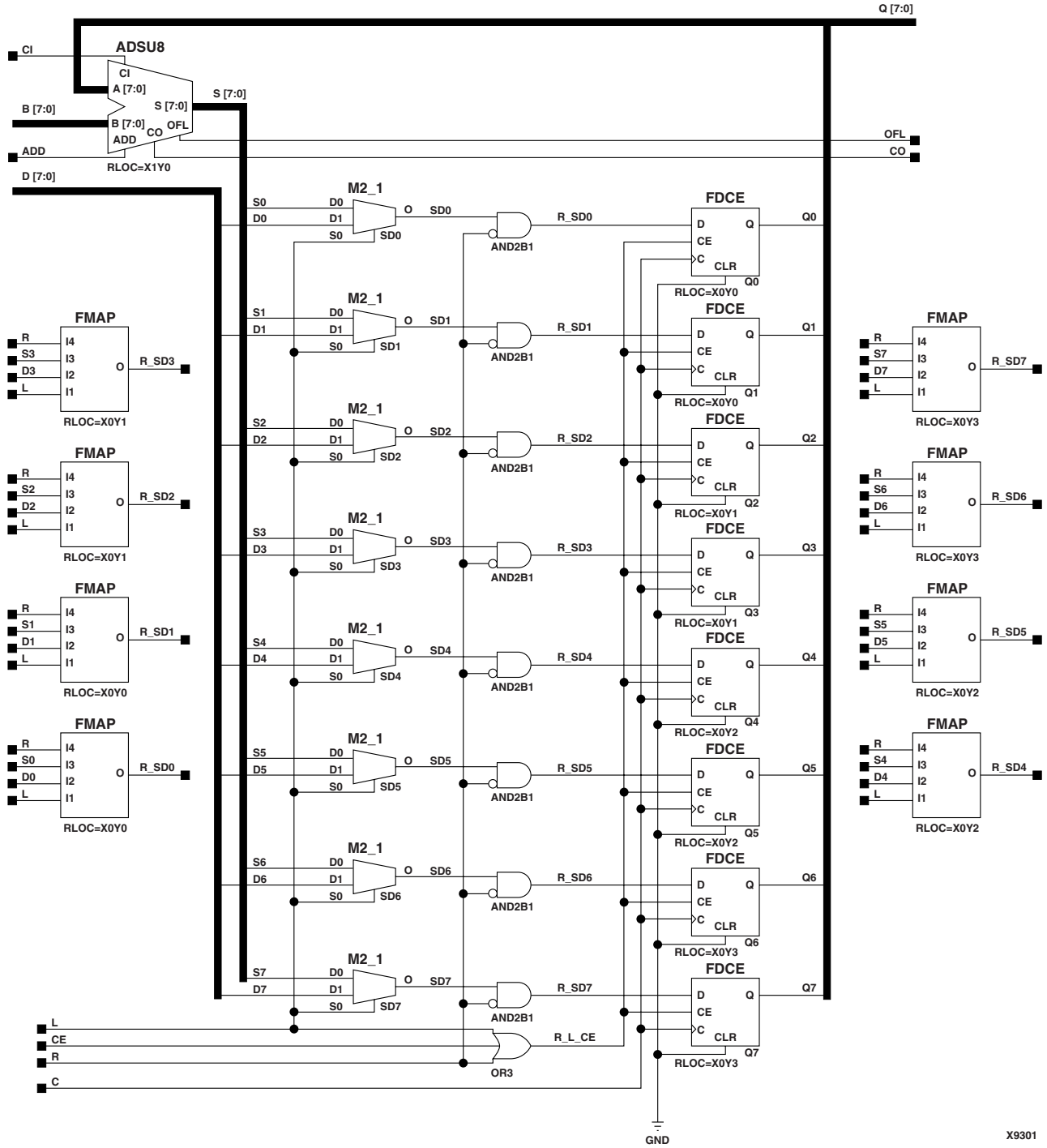
Twos-Complement Operation

For twos-complement operation, ACC4 can represent numbers between -8 and +7, inclusive; ACC8 between -128 and +127, inclusive; ACC16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High. The overflow (OFL) is not registered synchronously with the data outputs. OFL always reflects the accumulation of the B inputs (B3 – B0 for ACC4, B7 – B0 for ACC8, B15 – B0 for ACC16) and the contents of the register, which allows cascading of ACC4s, ACC8s, or ACC16s by connecting OFL of one stage to CI of the next stage.

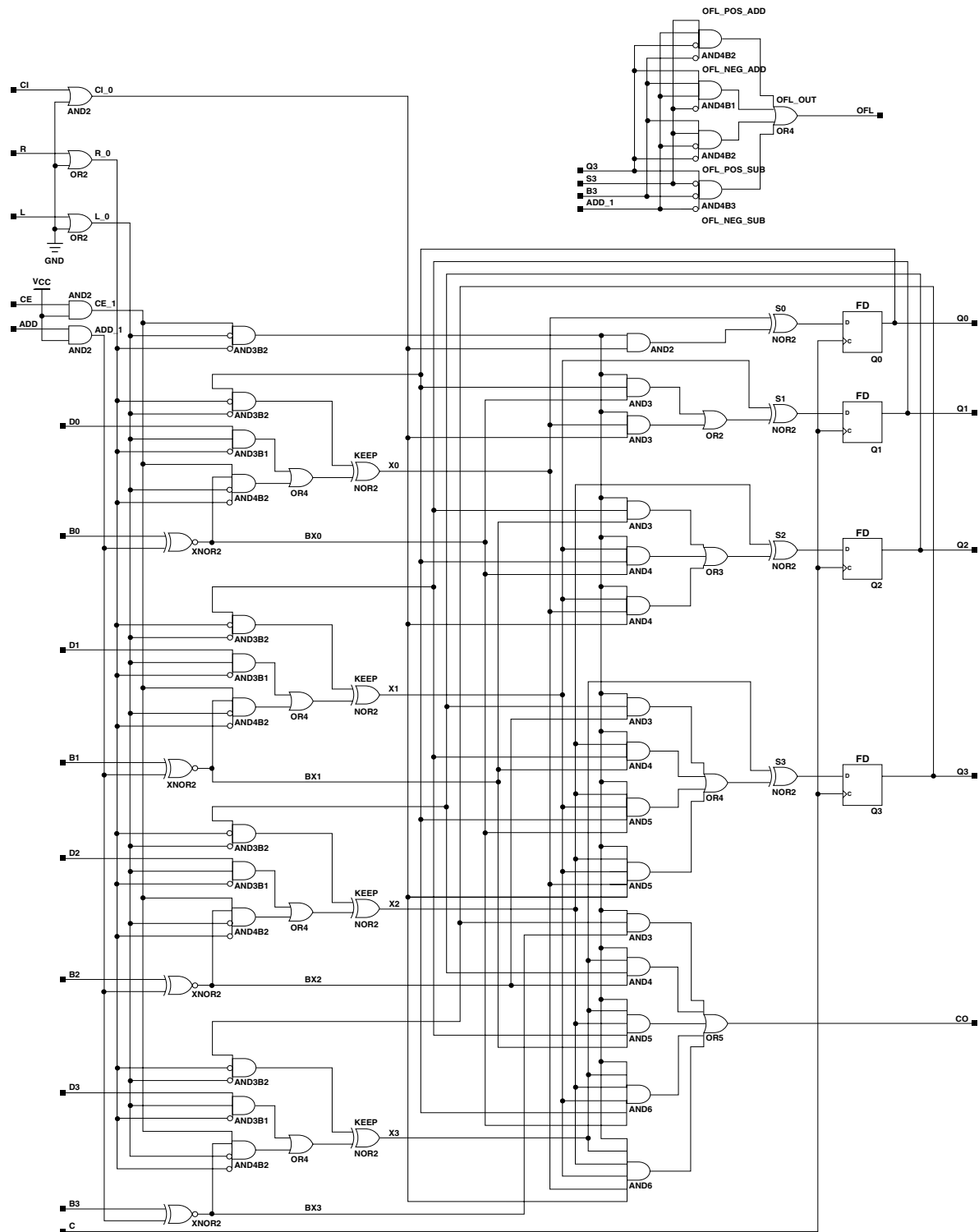
Ignore CO in twos-complement operation.



ACC8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E

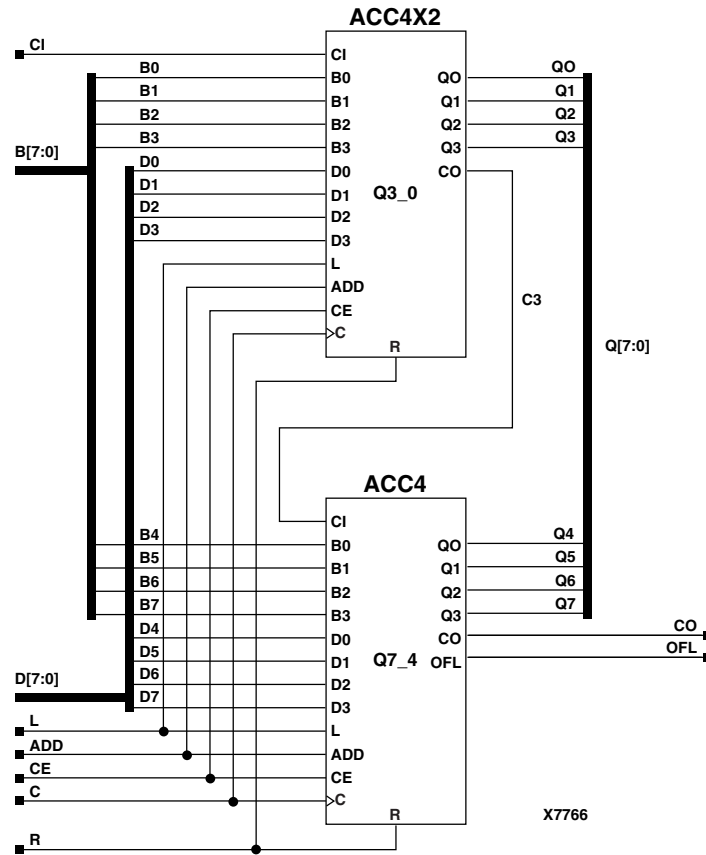


ACC8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7607

ACC4 Implementation XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II



ACC8 Implementation XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II

Usage

ACC is schematic and inference only -- not instantiated.

VHDL Inference Code (ACC4)

Following is some "basic" code for inference of the ACC modules.

```

architecture Behavioral of acc4 is
begin
  process(C)
  begin
    if (R = '1') then
      Q <= (others => '0');
    elsif (C'event and C = '1') then
      if (L = '1') then
        Q <= D;
      elsif (CE = '1') then
        if (ADD = '1') then
          Q <= Q + B;
        else
          Q <= Q - B;
        end if;
      end if;
    end if;
  end process;
end architecture Behavioral;

```

```
end process;  
end Behavioral;
```

Verilog Inference Code

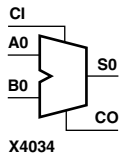
```
always @ (posedge C)  
begin  
  if (R)  
    Q <= 0;  
  else if (L)  
    Q <= D;  
  else if (CE)  
end  
  if (ADD)  
    Q <= Q + B;  
  else  
    Q <= Q - B;  
end
```


ADD1

1-Bit Full Adder with Carry-In and Carry-Out

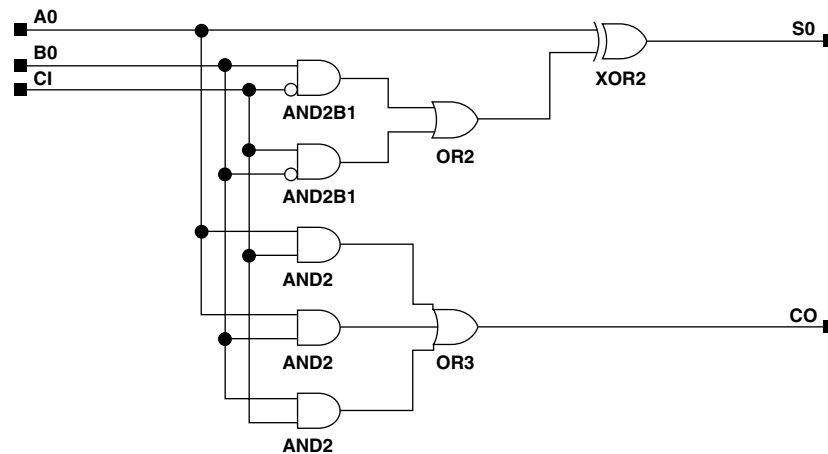
Architectures Supported

ADD1	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



ADD1 is a cascadable 1-bit full adder with carry-in and carry-out. It adds two 1-bit words (A and B) and a carry-in (CI), producing a binary sum (S0) output and a carry-out (CO).

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1



X7689

ADD1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is schematic or inference only -- no instantiation.

VHDL Inference Code

architecture Behavioral of ADD is

```
signal sum: std_logic_vector(WIDTH downto 0);
signal zeros: std_logic_vector(WIDTH-1 downto 0) := (others => '0');
```

```
begin
```

```
process (CI, A, B, sum)
```

```
begin
```

```
sum <= ('0' & A) + ('0' & B) + (zeros & CI);
```

```
S <= sum(WIDTH-1 downto 0);
```

```
CO <= sum(WIDTH);
```

```
end process;
```

```
end Behavioral;
```

Verilog Inference Code

```
always @ (A or B or CI)
```

```
begin
```

```
{CO,S} <= A + B + CI;
```

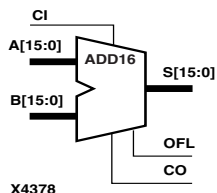
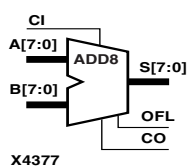
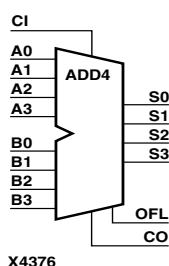
```
end
```


ADD4, 8, 16

4-, 8-, 16-Bit Cascadable Full Adders with Carry-In, Carry-Out, and OverflowA

Architectures Supported

ADD4, 8, 16	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



ADD4, ADD8, and ADD16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADD4 adds $A3 - A0$, $B3 - B0$, and CI producing the sum output $S3 - S0$ and CO (or OFL). ADD8 adds $A7 - A0$, $B7 - B0$, and CI, producing the sum output $S7 - S0$ and CO (or OFL). ADD16 adds $A15 - A0$, $B15 - B0$ and CI, producing the sum output $S15 - S0$ and CO (or OFL).

Unsigned Binary Versus Twos Complement

ADD4, ADD8, ADD16 can operate on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers, respectively. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when "overflow" occurs. Unsigned binary uses CO, while twos-complement uses OFL to determine when "overflow" occurs. To interpret the inputs as unsigned binary, follow the CO output. To interpret the inputs as twos complement, follow the OFL output.

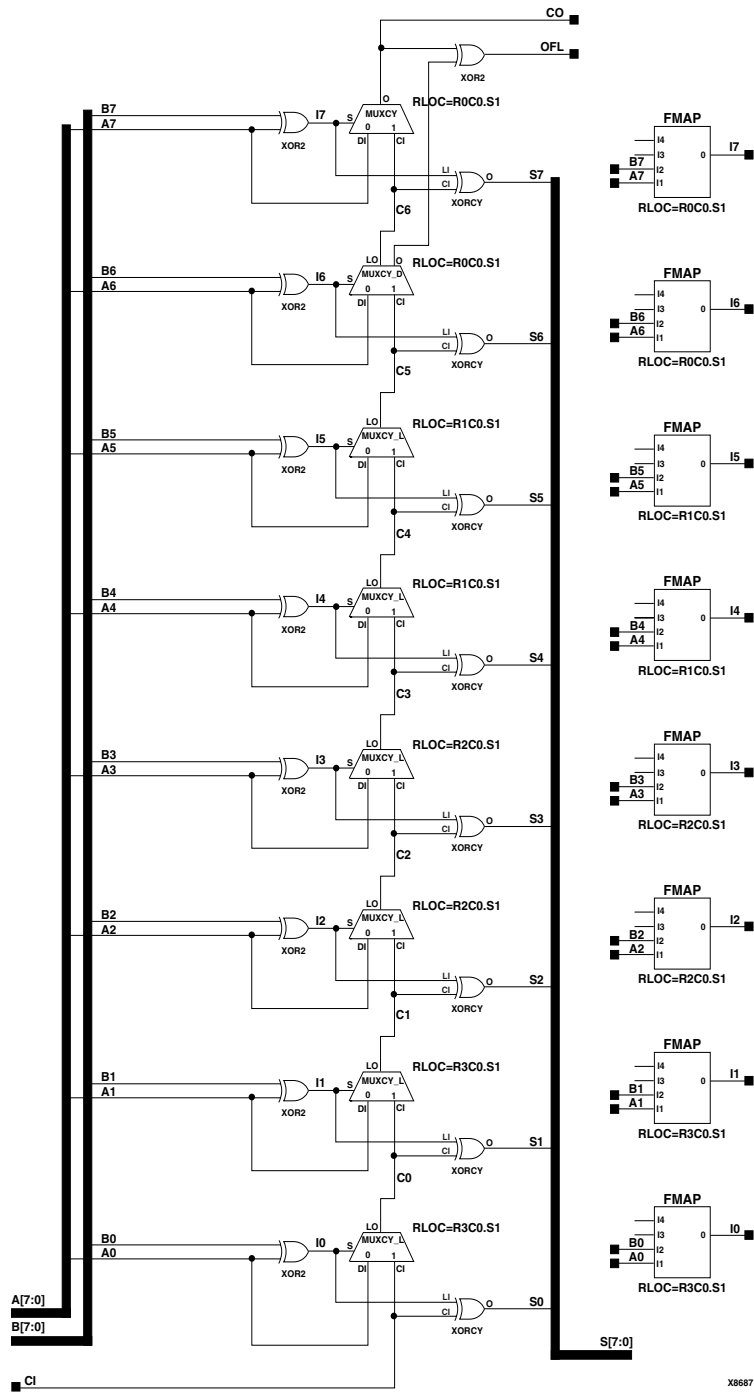
Unsigned Binary Operation

For unsigned binary operation, ADD4 can represent numbers between 0 and 15, inclusive; ADD8 between 0 and 255, inclusive; ADD16 between 0 and 65535, inclusive. CO is active (High) when the sum exceeds the bounds of the adder.

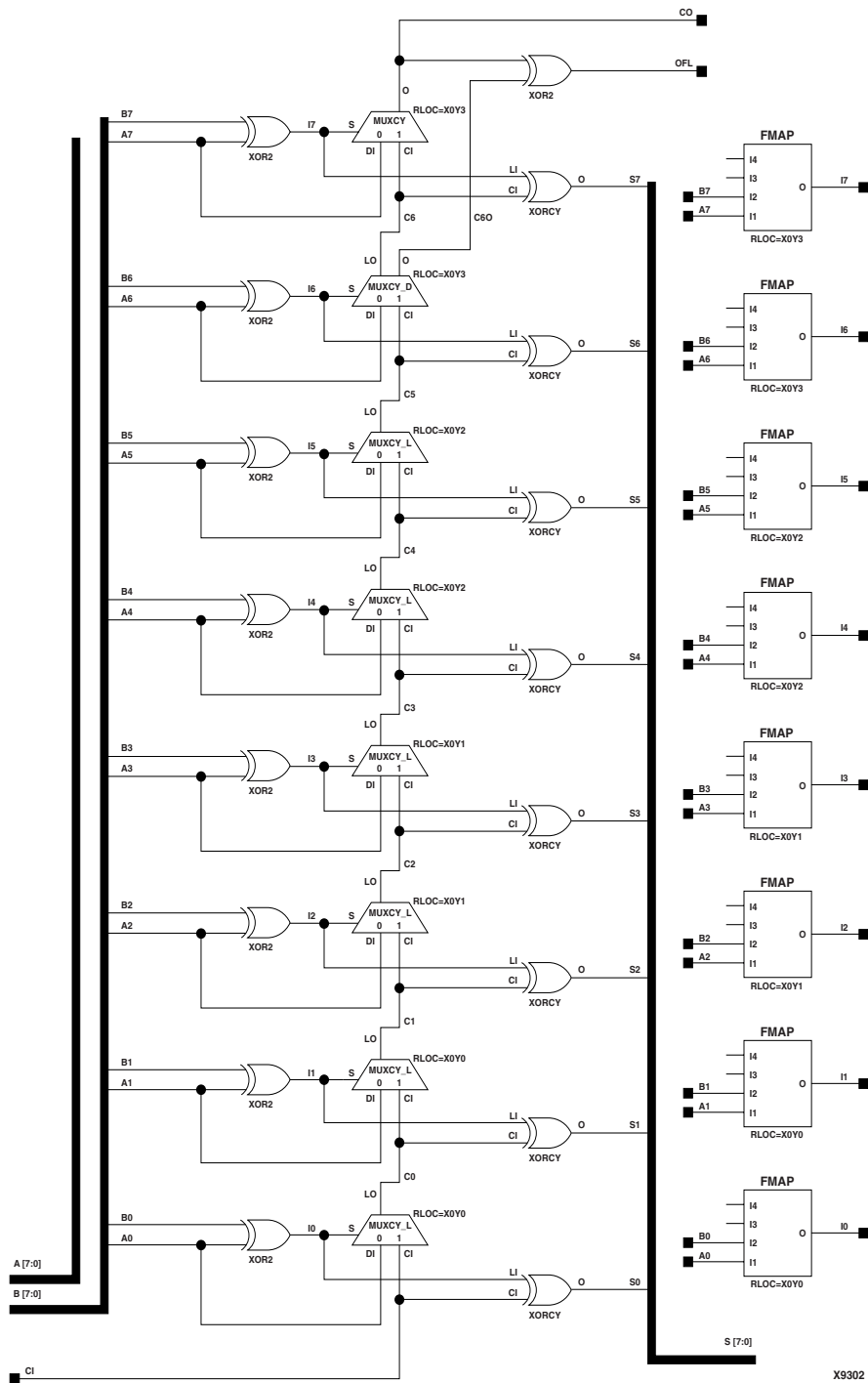
OFL is ignored in unsigned binary operation.

Twos-Complement Operation

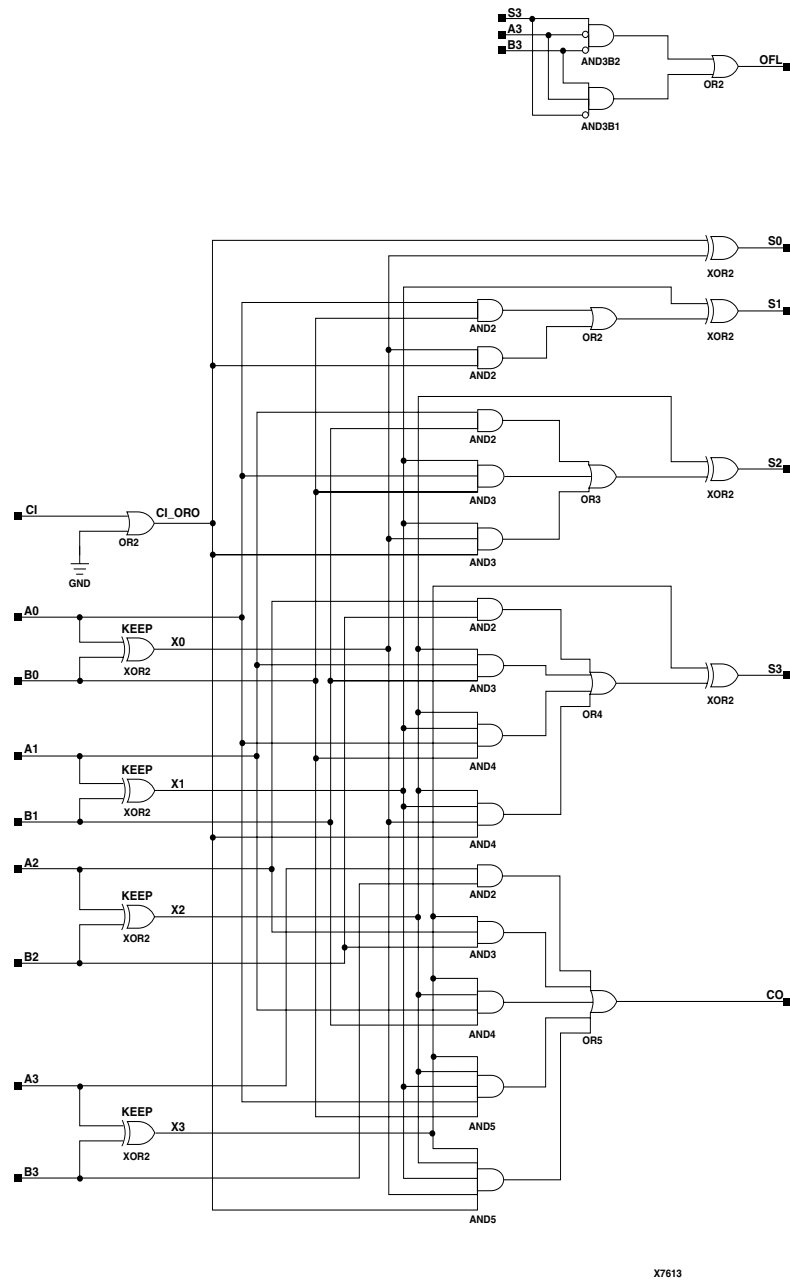
For twos-complement operation, ADD4 can represent numbers between -8 and +7, inclusive; ADD8 between -128 and +127, inclusive; ADD16 between -32768 and +32767, inclusive. OFL is active (High) when the sum exceeds the bounds of the adder. CO is ignored in twos-complement operation.



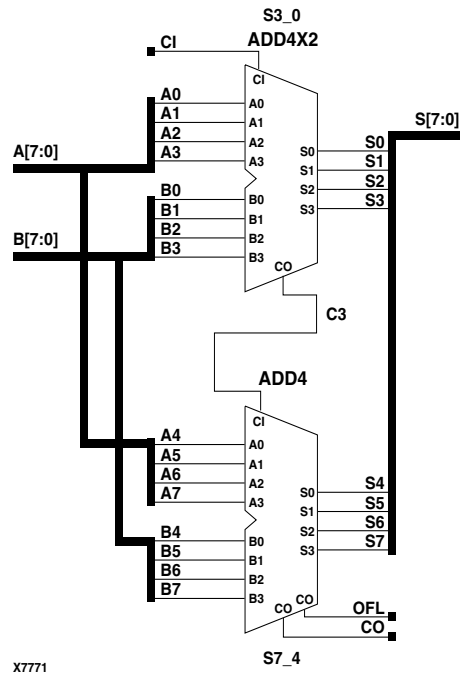
ADD8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ADD8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



ADD4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



ADD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is schematic or inference only -- no instantiation.

VHDL Inference Code (ADD4)

architecture Behavioral of ADD is

```
signal sum: std_logic_vector(WIDTH-1 downto 0);
signal zeros: std_logic_vector(WIDTH-1 downto 0) := (others => '0');
```

```
begin
```

```
process (CI, A, B, sum)
```

```
begin
```

```
sum <= ('0' & A) + ('0' & B) + (zeros & CI);
```

```
S <= sum(WIDTH-1 downto 0);
```

```
CO <= sum(WIDTH);
```

```
end process;
```

```
end Behavioral;
```

Verilog Inference Code (ADD4)

```
always @ (A or B or CI)
```

```
begin
```

```
{CO,sum} <= A + B + CI;
```

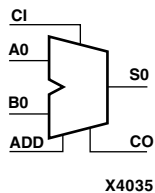
```
end
```


ADSU1

1-Bit Cascadable Adder/Subtractor with Carry-In and Carry-Out

Architectures Supported

ADSU1	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



When the ADD input is High, two 1-bit words (A0 and B0) are added with a carry-in (CI), producing a 1-bit output (S0) and a carry-out (CO). When the ADD input is Low, B0 is subtracted from A0, producing a result (S0) and borrow (CO). In add mode, CO represents a carry-out, and CO and CI are active-High. In subtract mode, CO represents a borrow, and CO and CI are active-Low.

Add Function, ADD=1

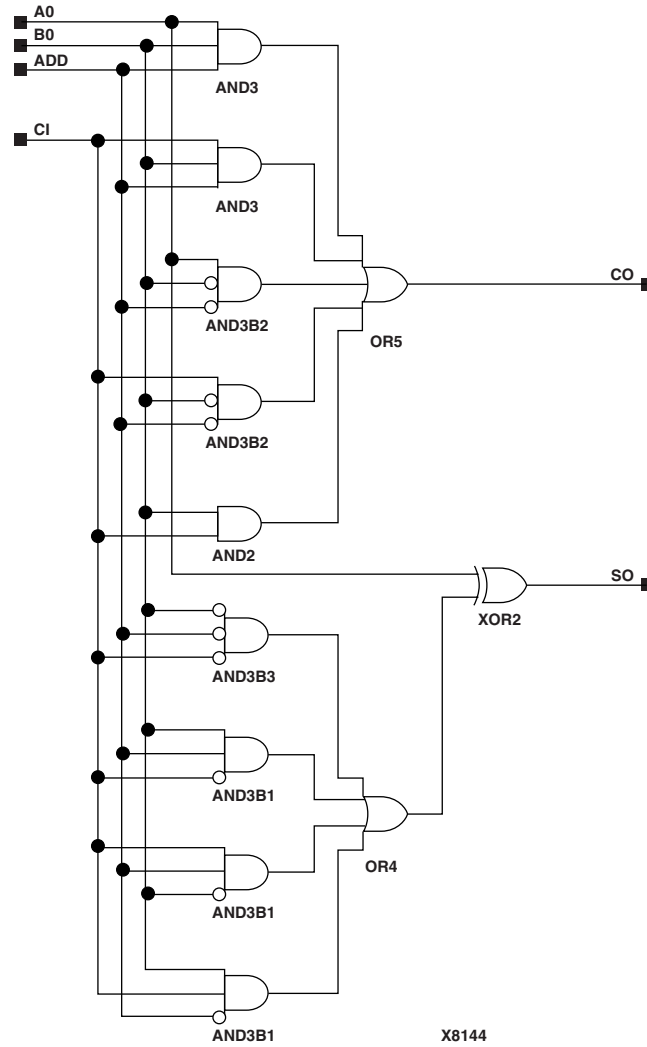
Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Subtract Function, ADD=0

Inputs			Outputs	
A0	B0	CI	S0	CO
0	0	0	1	0
0	1	0	0	0
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0

Subtract Function, ADD=0

Inputs			Outputs	
A0	B0	CI	S0	CO
1	0	1	1	1
1	1	1	0	1



ADSU1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

VHDL Instantiation Template

```
-- Component Declaration for ADSU1 should be placed
-- after architecture statement but before begin keyword
```

```
component ADSU1
```



```
        port (CO : out STD_ULOGIC;
              S0 : out STD_ULOGIC;
              A0 : in  STD_ULOGIC;
              ADD: in  STD_ULOGIC;
              B0 : in  STD_ULOGIC;
              CI : in  STD_ULOGIC);
end component;

-- Component Attribute specification for ADSU1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for ADSU1 should be placed
-- in architecture after the begin keyword

ADSU1_INSTANCE_NAME : ADSU1
-- synthesis translate_off
generic map (CDS_ACTION => "string_value")
-- synthesis translate_on
port map (CO =>user_CO,
          S0 => user_SO,
          A0 => user_A0,
          ADD => user_ADD,
          B0 => user_B0,
          CI => user_CI);
```

Verilog Instantiation Template

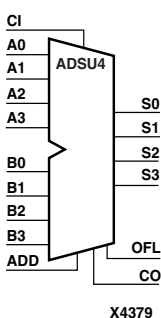
```
ADSU1 instance_name (.CO (user_CO),
                    .S0 (user_SO),
                    .A0 (user_A0),
                    .ADD (user_ADD),
                    .B0 (user_B0),
                    .CI (user_CI));
```


ADSU4, 8, 16

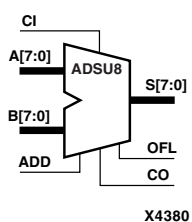
4-, 8-, 16-Bit Cascadable Adders/Subtractors with Carry-In, Carry-Out, and Overflow

Architectures Supported

ADSU4, ADSU8, ADSU16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



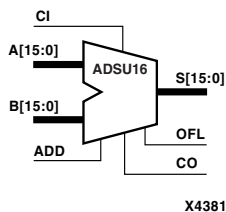
When the ADD input is High, ADSU4, ADSU8, and ADSU16 add two words and a carry-in (CI), producing a sum output and carry-out (CO) or overflow (OFL). ADSU4 adds two 4-bit words ($A_3 - A_0$ and $B_3 - B_0$) and a CI, producing a 4-bit sum output ($S_3 - S_0$) and CO or OFL. ADSU8 adds two 8-bit words ($A_7 - A_0$ and $B_7 - B_0$) and a CI producing, an 8-bit sum output ($S_7 - S_0$) and CO or OFL. ADSU16 adds two 16-bit words ($A_{15} - A_0$ and $B_{15} - B_0$) and a CI, producing a 16-bit sum output ($S_{15} - S_0$) and CO or OFL.



When the ADD input is Low, ADSU4, ADSU8, and ADSU16 subtract $B_z - B_0$ from $A_z - A_0$, producing a difference output and CO or OFL. ADSU4 subtracts $B_3 - B_0$ from $A_3 - A_0$, producing a 4-bit difference ($S_3 - S_0$) and CO or OFL. ADSU8 subtracts $B_7 - B_0$ from $A_7 - A_0$, producing an 8-bit difference ($S_7 - S_0$) and CO or OFL. ADSU16 subtracts $B_{15} - B_0$ from $A_{15} - A_0$, producing a 16-bit difference ($S_{15} - S_0$) and CO or OFL.

In add mode, CO and CI are active-High. In subtract mode, CO and CI are active-Low. OFL is active-High in add and subtract modes.

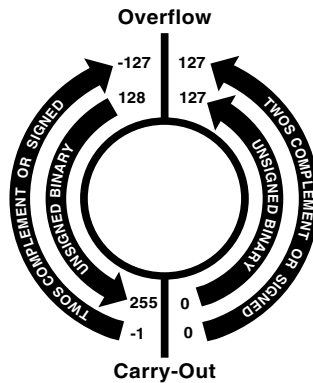
ADSU4, ADSU8, and ADSU16 CI and CO pins do not use the CPLD carry chain.



Unsigned Binary Versus Twos Complement

ADSU4, ADSU8, ADSU16 can operate, respectively, on either 4-, 8-, 16-bit unsigned binary numbers or 4-, 8-, 16-bit twos-complement numbers. If the inputs are interpreted as unsigned binary, the result can be interpreted as unsigned binary. If the inputs are interpreted as twos complement, the output can be interpreted as twos complement. The only functional difference between an unsigned binary operation and a twos-complement operation is how they determine when “overflow” occurs. Unsigned binary uses CO, while twos complement uses OFL to determine when “overflow” occurs.

With adder/subtractors, either unsigned binary or twos-complement operations cause an overflow. If the result crosses the overflow boundary, an overflow is generated. Similarly, when the result crosses the carry-out boundary, a carry-out is generated. The following figure shows the ADSU carry-out and overflow boundaries.



X4720

ADSU Carry-Out and Overflow Boundaries

Unsigned Binary Operation

For unsigned binary operation, ADSU4 can represent numbers between 0 and 15, inclusive; ADSU8 between 0 and 255, inclusive; ADSU16 between 0 and 65535, inclusive. In add mode, CO is active (High) when the sum exceeds the bounds of the adder/subtractor. In subtract mode, CO is an active-Low borrow-out and goes Low when the difference exceeds the bounds.

An unsigned binary “overflow” that is always active-High can be generated by gating the ADD signal and CO as follows.

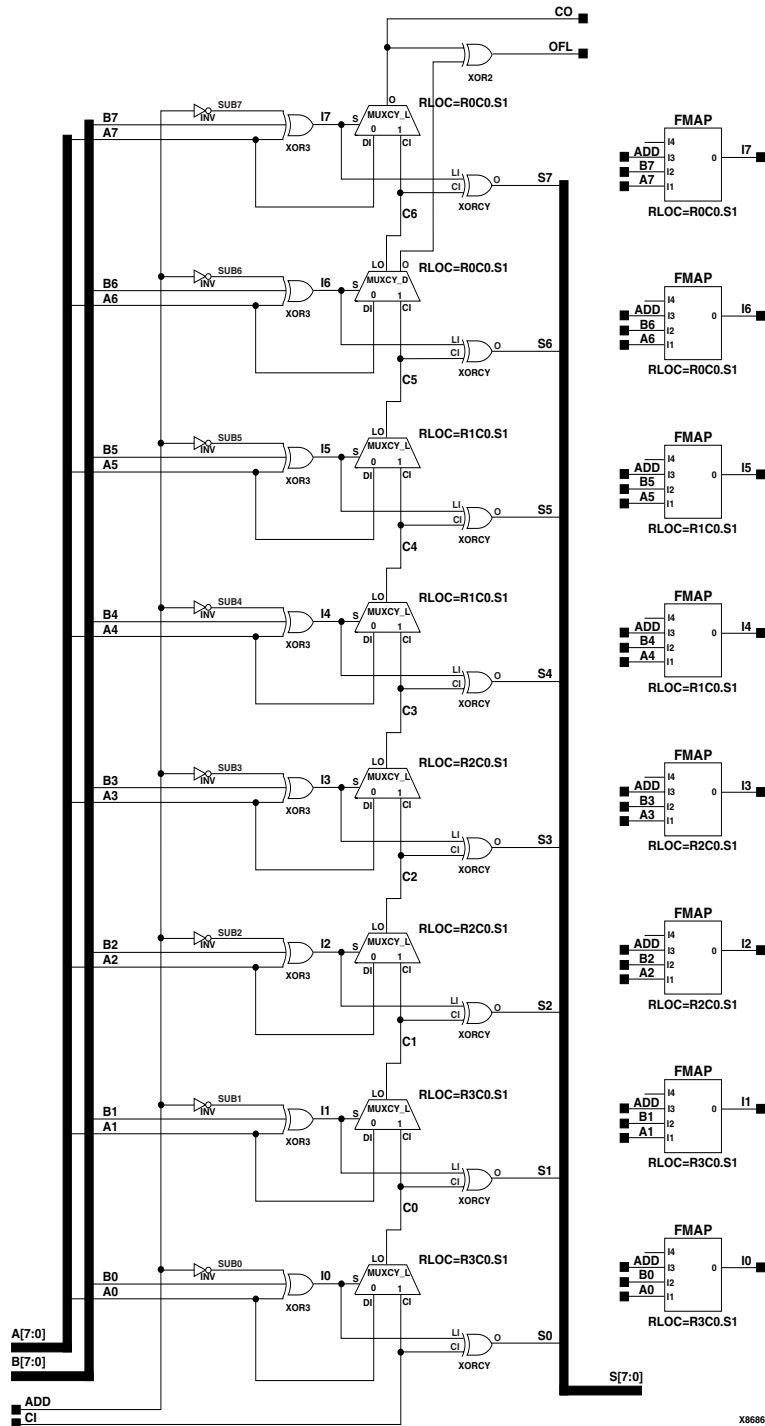
$$\text{unsigned overflow} = \text{CO XOR ADD}$$

OFL is ignored in unsigned binary operation.

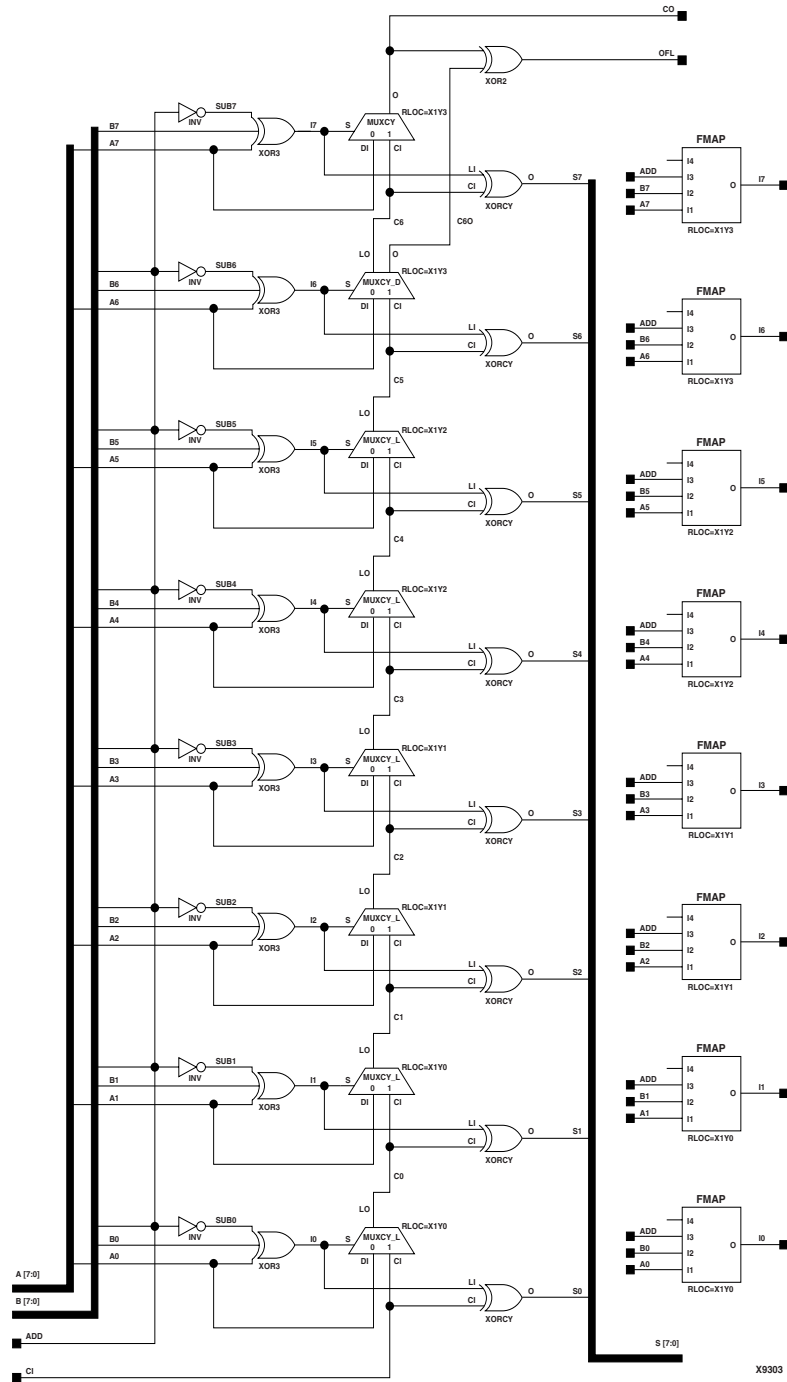
Twos-Complement Operation

For twos-complement operation, ADSU4 can represent numbers between -8 and +7, inclusive; ADSU8 between -128 and +127, inclusive; ADSU16 between -32768 and +32767, inclusive. If an addition or subtraction operation result exceeds this range, the OFL output goes High.

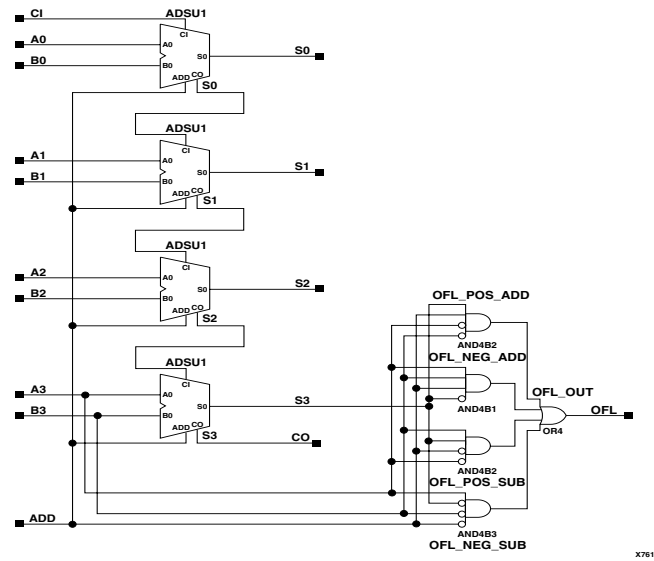
CO is ignored in twos-complement operation.



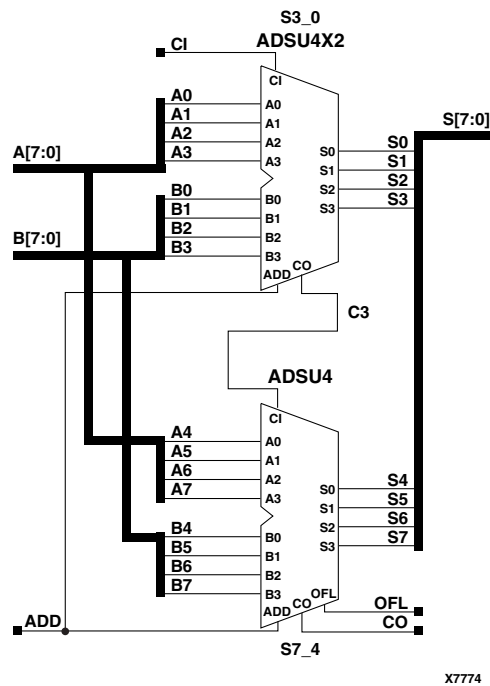
ADSU8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ADSU8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



ADSU4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



ADSU8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

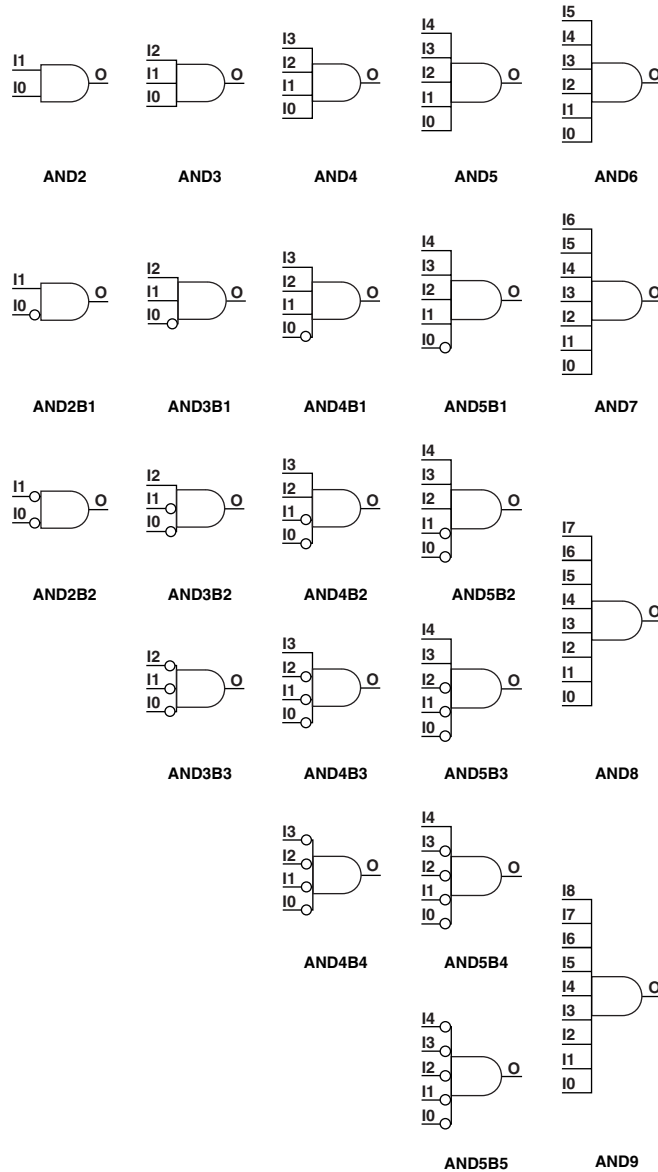
For HDL, these design elements are inferred rather than instantiated.

AND2-9

2- to 9-Input AND Gates with Inverted and Non-Inverted Inputs

Architectures Supported

AND2, AND3, AND4, AND5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
AND2B1, AND2B2, AND4B1, AND4B2, AND4B3, AND4B4, AND5B1, AND5B2, AND5B3, AND5B4, AND5B5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
AND3B1, AND3B2, AND3B3,	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
AND6, AND7, AND8, AND9	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

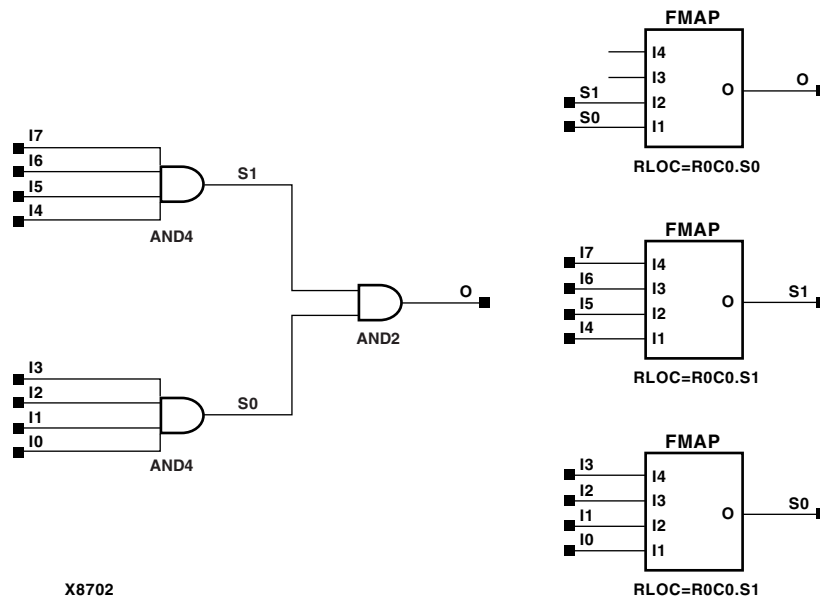


X9461

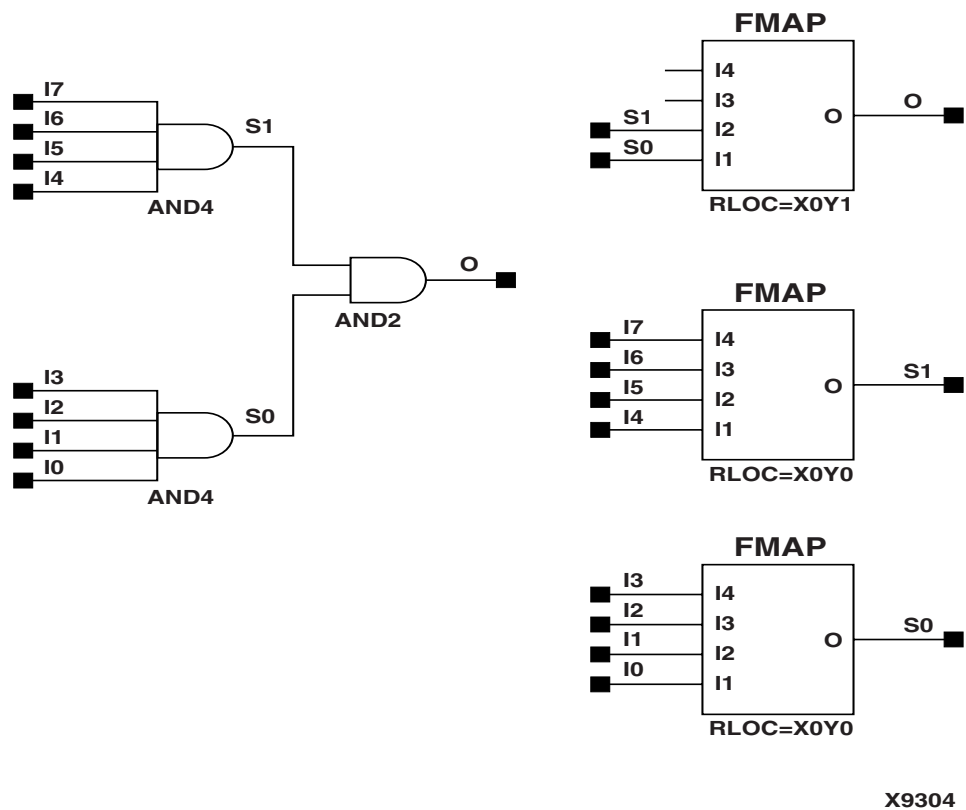
AND Gate Representations

AND functions of up to five inputs are available in any combination of inverting and non-inverting inputs. AND functions of six to nine inputs are available with only non-inverting inputs. To make some or all inputs inverting, use external inverters. Because each input uses a CLB resource in Spartan-II, Spartan-3, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X replace functions with unused inputs with functions having the appropriate number of inputs.

See “AND12, 16” for information on additional AND functions for Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, and Spartan-3.



AND8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



AND8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

If possible, it is recommended that these design elements be inferred rather than instantiated.

VHDL Instantiation Template for AND2, AND2B1, or AND2B2

```
-- Component Declaration for AND2, AND2B1, or AND2B2 should
-- be placed after architecture statement but before begin
-- keyword

component {AND2|AND2B1|AND2B2}
  port (O : out STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I0 : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND2, AND2B1, or
-- AND2B2 should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here

-- Component Instantiation for AND2, AND2B1, or AND2B2
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : {AND2|AND2B1|AND2B2}
  port map (O => user_O,
            I0 => user_I0,
            I1 => user_I1);
```

Verilog Instantiation Template for AND2, AND2B1, or AND2B2

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1));
```

VHDL Instantiation Template for AND3 Through AND3B3

```
-- Component Declaration for AND3 through AND3B3 should
-- be placed after architecture statement but before begin
-- keyword

component {AND3|AND3B1|AND3B2|AND3B3}
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND3 through AND3B3
-- should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for AND3 through AND3B3
-- should be placed in architecture after the begin
-- keyword
```

```
INSTANCE_NAME : {AND3|AND3B1|AND3B2|AND3B3}
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2);
```

Verilog Instantiation Template for AND3 Through AND3B3

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2));
```

VHDL Instantiation Template for AND4 Through AND4B4

```
-- Component Declaration for AND4 through AND4B4 should
-- be placed after architecture statement but before begin
-- keyword
```

```
component {AND4|AND4B1|AND4B2|AND4B3|AND4B4}
    port (O : out STD_ULOGIC;
          I0 : in STD_ULOGIC;
          I1 : in STD_ULOGIC;
          I2 : in STD_ULOGIC;
          I3 : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for AND4 through AND4B4
-- should be placed after architecture declaration
-- but before the begin keyword
```

```
-- Enter attributes here
```

```
-- Component Instantiation for AND4 through AND4B4
-- should be placed in architecture after the begin
-- keyword
```

```
INSTANCE_NAME : AND4_thru_AND4B4
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2,
              I3 => user_I3);
```

Verilog Instantiation Template for AND4 through AND4B4

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2),
                    .I3 (user_I3));
```

VHDL Instantiation Template for AND5 Through AND5B5

```
-- Component Declaration for AND5 through AND5B5 should
-- be placed after architecture statement but before begin
-- keyword

component {AND5|AND5B1|AND5B2|AND5B3|AND5B4|AND5B5}
    port (O : out STD_ULOGIC;
          I0 : in STD_ULOGIC;
          I1 : in STD_ULOGIC;
          I2 : in STD_ULOGIC;
          I3 : in STD_ULOGIC;
          I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for AND5 through AND5B5
-- should be placed after architecture declaration
-- but before the begin keyword

-- Enter attributes here

-- Component Instantiation for AND5 through AND5B5
-- should be placed in architecture after the begin
-- keyword

INSTANCE_NAME : AND5_thru_AND5B5
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2,
              I3 => user_I3,
              I4 => user_I4);
```

Verilog Instantiation Template for AND5 through AND5B5

```
ANDn instance_name (.O (user_O),
                    .I0 (user_I0),
                    .I1 (user_I1),
                    .I2 (user_I2),
                    .I3 (user_I3),
                    .I4 (user_I4));
```

AND12, 16

12- and 16-Input AND Gates with Non-Inverted Inputs

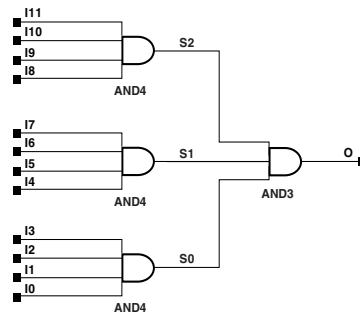
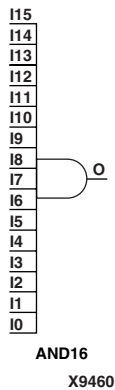
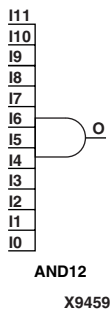
Architectures Supported

AND12, AND16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

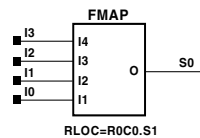
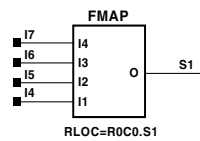
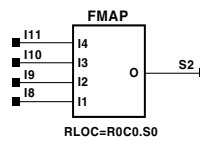
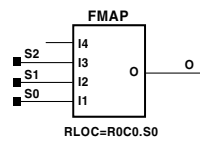
AND12 and AND16 functions are performed in the Configurable Logic Block (CLB) function generator.

The 12- and 16-input AND functions are available only with non-inverting inputs. To invert all of some inputs, use external inverters.

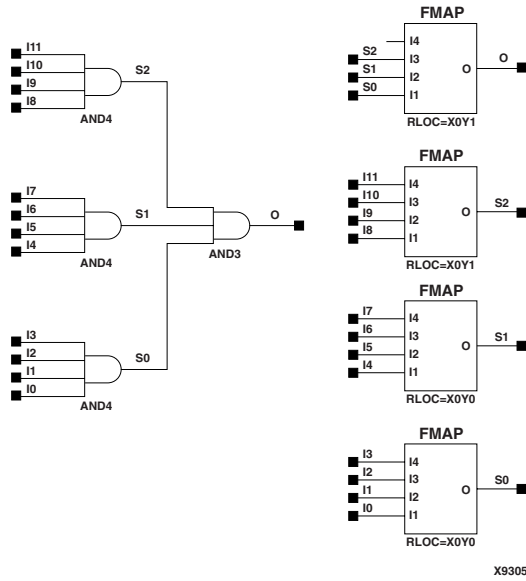
See "AND2-9" for information on more AND functions.



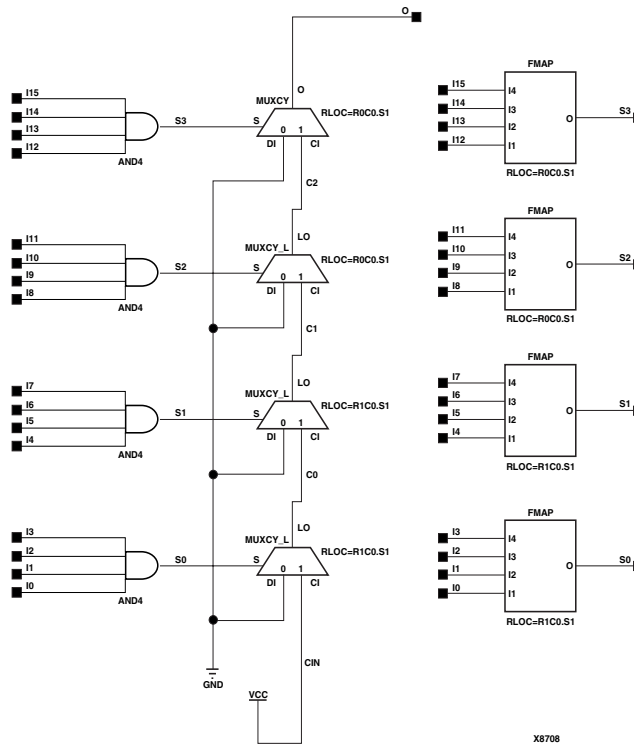
X8705



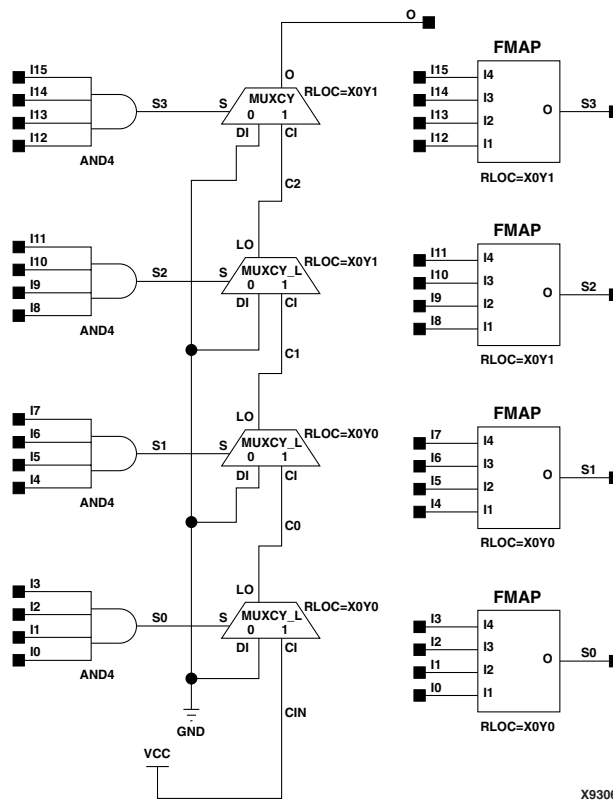
AND12 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



AND12 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



AND16 Implementation Spartan-II, Spartan-II E, Virtex, Virtex-E



AND16 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

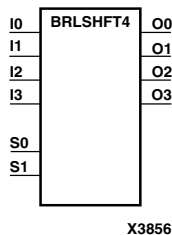
For HDL, it is recommended that these design elements be inferred rather than instantiated.

BRLSHFT4, 8

4-, 8-Bit Barrel Shifters

Architectures Supported

BRLSHFT4, BRLSHFT8	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



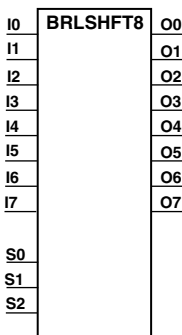
X3856

BRLSHFT4, a 4-bit barrel shifter, can rotate four inputs (I3 – I0) up to four places. The control inputs (S1 and S0) determine the number of positions, from one to four, that the data is rotated. The four outputs (O3 – O0) reflect the shifted data inputs.

BRLSHFT8, an 8-bit barrel shifter, can rotate the eight inputs (I7 – I0) up to eight places. The control inputs (S2 – S0) determine the number of positions, from one to eight, that the data is rotated. The eight outputs (O7 – O0) reflect the shifted data inputs.

BRLSHFT4 Truth Table

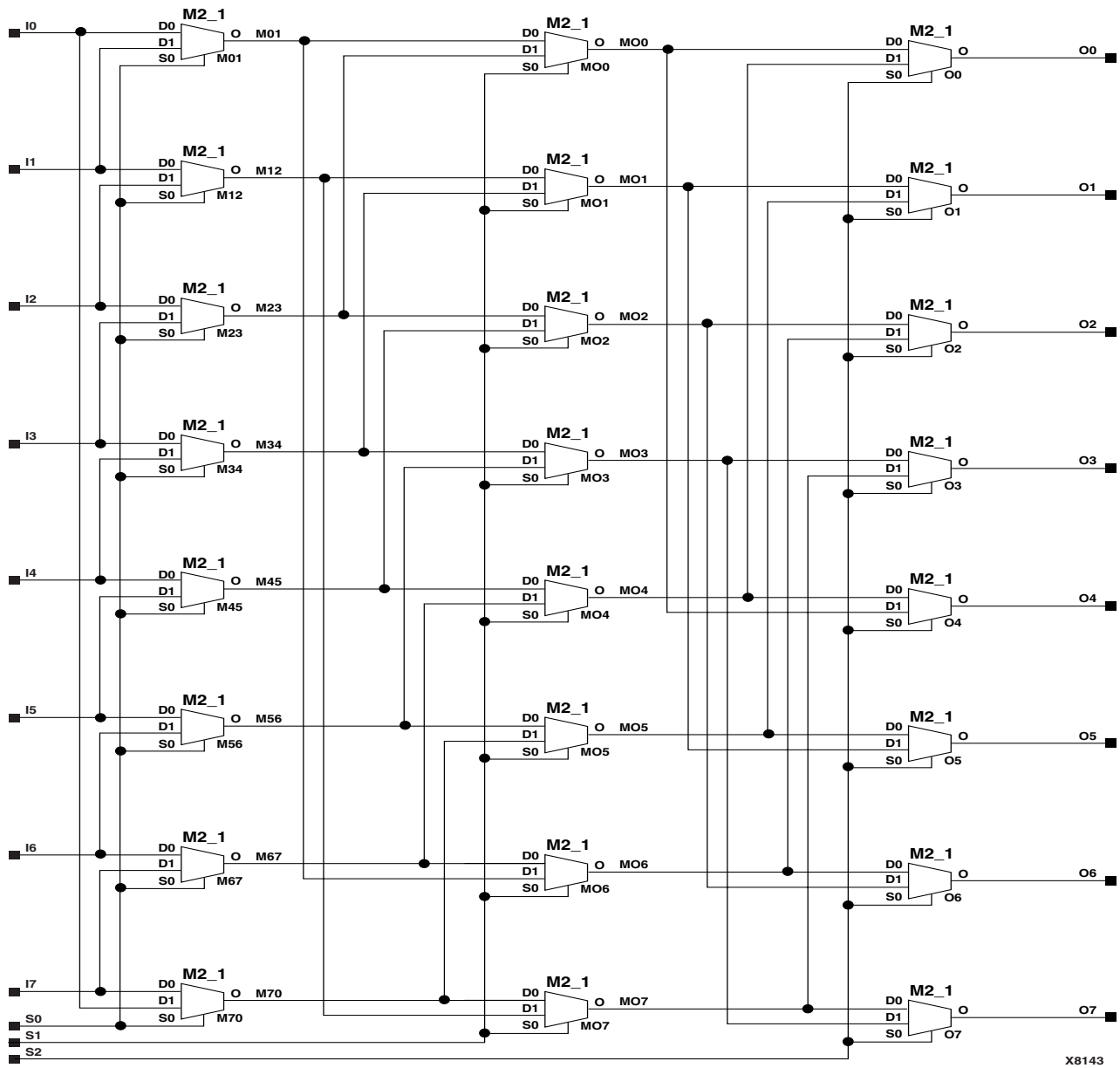
Inputs						Outputs			
S1	S0	I0	I1	I2	I3	O0	O1	O2	O3
0	0	a	b	c	d	a	b	c	d
0	1	a	b	c	d	b	c	d	a
1	0	a	b	c	d	c	d	a	b
1	1	a	b	c	d	d	a	b	c



X3857

BRLSHFT8 Truth Table

Inputs											Outputs							
S2	S1	S0	I0	I1	I2	I3	I4	I5	I6	I7	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
0	0	1	a	b	c	d	e	f	g	h	b	c	d	e	f	g	h	a
0	1	0	a	b	c	d	e	f	g	h	c	d	e	f	g	h	a	b
0	1	1	a	b	c	d	e	f	g	h	d	e	f	g	h	a	b	c
1	0	0	a	b	c	d	e	f	g	h	e	f	g	h	a	b	c	d
1	0	1	a	b	c	d	e	f	g	h	f	g	h	a	b	c	d	e
1	1	0	a	b	c	d	e	f	g	h	g	h	a	b	c	d	e	f
1	1	1	a	b	c	d	e	f	g	h	h	a	b	c	d	e	f	g



X8143

BRLSHFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

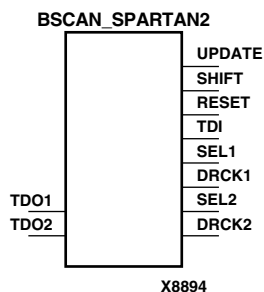
For HDL, these design elements are inferred rather than instantiated.

BSCAN_SPARTAN2

Spartan-II Boundary Scan Logic Control Circuit

Architectures Supported

BSCAN_SPARTAN2	
Spartan-II, Spartan-III	Primitive*
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Supported for Spartan-II, but not for Spartan-III, which is supported by BSCAN_VIRTEX.	



The BSCAN_SPARTAN2 symbol creates internal boundary scan chains in a Spartan-II device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Spartan-II. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_SPARTAN2 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note: For specific information on boundary scan for an architecture, see *The Programmable Logic Data Sheets*.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- Component Declaration for BSCAN_SPARTAN2 should be placed
-- after architecture statement but before begin keyword
-- <Cut code below this line and paste into the architecture body>

-- BSCAN_SPARTAN2: Boundary Scan primitive for connecting internal
--                   logic to. JTAG interface. Spartan-II
-- The current version of the Xilinx HDL Libraries Guide
```

```

BSCAN_SPARTAN2_inst : BSCAN_SPARTAN2
port map (
    DRCK1 => DRCK1,      -- Data register output for USER1 functions
    DRCK2 => DRCK2,      -- Data register output for USER2 functions
    RESET => RESET,      -- Reset output from TAP controller
    SEL1 => SEL1,        -- USER1 active output
    SEL2 => SEL2,        -- USER2 active output
    SHIFT => SHIFT,      -- SHIFT output from TAP controller
    TDI => TDI,          -- TDI output from TAP controller
    UPDATE => UPDATE,    -- UPDATE output from TAP controller
    TDO1 => TDO1,        -- Data input for USER1 function
    TDO2 => TDO2        -- Data input for USER2 function
);

```

Verilog Instantiation Template

```

// BSCAN_SPARTAN2:Boundary Scan primitive for connecting internal logic
//                               JTAG interface. Spartan-II
// The current version of the Xilinx HDL Libraries Guide

BSCAN_SPARTAN2 BSCAN_SPARTAN2_inst (
    .DRCK1(DRCK1),        // Data register output - USER1 functions
    .DRCK2(DRCK2),        // Data register output - USER2 functions
    .RESET(RESET),        // Reset output from TAP controller
    .SEL1(SEL1),          // USER1 active output
    .SEL2(SEL2),          // USER2 active output
    .SHIFT(SHIFT),        // SHIFT output from TAP controller
    .TDI(TDI),            // TDI output from TAP controller
    .UPDATE(UPDATE),      // UPDATE output from TAP controller
    .TDO1(TDO1),          // Data input for USER1 function
    .TDO2(TDO2)           // Data input for USER2 function
);

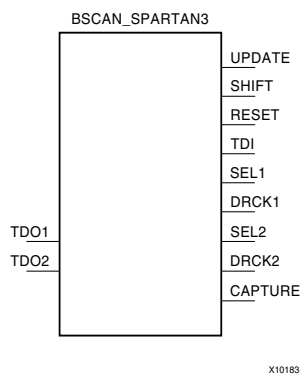
```

BSCAN_SPARTAN3

Spartan-3 Boundary Scan Logic Control Circuit

Architectures Supported

BSCAN_SPARTAN3	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BSCAN_SPARTAN3 provides access to the BSCAN sites on a Spartan-3 device. It is used to create internal boundary scan chains. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Spartan-3. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_SPARTAN3 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, SHIFT, and CAPTURE pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Usage

This design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for BSCAN_SPARTAN3 should be placed
-- after architecture statement but before begin keyword
```

```
component BSCAN_SPARTAN3
  port (CAPTURE : out STD_ULOGIC;
        DRCK1 : out STD_ULOGIC;
        DRCK2 : out STD_ULOGIC;
        RESET : out STD_ULOGIC;
        SEL1 : out STD_ULOGIC;
        SEL2 : out STD_ULOGIC;
        SHIFT : out STD_ULOGIC;
        TDI : out STD_ULOGIC;
        UPDATE : out STD_ULOGIC;
        TDO1 : in STD_ULOGIC;
```

```

        TD02 : in STD_ULONGIC);
end component;

-- Component Attribute specification for BSCAN_SPARTAN3
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BSCAN_SPARTAN3 should be
-- placed in architecture after the begin keyword

BSCAN_SPARTAN3_INSTANCE_NAME : BSCAN_SPARTAN3
    port map (CAPTURE => user_CAPTURE,
              DRCK1 => user_DRCK1,
              DRCK2 => user_DRCK2,
              RESET => user_RESET,
              SEL1 => user_SEL1,
              SEL2 => user_SEL2,
              SHIFT => user_SHIFT,
              TDI => user_TDI,
              UPDATE => user_UPDATE,
              TD01 => user_TD01,
              TD02 => user_TD02);

```

Verilog Instantiation Template

```

// BSCAN_SPARTAN3:Boundary Scan primitive for connecting internal logic
//           JTAG interface. Spartan-II
// The current version of the Xilinx HDL Libraries Guide

BSCAN_SPARTAN3 BSCAN_SPARTAN3_inst (
    .CAPTURE(CAPTURE), // CAPTURE output from TAP controller
    .DRCK1(DRCK1),     // Data register output - USER1 functions
    .DRCK2(DRCK2),     // Data register output - USER2 functions
    .RESET(RESET),     // Reset output from TAP controller
    .SEL1(SEL1),       // USER1 active output
    .SEL2(SEL2),       // USER2 active output
    .SHIFT(SHIFT),     // SHIFT output from TAP controller
    .TDI(TDI),         // TDI output from TAP controller
    .UPDATE(UPDATE),   // UPDATE output from TAP controller
    .TD01(TD01),      // Data input for USER1 function
    .TD02(TD02)       // Data input for USER2 function
);

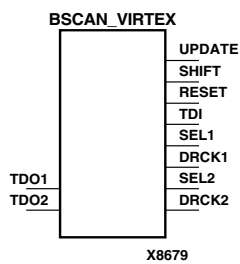
```


BSCAN_VIRTEX

Virtex Boundary Scan Logic Control Circuit

Architectures Supported

BSCAN_VIRTEX	
Spartan-II, Spartan-IIE	Primitive*
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Supported for Spartan-IIE, but not for Spartan-II, which is supported by BSCAN_SPARTAN2.	



The BSCAN_VIRTEX symbol is used to create internal boundary scan chains in a Virtex or Virtex-E device. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Virtex and Virtex-E. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_VIRTEX symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

Note: For Virtex-II, Virtex-II Pro, and Virtex-II Pro X, see “BSCAN_VIRTEX2”.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, and SHIFT pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note: For specific information on boundary scan for an architecture, see *The Programmable Logic Data Sheets*.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- BSCAN_VIRTEX: Boundary Scan primitive for connecting internal logic
-- to
-- JTAG interface. Virtex/E, Spartan-IIE
-- The current version of the Xilinx HDL Libraries Guide

BSCAN_VIRTEX_inst : BSCAN_VIRTEX
```

```

port map (
    DRCK1 => DRCK1,      -- Data register output for USER1 functions
    DRCK2 => DRCK2,      -- Data register output for USER2 functions
    RESET => RESET,      -- Reset output from TAP controller
    SEL1 => SEL1,        -- USER1 active output
    SEL2 => SEL2,        -- USER2 active output
    SHIFT => SHIFT,     -- SHIFT output from TAP controller
    TDI => TDI,          -- TDI output from TAP controller
    UPDATE => UPDATE,    -- UPDATE output from TAP controller
    TDO1 => TDO1,        -- Data input for USER1 function
    TDO2 => TDO2         -- Data input for USER2 function
);

```

Verilog Instantiation Template

```

// BSCAN_VIRTEX: Boundary Scan primitive for connecting internal logic
//                JTAG interface. Virtex/E, Spartan-IIE
// The current version of the Xilinx HDL Libraries Guide

BSCAN_VIRTEX BSCAN_VIRTEX_inst (
    .DRCK1(DRCK1),      // Data register output for USER1 functions
    .DRCK2(DRCK2),      // Data register output for USER2 functions
    .RESET(RESET),      // Reset output from TAP controller
    .SEL1(SEL1),        // USER1 active output
    .SEL2(SEL2),        // USER2 active output
    .SHIFT(SHIFT),     // SHIFT output from TAP controller
    .TDI(TDI),          // TDI output from TAP controller
    .UPDATE(UPDATE),    // UPDATE output from TAP controller
    .TDO1(TDO1),        // Data input for USER1 function
    .TDO2(TDO2)         // Data input for USER2 function
);

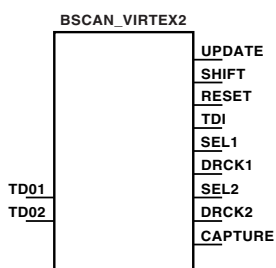
```

BSCAN_VIRTEX2

Virtex-II Boundary Scan Logic Control Circuit

Architectures Supported

BSCAN_VIRTEX2	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9402

BSCAN_VIRTEX2 provides access to the BSCAN sites on a Virtex-II, Virtex-II Pro, or Virtex-II Pro X device. It is used to create internal boundary scan chains. The 4-pin JTAG interface (TDI, TDO, TCK, and TMS) are dedicated pins in Virtex-II, Virtex-II Pro, and Virtex-II Pro X. To use normal JTAG for boundary scan purposes, just hook up the JTAG pins to the port and go. The pins on the BSCAN_VIRTEX2 symbol do not need to be connected, unless those special functions are needed to drive an internal scan chain.

Note: For Virtex and Virtex-E, see “BSCAN_VIRTEX”.

A signal on the TDO1 input is passed to the external TDO output when the USER1 instruction is executed; the SEL1 output goes High to indicate that the USER1 instruction is active. The DRCK1 output provides USER1 access to the data register clock (generated by the TAP controller). The TDO2 and SEL2 pins perform a similar function for the USER2 instruction and the DRCK2 output provides USER2 access to the data register clock (generated by the TAP controller). The RESET, UPDATE, SHIFT, and CAPTURE pins represent the decoding of the corresponding state of the boundary scan internal state machine. The TDI pin provides access to the TDI signal of the JTAG port in order to shift data into an internal scan chain.

Note: For specific information on boundary scan for an architecture, see *The Programmable Logic Data Sheets*.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- BSCAN_VIRTEX2: Boundary Scan primitive for connecting internal
--                   logic to. JTAG interface. Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

BSCAN_VIRTEX2_inst : BSCAN_VIRTEX2
port map (
    CAPTURE => CAPTURE, -- CAPTURE output from TAP controller
    DRCK1 => DRCK1,     -- Data register output for USER1 functions
```

```

DRCK2 => DRCK2,      -- Data register output for USER2 functions
RESET => RESET,     -- Reset output from TAP controller
SEL1 => SEL1,       -- USER1 active output
SEL2 => SEL2,       -- USER2 active output
SHIFT => SHIFT,     -- SHIFT output from TAP controller
TDI => TDI,         -- TDI output from TAP controller
UPDATE => UPDATE,   -- UPDATE output from TAP controller
TDO1 => TDO1,       -- Data input for USER1 function
TDO2 => TDO2        -- Data input for USER2 function
);

```

Verilog Instantiation Template

```

// BSCAN_VIRTEX2: Boundary Scan primitive for connecting internal logic
//                JTAG interface. Virtex-II/II-Pro
// The current version of the Xilinx HDL Libraries Guide

BSCAN_VIRTEX2 BSCAN_VIRTEX2_inst (
    .CAPTURE(CAPTURE), // CAPTURE output from TAP controller
    .DRCK1(DRCK1),     // Data register output - USER1 functions
    .DRCK2(DRCK2),     // Data register output - USER2 functions
    .RESET(RESET),     // Reset output from TAP controller
    .SEL1(SEL1),       // USER1 active output
    .SEL2(SEL2),       // USER2 active output
    .SHIFT(SHIFT),     // SHIFT output from TAP controller
    .TDI(TDI),         // TDI output from TAP controller
    .UPDATE(UPDATE),   // UPDATE output from TAP controller
    .TDO1(TDO1),       // Data input for USER1 function
    .TDO2(TDO2)        // Data input for USER2 function
);

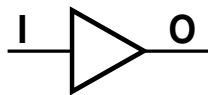
```

BUF

General-Purpose Buffer

Architectures Supported

BUF	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



X9444

BUF is a general purpose, non-inverting buffer.

In Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, BUF is usually not necessary and is removed by the partitioning software (MAP).

In XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, BUF is usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF symbol.

Usage

This design is supported in schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUF should be placed
-- after architecture statement but before begin keyword

component BUF
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUF should be placed
-- in architecture after the begin keyword

BUF_INSTANCE_NAME : BUF
  port map (O => user_O,
           I => user_I);
```

Verilog Instantiation Template

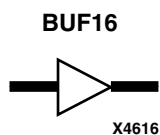
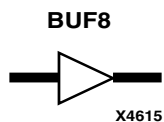
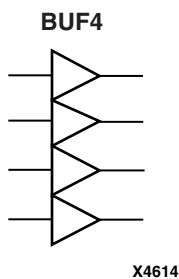
```
BUF instance_name (.O (user_0),  
                  .I (user_I));
```

BUF4, 8, 16

General-Purpose Buffers

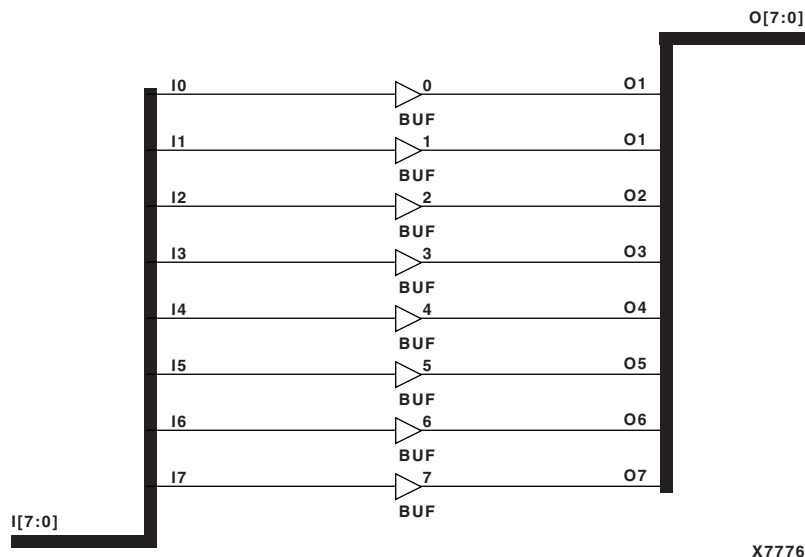
Architectures Supported

BUF4, BUF8, BUF16	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



BUF4, 8, 16 are general purpose, non-inverting buffers.

In XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, BUF4, BUF8, and BUF16 are usually removed, unless you inhibit optimization by applying the OPT=OFF attribute to the BUF4, BUF8, or BUF16 symbol or by using the LOGIC_OPT=OFF global attribute.



BUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

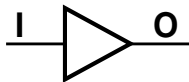
These design elements are schematic only.

BUFCF

Fast Connect Buffer

Architectures Supported

BUFCF	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9444

BUFCF is a single fast connect buffer used to connect the outputs of the LUTs and some dedicated logic directly to the input of another LUT. Using this buffer implies CLB packing. No more than four LUTs may be connected together as a group.

Usage

This design element is supported for schematics and instantiation but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFCF should be placed
-- after architecture statement but before begin keyword
```

```
component BUFCF
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BUFCF
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes here
```

```
-- Component Instantiation for BUFCF should be placed
-- in architecture after the begin keyword
```

```
BUFCF_INSTANCE_NAME : BUFCF
  port map (O => user_O,
           I => user_I);
```

Verilog Instantiation Template

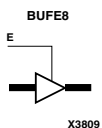
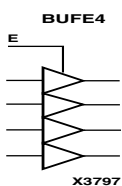
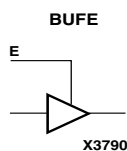
```
BUFCF instance_name (.O (user_O),
                    .I (user_I));
```


BUFE, 4, 8, 16

Internal 3-State Buffers with Active High Enable

Architectures Supported

BUFE	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	No
CoolRunner-II	No
BUFE4, BUFE8, BUFE16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Macro*
CoolRunner XPLA3	No
CoolRunner-II	No
*Not supported for XC9500XL and XC9500XV devices	

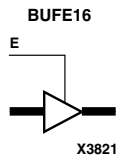


BUFE, BUFE4, BUFE8, and BUFE16 are single or multiple 3-state buffers with inputs I₃ – I₀, I₇ – I₀, and I₁₅ – I₀, respectively; outputs O₃ – O₀, O₇ – O₀, and O₁₅ – O₀, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is high impedance (Z state or Off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

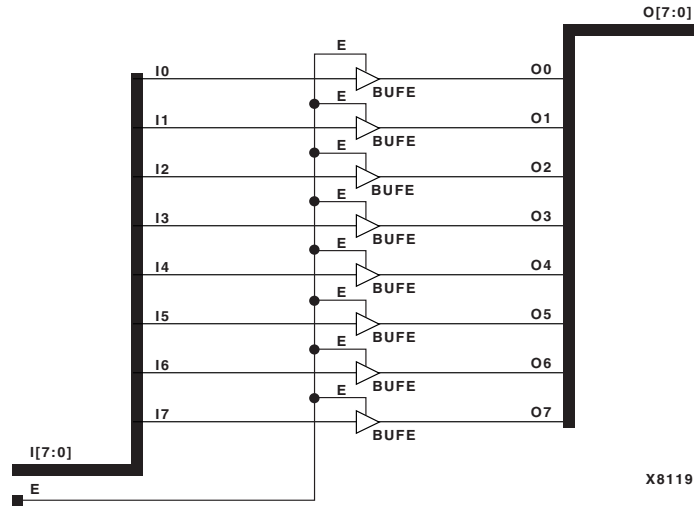
The outputs of separate BUFE symbols can be tied together to form a bus or a multiplexer. Make sure that only one E is High at any one time. If none of the E inputs is active-High, a “weak-keeper” circuit (Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X) keeps the output bus from floating but does not guarantee that the bus remains at the last value driven onto it.

For XC9500 devices, BUFE output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, BUFE elements need a PULLUP element connected to their output. NGDBuild inserts a PULLUP element if one is not connected.



Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0



BUFE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

These design elements are supported for schematic, inference, and instantiation.

VHDL Instantiation Template

```
-- Component Declaration for BUFE should be placed
-- after architecture statement but before begin keyword
```

```
component BUFE
  port (O : out STD_ULOGIC;
        E : in STD_ULOGIC;
        I : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BUFE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes here
```

```
-- Component Instantiation for BUFE should be placed
```

```
-- in architecture after the begin keyword

BUFE_INSTANCE_NAME : BUFE
    port map (O => user_O,
              E => user_E,
              I => user_I);
```

Verilog Instantiation Template

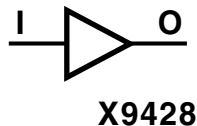
```
BUFE instance_name (.O (user_O),
                    .E (user_E),
                    .I (user_I));
```


BUFG

Global Clock Buffer

Architectures Supported

BUFG	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



BUFG, an architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device. The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. To use a specific type of buffer, instantiate it manually.

To use a BUFG in a schematic, connect the input of the BUFG symbol to the clock source. Depending on the target PLD family, the clock source can be an external PAD symbol, an IBUF symbol, or internal logic. For a negative-edge clock input, insert an INV (inverter) symbol between the BUFG output and the clock input. The inversion is implemented at the Configurable Logic Block (CLB) or Input Output Block (IOB) clock pin.

XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II

Consult the device data sheet for the number of available global pins. For these architectures BUFG is always implemented using an IOB. Connect the input of BUFG to an IPAD or an IOPAD that represents an external signal source. Each BUFG can drive any number of register clocks in a design. The output of a BUFG may also be used as an ordinary input signal to other logic elsewhere in the design.

Virtex, Virtex-E, Spartan-II, Spartan-IIE

In Virtex, Virtex-E, Spartan-II, and Spartan-IIE, the BUFG cannot be driven directly from a pad. It can be driven from an IBUG to indicate to use the dedicated pin (GCLKIOB pin) or from an internal driver to create an internal clock. BUFG can also be driven with an IBUF to represent an externally driven clock that does not use the dedicated pin.

Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, clock buffers are multiplexed clock buffers. In Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, a BUFG is implemented using a BUFGMUX with the S_B input tied high, basically meaning the S input is tied low. I1 is unused. I0 is used.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized. The BUFGP contains both a BUFG and an IBUFG.

VHDL Instantiation Template

```
-- Component Declaration for BUFG should be placed
-- after architecture statement but before begin keyword

component BUFG
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFG
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFG should be placed
-- in architecture after the begin keyword

BUFG_INSTANCE_NAME : BUFG
  port map (O => user_O,
           I => user_I);
```

Verilog Instantiation Template

```
BUFG instance_name (.O (user_O),
                   .I (user_I));
```

Commonly Used Constraints

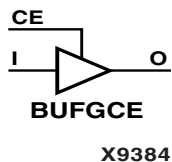
LOC

BUFGCE

Global Clock MUX Buffer with Clock Enable and Output State 0

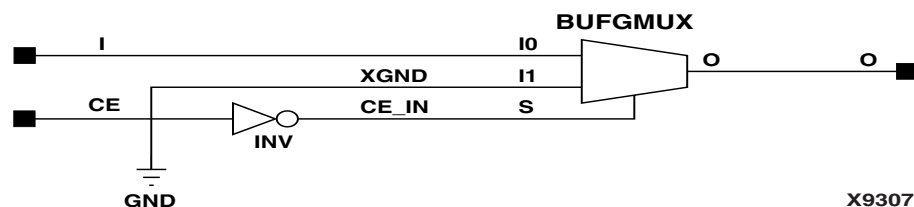
Architectures Supported

BUFGCE	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BUFGCE is a global clock buffer with a single gated input. Its O output is "0" when clock enable (CE) is Low (inactive). When clock enable (CE) is High, the I input is transferred to the O output.

Inputs		Outputs
I	CE	O
X	0	0
I	1	I



BUFGCE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGE should be placed
-- after architecture statement but before begin keyword

component BUFGE
  port (O : out STD_ULOGIC;
        CE : in STD_ULOGIC;
        I : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BUFGCE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFGCE should be placed
-- in architecture after the begin keyword

BUFGCE_INSTANCE_NAME : BUFGCE
    port map (O => user_O,
              CE => user_CE,
              I => user_I);
```

Verilog Instantiation Template

```
BUFGCE instance_name (.O (user_O),
                      .CE (user_CE),
                      .I (user_I));
```

Commonly Used Constraints

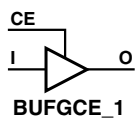
LOC

BUFGCE_1

Global Clock MUX Buffer with Clock Enable and Output State 1

Architectures Supported

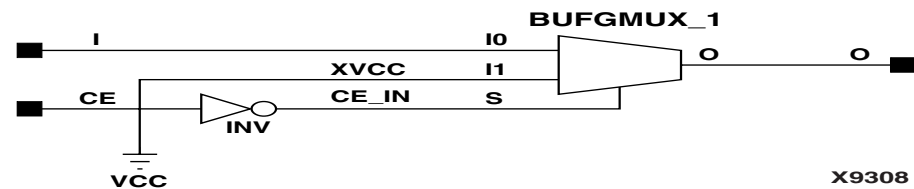
BUFGCE_1	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9385

BUFGCE_1 is a multiplexed global clock buffer with a single gated input. Its O output is High (1) when clock enable (CE) is Low (inactive). When clock enable (CE) is High, the I input is transferred to the O output.

Inputs		Outputs
I	CE	O
X	0	1
I	1	I



X9308

BUFGCE_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGE_1 should be placed
-- after architecture statement but before begin keyword

component BUFGE_1
  port (O : out STD_ULOGIC;
        CE : in STD_ULOGIC;
        I: in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGE_1
```

```
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUGCE_1 should be placed
-- in architecture after the begin keyword

BUGCE_1_INSTANCE_NAME : BUGCE_1
    port map (O => user_O,
              CE => user_CE,
              I => user_I);
```

Verilog Instantiation Template

```
BUGCE_1 instance_name (.O (user_O),
                       .CE (user_CE),
                       .I (user_I));
```

Commonly Used Constraints

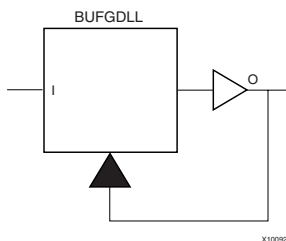
LOC

BUFGDLL

Clock Delay Locked Loop Buffer

Architectures Supported

BUFGDLL	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BUFGDLL is a special purpose clock delay locked loop buffer for clock skew management. It is provided as a user convenience for the most frequently used configuration of elements for clock skew management. Internally, it consists of an IBUFG driving the CLKIN pin of a CLKDLL followed by a BUFG that is driven by the CLK0 pin of the CLKDLL. Because BUFGDLL already contains an input buffer (IBUFG), it can only be driven by a top-level port (IPAD).

Any DUTY_CYCLE_CORRECTION attribute on a BUFGDLL applies to the underlying CLKDLL symbol.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGDLL should be placed
-- after architecture statement but before begin keyword

component BUFGDLL
    port (O : out STD_ULOGIC;
          I : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGDLL
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for BUFGDLL should be placed  
-- in architecture after the begin keyword
```

```
BUFGDLL_INSTANCE_NAME : BUFGDLL  
    port map (O => user_O,  
             I => user_I);
```

Verilog Instantiation Template

```
BUFGDLL instance_name (.O (user_O),  
                      .I (user_I));
```

Commonly Used Constraints

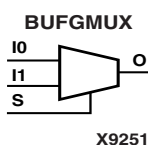
STARTUP_WAIT

BUFGMUX

Global Clock MUX Buffer with Output State 0

Architectures Supported

BUFGMUX	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BUFGMUX is a multiplexed global clock buffer that can select between two input clocks I0 and I1. When the select input (S) is Low, the signal on I0 is selected for output (O). When the select input (S) is High, the signal on I1 is selected for output.

BUFGMUX and BUFGMUX_1 are distinguished by which state the output assumes when it switches between clocks in response to a change in its select input. BUFGMUX assumes output state 0 and BUFGMUX_1 assumes output state 1.

Using a BUFGMUX element in your design may cause inaccurate simulation if all the following conditions occur: both clock inputs (I0 and I1) are used, GSR is activated during simulation (after simulation time `0`), and the secondary clock input (I1) is selected before or while GSR is active. In this case, the primary clock input (I0) is incorrectly selected. This occurs because there is a cross-coupled register pair that ensures the BUFGMUX output does not inadvertently generate a clock edge. When GSR is asserted, these registers initialize to the default state of I0. To select the secondary clock, you must send a clock pulse to both the primary and secondary clock inputs while GSR is inactive.

Note: BUFGMUX guarantees that when S is toggled, the state of the output will remain in the inactive state until the next active clock edge (either I0 or I1) occurs.

Inputs			Outputs
I0	I1	S	O
I0	X	0	I0
X	I1	1	I1
X	X	↑	0
X	X	↓	0

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGMUX should be placed
-- after architecture statement but before begin keyword

component BUFGMUX
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGMUX
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFGMUX should be placed
-- in architecture after the begin keyword

BUFGMUX_INSTANCE_NAME : BUFGMUX
  port map (O => user_O,
            I0 => user_I0,
            I1 => user_I1,
            S => user_S);
```

Verilog Instantiation Template

```
BUFGMUX instance_name (.O (user_O),
                       .I0 (user_I0),
                       .I1 (user_I1),
                       .S (user_S));
```

Commonly Used Constraints

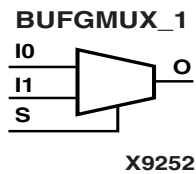
LOC

BUFGMUX_1

Global Clock MUX Buffer with Output State 1

Architectures Supported

BUFGMUX_1	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BUFGMUX_1 is a multiplexed global clock buffer that can select between two input clocks I0 and I1. When the select input (S) is Low, the signal on I0 is selected for output (O). When the select input (S) is High, the signal on I1 is selected for output.

BUFGMUX and BUFGMUX_1 are distinguished by which state the output assumes when it switches between clocks in response to a change in its select input. BUFGMUX assumes output state 0 and BUFGMUX_1 assumes output state 1.

Using a BUFGMUX_1 element in your design may cause inaccurate simulation if all the following conditions occur: both clock inputs (I0 and I1) are used, GSR is activated during simulation (after simulation time `0`), and the secondary clock input (I1) is selected before or while GSR is active. In this case, the primary clock input (I0) is incorrectly selected. This occurs because there is a cross-coupled register pair that ensures the BUFGMUX_1 output does not inadvertently generate a clock edge. When GSR is asserted, these registers initialize to the default state of I0. To select the secondary clock, you must send a clock pulse to both the primary and secondary clock inputs while GSR is inactive.

Inputs			Outputs
I0	I1	S	O
I0	X	0	I0
X	I1	1	I1
X	X	↑	1
X	X	↓	1

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGMUX_1 should be placed
-- after architecture statement but before begin keyword

component BUFGMUX_1
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGMUX_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFGMUX should be placed
-- in architecture after the begin keyword

BUFGMUX_1_INSTANCE_NAME : BUFGMUX_1
  port map (O => user_O,
           I0 => user_I0,
           I1 => user_I1,
           S => user_S);
```

Verilog Instantiation Template

```
BUFGMUX_1 instance_name (.O (user_O),
                        .I0 (user_I0),
                        .I1 (user_I1),
                        .S (user_S));
```

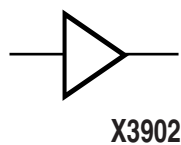
Commonly Used Constraints

LOC

BUFGP

Primary Global Buffer for Driving Clocks or Longlines (Four per PLD Device)

BUFGP	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



BUFGP, a primary global buffer, is used to distribute high fan-out clock or control signals throughout PLD devices.

In Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, BUFGP is equivalent to an IBUF driving a BUFG.

In XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, BUFGP is treated like BUFG.

A BUFGP provides direct access to Configurable Logic Block (CLB) and Input Output Block (IOB) clock pins and limited access to other CLB inputs. The input to a BUFGP comes only from a dedicated IOB.

Because of its structure, a BUFGP can always access a clock pin directly. However, it can access only one of the F3, G1, C3, or C1 pins, depending on the corner in which the BUFGP is placed. When the required pin cannot be accessed directly from the vertical line, PAR feeds the signal through another CLB and uses general purpose routing to access the load pin.

To use a BUFGP in a schematic, connect the input of the BUFGP element directly to the PAD symbol. Do not use any IBUFs, because the signal comes directly from a dedicated IOB. The output of the BUFGP is then used throughout the schematic. For a negative-edge clock, insert an INV (inverter) element between the output of the BUFGP and the clock input. This inversion is performed inside each CLB or IOB.

A Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, or Spartan-3 BUFGP must be sourced by an external signal.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized.

VHDL Instantiation Template

```
-- Component Declaration for BUFGP should be placed
-- after architecture statement but before begin keyword

component BUFGP
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGP
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for BUFGP should be placed
-- in architecture after the begin keyword

BUFGP_INSTANCE_NAME : BUFGP

  port map (O => user_O,
            I => user_I);
```

Verilog Instantiation Template

```
BUFGP instance_name (.O (user_O),
                    .I (user_I));
```

Commonly Used Constraints

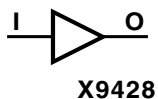
HBLKNM

BUFGSR

Global Set/Reset Input Buffer

Architectures Supported

BUFGSR	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



BUFGSR distributes global set/reset signals throughout selected flip-flops of an XC9500/XV/XL, CoolRunner XPLA3, or CoolRunner-II device. Global Set/Reset (GSR) control pins are available on these CPLD devices. Consult device data sheets for availability.

BUFGSR always acts as an input buffer. To use it in a schematic, connect the input of the BUFGSR symbol to an IPAD or an IOPAD representing the GSR signal source. GSR signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGSR.

For global set/reset control, the output of BUFGSR normally connects to the CLR or PRE input of a flip-flop symbol, like FDCEP, or any registered symbol with asynchronous clear or preset. The global set/reset control signal may pass through an inverter to perform an active-low set/reset. The output of BUFGSR may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGSR can control any number of flip-flops in a design.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGSR should be placed
-- after architecture statement but before begin keyword
```

```
component BUFGSR
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for BUFGSR
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes here
```

```
-- Component Instantiation for BUFGSR should be placed  
-- in architecture after the begin keyword
```

```
BUFGSR_INSTANCE_NAME : BUFGSR  
    port map (O => user_O,  
             I => user_I);
```

Verilog Instantiation Template

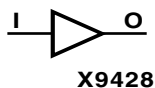
```
BUFGSR instance_name (.O (user_O),  
                     .I (user_I));
```

BUFGTS

Global 3-State Input Buffer

Architectures Supported

BUFGTS	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



BUFGTS distributes global output-enable signals throughout the output pad drivers of an XC9500/XV/XL, CoolRunner XPLA3, or CoolRunner-II device. Global Three-State (GTS) control pins are available on these CPLD devices. Consult device data sheets for availability.

BUFGTS always acts as an input buffer. To use it in a schematic, connect the input of the BUFGTS symbol to an IPAD or an IOPAD representing the GTS signal source. GTS signals generated on-chip must be passed through an OBUF-type buffer before they are connected to BUFGTS.

For global 3-state control, the output of BUFGTS normally connects to the E input of a 3-state output buffer symbol, OBUFE. The global 3-state control signal may pass through an inverter or control an OBUFT symbol to perform an active-low output-enable. The same 3-state control signal may even be used both inverted and non-inverted to enable alternate groups of device outputs. The output of BUFGTS may also be used as an ordinary input signal to other logic elsewhere in the design. Each BUFGTS can control any number of output buffers in a design.

Usage

This design element is supported for schematics and instantiations but not for inference.

VHDL Instantiation Template

```
-- Component Declaration for BUFGTS should be placed
-- after architecture statement but before begin keyword

component BUFGTS
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFGTS
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes here

-- Component Instantiation for BUFGTS should be placed
-- in architecture after the begin keyword

BUFGTS_INSTANCE_NAME : BUFGTS
    port map (O => user_O,
             I => user_I);
```

Verilog Instantiation Template

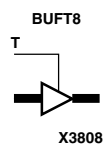
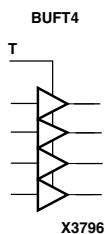
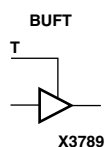
```
BUFGTS instance_name (.O (user_O),
                     .I (user_I));
```


BUFT, 4, 8, 16

Internal 3-State Buffers with Active-Low Enable

Architectures Supported

BUFT	
Spartan-II, Spartan-III	Primitive
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive*
CoolRunner XPLA3	No
CoolRunner-II	No
BUFT4, BUFT8, BUFT16	
Spartan-II, Spartan-III	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	No
CoolRunner-II	No

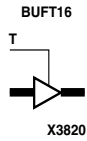


BUFT, BUFT4, BUFT8, and BUFT16 are single or multiple 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15 – I0, respectively; outputs O, O3 – O0, O7 – O0, and O15 – O0, respectively; and active-Low output enable (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (Z state or off). The outputs of the buffers are connected to horizontal longlines in FPGA architectures.

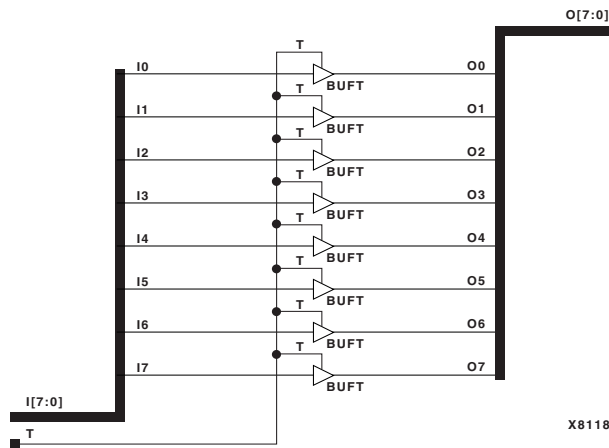
The outputs of separate BUFT symbols can be tied together to form a bus or a multiplexer. Make sure that only one T is Low at one time.

For XC9500 devices, BUFT output nets assume the High logic level when all connected BUFE/BUFT buffers are disabled.

For Spartan-II, Spartan-III, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, when all BUFTs on a net are disabled, the net is High. For correct simulation of this effect, a PULLUP element must be connected to the net. NGDBuild inserts a PULLUP element if one is not connected so that back-annotation simulation reflects the true state of the device.



Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0



BUFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-II E, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

These design elements are supported for schematics, instantiations, or inferences.

VHDL Instantiation Template

```
-- Component Declaration for BUFT should be placed
-- after architecture statement but before begin keyword

component BUFT
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC;
        T : in STD_ULOGIC);
end component;

-- Component Attribute specification for BUFT
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
```

```
-- Component Instantiation for BUFT should be placed  
-- in architecture after the begin keyword
```

```
BUFT_INSTANCE_NAME : BUFT  
    port map (O => user_O,  
             I => user_I,  
             T => user_T);
```

Verilog Instantiation Template

```
BUFT instance_name (.O (user_O),  
                  .I (user_I),  
                  .T (user_T));
```

Commonly Used Constraints

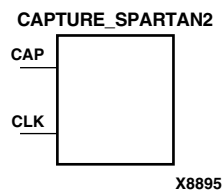
BLKNM, HBLKNM, HU_SET, LOC, U_SET

CAPTURE_SPARTAN2

Spartan-II Register State Capture for Bitstream Readback

Architectures Supported

CAPTURE_SPARTAN2	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CAPTURE_SPARTAN2 provides user control over when to capture register (flip-flop and latch) information for readback. Spartan-II and Spartan-IIE devices provide the readback function through dedicated configuration port instructions. The CAPTURE_SPARTAN2 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note: Spartan-II and Spartan-IIE devices only allow for capturing register (flip-flop and latch) states. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_SPARTAN2. See the *Constraints Guide* for information on the ONESHOT attribute.

Note: For details on the Spartan-II and Spartan-IIE readback functions, see *The Programmable Logic Data Sheets*.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- CAPTURE_SPARTAN2: Register State Capture for Bitstream Readback
--                               Spartan-II/IIE
-- The current version of the Xilinx HDL Libraries Guide

CAPTURE_SPARTAN2_inst : CAPTURE_SPARTAN2
port map (
    CAP => CAP,    -- Capture input
    CLK => CLK     -- Clock input
);
```

Verilog Instantiation Template

```
// CAPTURE_SPARTAN2: Register State Capture for Bitstream Readback
//                               Spartan-II/IIE
// The current version of the Xilinx HDL Libraries Guide

CAPTURE_SPARTAN2 CAPTURE_SPARTAN2_inst (
    .CAP(CAP),    // Capture input
    .CLK(CLK)    // Clock input
);
```

Commonly Used Constraints

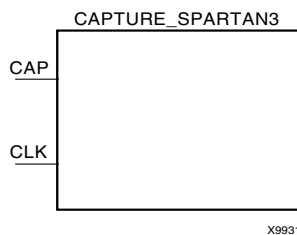
ONESHOT

CAPTURE_SPARTAN3

Spartan-3 Register State Capture for Bitstream Readback

Architectures Supported

CAPTURE_SPARTAN3	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CAPTURE_SPARTAN3 provides user control over when to capture register (flip-flop and latch) information for readback. Spartan-3 devices provide the readback function through dedicated configuration port instructions.

The CAPTURE_SPARTAN3 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Spartan-3 allows for capturing register (flip-flop and latch) states only. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured. An asserted high CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition.

By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_SPARTAN3. See the *Constraints Guide* for information on the ONESHOT attribute.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- CAPTURE_SPARTAN3: Register State Capture for Bitstream Readback
--                               Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

CAPTURE_SPARTAN3_inst : CAPTURE_SPARTAN3
port map (
    CAP => CAP,    -- Capture input
    CLK => CLK     -- Clock input
);
```

Verilog Instantiation Template

```
// CAPTURE_SPARTAN3: Register State Capture for Bitstream Readback
//                               Spartan-3
```

```
// The current version of the Xilinx HDL Libraries Guide

CAPTURE_SPARTAN3 CAPTURE_SPARTAN3_inst (
    .CAP(CAP),    // Capture input
    .CLK(CLK)    // Clock input
);
```

Commonly Used Constraints

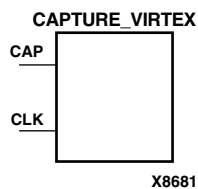
ONESHOT

CAPTURE_VIRTEX

Virtex Register State Capture for Bitstream Readback

Architectures Supported

CAPTURE_VIRTEX	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CAPTURE_VIRTEX provides user control over when to capture register (flip-flop and latch) information for readback. Virtex and Virtex-E devices provide the readback function through dedicated configuration port instructions.

The CAPTURE_VIRTEX symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Note: Virtex and Virtex-E allow for capturing register (flip-flop and latch) states only. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted High CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_VIRTEX. See the *Constraints Guide* for information on the ONESHOT attribute.

For details on the Virtex and Virtex-E readback functions, see the Virtex datasheets on the Xilinx web site, <http://support.xilinx.com>.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- CAPTURE_VIRTEX: Register State Capture for Bitstream Readback
--                               Virtex/E
-- The current version of the Xilinx HDL Libraries Guide

CAPTURE_VIRTEX_inst : CAPTURE_VIRTEX
port map (
    CAP => CAP,    -- Capture input
    CLK => CLK,    -- Clock input
);
```

Verilog Instantiation Template

```
// CAPTURE_VIRTEX: Register State Capture for Bitstream Readback
//                               Virtex/E
// The current version of the Xilinx HDL Libraries Guide

CAPTURE_VIRTEX CAPTURE_VIRTEX_inst (
    .CAP(CAP),    // Capture input
    .CLK(CLK)    // Clock input
);
```

Commonly Used Constraints

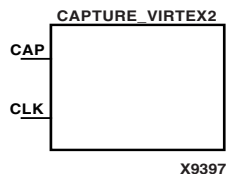
ONESHOT

CAPTURE_VIRTEX2

Virtex-II Register State Capture for Bitstream Readback

Architectures Supported

CAPTURE_VIRTEX2	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CAPTURE_VIRTEX2 provides user control over when to capture register (flip-flop and latch) information for readback. Virtex-II, Virtex-II Pro, and Virtex-II Pro X devices provide the readback function through dedicated configuration port instructions.

The CAPTURE_VIRTEX2 symbol is optional. Without it readback is still performed, but the asynchronous capture function it provides for register states is not available.

Virtex-II, Virtex-II Pro, and Virtex-II Pro X allow for capturing register (flip-flop and latch) states only. Although LUT RAM, SRL, and block RAM states are read back, they cannot be captured.

An asserted high CAP signal indicates that the registers in the device are to be captured at the next Low-to-High clock transition. By default, data is captured after every trigger (transition on CLK while CAP is asserted). To limit the readback operation to a single data capture, add the ONESHOT attribute to CAPTURE_VIRTEX2. See the *Constraints Guide* for information on the ONESHOT attribute.

The GRDBK (internal capture signal) is asserted at the rising edge of the Capture clock when the Capture signal is high. The capture memory cell is continuously updated if the value of the FF is continuously changing. The final value of the capture memory cell occurs at the rising edge of the Capture clock when the Capture signal is low.

For details on the Virtex-II, Virtex-II Pro, and Virtex-II Pro X readback functions, see *The Programmable Logic Data Sheets*.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- CAPTURE_VIRTEX2: Register State Capture for Bitstream Readback
--                               Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide
```

```
CAPTURE_VIRTEX2_inst : CAPTURE_VIRTEX2
port map (
    CAP => CAP,    -- Capture input
    CLK => CLK     -- Clock input
);
```

Verilog Instantiation Template

```
// CAPTURE_VIRTEX2: Register State Capture for Bitstream Readback
//                               Virtex-II/II-Pro
// The current version of the Xilinx HDL Libraries Guide

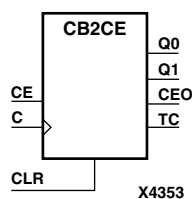
CAPTURE_VIRTEX2 CAPTURE_VIRTEX2_inst (
    .CAP(CAP),    // Capture input
    .CLK(CLK)    // Clock input
);
```

CB2CE, CB4CE, CB8CE, CB16CE

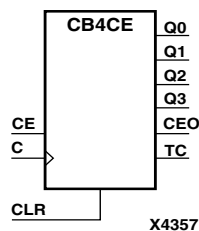
2-, 4-, 8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CB2CE, CB4CE, CB8CE, CB16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



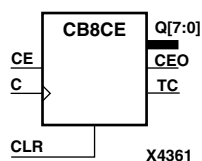
CB2CE, CB4CE, CB8CE, and CB16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

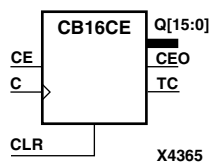
The counter is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

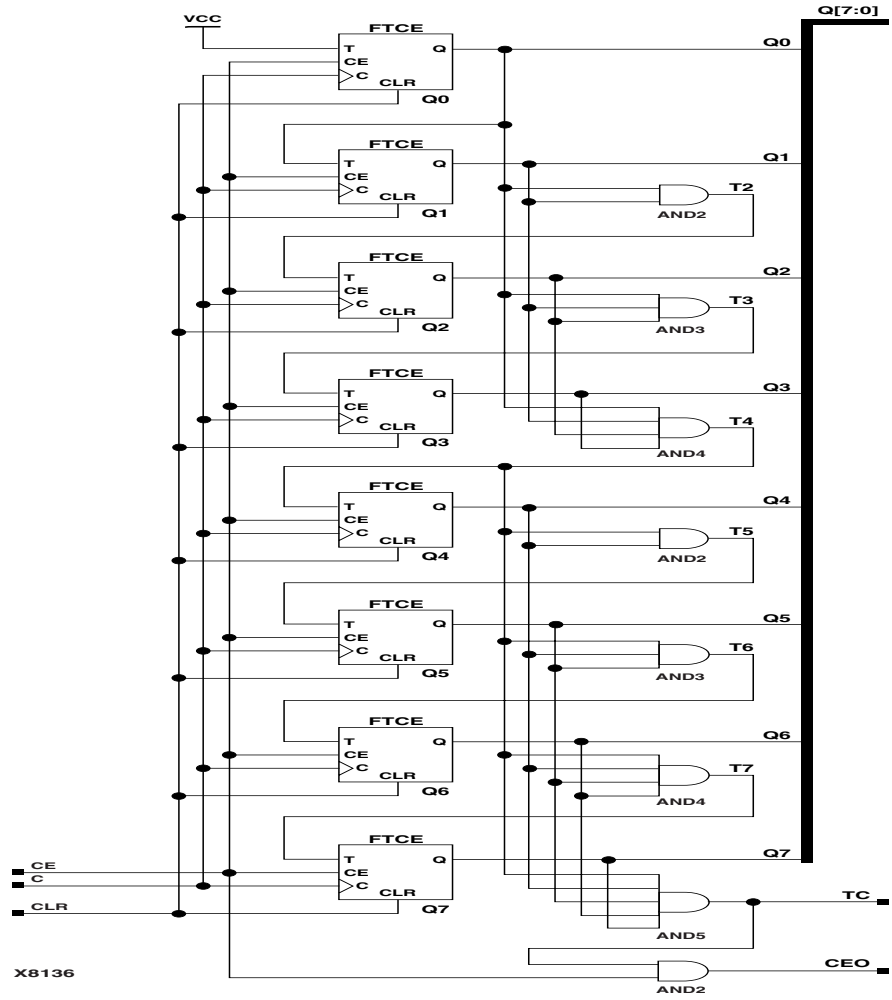


Inputs			Outputs		
CLR	CE	C	Qz-Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

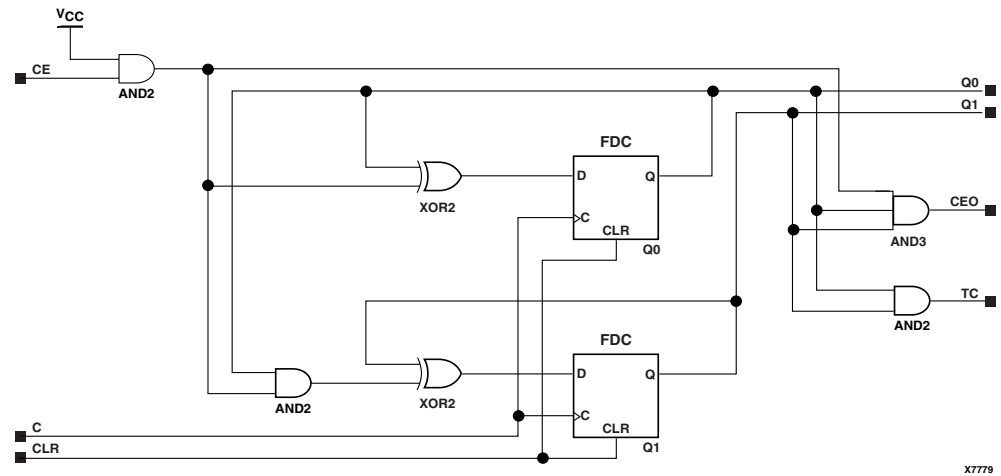
z = 1 for CB2CE; z = 3 for CB4CE; z = 7 for CB8CE; z = 15 for CB16CE

$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$

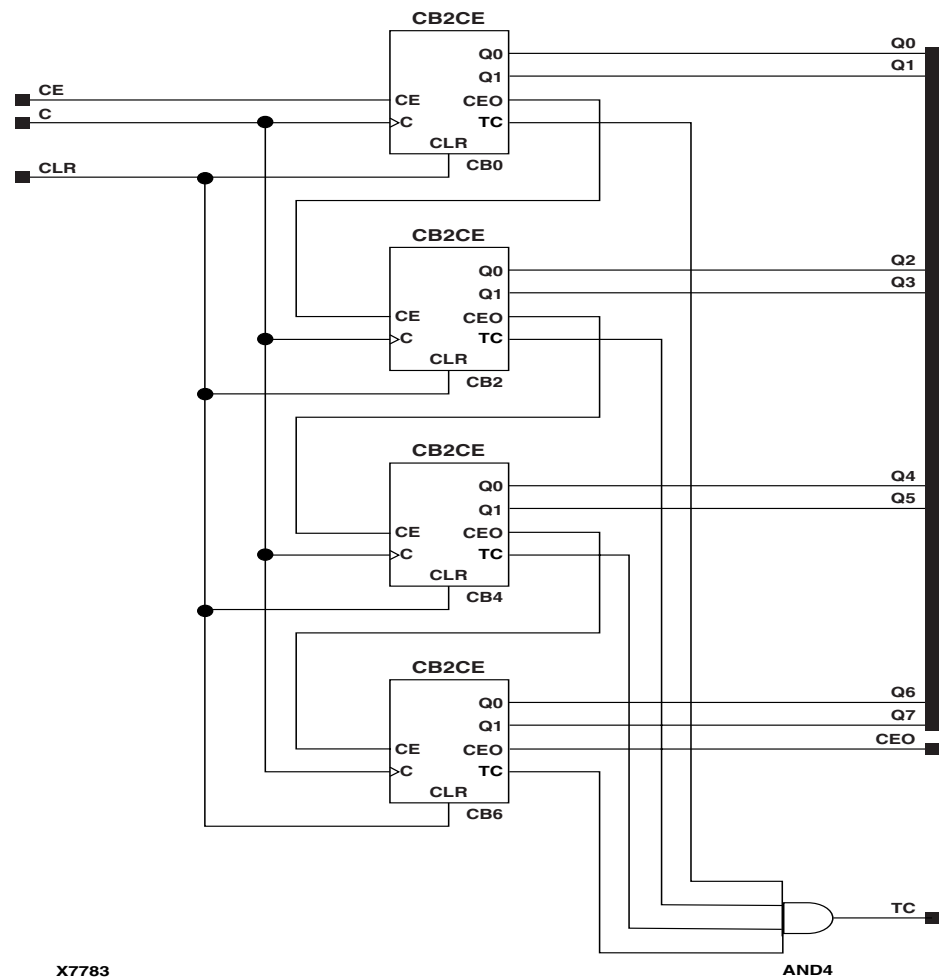
$CEO = TC \cdot CE$



CB8CE Implementation Spartan-II, Spartan-IIIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



CB2CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

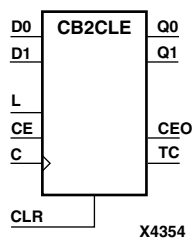
These design elements are inferred rather than instantiated.

CB2CLE, CB4CLE, CB8CLE, CB16CLE

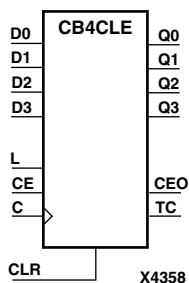
2-, 4-, 8-, 16-Bit Loadable, Cascadable, Bidirectional, Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

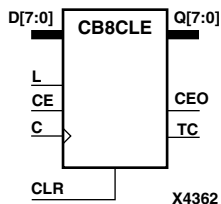
CB2CLE, CB4CLE, CB8CLE, CB16CLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CB2CLE, CB4CLE, CB8CLE, and CB16CLE are, respectively, 2-, 4-, 8-, and 16-bit (stage) synchronously loadable, asynchronously clearable, cascadable binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

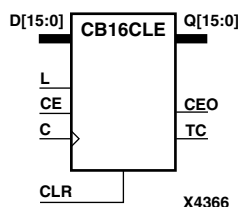


The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

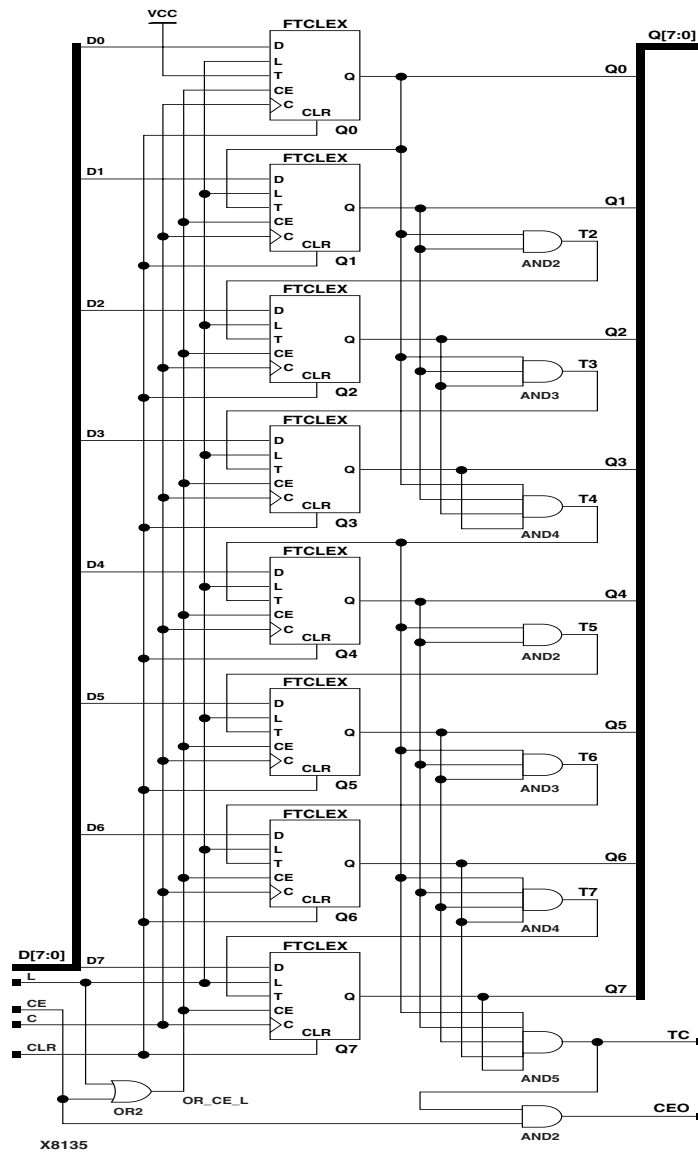


Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	Dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

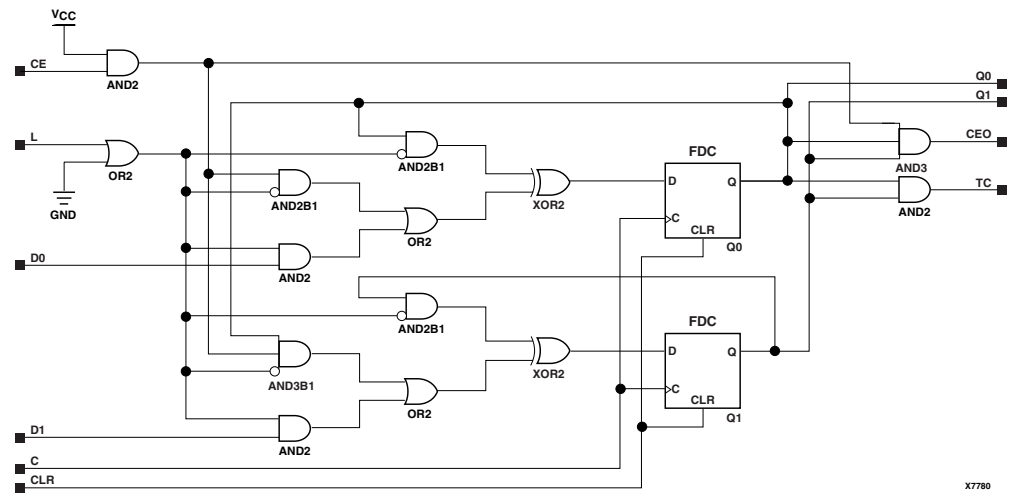
z = 1 for CB2CLE; z = 3 for CB4CLE; z = 7 for CB8CLE; z = 15 for CB16CLE

TC = $Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$

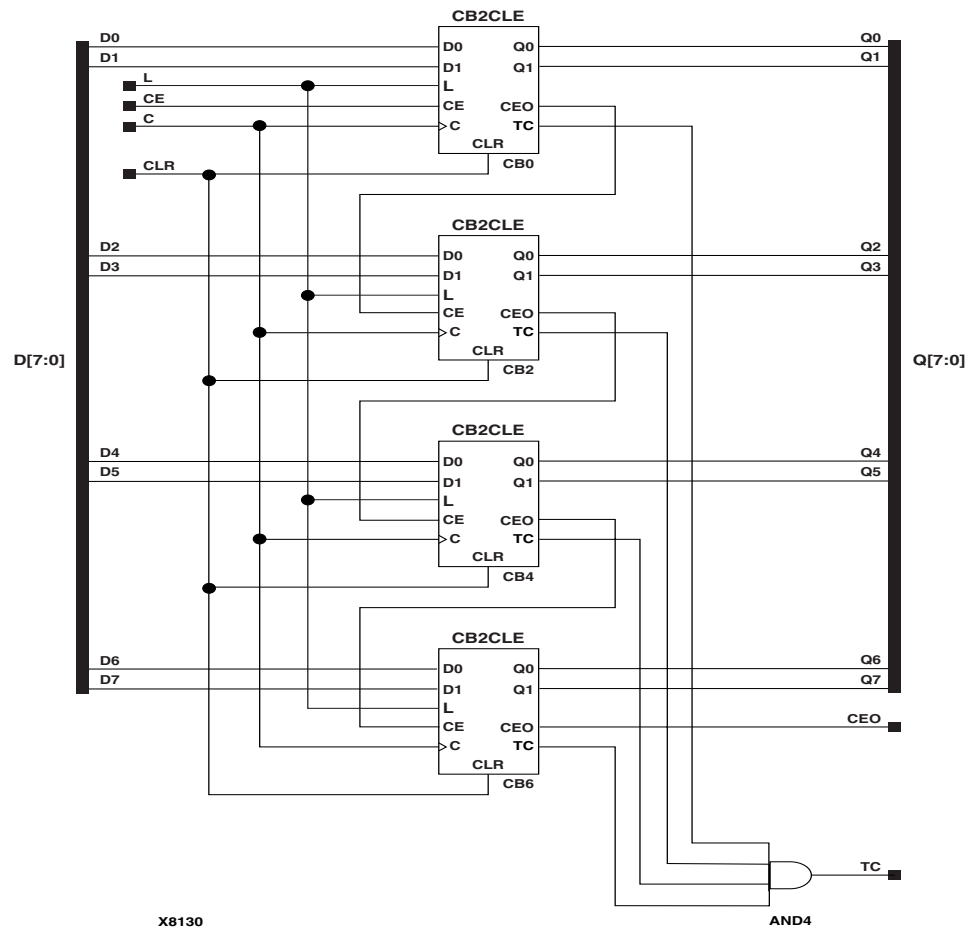
CEO = TC • CE



CB8CLE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



CB2CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

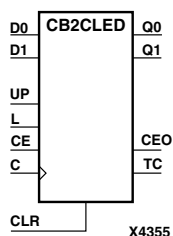
These design elements are inferred rather than instantiated.

CB2CLED, CB4CLED, CB8CLED, CB16CLED

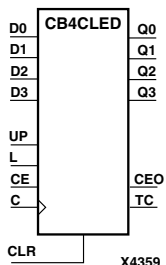
2-, 4-, 8-, 16-Bit Loadable, Cascadable, Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

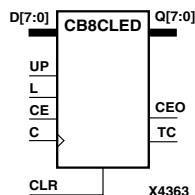


CB2CLED, CB4CLED, CB8CLED, and CB16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.



For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, see “[CB2X1](#), [CB4X1](#), [CB8X1](#), [CB16X1](#)” for high-performance cascadable, bidirectional counters.

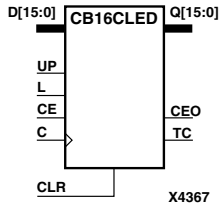


The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted with an inverter in front of the GSR input of STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2.

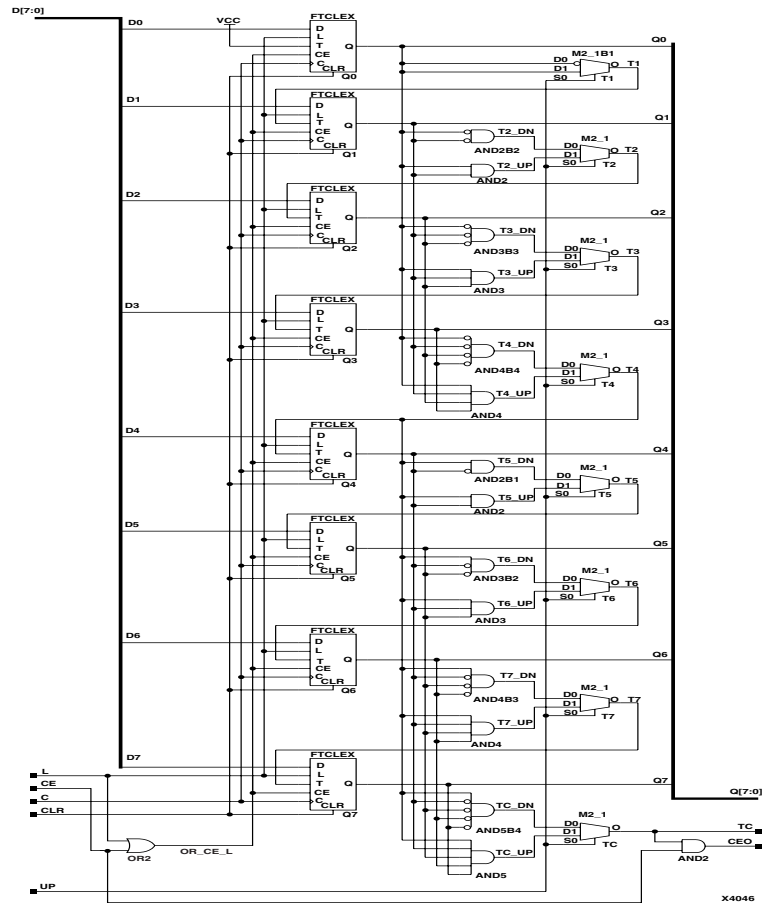


Inputs						Outputs		
CLR	L	CE	C	UP	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	X	0	↑	↑ *CE
0	1	X	↑	X	Dn	Dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

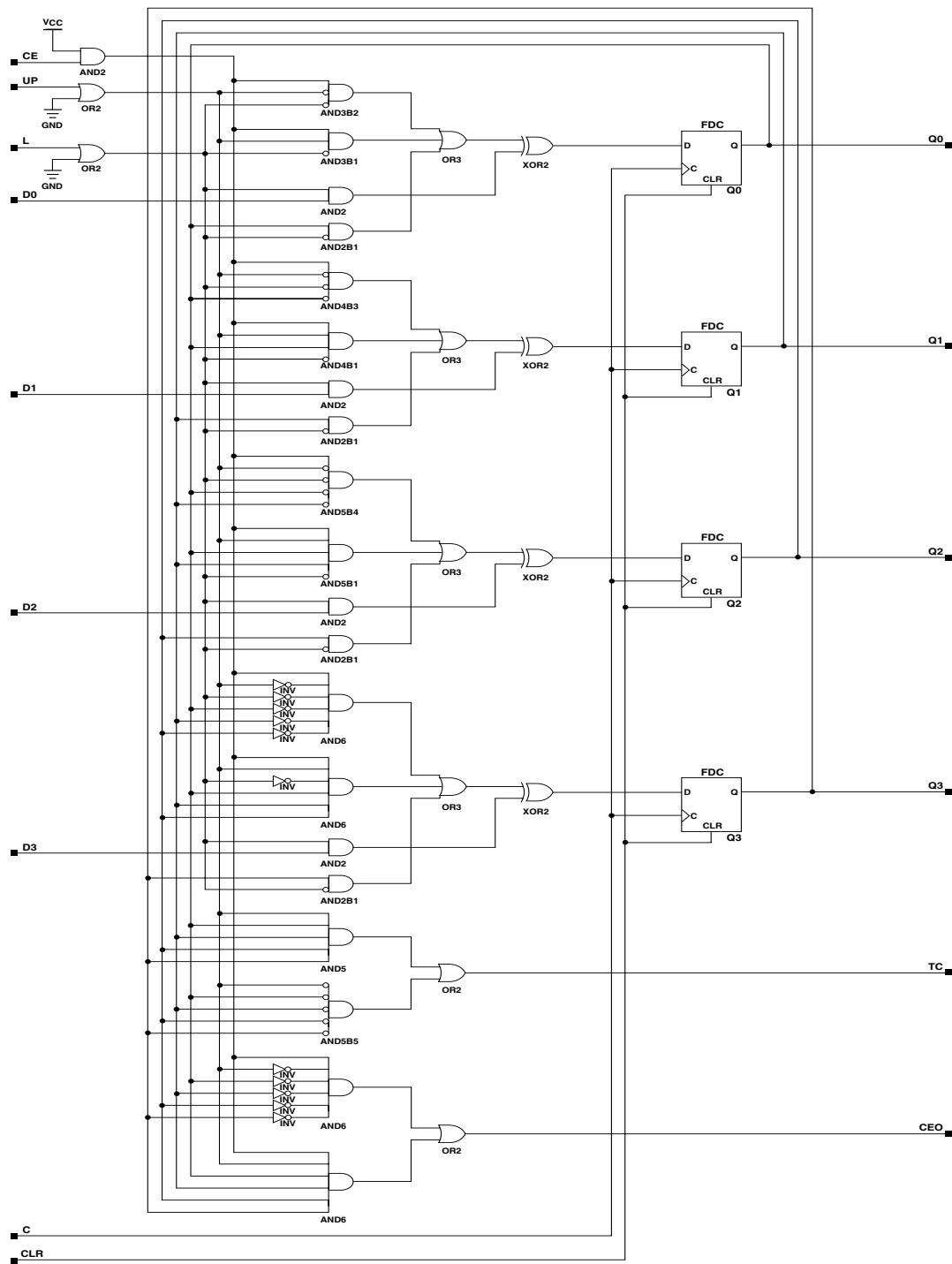
z = 1 for CB2CLED; z = 3 for CB4CLED; z = 7 for CB8CLED; z = 15 for CB16CLED

$$TC = (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot UP) + (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot \overline{UP})$$

$$CEO = TC \cdot CE$$



CB8CLED Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7425

CB4CLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

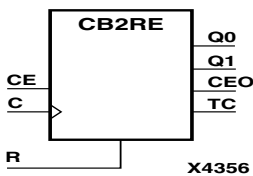
These design elements are inferred rather than instantiated.

CB2RE, CB4RE, CB8RE, CB16RE

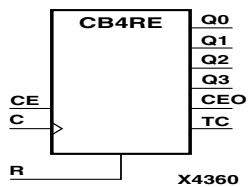
2-, 4-, 8-, 16-Bit Cascadable, Bidirectional, Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

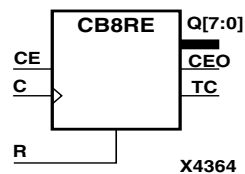
CB2RE, CB4RE, CB8RE, CB16RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CB2RE, CB4RE, CB8RE, and CB16RE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, resettable, cascadable binary counters. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero during the Low-to-High clock transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when both Q outputs are High.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

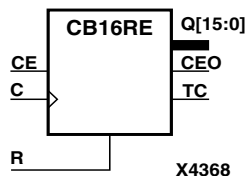


The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



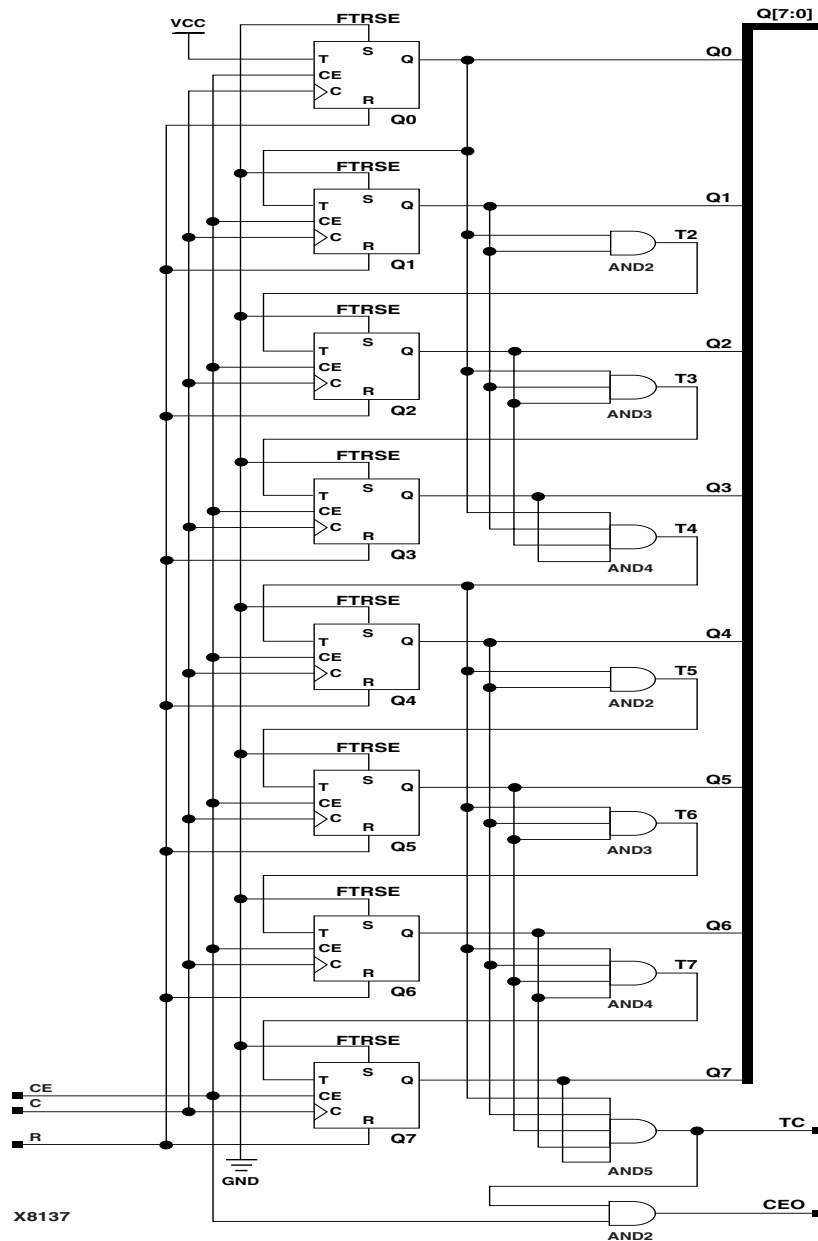
Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
0	1	↑	Inc	TC	CEO

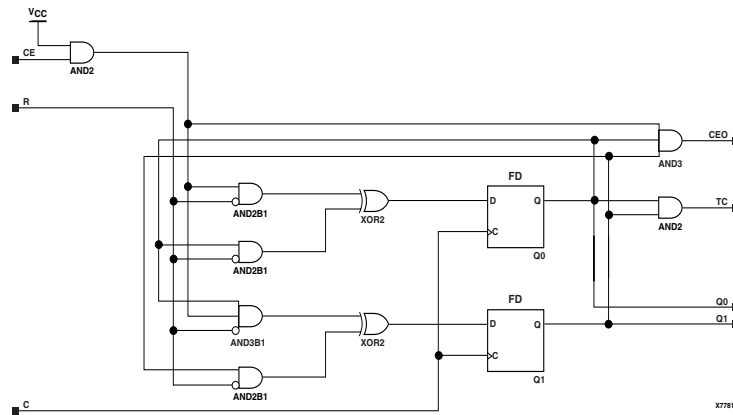
z = 1 for CB2RE; z = 3 for CB4RE; z = 7 for CB8RE; z = 15 for CB16RE

$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

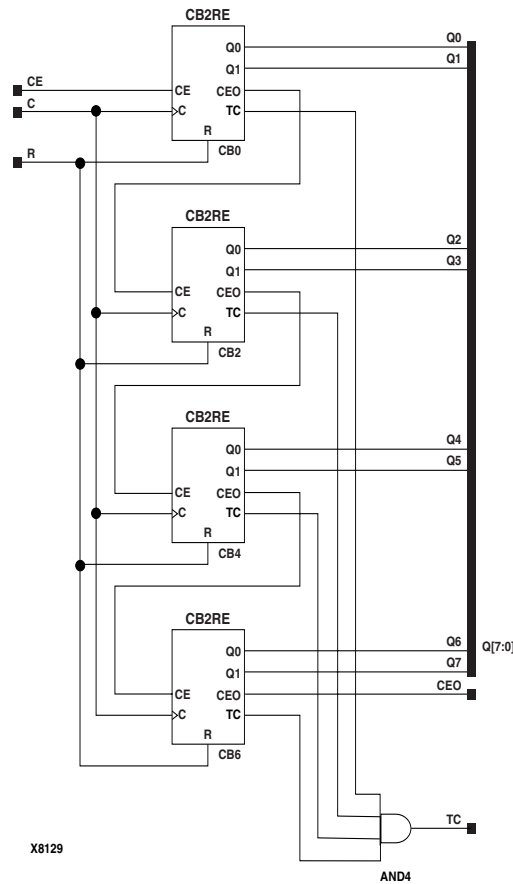
$$CEO = TC \cdot CE$$



CB8RE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



CB2RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

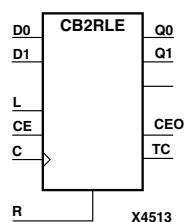
For HDL, these design elements are inferred rather than instantiated.

CB2RLE, CB4RLE, CB8RLE, CB16RLE

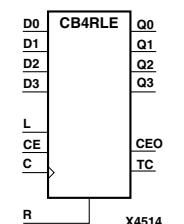
2-, 4-, 8-, 16-Bit Loadable, Cascadable, Bidirectional, Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

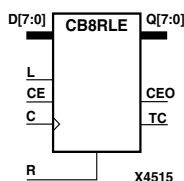
CB2RLE, CB4RLE, CB8RLE, CB16RLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



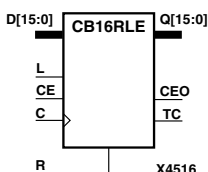
CB2RLE, CB4RLE, CB8RLE, and CB16RLE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, cascadable binary counters. The synchronous reset (R) is the highest priority input. The synchronous R, when High, overrides all other inputs and resets the Q outputs, terminal count (TC), and clock enable out (CEO) outputs to Low on the Low-to-High clock (C) transition.



The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of CE. The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High. The CEO output is High when all Q outputs and CE are High to allow direct cascading of counters.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and by connecting the C, L, and R inputs in parallel. The maximum length of the counter is determined by the accumulated CE-to-CEO propagation delays versus the clock period. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.



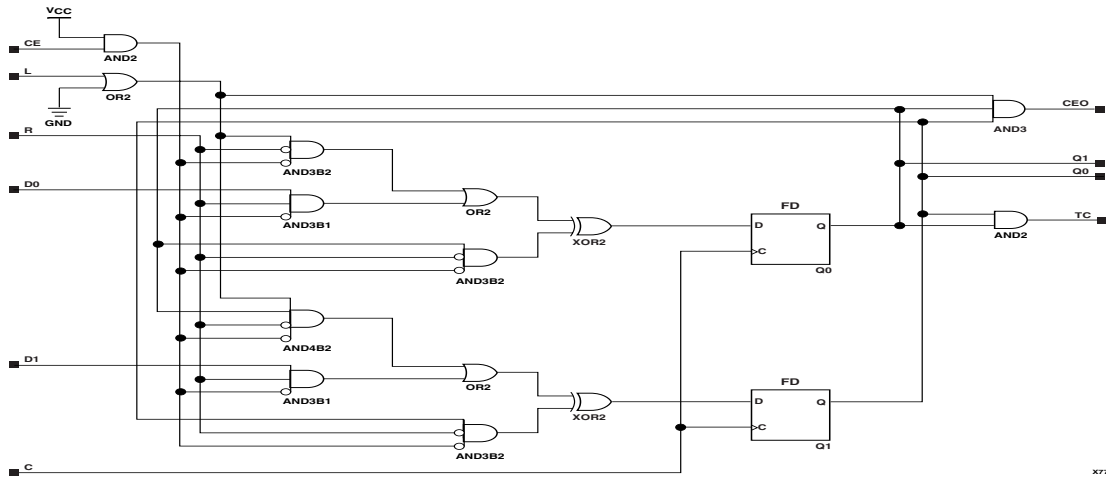
The counter is asynchronously cleared, output Low, when power is applied. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs		
R	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	↑	X	0	0	0
0	1	X	↑	Dn	Dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

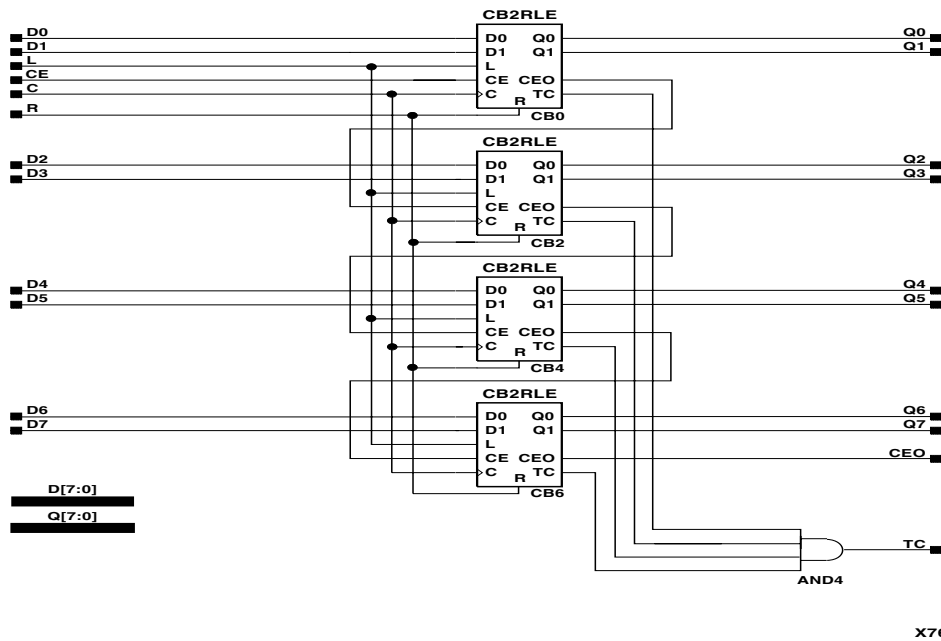
z = 1 for CB2RLE; z = 3 for CB4RLE; z = 7 for CB8RLE; z = 15 for CB16RLE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0

CEO = TC • CE



CB2RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



CB8RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

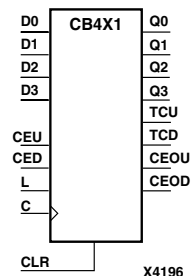
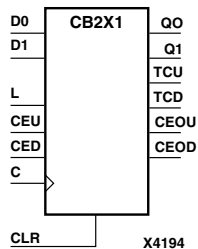
These design elements are inferred rather than instantiated.

CB2X1, CB4X1, CB8X1, CB16X1

2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CB2X1, CB4X1	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
CB8X1, CB16X1	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

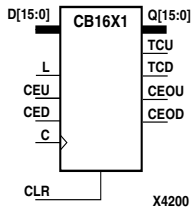
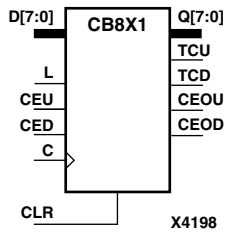


CB2X1, CB4X1, CB8X1, and CB16X1 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronously loadable, asynchronously clearable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, clock enable outputs CEOU and CEOD go to Low and High, respectively, independent of clock transitions. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

The Q outputs increment when CEU is High, provided CLR and L are Low, during the Low-to-High clock transition. The Q outputs decrement when CED is High, provided CLR and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are



connected directly to the CEU and CED inputs, respectively, of the next stage. The clock, L, and CLR inputs are connected in parallel.

The maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
CLR	L	CEU	CED	C	Dz-D0	Qz-Q0	TCU	TCD	CEOU	CEOD
1	X	X	X	X	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	Dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

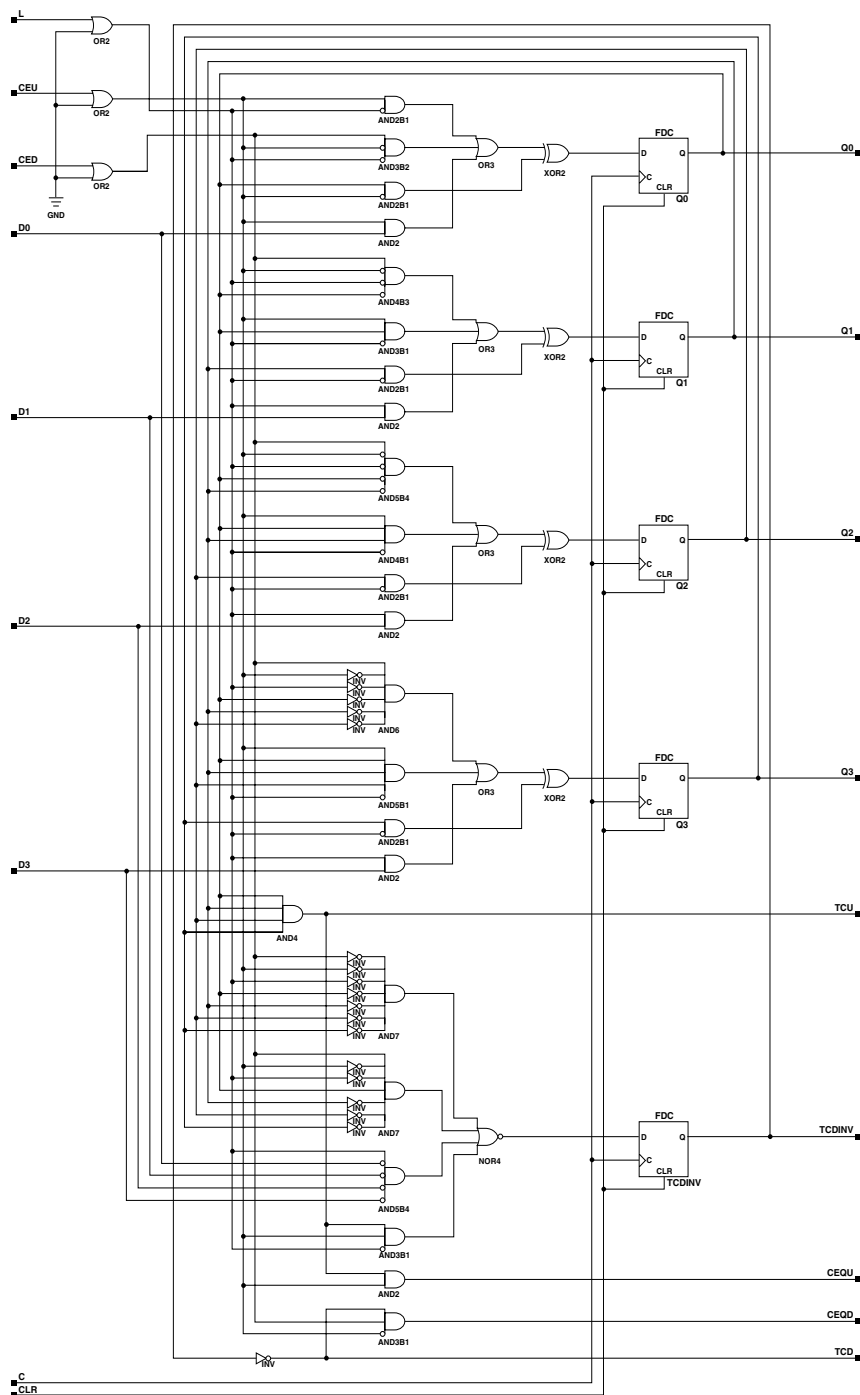
z = 1 for CB2X1; z = 3 for CB4X1; z = 7 for CB8X1; z = 15 for CB16X1

$$TCU = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$TCD = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEOU = TCU \cdot CEU$$

$$CEOD = TCD \cdot CED$$



X7624

CB4X1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

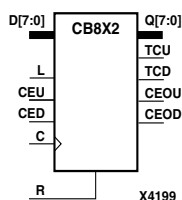
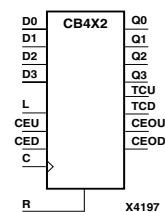
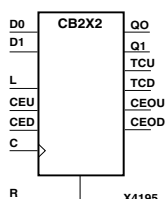
For HDL, these design elements are inferred rather than instantiated.

CB2X2, CB4X2, CB8X2, CB16X2

2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

CB2X2	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
CB4X2, CB8X2, CB16X2	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

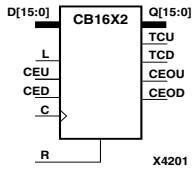


CB2X2, CB4X2, CB8X2, and CB16X2 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, bidirectional binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in CPLD architectures.

The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, and clock enable outputs CEOU and CEOD go to Low and High, respectively, on the Low-to-High clock (C) transition. The data on the D inputs loads into the counter on the Low-to-High clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

All Q outputs increment when CEU is High, provided R and L are Low during the Low-to-High clock transition. All Q outputs decrement when CED is High, provided R and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are, respectively, connected directly to the CEU and CED inputs of the next stage. The C, L, and R inputs are connected in parallel.



The maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
R	L	CEU	CED	C	Dz – D0	Qz – Q0	TCU	TCD	CEOU	CEOD
1	X	X	X	↑	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	Dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid

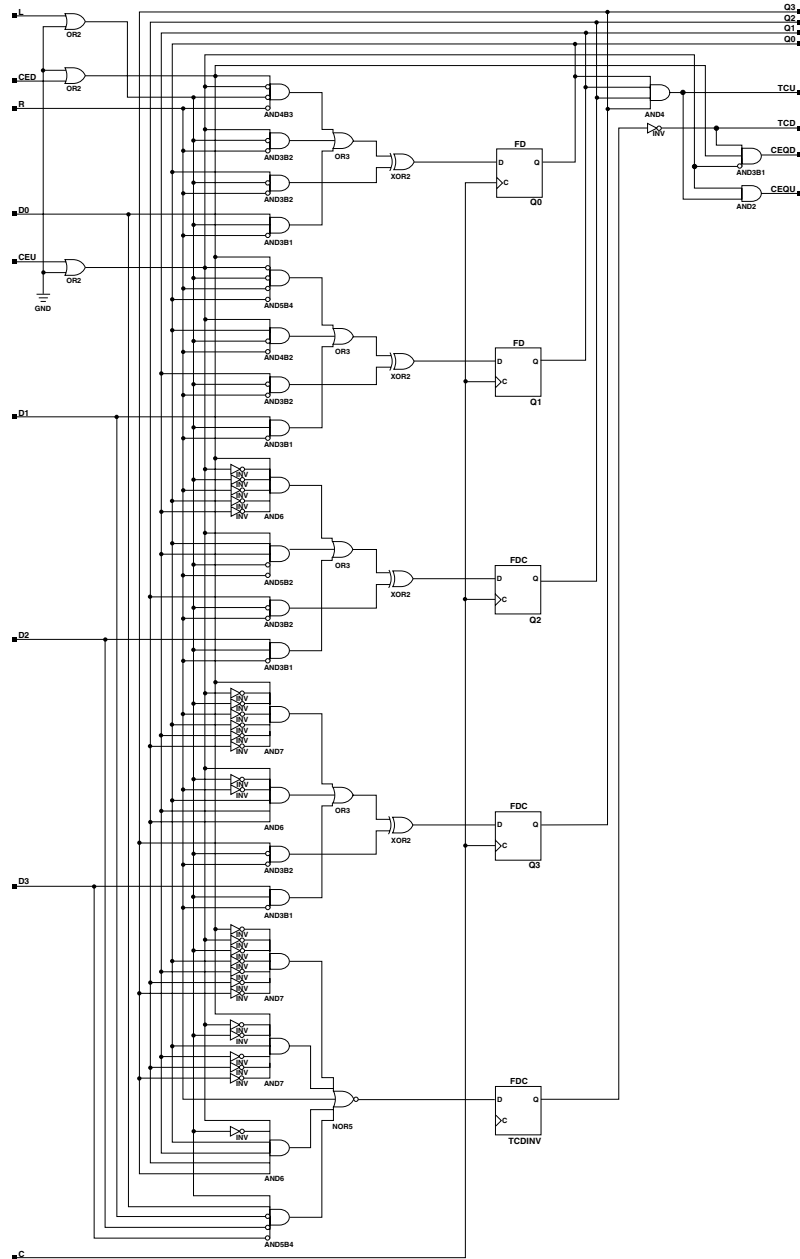
z = 1 for CB2X2; z = 3 for CB4X2; z = 7 for CB8X2; z = 15 for CB16X2

$$TCU = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$TCD = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEOU = TCU \cdot CEU$$

$$CEOD = TCD \cdot CED$$



X7623

CB4X2 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

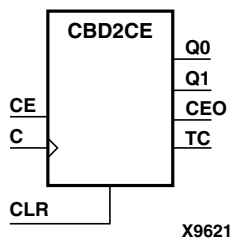
For HDL, these design elements are inferred rather than instantiated.

CBD2CE, CBD4CE, CBD8CE, CBD16CE

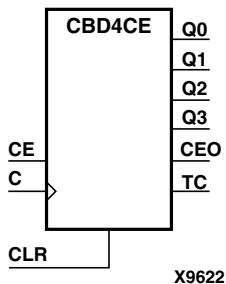
2-, 4-, 8-, 16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

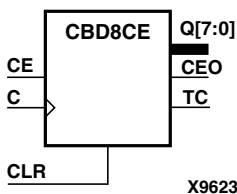
CBD2CE, CBD4CE, CBD8CE, CBD16CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



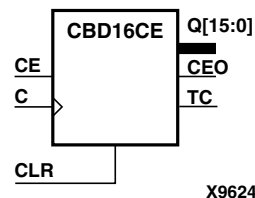
CBD2CE, CBD4CE, CBD8CE, and CBD16CE are, respectively, 2-, 4-, 8-, and 16-bit (stage), asynchronous, clearable, cascadable dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

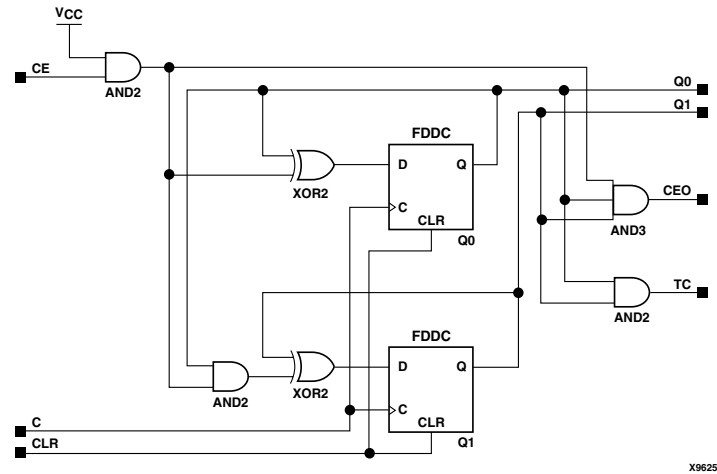


Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

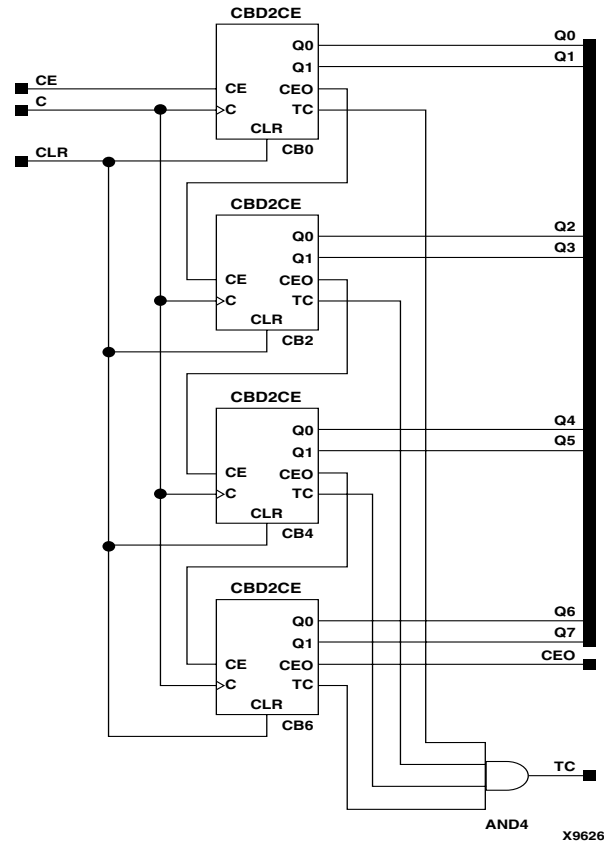


The counter is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.





CBD2CE Implementation CoolRunner-II



CBD8CE Implementation CoolRunner-II

Usage

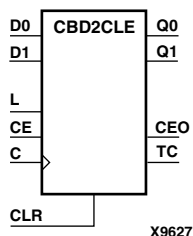
For HDL, these design elements are supported for inference but not instantiation.

CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE

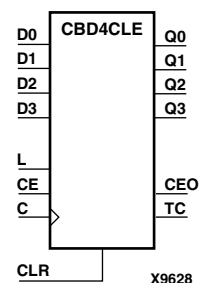
2-, 4-, 8-, 16-Bit Loadable Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

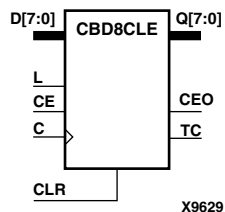
CBD2CLE, CBD4CLE, CBD8CLE, CBD16CLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



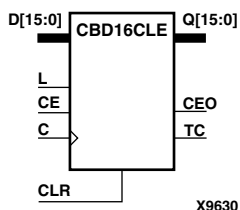
CBD2CLE, CBD4CLE, CBD8CLE, and CBD16CLE are, respectively, 2-, 4-, 8-, and 16-bit (stage) synchronously loadable, asynchronously clearable, cascadable dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.



The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



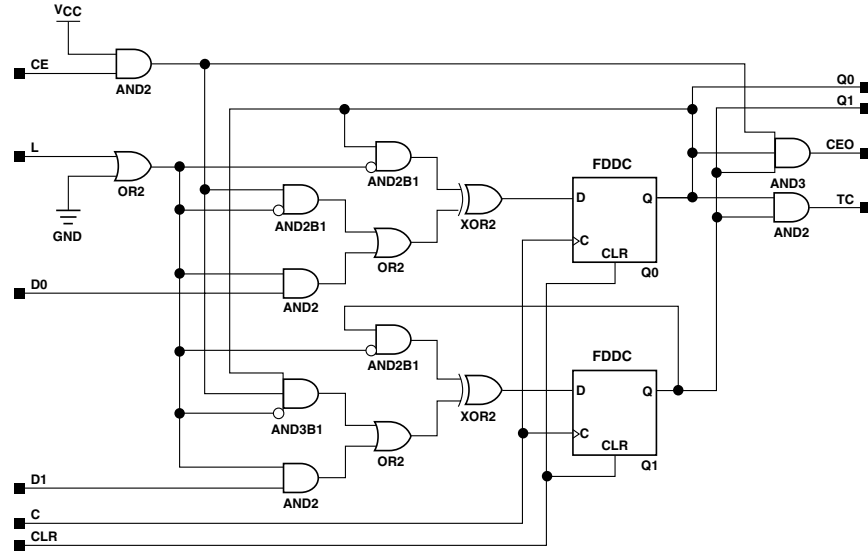
Inputs					Outputs			
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO	
1	X	X	X	X	0	0	0	
0	1	X	↑	Dn	Dn	TC	CEO	
0	1	X	↓	Dn	Dn	TC	CEO	
0	0	0	X	X	No Chg	No Chg	0	

Inputs					Outputs		
CLR	L	CE	C	Dz - D0	Qz - Q0	TC	CEO
0	0	1	↑	X	Inc	TC	CEO
0	0	1	↓	X	Inc	TC	CEO

z = 1 for CBD2CLE; z = 3 for CBD4CLE; z = 7 for CBD8CLE; z = 15 for CBD16CLE

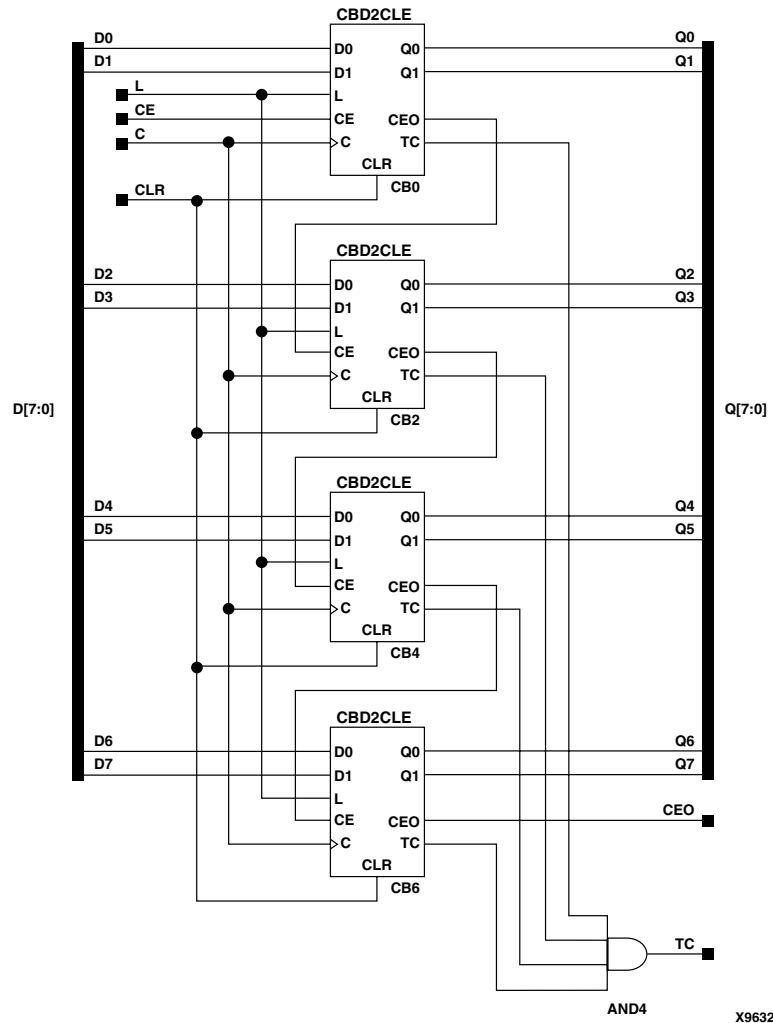
$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEO = TC \cdot CE$$



X9631

CBD2CLE Implementation CoolRunner-II



CBD8CLE Implementation CoolRunner-II

Usage

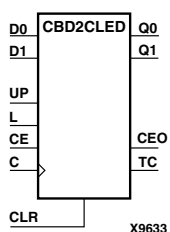
For HDL, these design elements are supported for inference but not instantiation.

CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED

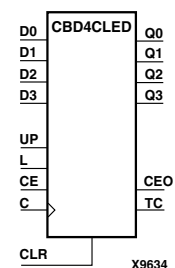
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

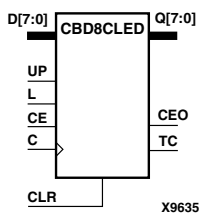
CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



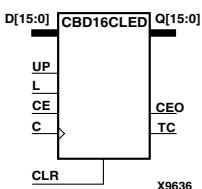
CBD2CLED, CBD4CLED, CBD8CLED, and CBD16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional dual edge triggered binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High and High-to-Low clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.



For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage.



When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For CoolRunner-II, see “[CB2X1](#), [CB4X1](#), [CB8X1](#), [CB16X1](#)” for high-performance cascadable, bidirectional counters.



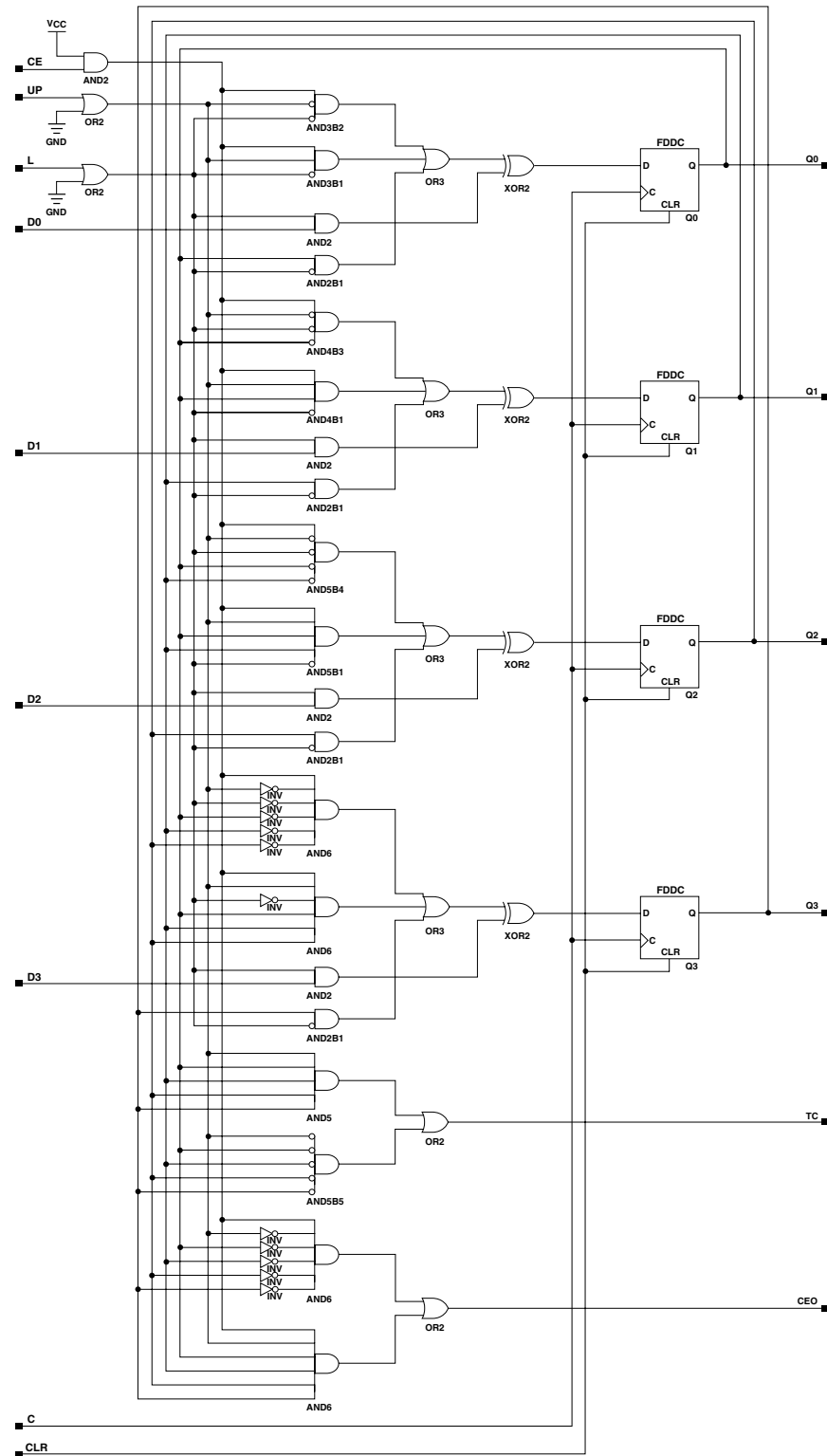
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs		
CLR	L	CE	C	UP	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	Dn	TC	CEO
0	1	X	↓	X	Dn	Dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↓	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO
0	0	1	↓	0	X	Dec	TC	CEO

$z = 1$ for CBD2CLED; $z = 3$ for CBD4CLED; $z = 7$ for CBD8CLED; $z = 15$ for CBD16CLED

$TC = (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot UP) + (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot \overline{UP})$

$CEO = TC \cdot CE$



X3637

CBD4CLED Implementation CoolRunner-II

Usage

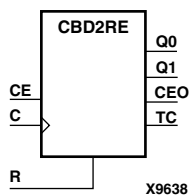
For HDL, these design elements are supported for inference but not instantiation.

CBD2RE, CBD4RE, CBD8RE, CBD16RE

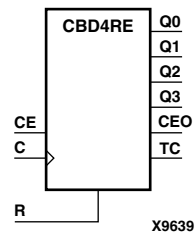
2-, 4-, 8-, 16-Bit Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

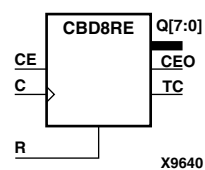
CBD2RE, CBD4RE, CBD8RE, CBD16RE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



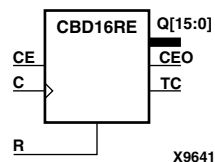
CBD2RE, CBD4RE, CBD8RE, and CBD16RE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, resettable, cascadable dual edge triggered binary counters. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero during the Low-to-High or High-to-Low clock transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when both Q outputs are High.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.



The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



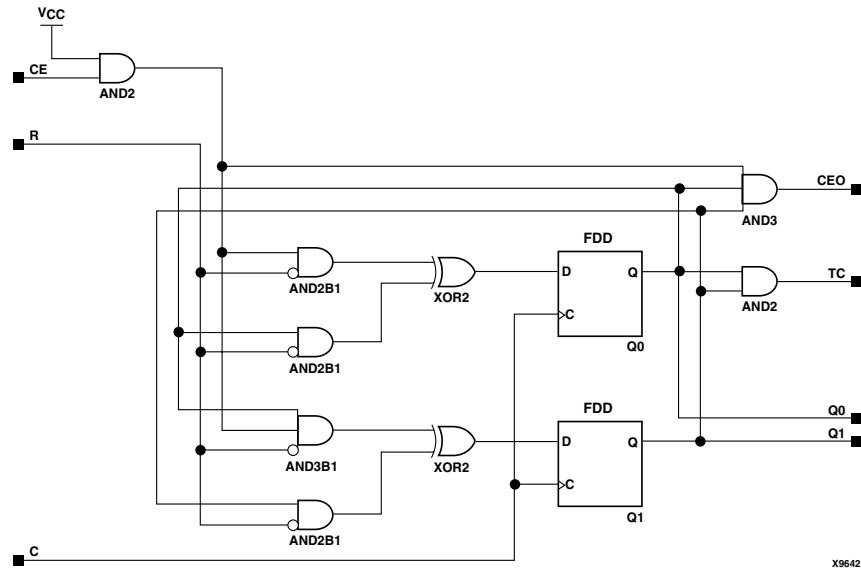
Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
1	X	↓	0	0	0
0	0	X	No Chg	No Chg	0
0	1	↑	Inc	TC	CEO

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
0	1	↓	Inc	TC	CEO

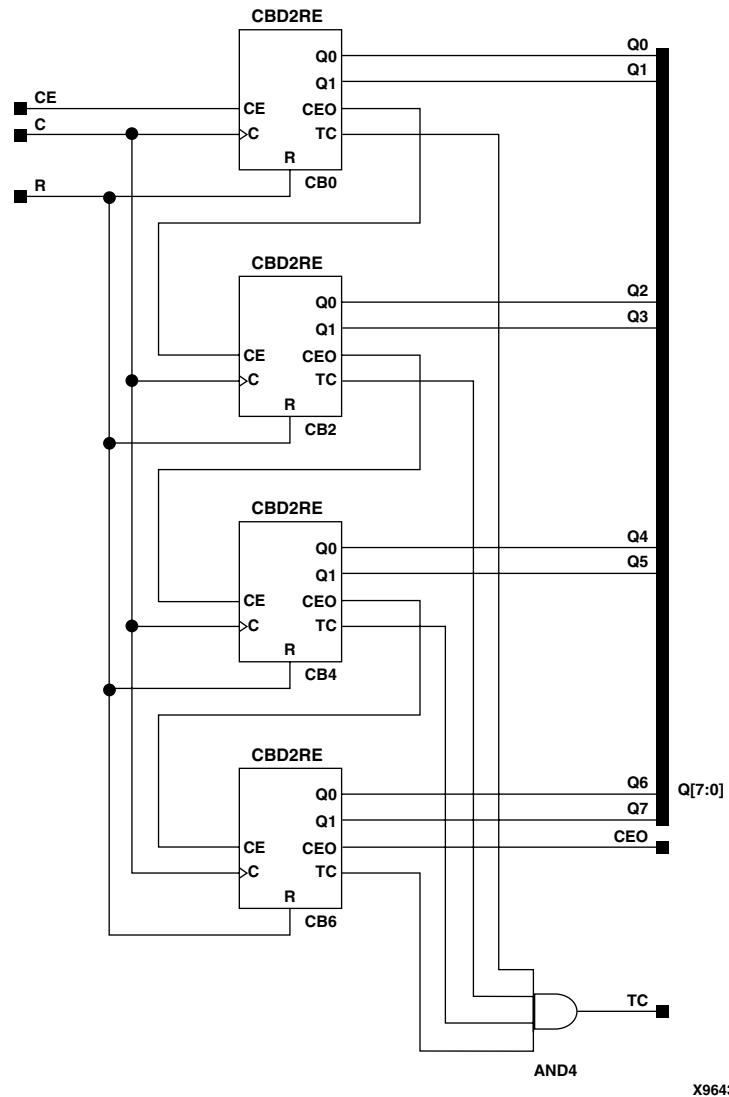
z = 1 for CBD2RE; z = 3 for CBD4RE; z = 7 for CBD8RE; z = 15 for CBD16RE

$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEO = TC \cdot CE$$



CBD2RE Implementation CoolRunner-II



CBD8RE Implementation CoolRunner-II

Usage

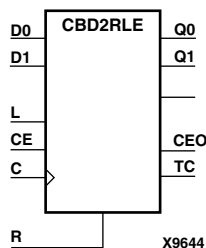
For HDL, these design elements are supported for inference but not instantiation.

CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE

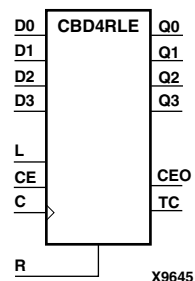
2-, 4-, 8-, 16-Bit Loadable Cascadable Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

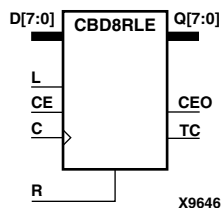
CBD2RLE, CBD4RLE, CBD8RLE, CBD16RLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



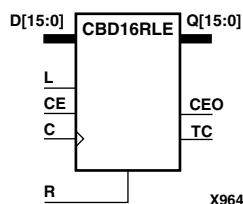
CBD2RLE, CBD4RLE, CBD8RLE, and CBD16RLE are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, cascadable dual edge triggered binary counters. The synchronous reset (R) is the highest priority input. The synchronous R, when High, overrides all other inputs and resets the Q outputs, terminal count (TC), and clock enable out (CEO) outputs to Low on the Low-to-High or High-to-Low clock (C) transition.



The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transition, independent of the state of CE. The Q outputs increment when CE is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High. The CEO output is High when all Q outputs and CE are High to allow direct cascading of counters.



Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and by connecting the C, L, and R inputs in parallel. The maximum length of the counter is determined by the accumulated CE-to-CEO propagation delays versus the clock period. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.



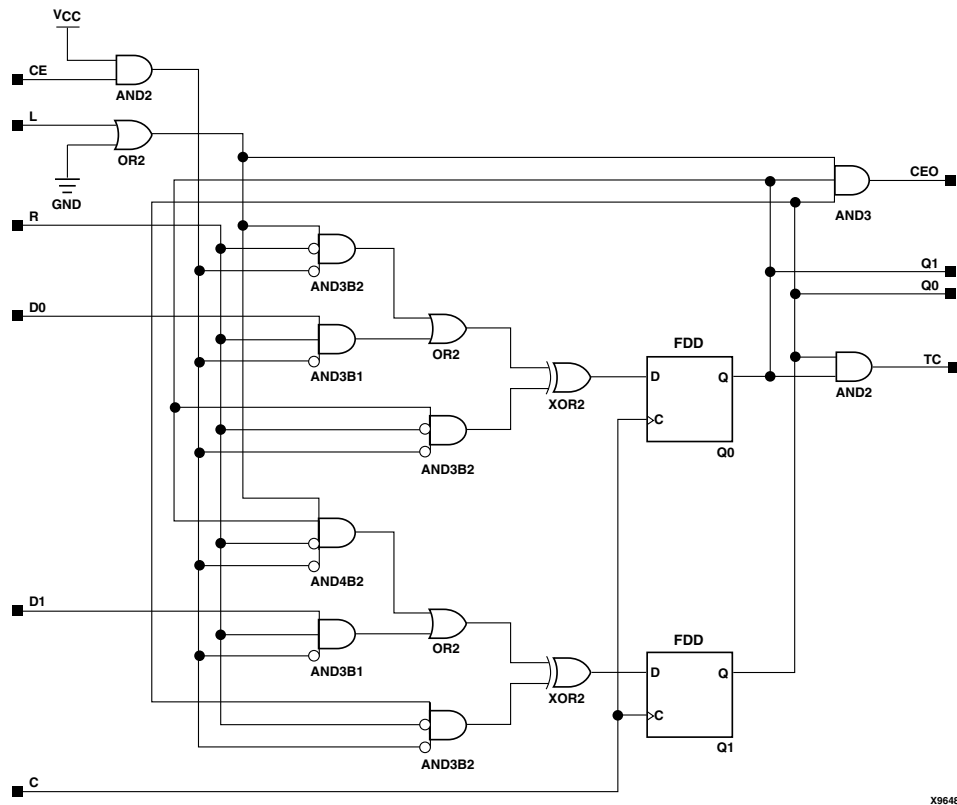
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs		
R	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	↑	X	0	0	0
1	X	X	↓	X	0	0	0
0	1	X	↑	Dn	Dn	TC	CEO
0	1	X	↓	Dn	Dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO
0	0	1	↓	X	Inc	TC	CEO

z = 1 for CBD2RLE; z = 3 for CBD4RLE; z = 7 for CBD8RLE; z = 15 for CBD16RLE

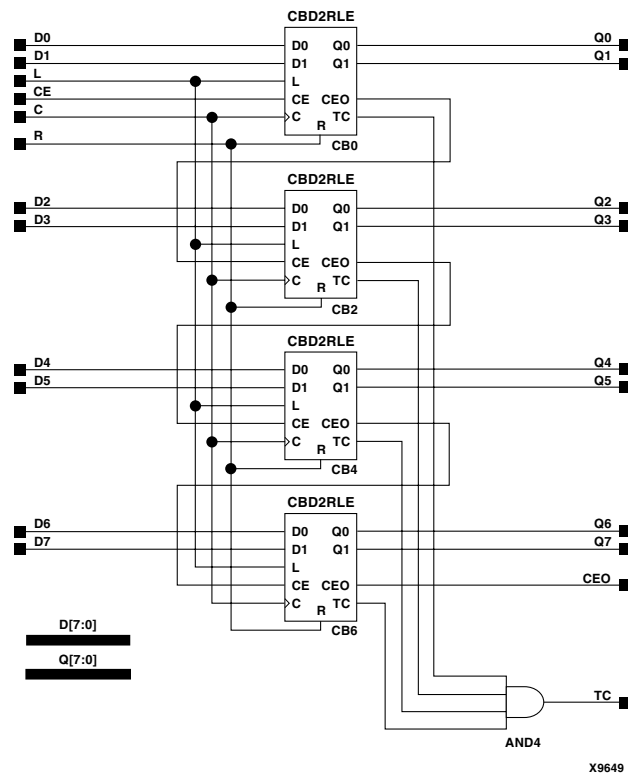
$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEO = TC \cdot CE$$



X9648

CBD2RLE Implementation CoolRunner-II



CBD8RLE Implementation CoolRunner-II

Usage

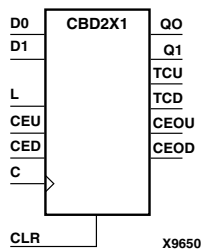
For HDL, these design elements are supported for inference but not instantiation.

CBD2X1, CBD4X1, CBD8X1, CBD16X1

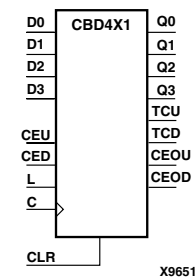
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

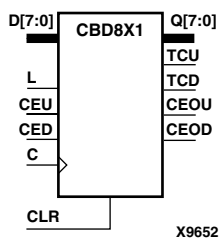
CBD2X1, CBD4X1, CBD8X1, CBD16X1	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CBD2X1, CBD4X1, CBD8X1, and CBD16X1 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronously loadable, asynchronously clearable, bidirectional dual edge triggered binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CoolRunner-II architecture.



The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, clock enable outputs CEOU and CEOD go to Low and High, respectively, independent of clock transitions. The data on the D inputs loads into the counter on the Low-to-High and High-to-Low clock (C) transition when the load enable input (L) is High, independent of the CE inputs.

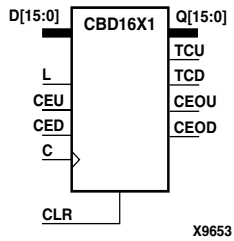


The Q outputs increment when CEU is High, provided CLR and L are Low, during the Low-to-High and High-to-Low clock transition. The Q outputs decrement when CED is High, provided CLR and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are connected directly to the CEU and CED inputs, respectively, of the next stage. The clock, L, and CLR inputs are connected in parallel.

In CoolRunner-II, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable



AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
CLR	L	CEU	CED	C	Dz-D0	Qz-Q0	TCU	TCD	CEOU	CEOD
1	X	X	X	X	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	Dn	TCU	TCD	CEOU	CEOD
0	1	X	X	↓	Dn	Dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	1	0	↓	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	0	1	↓	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid
0	0	1	1	↓	X	Inc	TCU	TCD	Invalid	Invalid

z = 1 for CBD2X1; z = 3 for CBD4X1; z = 7 for CBD8X1; z = 15 for CBD16X1

$$TCU = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$TCD = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEOU = TCU \cdot CEU$$

$$CEOD = TCD \cdot CED$$

CBD4X1 Implementation CoolRunner-II

Usage

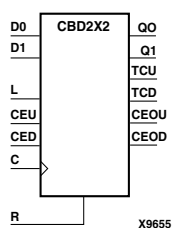
For HDL, these design elements are inferred rather than instantiated.

CBD2X2, CBD4X2, CBD8X2, CBD16X2

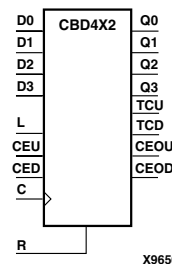
2-, 4-, 8-, and 16-Bit Loadable Cascadable Bidirectional Dual Edge Triggered Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

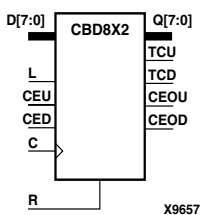
CBD2X2, CBD4X2, CBD8X2, CBD16X2	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CBD2X2, CBD4X2, CBD8X2, and CBD16X2 are, respectively, 2-, 4-, 8-, and 16-bit (stage), synchronous, loadable, resettable, bidirectional dual edge triggered binary counters. These counters have separate count-enable inputs and synchronous terminal-count outputs for up and down directions to support high-speed cascading in the CoolRunner-II architecture.

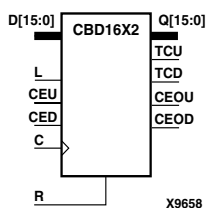


The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the data outputs (Q) go to logic level zero, terminal count outputs TCU and TCD go to zero and one, respectively, and clock enable outputs CEOU and CEOD go to Low and High, respectively, on the Low-to-High and High-to-Low clock (C) transition. The data on the D inputs loads into the counter on the Low-to-High and High-to-Low clock (C) transition when the load enable input (L) is High, independent of the CE inputs.



All Q outputs increment when CEU is High, provided R and L are Low during the Low-to-High and High-to-Low clock transition. All Q outputs decrement when CED is High, provided R and L are Low. The counter ignores clock transitions when CEU and CED are Low. Both CEU and CED should not be High during the same clock transition; the CEOU and CEOD outputs might not function properly for cascading when CEU and CED are both High.

For counting up, the CEOU output is High when all Q outputs and CEU are High. For counting down, the CEOD output is High when all Q outputs are Low and CED is High. To cascade counters, the CEOU and CEOD outputs of each counter are, respectively, connected directly to the CEU and CED inputs of the next stage. The C, L, and R inputs are connected in parallel.



In CoolRunner-II, the maximum clocking frequency of these counter components is unaffected by the number of cascaded stages for all counting and loading functions. The TCU terminal count output is High when all Q outputs are High, regardless of CEU. The TCD output is High when all Q outputs are Low, regardless of CED.

When cascading counters, the final terminal count signals can be produced by AND wiring all the TCU outputs (for the up direction) and all the TCD outputs (for the down direction). The TCU, CEOU, and CEOD outputs are produced by optimizable

AND gates within the component. This results in zero propagation from the CEU and CED inputs and from the Q outputs, provided all connections from each such output remain on-chip. Otherwise, a macrocell buffer delay is introduced.

The counter is initialized to zero (TCU Low and TCD High) when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs				
R	L	CEU	CED	C	Dz – D0	Qz – Q0	TCU	TCD	CEOU	CEOD
1	X	X	X	↑	X	0	0	1	0	CEOD
1	X	X	X	↓	X	0	0	1	0	CEOD
0	1	X	X	↑	Dn	Dn	TCU	TCD	CEOU	CEOD
0	1	X	X	↓	Dn	Dn	TCU	TCD	CEOU	CEOD
0	0	0	0	X	X	No Chg	No Chg	No Chg	0	0
0	0	1	0	↑	X	Inc	TCU	TCD	CEOU	0
0	0	1	0	↓	X	Inc	TCU	TCD	CEOU	0
0	0	0	1	↑	X	Dec	TCU	TCD	0	CEOD
0	0	0	1	↓	X	Dec	TCU	TCD	0	CEOD
0	0	1	1	↑	X	Inc	TCU	TCD	Invalid	Invalid
0	0	1	1	↓	X	Inc	TCU	TCD	Invalid	Invalid

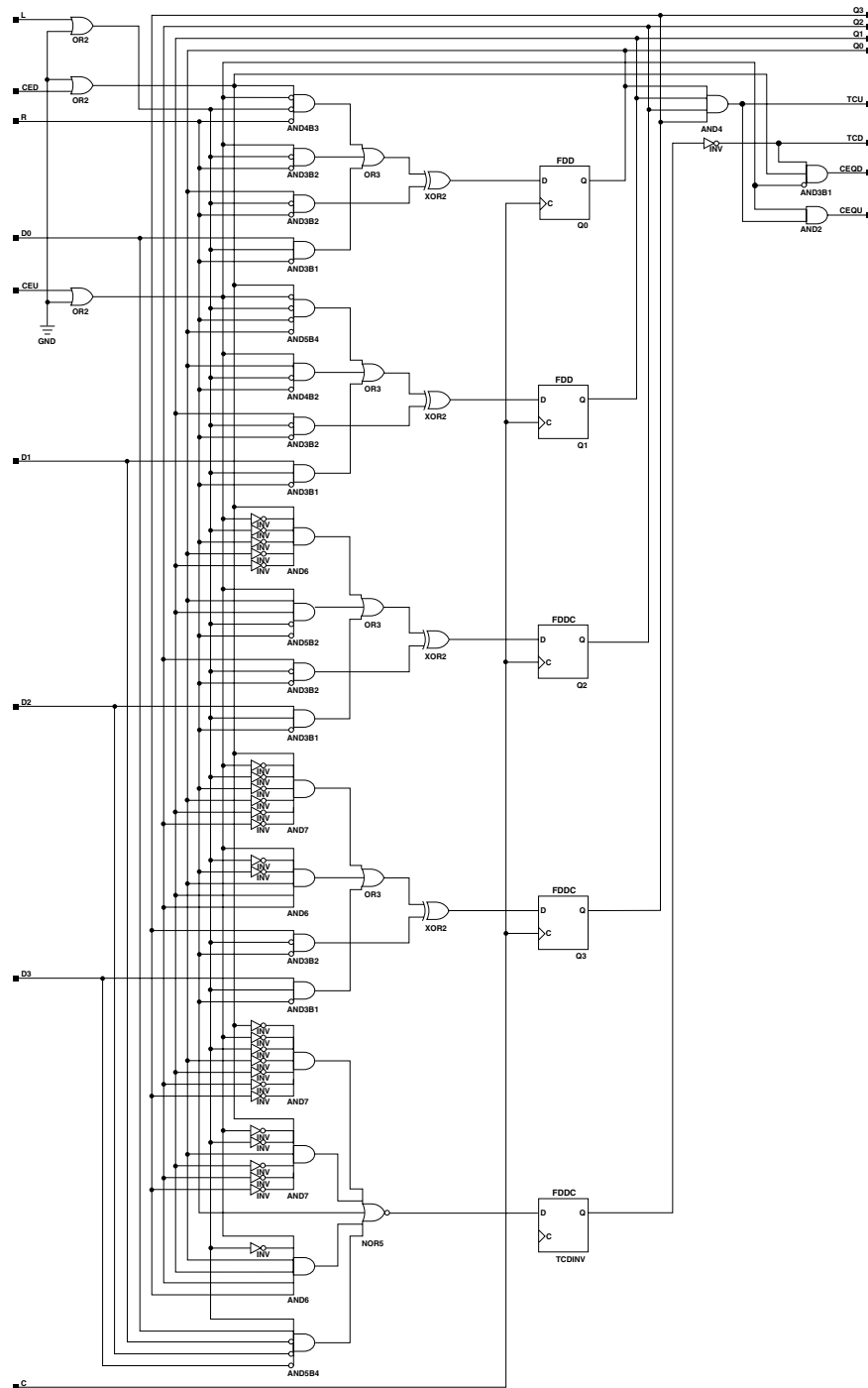
$z = 1$ for CBD2X2; $z = 3$ for CBD4X2; $z = 7$ for CBD8X2; $z = 15$ for CBD16X2

$TCU = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$

$TCD = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$

$CEOU = TCU \cdot CEU$

$CEOD = TCD \cdot CED$



X0699

CBD4X2 Implementation CoolRunner-II

Usage

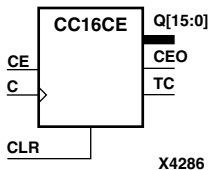
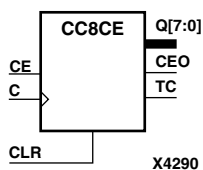
For HDL, these design elements are inferred rather than instantiated.

CC8CE, CC16CE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CC8CE, CC16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CC8CE and CC16CE are, respectively, 8- and 16-bit (stage), asynchronous clearable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low outputs, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

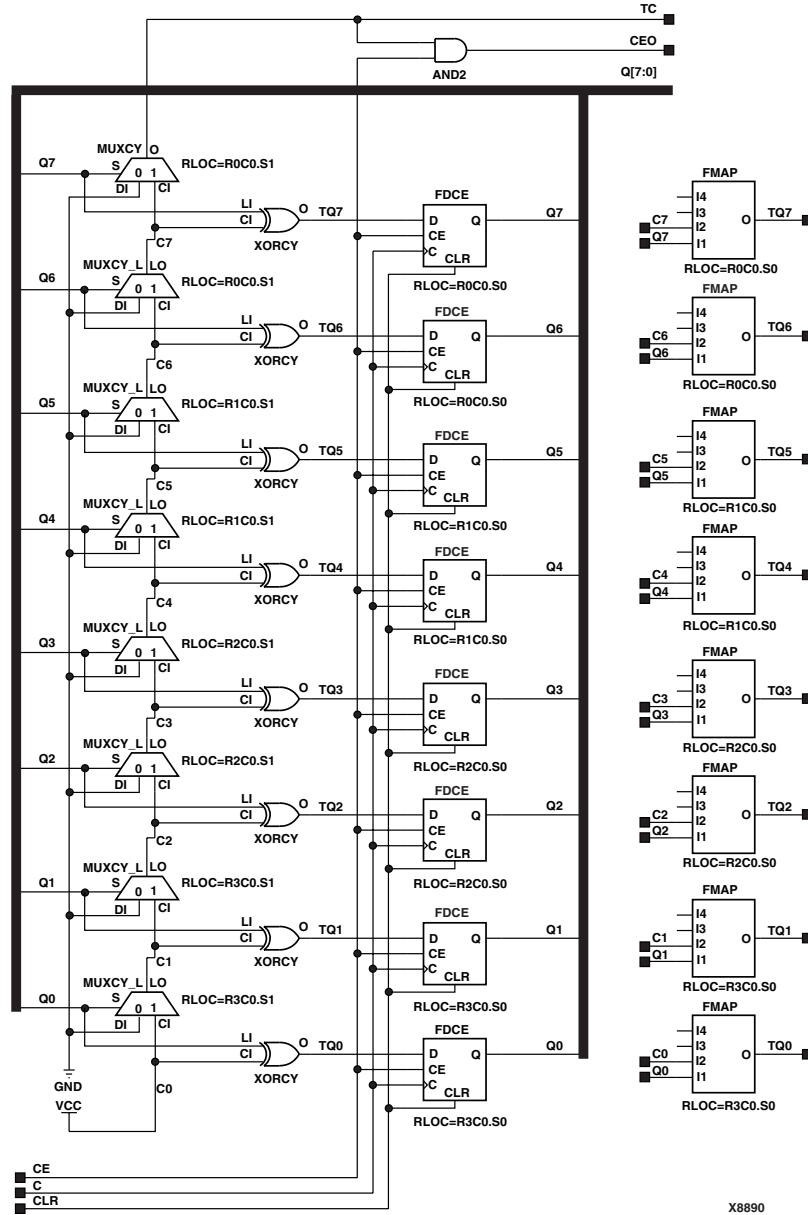
Inputs			Outputs		
CLR	CE	C	Qz – Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No Chg	No Chg	0

Inputs			Outputs		
CLR	CE	C	Qz – Q0	TC	CEO
0	1	↑	Inc	TC	CEO

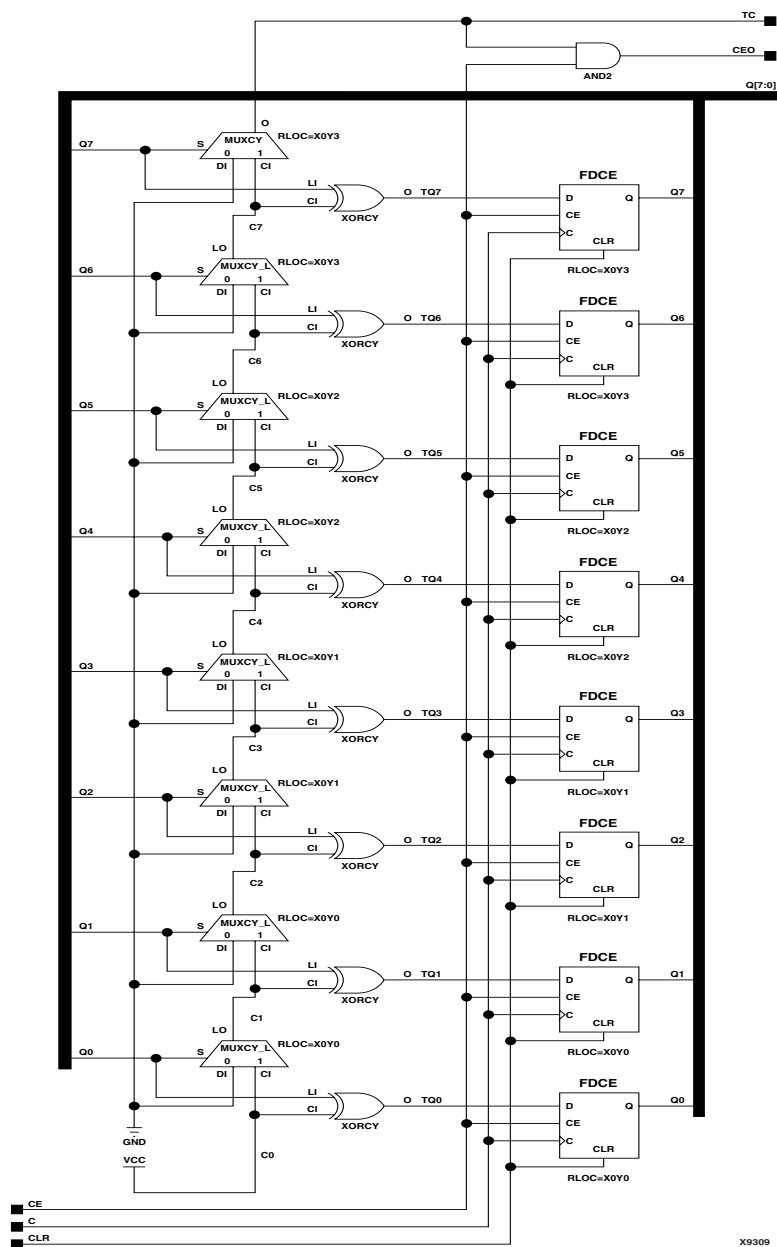
z = 7 for CC8CE; z = 15 for CC16CE

$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEO = TC \cdot CE$$



CC8CE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

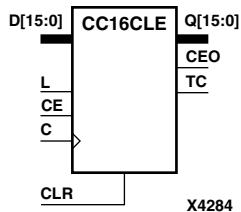
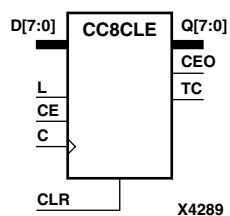
For HDL, these design elements are inferred rather than instantiated.

CC8CLE, CC16CLE

8-, 16-Bit Loadable Cascadable Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CC8CLE, CC16CLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CC8CLE and CC16CLE are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs increment when CE is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the C, L, and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low output, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

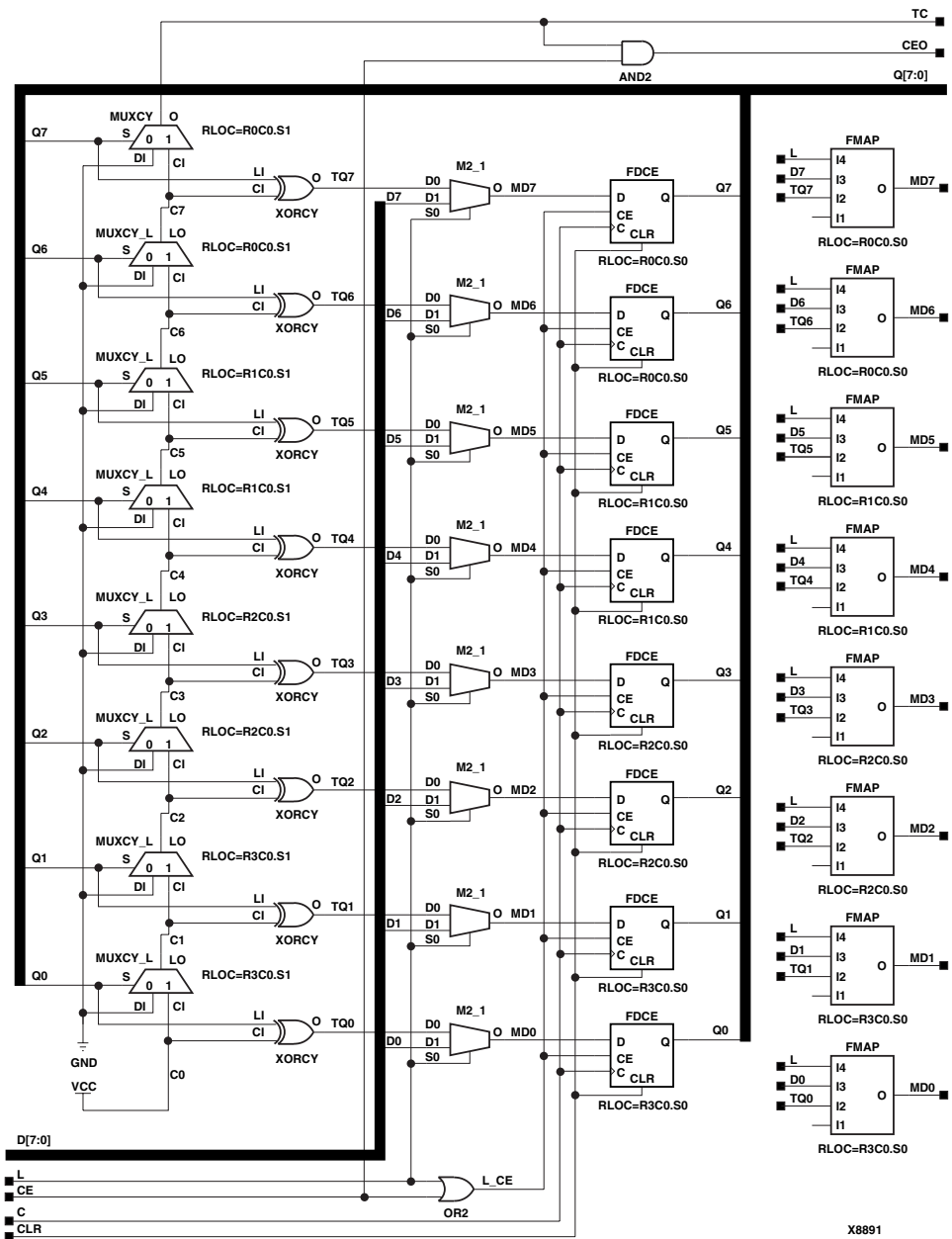
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs		
CLR	L	CE	C	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	0	0	0
0	1	X	↑	Dn	Dn	TC	CEO
0	0	0	X	X	No Chg	No Chg	0
0	0	1	↑	X	Inc	TC	CEO

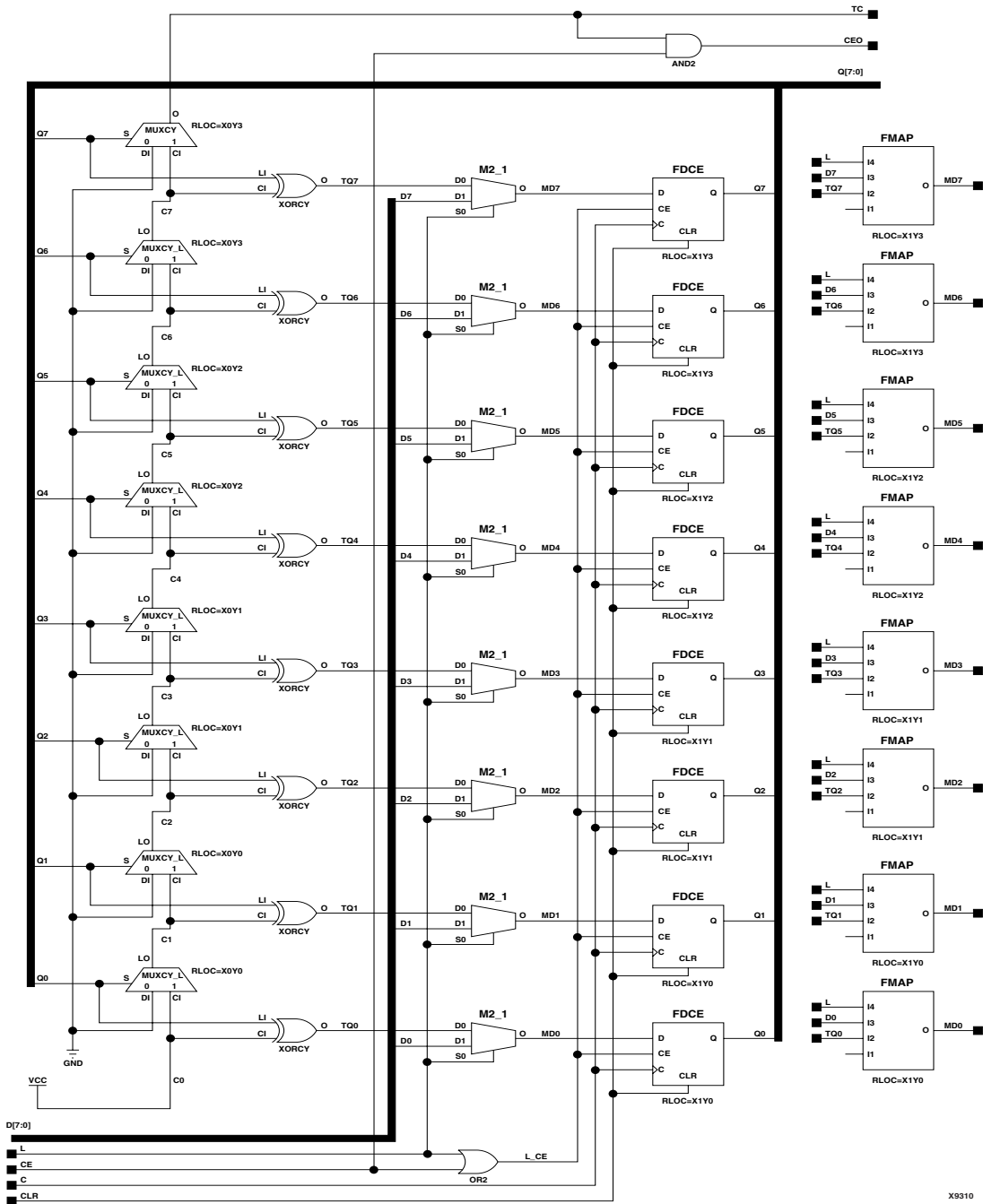
$z = 7$ for CC8CLE; $z = 15$ for CC16CLE

$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$

$CEO = TC \cdot CE$



CC8CLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CLE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

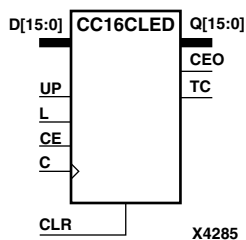
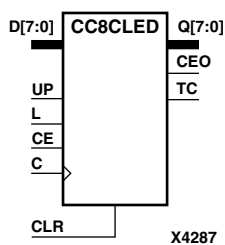
For HDL, these design elements are inferred rather than instantiated.

CC8CLED, CC16CLED

8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CC8CLED, CC16CLED	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CC8CLED and CC16CLED are, respectively, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. These counters are implemented using carry logic with relative location constraints, which assures most efficient logic placement.

The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the count enable out (CEO) output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, outputs Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

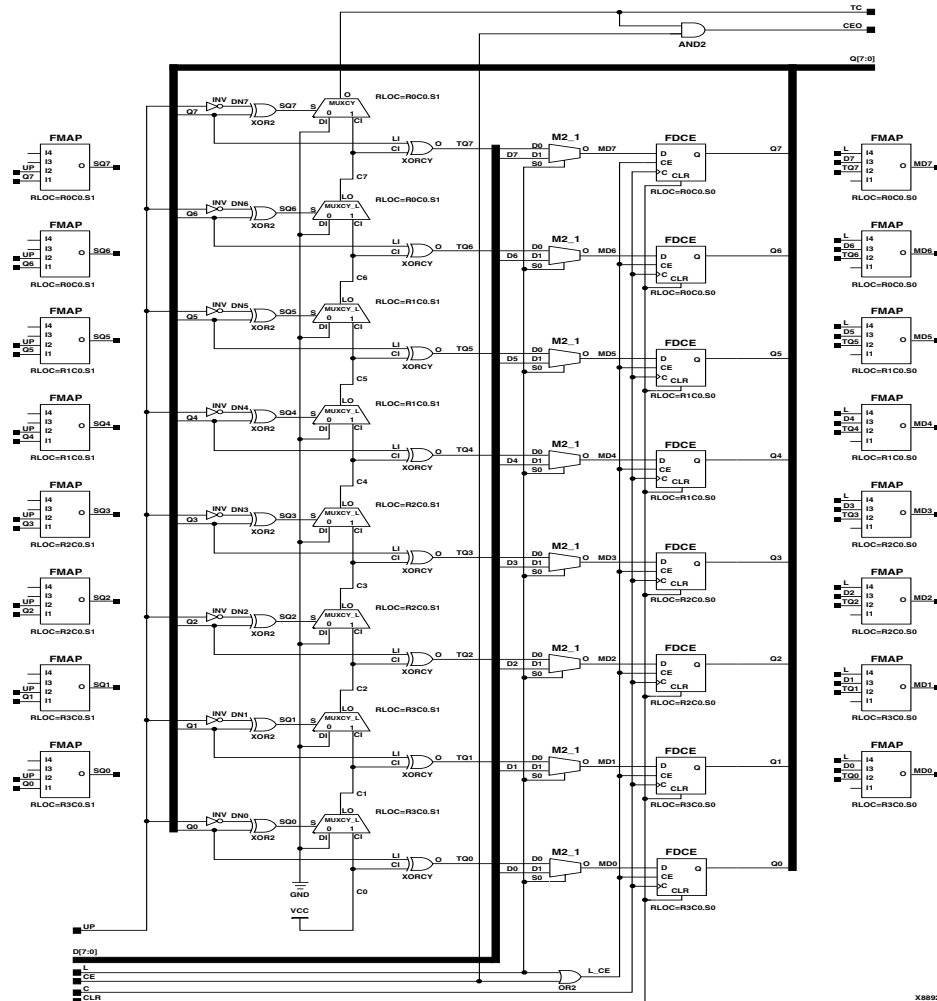
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs						Outputs		
CLR	L	CE	C	UP	Dz - D0	Qz - Q0	TC	CEO
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	Dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

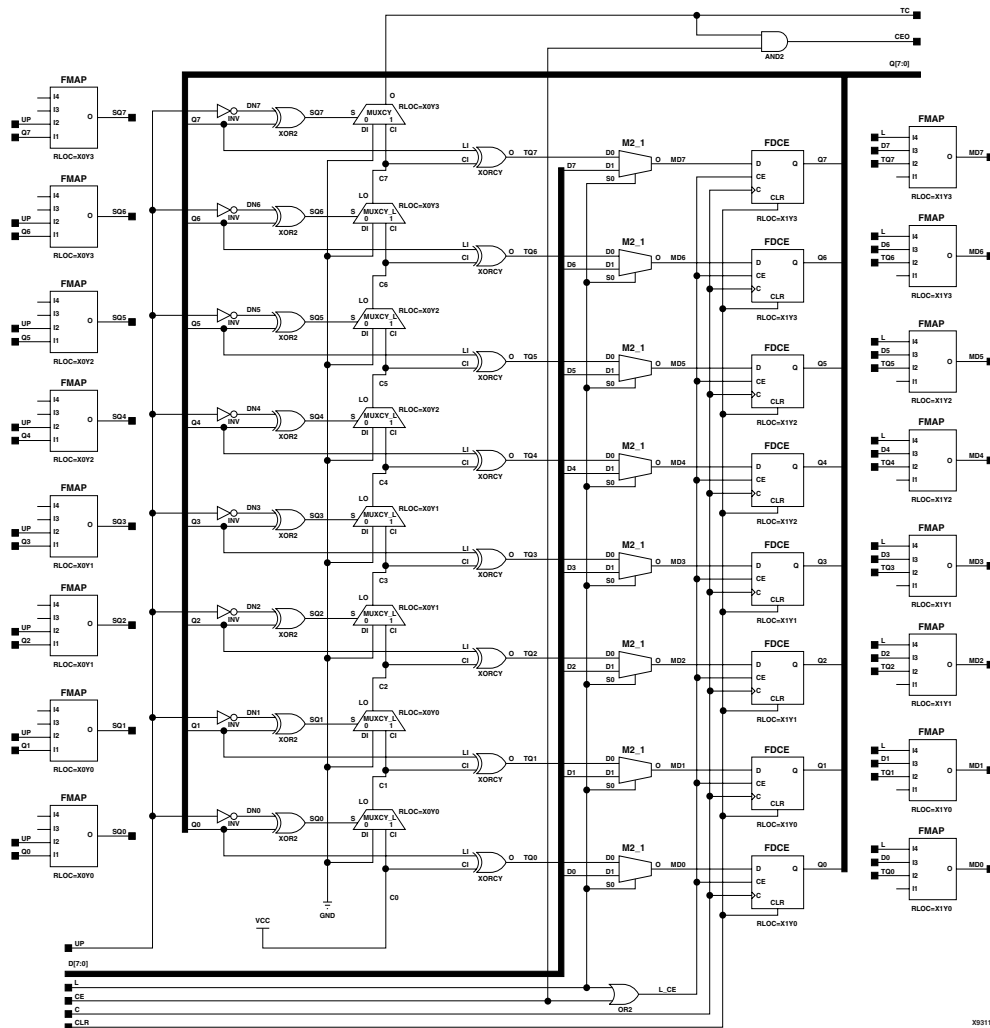
z = 7 for CC8CLED; z = 15 for CC16CLED

$$TC = (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot UP) + (Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0 \cdot \overline{UP})$$

$$CEO = TC \cdot CE$$



CC8CLED Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8CLED Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

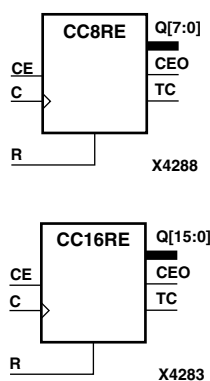
For HDL, these design elements are inferred rather than instantiated.

CC8RE, CC16RE

8-, 16-Bit Cascadable Binary Counters with Clock Enable and Synchronous Reset

Architectures Supported

CC8RE, CC16RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



CC8RE and CC16RE are, respectively, 8- and 16-bit (stage), synchronous resettable, cascadable binary counters. These counters are implemented using carry logic with relative location constraints to ensure efficient placement of logic. The synchronous reset (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs and CE are High.

Larger counters are created by connecting the CEO output of the first stage to the CE input of the next stage and connecting the C and R inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, with Low outputs, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

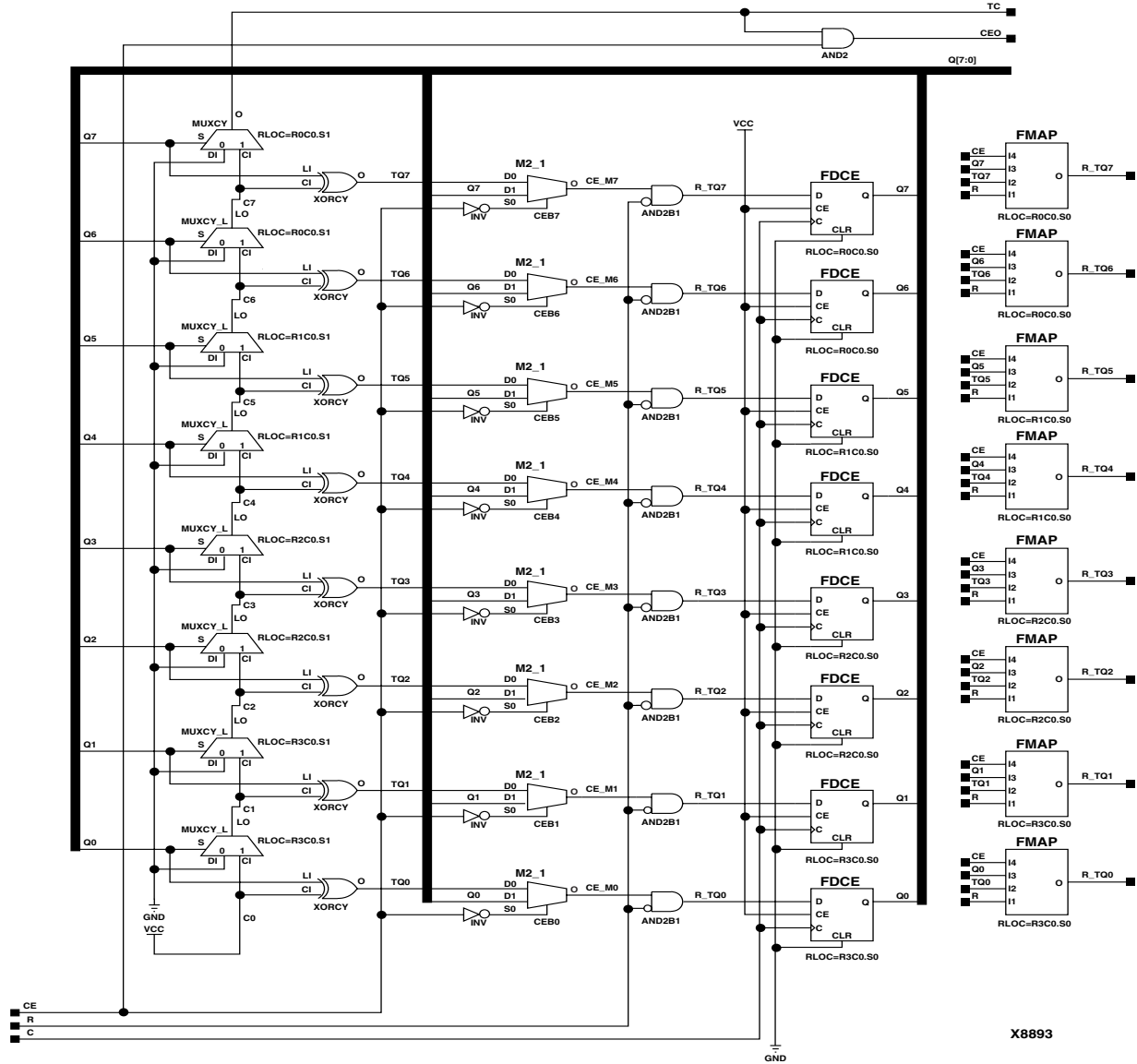
Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
1	X	↑	0	0	0
0	0	X	No Chg	No Chg	0

Inputs			Outputs		
R	CE	C	Qz – Q0	TC	CEO
0	1	↑	Inc	TC	CEO

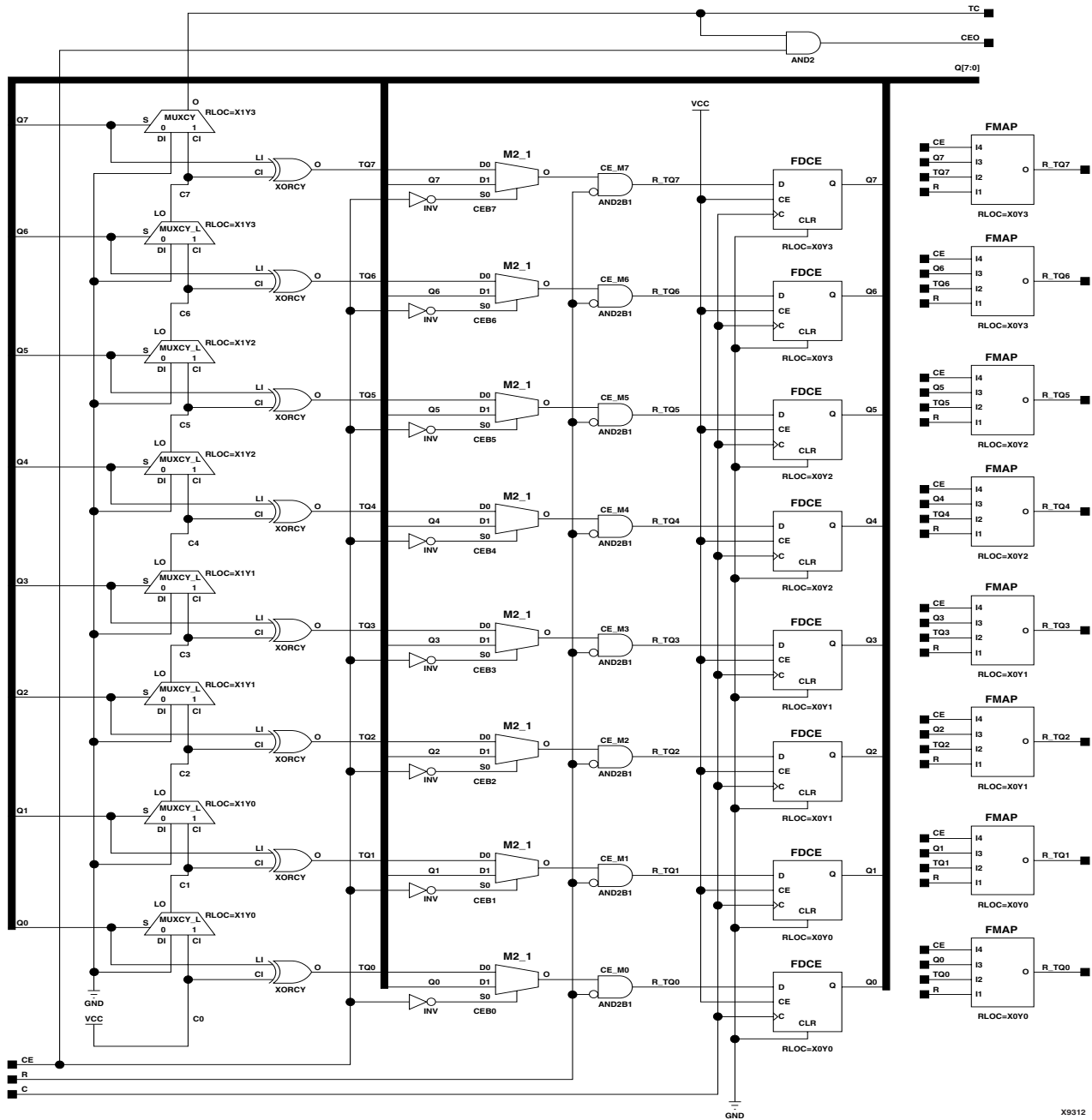
z = 7 for CC8RE; z = 15 for CC16RE

TC = Qz • Q(z-1) • Q(z-2) • ... • Q0 • CE

CEO = TC • CE



CC8RE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



CC8RE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

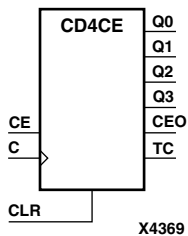
For HDL, these design elements are inferred rather than instantiated.

CD4CE

4-Bit Cascadable BCD Counter with Clock Enable and Asynchronous Clear

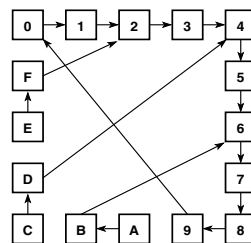
Architectures Supported

CD4CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CD4CE is a 4-bit (stage), asynchronous clearable, cascadable binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when clock enable (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X as shown in the following state diagram. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



X2355

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse to the PRLD global net.

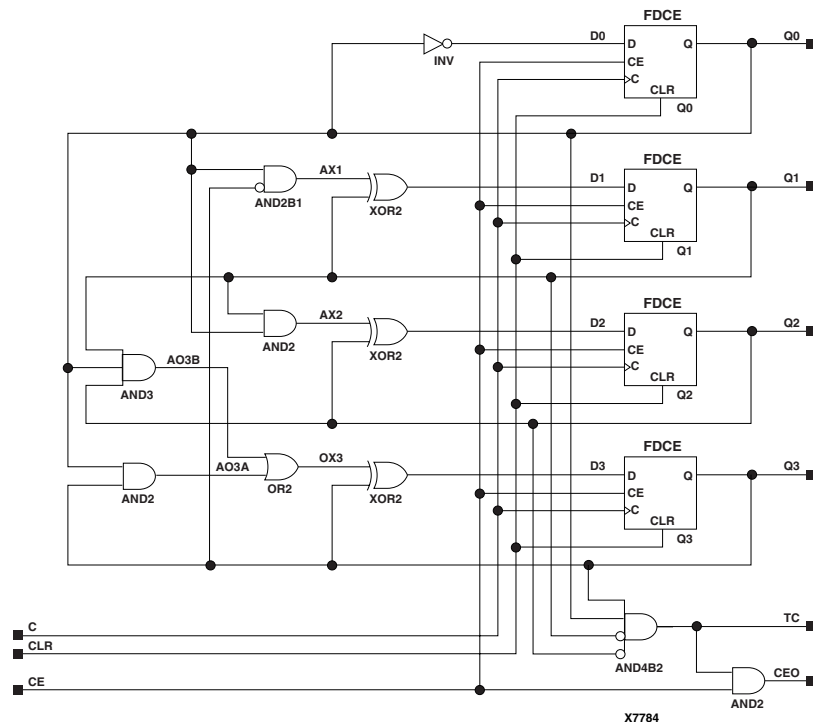
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

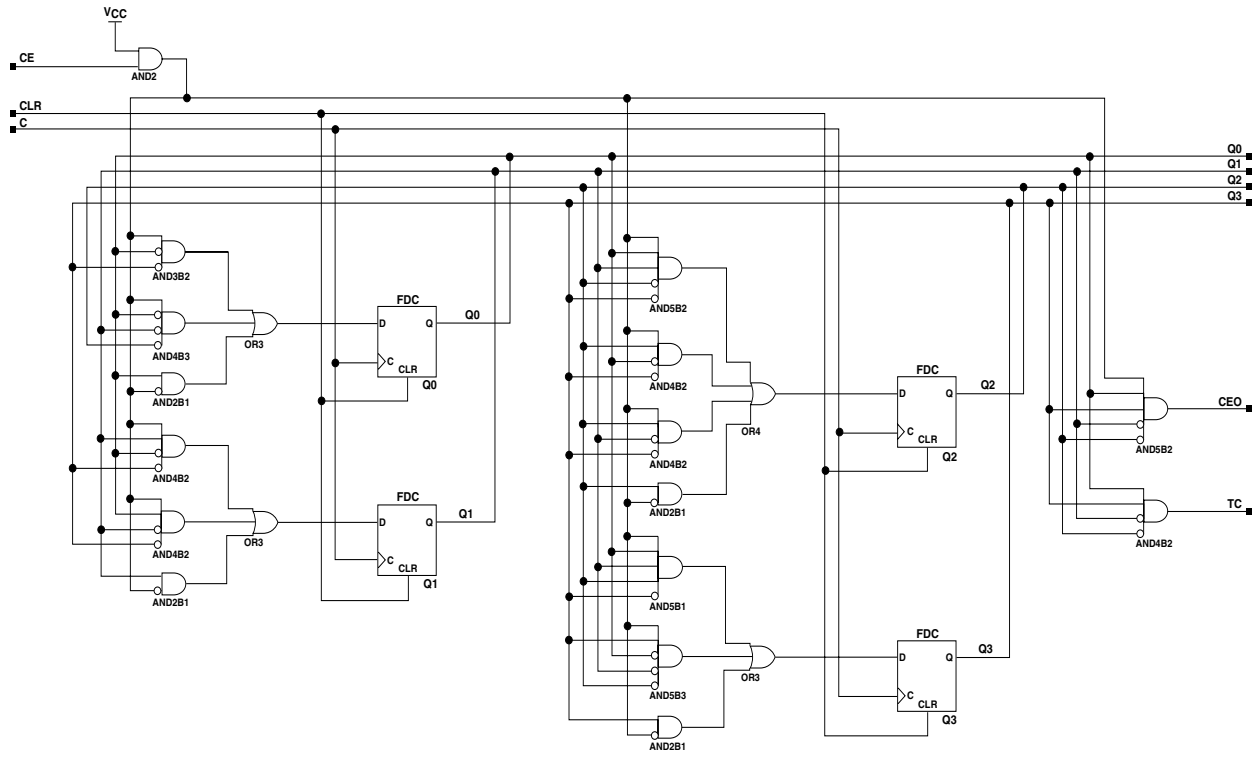
Inputs			Outputs					
CLR	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

$$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$$

$$CEO = TC \cdot CE$$



CD4CE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7629

CD4CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

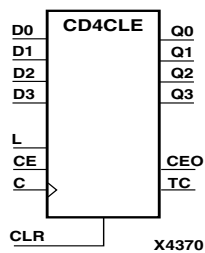
For HDL, this design element can be inferred.

CD4CLE

4-Bit Loadable Cascadable BCD Counter with Clock Enable and Asynchronous Clear

Architectures Supported

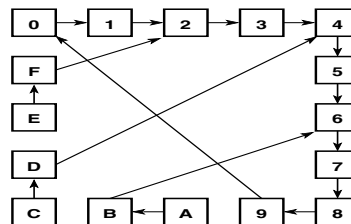
CD4CLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CD4CLE is a 4-bit (stage), synchronously loadable, asynchronously clearable, binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X as shown in the following state diagram.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



X2355

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the

clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

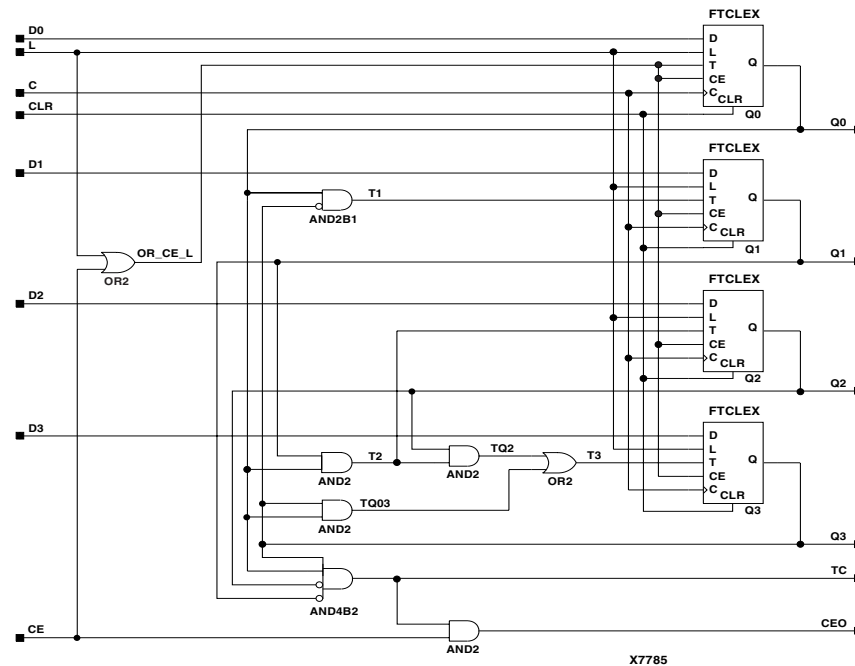
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

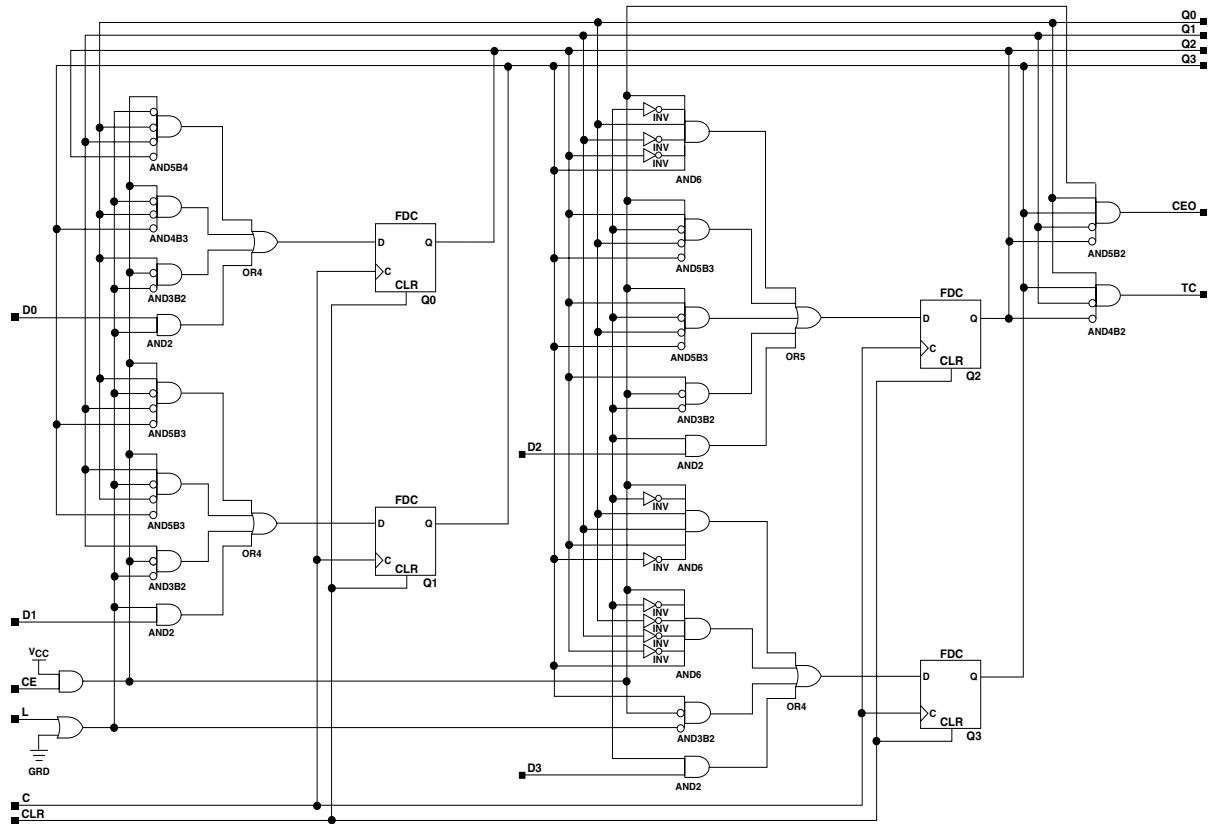
Inputs					Outputs					
CLR	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	X	0	0	0	0	0	0
0	1	X	D3 – D0	↑	D3	D2	D1	D0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

$$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$$

$$CEO = TC \cdot CE$$



CD4CLE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7628

CD4CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

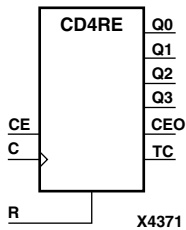
For HDL, these design elements are supported for inference *and* instantiation.

CD4RE

4-Bit Cascadable BCD Counter with Clock Enable and Synchronous Reset

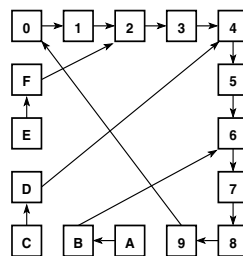
Architectures Supported

CD4RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CD4RE is a 4-bit (stage), synchronous resettable, cascadable binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X as shown in the following state diagram. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



X2355

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When

cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

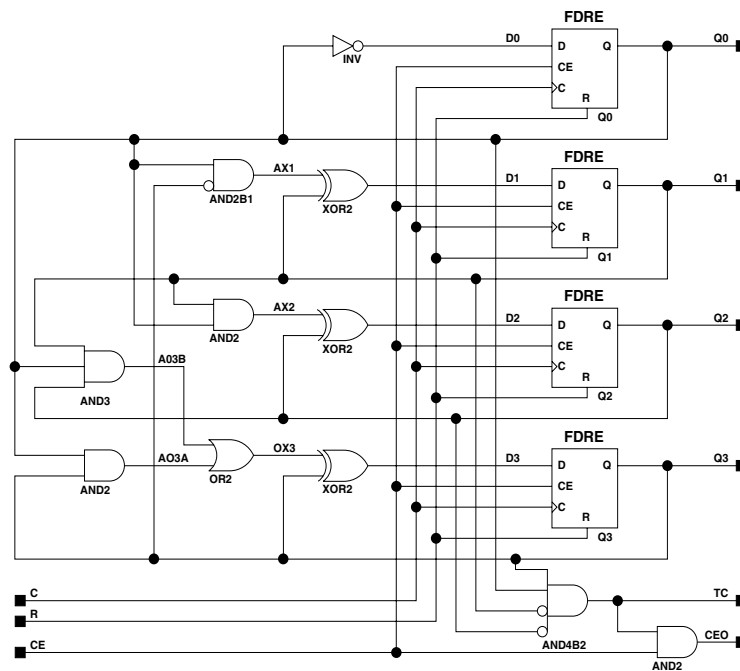
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs					
R	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	↑	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

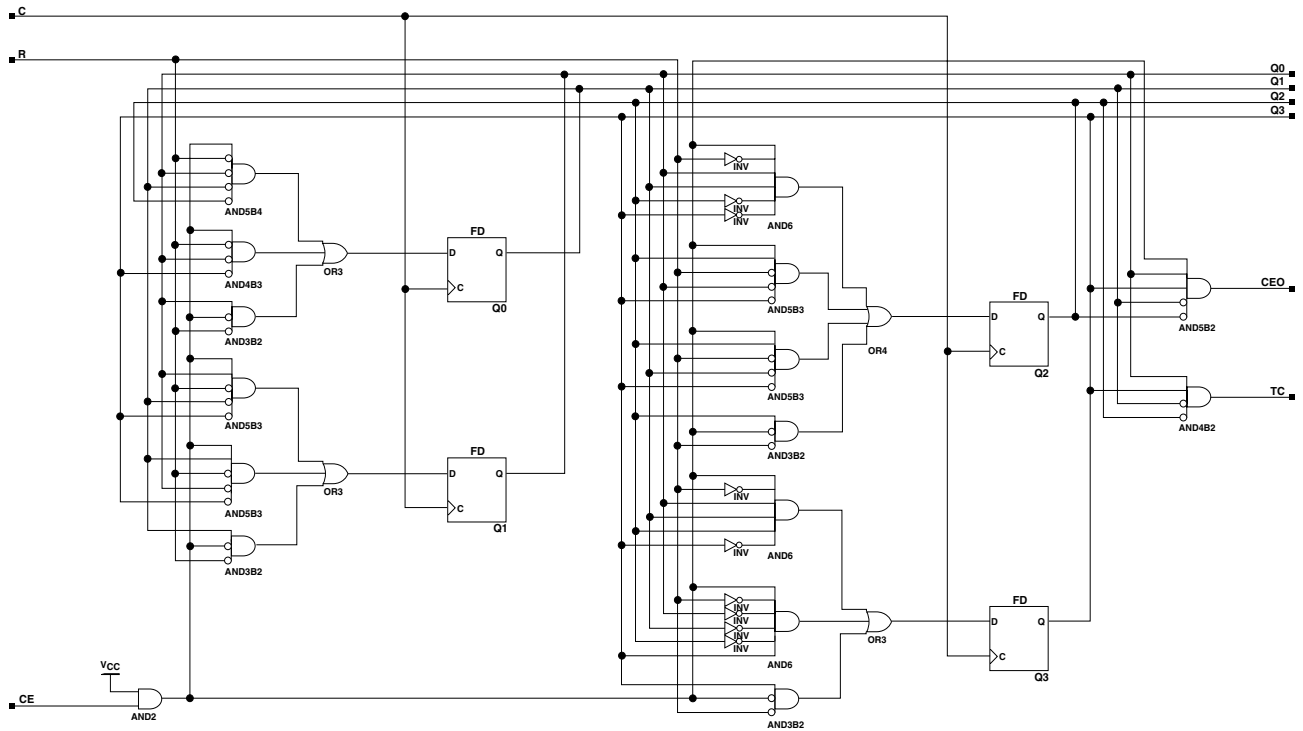
$$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$$

$$CEO = TC \cdot CE$$



X9315

CD4RE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7627

CD4RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

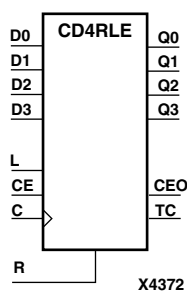
For HDL, this design element can be inferred.

CD4RLE

4-Bit Loadable Cascadable BCD Counter with Clock Enable and Synchronous Reset

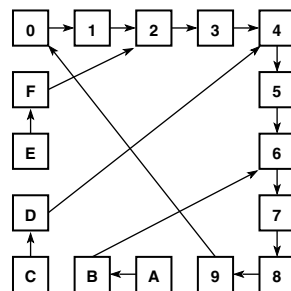
Architectures Supported

CD4RLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CD4RLE is a 4-bit (stage), synchronous loadable, resettable, binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low.

The counter recovers from any of six possible illegal states and returns to a normal count sequence within two clock cycles for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X as shown in the following state diagram. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the counter resets to zero or recovers within the first clock cycle.



X2355

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When

cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

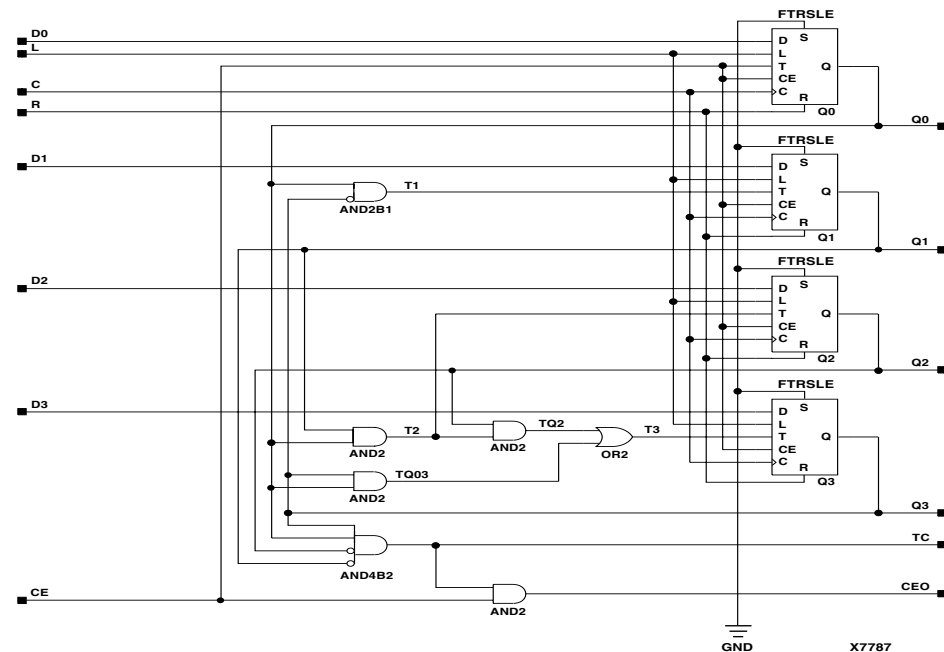
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

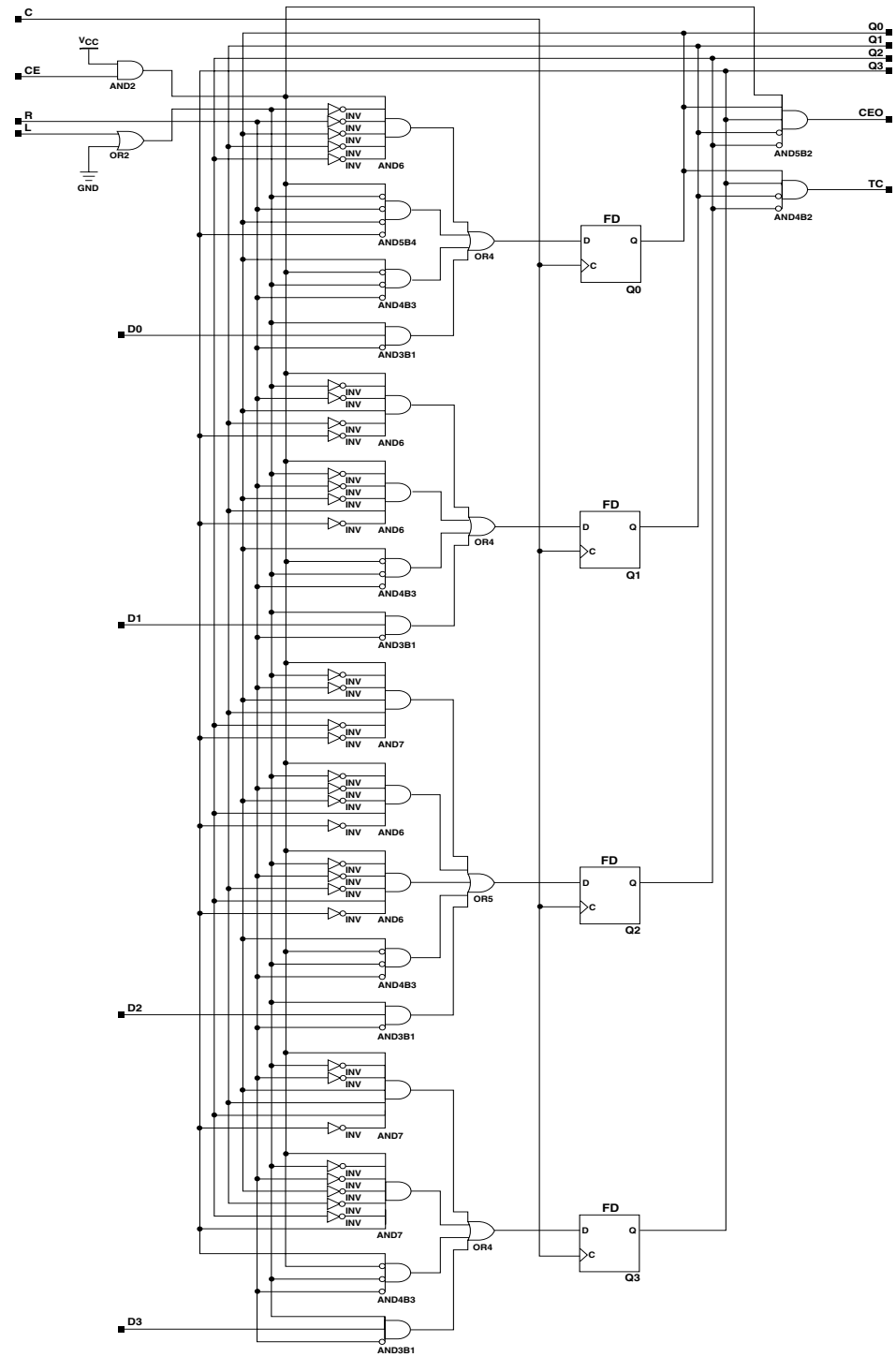
Inputs					Outputs					
R	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	↑	0	0	0	0	0	0
0	1	X	D3 – D0	↑	D3	D	D	D0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

$$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$$

$$CEO = TC \cdot CE$$



CD4RLE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7626

CD4RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

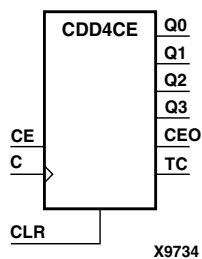
For HDL, this design element is supported for inference *and* instantiation.

CDD4CE

4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear

Architectures Supported

CDD4CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CDD4CE is a 4-bit (stage), asynchronous clearable, cascadable dual edge triggered Binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The Q outputs increment when clock enable (CE) is High during the Low-to-High and High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low. The counter recovers to zero from any illegal state within the first clock cycle.

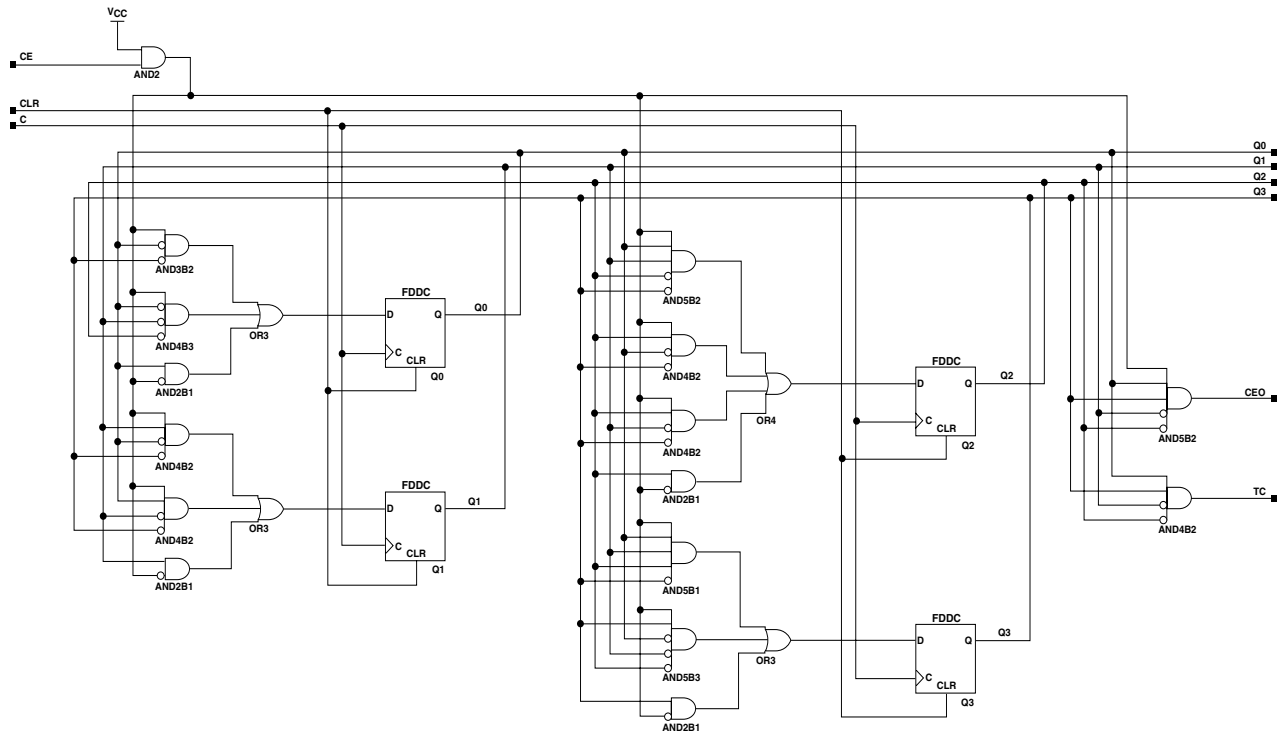
Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse to the PRLD global net.

Inputs			Outputs					
CLR	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	1	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	1	X	1	0	0	1	1	1

$$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$$

$$CEO = TC \cdot CE$$



CDD4CE Implementation CoolRunner-II

Usage

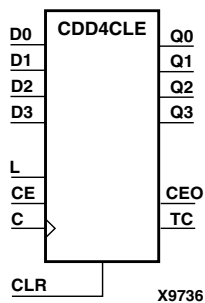
For HDL, this design element is supported for inference *and* instantiation.

CDD4CLE

4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Asynchronous Clear

Architectures Supported

CDD4CLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CDD4CLE is a 4-bit (stage), synchronously loadable, asynchronously clearable, dual edge triggered Binary-coded-decimal (BCD) counter. The asynchronous clear input (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transitions. The Q outputs increment when clock enable input (CE) is High during the Low- to-High clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low. The counter recovers to zero from any illegal state within the first clock cycle.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the CLR, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

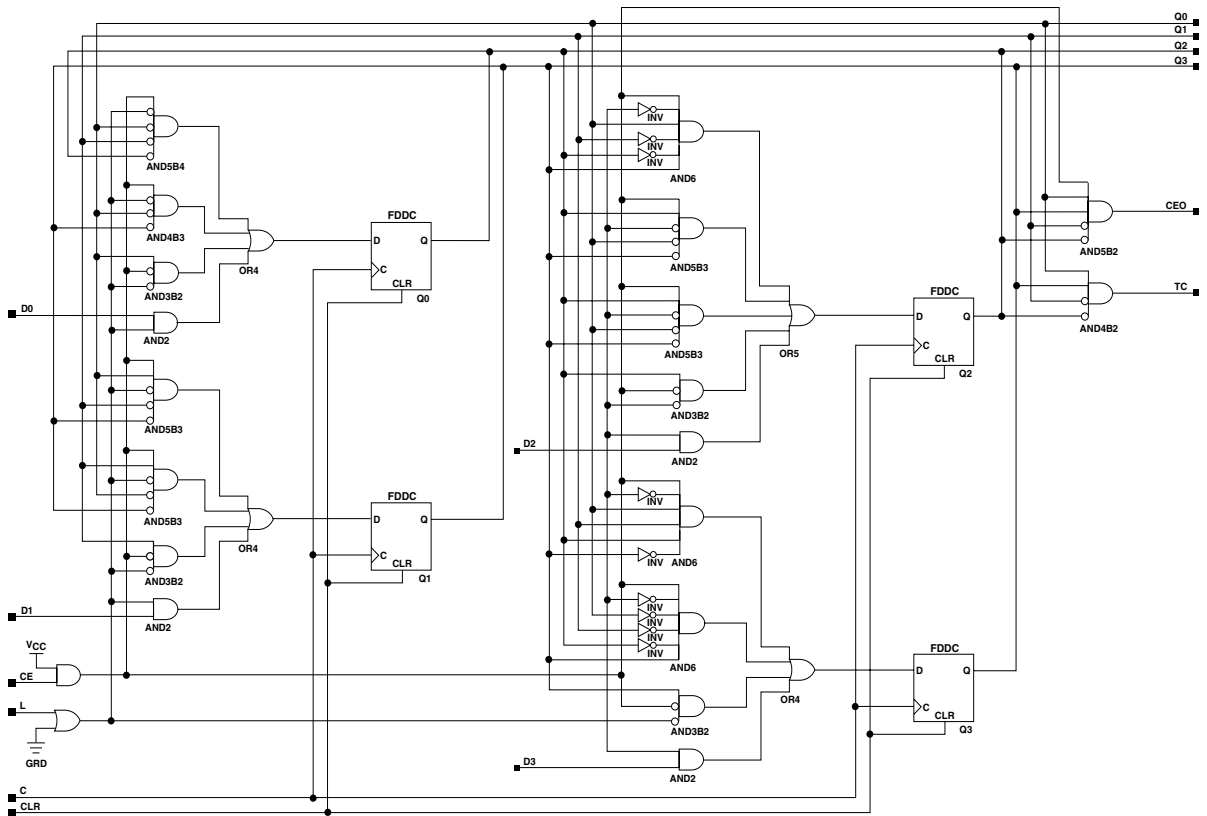
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs					
CLR	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	X	0	0	0	0	0	0
0	1	X	D3 – D0	↑	D3	D2	D1	D0	TC	CEO
0	1	X	D3 – D0	↓	D3	D2	D1	D0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO

Inputs					Outputs					
CLR	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
0	0	1	X	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$

$CEO = TC \cdot CE$



X9737

CDD4CLE Implementation CoolRunner-II

Usage

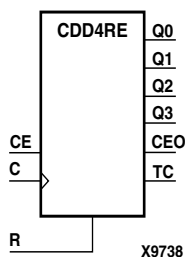
For HDL, this design element is supported for inference *and* instantiation.

CDD4RE

4-Bit Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset

Architectures Supported

CDD4RE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CDD4RE is a 4-bit (stage), synchronous resettable, cascadable dual edge triggered binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High or High-to-Low clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low. The counter recovers to zero from any illegal state within the first clock cycle.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R and clock inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

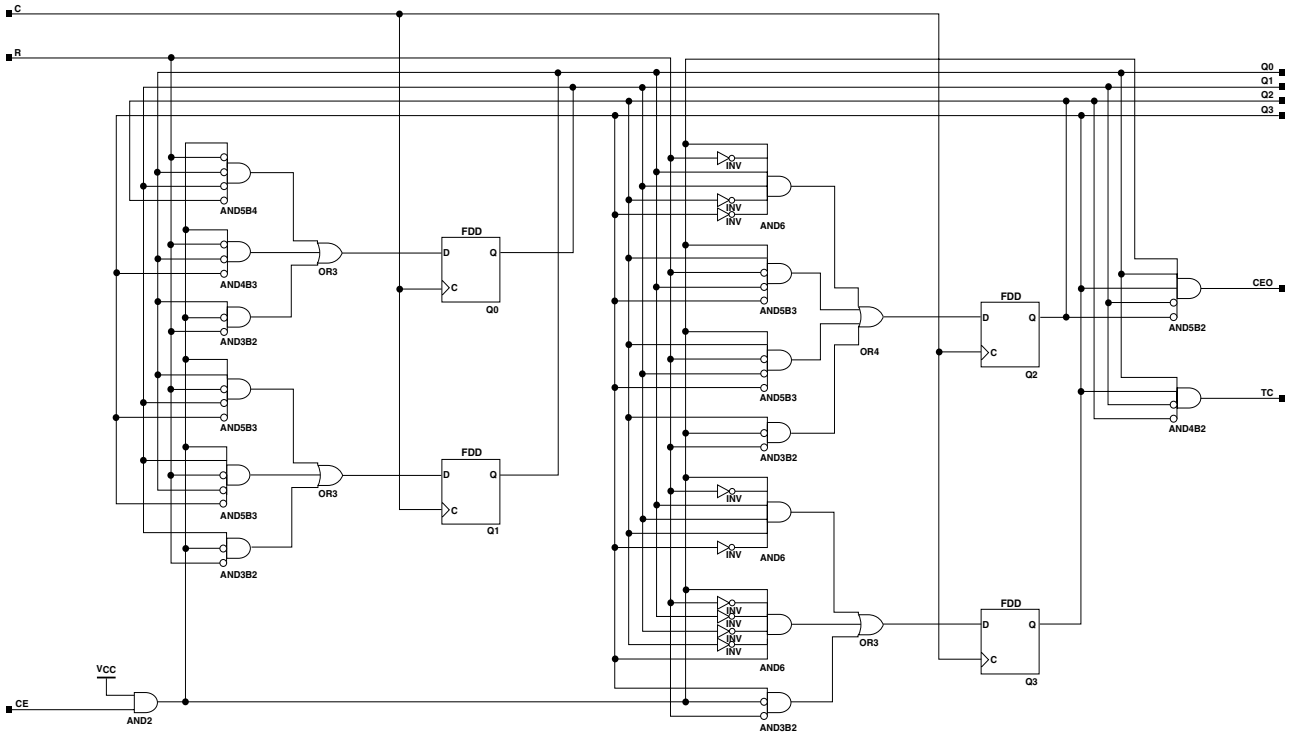
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs					
R	CE	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	↑	0	0	0	0	0	0
1	X	↓	0	0	0	0	0	0
0	1	↑	Inc	Inc	Inc	Inc	TC	CEO
0	1	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	X	No Chg	No Chg	No Chg	No Chg	TC	0

Inputs			Outputs					
R	CE	C	Q3	Q2	Q1	Q0	TC	CEO
0	1	X	1	0	0	1	1	1

$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$

$CEO = TC \cdot CE$



X8739

CDD4RE Implementation CoolRunner-II

Usage

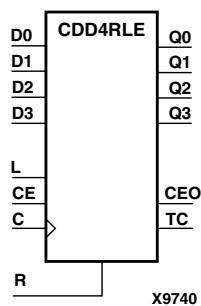
For HDL, this design element can be inferred but not instantiated.

CDD4RLE

4-Bit Loadable Cascadable Dual Edge Triggered BCD Counter with Clock Enable and Synchronous Reset

Architectures Supported

CDD4RLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CDD4RLE is a 4-bit (stage), synchronous loadable, resettable, dual edge triggered binary-coded-decimal (BCD) counter. The synchronous reset input (R) is the highest priority input. When R is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero on the Low-to-High or High-to-Low clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High and High-to-Low clock (C) transition. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. The counter ignores clock transitions when CE is Low. The TC output is High when Q3 and Q0 are High and Q2 and Q1 are Low. The counter recovers to zero from any illegal state within the first clock cycle.

Larger counters are created by connecting the count enable out (CEO) output of the first stage to the CE input of the next stage and connecting the R, L, and C inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE-TC})$, where n is the number of stages and the time t_{CE-TC} is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not.

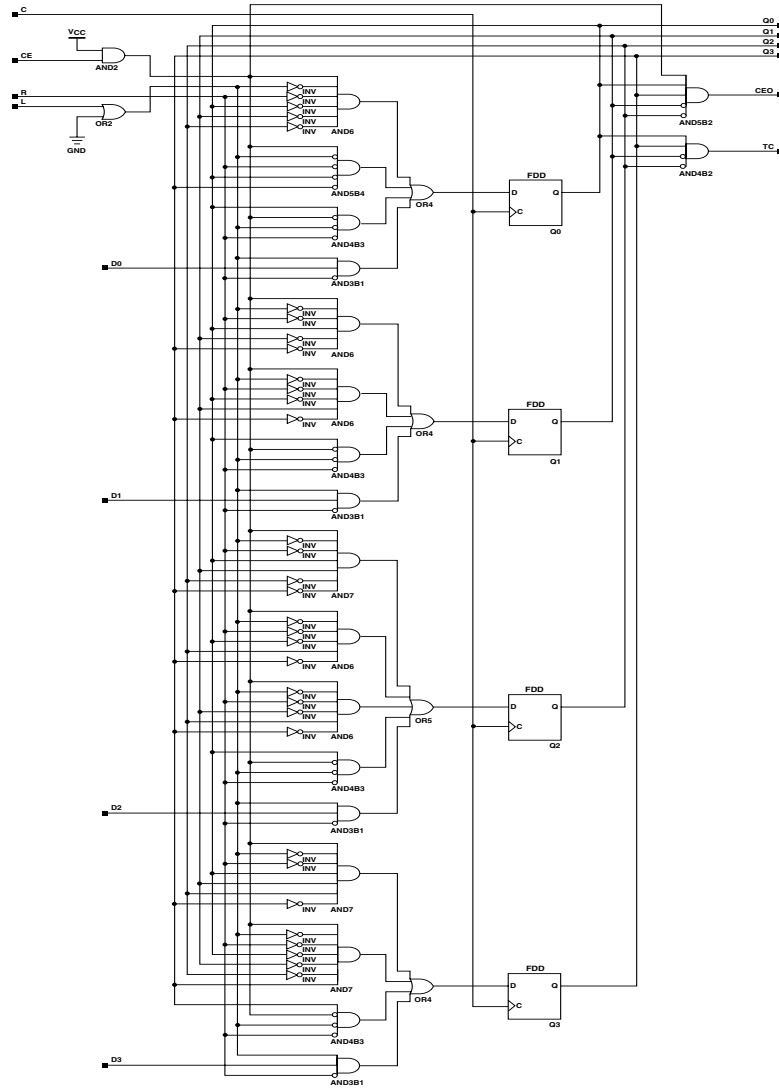
The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs					
R	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
1	X	X	X	↑	0	0	0	0	0	0
1	X	X	X	↓	0	0	0	0	0	0
0	1	X	D3 – D0	↑	D3	D2	D1	D0	TC	CEO
0	1	X	D3 – D0	↓	D3	D2	D1	D0	TC	CEO
0	0	1	X	↑	Inc	Inc	Inc	Inc	TC	CEO

Inputs					Outputs					
R	L	CE	D3 – D0	C	Q3	Q2	Q1	Q0	TC	CEO
0	0	1	X	↓	Inc	Inc	Inc	Inc	TC	CEO
0	0	0	X	X	No Chg	No Chg	No Chg	No Chg	TC	0
0	0	1	X	X	1	0	0	1	1	1

$TC = Q3 \cdot !Q2 \cdot !Q1 \cdot Q0$

$CEO = TC \cdot CE$



CDD4RLE Implementation CoolRunner-II

Usage

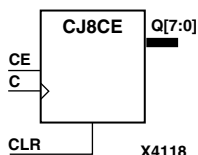
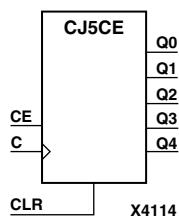
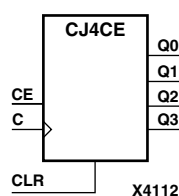
For HDL, this design element can be inferred but not instantiated.

CJ4CE, CJ5CE, CJ8CE

4-, 5-, 8-Bit Johnson Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CJ4CE, CJ5CE, CJ8CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CJ4CE, CJ5CE, and CJ8CE are clearable Johnson/shift counters. The asynchronous clear (CLR) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero, independent of clock (C) transitions. The counter increments (shifts Q0 to Q1, Q1 to Q2, etc.) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

For CJ4CE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operation. For CJ5CE, the Q4 output is inverted and fed back to input Q0. For CJ8CE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

CJ4CE Truth Table

Inputs			Outputs			
CLR	CE	C	Q0	Q1	Q2	Q3
1	X	X	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q_3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJ5CE Truth Table

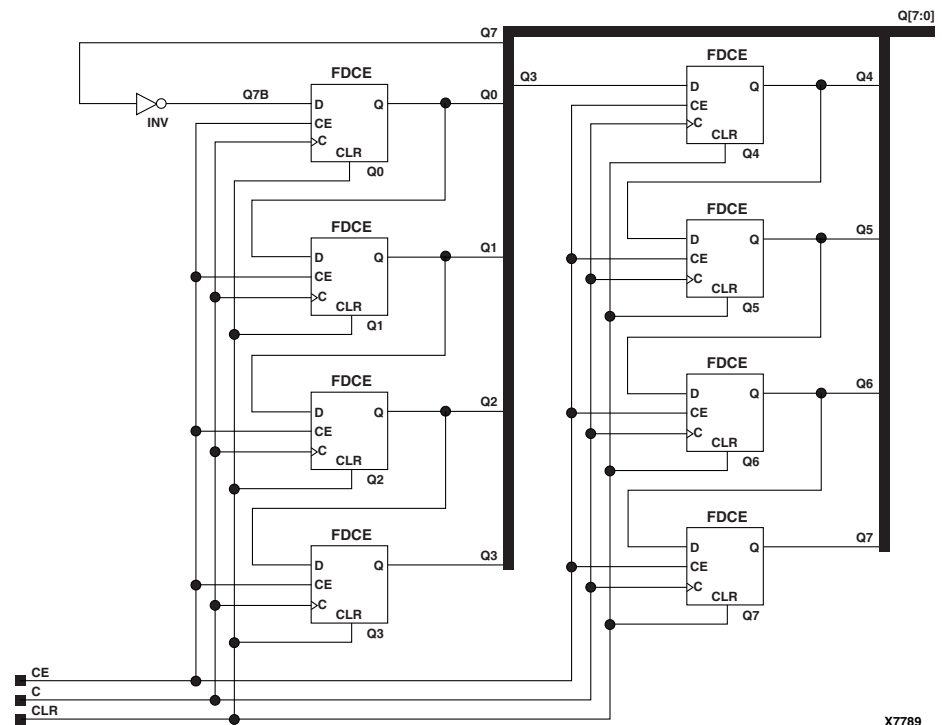
Inputs			Outputs				
CLR	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	X	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q_4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJ8CE Truth Table

Inputs			Outputs	
CLR	CE	C	Q0	Q1 – Q7
1	X	X	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q_7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition



CJ8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

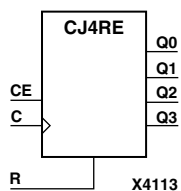
For HDL, this design element can be inferred but not instantiated.

CJ4RE, CJ5RE, CJ8RE

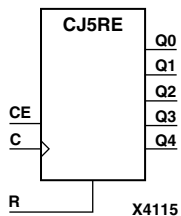
4-, 5-, 8-Bit Johnson Counters with Clock Enable and Synchronous Reset

Architectures Supported

CJ4RE, CJ5RE, CJ8RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CJ4RE, CJ5RE, and CJ8RE are resettable Johnson/shift counters. The synchronous reset (R) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero during the Low-to-High clock (C) transition. The counter increments (shifts Q0 to Q1, Q1 to Q2, etc.) when the clock enable input (CE) is High during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.



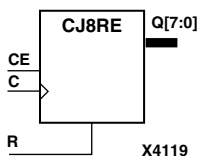
For CJ4RE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJ5RE, the Q4 output is inverted and fed back to input Q0. For CJ8RE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



CJ4RE Truth Table

Inputs			Outputs			
R	CE	C	Q0	Q1	Q2	Q3
1	X	↑	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q_3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJ5RE Truth Table

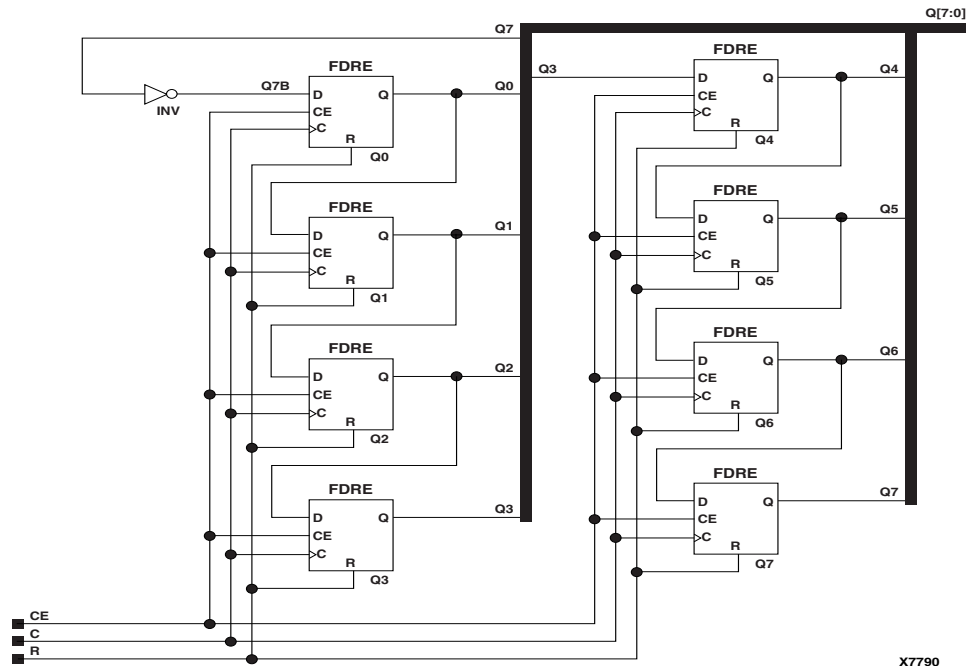
Inputs			Outputs				
R	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	↑	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q_4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJ8RE Truth Table

Inputs			Outputs	
R	CE	C	Q0	Q1 – Q7
1	X	↑	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q_7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition



CJ8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

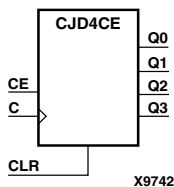
For HDL, this design element can be inferred but not instantiated.

CJD4CE, CJD5CE, CJD8CE

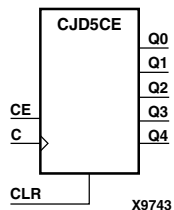
4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CJD4CE, CJD5CE, CJD8CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive

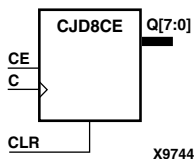


CJD4CE, CJD5CE, and CJD8CE are dual edge triggered clearable Johnson/shift counters. The asynchronous clear (CLR) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero, independent of clock (C) transitions. The counter increments (shifts Q0 to Q1, Q1 to Q2, etc.) when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. Clock transitions are ignored when CE is Low.



For CJD4CE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJD5CE, the Q4 output is inverted and fed back to input Q0. For CJD8CE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



CJD4CE Truth Table

Inputs			Outputs			
CLR	CE	C	Q0	Q1	Q2	Q3
1	X	X	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q3	q0	q1	q2
0	1	↓	!q3	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJD5CE Truth Table

Inputs			Outputs				
CLR	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	X	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	!q4	q0	q1	q2	q3
0	1	↓	!q4	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJD8CE Truth Table

Inputs			Outputs	
CLR	CE	C	Q0	Q1 – Q7
1	X	X	0	0
0	0	X	No Chg	No Chg
0	1	↑	!q7	q0 – q6
0	1	↓	!q7	q0 – q6

q = state of referenced output one setup time prior to active clock transition

Usage

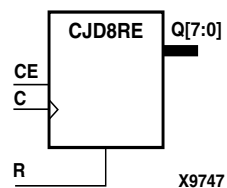
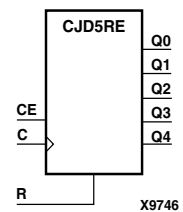
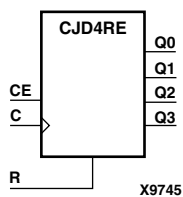
For HDL, this design element can be inferred but not instantiated.

CJD4RE, CJD5RE, CJD8RE

4-, 5-, 8-Bit Dual Edge Triggered Johnson Counters with Clock Enable and Synchronous Reset

Architectures Supported

CJD4RE, CJD5RE, CJD8RE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



CJD4RE, CJD5RE, and CJD8RE are resettable dual edge triggered Johnson/shift counters. The synchronous reset (R) input, when High, overrides all other inputs and causes the data (Q) outputs to go to logic level zero during the Low-to-High and High-to-Low clock (C) transition. The counter increments (shifts Q0 to Q1, Q1 to Q2, etc.) when the clock enable input (CE) is High during the Low-to-High and High-to-Low clock transition. Clock transitions are ignored when CE is Low.

For CJD4RE, the Q3 output is inverted and fed back to input Q0 to provide continuous counting operations. For CJD5RE, the Q4 output is inverted and fed back to input Q0. For CJD8RE, the Q7 output is inverted and fed back to input Q0.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

CJD4RE Truth Table

Inputs			Outputs			
R	CE	C	Q0	Q1	Q2	Q3
1	X	↑	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q_3}$	q0	q1	q2
0	1	↓	$\overline{q_3}$	q0	q1	q2

q = state of referenced output one setup time prior to active clock transition

CJD5RE Truth Table

Inputs			Outputs				
R	CE	C	Q0	Q1	Q2	Q3	Q4
1	X	↑	0	0	0	0	0
0	0	X	No Chg	No Chg	No Chg	No Chg	No Chg
0	1	↑	$\overline{q4}$	q0	q1	q2	q3
0	1	↓	$\overline{q4}$	q0	q1	q2	q3

q = state of referenced output one setup time prior to active clock transition

CJD8RE Truth Table

Inputs			Outputs	
R	CE	C	Q0	Q1 – Q7
1	X	↑	0	0
1	X	↓	0	0
0	0	X	No Chg	No Chg
0	1	↑	$\overline{q7}$	q0 – q6
0	1	↓	$\overline{q7}$	q0 – q6

q = state of referenced output one setup time prior to active clock transition

Usage

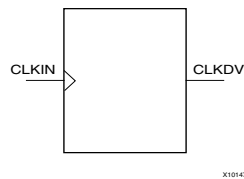
For HDL, this design element can be inferred but not instantiated.

CLK_DIV2,4,6,8,10,12,14,16

Global Clock Divider

Architectures Supported

CLK_DIV2	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive
CLK_DIV4, CLK_DIV6, CLK_DIV8, CLK_DIV10, CLK_DIV12, CLK_DIV14, CLK_DIV16	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



The CLK_DIV2,4,6,8,10,12,14,16 Global Clock Dividers divide a user-provided external clock signal `gclk<2>` by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN input can only be connected to the device `gclk<2>` pin. The duty cycle of the CLKDV output is 50-50.

The CLKDV output is reset low by power-on reset circuitry.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Templates

```

CLK_DIV2
-- CLK_DIV2: Simple clock Divide by 2
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV2_inst : CLK_DIV2
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input

```

```
);

CLK_DIV4
-- CLK_DIV4: Simple clock Divide by 4
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV4_inst : CLK_DIV4
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV6
-- CLK_DIV6: Simple clock Divide by 6
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV6_inst : CLK_DIV6
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV8
-- CLK_DIV8: Simple clock Divide by 8
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV8_inst : CLK_DIV8
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV10
-- CLK_DIV10: Simple clock Divide by 10
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV10_inst : CLK_DIV10
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV12
-- CLK_DIV12: Simple clock Divide by 12
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV12_inst : CLK_DIV12
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);
```



```

CLK_DIV14
-- CLK_DIV14: Simple clock Divide by 14
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV14_inst : CLK_DIV14
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV16
-- CLK_DIV16: Simple clock Divide by 16
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV16_inst : CLK_DIV16
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

```

Verilog Instantiation Templates

```

CLK_DIV2
// CLK_DIV2: Simple clock Divide by 2
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV2 CLK_DIV2_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV4
// CLK_DIV4: Simple clock Divide by 4
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV4 CLK_DIV4_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV6
// CLK_DIV6: Simple clock Divide by 6
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV6 CLK_DIV6_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV8
// CLK_DIV8: Simple clock Divide by 8

```

```
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV8 CLK_DIV8_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV10
// CLK_DIV10: Simple clock Divide by 10
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV10 CLK_DIV10_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV12
// CLK_DIV12: Simple clock Divide by 12
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV12 CLK_DIV12_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV14
// CLK_DIV14: Simple clock Divide by 14
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV14 CLK_DIV14_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);

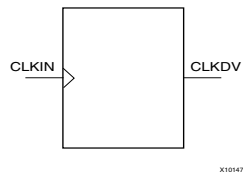
CLK_DIV16
// CLK_DIV16: Simple clock Divide by 16
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
CLK_DIV16 CLK_DIV16_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CLKIN(CLKIN)    // Clock input
);
```

CLK_DIV2,4,6,8,10,12,14,16R

Global Clock Divider with Synchronous Reset

Architectures Supported

CLK_DIV2R	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive
CLK_DIV4R, CLK_DIV6R, CLK_DIV8R, CLK_DIV10R, CLK_DIV12R, CLK_DIV14R, CLK_DIV16R	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



The CLK_DIV2,4,6,8,10,12,14,16R Global Clock Dividers with Synchronous Reset divide a user-provided external clock signal `gclk<2>` by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN and CDRST inputs can only be connected to the device `gclk<2>` and CDRST pins. The duty cycle of the CLKDV output is 50-50.

The CDRST input is an active High synchronous reset. If CDRST is input High when the CLKDV output is High, the CLKDV output remains High to complete the last clock pulse, and then goes Low.

The CLKDV output is reset low by power-on reset circuitry.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
CLK_DIV2R
-- CLK_DIV2R: Clock Divide by 2 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide
```

```
CLK_DIV2R_inst : CLK_DIV2R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV4R
-- CLK_DIV4R: Clock Divide by 4 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV4R_inst : CLK_DIV4R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV6R
-- CLK_DIV6R: Clock Divide by 6 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV6R_inst : CLK_DIV6R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV8R
-- CLK_DIV8R: Clock Divide by 8 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV8R_inst : CLK_DIV8R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV10R
-- CLK_DIV10R: Clock Divide by 10 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV10R_inst : CLK_DIV10R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);
```

```

CLK_DIV12R
-- CLK_DIV12R: Clock Divide by 12 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV12R_inst : CLK_DIV12R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV14R
-- CLK_DIV14R: Clock Divide by 14 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV14R_inst : CLK_DIV14R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

CLK_DIV16R
-- CLK_DIV16R: Clock Divide by 16 with synchronous reset
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV16R_inst : CLK_DIV16R
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,    -- Synchronous reset input
    CLKIN => CLKIN     -- Clock input
);

```

Verilog Instantiation Template

```

CLK_DIV2R
// CLK_DIV2R: Clock Divide by 2 with synchronous reset
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV2R CLK_DIV2R_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CDRST(CDRST),   // Synchronous reset input
    .CLKIN(CLKIN)    // Clock input
);

CLK_DIV4R
// CLK_DIV4R: Clock Divide by 4 with synchronous reset
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV4R CLK_DIV4R_inst (

```

```
.CLKDV(CLKDV), // Divided clock output
.CDRST(CDRST), // Synchronous reset input
.CLKIN(CLKIN) // Clock input
);

CLK_DIV6R
// CLK_DIV6R: Clock Divide by 6 with synchronous reset
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV6R CLK_DIV6R_inst (
.CLKDV(CLKDV), // Divided clock output
.CDRST(CDRST), // Synchronous reset input
.CLKIN(CLKIN) // Clock input
);

CLK_DIV8R
// CLK_DIV8R: Clock Divide by 8 with synchronous reset
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV8R CLK_DIV8R_inst (
.CLKDV(CLKDV), // Divided clock output
.CDRST(CDRST), // Synchronous reset input
.CLKIN(CLKIN) // Clock input
);

CLK_DIV10R
// CLK_DIV10R: Clock Divide by 10 with synchronous reset
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV10R CLK_DIV10R_inst (
.CLKDV(CLKDV), // Divided clock output
.CDRST(CDRST), // Synchronous reset input
.CLKIN(CLKIN) // Clock input
);

CLK_DIV12R
// CLK_DIV12R: Clock Divide by 12 with synchronous reset
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV12R CLK_DIV12R_inst (
.CLKDV(CLKDV), // Divided clock output
.CDRST(CDRST), // Synchronous reset input
.CLKIN(CLKIN) // Clock input
);

CLK_DIV14R
// CLK_DIV14R: Clock Divide by 14 with synchronous reset
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV14R CLK_DIV14R_inst (
.CLKDV(CLKDV), // Divided clock output
```

```
.CDRST(CDRST),    // Synchronous reset input
.CLKIN(CLKIN)     // Clock input
);

CLK_DIV16R
// CLK_DIV16R: Clock Divide by 16 with synchronous reset
//              CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

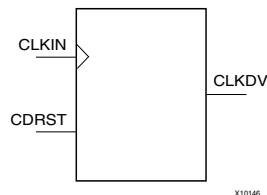
CLK_DIV16R CLK_DIV16R_inst (
    .CLKDV(CLKDV),    // Divided clock output
    .CDRST(CDRST),   // Synchronous reset input
    .CLKIN(CLKIN)    // Clock input
);
```


CLK_DIV2,4,6,8,10,12,14,16RSD

Global Clock Divider with Synchronous Reset and Start Delay

Architectures Supported

CLK_DIV2RSD	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive
CLK_DIV4RSD, CLK_DIV6RSD, CLK_DIV8RSD, CLK_DIV10RSD, CLK_DIV12RSD, CLK_DIV14RSD, CLK_DIV16RSD	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



The CLK_DIV2,4,6,8,10,12,14,16 Global Clock Dividers with Synchronous Reset and Start Delay divide a user-provided external clock signal `gclk<2>` by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN and CDRST inputs can only be connected to the device `gclk<2>` and CDRST pins. The duty cycle of the CLKDV output is 50-50.

The CDRST input is an active High synchronous reset. If CDRST is input High when the CLKDV output is High, the CLKDV output remains High to complete the last clock pulse, and then goes Low.

The start delay function delays the start of the CLKDV output by $(n + 1)$ clocks, where n is the divisor for the clock divider.

The CLKDV output is reset low by power-on reset circuitry.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
CLK_DIV2RSD
-- CLK_DIV2RSD: Clock Divide by 2 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV2RSD_inst : CLK_DIV2RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,   -- Synchronous reset input
    CLKIN => CLKIN    -- Clock input
);

CLK_DIV4RSD
-- CLK_DIV4RSD: Clock Divide by 4 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV4RSD_inst : CLK_DIV4RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,   -- Synchronous reset input
    CLKIN => CLKIN    -- Clock input
);

CLK_DIV6RSD
-- CLK_DIV6RSD: Clock Divide by 6 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV6RSD_inst : CLK_DIV6RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CDRST => CDRST,   -- Synchronous reset input
    CLKIN => CLKIN    -- Clock input
);

CLK_DIV8RSD
-- CLK_DIV8RSD: Clock Divide by 8 with synchronous reset and start
-- delay
```

```
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV8RSD_inst : CLK_DIV8RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CDRST => CDRST,    -- Synchronous reset input
  CLKIN => CLKIN     -- Clock input
);

CLK_DIV10RSD
-- CLK_DIV10RSD: Clock Divide by 10 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV10RSD_inst : CLK_DIV10RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CDRST => CDRST,    -- Synchronous reset input
  CLKIN => CLKIN     -- Clock input
);

CLK_DIV12RSD
-- CLK_DIV12RSD: Clock Divide by 12 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV12RSD_inst : CLK_DIV12RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CDRST => CDRST,    -- Synchronous reset input
  CLKIN => CLKIN     -- Clock input
);

CLK_DIV14RSD
-- CLK_DIV14RSD: Clock Divide by 14 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV14RSD_inst : CLK_DIV14RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
```

```

generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,      -- Divided clock output
    CDRST => CDRST,     -- Synchronous reset input
    CLKIN => CLKIN      -- Clock input
);

CLK_DIV16RSD
-- CLK_DIV16RSD: Clock Divide by 16 with synchronous reset and start
-- delay
--          CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV16RSD_inst : CLK_DIV16RSD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,      -- Divided clock output
    CDRST => CDRST,     -- Synchronous reset input
    CLKIN => CLKIN      -- Clock input
);

```

Verilog Instantiation Template

```

CLK_DIV2RSD
// CLK_DIV2RSD: Clock Divide by 2 with synchronous reset and start
// delay
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV2RSD CLK_DIV2RSD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV2RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV2RSD_inst instantiation

CLK_DIV4RSD
// CLK_DIV4RSD: Clock Divide by 4 with synchronous reset and start
// delay
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV4RSD CLK_DIV4RSD_inst (
    .CLKDV(CLKDV), // Divided clock output

```

```

        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

    defparam CLK_DIV4RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV4RSD_inst instantiation

CLK_DIV6RSD
// CLK_DIV6RSD: Clock Divide by 6 with synchronous reset and start
// delay
//
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

    CLK_DIV6RSD CLK_DIV6RSD_inst (
        .CLKDV(CLKDV), // Divided clock output
        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

    defparam CLK_DIV6RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV6RSD_inst instantiation

CLK_DIV8RSD
// CLK_DIV8RSD: Clock Divide by 8 with synchronous reset and start
// delay
//
// CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

    CLK_DIV8RSD CLK_DIV8RSD_inst (
        .CLKDV(CLKDV), // Divided clock output
        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

    defparam CLK_DIV8RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV8RSD_inst instantiation

CLK_DIV10RSD
// CLK_DIV10RSD: Clock Divide by 10 with synchronous reset and start

```

```
// delay
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV10RSD CLK_DIV10RSD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV10RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV10RSD_inst instantiation

CLK_DIV12RSD
// CLK_DIV12RSD: Clock Divide by 12 with synchronous reset and start
// delay
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV12RSD CLK_DIV12RSD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV12RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV12RSD_inst instantiation

CLK_DIV14RSD
// CLK_DIV14RSD: Clock Divide by 14 with synchronous reset and start
// delay
//          CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV14RSD CLK_DIV14RSD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.
```

```
        defparam CLK_DIV14RSD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV14RSD_inst instantiation

CLK_DIV16RSD
// CLK_DIV16RSD: Clock Divide by 16 with synchronous reset and start
// delay
//                               CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV16RSD CLK_DIV16RSD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

        defparam CLK_DIV16RSD_inst.DIVIDER_DELAY = 1;

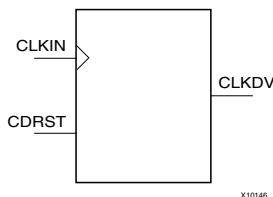
// End of CLK_DIV16RSD_inst instantiation
```


CLK_DIV2,4,6,8,10,12,14,16SD

Global Clock Divider with Start Delay

Architectures Supported

CLK_DIV2SD	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive
CLK_DIV4SD, CLK_DIV6SD, CLK_DIV8SD, CLK_DIV10SD, CLK_DIV12SD, CLK_DIV14SD, CLK_DIV16SD	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



The CLK_DIV2,4,6,8,10,12,14,16SD Global Clock Dividers with Start Delay divide a user-provided external clock signal `gclk<2>` by 2, 4, 6, 8, 10, 12, 14, and 16, respectively. Only one clock divider may be used per design. The global clock divider is available on the XC2C128, XC2C256, XC2C384, and XC2C512 CoolRunner-II devices, but not the XC2C32 or XC2C64. The CLKIN input can only be connected to the device `gclk<2>` pin. The duty cycle of the CLKDVK output is 50-50.

The start delay function delays the CLKDVK output $(n + 1)$ clocks, where n is the divisor for the clock divider.

The CLKDVK output is reset low by power-on reset circuitry.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```

CLK_DIV2SD
-- CLK_DIV2SD: Clock Divide by 2 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

    CLK_DIV2SD_inst : CLK_DIV2SD
-- Edit the following generic to specify the number of clock cycles

```

```
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV2SD_inst instantiation

CLK_DIV4SD
-- CLK_DIV4SD: Clock Divide by 4 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV4SD_inst : CLK_DIV4SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV4SD_inst instantiation

CLK_DIV6SD
-- CLK_DIV6SD: Clock Divide by 6 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV6SD_inst : CLK_DIV6SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV6SD_inst instantiation

CLK_DIV8SD
-- CLK_DIV8SD: Clock Divide by 8 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV8SD_inst : CLK_DIV8SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
  DIVIDER_DELAY => 1)
port map (
  CLKDV => CLKDV,    -- Divided clock output
  CLKIN => CLKIN     -- Clock input
```

```
);

-- End of CLK_DIV8SD_inst instantiation

CLK_DIV10SD
-- CLK_DIV10SD: Clock Divide by 10 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV10SD_inst : CLK_DIV10SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV10SD_inst instantiation

CLK_DIV12SD
-- CLK_DIV12SD: Clock Divide by 12 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV12SD_inst : CLK_DIV12SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV12SD_inst instantiation

CLK_DIV14SD
-- CLK_DIV14SD: Clock Divide by 14 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV14SD_inst : CLK_DIV14SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN     -- Clock input
);

-- End of CLK_DIV14SD_inst instantiation
```

```

CLK_DIV16SD
-- CLK_DIV16SD: Clock Divide by 16 with start delay
--           CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

CLK_DIV16SD_inst : CLK_DIV16SD
-- Edit the following generic to specify the number of clock cycles
-- to delay before starting.
generic map (
    DIVIDER_DELAY => 1)
port map (
    CLKDV => CLKDV,    -- Divided clock output
    CLKIN => CLKIN    -- Clock input
);

-- End of CLK_DIV16SD_inst instantiation

```

Verilog Instantiation Template

```

CLK_DIV2SD
// CLK_DIV2SD: Clock Divide by 2 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV2SD CLK_DIV2SD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV2SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV2SD_inst instantiation

CLK_DIV4SD
// CLK_DIV4SD: Clock Divide by 4 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV4SD CLK_DIV4SD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV4SD_inst.DIVIDER_DELAY = 1;

```

```
// End of CLK_DIV4SD_inst instantiation

CLK_DIV6SD
// CLK_DIV6SD: Clock Divide by 6 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV6SD CLK_DIV6SD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV6SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV6SD_inst instantiation

CLK_DIV8SD
// CLK_DIV8SD: Clock Divide by 8 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV8SD CLK_DIV8SD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

defparam CLK_DIV8SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV8SD_inst instantiation

CLK_DIV10SD
// CLK_DIV10SD: Clock Divide by 10 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

CLK_DIV10SD CLK_DIV10SD_inst (
    .CLKDV(CLKDV), // Divided clock output
    .CDRST(CDRST), // Synchronous reset input
    .CLKIN(CLKIN) // Clock input
);

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
```

```
// reflected in the defparam statements.

    defparam CLK_DIV10SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV10SD_inst instantiation

CLK_DIV12SD
// CLK_DIV12SD: Clock Divide by 12 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

    CLK_DIV12SD CLK_DIV12SD_inst (
        .CLKDV(CLKDV), // Divided clock output
        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

    defparam CLK_DIV12SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV12SD_inst instantiation

CLK_DIV14SD
// CLK_DIV14SD: Clock Divide by 14 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

    CLK_DIV14SD CLK_DIV14SD_inst (
        .CLKDV(CLKDV), // Divided clock output
        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );

// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.

    defparam CLK_DIV14SD_inst.DIVIDER_DELAY = 1;

// End of CLK_DIV14SD_inst instantiation

CLK_DIV16SD
// CLK_DIV16SD: Clock Divide by 16 with start delay
//           CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

    CLK_DIV16SD CLK_DIV16SD_inst (
        .CLKDV(CLKDV), // Divided clock output
        .CDRST(CDRST), // Synchronous reset input
        .CLKIN(CLKIN) // Clock input
    );
```

```
// Edit the following defparam to specify the number of clock
// cycles to delay before starting. If the instance name to
// the clock divider is changed, that change needs to be
// reflected in the defparam statements.
```

```
defparam CLK_DIV16SD_inst.DIVIDER_DELAY = 1;
```

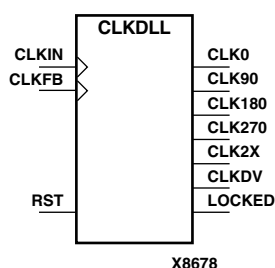
```
// End of CLK_DIV16SD_inst instantiation
```


CLKDLL

Clock Delay Locked Loop

Architectures Supported

CLKDLL	
Spartan-II, Spartan-III	Primitive*
Spartan-3	No
Virtex, Virtex-E	Primitive*
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Use CLKDLLE for Spartan-III and Virtex-E.	



CLKDLL is a clock delay locked loop used to minimize clock skew. CLKDLL synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Sheets* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Sheets* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG. If phase alignment is not required, CLKIN can also be driven by IBUF.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG connected to the CLKFB input of the CLKDLL must be sourced from either the CLK0 or CLK2X outputs of the same CLKDLL. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X and CLKDV outputs is always 50-50. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLL to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLL Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLKIN frequency, shifted 180° with regards to CLK0
CLK270	Clock at 1x CLKIN frequency, shifted 270° with regards to CLK0
CLK2X	Clock at 2x CLKIN frequency, in phase with CLK0
CLK90	Clock at 1x CLKIN frequency, shifted 90° with regards to CLK0
CLKDV	Clock at (1/n)x CLKIN frequency, n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	CLKDLL locked

Note: See the "PERIOD Specifications on CLKDLLs and DCM" in the *Constraints Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLL components.

Usage

This component is generally instantiated in the code as it can not be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. See your synthesis tool's documentation. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLL component to construct the proper clocking circuit. See the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Data Sheets* for more information on using the CLKDLL component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLL should be placed
-- after architecture statement but before begin keyword

component CLKDLL
  -- synthesis translate_off
  generic map (CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5,
    3.0, 4.0, 5.0, 8.0, 16.0)
    DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
    STARTUP_WAIT : boolean := FALSE) -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0 : out STD_ULOGIC;
    CLK180 : out STD_ULOGIC;
    CLK270 : out STD_ULOGIC;
    CLK2X : out STD_ULOGIC;
    CLK90 : out STD_ULOGIC;
    CLKDV : out STD_ULOGIC;
    LOCKED : out STD_ULOGIC;
    CLKFB : in STD_ULOGIC;
    CLKIN : in STD_ULOGIC;
    RST : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLL
-- should be placed after architecture declaration but
-- before the begin keyword
```

```

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute STARTUP_WAIT : boolean;

attribute CLKDV_DIVIDE of CLKDLL_instance_name: label is 2.0;
-- 1.5,2,2.5,3,4, 5, 8, 16 are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLL_instance_name: label is
"TRUE";
-- TRUE, FALSE are valid for DUTY_CYCLE_CORRECTION
attribute STARTUP_WAIT of CLKDLL_instance_name: label is "FALSE"; --
(TRUE,FALSE)

-- Component Instantiation for CLKDLL should be placed
-- in architecture after the begin keyword

CLKDLL_INSTANCE_NAME : CLKDLL
-- synthesis translate_off
generic map (CLKDV_DIVIDE => real_value,
-- (1.5,2,2.5,3,4,5,8,16)
DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE,
FALSE)
STARTUP_WAIT => boolean_value); -- (TRUE, FALSE)
-- synthesis translate_on
port map (CLK0 => user_CLK0,
CLK180 => user_CLK180,
CLK270 => user_CLK270,
CLK2X => user_CLK2X,
CLK90 => user_CLK90,
CLKDV => user_CLKDV,
LOCKED => user_LOCKED,
CLKFB => user_CLKFB,
CLKIN0 => user_CLKIN,
RST => user_RST);

```

Verilog Instantiation Template

```

CLKDLL CLKDLL_instance_name (.CLK0 (user_CLK0),
                             .CLK180 (user_CLK180),
                             .CLK270 (user_CLK270),
                             .CLK2X (user_CLK2X),
                             .CLK90 (user_CLK90),
                             .CLKDV (user_CLKDV),
                             .LOCKED (user_LOCKED),
                             .CLKFB (user_CLKFB),
                             .CLKIN (user_CLKIN),
                             .RST (user_RST));

defparam CLKDLL_instance_name.CLKDV_DIVIDE = integer_value;
// (1.5,2,2.5,3,4,5,8,16)
defparam CLKDLL_instance_name.DUTY_CYCLE_CORRECTION = boolean_value; //
(TRUE, FALSE)
defparam CLKDLL_instance_name.STARTUP_WAIT = boolean_value; // (TRUE,
FALSE)

```

Note: Additional syntax may be necessary in order to pass the CLKDLL attributes via the synthesis tool. The above defparam statements may need to be isolated from the synthesis tool with translate_off/translate_on directives. See your synthesis tool documentation for more

information on Verilog attribute passing to ensure that you properly pass these attributes to the synthesis tool. Otherwise, you may pass these attributes to the UCF file.

Commonly Used Constraints

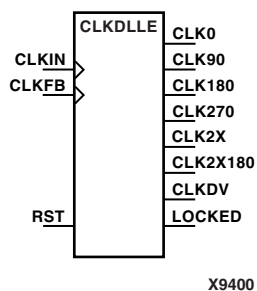
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE and LOC.

CLKDLLE

Virtex-E Clock Delay Locked Loop

Architectures Supported

CLKDLLE	
Spartan-II, Spartan-III	Primitive*
Spartan-3	No
Virtex, Virtex-E	Primitive*
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Supported for Spartan-III and Virtex-E devices only.	



CLKDLLE is a clock delay locked loop used to minimize clock skew for Virtex-E devices. CLKDLLE synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Sheets* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Sheets* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 or CLK2X output of CLKDLLE. The BUFG connected to the CLKFB input of the CLKDLLE must be sourced from either the CLK0 or CLK2X outputs of the same CLKDLLE. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X, CLK2X180, and CLKDV outputs is always 50-50. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLLE to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLLE Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLK0 frequency, shifted 180° with regards to CLK0
CLK270	Clock at 1x CLK0 frequency, shifted 270° with regards to CLK0
CLK2X	Clock at 2x CLK0 frequency, in phase with CLK0
CLK2X180	Clock at 1x CLK2X frequency shifted 180° with regards to CLK2X
CLK90	Clock at 1x CLK0 frequency, shifted 90° with regards to CLK0
CLKDV	Clock at $(1/n) \times \text{CLK0}$ frequency, where $n = \text{CLKDV_DIVIDE}$ value. CLKDV is in phase with CLK0.
LOCKED	CLKDLLE locked. CLKIN and CLKFB synchronized.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. See your synthesis tool documentation. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLLE component to construct the proper clocking circuit. See the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Data Sheets* for more information on using the CLKDLLE component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLLE should be placed
-- after architecture statement but before begin keyword

component CLKDLLE
  -- synthesis translate_off
  generic map (CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0,
    3.5, 4.0, 4.5, 5.0,5.5, 6.0, 6.5, 7.5, 8.0, 9.0,
    10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
    DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
    STARTUP_WAIT :boolean := FALSE) -- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0 : out STD_ULOGIC;
    CLK180 : out STD_ULOGIC;
    CLK270 : out STD_ULOGIC;
    CLK2X : out STD_ULOGIC;
    CLK2X180: out STD_ULOGIC;
    CLK90 : out STD_ULOGIC;
    CLKDV : out STD_ULOGIC;
    LOCKED : out STD_ULOGIC;
    CLKFB : in STD_ULOGIC;
    CLKIN : in STD_ULOGIC;
    RST : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLLE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute STARTUP_WAIT : boolean;

attribute CLKDV_DIVIDE of CLKDLLE_instance_name: label is 2.0;
-- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.5, 8.0,
   9.0, 10.0, 11.0,
-- 12.0, 13.0, 14.0, 15.0, 16.0) are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLLE_instance_name: label is
TRUE;
-- (TRUE, FALSE) are valid for DUTY_CYCLE_CORRECTION
attribute STARTUP_WAIT of CLKDLLE_instance_name: label is FALSE; --
(TRUE, FALSE)

-- Component Instantiation for CLKDLLE should be placed
-- in architecture after the begin keyword

CLKDLLE_INSTANCE_NAME : CLKDLLE
-- synthesis translate_off
generic map (CLKDV_DIVIDE => real_value, --
(1.5,2,2.5,3,4,5,8,16)
DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE, FALSE)
STARTUP_WAIT => boolean_value); -- (TRUE, FALSE)
-- synthesis translate_on
port map (CLK0 => user_CLK0,
CLK180 => user_CLK180,
CLK270 => user_CLK270,
CLK2X => user_CLK2X,
CLK2X180 => user_CLK2X,
CLK90 => user_CLK90,
CLKDV => user_CLKDV,
LOCKED => user_LOCKED,
CLKFB => user_CLKFB,
CLKIN0 => user_CLKIN,
RST => user_RST);

```

Verilog Instantiation Template

```

CLKDLLE CLKDLLE_instance_name (.CLK0 (user_CLK0),
                               .CLK180 (user_CLK180),
                               .CLK270 (user_CLK270),
                               .CLK2X (user_CLK2X),
                               .CLK2X180 (user_CLK2X180),
                               .CLK90 (user_CLK90),
                               .CLKDV (user_CLKDV),
                               .LOCKED (user_LOCKED),
                               .CLKFB (user_CLKFB),
                               .CLKIN (user_CLKIN),
                               .RST (user_RST));

defparam CLKDLLE_instance_name.CLKDV_DIVIDE = integer_value;
// 1.5,2,2.5,3,4,5,8,16 are valid for CLKDV_DIVIDE
defparam CLKDLLE_instance_name.DUTY_CYCLE_CORRECTION =
boolean_value;// (TRUE,FALSE)
defparam CLKDLLE_instance_name.STARTUP_WAIT = boolean_value; // (TRUE,
FALSE)

```

Note: Additional syntax may be necessary in order to pass the CLKDLLE attributes via the synthesis tool. The above defparam statements may need to be isolated from the synthesis tool with `translate_off/translate_on` directives. See your synthesis tool documentation for more information on Verilog attribute passing to ensure that you properly pass these attributes to the synthesis tool. Otherwise, you may pass these attributes to the UCF file.

Commonly Used Constraints

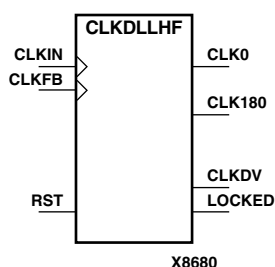
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE, and LOC

CLKDLLHF

High Frequency Clock Delay Locked Loop

Architectures Supported

CLKDLLHF	
Spartan-II, Spartan-III	Primitive
Spartan-3	No
Virtex, Virtex-E	Primitive*
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Use CLKDLLHF for the Virtex-E DLL in HF mode. In LF mode, both the separate CLKDLLE and CLKDLL primitive can be used.	



CLKDLLHF is a high frequency clock delay locked loop used to minimize clock skew. CLKDLLHF synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specific range of each other (see *The Programmable Logic Data Sheets* for the most current value).

The frequency of the clock signal at the CLKIN input must be in a specific range depending on speed grade (see *The Programmable Logic Data Sheets* for the most current values). The CLKIN pin must be driven by an IBUFG or a BUFG.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG input can only be connected to the CLK0 output of CLKDLLHF. The BUFG connected to the CLKFB input of the CLKDLLHF must be sourced from the CLK0 output of the same CLKDLLHF. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Only the CLK0 output can be used. CLK0 must be connected to the input of OBUF, an output buffer.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted output (CLK180) is the same as that of the CLK0 output. The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

The master reset input (RST) resets CLKDLLHF to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for just 2ns.

CLKDLLHF Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLKIN frequency, shifted 180° with regards to CLK0
CLKDV	Clock at (1/n)x CLKIN frequency, n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	CLKDLLHF locked

Note: See the "PERIOD Specifications on CLKDLLs and DCM" section of the "Xilinx Constraints P" chapter in the *Constraints Guide* for additional information on using the TNM, TNM_NET, and PERIOD attributes with CLKDLLHF components.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. See your synthesis tool documentation. Generally, global buffers (IBUFG, BUFG) are instantiated with the CLKDLLHF component to construct the proper clocking circuit. See the XAPP 132 application note, "Using the Virtex Delay-Locked Loop" and the *Xilinx Data Sheets* for more information on using the CLKDLLHF component.

VHDL Instantiation Template

```
-- Component Declaration for CLKDLLHF should be placed
-- after architecture statement but before begin keyword

component CLKDLLHF
  -- synthesis translate_off
  generic map (CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0,
    4.0, 5.0, 8.0, 16.0) DUTY_CYCLE_CORRECTION : Boolean := TRUE --
    (TRUE, FALSE) STARTUP_WAIT : boolean := FALSE)-- (TRUE, FALSE)
  -- synthesis translate_on
  port (CLK0 : out STD_ULOGIC;
    CLK180 : out STD_ULOGIC;
    CLKDV : out STD_ULOGIC;
    LOCKED : out STD_ULOGIC;
    CLKFB : in STD_ULOGIC;
    CLKIN : in STD_ULOGIC;
    RST : in STD_ULOGIC);
end component;

-- Component Attribute specification for CLKDLLHF
-- should be placed after architecture declaration but
-- before the begin keyword

attribute CLKDV_DIVIDE : real;
attribute DUTY_CYCLE_CORRECTION : boolean;
attribute STARTUP_WAIT : boolean;

attribute CLKDV_DIVIDE of CLKDLLHF_instance_name: label is 2.0;
-- (1.5,2,2.5,3,4, 5, 8, 16) are valid for CLKDV_DIVIDE
attribute DUTY_CYCLE_CORRECTION of CLKDLLHF_instance_name: label is
  TRUE;
```

```

-- (TRUE, FALSE) are valid for DUTY_CYCLE_CORRECTION
attribute STARTUP_WAIT of CLKDLLHF_instance_name: label is FALSE; --
    (TRUE,FALSE)

-- Component Instantiation for CLKDLLHF should be placed
-- in architecture after the begin keyword

CLKDLLHF_INSTANCE_NAME : CLKDLLHF
-- synthesis translate_off
    generic map(CLKDV_DIVIDE => real_value, -- (1.5,2,2.5,3,4,5,8,16)
                DUTY_CYCLE_CORRECTION => boolean_value, -- (TRUE,
    FALSE)
                STARTUP_WAIT => boolean_value); -- (TRUE, FALSE)
-- synthesis translate_on
port map (CLK0 => user_CLK0,
          CLK180 => user_CLK180,
          CLKDV => user_CLKDV,
          LOCKED => user_LOCKED,
          CLKFB => user_CLKFB,
          CLKIN => user_CLKIN,
          RST => user_RST);
Verilog Instantiation Template
CLKDLLHF CLKDLLHF_instance_name (.CLK0 (user_CLK0),
                                .CLK180 (user_CLK180),
                                .CLKDV (user_CLKDV),
                                .LOCKED (user_LOCKED),
                                .CLKFB (user_CLKFB),
                                .CLKIN (user_CLKIN),
                                .RST (user_RST));

defparam CLKDLLHF_instance_name.CLKDV_DIVIDE = integer_value;
// 1.5,2,2.5,3,4,5,8,16 are valid for CLKDV_DIVIDE
defparam CLKDLLHF_instance_name.DUTY_CYCLE_CORRECTION =
    boolean_value;// (TRUE,FALSE)
defparam CLKDLLHF_instance_name.STARTUP_WAIT = boolean_value; //
    (TRUE, FALSE)

```

Note: Additional syntax may be necessary in order to pass the CLKDLLHF attributes via the synthesis tool. The above defparam statements may need to be isolated from the synthesis tool with translate_off/translate_on directives. See your synthesis tool documentation for more information on Verilog attribute passing to ensure that you properly pass these attributes to the synthesis tool. Otherwise, you may pass these attributes to the UCF file.

Commonly Used Constraints

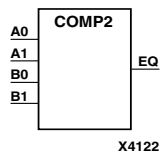
STARTUP_WAIT, DUTY_CYCLE_CORRECTION, CLKDV_DIVIDE, LOC

COMP2, 4, 8, 16

2-, 4-, 8-, 16-Bit Identity Comparators

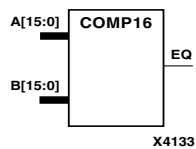
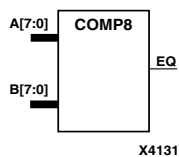
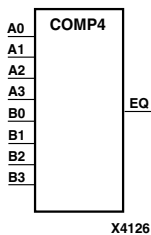
Architectures Supported

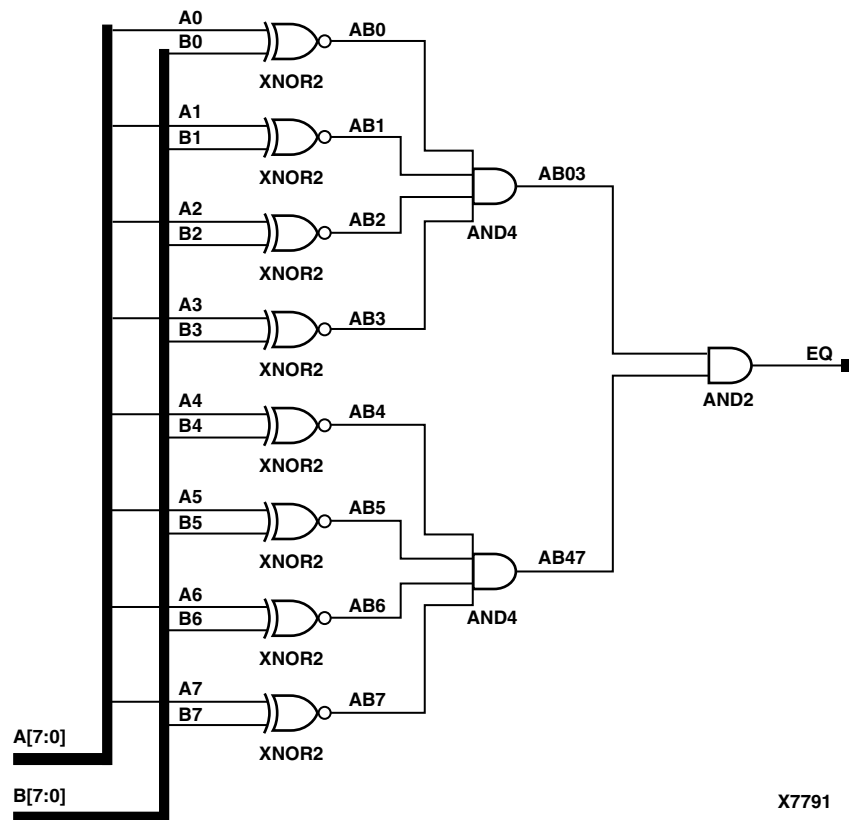
COMP2, COMP4, COMP8, COMP16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



COMP2, COMP4, COMP8, and COMP16 are, respectively, 2-, 4-, 8-, and 16-bit identity comparators. The equal output (EQ) of the COMP2 2-bit, identity comparator is High when the two words A1 – A0 and B1 – B0 are equal. EQ is high for COMP4 when A3 – A0 and B3 – B0 are equal; for COMP8, when A7 – A0 and B7 – B0 are equal; and for COMP16, when A15 – A0 and B15 – B0 are equal.

Equality is determined by a bit comparison of the two words. When any two of the corresponding bits from each word are not the same, the EQ output is Low.





COMP8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

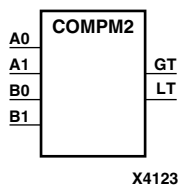
For HDL, these design elements are inferred rather than instantiated.

COMP2, 4, 8, 16

2-, 4-, 8-, 16-Bit Magnitude Comparators

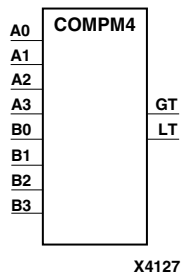
Architectures Supported

COMP2, COMP4, COMP8, COMP16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



COMP2, COMP4, COMP8, and COMP16 are, respectively, 2-, 4-, 8-, and 16-bit magnitude comparators that compare two positive binary-weighted words.

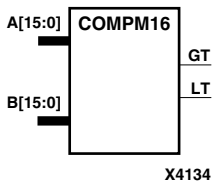
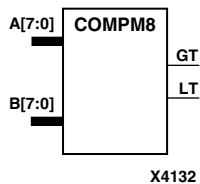
COMP2 compares A1 – A0 and B1 – B0, where A1 and B1 are the most significant bits. COMP4 compares A3 – A0 and B3 – B0, where A3 and B3 are the most significant bits. COMP8 compares A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMP16 compares A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.



The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be measured with this macro by comparing both outputs with a NOR gate.

COMP2 Truth Table

Inputs				Outputs	
A1	B1	A0	B0	GT	LT
0	0	0	0	0	0
0	0	1	0	1	0
0	0	0	1	0	1
0	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	0
1	1	0	1	0	1
1	1	1	1	0	0
1	0	X	X	1	0
0	1	X	X	0	1

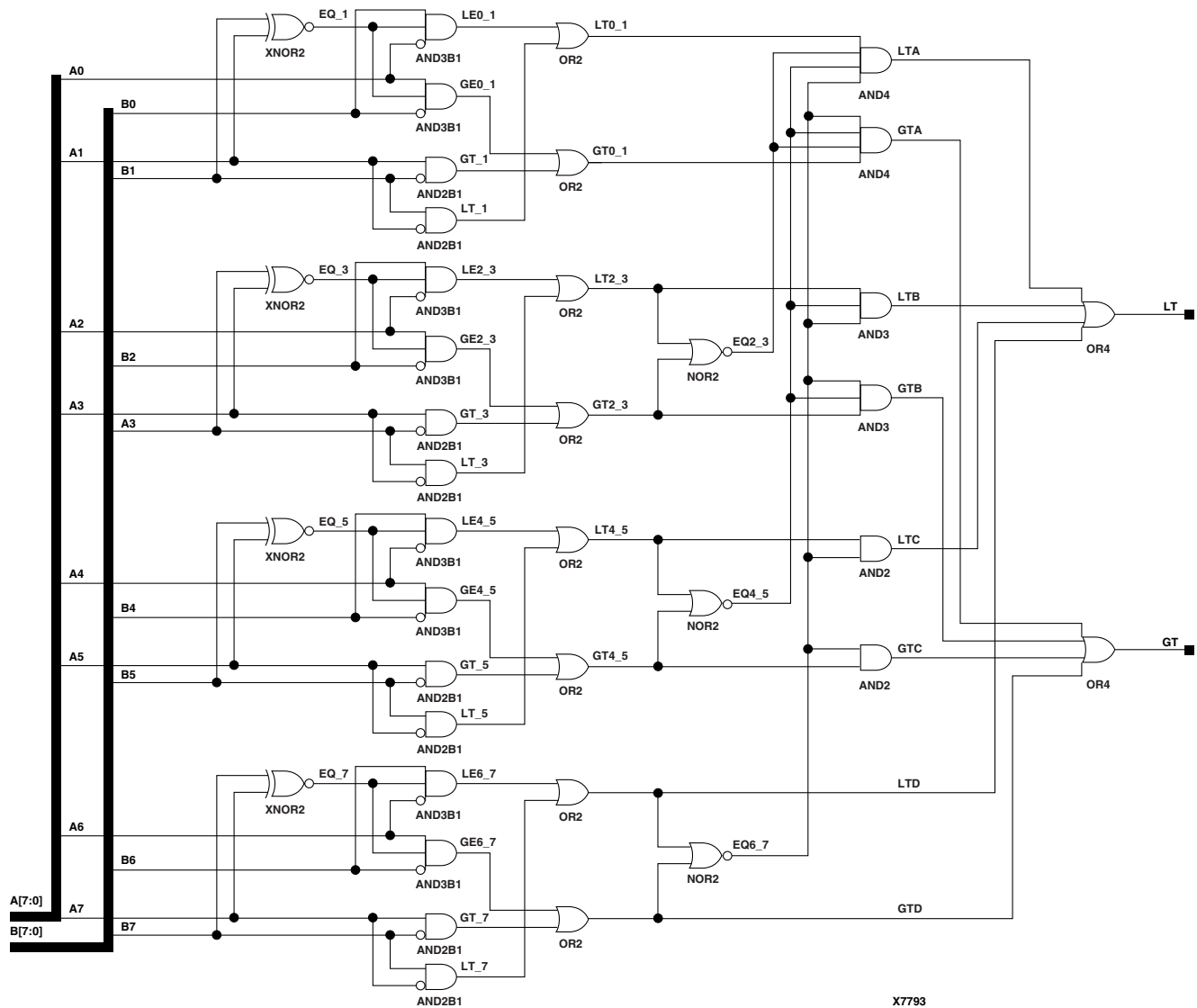


COMPM4 Truth Table

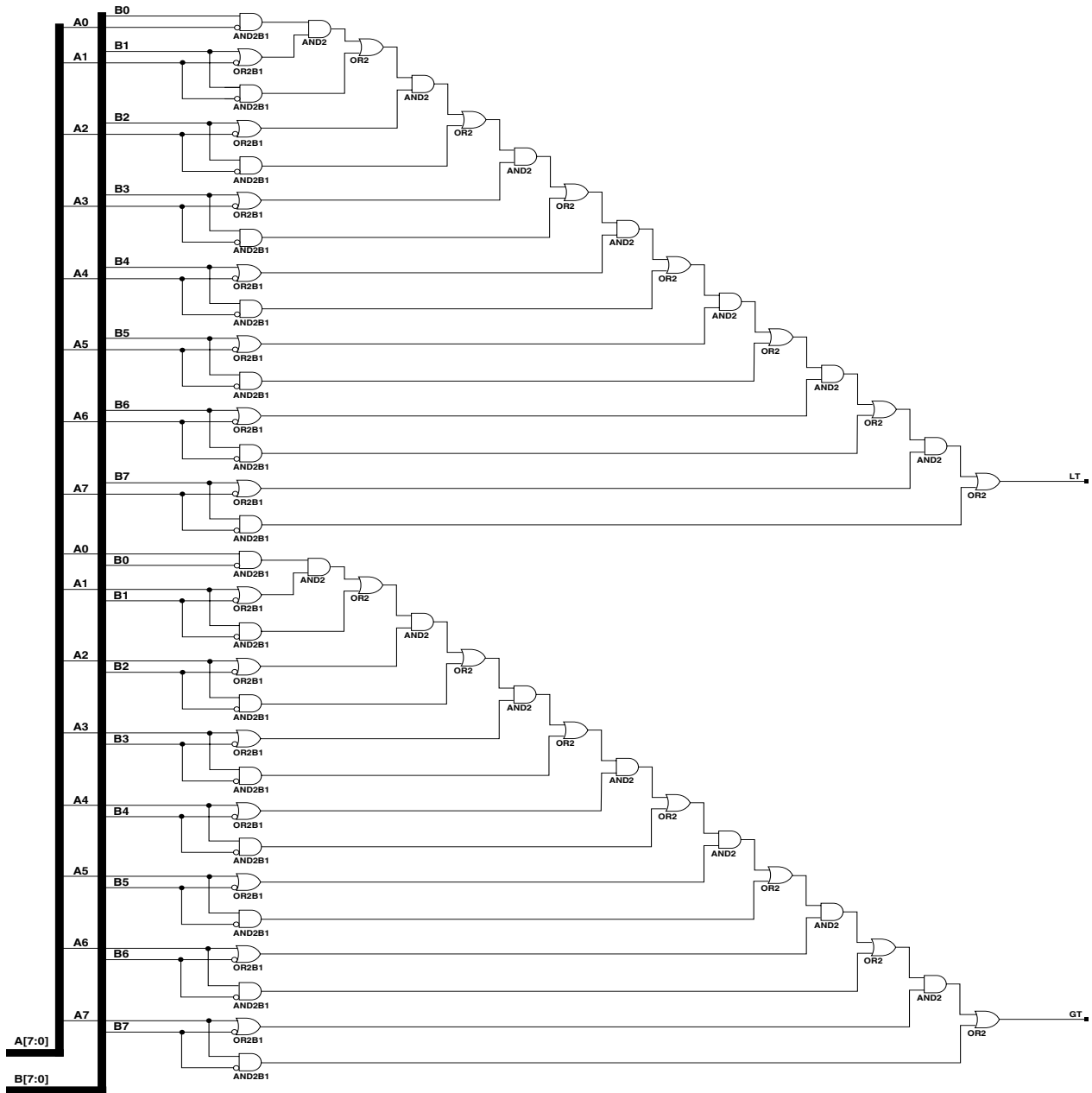
Inputs				Outputs	
A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A3>B3	X	X	X	1	0
A3<B3	X	X	X	0	1
A3=B3	A2>B2	X	X	1	0
A3=B3	A2<B2	X	X	0	1
A3=B3	A2=B2	A1>B1	X	1	0
A3=B3	A2=B2	A1<B1	X	0	1
A3=B3	A2=A2	A1=B1	A0>B0	1	0
A3=B3	A2=B2	A1=B1	A0<B0	0	1
A3=B3	A2=B2	A1=B1	A0=B0	0	0

COMPM8 Truth Table (also representative of COMPM16)

Inputs								Outputs	
A7, B7	A6, B6	A5, B5	A4, B4	A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0



COMPM8 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7632

COMP8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

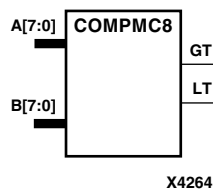
For HDL, these design elements are supported for inference rather than instantiation.

COMP8, 16

8-, 16-Bit Magnitude Comparators

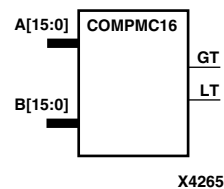
Architectures Supported

COMP8, COMP8, COMP16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



COMP8 is an 8-bit, magnitude comparator that compares two positive binary-weighted words A7 – A0 and B7 – B0, where A7 and B7 are the most significant bits. COMP16 is a 16-bit, magnitude comparator that compares two positive binary-weighted words A15 – A0 and B15 – B0, where A15 and B15 are the most significant bits.

These comparators are implemented using carry logic with relative location constraints to ensure efficient logic placement.



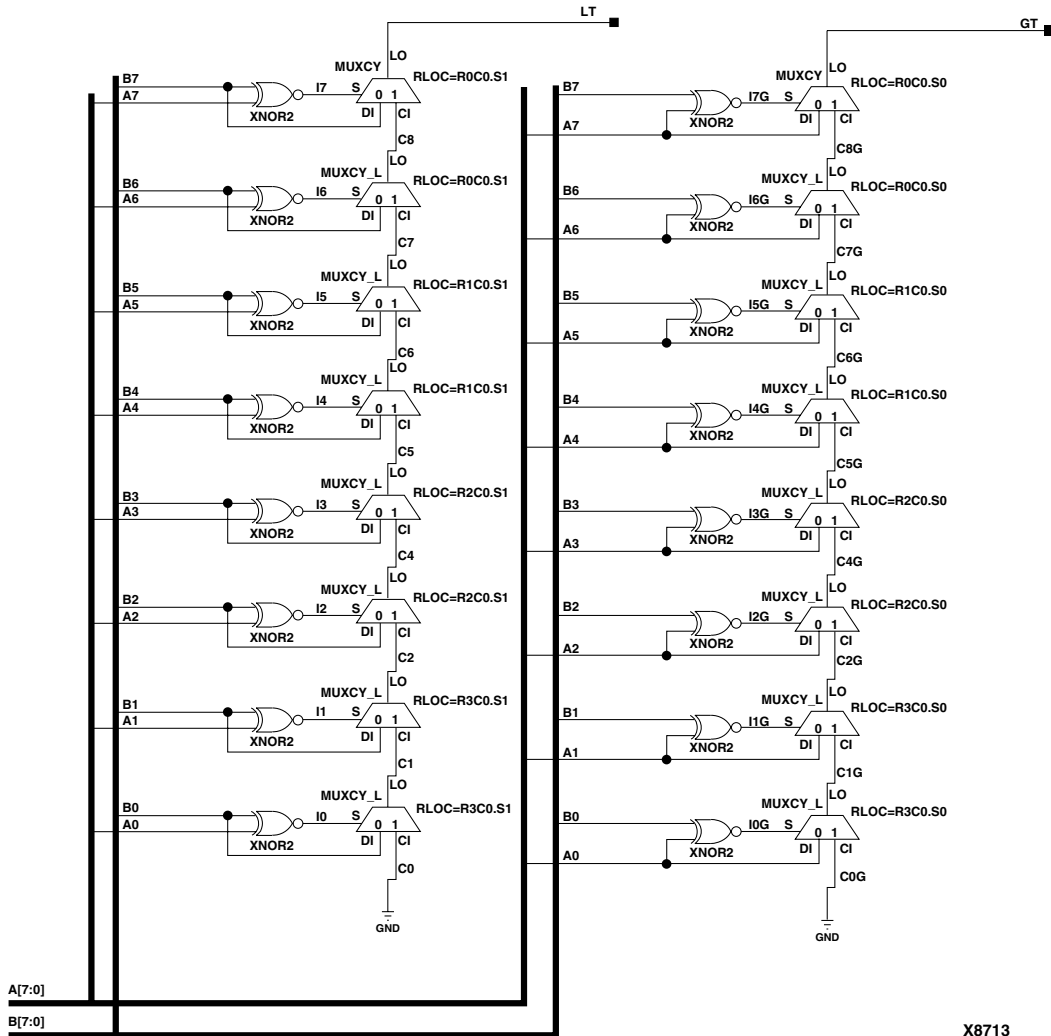
The greater-than output (GT) is High when A>B, and the less-than output (LT) is High when A<B. When the two words are equal, both GT and LT are Low. Equality can be flagged with this macro by connecting both outputs to a NOR gate.

COMP8 Truth Table (also representative of COMP16)

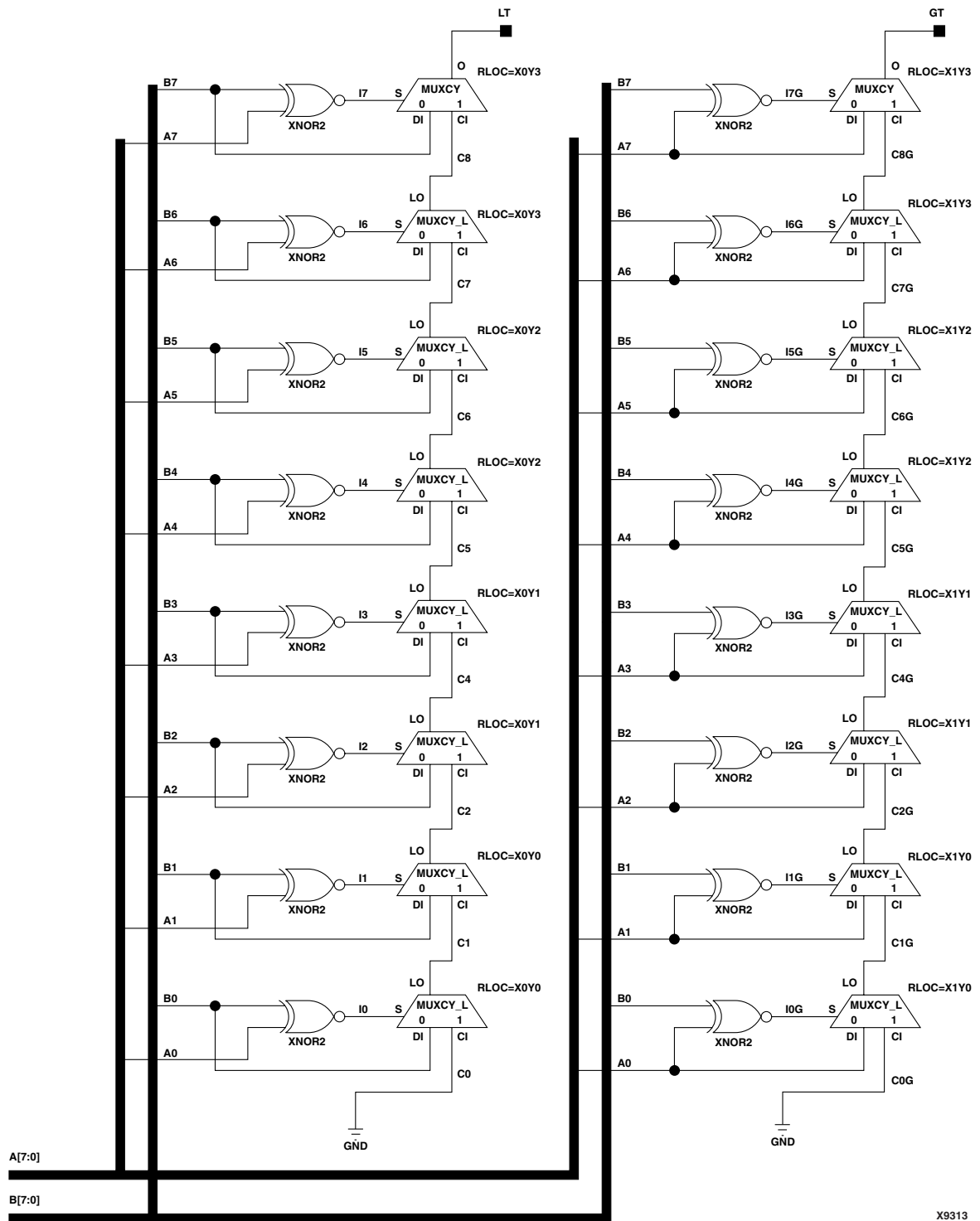
Inputs								Outputs	
A7, B7	A6, B6	A5, B5	A4, B4	A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A7>B7	X	X	X	X	X	X	X	1	0
A7<B7	X	X	X	X	X	X	X	0	1
A7=B7	A6>B6	X	X	X	X	X	X	1	0
A7=B7	A6<B6	X	X	X	X	X	X	0	1
A7=B7	A6=B6	A5>B5	X	X	X	X	X	1	0
A7=B7	A6=B6	A5<B5	X	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4>B4	X	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4<B4	X	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3>B3	X	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3<B3	X	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2>B2	X	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2<B2	X	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1>B1	X	1	0
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1<B1	X	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0>B0	1	0

COMP8 Truth Table (also representative of COMP16)

Inputs								Outputs	
A7, B7	A6, B6	A5, B5	A4, B4	A3, B3	A2, B2	A1, B1	A0, B0	GT	LT
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0<B0	0	1
A7=B7	A6=B6	A5=B5	A4=B4	A3=B3	A2=B2	A1=B1	A0=B0	0	0



COMP8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



COMP8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

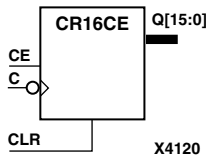
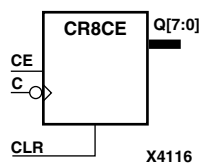
For HDL, these design elements are supported for inference rather than instantiation.

CR8CE, CR16CE

8-, 16-Bit Negative-Edge Binary Ripple Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CR8CE, CR16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



CR8CE and CR16CE are 8-bit and 16-bit, cascadable, clearable, binary, ripple counters. The asynchronous clear (CLR), when High, overrides all other inputs and causes the Q outputs to go to logic level zero. The counter increments when the clock enable input (CE) is High during the High-to-Low clock (C) transition. The counter ignores clock transitions when CE is Low.

Larger counters can be created by connecting the last Q output (Q7 for CR8CE, Q15 for CR16CE) of the first stage to the clock input of the next stage. CLR and CE inputs are connected in parallel. The clock period is not affected by the overall length of a ripple counter. The overall clock-to-output propagation is $n(t_{C-Q})$, where n is the number of stages and the time t_{C-Q} is the C-to-Qz propagation delay of each stage.

The counter is asynchronously cleared, output Low, when power is applied.

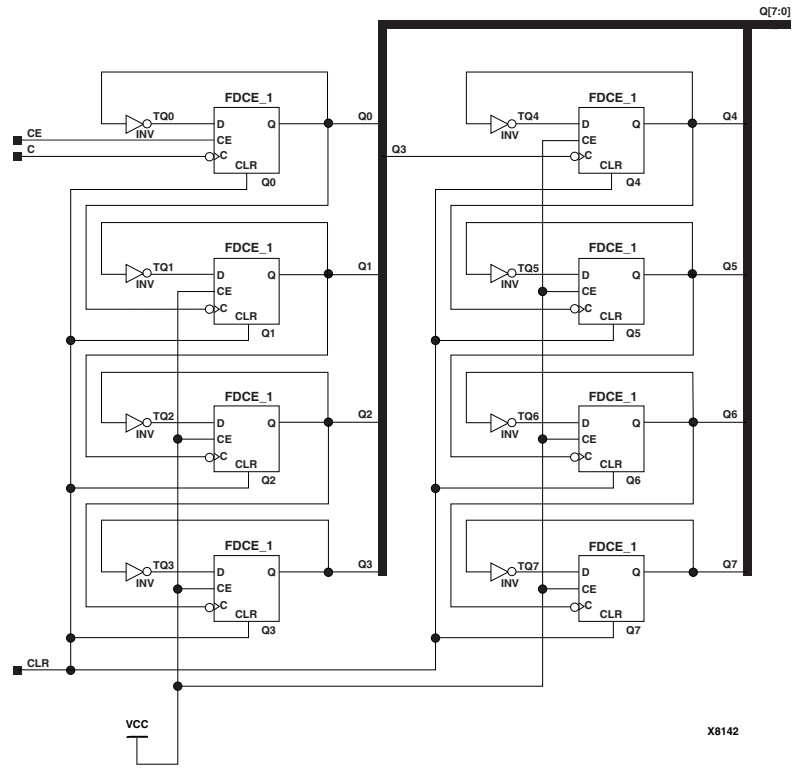
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

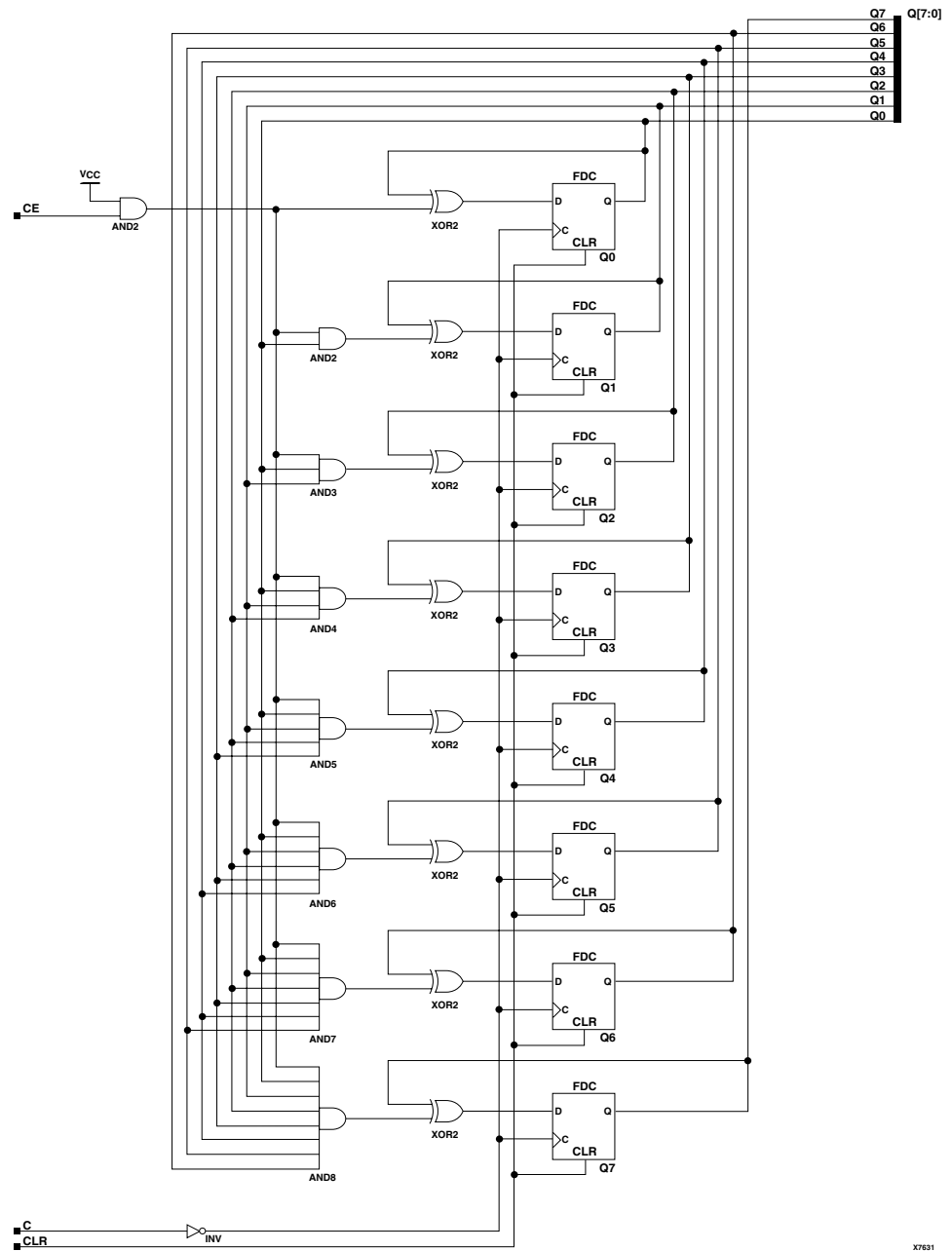
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	CE	C	Qz – Q0
1	X	X	0
0	0	X	No Chg
0	1	↓	Inc

z = 7 for CR8CE; z = 15 for CR16CE.



CR8CE Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



CR8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

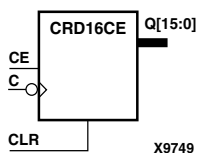
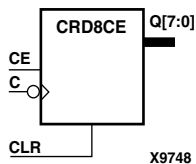
For HDL, these design elements are inferred rather than instantiated.

CRD8CE, CRD16CE

8-, 16-Bit Dual-Edge Triggered Binary Ripple Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CRD8CE, CRD16CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



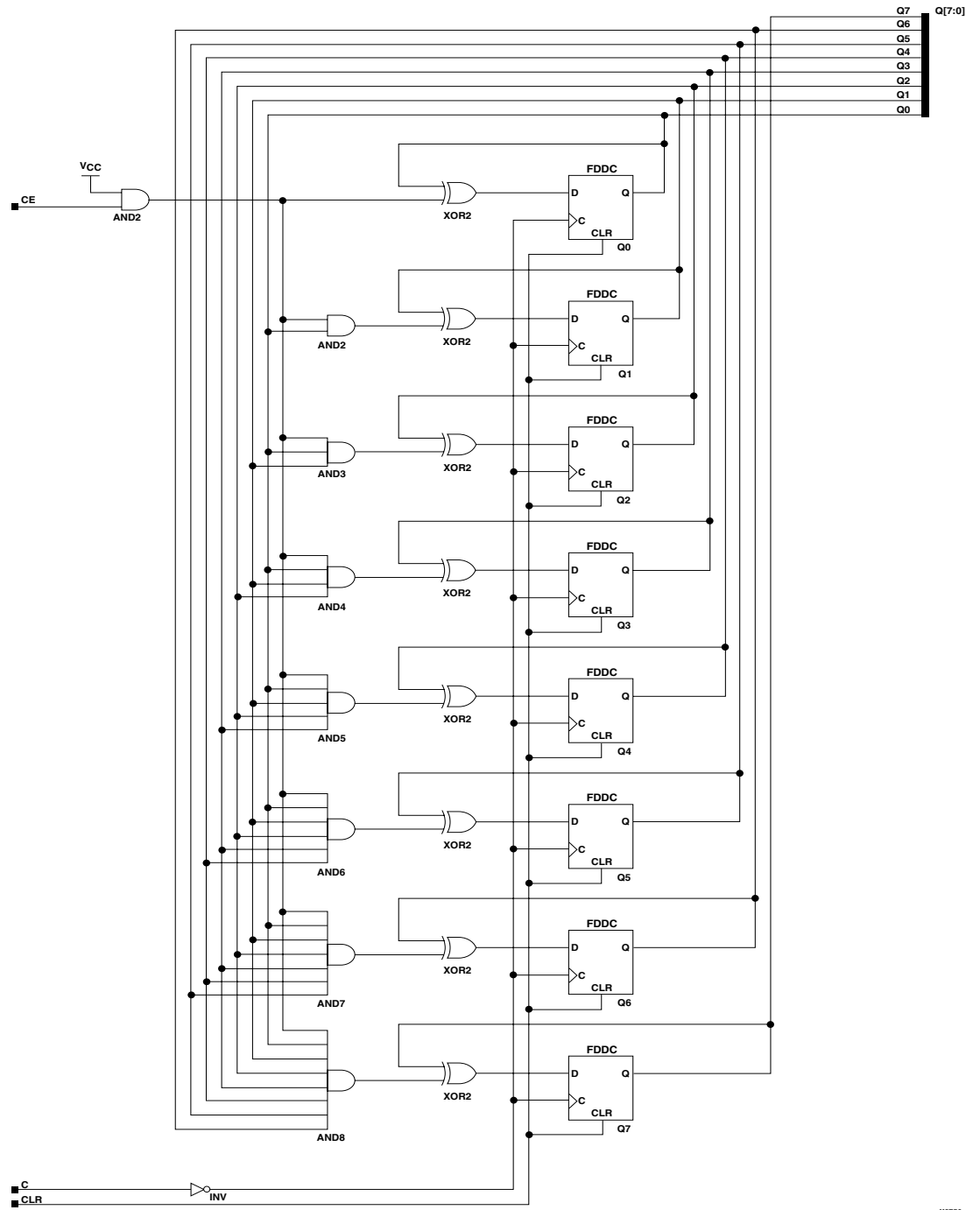
CRD8CE and CRD16CE are dual edge triggered 8-bit and 16-bit, cascadable, clearable, binary, ripple counters. The asynchronous clear (CLR), when High, overrides all other inputs and causes the Q outputs to go to logic level zero. The counter increments when the clock enable input (CE) is High during the High-to-Low and Low-to-High clock (C) transitions. The counter ignores clock transitions when CE is Low.

Larger counters can be created by connecting the last Q output (Q7 for CRD8CE, Q15 for CRD16CE) of the first stage to the clock input of the next stage. CLR and CE inputs are connected in parallel. The clock period is not affected by the overall length of a ripple counter. The overall clock-to-output propagation is $n(t_{C-Q})$, where n is the number of stages and the time t_{C-Q} is the C-to-Qz propagation delay of each stage.

The counter is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
CLR	CE	C	Qz – Q0
1	X	X	0
0	0	X	No Chg
0	1	↑	Inc
0	1	↓	Inc

$z = 7$ for CR8CE; $z = 15$ for CR16CE.



CRD8CE Implementation CoolRunner-II

Usage

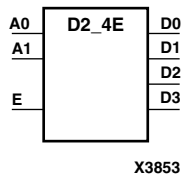
For HDL, these design elements are inferred rather than instantiated.

D2_4E

2- to 4-Line Decoder/Demultiplexer with Enable

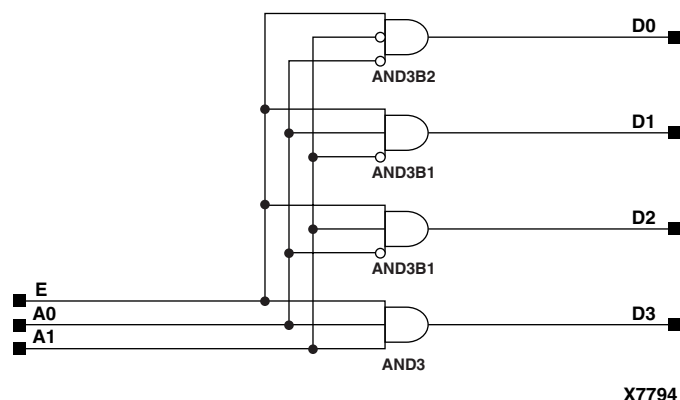
Architectures Supported

D2_4E	
Spartan-II, Spartan-IIIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



When the enable (E) input of the D2_4E decoder/demultiplexer is High, one of four active-High outputs (D3 – D0) is selected with a 2-bit binary address (A1 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

Inputs			Outputs			
A1	A0	E	D3	D2	D1	D0
X	X	0	0	0	0	0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0



D2_4E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

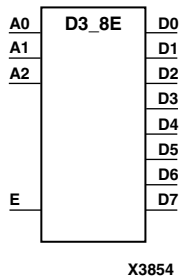
For HDL, this design element is inferred rather than instantiated.

D3_8E

3- to 8-Line Decoder/Demultiplexer with Enable

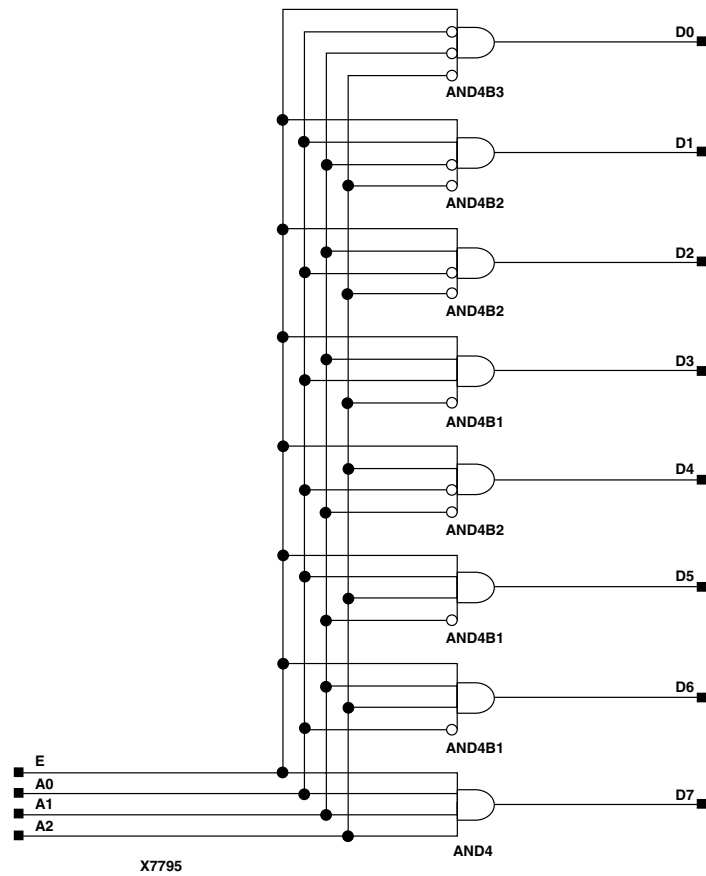
Architectures Supported

D3_8E	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



When the enable (E) input of the D3_8E decoder/demultiplexer is High, one of eight active-High outputs (D7 – D0) is selected with a 3-bit binary address (A2 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

Inputs				Outputs							
A2	A1	A0	E	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



D3_8E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

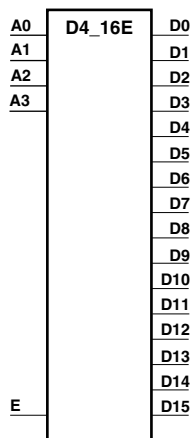
For HDL, this design element is inferred rather than instantiated.

D4_16E

4- to 16-Line Decoder/Demultiplexer with Enable

Architectures Supported

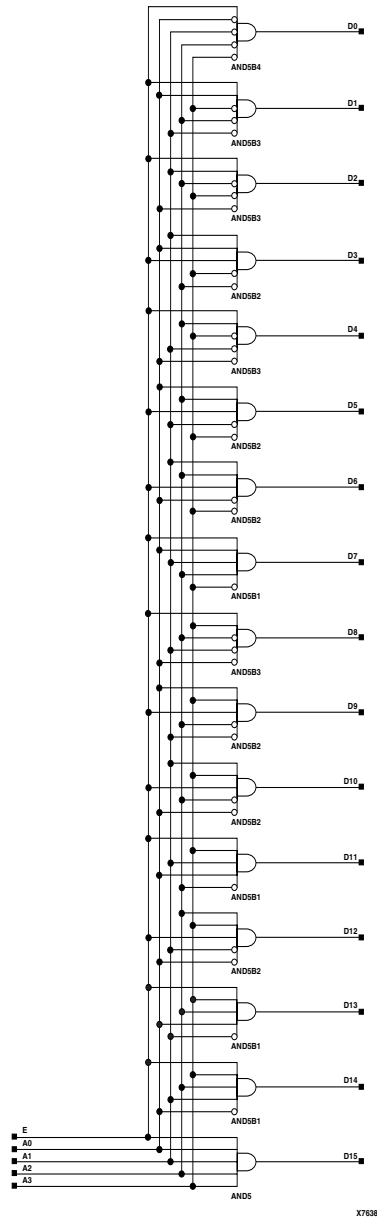
D4_16E	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



X3855

When the enable (E) input of the D4_16E decoder/demultiplexer is High, one of 16 active-High outputs (D15 – D0) is selected with a 4-bit binary address (A3 – A0) input. The non-selected outputs are Low. Also, when the E input is Low, all outputs are Low. In demultiplexer applications, the E input is the data input.

See “D3_8E” for a representative truth table derivation.



D4_16E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

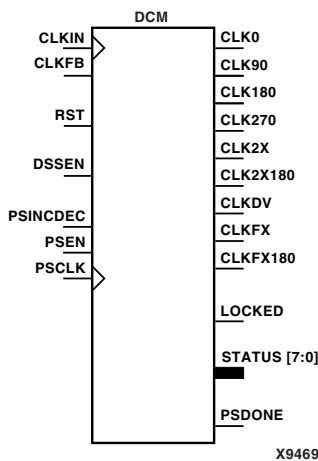
For HDL, this design element is inferred rather than instantiated.

DCM

Digital Clock Manager

Architectures Supported

DCM	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



DCM is a digital clock manager that provides multiple functions. It can implement a clock delay locked loop, a digital frequency synthesizer, digital phase shifter, and a digital spread spectrum.

Note: All unused inputs must be driven Low. The program will automatically tie the inputs Low if they are unused.

Clock Delay Locked Loop (DLL)

DCM includes a clock delay locked loop used to minimize clock skew for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X devices. DCM synchronizes the clock signal at the feedback clock input (CLKFB) to the clock signal at the input clock (CLKIN). The locked output (LOCKED) is high when the two signals are in phase. The signals are considered to be in phase when their rising edges are within a specified time (ps) of each other.

DCM supports two frequency modes for the DLL. By default, the DLL_FREQUENCY_MODE attribute is set to Low and the frequency of the clock signal at the CLKIN input must be in the Low (DLL_CLKIN_MIN_LF to DLL_CLKIN_MAX_LF) frequency range (MHz). In Low frequency mode, the CLK0, CLK90, CLK180, CLK270, CLK2X, CLKDV, and CLK2X180 outputs are available.

For up to and including Virtex-II Pro, you get only CLK0, CLK180, CLKDV, CLKFX and CLKFX180 in the HF mode. In Virtex-4, you get all outputs.

When the DLL_FREQUENCY_MODE attribute is set to High, the frequency of the clock signal at the CLKIN input must be in the High (DLL_CLKIN_MIN_HF to DLL_CLKIN_MAX_HF) frequency range (MHz). In High frequency mode, only the CLK0, CLK180, and CLKDV outputs are available.

On-chip synchronization is achieved by connecting the CLKFB input to a point on the global clock network driven by a BUFG, a global clock buffer. The BUFG connected to the CLKFB input of the DCM must be sourced from either the CLK0 or CLK2X outputs of the same DCM. The CLKIN input should be connected to the output of an IBUFG, with the IBUFG input connected to a pad driven by the system clock.

Off-chip synchronization is achieved by connecting the CLKFB input to the output of an IBUFG, with the IBUFG input connected to a pad. Either the CLK0 or CLK2X output can be used but not both. The CLK0 or CLK2X must be connected to the input

of OBUF, an output buffer. The CLK_FEEDBACK attribute controls whether the CLK0 output, the default, or the CLK2X output is the source of the CLKFB input.

The duty cycle of the CLK0 output is 50-50 unless the DUTY_CYCLE_CORRECTION attribute is set to FALSE, in which case the duty cycle is the same as that of the CLKIN input. The duty cycle of the phase shifted outputs (CLK90, CLK180, and CLK270) is the same as that of the CLK0 output. The duty cycle of the CLK2X, CLK2X180, and CLKDV outputs is 50-50 unless CLKDV_DIVIDE is a non-integer and the DLL_FREQUENCY_MODE is High (see “CLKDV_DIVIDE,” in the *Constraints Guide* for details). The frequency of the CLKDV output is determined by the value assigned to the CLKDV_DIVIDE attribute.

DCM Clock Delay Lock Loop Outputs

Output	Description
CLK0	Clock at 1x CLKIN frequency
CLK180	Clock at 1x CLK0 frequency, shifted 180° with regards to CLK0
CLK270*	Clock at 1x CLK0 frequency, shifted 270° with regards to CLK0
CLK2X*	Clock at 2x CLK0 frequency, in phase with CLK0
CLK2X180*	Clock at 2x CLK0 frequency shifted 180° with regards to CLK2X
CLK90*	Clock at 1x CLK0 frequency, shifted 90° with regards to CLK0
CLKDV	Clock at (1/n) x CLK0 frequency, where n=CLKDV_DIVIDE value. CLKDV is in phase with CLK0.
LOCKED	All enabled DCM features locked.

* The CLK90, CLK270, CLK2X, and CLK2X180 outputs are *not* available if the DLL_FREQUENCY_MODE is set to High.

Digital Frequency Synthesizer (DFS)

The CLKFX and CLKFX180 outputs in conjunction with the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes provide a frequency synthesizer that can be any multiple or division of CLKIN. CLKFX and CLKIN are in phase every CLKFX_MULTIPLY cycles of CLKFX and every CLKFX_DIVIDE cycles of CLKIN when a feedback is provided to the CLKFB input of the DLL. The frequency of CLKFX is defined by the following equation.

$$\text{Frequency}_{\text{CLKFX}} = (\text{CLKFX_MULTIPLY_value} / \text{CLKFX_DIVIDE_value}) * \text{Frequency}_{\text{CLKIN}}$$

Both the CLKFX or CLKFX180 output can be used simultaneously.

CLKFX180 is 1x the CLKFX frequency, shifted 180° with regards to CLKFX. CLKFX and CLKFX180 always have a 50/50 duty cycle.

The DFS_FREQUENCY_MODE attribute specifies the allowable input clock and output clock frequency ranges.

The CLK_FEEDBACK attribute set to NONE will cause the DCM to be in the Digital Frequency Synthesizer mode. The CLKFX and CLKFX180 will be generated without phase correction with respect to CLKIN.

The DSEN input pin for the DCM is no longer recommended for use and should remain unconnected in the design.

Digital Phase Shifter (DPS)

The phase shift (skew) between the rising edges of CLKIN and CLKFB may be configured as a fraction of the CLKIN period with the PHASE_SHIFT attribute. This allows the phase shift to remain constant as ambient conditions change. The CLKOUT_PHASE_SHIFT attribute controls the use of the PHASE_SHIFT value. By default, the CLKOUT_PHASE_SHIFT attribute is set to NONE and the PHASE_SHIFT attribute has no effect.

By creating skew between CLKIN and CLKFB, all DCM output clocks are phase shifted by the amount of the skew.

When the CLKOUT_PHASE_SHIFT attribute is set to FIXED, the skew set by the PHASE_SHIFT attribute is used at configuration for the rising edges of CLKIN and CLKFB. The skew remains constant.

When the CLKOUT_PHASE_SHIFT attribute is set to VARIABLE, the skew set at configuration is used as a starting point and the skew value can be changed dynamically during operation using the PS* signals. This digital phase shifter feature is controlled by a synchronous interface. The inputs PSEN (phase shift enable) and PSINCDEC (phase shift increment/decrement) are set up to the rising edge of PSCLK (phase shift clock). The PSDONE (phase shift done) output is clocked with the rising edge of PSCLK (the phase shift clock). PSDONE must be connected to implement the complete synchronous interface. The rising-edge skew between CLKIN and CLKFB may be dynamically adjusted after the LOCKED output goes High.

The PHASE_SHIFT attribute value specifies the initial phase shift amount when the device is configured. Then the PHASE_SHIFT value is changed one unit when PSEN is activated for one period of PSCLK. The PHASE_SHIFT value is incremented when PSINCDEC is High and decremented when PSINCDEC is Low during the period that PSEN is High. When the DCM completes an increment or decrement operation, the PSDONE output goes High for a single PSCLK cycle to indicate the operation is complete. At this point the next change may be made. When RST (reset) is High, the PHASE_SHIFT attribute value is reset to the skew value set at configuration.

If CLKOUT_PHASE_SHIFT is FIXED or NONE, the PSEN, PSINCDEC, and PSCLK inputs must be tied to GND. The program will automatically tie the inputs to GND if they are not connected by the user.

Additional Status Bits

The STATUS output bits return the following information.

DCM Additional Status Bits

Bit	Description
0	Phase Shift Overflow* 1 = PHASE_SHIFT > 255
1	DLL CLKIN stopped** 1 = CLKIN stopped toggling
2	DLL CLKFX stopped 1 = CLKFX stopped toggling
3	No
4	No
5	No

DCM Additional Status Bits

Bit	Description
6	No
7	No

* Phase Shift Overflow will also go high if the end of the phase shift delay line is reached (see the product data sheet for the most current value of the maximum shifting delay).

** If only the DFS outputs are used (CLKFX & CLKFX180), this status bit will not go high if CLKIN stops.

LOCKED

When LOCKED is high, all enabled signals are locked.

RST

The master reset input (RST) resets DCM to its initial (power-on) state. The signal at the RST input is asynchronous and must be held High for 2ns.

Usage

This component is generally instantiated in the code as it cannot be easily inferred in synthesis tools. Some synthesis tools may allow inference via an attribute. See your synthesis tool documentation.

VHDL Instantiation Template

```
-- DCM      : In order to incorporate this function into the design,
-- VHDL     : the following instance declaration needs to be placed
-- instance  : in the architecture body of the design code. The
-- declaration : instance name (DCM_inst) and/or the port declarations
-- code      : after the "=">" assignment maybe changed to properly
--           : connect this function to the design. Unused inputs
--           : and outputs may be removed or commented out.

-- Library  : In addition to adding the instance declaration, a use
-- declaration : statement for the UNISIM.vcomponents library needs to
-- be
-- for      : added before the entity declaration. This library
-- Xilinx   : contains the component declarations for all Xilinx
-- primitives : primitives and points to the models that will be used
--           : for simulation.

-- Copy the following two statements and paste them before the
-- Entity declaration, unless they already exists.

Library UNISIM;
use UNISIM.vcomponents.all;

-- <-----Cut code below this line and paste into the architecture
-- body----->

-- DCM: Digital Clock Manager Circuit for Virtex-II/II-Pro and
-- Spartan-3
```

```

-- The current version of the Xilinx HDL Libraries Guide

DCM_inst : DCM
-- The following generics are only necessary if you wish to change
the default behavior.
generic map (
    CLKDV_DIVIDE => 2.0, -- Divide by:
    1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
    --
    7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
    CLKFX_DIVIDE => 1, -- Can be any interger from 1 to 32
    CLKFX_MULTIPLY => 4, -- Can be any integer from 1 to 32
    CLKIN_DIVIDE_BY_2 => FALSE, -- TRUE/FALSE to enable CLKIN divide
by two feature
    CLKIN_PERIOD => 0.0, -- Specify period of input clock
    CLKOUT_PHASE_SHIFT => "NONE", -- Specify phase shift of NONE,
FIXED or VARIABLE
    CLK_FEEDBACK => "1X", -- Specify clock feedback of NONE,
1X or 2X
    DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS", -- SOURCE_SYNCHRONOUS,
SYSTEM_SYNCHRONOUS or
    -- an integer from 0 to 15
    DFS_FREQUENCY_MODE => "LOW", -- HIGH or LOW frequency mode
for frequency synthesis
    DLL_FREQUENCY_MODE => "LOW", -- HIGH or LOW frequency mode
for DLL
    DUTY_CYCLE_CORRECTION => TRUE, -- Duty cycle correction, TRUE or
FALSE
    FACTORY_JF => X"C080", -- FACTORY JF Values
    PHASE_SHIFT => 0, -- Amount of fixed phase shift from -
255 to 255
    STARTUP_WAIT => FALSE) -- Delay configuration DONE until DCM
LOCK, TRUE/FALSE
port map (
    CLK0 => CLK0, -- 0 degree DCM CLK ouptput
    CLK180 => CLK180, -- 180 degree DCM CLK output
    CLK270 => CLK270, -- 270 degree DCM CLK output
    CLK2X => CLK2X, -- 2X DCM CLK output
    CLK2X180 => CLK2X180, -- 2X, 180 degree DCM CLK out
    CLK90 => CLK90, -- 90 degree DCM CLK output
    CLKDV => CLKDV, -- Divided DCM CLK out (CLKDV_DIVIDE)
    CLKFX => CLKFX, -- DCM CLK synthesis out (M/D)
    CLKFX180 => CLKFX180, -- 180 degree CLK synthesis out
    LOCKED => LOCKED, -- DCM LOCK status output
    PSDONE => PSDONE, -- Dynamic phase adjust done output
    STATUS => STATUS, -- 8-bit DCM status bits output
    CLKFB => CLKFB, -- DCM clock feedback
    CLKIN => CLKIN, -- Clock input (from IBUFG, BUFG or DCM)
    PSCLK => PSCLK, -- Dynamic phase adjust clock input
    PSEN => PSEN, -- Dynamic phase adjust enable input
    PSINCDEC => PSINCDEC, -- Dynamic phase adjust increment/decrement
    RST => RST -- DCM asynchronous reset input
);

-- End of DCM_inst instantiation

```

Verilog Instantiation Template

```
//      DCM      : In order to incorporate this function into the design,
// Verilog      : the following instance declaration needs to be placed
// instance     : in the body of the design code. The instance name
// declaration  : (DCM_inst) and/or the port declarations within the
// code         : parenthesis maybe changed to properly reference and
//              : connect this function to the design. Unused inputs
//              : and outputs may be removed or commented out.

// <-----Cut code below this line----->

// DCM: Digital Clock Manager Circuit for Virtex-II/II-Pro and
// Spartan-3
// The current version of the Xilinx HDL Libraries Guide

DCM DCM_inst (
    .CLK0(CLK0),      // 0 degree DCM CLK ouptput
    .CLK180(CLK180), // 180 degree DCM CLK output
    .CLK270(CLK270), // 270 degree DCM CLK output
    .CLK2X(CLK2X),   // 2X DCM CLK output
    .CLK2X180(CLK2X180), // 2X, 180 degree DCM CLK out
    .CLK90(CLK90),   // 90 degree DCM CLK output
    .CLKDV(CLKDV),   // Divided DCM CLK out (CLKDV_DIVIDE)
    .CLKFX(CLKFX),   // DCM CLK synthesis out (M/D)
    .CLKFX180(CLKFX180), // 180 degree CLK synthesis out
    .LOCKED(LOCKED), // DCM LOCK status output
    .PSDONE(PSDONE), // Dynamic phase adjust done output
    .STATUS(STATUS), // 8-bit DCM status bits output
    .CLKFB(CLKFB),   // DCM clock feedback
    .CLKIN(CLKIN),   // Clock input (from IBUFG, BUFG or DCM)
    .PSCLK(PSCLK),   // Dynamic phase adjust clock input
    .PSEN(PSEN),     // Dynamic phase adjust enable input
    .PSINCDEC(PSINCDEC), // Dynamic phase adjust increment/decrement
    .RST(RST)        // DCM asynchronous reset input
);

// The following defparam declarations are only necessary if you
// wish to change the default behavior
// of the DCM. If the instance name is changed, these defparams need
// to be updated accordingly.

defparam DCM_inst.CLKDV_DIVIDE = 2.0; // Divide by:
1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
//
7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
defparam DCM_inst.CLKFX_DIVIDE = 1; // Can be any interger from 1
to 32
defparam DCM_inst.CLKFX_MULTIPLY = 4; // Can be any integer from 2
to 32
defparam DCM_inst.CLKIN_DIVIDE_BY_2 = "FALSE"; // TRUE/FALSE to
enable CLKIN divide by two feature
defparam DCM_inst.CLKIN_PERIOD = 0.0; // Specify period of input
clock
defparam DCM_inst.CLKOUT_PHASE_SHIFT = "NONE"; // Specify phase
shift of NONE, FIXED or VARIABLE
defparam DCM_inst.CLK_FEEDBACK = "1X"; // Specify clock feedback of
NONE, 1X or 2X
```



```
defparam DCM_inst.DESKEW_ADJUST = "SYSTEM_SYNCHRONOUS"; //
SOURCE_SYNCHRONOUS, SYSTEM_SYNCHRONOUS or
// an
integer from 0 to 15
defparam DCM_inst.DFS_FREQUENCY_MODE = "LOW"; // HIGH or LOW
frequency mode for frequency synthesis
defparam DCM_inst.DLL_FREQUENCY_MODE = "LOW"; // HIGH or LOW
frequency mode for DLL
defparam DCM_inst.DUTY_CYCLE_CORRECTION = "TRUE"; // Duty cycle
correction, TRUE or FALSE
defparam DCM_inst.FACTORY_JF = 16'hC080; // FACTORY JF Values
defparam DCM_inst.PHASE_SHIFT = 0; // Amount of fixed phase
shift from -255 to 255
defparam DCM_inst.STARTUP_WAIT = "FALSE"; // Delay configuration
DONE until DCM LOCK, TRUE/FALSE

// End of DCM_inst instantiation
```

Commonly Used Constraints

CLKDV_DIVIDE, CLK_FEEDBACK, CLKFX_DIVIDE, CLKFX_MULTIPLY,
CLKIN_DIVIDE_BY_2, CLKOUT_PHASE_SHIFT, DUTY_CYCLE_CORRECTION,
DFS_FREQUENCY_MODE, DLL_FREQUENCY_MODE, LOC, PHASE_SHIFT,
STARTUP_WAIT, DESKEW_ADJUST

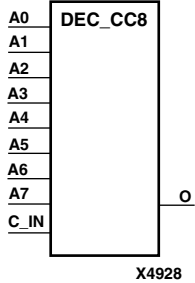
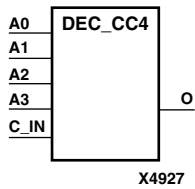
DEC_CC4, 8, 16

4-, 8-, 16-Bit Active Low Decoders

Architectures Supported

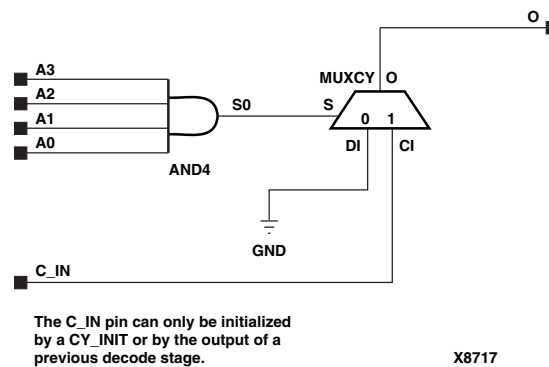
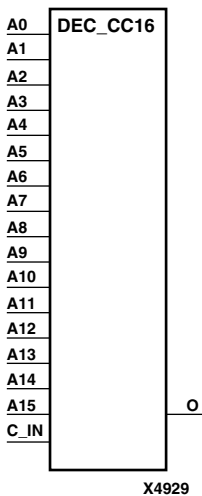
DEC_CC4, DEC_CC8, DEC_CC16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

These decoders are used to build wide-decoder functions. They are implemented by cascading CY_MUX elements driven by lookup tables (LUTs). The C_IN pin can only be driven by the output (O) of a previous decode stage. When one or more of the inputs (A) are Low, the output is Low. When all the inputs are High and the C_IN input is High, the output is High. You can decode patterns by adding inverters to inputs.

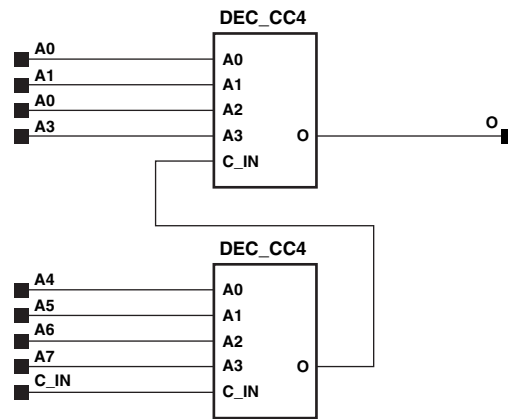


Inputs					Outputs
A0	A1	...	Az	C_IN	O
1	1	1	1	1	1
X	X	X	X	0	0
0	X	X	X	X	0
X	0	X	X	X	0
X	X	X	0	X	0

z = 3 for DEC_CC4; z = 7 for DEC_CC8; z = 15 for DEC_CC16



DEC_CC4 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



The C_IN pin can only be initialized by a CY_INIT or by the output of a previous decode stage.

X6396

DEC_CC8 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

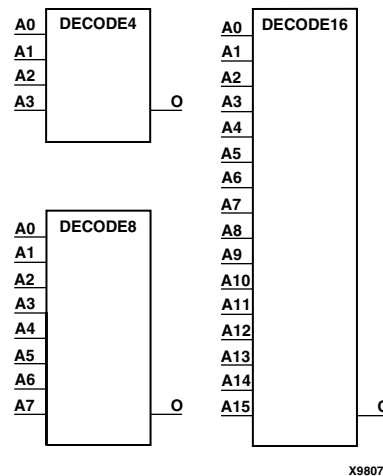
DEC_CC4 cannot be directly inferred or instantiated. The proper way to use a DEC_CC4 is to infer the primitive components that make up the DEC_CC4.

DECODE4, 8, 16

4-, 8-, 16-Bit Active-Low Decoders

Architectures Supported

DECODE4, DECODE8, DECODE16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



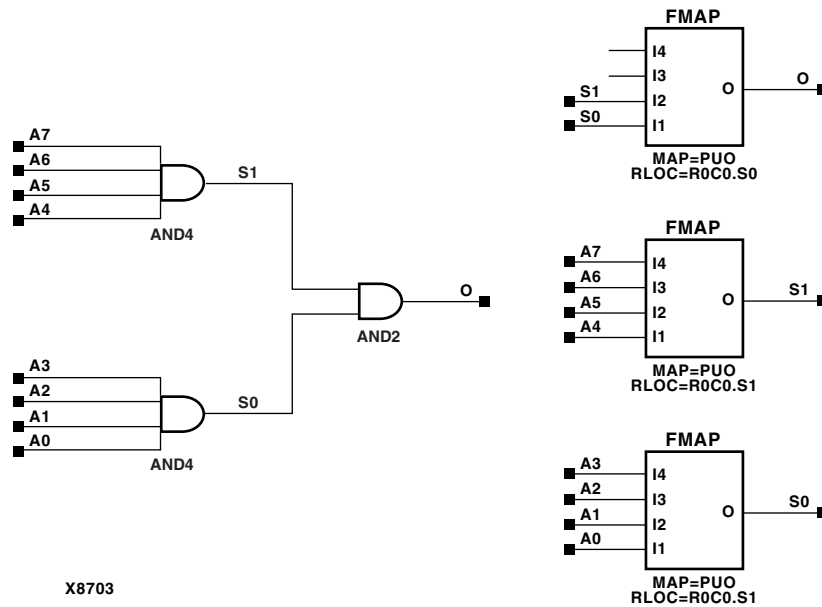
DECODE Representations

In Spartan-II, , Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X decoders are implemented using combinations of LUTs and MUXCYs.

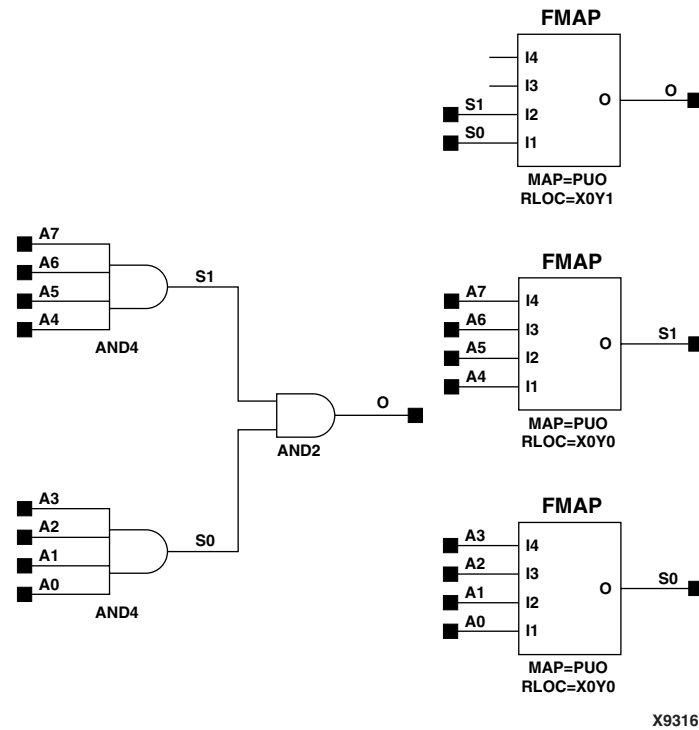
Inputs				Outputs*
A0	A1	...	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

z = 3 for DECODE4, z = 7 for DECODE8; z = 15 for DECODE16

*A pull-up resistor must be connected to the output to establish High-level drive current.



DECODE8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



DECODE8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, these design elements are inferred rather than instantiated.

DECODE32, 64

32- and 64-Bit Active-Low Decoders

Architectures Supported

DECODE32, DECODE64	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



DECODE32 and DECODE64 are 32- and 64-bit active-low decoders. These decoders are implemented using combinations of LUTs and MUXCYs.

See “[DECODE4, 8, 16](#)” for a representative schematic.

Inputs				Outputs
A0	A1	...	Az	O
1	1	1	1	1
0	X	X	X	0
X	0	X	X	0
X	X	X	0	0

$z = 31$ for DECODE32, $z = 63$ for DECODE64

Usage

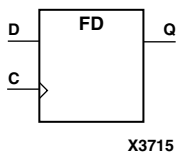
For HDL, these design elements are inferred rather than instantiated.

FD

D Flip-Flop

Architectures Supported

FD	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FD is a single D-type flip-flop with data input (D) and data output (Q). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

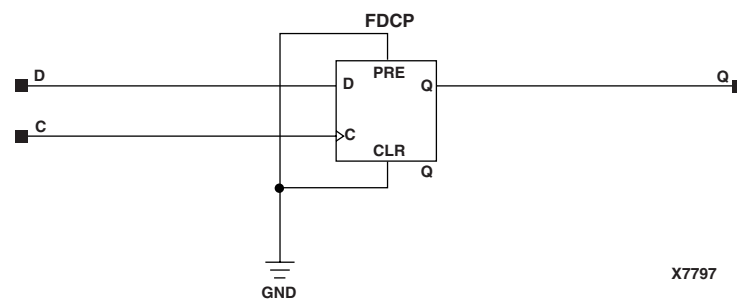
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

See “FD4, 8, 16” for information on multiple D flip-flops for XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



FD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FD should be placed
-- after architecture statement but before begin keyword

component FD
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for FD
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FD_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FD should be placed
-- in architecture after the begin keyword

FD_INSTANCE_NAME : FD
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           D => user_D);
```

Verilog Instantiation Template

```
FD FD_instance_name (.Q (user_Q),
                    .C (user_C),
                    .D (user_D));

defparam FD_instance_name.INIT = bit_value;
```

Commonly Used Constraints

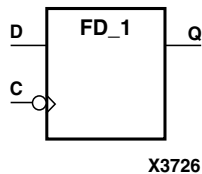
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FD_1

D Flip-Flop with Negative-Edge Clock

Architectures Supported

FD_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FD_1 is a single D-type flip-flop with data input (D) and data output (Q). The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FD_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FD_1
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FD_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FD_1_instance_name : label is "0";
-- values can be (0 or 1)

begin

FD_1_INSTANCE_NAME : FD_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value) -- INIT value can be '0' or '1'
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D);
end Behavioral
```

Verilog Instantiation Template

```
FD_1 FD_1_instance_name (.Q (user_Q),
                        .C (user_C),
                        .D (user_D));

defparam FD_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

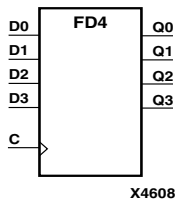
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
and XBLKNM

FD4, 8, 16

Multiple D Flip-Flops

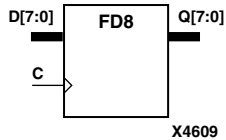
Architectures Supported

FD4, FD8, FD16	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



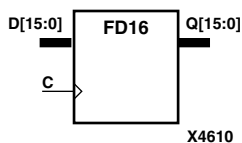
FD4, FD8, FD16 are multiple D-type flip-flops with data inputs (D) and data outputs (Q). FD4, FD8, and FD16 are, respectively, 4-bit, 8-bit, and 16-bit registers, each with a common clock (C). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.

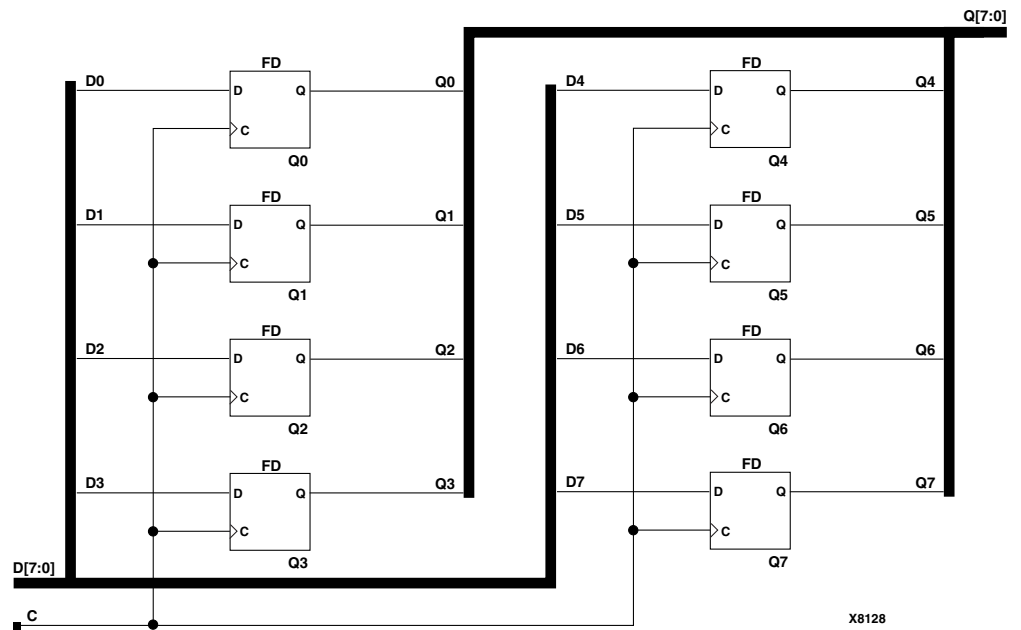
The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



Inputs		Outputs
Dz – D0	C	Qz – Q0
0	↑	0
1	↑	1

z = 3 for FD4; z = 7 for FD8; z = 15 for FD16





FD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

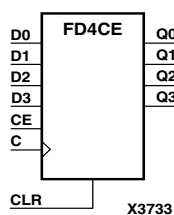
For HDL, these design elements are inferred rather than instantiated.

FD4CE, FD8CE, FD16CE

4-, 8-, 16-Bit Data Registers with Clock Enable and Asynchronous Clear

Architectures Supported

FD4CE, FD8CE, FD16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



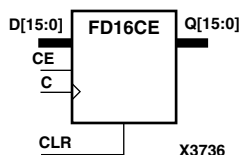
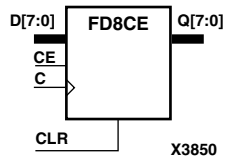
FD4CE, FD8CE, and FD16CE are, respectively, 4-, 8-, and 16-bit data registers with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data outputs (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

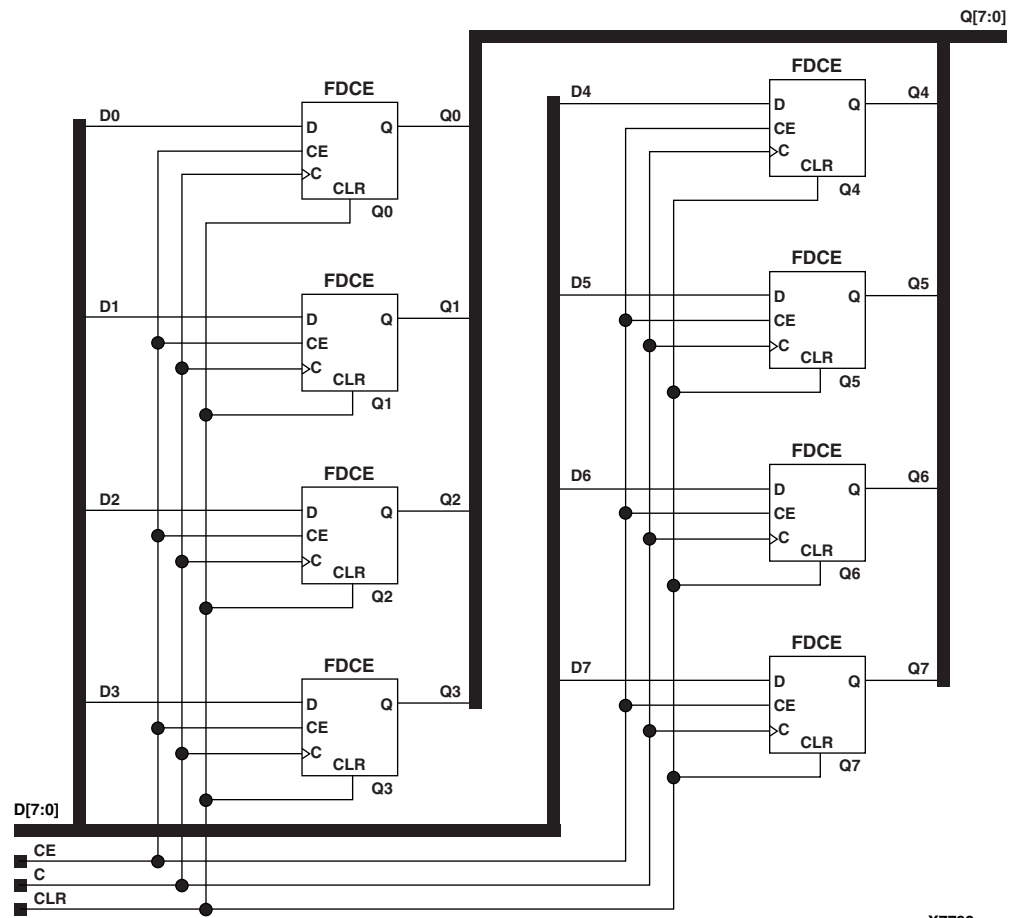
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



Inputs				Outputs
CLR	CE	Dz – D0	C	Qz – Q0
1	X	X	X	0
0	0	X	X	No Chg
0	1	Dn	↑	Dn

z = 3 for FD4CE; z = 7 for FD8CE; z = 15 for FD16CE.



X7799

FD8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-II-E, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

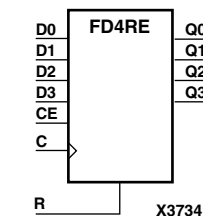
For HDL, these design elements are inferred rather than instantiated.

FD4RE, FD8RE, FD16RE

4-, 8-, 16-Bit Data Registers with Clock Enable and Synchronous Reset

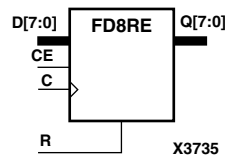
Architectures Supported

FD4RE, FD8RE, FD16RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FD4RE, FD8RE, and FD16RE are, respectively, 4-, 8-, and 16-bit data registers. When the clock enable (CE) input is High, and the synchronous reset (R) input is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High clock (C) transition. When R is High, it overrides all other inputs and resets the data outputs (Q) Low on the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

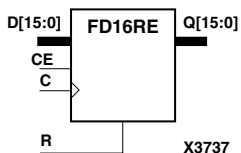
The flip-flops are asynchronously cleared, output Low, when power is applied.



For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

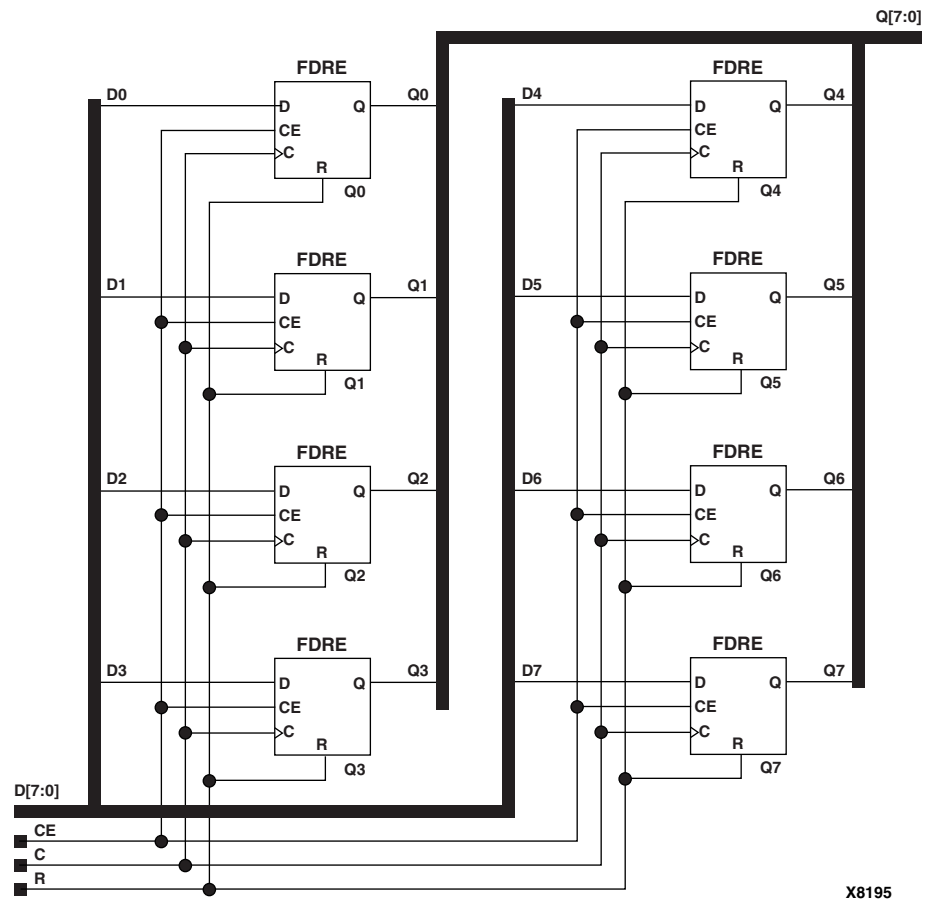
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



Inputs				Outputs
R	CE	Dz – D0	C	Qz – Q0
1	X	X	↑	0
0	0	X	X	No Chg
0	1	Dn	↑	Dn

z = 3 for FD4RE; z = 7 for FD8RE; z = 15 for FD16RE



X8195

FD8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

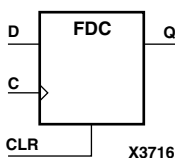
For HDL, these design elements are inferred rather than instantiated.

FDC

D Flip-Flop with Asynchronous Clear

Architectures Supported

FDC	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDC is a single D-type flip-flop with data (D) and asynchronous clear (CLR) inputs and data output (Q). The asynchronous CLR, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low on the Low-to-High clock transition.

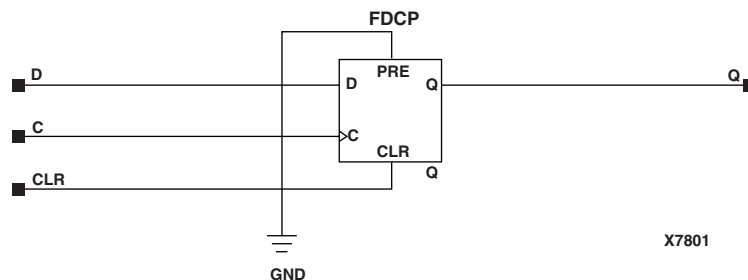
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↑	1
0	0	↑	0



FDC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDC should be placed
-- after architecture statement but before begin keyword

component FDC
  -- synthesis translate_off
  generic (INIT : bit:= '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDC
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDC_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDC should be placed
-- in architecture after the begin keyword

FDC_INSTANCE_NAME : FDC
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            CLR => user_CLR,
            D => user_D);
```

Verilog Instantiation Template

```
FDC FDC_instance_name (.Q (user_Q),
                      .C (user_C),
                      .CLR (user_CLR),
                      .D (user_D));

defparam FDC_instance_name.INIT = bit_value;
```

Commonly Used Constraints

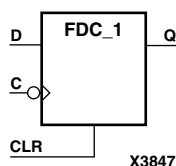
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDC_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Clear

Architectures Supported

FDC_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDC_1 is a single D-type flip-flop with data input (D), asynchronous clear input (CLR), and data output (Q). The asynchronous CLR, when active, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↓	1
0	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDC_1 should be placed
-- after architecture statement but before begin keyword

component FDC_1
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
```

```
        C : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDC_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDC_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDC_1 should be placed
-- in architecture after the begin keyword

FDC_1_INSTANCE_NAME : FDC_1
-- synthesis translate_off
generic map (INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
         C => user_C,
         CLR => user_CLR,
         D => user_D);
```

Verilog Instantiation Template

```
FDC_1 FDC_1_instance_name (.Q (user_Q),
                          .C (user_C),
                          .CLR (user_CLR),
                          .D (user_D));

defparam FDC_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

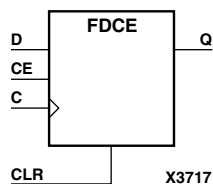
BLKNM, HBLKNM, U_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
and XBLKNM

FDCE

D Flip-Flop with Clock Enable and Asynchronous Clear

Architectures Supported

FDCE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDCE is a single D-type flip-flop with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data input (D) of FDCE is transferred to the corresponding data output (Q) during the Low-to-High clock (C) transition. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flops primitives may take advantage of the clock-enable p-term.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDCE: Single Data Rate D Flip-Flop with Asynchronous Clear and
--       Clock Enable (posedge clk). All families.
-- The current version of the Xilinx HDL Libraries Guide

    FDCE_inst : FDCE
    port map (
        Q => Q,      -- Data output
        C => C,      -- Clock input
        CE => CE,    -- Clock enable input
        CLR => CLR,  -- Asynchronous clear input
        D => D       -- Data input
    );

-- End of FDCE_inst instantiation
```

Verilog Instantiation Template

```
// FDCE: Single Data Rate D Flip-Flop with Asynchronous Clear and
//       Clock Enable (posedge clk). All families.
// The current version of the Xilinx HDL Libraries Guide

    FDCE FDCE_inst (
        .Q(Q),      // Data output
        .C(C),      // Clock input
        .CE(CE),    // Clock enable input
        .CLR(CLR),  // Asynchronous clear input
        .D(D)       // Data input
    );

// The following defparam declaration is only necessary if you wish to
// change the initial value of the register to a one. If the instance
// name to the FDCE is changed, that change needs to be reflected in
// the defparam statements.

    defparam FDCE_inst.INIT = 1'b0;

// End of FDCE_inst instantiation
```

Commonly Used Constraints

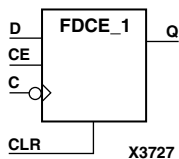
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Clear

Architectures Supported

FDCE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDCE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous clear (CLR) inputs, and data output (Q). The asynchronous CLR input, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	↓	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDCE_1: Single Data Rate D Flip-Flop with Asynchronous Clear and
--         Clock Enable (negedge clock). All families.
-- The current version of the Xilinx HDL Libraries Guide
```

```
FDCE_1_inst : FDCE_1
```

```
port map (  
    Q => Q,      -- Data output  
    C => C,      -- Clock input  
    CE => CE,    -- Clock enable input  
    CLR => CLR,  -- Asynchronous clear input  
    D => D       -- Data input  
);  
  
-- End of FDCE_1_inst instantiation
```

Verilog Instantiation Template

```
// FDCE_1: Single Data Rate D Flip-Flop with Asynchronous Clear and  
//         Clock Enable (negedge clock). All families.  
// The current version of the Xilinx HDL Libraries Guide  
  
FDCE_1 FDCE_1_inst (  
    .Q(Q),      // Data output  
    .C(C),      // Clock input  
    .CE(CE),    // Clock enable input  
    .CLR(CLR),  // Asynchronous clear input  
    .D(D)       // Data input  
);  
  
// The following defparam declaration is only necessary if you wish to  
// change the initial value of the register to a one. If the instance  
// name to the FDCE_1 is changed, that change needs to be reflected in  
// the defparam statements.  
  
defparam FDCE_1_inst.INIT = 1'b0;  
  
// End of FDCE_1_inst instantiation
```

Commonly Used Constraints

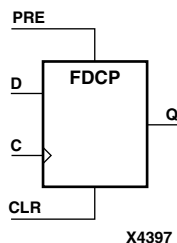
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
XBLKNM

FDCP

D Flip-Flop Asynchronous Preset and Clear

Architectures Supported

FDCP	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDCP is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↑	0
0	0	1	↑	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDCP should be placed
-- after architecture statement but before begin keyword
```

```
component FDCP
```

```

-- synthesis translate_off
generic (INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      C : in STD_ULOGIC;
      CLR : in STD_ULOGIC;
      D : in STD_ULOGIC;
      PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCP_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDCP should be placed
-- in architecture after the begin keyword

FDCP_INSTANCE_NAME : FDCP
-- synthesis translate_off
generic map (INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          CLR => user_CLR,
          D => user_D,
          PRE => user_PRE);

```

Verilog Instantiation Template

```

FDCP FDCP_instance_name (.Q (user_Q),
                        .C (user_C),
                        .CLR (user_CLR),
                        .D (user_D),
                        .PRE (user_PRE));

defparam FDCP_instance_name.INIT = bit_value;

```

Commonly Used Constraints

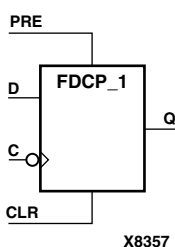
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset and Clear

Architectures Supported

FDCP_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDCP_1 is a single D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↓	0
0	0	1	↓	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDCP_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FDCP_1
  -- synthesis translate_off
```

```

        generic (INIT : bit := '1');
        -- synthesis translate_on
    port (Q : out STD_ULOGIC;
          C : in STD_ULOGIC;
          CLR : in STD_ULOGIC;
          D : in STD_ULOGIC;
          PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDCP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDCP_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDCP_1 should be placed
-- in architecture after the begin keyword

FDCP_1_INSTANCE_NAME : FDCP_1
    -- synthesis translate_off
    generic map (INIT => bit_value)
    -- synthesis translate_on
    port map (Q => user_Q,
              C => user_C,
              CLR => user_CLR,
              D => user_D,
              PRE => user_PRE);

```

Verilog Instantiation Template

```

FDCP_1 FDCP_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CLR (user_CLR),
                             .D (user_D),
                             .PRE (user_PRE));

defparam FDCP_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

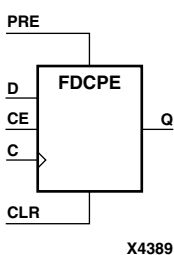
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDCPE

D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Architectures Supported

FDCPE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDCPE is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

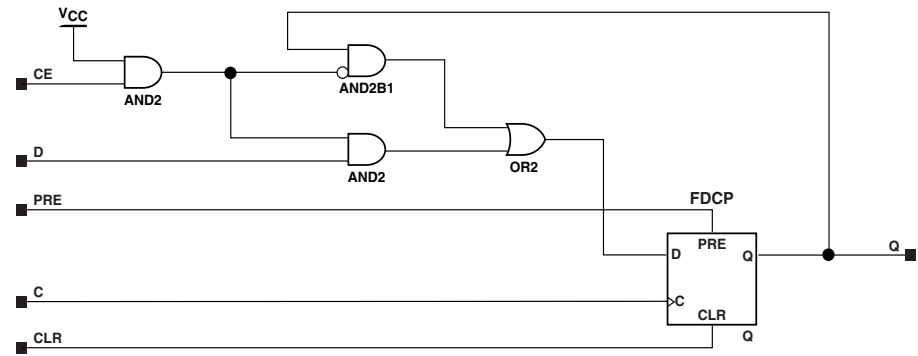
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↑	0
0	0	1	1	↑	1



X7804

FDCPE Implementation XC9500/XV/XL, CoolRunner XPLA3

Usage

Below are example templates for instantiating this component into a design. These end

VHDL Instantiation Template

```
-- FDCPE: Single Data Rate D Flip-Flop with Asynchronous Clear,
--       Set and Clock Enable (posedge clk). All families.
-- The current version of the Xilinx HDL Libraries Guide
```

```
FDCPE_inst : FDCPE
port map (
    Q => Q,      -- Data output
    C => C,      -- Clock input
    CE => CE,    -- Clock enable input
    CLR => CLR,  -- Asynchronous clear input
    D => D,      -- Data input
    PRE => PRE   -- Asynchronous set input
);
```

```
-- End of FDCPE_inst instantiation
```

Verilog Instantiation Template

```
// FDCPE: Single Data Rate D Flip-Flop with Asynchronous Clear, Set and
//       Clock Enable (posedge clk). All families.
// The current version of the Xilinx HDL Libraries Guide
```

```
FDCPE FDCPE_inst (
    .Q(Q),      // Data output
    .C(C),      // Clock input
    .CE(CE),    // Clock enable input
    .CLR(CLR),  // Asynchronous clear input
    .D(D),      // Data input
    .PRE(PRE)   // Asynchronous set input
);
```

```
// The following defparam declaration is only necessary if you wish to
```



```
// change the initial value of the register to a one.  If the instance
// name to the FDCPE is changed, that change needs to be reflected in
// the defparam statements.
```

```
    defparam FDCPE_inst.INIT = 1'b0;
```

```
// End of FDCPE_inst instantiation
```

Commonly Used Constraints

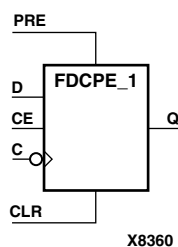
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
and XBLKNM.

FDCPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset and Clear

Architectures Supported

FDCPE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDCPE_1 is a single D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↓	0
0	0	1	1	↓	1

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDCPE_1: Single Data Rate D Flip-Flop with Asynchronous Clear,
--           Set and Clock Enable (negedge clock). All families.
-- The current version of the Xilinx HDL Libraries Guide
```

```

FDCPE_1_inst : FDCPE_1
port map (
    Q => Q,          -- Data output
    C => C,          -- Clock input
    CE => CE,       -- Clock enable input
    CLR => CLR,     -- Asynchronous clear input
    D => D,         -- Data input
    PRE => PRE     -- Asynchronous set input
);

-- End of FDCPE_1_inst instantiation

```

Verilog Instantiation Template

```

// FDCPE: Single Data Rate D Flip-Flop with Asynchronous Clear, Set and
//          Clock Enable (posedge clk). All families.
// The current version of the Xilinx HDL Libraries Guide

FDCPE_1 FDCPE_1_inst (
    .Q(Q),          // Data output
    .C(C),          // Clock input
    .CE(CE),       // Clock enable input
    .CLR(CLR),     // Asynchronous clear input
    .D(D),         // Data input
    .PRE(PRE)     // Asynchronous set input
);

// The following defparam declaration is only necessary if you wish to
// change the initial value of the register to a one. If the instance
// name to the FDCPE_1 is changed, that change needs to be reflected in
// the defparam statements.

defparam FDCPE_1_inst.INIT = 1'b0;

// End of FDCPE_1_inst instantiation

```

Commonly Used Constraints

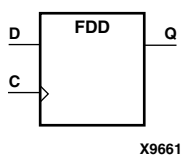
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDD

Dual Edge Triggered D Flip-Flop

Architectures Supported

FDD	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive

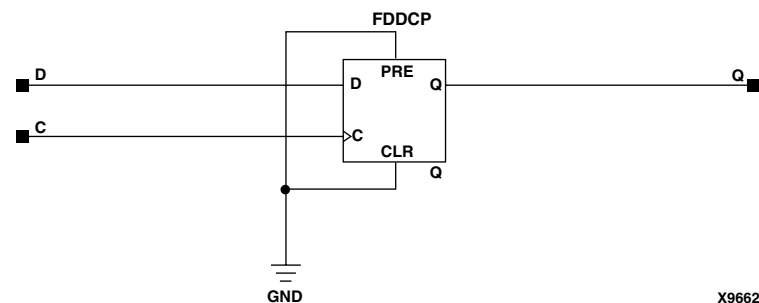


FDD is a single dual edge triggered D-type flip-flop with data input (D) and data output (Q). The data on the D input is loaded into the flip-flop during the Low-to-High and the High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

See [FDD4,8,16](#) for information on multiple D flip-flops for CoolRunner-II.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1
0	↓	0
1	↓	1



FDD Implementation CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDD should be placed
-- after architecture statement but before begin keyword

component FDD
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDD
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDD_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDD should be placed
-- in architecture after the begin keyword

FDD_INSTANCE_NAME : FDD
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           D => user_D);
```

Verilog Instantiation Template

```
FDD FDD_instance_name (.Q (user_Q),
                       .C (user_C),
                       .D (user_D));

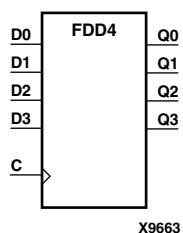
defparam FDD_instance_name.INIT = bit_value;
```

FDD4,8,16

Multiple Dual Edge Triggered D Flip-Flops

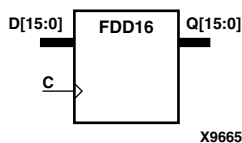
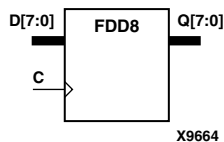
Architectures Supported

FDD4, FDD8, FDD16	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



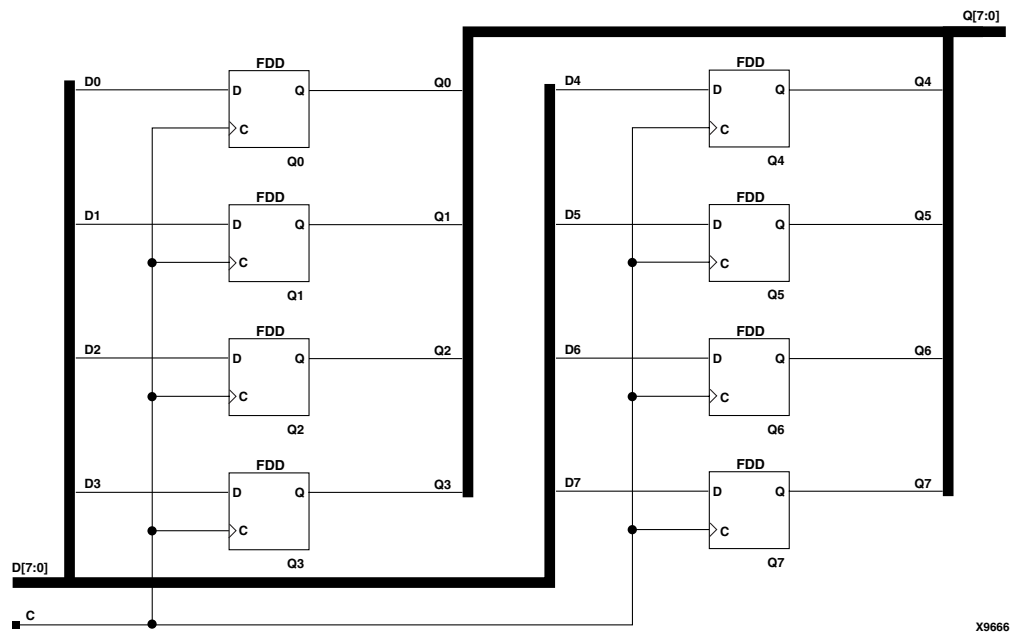
FDD4, FDD8, FDD16 are multiple dual edge triggered D-type flip-flops with data inputs (D) and data outputs (Q). FDD4, FDD8, and FDD16 are, respectively, 4-bit, 8-bit, and 16-bit registers, each with a common clock (C). The data on the D inputs is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.



Inputs		Outputs
Dz – D0	C	Qz – Q0
0	↑	0
1	↑	1
0	↓	0
1	↓	1

z = 3 for FDD4; z = 7 for FDD8; z = 15 for FDD16



X9666

FDD8 Implementation CoolRunner-II

Usage

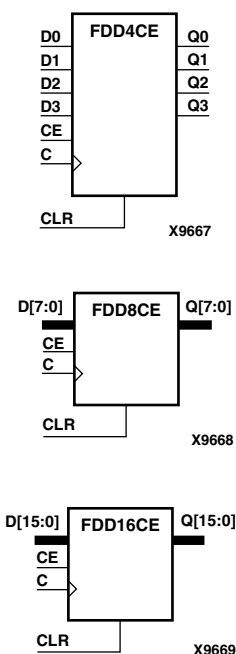
For HDL, these deign elements are inferred rather than instantiated.

FDD4CE, FDD8CE, FDD16CE

4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Asynchronous Clear

Architectures Supported

FDD4CE, FDD8CE, FDD16CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive

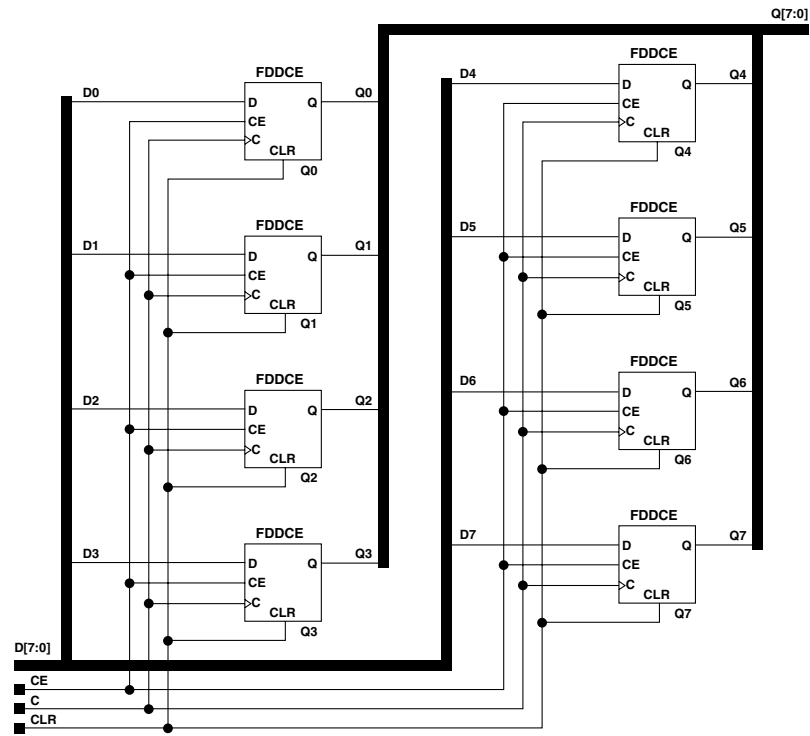


FDD4CE, FDD8CE, and FDD16CE are, respectively, 4-, 8-, and 16-bit data registers with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q) during the Low-to-High and High-to-Low clock (C) transitions. When CLR is High, it overrides all other inputs and resets the data outputs (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	CE	Dz – D0	C	Qz – Q0
1	X	X	X	0
0	0	X	X	No Chg
0	1	Dn	↑	Dn
0	1	Dn	↓	Dn

z = 3 for FDD4CE; z = 7 for FDD8CE; z = 15 for FDD16CE.



X9670

FDD8CE Implementation CoolRunner-II

Usage

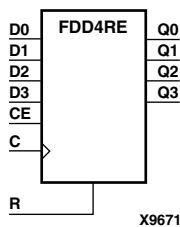
For HDL, these design elements are inferred rather than instantiated.

FDD4RE, FDD8RE, FDD16RE

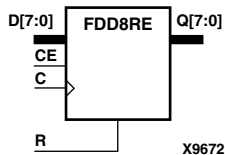
4-, 8-, 16-Bit Dual Edge Triggered Data Registers with Clock Enable and Synchronous Reset

Architectures Supported

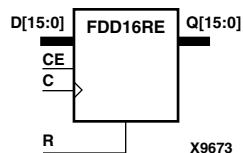
FDD4RE, FDD8RE, FDD16RE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDD4RE, FDD8RE, and FDD16RE are, respectively, 4-, 8-, and 16-bit data registers. When the clock enable (CE) input is High, and the synchronous reset (R) input is Low, the data on the data inputs (D) is transferred to the corresponding data outputs (Q0) during the Low-to-High or High-to-Low clock (C) transition. When R is High, it overrides all other inputs and resets the data outputs (Q) Low on the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

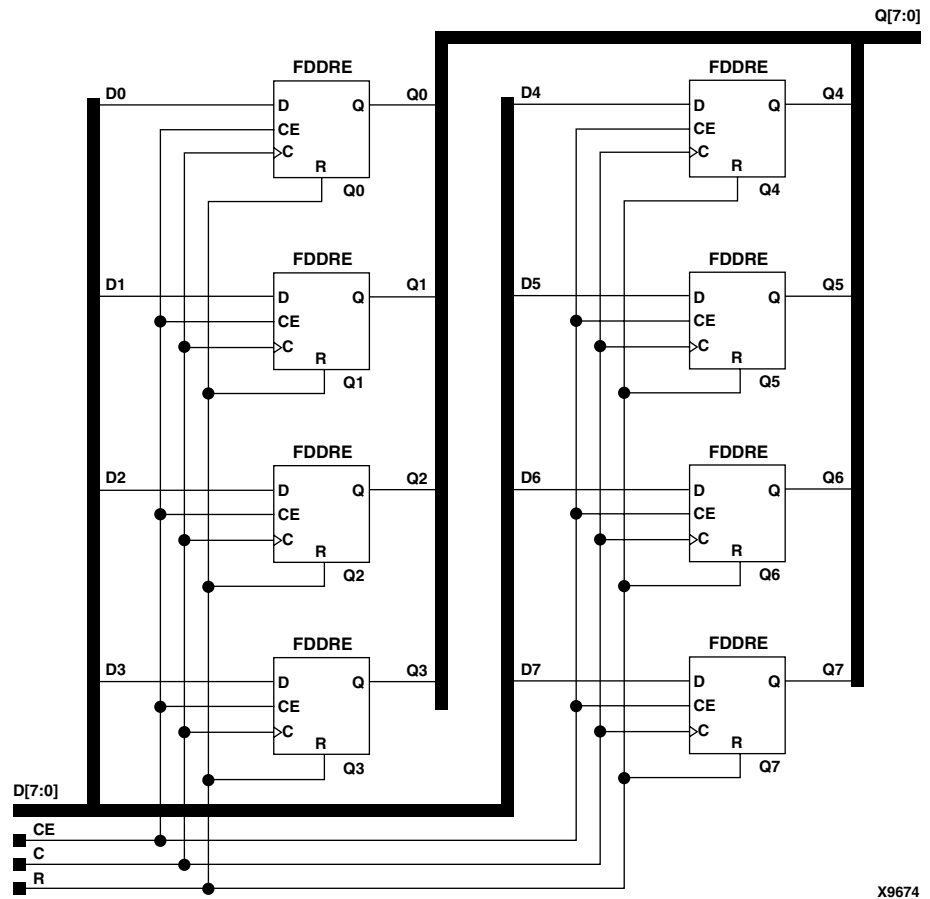


The flip-flops are asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.



Inputs				Outputs
R	CE	Dz – D0	C	Qz – Q0
1	X	X	↑	0
1	X	X	↓	0
0	0	X	X	No Chg
0	1	Dn	↑	Dn
0	1	Dn	↓	Dn

z = 3 for FDD4RE; z = 7 for FDD8RE; z = 15 for FDD16RE



X9674

FDD8RE Implementation CoolRunner-II

Usage

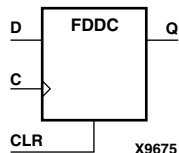
For HDL, these design elements are inferred rather than instantiated.

FDDC

D Dual Edge Triggered Flip-Flop with Asynchronous Clear

Architectures Supported

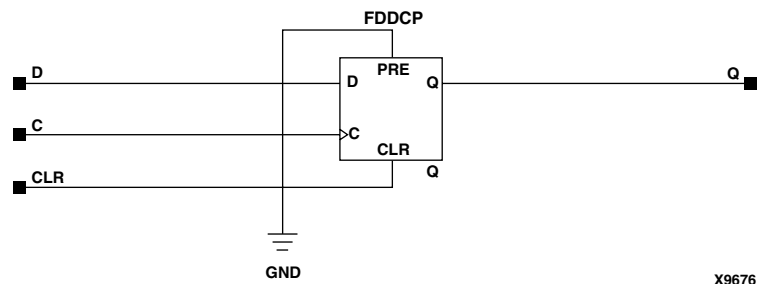
FDDC	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDC is a single dual edge triggered D-type flip-flop with data (D) and asynchronous clear (CLR) inputs and data output (Q). The asynchronous CLR, when High, overrides all other inputs and sets the Q output Low. The data on the D input is loaded into the flip-flop when CLR is Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
CLR	D	C	Q
1	X	X	0
0	1	↑	1
0	1	↓	1
0	0	↑	0
0	0	↓	0



FDDC Implementation CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDDC should be placed
-- after architecture statement but before begin keyword
```

```
component FDDC
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDDC
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDDC_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDDC should be placed
-- in architecture after the begin keyword
```

```
FDDC_INSTANCE_NAME : FDDC
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           CLR => user_CLR,
           D => user_D);
```

Verilog Instantiation Template

```
FDDC FDDC_instance_name (.Q (user_Q),
                        .C (user_C),
                        .CLR (user_CLR),
                        .D (user_D));
```

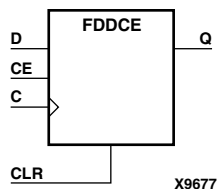
```
defparam FDDC_instance_name.INIT = bit_value;
```

FDDCE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Clear

Architectures Supported

FDDCE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDCE is a single dual edge triggered D-type flip-flop with clock enable and asynchronous clear. When clock enable (CE) is High and asynchronous clear (CLR) is Low, the data on the data input (D) of FDDCE is transferred to the corresponding data output (Q) during the Low-to-High and High-to-Low clock (C) transitions. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDDCE and FDDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDDCE should be placed
-- after architecture statement but before begin keyword

component FDDCE
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDCE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FDDCE should be placed
-- in architecture after the begin keyword

FDDCE_INSTANCE_NAME : FDDCE
port map (Q => user_Q,
          C => user_C,
          CE => user_CE,
          CLR => user_CLR,
          D => user_D);
```

Verilog Instantiation Template

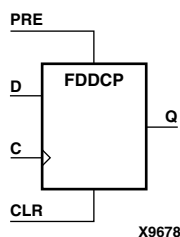
```
FDDCE FDDCE_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .CLR (user_CLR),
                             .D (user_D));
```


FDDCP

Dual Edge Triggered D Flip-Flop Asynchronous Preset and Clear

Architectures Supported

FDDCP	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDCP is a single dual edge triggered D-type flip-flop with data (D), asynchronous preset (PRE) and clear (CLR) inputs, and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	D	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	↑	0
0	0	1	↑	1
0	0	0	↓	0
0	0	1	↓	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDDCP should be placed
-- after architecture statement but before begin keyword
```

```
component FDDCP
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
```

```
    port (Q : out STD_ULOGIC;
          C : in STD_ULOGIC;
          CLR : in STD_ULOGIC;
          D : in STD_ULOGIC;
          PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDCP_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDDCP should be placed
-- in architecture after the begin keyword

FDDCP_INSTANCE_NAME : FDDCP
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            CLR => user_CLR,
            D => user_D,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
FDDCP FDDCP_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CLR (user_CLR),
                           .D (user_D),
                           .PRE (user_PRE));

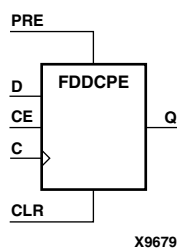
defparam FDDCP_instance_name.INIT = bit_value;
```

FDDCPE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Architectures Supported

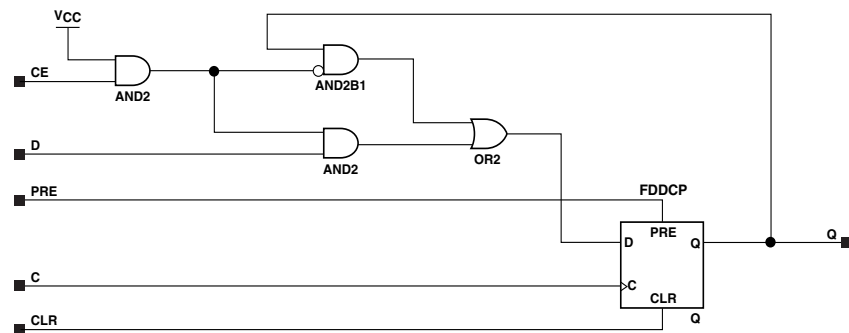
FDDCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDCPE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	CE	D	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	↑	0
0	0	1	1	↑	1
0	0	1	0	↓	0
0	0	1	1	↓	1



X9680

FDDCPE Implementation CoolRunner-II

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDDCPE: Double Data Rate Register with Asynchronous Clear and Set
--           and Clock Enable (Clear has priority). CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

FDDCPE_inst : FDDCPE
port map (
    Q => Q,      -- Data output
    C => C,      -- Clock input
    CE => CE,    -- Clock enable input
    CLR => CLR,  -- Asynchronous clear input
    D => D,      -- Data input
    PRE => PRE  -- Asynchronous set input
);

-- End of FDDCPE_inst instantiation
```

Verilog Instantiation Template

```
// FDDCPE: Double Data Rate Register with Asynchronous Clear and Set
//           and Clock Enable (Clear has priority). CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

FDDCPE FDDCPE_inst (
    .Q(Q),      // Data output
    .C(C),      // Clock input
    .CE(CE),    // Clock enable input
    .CLR(CLR),  // Asynchronous clear input
    .D(D),      // Data input
    .PRE(PRE)   // Asynchronous set input
);

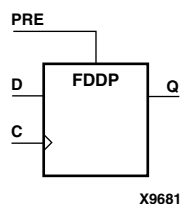
// End of FDDCPE_inst instantiation
```

FDDP

Dual Edge Triggered D Flip-Flop with Asynchronous Preset

Architectures Supported

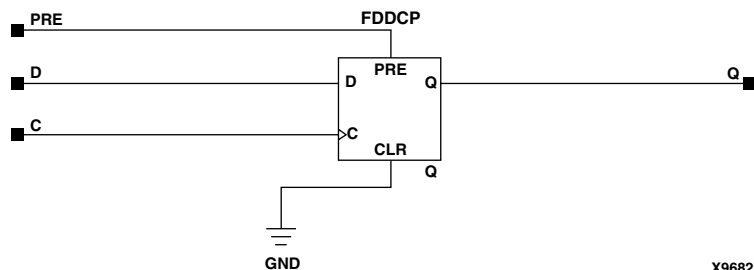
FDDP	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDP is a single dual edge triggered D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↑	1	1
0	↑	0	0
0	↓	1	1
0	↓	0	0



FDDP Implementation CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- Component Declaration for FDDP should be placed
-- after architecture statement but before begin keyword

component FDDP
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDP_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDDP should be placed
-- in architecture after the begin keyword

FDDP_INSTANCE_NAME : FDDP
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           D => user_D,
           PRE => user_PRE);
```

Verilog Instantiation Template

```
FDDP FDDP_instance_name (.Q (user_Q),
                        .C (user_C),
                        .D (user_D),
                        .PRE (user_PRE));

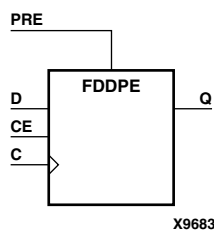
defparam FDDP_instance_name.INIT = bit_value;
```

FDDPE

Dual Edge Triggered D Flip-Flop with Clock Enable and Asynchronous Preset

Architectures Supported

FDDPE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDPE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDDCE and FDDPE flip-flops primitives may take advantage of the clock-enable p-term.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	↑	0
0	1	1	↑	1
0	1	0	↓	0
0	1	1	↓	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDDPE should be placed
-- after architecture statement but before begin keyword

component FDDPE
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDPE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FDDPE should be placed
-- in architecture after the begin keyword

FDDPE_INSTANCE_NAME : FDDPE
  port map (Q => user_Q,
           C => user_C,
           CE => user_CE,
           D => user_D,
           PRE => user_PRE);
```

Verilog Instantiation Template

```
FDDPE FDDPE_instance_name (.Q (user_Q),
                          .C (user_C),
                          .CE (user_CE),
                          .D (user_D),
                          .PRE (user_PRE));

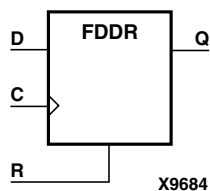
defparam FDDPE_instance_name.INIT = bit_value;
```


FDDR

Dual Edge Triggered D Flip-Flop with Synchronous Reset

Architectures Supported

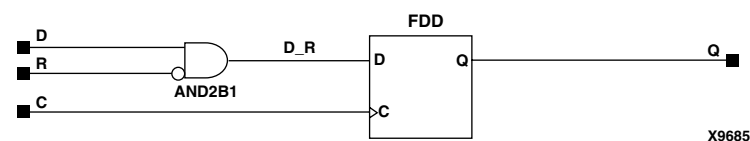
FDDR	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDR is a single dual edge triggered D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High and High-to-Low clock (C) transitions. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High or High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
1	X	↓	0
0	1	↑	1
0	0	↑	0
0	1	↓	1
0	0	↓	0



FDDR Implementation CoolRunner-II

Usage

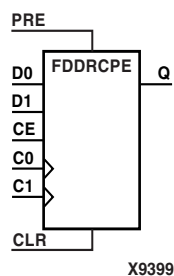
For HDL, this design element is inferred rather than instantiated.

FDDRCPE

Dual Data Rate D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Architectures Supported

FDDRCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDDRCPE is a Dual Data Rate (DDR) D flip-flop with two separate clocks (C0 and C1) phase shifted 180 degrees that allow selection of two separate data inputs (D0 and D1). It also has clock enable (CE), asynchronous preset (PRE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous PRE, when High, sets the Q output High; CLR, when High, resets the output Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Use the INIT attribute to initialize FDDRCPE during configuration.

Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

-- Component Declaration for FDDRCPE should be placed
-- after architecture statement but before begin keyword

```
component FDDRCPE
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C0 : in STD_ULOGIC;
        C1 : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D0 : in STD_ULOGIC;
        D1 : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for FDDRCPE
-- should be placed after architecture declaration but
-- before the begin keyword

```
attribute INIT : string;
attribute INIT of FDDRCPE_instance_name : label is "0";
-- values can be (0 or 1)
```

-- Component Instantiation for FDDRCPE should be placed
-- in architecture after the begin keyword

```
FDDRCPE_INSTANCE_NAME : FDDRCPE
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C0 => user_C0,
            C1 => user_C1,
            CE => user_CE,
            CLR => user_CLR,
            D0 => user_D0,
            D1 => user_D1,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
FDDRCPE FDDRCPE_instance_name (.Q (user_Q),
                                .C0 (user_C0),
                                .C1 (user_C1),
                                .CE (user_CE),
                                .CLR (user_CLR),
                                .D0 (user_D0),
                                .D1 (user_D1),
                                .PRE (user_PRE));

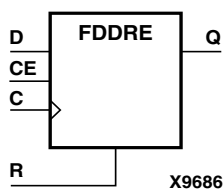
defparam FDDRCPE_instance_name.INIT = bit_value;
```

FDDRE

Dual Edge Triggered D Flip-Flop with Clock Enable and Synchronous Reset

Architectures Supported

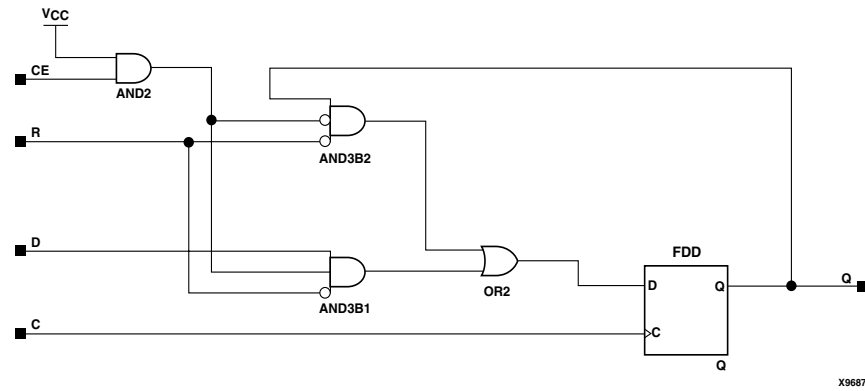
FDDRE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDRE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↑	0
1	X	X	↓	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0



FDDRE Implementation CoolRunner-II

Usage

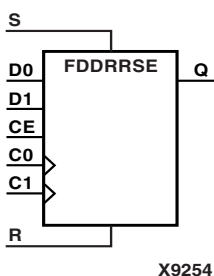
For HDL, this design element can be inferred but not instantiated.

FDDRRSE

Dual Data Rate D Flip-Flop with Clock Enable and Synchronous Reset and Set

Architectures Supported

FDDRRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDDRRSE is a Dual Data Rate (DDR) D flip-flop with two separate clocks (C0 and C1) phase shifted 180 degrees that allow selection of two separate data inputs (D0 and D1). It also has synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during any Low-to-High clock transition (C0 or C1). (Reset has precedence over Set.) When the S input is High and R is Low, the flip-flop is set, output High, during a Low-to-High clock transition (C0 or C1). Data on the D0 input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High C1 clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Use the INIT attribute to initialize FDDRRSE during configuration.

Inputs							Outputs
C0	C1	CE	D0	D1	R	S	Q
↑	X	X	X	X	1	0	0
↑	X	X	X	X	0	1	1
↑	X	X	X	X	1	1	0
X	↑	X	X	X	1	0	0
X	↑	X	X	X	0	1	1
X	↑	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for FDDRSE should be placed
-- after architecture statement but before begin keyword
```

```
component FDDRSE
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C0 : in STD_ULOGIC;
        C1 : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D0 : in STD_ULOGIC;
        D1 : in STD_ULOGIC;
        R : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDDRSE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDDRSE_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDDRSE should be placed
-- in architecture after the begin keyword
```

```
FDDRSE_INSTANCE_NAME : FDDRSE
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C0 => user_C0,
            C1 => user_C1,
            CE => user_CE,
            D0 => user_D0,
            D1 => user_D1,
            R => user_R,
            S => user_S);
```

Verilog Instantiation Template

```
FDDRSE FDDRSE_instance_name (.Q (user_Q),
                              .C0 (user_C0),
                              .C1 (user_C1),
                              .CE (user_CE),
                              .D0 (user_D0),
                              .D1 (user_D1),
                              .R (user_R),
                              .S (user_S));
```

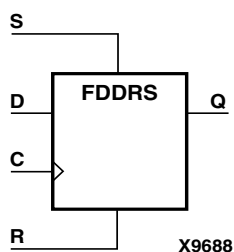
```
defparam FDDRSE_instance_name.INIT = bit_value;
```


FDDRS

Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set

Architectures Supported

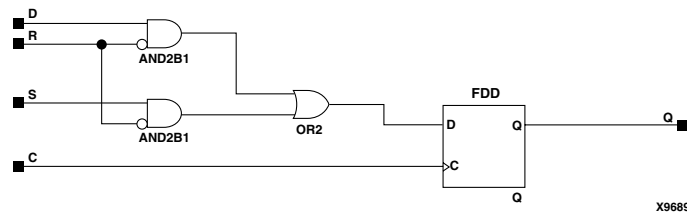
FDDRS	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDRS is a single dual edge triggered D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High or High-to-Low clock (C) transitions. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the Low-to-High or High-to-Low clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↑	0
1	X	X	↓	0
0	1	X	↑	1
0	1	X	↓	1
0	0	1	↑	1
0	0	1	↓	1
0	0	0	↑	0
0	0	0	↓	0



FDDRS Implementation CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDDRS should be placed
-- after architecture statement but before begin keyword

component FDDRS
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC;
        R : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDDRS
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDDRS_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDDRS should be placed
-- in architecture after the begin keyword

FDDRS_INSTANCE_NAME : FDDRS
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           D => user_D,
           R => user_R,
           S => user_S);
```

Verilog Instantiation Template

```
FDDRS FDDRS_instance_name (.Q (user_Q),
                          .C (user_C),
                          .D (user_D),
                          .R (user_R),
                          .S (user_S));

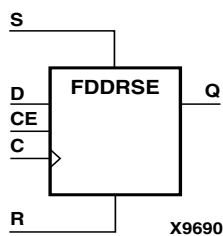
defparam FDDRS_instance_name.INIT = bit_value;
```

FDDRSE

Dual Edge Triggered D Flip-Flop with Synchronous Reset and Set and Clock Enable

Architectures Supported

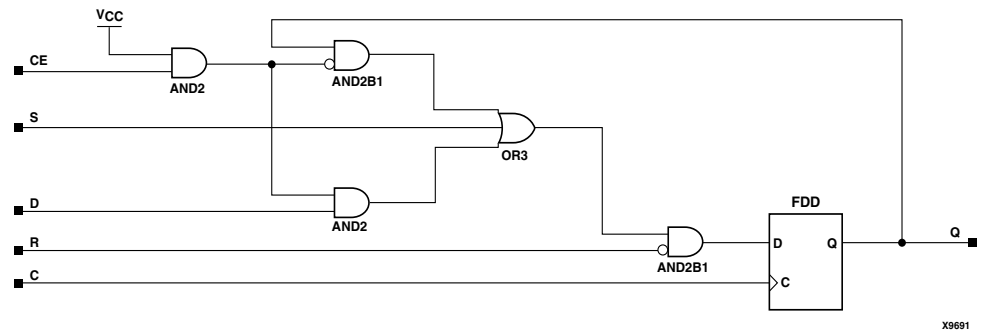
FDDRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDRSE is a single dual edge triggered D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High or High-to-Low clock transitions. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High or High-to-Low clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↑	0
1	X	X	X	↓	0
0	1	X	X	↑	1
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0
0	0	1	1	↓	1
0	0	1	0	↓	0



FDDRSE Implementation CoolRunner-II

Usage

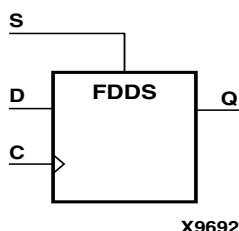
For HDL, this design element is inferred rather than instantiated.

FDDS

Dual Edge Triggered D Flip-Flop with Synchronous Set

Architectures Supported

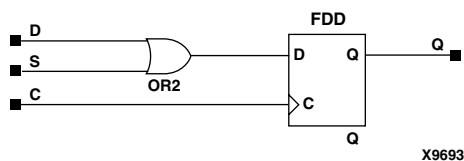
FDDS	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDS is a single dual edge triggered D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs			Outputs
S	D	C	Q
1	X	↑	1
1	X	↓	1
0	1	↑	1
0	0	↑	0
0	1	↓	1
0	0	↓	0



FDDS Implementation CoolRunner-II

Usage

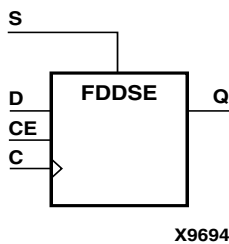
For HDL, this design element is inferred rather than instantiated.

FDDSE

D Flip-Flop with Clock Enable and Synchronous Set

Architectures Supported

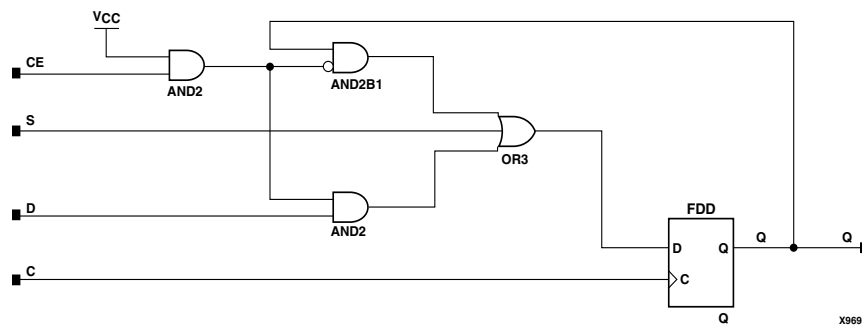
FDDSE	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDSE is a single dual edge triggered D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the Low-to-High or High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the Low-to-High and High-to-Low clock (C) transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
1	X	X	↓	1
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0
0	1	1	↓	1
0	1	0	↓	0



FDDSE Implementation CoolRunner-II

Usage

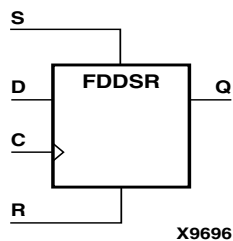
For HDL, this design element is inferred rather than instantiated.

FDDSR

Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset

Architectures Supported

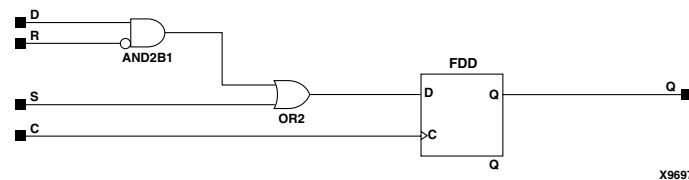
FDDSR	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDSR is a single dual edge triggered D-type flip-flop with data (D), synchronous reset (R) and synchronous set (S) inputs and data output (Q). When the set (S) input is High, it overrides all other inputs and sets the Q output High during the Low-to-High or High-to-Low clock transition. (Set has precedence over Reset.) When reset (R) is High and S is Low, the flip-flop is reset, output Low, on the Low-to-High or High-to-Low clock transition. Data on the D input is loaded into the flip-flop when S and R are Low on the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	R	D	C	Q
1	X	X	↑	1
1	X	X	↓	1
0	1	X	↑	0
0	1	X	↓	0
0	0	1	↑	1
0	0	0	↑	0
0	0	1	↓	1
0	0	0	↓	0



FDDSR Implementation CoolRunner-II

Usage

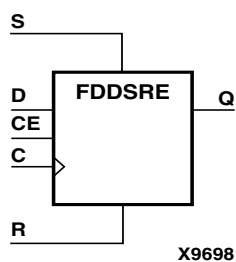
For HDL, this design element is inferred rather than instantiated.

FDDSRE

Dual Edge Triggered D Flip-Flop with Synchronous Set and Reset and Clock Enable

Architectures Supported

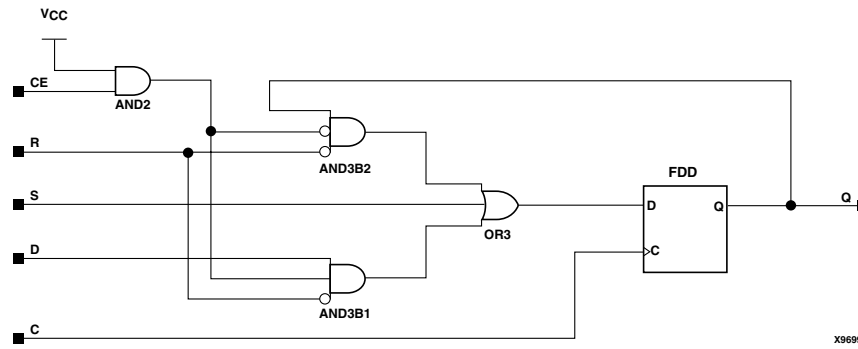
FDDSRE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FDDSRE is a single dual edge triggered D-type flip-flop with synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, it overrides all other inputs and sets the Q output High during the Low-to-High or High-to-Low clock transition. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low during the Low-to-High or High-to-Low clock transition. Data is loaded into the flip-flop when S and R are Low and CE is High during the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated in Verilog by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
S	R	CE	D	C	Q
1	X	X	X	↑	1
1	X	X	X	↓	1
0	1	X	X	↑	0
0	1	X	X	↓	0
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0
0	0	1	1	↓	1
0	0	1	0	↓	0



FDDSRE Implementation CoolRunner-II

Usage

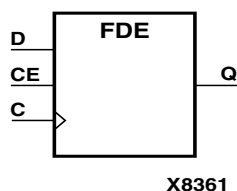
For HDL, this design element is inferred rather than instantiated.

FDE

D Flip-Flop with Clock Enable

Architectures Supported

FDE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDE is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↑	0
1	1	↑	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDE should be placed
-- after architecture statement but before begin keyword

component FDE
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
```

```
        CE : in STD_ULONGIC;
        D : in STD_ULONGIC);
end component;

-- Component Attribute specification for FDE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDE should be placed
-- in architecture after the begin keyword

FDE_INSTANCE_NAME : FDE
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           CE => user_CE,
           D => user_D);
```

Verilog Instantiation Template

```
FDE FDE_instance_name (.Q (user_Q),
                       .C (user_C),
                       .CE (user_CE),
                       .D (user_D));

defparam FDE_instance_name.INIT = bit_value;
```

Commonly Used Constraints

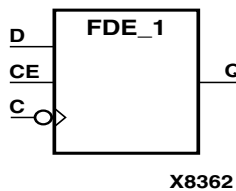
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDE_1

D Flip-Flop with Negative-Edge Clock and Clock Enable

Architectures Supported

FDE_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDE_1 is a single D-type flip-flop with data input (D), clock enable (CE), and data output (Q). When clock enable is High, the data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
0	X	X	No Chg
1	0	↓	0
1	1	↓	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDE_1 should be placed
-- after architecture statement but before begin keyword

component FDE_1
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
```

```
        CE : in STD_ULONGIC;
        D : in STD_ULONGIC);
end component;

-- Component Attribute specification for FDE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDE_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDE_1 should be placed
-- in architecture after the begin keyword

FDE_1_INSTANCE_NAME : FDE_1
-- synthesis translate_off
generic map (INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          CE => user_CE,
          D => user_D);
```

Verilog Instantiation Template

```
FDE_1 FDE_1_instance_name (.Q (user_Q),
                           .C (user_C),
                           .CE (user_CE),
                           .D (user_D));

defparam FDE_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

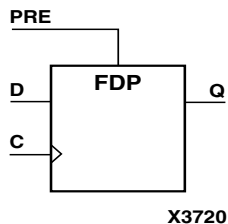
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDP

D Flip-Flop with Asynchronous Preset

Architectures Supported

FDP	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDP is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the Low-to-High clock (C) transition.

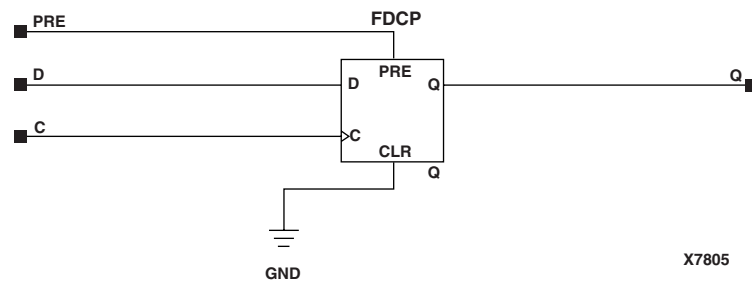
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↑	1	1
0	↑	0	0



FDP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDP should be placed
-- after architecture statement but before begin keyword
```

```
component FDP
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDP
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDP_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDP should be placed
-- in architecture after the begin keyword
```

```
FDP_INSTANCE_NAME : FDP
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
FDP FDP_instance_name (.Q (user_Q),  
                      .C (user_C),  
                      .D (user_D),  
                      .PRE (user_PRE));  
  
defparam FDP_instance_name.INIT = bit_value;
```

Commonly Used Constraints

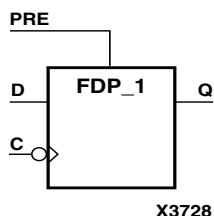
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
XBLKNM

FDP_1

D Flip-Flop with Negative-Edge Clock and Asynchronous Preset

Architectures Supported

FDP_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDP_1 is a single D-type flip-flop with data (D) and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and presets the Q output High. The data on the D input is loaded into the flip-flop when PRE is Low on the High-to-Low clock (C) transition.

The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	C	D	Q
1	X	X	1
0	↓	1	1
0	↓	0	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDP_1 should be placed
-- after architecture statement but before begin keyword

component FDP_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
```

```

    port (Q : out STD_ULONGIC;
          C : in  STD_ULONGIC;
          D : in  STD_ULONGIC;
          PRE : in STD_ULONGIC);
end component;

-- Component Attribute specification for FDP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDP_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDP_1 should be placed
-- in architecture after the begin keyword

FDP_1_INSTANCE_NAME : FDP_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D,
            PRE => user_PRE);

```

Verilog Instantiation Template

```

FDP_1 FDP_1_instance_name (.Q (user_Q),
                          .C (user_C),
                          .D (user_D),
                          .PRE (user_PRE));

defparam FDP_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

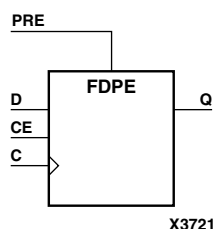
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDPE

D Flip-Flop with Clock Enable and Asynchronous Preset

Architectures Supported

FDPE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDPE is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the Low-to-High clock (C) transition. When CE is Low, the clock transitions are ignored.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously preset, output High, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For XC9500XL and XC9500XV devices, logic connected to the clock enable (CE) input may be implemented using the clock enable product term (p-term) in the macrocell, provided the logic can be completely implemented using the single p-term available for clock enable without requiring feedback from another macrocell. Only FDCE and FDPE flip-flop primitives may take advantage of the clock-enable p-term.

For Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied. These devices simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	↑	0
0	1	1	↑	1

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDPE should be placed
-- after architecture statement but before begin keyword

component FDPE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of

FDPE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDPE should be placed
-- in architecture after the begin keyword FDPE_INSTANCE_NAME : FDPE
-- synthesis translate_off
generic map (
  INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          CE => user_CE,
          D => user_D,
          PRE => user_PRE);
```

Verilog Instantiation Template

```
FDPE FDPE_instance_name (.Q (user_Q),
                          .C (user_C),
                          .CE (user_CE),
                          .D (user_D),
                          .PRE (user_PRE));

defparam

FDPE_instance_name.INIT = bit_value;
```


Commonly Used Constraints

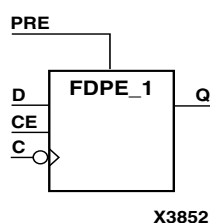
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
and XBLKNM

FDPE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Asynchronous Preset

Architectures Supported

FDPE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDPE_1 is a single D-type flip-flop with data (D), clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous PRE, when High, overrides all other inputs and sets the Q output High. Data on the D input is loaded into the flip-flop when PRE is Low and CE is High on the High-to-Low clock (C) transition. When CE is Low, the clock transitions are ignored.

The flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-III, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The active level of the GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	CE	D	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDPE_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FDPE_1
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      C : in STD_ULOGIC;
      CE : in STD_ULOGIC;
      D : in STD_ULOGIC;
      PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDPE_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDPE_1 should be placed
-- in architecture after the begin keyword --

FDPE_1_INSTANCE_NAME : FDPE_1
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          CE => user_CE,
          D => user_D,
          PRE => user_PRE);

```

Verilog Instantiation Template

```

FDPE_1 FDPE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .PRE (user_PRE));

defparam FDPE_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

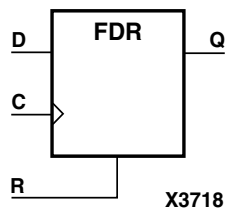
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDR

D Flip-Flop with Synchronous Reset

Architectures Supported

FDR	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDR is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
0	1	↑	1
0	0	↑	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDR should be placed
-- after architecture statement but before begin keyword
```

```
component FDR
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      C : in STD_ULOGIC;
      D : in STD_ULOGIC;
      R : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDR
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDR_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDR should be placed
-- in architecture after the begin keyword

FDR_INSTANCE_NAME : FDR
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          D => user_D,
          R => user_R);

```

Verilog Instantiation Template

```

FDR FDR_instance_name (.Q (user_Q),
                      .C (user_C),
                      .D (user_D),
                      .R (user_R));

defparam FDR_instance_name.INIT = bit_value;

```

Commonly Used Constraints

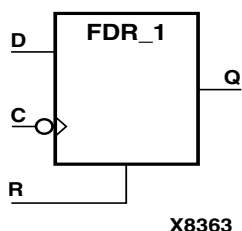
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDR_1

D Flip-Flop with Negative-Edge Clock and Synchronous Reset

Architectures Supported

FDR_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDR_1 is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
R	D	C	Q
1	X	↓	0
0	1	↓	1
0	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDR_1 should be placed
-- after architecture statement but before begin keyword

component FDR_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
end component;
```

```

-- synthesis translate_on
port (Q : out STD_ULOGIC;
      C : in  STD_ULOGIC;
      D : in  STD_ULOGIC;
      R : in  STD_ULOGIC);
end component;

-- Component Attribute specification for FDR_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDR_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDR_1 should be placed
-- in architecture after the begin keyword

FDR_1_INSTANCE_NAME : FDR_1
-- synthesis translate_off
generic map (
  INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          D => user_D,
          R => user_R);

```

Verilog Instantiation Template

```

FDR_1 FDR_1_instance_name (.Q (user_Q),
                          .C (user_C),
                          .D (user_D),
                          .R (user_R));

```

```

defparam FDR_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

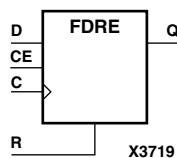
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRE

D Flip-Flop with Clock Enable and Synchronous Reset

Architectures Supported

FDRE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDRE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the Low-to-High clock transition.

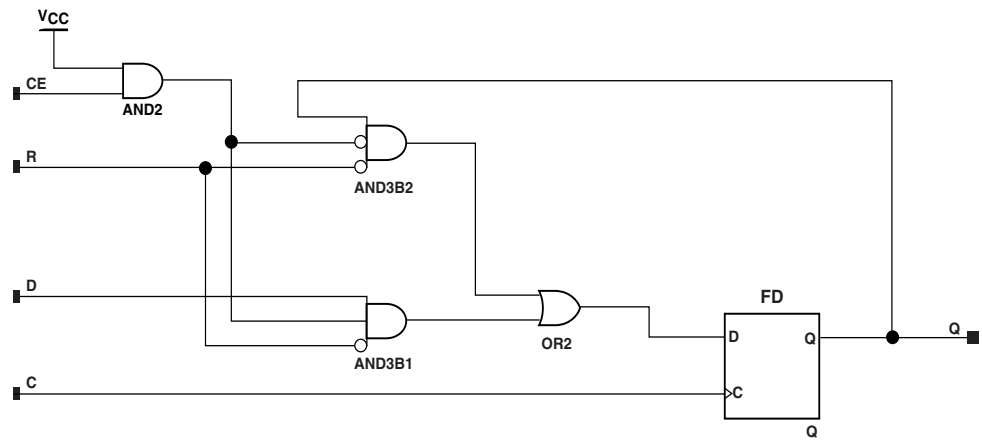
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↑	0
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0



X7808

FDRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDRE should be placed
-- after architecture statement but before begin keyword
```

```
component FDRE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC;
        R : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDRE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDRE_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDRE should be placed
-- in architecture after the begin keyword
```

```
FDRE_INSTANCE_NAME : FDRE
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
```

```
port map (Q => user_Q,  
         C => user_C,  
         CE => user_CE,  
         D => user_D,  
         R => user_R);
```

Verilog Instantiation Template

```
FDRE FDRE_instance_name (.Q (user_Q),  
                        .C (user_C),  
                        .CE (user_CE),  
                        .D (user_D),  
                        .R (user_R));
```

```
defparam FDRE_instance_name.INIT = bit_value;
```

Commonly Used Constraints

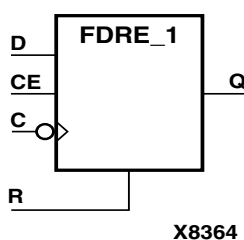
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRE_1

D Flip-Flop with Negative-Clock Edge, Clock Enable, and Synchronous Reset

Architectures Supported

FDRE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDRE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low and CE is High during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	CE	D	C	Q
1	X	X	↓	0
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDRE_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FDRE_1
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      C : in STD_ULOGIC;
      CE : in STD_ULOGIC;
      D : in STD_ULOGIC;
      R : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDRE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRE_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDRE_1 should be placed
-- in architecture after the begin keyword

FDRE_1_INSTANCE_NAME : FDRE_1
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          CE => user_CE,
          D => user_D,
          R => user_R);

```

Verilog Instantiation Template

```

FDRE_1 FDRE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .R (user_R));

defparam FDRE_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

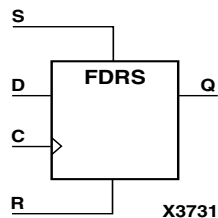
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRS

D Flip-Flop with Synchronous Reset and Set

Architectures Supported

FDRS	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDRS is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the Low-to-High clock transition.

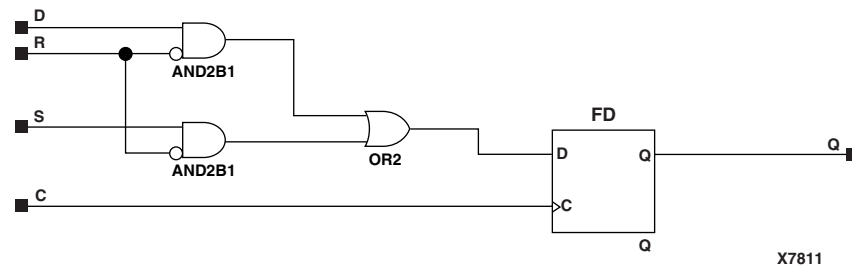
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↑	0
0	1	X	↑	1
0	0	1	↑	1
0	0	0	↑	0



FDRS Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDRS should be placed
-- after architecture statement but before begin keyword
```

```
component FDRS
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC;
        R : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDRS
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDRS_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDRS should be placed
-- in architecture after the begin keyword
```

```
FDRS_INSTANCE_NAME : FDRS
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D,
            R => user_R,
            S => user_S);
```


Verilog Instantiation Template

```
FDRS FDRS_instance_name (.Q (user_Q),  
                        .C (user_C),  
                        .D (user_D),  
                        .R (user_R),  
                        .S (user_S));  
  
defparam FDRS_instance_name.INIT = bit_value;
```

Commonly Used Constraints

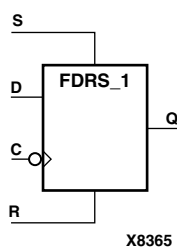
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
XBLKNM

FDRS_1

D Flip-Flop with Negative-Clock Edge and Synchronous Reset and Set

Architectures Supported

FDRS_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDRS_1 is a single D-type flip-flop with data (D), synchronous set (S), and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock (C) transition. (Reset has precedence over Set.) When S is High and R is Low, the flip-flop is set, output High, during the High-to-Low clock transition. When R and S are Low, data on the (D) input is loaded into the flip-flop during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
R	S	D	C	Q
1	X	X	↓	0
0	1	X	↓	1
0	0	1	↓	1
0	0	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDRS_1 should be placed
-- after architecture statement but before begin keyword
```

```

component FDRS_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULONGIC;
        C : in STD_ULONGIC;
        D : in STD_ULONGIC;
        R : in STD_ULONGIC;
        S : in STD_ULONGIC);
end component;

-- Component Attribute specification for FDRS_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDRS_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDRS_1 should be placed
-- in architecture after the begin keyword

FDRS_1_INSTANCE_NAME : FDRS_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D,
            R => user_R,
            S => user_S);

```

Verilog Instantiation Template

```

FDRS_1 FDRS_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .D (user_D),
                             .R (user_R),
                             .S (user_S));

defparam FDRS_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

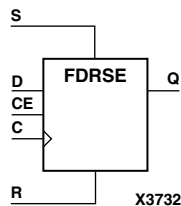
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDRSE

D Flip-Flop with Synchronous Reset and Set and Clock Enable

Architectures Supported

FDRSE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDRSE is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the Low-to-High clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the Low-to-High clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the Low-to-High clock transition.

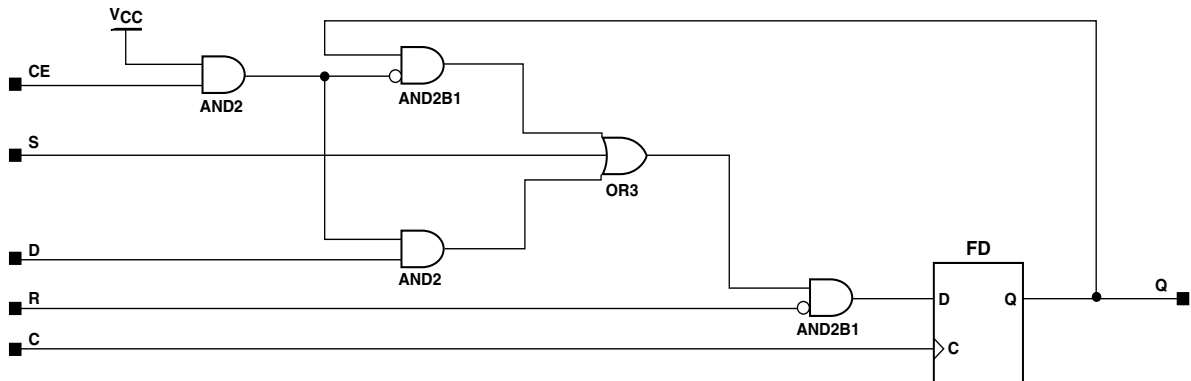
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0



X7813

FDRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDRSE: Single Data Rate D Flip-Flop with Synchronous Clear,
--       Set and Clock Enable (posedge clk). All families.
-- The current version of the Xilinx HDL Libraries Guide
```

```
FDRSE_inst : FDRSE
port map (
    Q => Q,      -- Data output
    C => C,      -- Clock input
    CE => CE,    -- Clock enable input
    D => D,      -- Data input
    R => R,      -- Synchronous reset input
    S => S       -- Synchronous set input
);
```

```
-- End of FDRSE_inst instantiation
```

Verilog Instantiation Template

```
// FDRSE: Single Data Rate D Flip-Flop with Synchronous Clear,
//       Set and Clock Enable (posedge clk). All families.
// The current version of the Xilinx HDL Libraries Guide
```

```
FDRSE FDRSE_inst (
    .Q(Q),      // Data output
    .C(C),      // Clock input
    .CE(CE),    // Clock enable input
    .D(D),      // Data input
    .R(R),      // Synchronous reset input
    .S(S)       // Synchronous set input
);
```

```
// The following defparam declaration is only necessary if you wish to
// change the initial value of the register to a one. If the instance
```

```
// name to the FDRSE is changed, that change needs to be reflected in  
// the defparam statements.
```

```
defparam FDRSE_inst.INIT = 1'b0;
```

```
// End of FDRSE_inst instantiation
```

Commonly Used Constraints

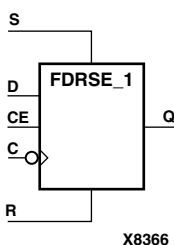
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET
and XBLKNM

FDRSE_1

D Flip-Flop with Negative-Clock Edge, Synchronous Reset and Set, and Clock Enable

Architectures Supported

FDRSE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDRSE_1 is a single D-type flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). The reset (R) input, when High, overrides all other inputs and resets the Q output Low during the High-to-Low clock transition. (Reset has precedence over Set.) When the set (S) input is High and R is Low, the flip-flop is set, output High, during the High-to-Low clock (C) transition. Data on the D input is loaded into the flip-flop when R and S are Low and CE is High during the High-to-Low clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
R	S	CE	D	C	Q
1	X	X	X	↓	0
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	1	↓	1
0	0	1	0	↓	0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- FDRSE_1: Single Data Rate D Flip-Flop with Synchronous Clear,
--           Set and Clock Enable (negedge clock). All families.
```

```
-- The current version of the Xilinx HDL Libraries Guide

FDRSE_1_inst : FDRSE_1
port map (
    Q => Q,      -- Data output
    C => C,      -- Clock input
    CE => CE,    -- Clock enable input
    CLR => CLR,  -- Asynchronous clear input
    D => D,      -- Data input
    R => R,      -- Synchronous reset input
    S => S      -- Synchronous set input
);

-- End of FDCPE_1_inst instantiation
```

Verilog Instantiation Template

```
// FDRSE_1: Single Data Rate D Flip-Flop with Synchronous Clear,
//          Set and Clock Enable (negedge clock). All families.
// The current version of the Xilinx HDL Libraries Guide

FDRSE_1 FDRSE_1_inst (
    .Q(Q),      // Data output
    .C(C),      // Clock input
    .CE(CE),    // Clock enable input
    .CLR(CLR),  // Asynchronous clear input
    .D(D),      // Data input
    .R(R),      // Synchronous reset input
    .S(S)       // Synchronous set input
);

// The following defparam declaration is only necessary if you wish to
// change the initial value of the register to a one. If the instance
// name to the FDRSE_1 is changed, that change needs to be reflected in
// the defparam statements.

defparam FDRSE_1_inst.INIT = 1'b0;

// End of FDRSE_1_inst instantiation
```

Commonly Used Constraints

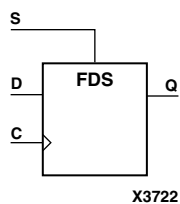
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDS

D Flip-Flop with Synchronous Set

Architectures Supported

FDS	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDS is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the Low-to-High clock (C) transition.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied. For all other devices (XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II), the flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDS will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

Inputs			Outputs
S	D	C	Q
1	X	↑	1
0	1	↑	1
0	0	↑	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDS should be placed
-- after architecture statement but before begin keyword

component FDS
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        D : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;

-- Component Attribute specification for FDS
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDS_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDS should be placed
-- in architecture after the begin keyword

FDS_INSTANCE_NAME : FDS
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            C => user_C,
            D => user_D,
            S => user_S);
```

Verilog Instantiation Template

```
FDS FDS_instance_name (.Q (user_Q),
                      .C (user_C),
                      .D (user_D),
                      .S (user_S));

defparam FDS_instance_name.INIT = bit_value;
```

Commonly Used Constraints

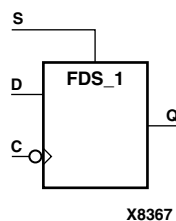
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDS_1

D Flip-Flop with Negative-Edge Clock and Synchronous Set

Architectures Supported

FDS_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDS_1 is a single D-type flip-flop with data (D) and synchronous set (S) inputs and data output (Q). The synchronous set input, when High, sets the Q output High on the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low during the High-to-Low clock (C) transition.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDS_1 will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

Inputs			Outputs
S	D	C	Q
1	X	↓	1
0	1	↓	1
0	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDS_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FDS_1
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULONGIC;
      C : in STD_ULONGIC;
      D : in STD_ULONGIC;
      S : in STD_ULONGIC);
end component;

-- Component Attribute specification for FDS_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of FDS_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for FDS_1 should be placed
-- in architecture after the begin keyword

FDS_1_INSTANCE_NAME : FDS_1
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          C => user_C,
          D => user_D,
          S => user_S);

```

Verilog Instantiation Template

```

FDS_1 FDS_1_instance_name (.Q (user_Q),
                          .C (user_C),
                          .D (user_D),
                          .S (user_S));

defparam FDS_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

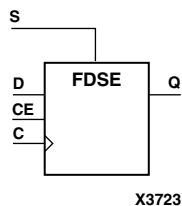
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDSE

D Flip-Flop with Clock Enable and Synchronous Set

Architectures Supported

FDSE	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDSE is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the Low-to-High clock (C) transition.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

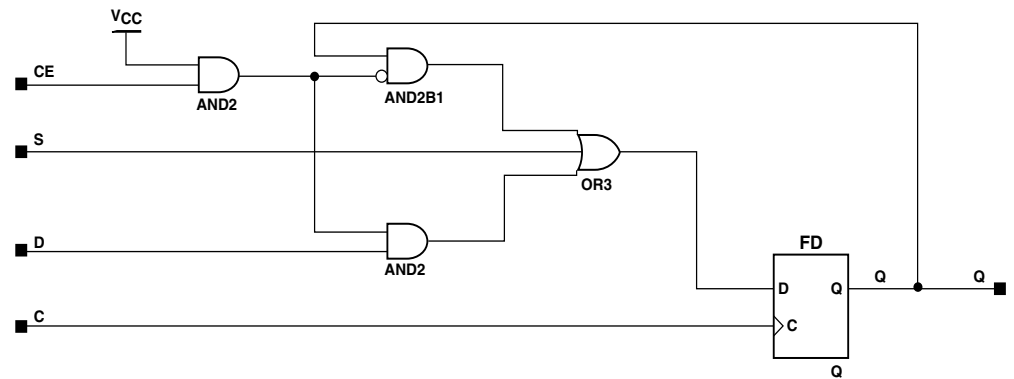
For all other devices (XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II), the flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDSE will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
0	0	X	X	No Chg
0	1	1	↑	1
0	1	0	↑	0



X7815

FDSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDSE should be placed
-- after architecture statement but before begin keyword
```

```
component FDSE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDSE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDSE_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDSE should be placed
-- in architecture after the begin keyword
```

```
FDSE_INSTANCE_NAME : FDSE
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
```



```
port map (Q => user_Q,  
         C => user_C,  
         CE => user_CE,  
         D => user_D,  
         S => user_S);
```

Verilog Instantiation Template

```
FDSE FDSE_instance_name (.Q (user_Q),  
                        .C (user_C),  
                        .CE (user_CE),  
                        .D (user_D),  
                        .S (user_S));
```

```
defparam FDSE_instance_name.INIT = bit_value;
```

Commonly Used Constraints

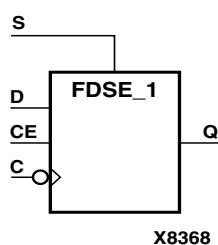
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
XBLKNM

FDSE_1

D Flip-Flop with Negative-Edge Clock, Clock Enable, and Synchronous Set

Architectures Supported

FDSE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



FDSE_1 is a single D-type flip-flop with data (D), clock enable (CE), and synchronous set (S) inputs and data output (Q). The synchronous set (S) input, when High, overrides the clock enable (CE) input and sets the Q output High during the High-to-Low clock (C) transition. The data on the D input is loaded into the flip-flop when S is Low and CE is High during the High-to-Low clock (C) transition.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FDSE_1 will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↓	1
0	0	X	X	No Chg
0	1	1	↓	1
0	1	0	↓	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDSE_1 should be placed
-- after architecture statement but before begin keyword
```

```
component FDSE_1
  -- synthesis translate_off
  generic (INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FDSE_1
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of FDSE_1_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for FDSE_1 should be placed
-- in architecture after the begin keyword
```

```
FDSE_1_INSTANCE_NAME : FDSE_1
  -- synthesis translate_off
  generic map (INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
           C => user_C,
           CE => user_CE,
           D => user_D,
           S => user_S);
```

Verilog Instantiation Template

```
FDSE_1 FDSE_1_instance_name (.Q (user_Q),
                             .C (user_C),
                             .CE (user_CE),
                             .D (user_D),
                             .S (user_S));
```

```
defparam FDSE_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

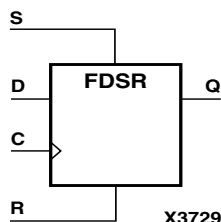
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FDSR

D Flip-Flop with Synchronous Set and Reset

Architectures Supported

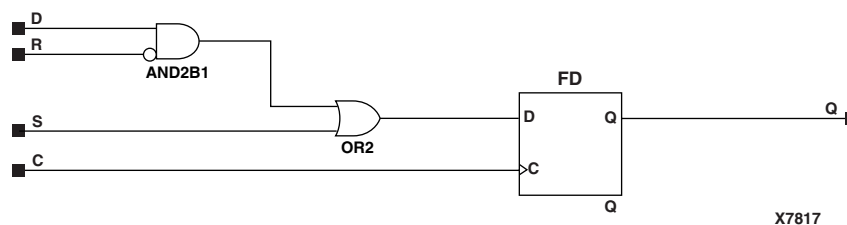
FDSR	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDSR is a single D-type flip-flop with data (D), synchronous reset (R) and synchronous set (S) inputs and data output (Q). When the set (S) input is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When reset (R) is High and S is Low, the flip-flop is reset, output Low, on the Low-to-High clock transition. Data on the D input is loaded into the flip-flop when S and R are Low on the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
S	R	D	C	Q
1	X	X	↑	1
0	1	X	↑	0
0	0	1	↑	1
0	0	0	↑	0



FDSR Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

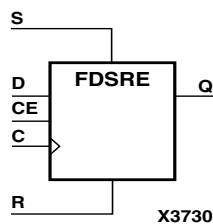
For HDL, this design element is inferred rather than instantiated.

FDSRE

D Flip-Flop with Synchronous Set and Reset and Clock Enable

Architectures Supported

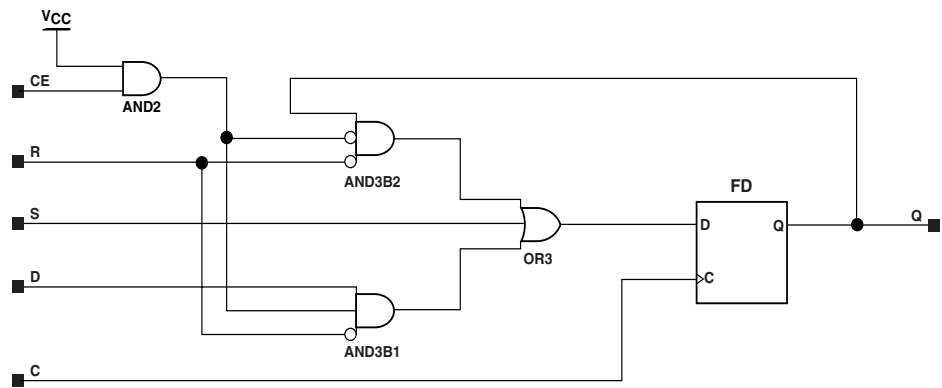
FDSRE	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDSRE is a single D-type flip-flop with synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, it overrides all other inputs and sets the Q output High during the Low-to-High clock transition. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low during the Low-to-High clock transition. Data is loaded into the flip-flop when S and R are Low and CE is High during the Low-to-high clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
S	R	CE	D	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	1	↑	1
0	0	1	0	↑	0



X7819

FDSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

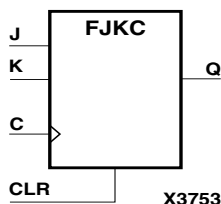
For HDL, this design element is inferred rather than instantiated.

FJKC

J-K Flip-Flop with Asynchronous Clear

Architectures Supported

FJKC	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKC is a single J-K-type flip-flop with J, K, and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the Q output Low. When CLR is Low, the output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition.

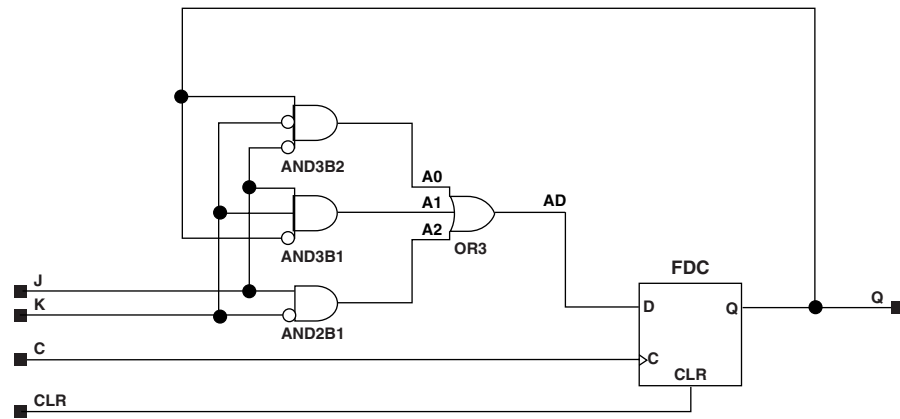
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

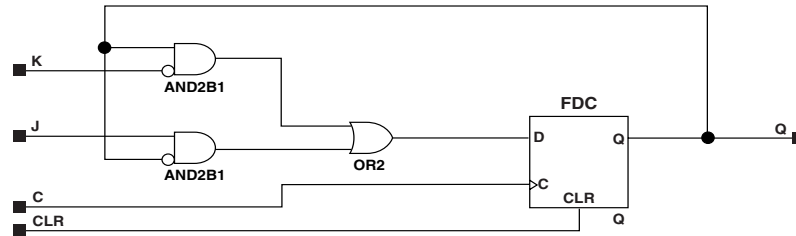
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	J	K	C	Q
1	X	X	X	0
0	0	0	↑	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle



X7820

FJKC Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7821

FJKC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

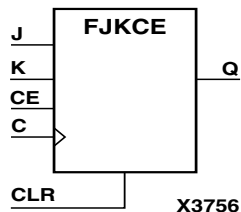
For HDL, this design element is inferred rather than instantiated.

FJKCE

J-K Flip-Flop with Clock Enable and Asynchronous Clear

Architectures Supported

FJKCE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKCE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous clear (CLR) inputs and data output (Q). The asynchronous clear (CLR), when High, overrides all other inputs and resets the Q output Low. When CLR is Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. When CE is Low, the clock transitions are ignored.

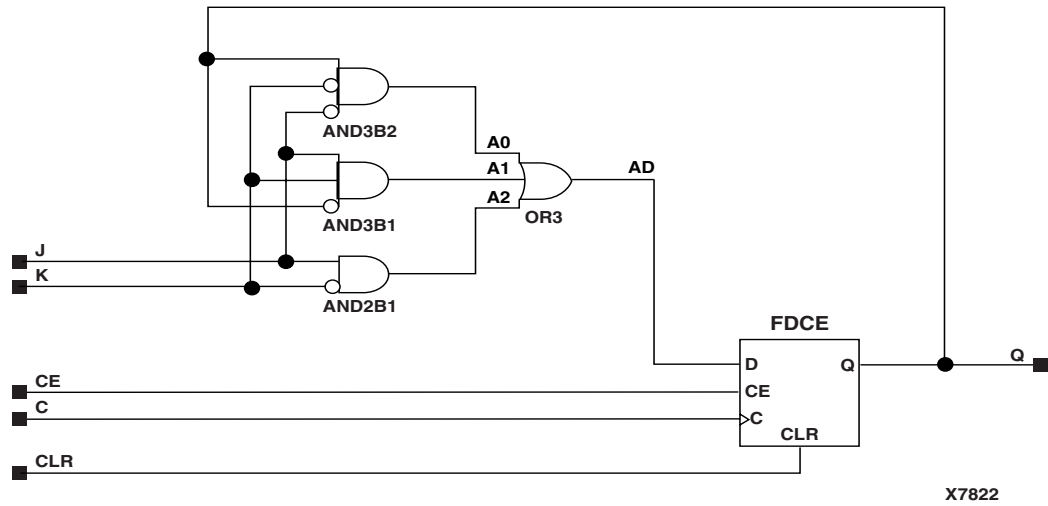
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

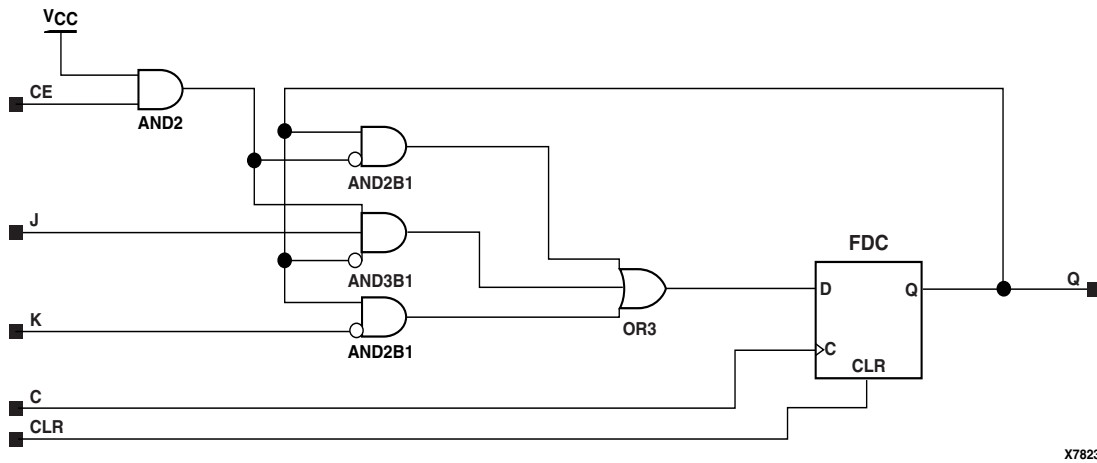
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	CE	J	K	C	Q
1	X	X	X	X	0
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle



FJKCE Implementation Spartan-II, Spartan-II E, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FJKCE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

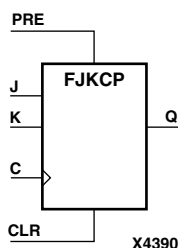
For HDL, this design element is inferred rather than instantiated.

FJKCP

J-K Flip-Flop with Asynchronous Clear and Preset

Architectures Supported

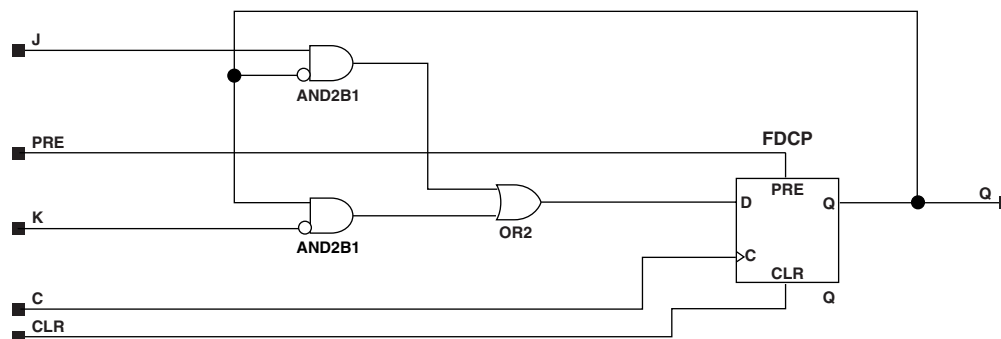
FJKCP	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKCP is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), and asynchronous preset (PRE) inputs and data output (Q). When the asynchronous clear (CLR) is High, all other inputs are ignored and Q is reset 0. The asynchronous preset (PRE), when High, and CLR set to Low overrides all other inputs and sets the Q output High. When CLR and PRE are Low, Q responds to the state of the J and K inputs during the Low-to-High clock transition, as shown in the following truth table.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	J	K	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	0	X	No Chg
0	0	0	1	↑	0
0	0	1	0	↑	1
0	0	1	1	↑	Toggle



X8124

FJKCP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

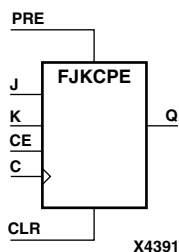
For HDL, this design element is inferred rather than instantiated.

FJKCPE

J-K Flip-Flop with Asynchronous Clear and Preset and Clock Enable

Architectures Supported

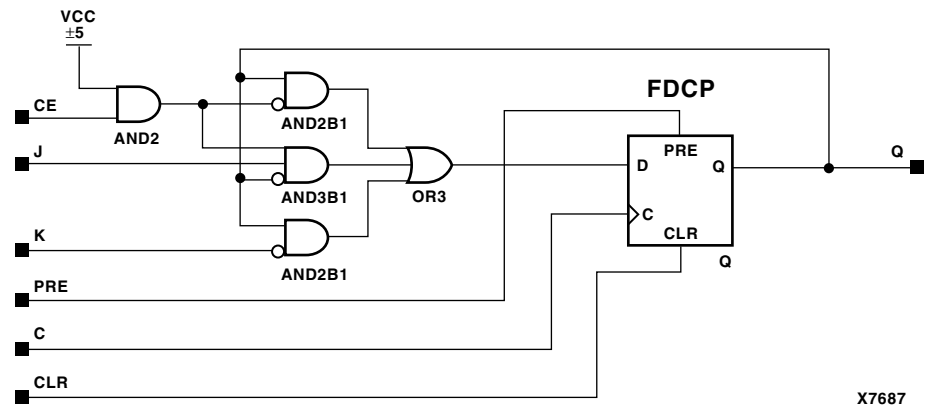
FJKCPE	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKCPE is a single J-K-type flip-flop with J, K, asynchronous clear (CLR), asynchronous preset (PRE), and clock enable (CE) inputs and data output (Q). When the asynchronous clear (CLR) is High, all other inputs are ignored and Q is reset 0. The asynchronous preset (PRE), when High, and CLR set to Low overrides all other inputs and sets the Q output High. When CLR and PRE are Low and CE is High, Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
CLR	PRE	CE	J	K	C	Q
1	X	X	X	X	X	0
0	1	X	X	X	X	1
0	0	0	0	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle



FJKCPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

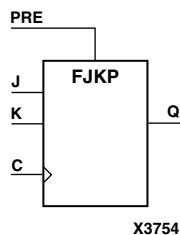
For HDL, this design element is inferred rather than instantiated.

FJKP

J-K Flip-Flop with Asynchronous Preset

Architectures Supported

FJKP	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKP is a single J-K-type flip-flop with J, K, and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE) input, when High, overrides all other inputs and sets the Q output High. When PRE is Low, the Q output responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock transition.

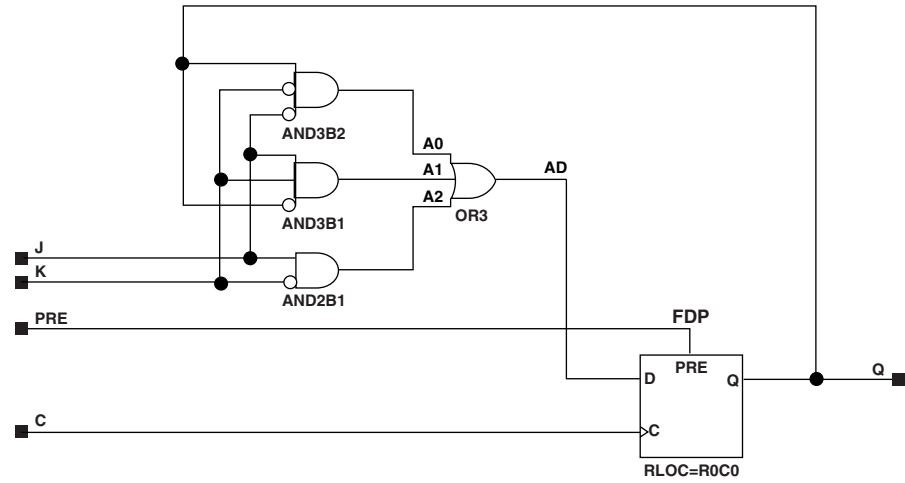
For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

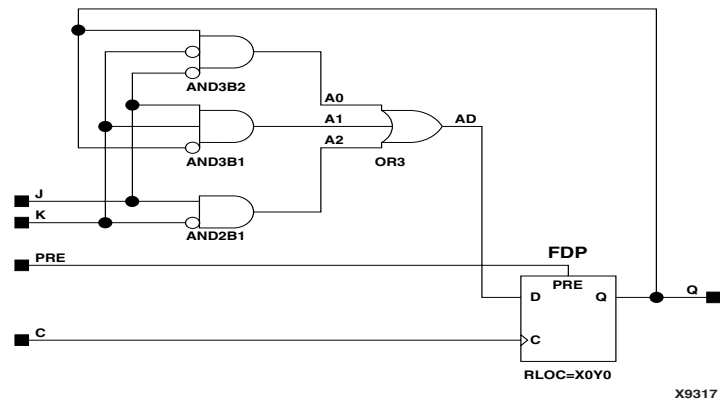
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
PRE	J	K	C	Q
1	X	X	X	1
0	0	0	X	No Chg
0	0	1	↑	0
0	1	0	↑	1
0	1	1	↑	Toggle



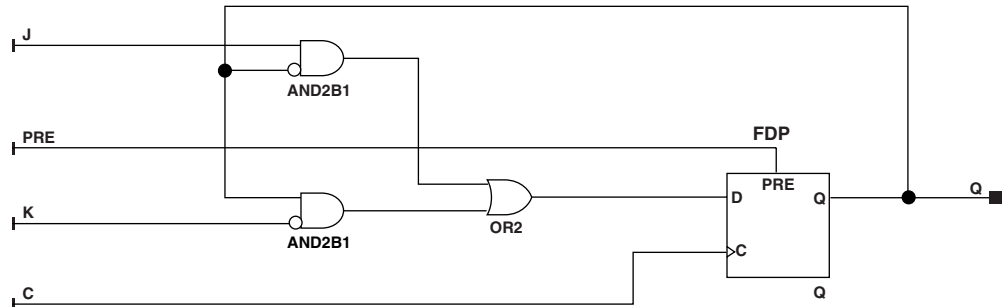
X7824

FJKP Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



X9317

FJKP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



X8125

FJKP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

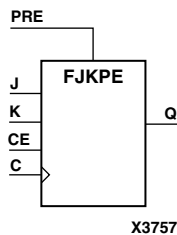
For HDL, this design element is inferred rather than instantiated.

FJKPE

J-K Flip-Flop with Clock Enable and Asynchronous Preset

Architectures Supported

FJKPE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKPE is a single J-K-type flip-flop with J, K, clock enable (CE), and asynchronous preset (PRE) inputs and data output (Q). The asynchronous preset (PRE), when High, overrides all other inputs and sets the Q output High. When PRE is Low and CE is High, the Q output responds to the state of the J and K inputs, as shown in the truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

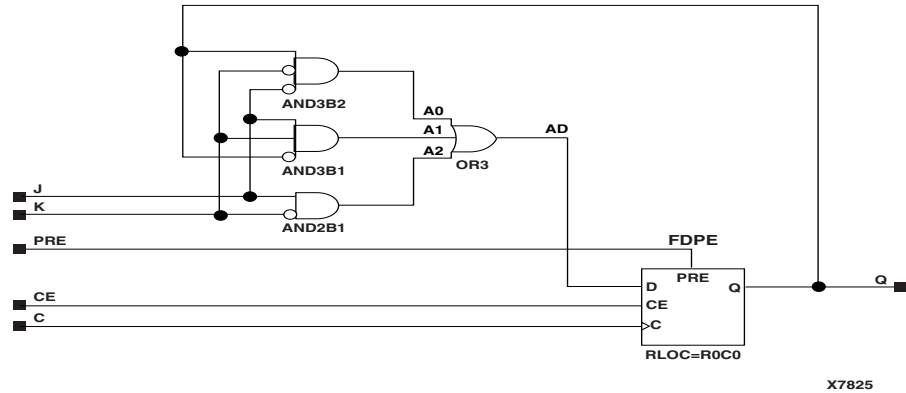
For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

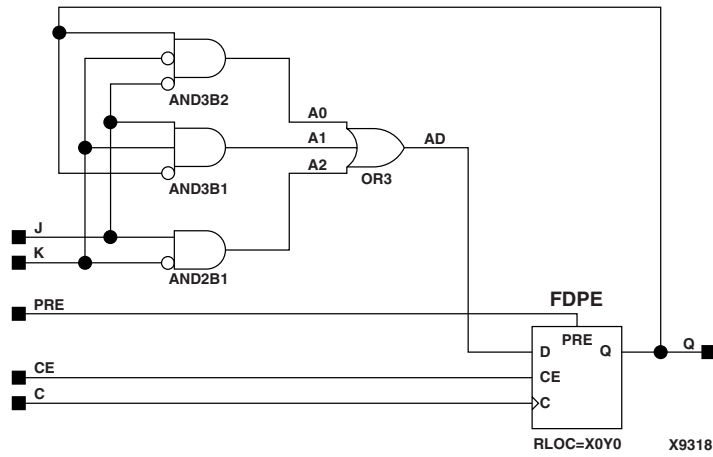
The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

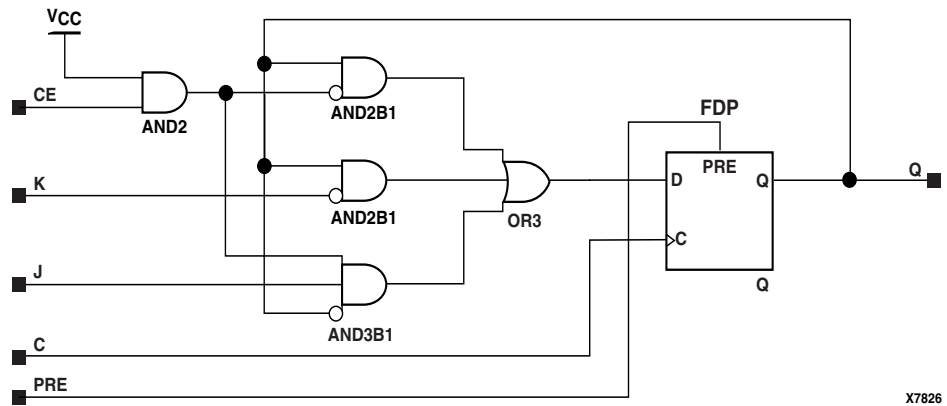
Inputs					Outputs
PRE	CE	J	K	C	Q
1	X	X	X	X	1
0	0	X	X	X	No Chg
0	1	0	0	X	No Chg
0	1	0	1	↑	0
0	1	1	0	↑	1
0	1	1	1	↑	Toggle



FJKPE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FJKPE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FJKPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

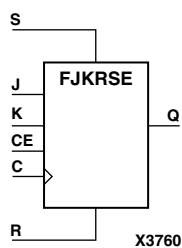
For HDL, this design element is inferred rather than instantiated.

FJKRSE

J-K Flip-Flop with Clock Enable and Synchronous Reset and Set

Architectures Supported

FJKRSE	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKRSE is a single J-K-type flip-flop with J, K, synchronous reset (R), synchronous set (S), and clock enable (CE) inputs and data output (Q). When synchronous reset (R) is High, all other inputs are ignored and output Q is reset Low. (Reset has precedence over Set.) When synchronous set (S) is High and R is Low, output Q is set High. When R and S are Low and CE is High, output Q responds to the state of the J and K inputs, according to the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

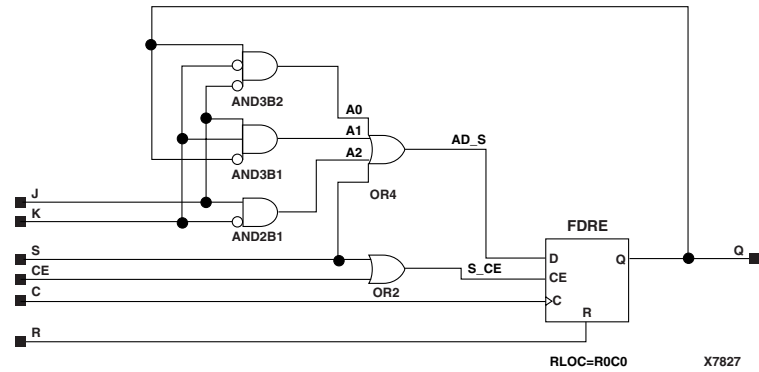
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

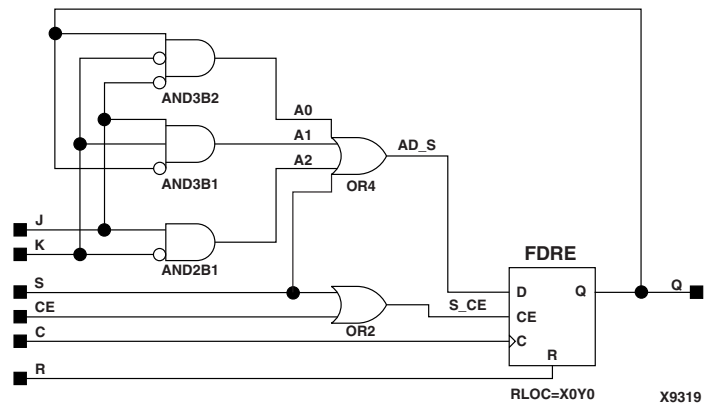
Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

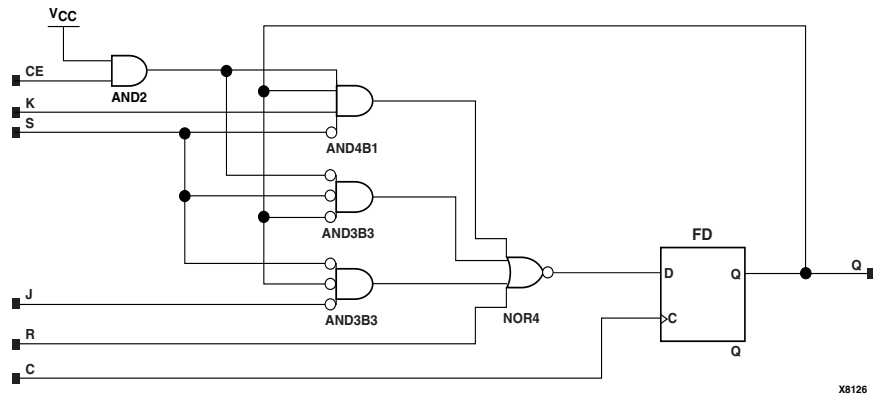
Inputs						Outputs
R	S	CE	J	K	C	Q
1	X	X	X	X	↑	0
0	1	X	X	X	↑	1
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	1	↑	Toggle
0	0	1	1	0	↑	1



FJKRSE Implementation Spartan-II, Spartan-II-E, Virtex, Virtex-E



FJKRSE Implementation Spartan-3, Virtex-II, Virtex-II Pro



FJKRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

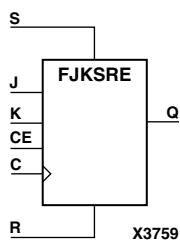
For HDL, this design element is inferred rather than instantiated.

FJKSRE

J-K Flip-Flop with Clock Enable and Synchronous Set and Reset

Architectures Supported

FJKSRE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FJKSRE is a single J-K-type flip-flop with J, K, synchronous set (S), synchronous reset (R), and clock enable (CE) inputs and data output (Q). When synchronous set (S) is High, all other inputs are ignored and output Q is set High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, output Q is reset Low. When S and R are Low and CE is High, output Q responds to the state of the J and K inputs, as shown in the following truth table, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

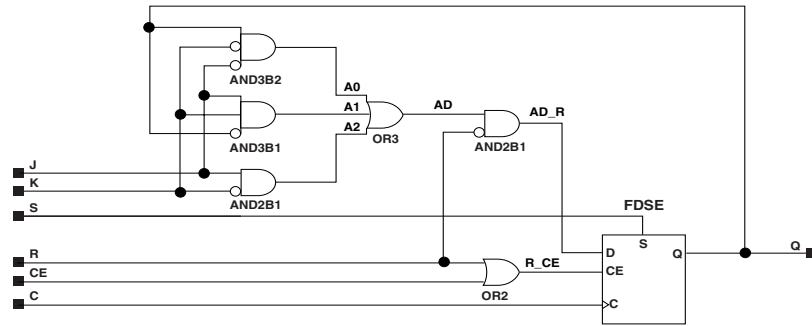
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

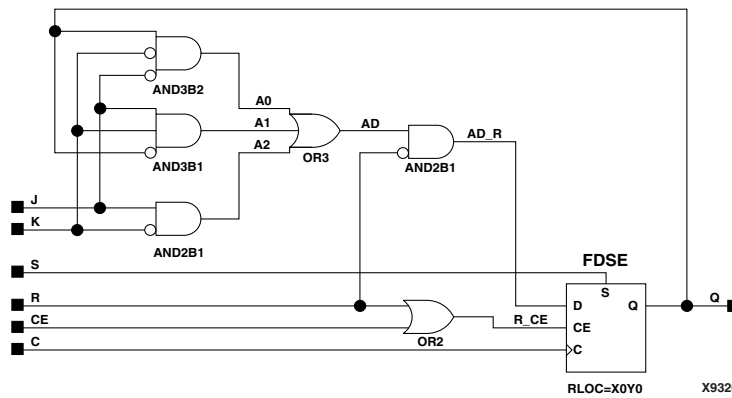
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FJKSRE will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

Inputs						Outputs
S	R	CE	J	K	C	Q
1	X	X	X	X	↑	1
0	1	X	X	X	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	0	↑	1
0	0	1	1	1	↑	Toggle

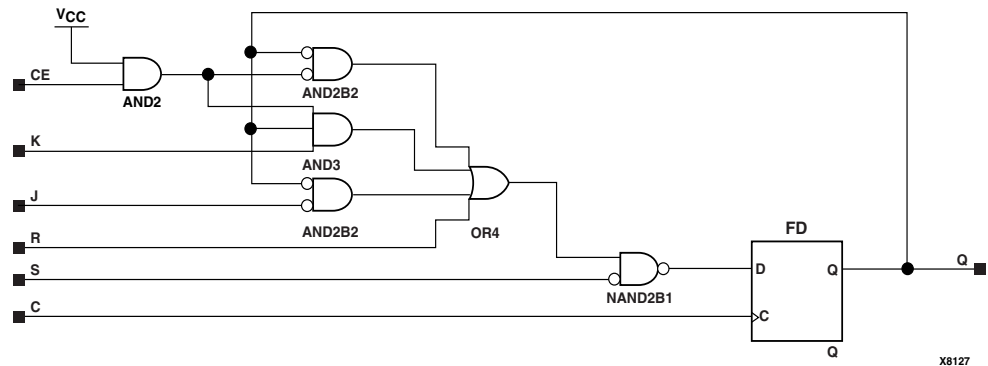


X7828

FJKSRE Implementation Spartan-II, Spartan-II-E, Virtex, Virtex-E



FJKSRE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X8127

FJKSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

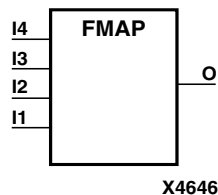
For HDL, this design element is inferred rather than instantiated.

FMAP

F Function Generator Partitioning Control Symbol

Architectures Supported

FMAP	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The FMAP symbol is used to map logic to the function generator of a slice. See the appropriate CAE tool interface user guide for information about specifying this attribute in your schematic design editor.

The `MAP=type` parameter can be used with the FMAP symbol to further define how much latitude you want to give the mapping program. The following table shows MAP option characters and their meanings.

MAP Option Character	Function
P	Pins.
C	Closed — Adding logic to or removing logic from the CLB is not allowed.
L	Locked — Locking CLB pins.
O	Open — Adding logic to or removing logic from the CLB is allowed.
U	Unlocked — No locking on CLB pins.

Possible types of MAP parameters for FMAP are `MAP=PUC`, `MAP=PLC`, `MAP=PLO`, and `MAP=PUO`. The default parameter is `PUO`. If one of the “open” parameters is used (`PLO` or `PUO`), only the output signals must be specified.

Note: Currently, only `PUC` and `PUO` are observed. `PLC` and `PLO` are translated into `PUC` and `PUO`, respectively.

The FMAP symbol can be assigned to specific CLB locations using `LOC` attributes.

Usage

FMAPs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate FMAPs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template

```
-- Component Declaration for FMAP should be placed
-- after architecture statement but before begin keyword

component FMAP
  port (I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC;
        I3 : in STD_ULOGIC;
        I4 : in STD_ULOGIC;
        O  : in STD_ULOGIC);
end component;

-- Component Attribute specification for FMAP
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here

-- Component Instantiation for FMAP should be placed
-- in architecture after the begin keyword

FMAP_INSTANCE_NAME : FMAP
  port map (I1 => user_I1,
            I2 => user_I2,
            I3 => user_I3,
            I4 => user_I4,
            O  => user_O);
```

Verilog Instantiation Template

```
FMAP FMAP_instanceb_name (.I1 (user_I1),
                          .I2 (user_I2),
                          .I3 (user_I3),
                          .I4 (user_I4),
                          .O  (user_O));
```

Commonly Used Constraints

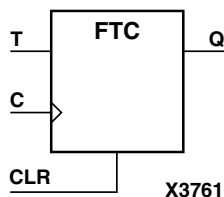
BEL, BLKNM, HBLKNM, HU_SET, LOC, MAP, U_SET

FTC

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear

Architectures Supported

FTC	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTC is a synchronous, resettable toggle flip-flop. The asynchronous clear (CLR) input, when High, overrides all other inputs and resets the data output (Q) Low. The Q output toggles, or changes state, when the toggle enable (T) input is High and CLR is Low during the Low-to-High clock transition.

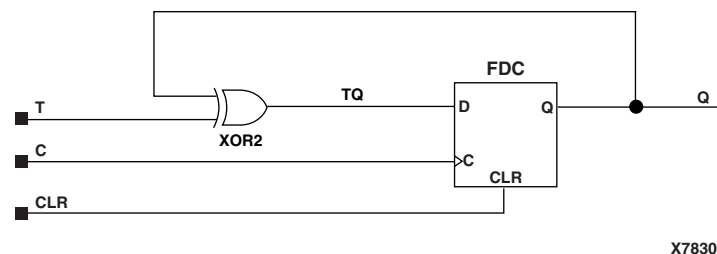
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

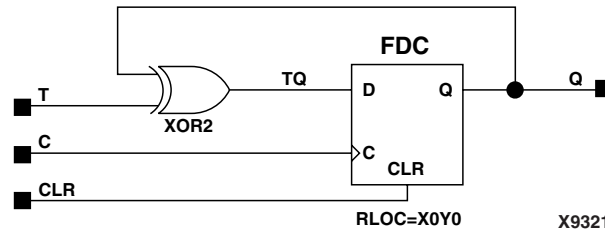
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

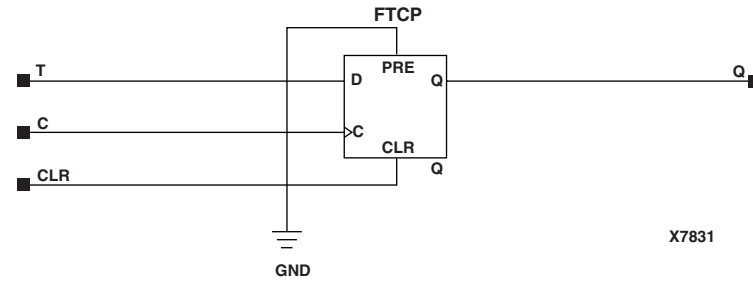
Inputs			Outputs
CLR	T	C	Q
1	X	X	0
0	0	X	No Chg
0	1	↑	Toggle



FTC Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTC Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTC Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Instantiation Template

-- Component Declaration for FTC should be placed
 -- after architecture statement but before begin keyword

```
component FTC
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        T : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for FTC
 -- should be placed after architecture declaration but
 -- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FTC should be placed
 -- in architecture after the begin keyword

```
FTC_INSTANCE_NAME : FTC
  port map (Q => user_Q,
            C => user_C,
            CLR => user_CLR,
            T => user_T);
```

Verilog Instantiation Template

```
FTC FTC_instance_name (.Q (user_Q),  
                        .C (user_C),  
                        .CLR (user_CLR),  
                        .T (user_T));
```

Commonly Used Constraints

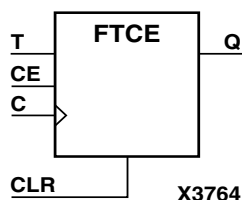
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
XBLKNM

FTCE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

FTCE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTCE is a toggle flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the data output (Q) is reset Low. When CLR is Low and toggle enable (T) and clock enable (CE) are High, Q output toggles, or changes state, during the Low-to-High clock (C) transition. When CE is Low, clock transitions are ignored.

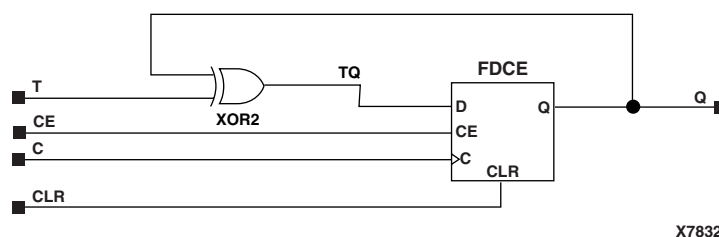
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

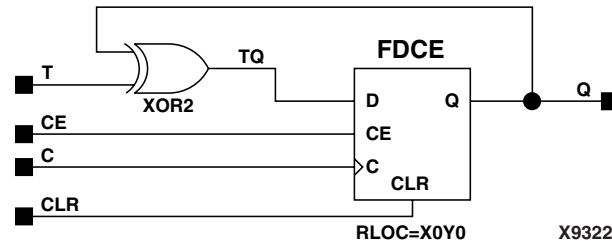
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

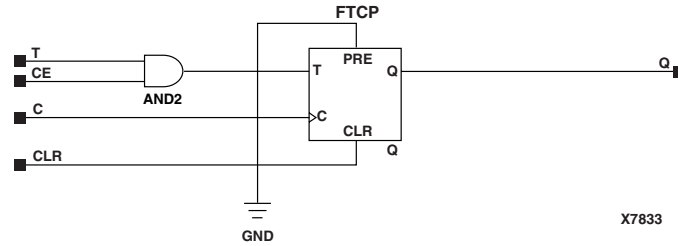
Inputs				Outputs
CLR	CE	T	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle



FTCE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTCE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

-- Component Declaration for FTCE should be placed
 -- after architecture statement but before begin keyword

```

component FTCE
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        CLR : in STD_ULOGIC;
        PRE : in STD_ULOGIC;
        T : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for FTCE
 -- should be placed after architecture declaration but
 -- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FTCE should be placed
 -- in architecture after the begin keyword

```

FTCE_INSTANCE_NAME : FTCE
  port map (Q => user_Q,
           C => user_C,
           CE => user_CE,
           CLR => user_CLR,
           PRE => user_PRE,
           T => user_T);
```


Verilog Instantiation Template

```
FTCE FTCE_instance_name (.Q (user_Q),  
                          .C (user_C),  
                          .CE (user_CE),  
                          .CLR (user_CLR),  
                          .PRE (user_PRE),  
                          .T (user_T));
```

Commonly Used Constraints

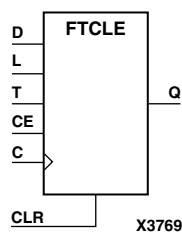
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET,
and XBLKNM.

FTCLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

FTCLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTCLE is a toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High and CLR is Low, clock enable (CE) is overridden and the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

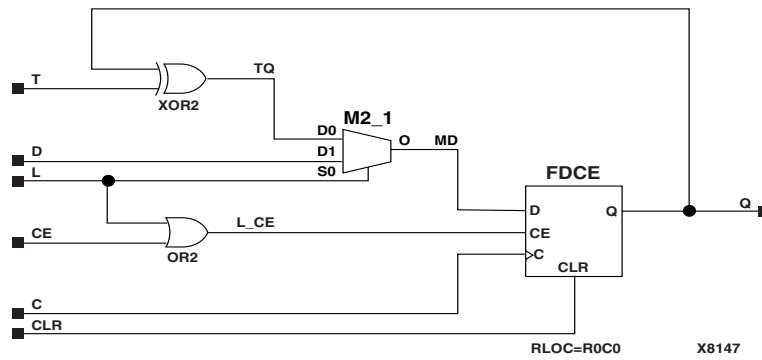
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

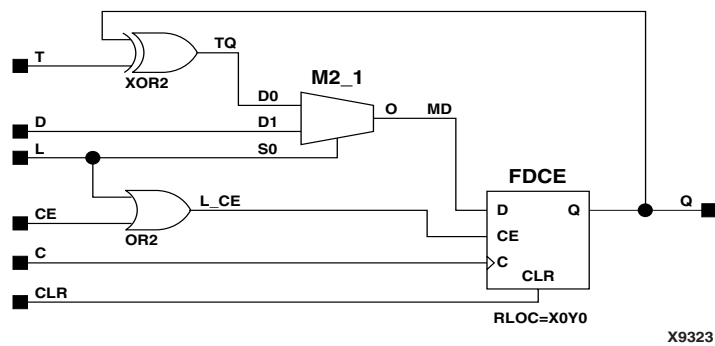
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

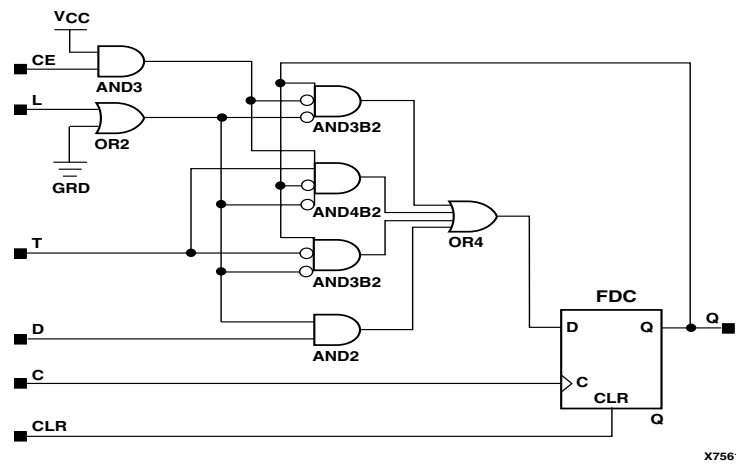
Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



FTCLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCLE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTCLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

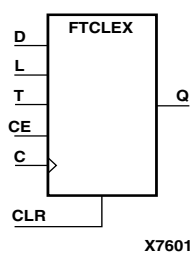
For HDL, this design element is inferred rather than instantiated.

FTCLEX

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

FTCLEX	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



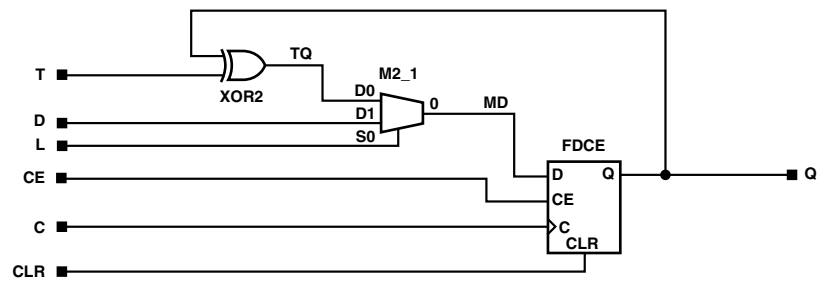
FTCLEX is a toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High, CLR is Low, and CE is High, the data on data input (D) is loaded into the flip-flop during the Low-to-High clock (C) transition. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High clock transition. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

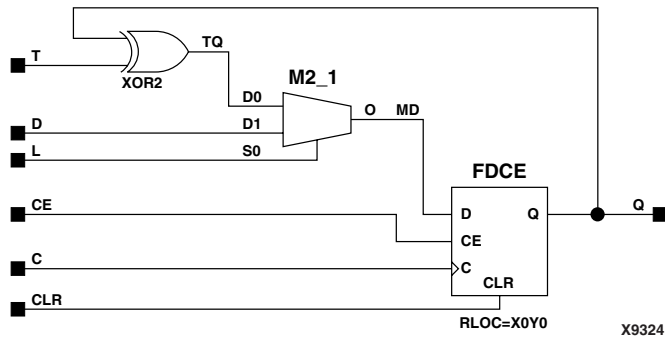
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	1	X	1	↑	1
0	1	1	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



X6995

FTCLEX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTCLEX Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

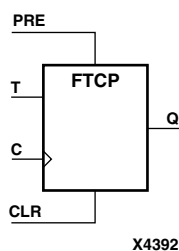
For HDL, this design element is inferred rather than instantiated.

FTCP

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset

Architectures Supported

FTCP	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTCP is a toggle flip-flop with toggle enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) is High and CLR is Low, all other inputs are ignored and Q is set High. When the toggle enable input (T) is High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	T	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	X	No Chg
0	0	1	↑	Toggle

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- Component Declaration for FTCP should be placed
-- after architecture statement but before begin keyword
```

```
component FTCP
port (Q : out STD_ULOGIC;
      C : in STD_ULOGIC;
      CLR : in STD_ULOGIC;
      PRE : in STD_ULOGIC;
      T : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for FTCP
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Enter attributes here
```

```
-- Component Instantiation for FTCP should be placed
-- in architecture after the begin keyword
```

```
FTCP_INSTANCE_NAME : FTCP
    port map (Q => user_Q,
              C => user_C,
              CLR => user_CLR,
              PRE => user_PRE,
              T => user_T);
```

Verilog Instantiation Template

```
FTCP FTCP_instance_name (.Q (user_Q),
                          .C (user_C),
                          .CLR (user_CLR),
                          .PRE (user_PRE),
                          .T (user_T));
```

Commonly Used Constraints

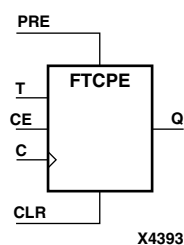
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTCPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

Architectures Supported

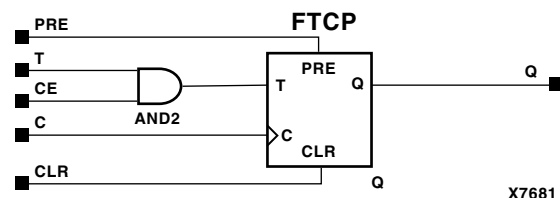
FTCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTCPE is a toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) is High and CLR is Low, all other inputs are ignored and Q is set High. When the toggle enable input (T) and the clock enable input (CE) are High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
CLR	PRE	CE	T	C	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTCPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

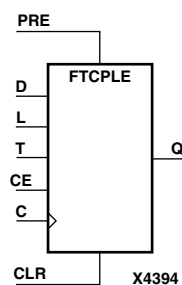
For HDL, this design element is inferred rather than instantiated.

FTCPL

Loadable Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear and Preset

Architectures Supported

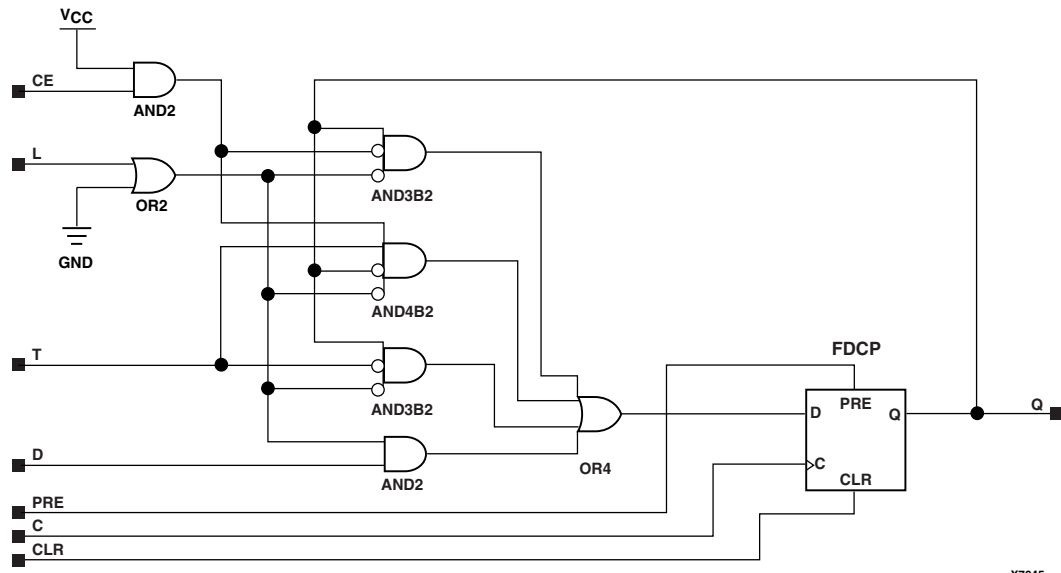
FTCPL	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTCPL is a loadable toggle flip-flop with toggle and clock enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) is High and CLR is Low, all other inputs are ignored and Q is set High. When the load input (L) is High, the clock enable input (CE) is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When the toggle enable input (T) and the clock enable input (CE) are High and CLR, PRE, and L are Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition. Clock transitions are ignored when CE is Low.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs							Outputs
CLR	PRE	L	CE	T	C	D	Q
1	X	X	X	X	X	X	0
0	1	X	X	X	X	X	1
0	0	1	X	X	↑	0	0
0	0	1	X	X	↑	1	1
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	↑	X	Toggle



X7845

FTCPLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

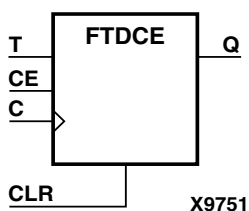
For HDL, this design element is inferred rather than instantiated.

FTDCE

Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

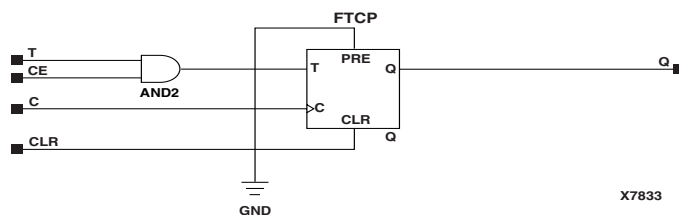
FTDCE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FTDCE is a dual edge triggered toggle flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the data output (Q) is reset Low. When CLR is Low and toggle enable (T) and clock enable (CE) are High, Q output toggles, or changes state, during the Low-to-High and High-to-Low clock (C) transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	CE	T	C	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle
0	1	1	↓	Toggle



FTDCE Implementation CoolRunner-II

Usage

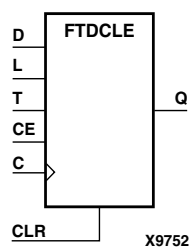
For HDL, this design element is inferred rather than instantiated.

FTDCLE

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

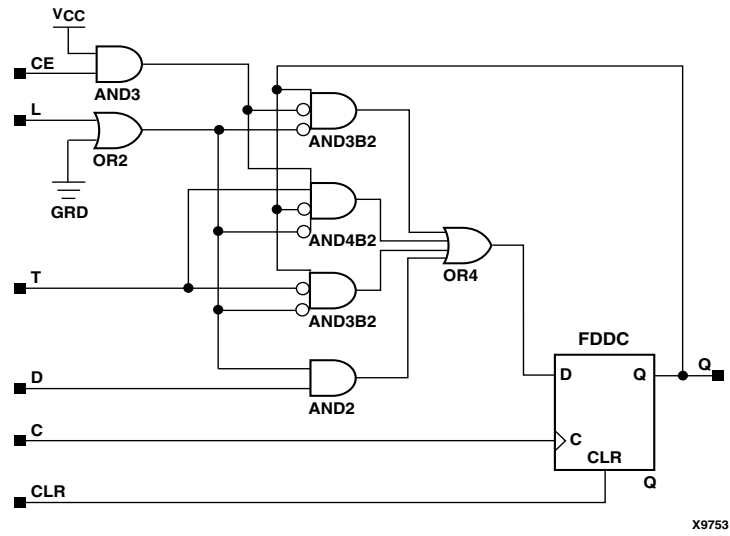
FTDCLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FTDCLE is a dual edge triggered toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High and CLR is Low, clock enable (CE) is overridden and the data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	X	X	1	↑	1
0	1	X	X	1	↓	1
0	1	X	X	0	↑	0
0	1	X	X	0	↓	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle
0	0	1	1	X	↓	Toggle



FTDCLE Implementation CoolRunner-II

Usage

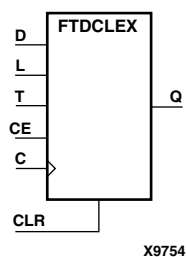
For HDL, this design element is inferred rather than instantiated.

FTDCLEX

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Clear

Architectures Supported

FTDCLEX	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FTDCLEX is a dual edge triggered toggle/loadable flip-flop with toggle and clock enable and asynchronous clear. When the asynchronous clear input (CLR) is High, all other inputs are ignored and output Q is reset Low. When load enable input (L) is High, CLR is Low, and CE is High, the data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock (C) transitions. When toggle enable (T) and CE are High and L and CLR are Low, output Q toggles, or changes state, during the Low- to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied.

Inputs						Outputs
CLR	L	CE	T	D	C	Q
1	X	X	X	X	X	0
0	1	1	X	1	↑	1
0	1	1	X	1	↓	1
0	1	1	X	0	↑	0
0	1	1	X	0	↓	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle
0	0	1	1	X	↓	Toggle

Usage

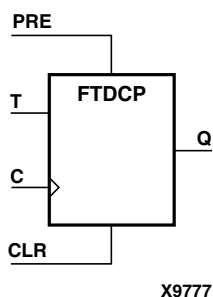
For HDL, this design element is inferred rather than instantiated.

FTDCP

Toggle Flip-Flop with Toggle Enable and Asynchronous Clear and Preset

Architectures Supported

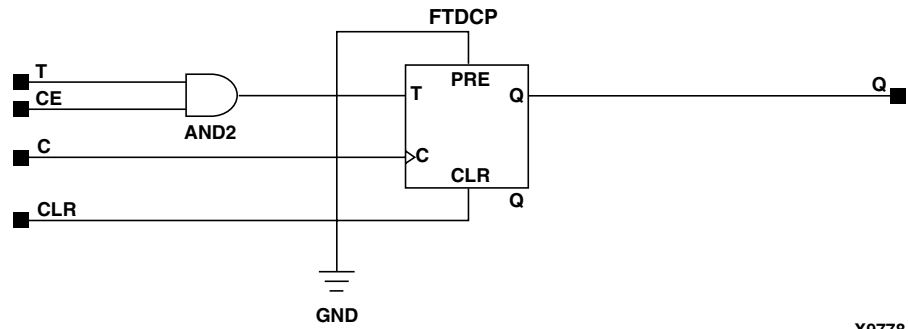
FTDCP	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTDCP is a toggle flip-flop with toggle enable and asynchronous clear and preset. When the asynchronous clear (CLR) input is High, all other inputs are ignored and the output (Q) is reset Low. When the asynchronous preset (PRE) is High and CLR is Low, all other inputs are ignored and Q is set High. When the toggle enable input (T) is High and CLR and PRE are Low, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock (C) transition.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs
CLR	PRE	T	C	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	X	No Chg
0	0	1	↑	Toggle
0	0	1	↓	Toggle



X9778

FTDCP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

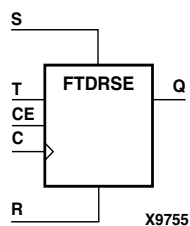
For HDL, this design element is inferred rather than instantiated.

FTDRSE

Dual Edge Triggered Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Architectures Supported

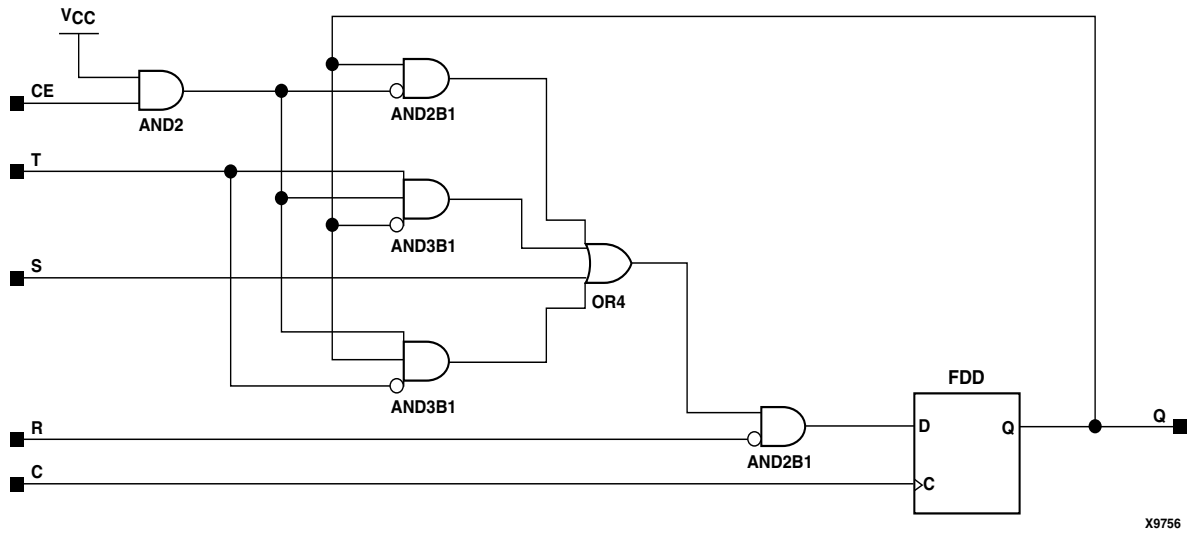
FTDRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



FTDRSE is a dual edge triggered toggle flip-flop with toggle and clock enable and synchronous reset and set. When the synchronous reset input (R) is High, it overrides all other inputs and the data output (Q) is reset Low. When the synchronous set input (S) is High and R is Low, clock enable input (CE) is overridden and output Q is set High. (Reset has precedence over Set.) When toggle enable input (T) and CE are High and R and S are Low, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock transitions.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs					Outputs
R	S	CE	T	C	Q
1	X	X	X	↑	0
1	X	X	X	↓	0
0	1	X	X	↑	1
0	1	X	X	↓	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle
0	0	1	1	↓	Toggle



FTDRSE Implementation CoolRunner-II

Usage

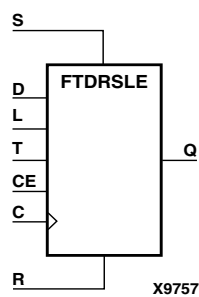
For HDL, this design element is inferred rather than instantiated.

FTDRSLE

Dual Edge Triggered Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Architectures Supported

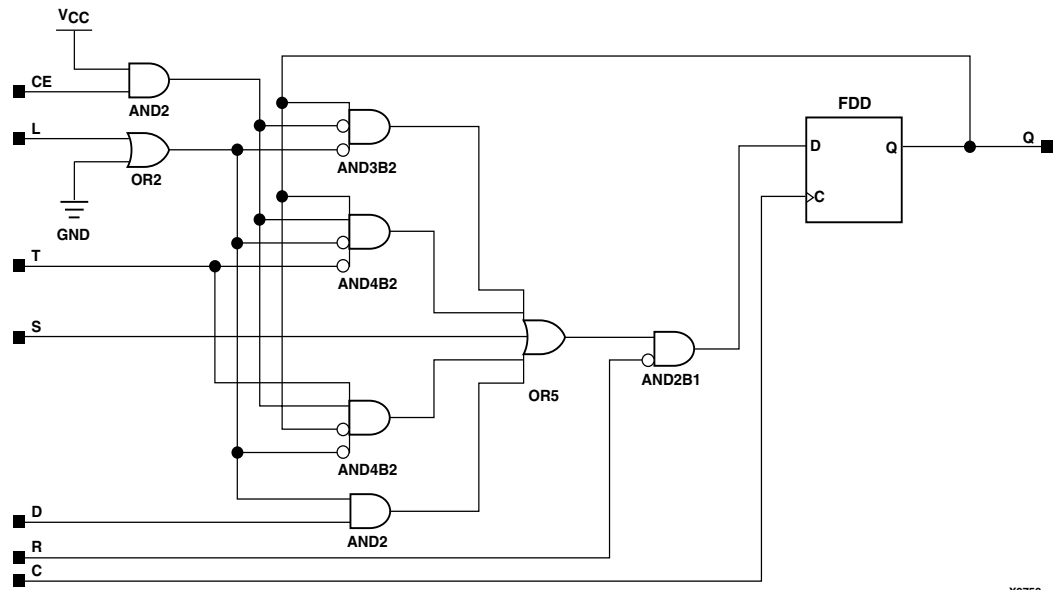
FTDRSLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTDRSLE is a dual edge triggered toggle/loadable flip-flop with toggle and clock enable and synchronous reset and set. The synchronous reset input (R), when High, overrides all other inputs and resets the data output (Q) Low. (Reset has precedence over Set.) When R is Low and synchronous set input (S) is High, the clock enable input (CE) is overridden and output Q is set High. When R and S are Low and load enable input (L) is High, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High and High-to-Low clock transitions. When R, S, and L are Low and CE is High, output Q toggles, or changes state, during the Low-to-High and High-to-Low clock transitions. When CE is Low, clock transitions are ignored.

The flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs							Outputs
R	S	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	0
1	0	X	X	X	X	↓	0
0	1	X	X	X	X	↑	1
0	1	X	X	X	X	↓	1
0	0	1	X	X	1	↑	1
0	0	1	X	X	1	↓	1
0	0	1	X	X	0	↑	0
0	0	1	X	X	0	↓	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle
0	0	0	1	1	X	↓	Toggle



X9758

FTDRSLE Implementation CoolRunner-II

Usage

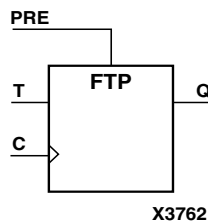
For HDL, this design element is inferred rather than instantiated.

FTP

Toggle Flip-Flop with Toggle Enable and Asynchronous Preset

Architectures Supported

FTP	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTP is a toggle flip-flop with toggle enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When toggle-enable input (T) is High and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock (C) transition.

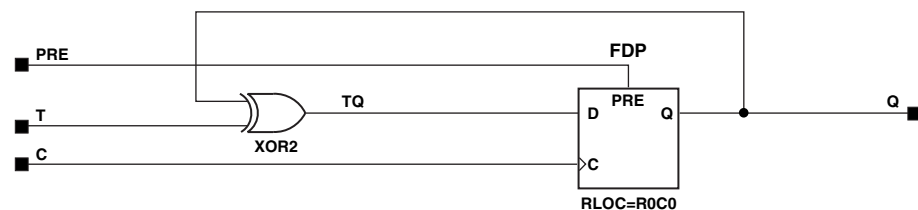
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset to output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

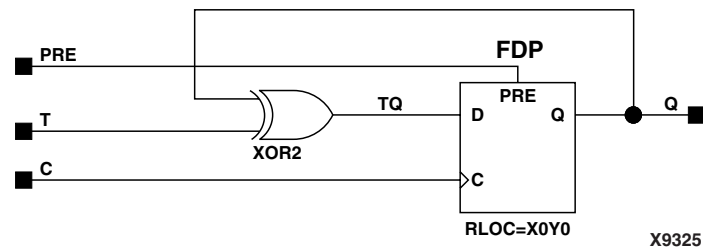
The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_VIRTEX, STARTUP_SPARTAN3, or the STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	T	C	Q
1	X	X	1
0	0	X	No Chg
0	1	↑	Toggle

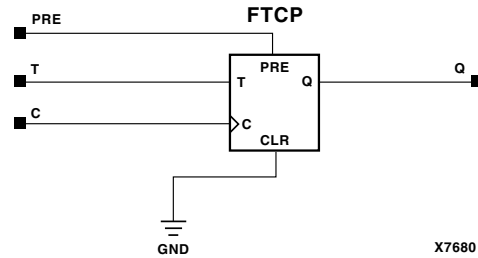


X6371

FTP Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTP Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTP Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

-- Component Declaration for FTP should be placed
 -- after architecture statement but before begin keyword

```
component FTP
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        PRE : in STD_ULOGIC;
        T : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for FTP
 -- should be placed after architecture declaration but
 -- before the begin keyword

-- Enter attributes here

-- Component Instantiation for FTP should be placed
 -- in architecture after the begin keyword

```
FTP_INSTANCE_NAME : FTP
  port map (Q => user_Q,
            C => user_C,
            PRE => user_PRE,
            T => user_T);
```

Verilog Instantiation Template

```
FTP FTP_instance_name (.Q (user_Q),  
                        .C (user_C),  
                        .PRE (user_PRE),  
                        .T (user_T));
```

Commonly Used Constraints

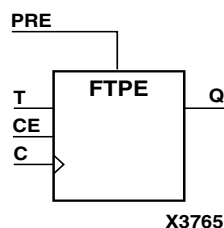
BLKNM, HBLKNM, HU_SET, INIT, IOB, LOC, REG, RLOC, TIMEGRP, TNM, U_SET, XBLKNM

FTPE

Toggle Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

Architectures Supported

FTPE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTPE is a toggle flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset (PRE) input is High, all other inputs are ignored and output Q is set High. When the toggle enable input (T) is High, clock enable (CE) is High, and PRE is Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

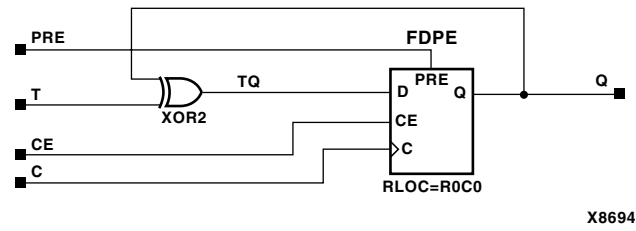
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset to output High, when power is applied.

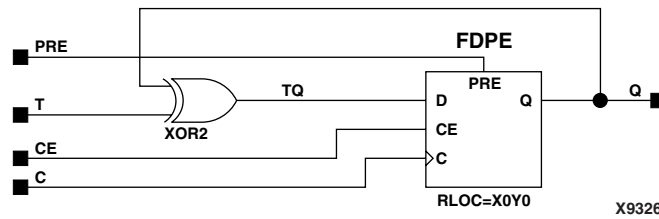
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

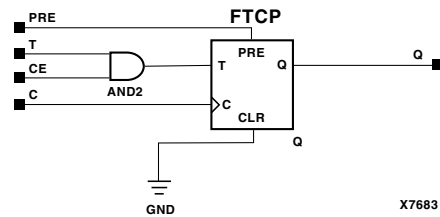
Inputs				Outputs
PRE	CE	T	C	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	X	No Chg
0	1	1	↑	Toggle



FTPE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTPE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTPE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

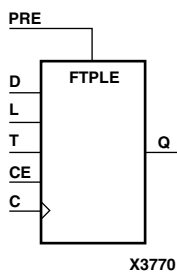
For HDL, this design element is inferred rather than instantiated.

FTPLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Asynchronous Preset

Architectures Supported

FTPLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTPLE is a toggle/loadable flip-flop with toggle and clock enable and asynchronous preset. When the asynchronous preset input (PRE) is High, all other inputs are ignored and output Q is set High. When the load enable input (L) is High and PRE is Low, the clock enable (CE) is overridden and the data (D) is loaded into the flip-flop during the Low-to-High clock transition. When L and PRE are Low and toggle-enable input (T) and CE are High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

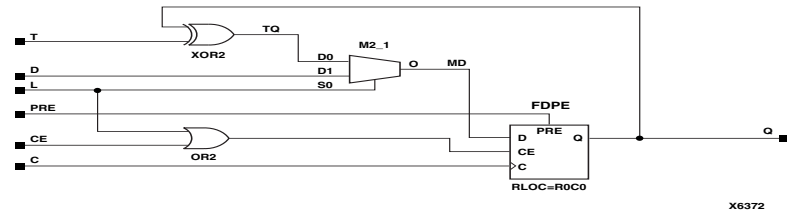
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously cleared, output Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously preset to output High, when power is applied.

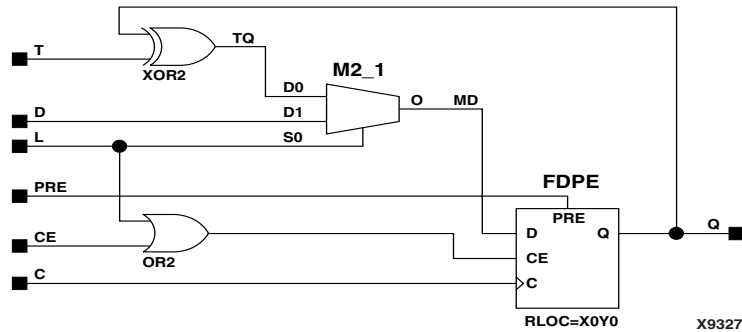
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

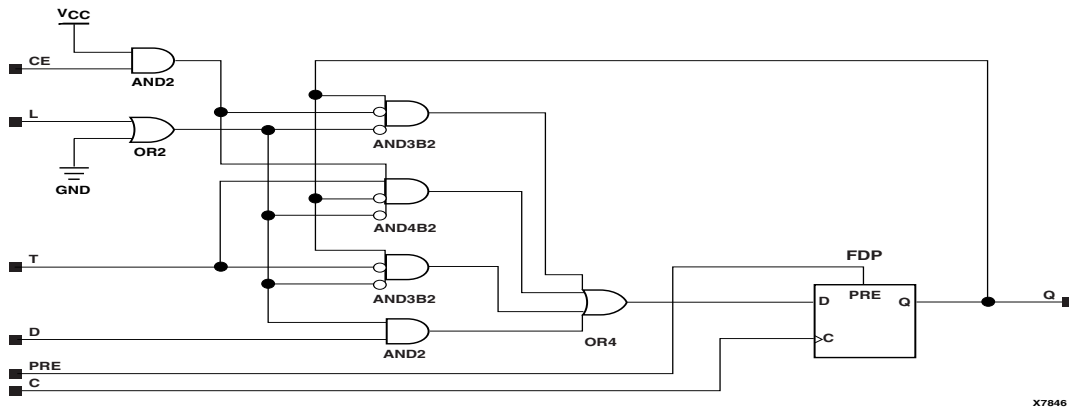
Inputs						Outputs
PRE	L	CE	T	D	C	Q
1	X	X	X	X	X	1
0	1	X	X	1	↑	1
0	1	X	X	0	↑	0
0	0	0	X	X	X	No Chg
0	0	1	0	X	X	No Chg
0	0	1	1	X	↑	Toggle



FTPLE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



FTPLE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTPLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

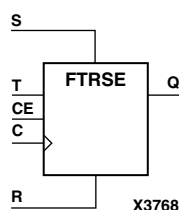
For HDL, this design element is inferred rather than instantiated.

FTRSE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Architectures Supported

FTRSE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTRSE is a toggle flip-flop with toggle and clock enable and synchronous reset and set. When the synchronous reset input (R) is High, it overrides all other inputs and the data output (Q) is reset Low. When the synchronous set input (S) is High and R is Low, clock enable input (CE) is overridden and output Q is set High. (Reset has precedence over Set.) When toggle enable input (T) and CE are High and R and S are Low, output Q toggles, or changes state, during the Low-to-High clock transition.

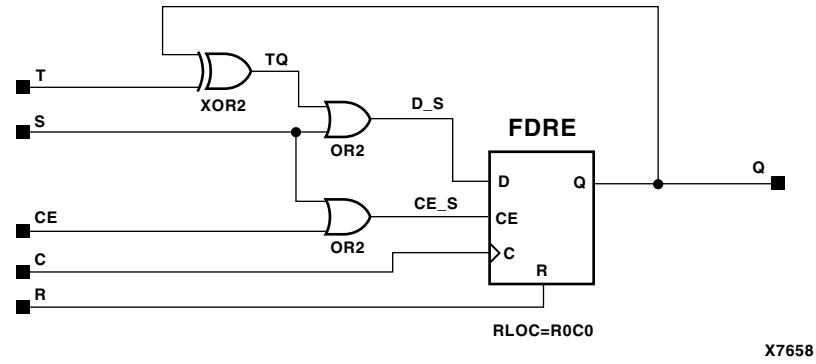
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

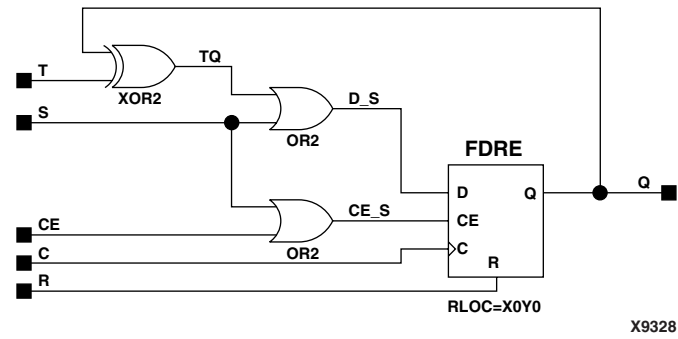
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

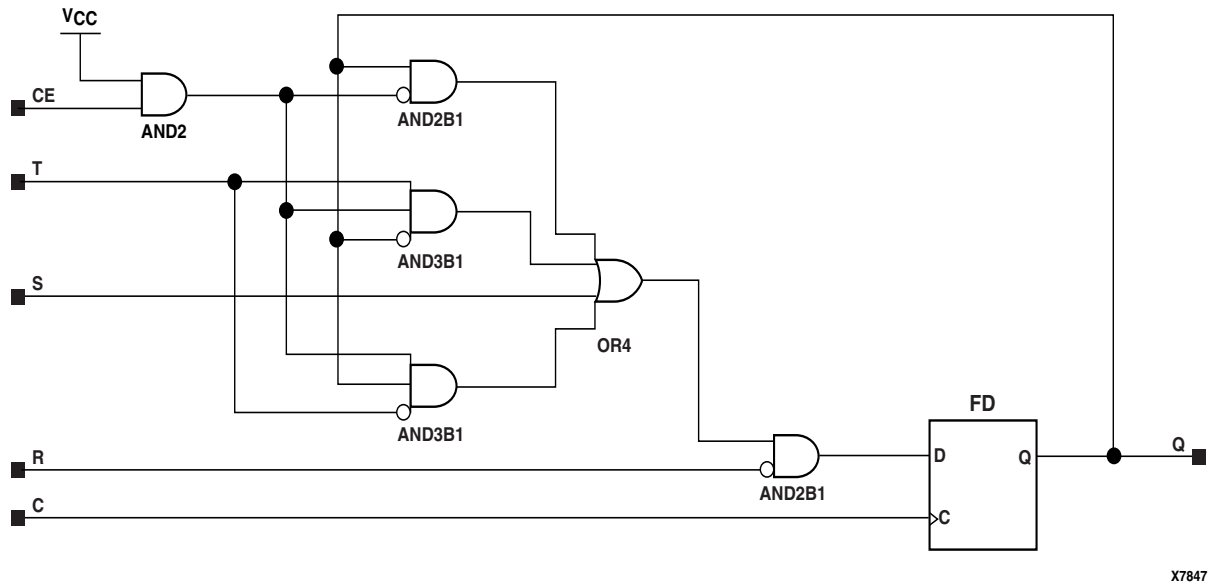
Inputs					Outputs
R	S	CE	T	C	Q
1	X	X	X	↑	0
0	1	X	X	↑	1
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTRSE Implementation Spartan-II, Spartan-II-E, Virtex, Virtex-E



FTRSE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTRSE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

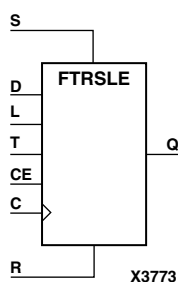
For HDL, this design element is inferred rather than instantiated.

FTRSLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Reset and Set

Architectures Supported

FTRSLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTRSLE is a toggle/loadable flip-flop with toggle and clock enable and synchronous reset and set. The synchronous reset input (R), when High, overrides all other inputs and resets the data output (Q) Low. (Reset has precedence over Set.) When R is Low and synchronous set input (S) is High, the clock enable input (CE) is overridden and output Q is set High. When R and S are Low and load enable input (L) is High, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When R, S, and L are Low, CE is High and T is High, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

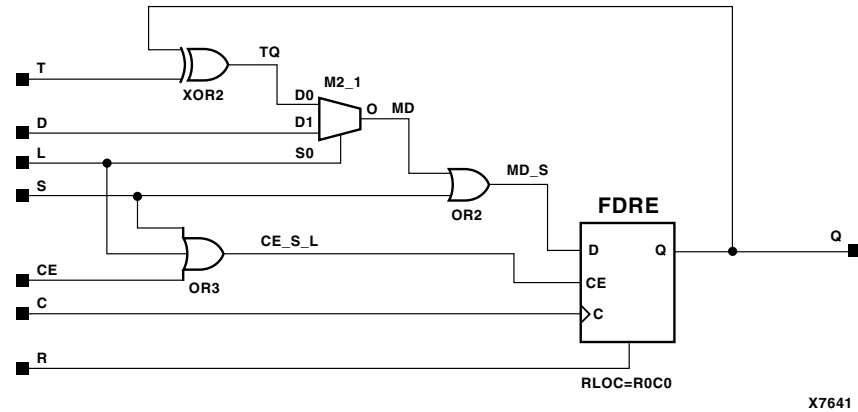
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

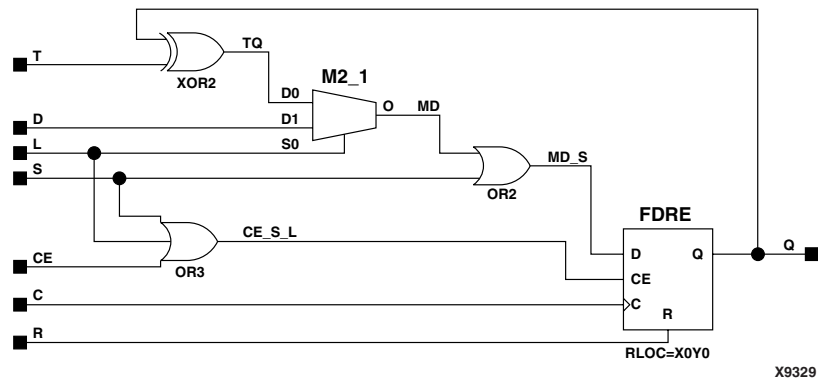
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

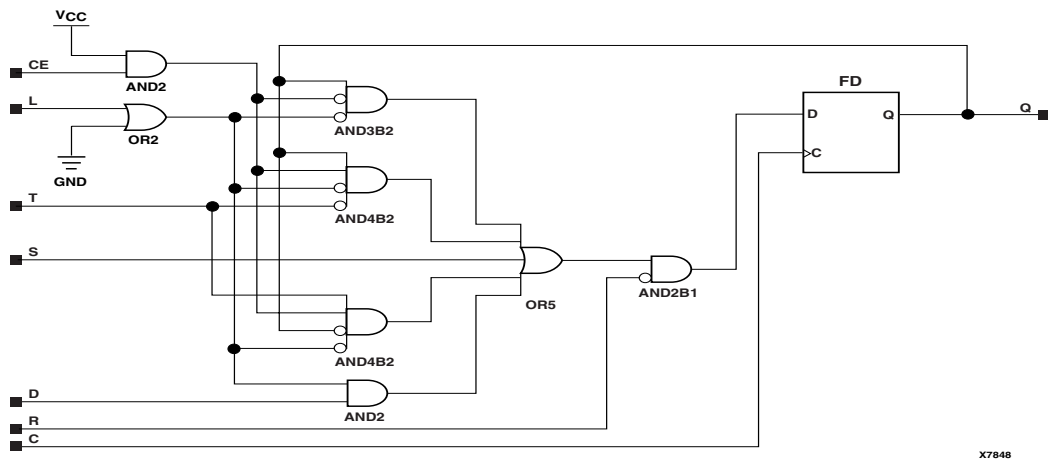
Inputs							Outputs
R	S	L	CE	T	D	C	Q
1	0	X	X	X	X	↑	0
0	1	X	X	X	X	↑	1
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle



FTRSLE Implementation Spartan-II, Spartan-II E, Virtex, Virtex-E



FTRSLE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTRSLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

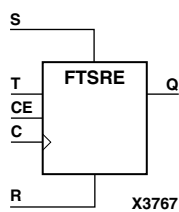
For HDL, this design element is inferred rather than instantiated.

FTSRE

Toggle Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

Architectures Supported

FTSRE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FTSRE is a toggle flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input, when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset input (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When toggle enable input (T) and CE are High and S and R are Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

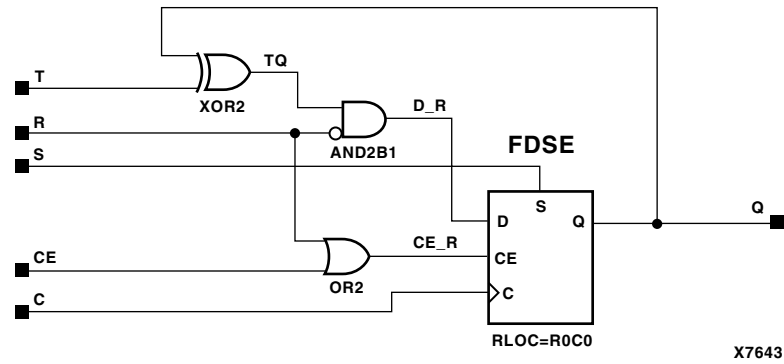
The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

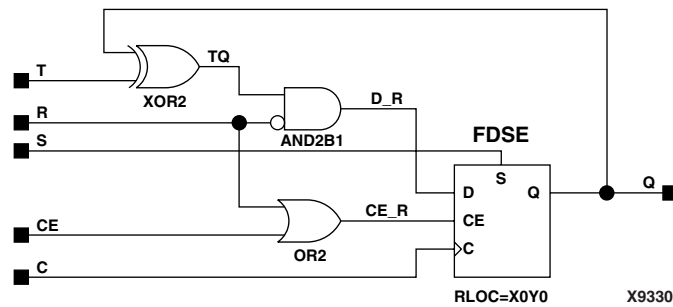
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. FTSRE will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

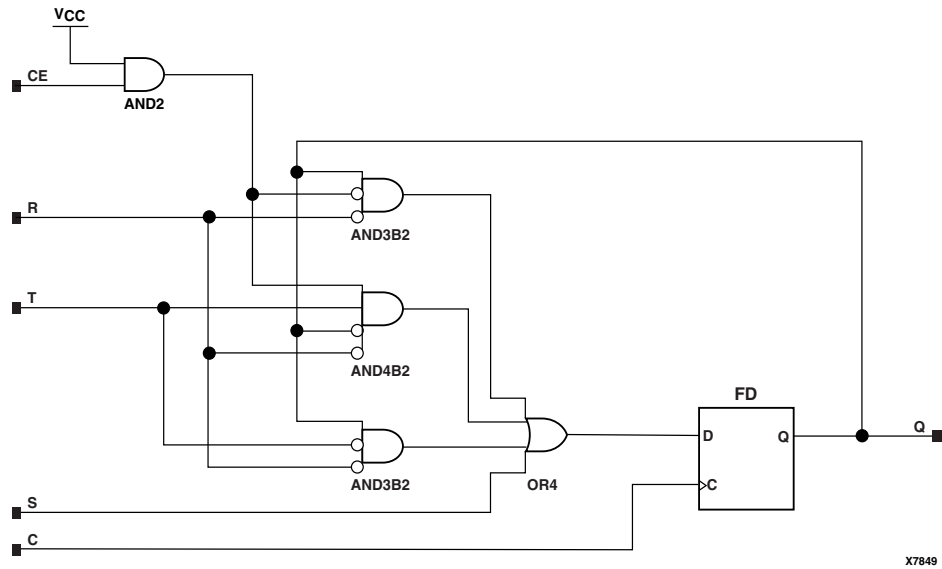
Inputs					Outputs
S	R	CE	T	C	Q
1	X	X	X	↑	1
0	1	X	X	↑	0
0	0	0	X	X	No Chg
0	0	1	0	X	No Chg
0	0	1	1	↑	Toggle



FTSRE Implementation Spartan-II, Spartan-II-E, Virtex, Virtex-E



FTSRE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTSRE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

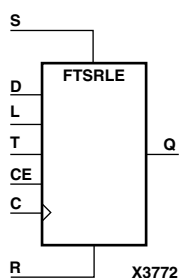
For HDL, this design element is inferred rather than instantiated.

F TSRLE

Toggle/Loadable Flip-Flop with Toggle and Clock Enable and Synchronous Set and Reset

Architectures Supported

F TSRLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



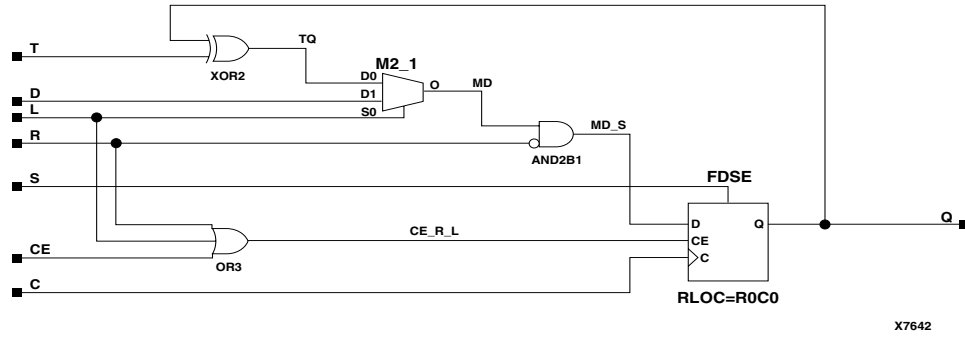
F TSRLE is a toggle/loadable flip-flop with toggle and clock enable and synchronous set and reset. The synchronous set input (S), when High, overrides all other inputs and sets data output (Q) High. (Set has precedence over Reset.) When synchronous reset (R) is High and S is Low, clock enable input (CE) is overridden and output Q is reset Low. When load enable input (L) is High and S and R are Low, CE is overridden and data on data input (D) is loaded into the flip-flop during the Low-to-High clock transition. When the toggle enable input (T) and CE are High and S, R, and L are Low, output Q toggles, or changes state, during the Low-to-High clock transition. When CE is Low, clock transitions are ignored.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the flip-flop is asynchronously preset when a High-level pulse is applied on the PRLD global net.

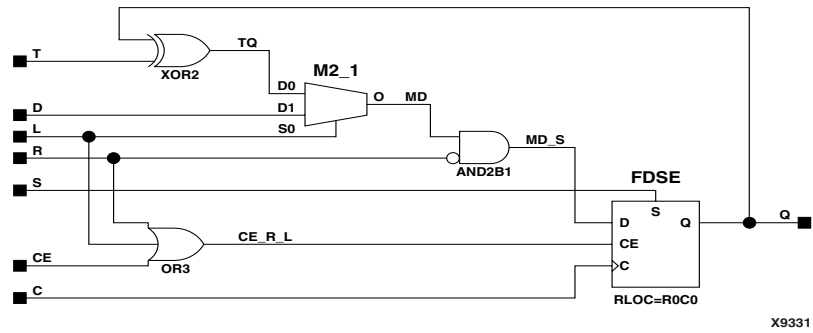
For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is asynchronously cleared, output Low, when global set/reset (GSR) is active.

The GSR active level defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol. F TSRLE will set when GSR is active. For Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the flip-flop is preset to active high when GSR is active.

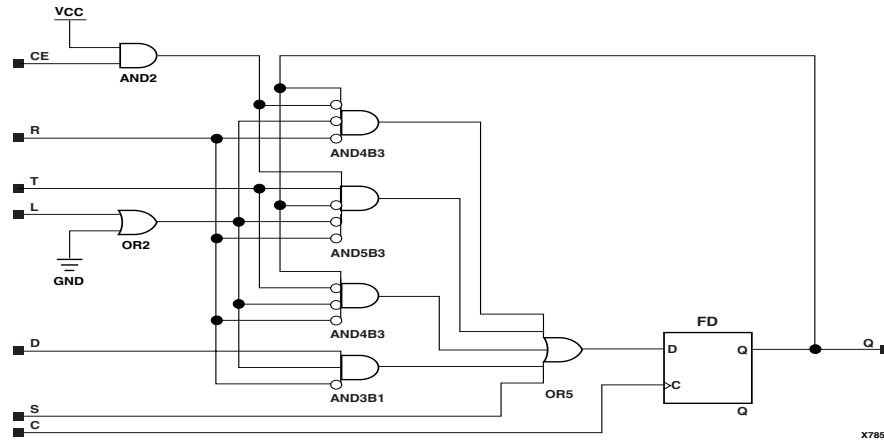
Inputs							Outputs
S	R	L	CE	T	D	C	Q
1	X	X	X	X	X	↑	1
0	1	X	X	X	X	↑	0
0	0	1	X	X	1	↑	1
0	0	1	X	X	0	↑	0
0	0	0	0	X	X	X	No Chg
0	0	0	1	0	X	X	No Chg
0	0	0	1	1	X	↑	Toggle



FTSRLE Implementation Spartan-II, Spartan-II-E, Virtex, Virtex-E



FTSRLE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



FTSRLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, this design element is inferred rather than instantiated.

GND

Ground-Connection Signal Tag

Architectures Supported

GND	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

The GND signal tag, or parameter, forces a net or input function to a Low logic level. A net tied to GND cannot have any other source.

When the logic-trimming software or fitter encounters a net or input function tied to GND, it removes any logic that is disabled by the GND signal. The GND signal is only implemented when the disabled logic cannot be removed.



X3858

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Instantiation Template

```
-- Component Declaration for GND should be placed
-- after architecture statement but before begin keyword

component GND
  port (G : out STD_ULOGIC);
end component;

-- Component Attribute specification for GND
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter constraints here

-- Component Instantiation for GND should be placed
-- in architecture after the begin keyword

GND_INSTANCE_NAME : GND
  port map (G => user_G);
```

Verilog Instantiation Template

```
GND GND_instance_name (.G (user_G));
```


GT_AURORA_n

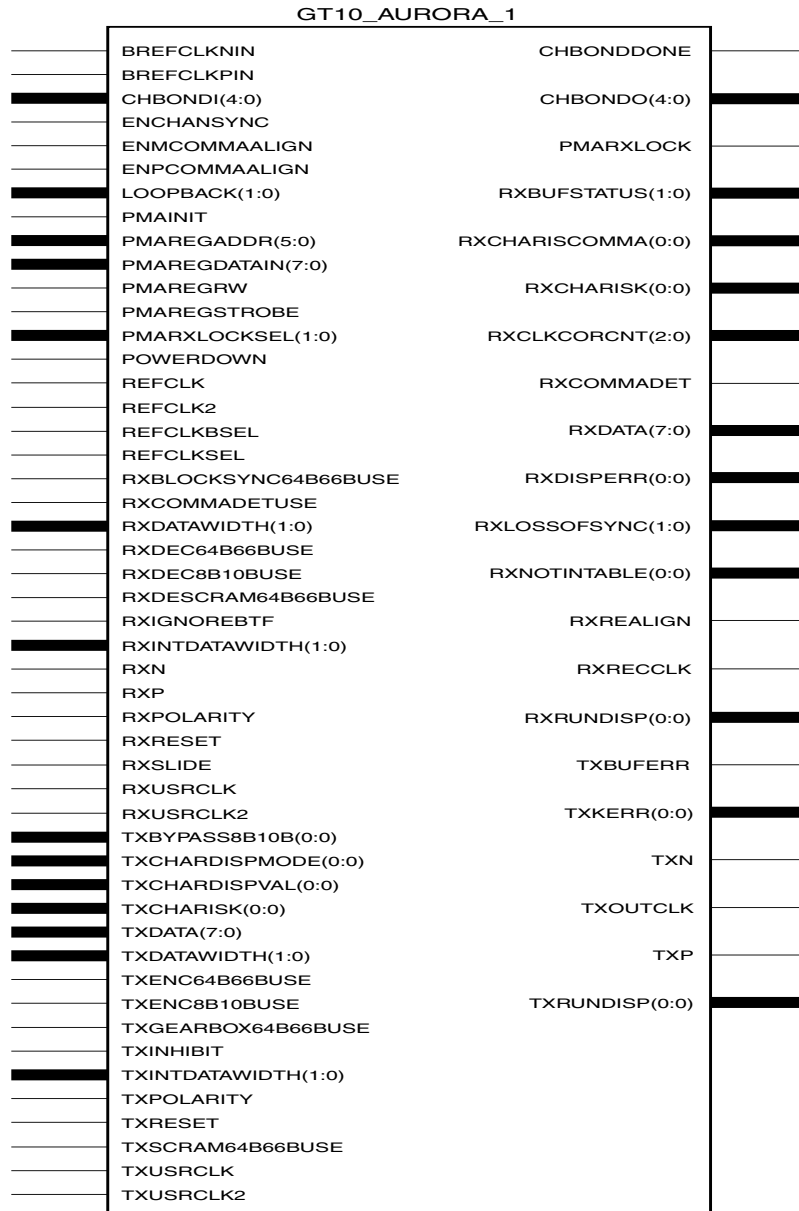
Gigabit Transceiver for High-Speed I/O

Architectures Supported

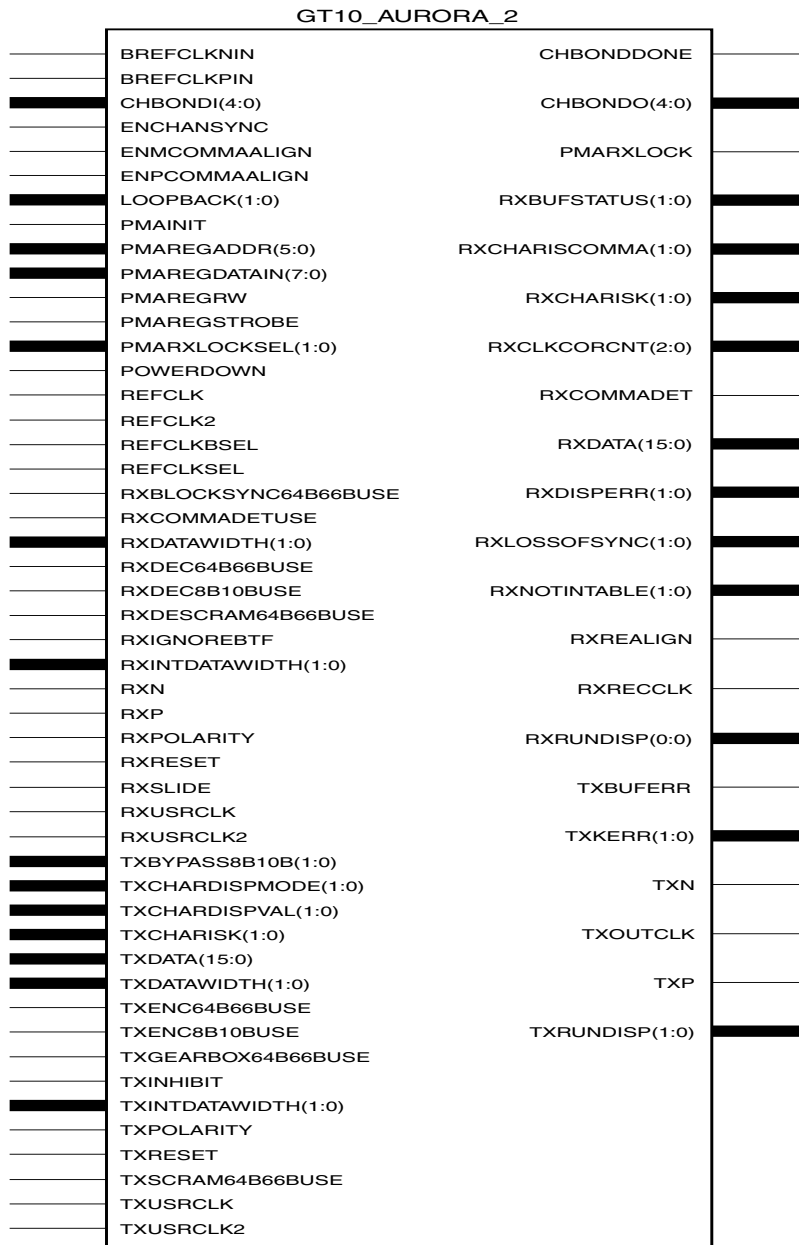
GT_AURORA_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

This Xilinx protocol gigabit transceiver supports 1, 2, and 4-byte data paths. The letter n represents number of bytes of the data path. Valid values are 1, 2 or 4.

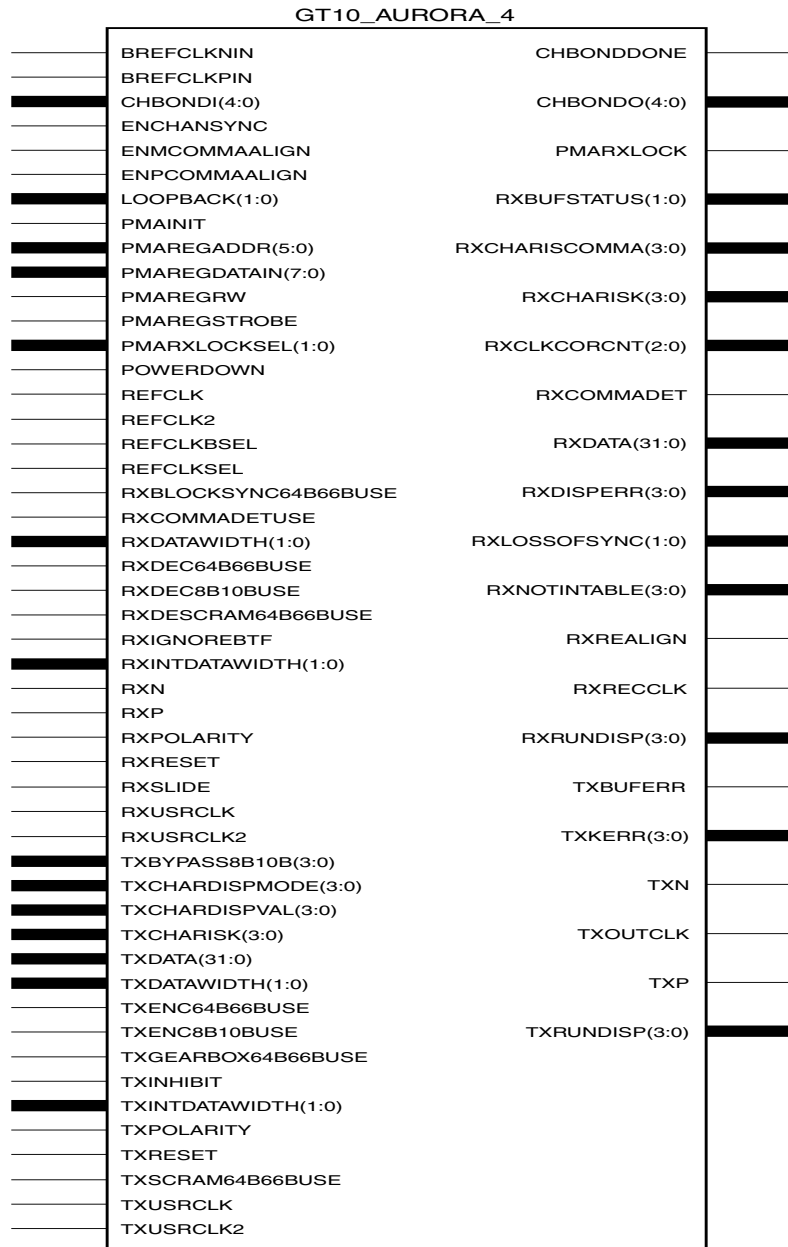
You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of n and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10045



X10046



X10047

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT_CUSTOM

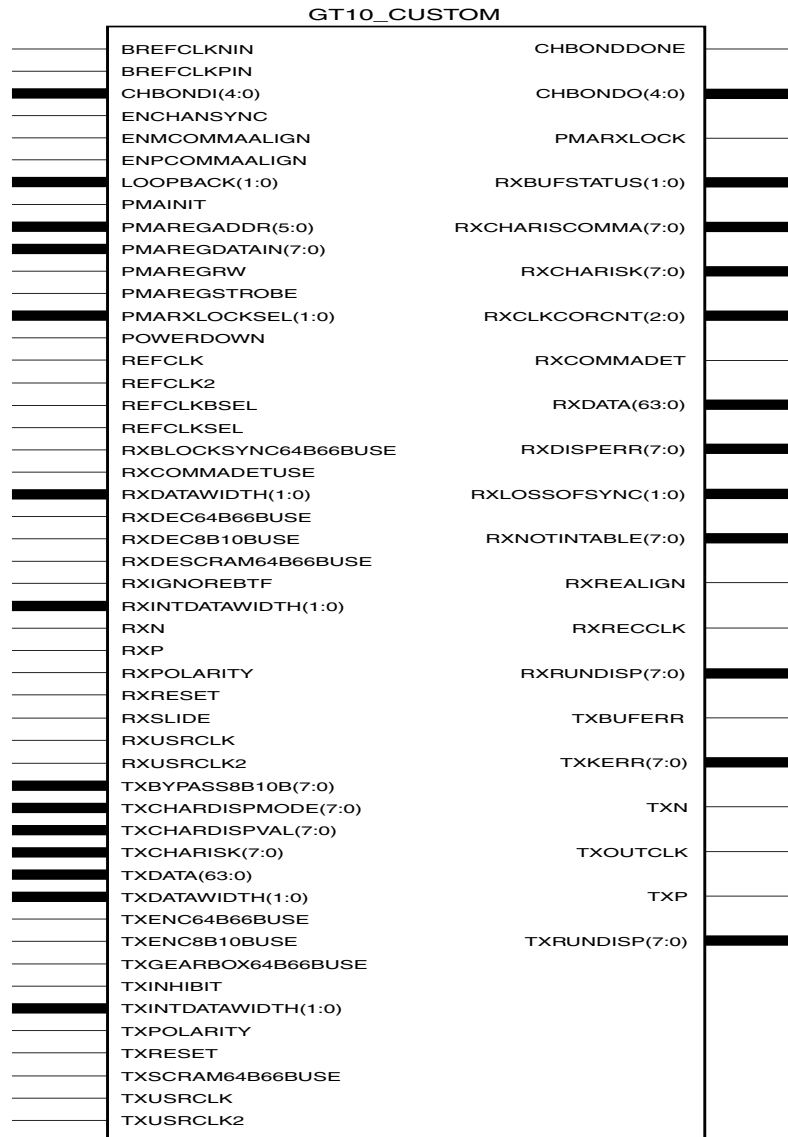
Gigabit Transceiver for High-Speed I/O

Architectures Supported

GT_CUSTOM	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

This gigabit transceiver is fully customizable. You can set attributes for the primitives. You can also set attributes for the primitives. See the *RocketIO Transceiver User Guide* for a description of these attributes and their default attribute values.

The following figure lists the input and output ports for all values of n . For a description of each of the ports, see the *RocketIO Transceiver User Guide*.



Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT_ETHERNET_n

Gigabit Transceiver for High-Speed I/O

Architectures Supported

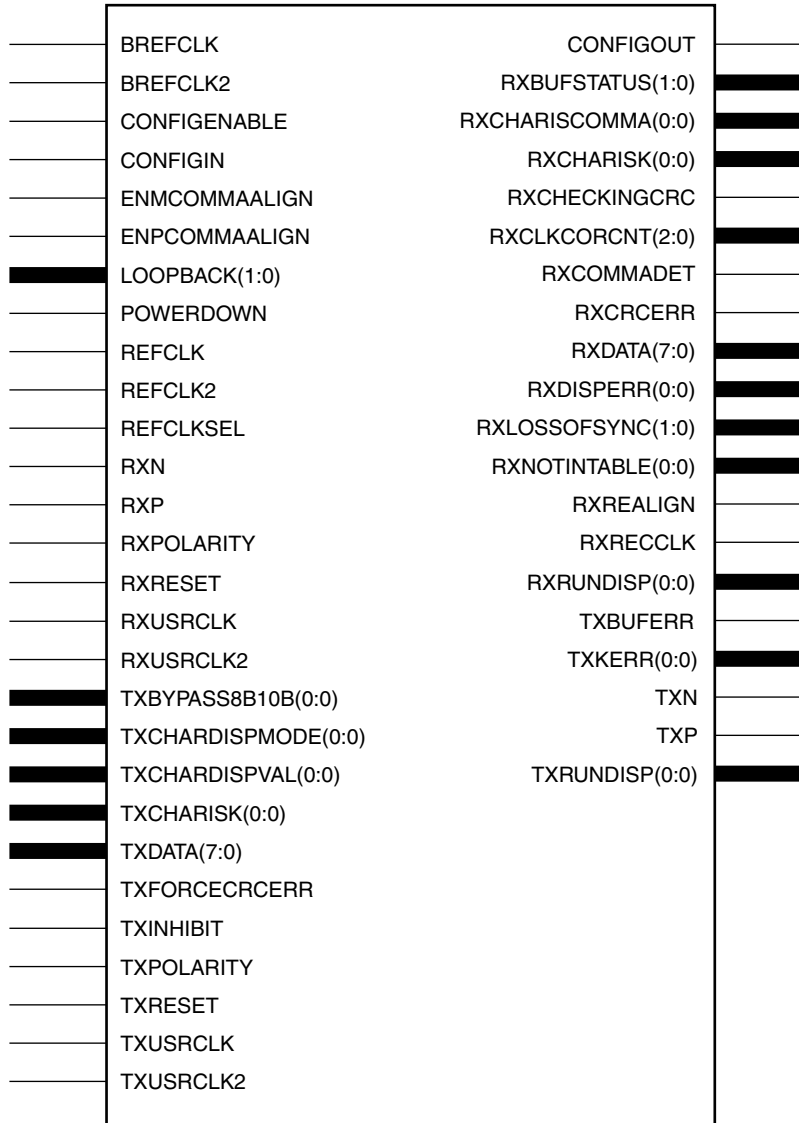
GT_ETHERNET_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

This Ethernet gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

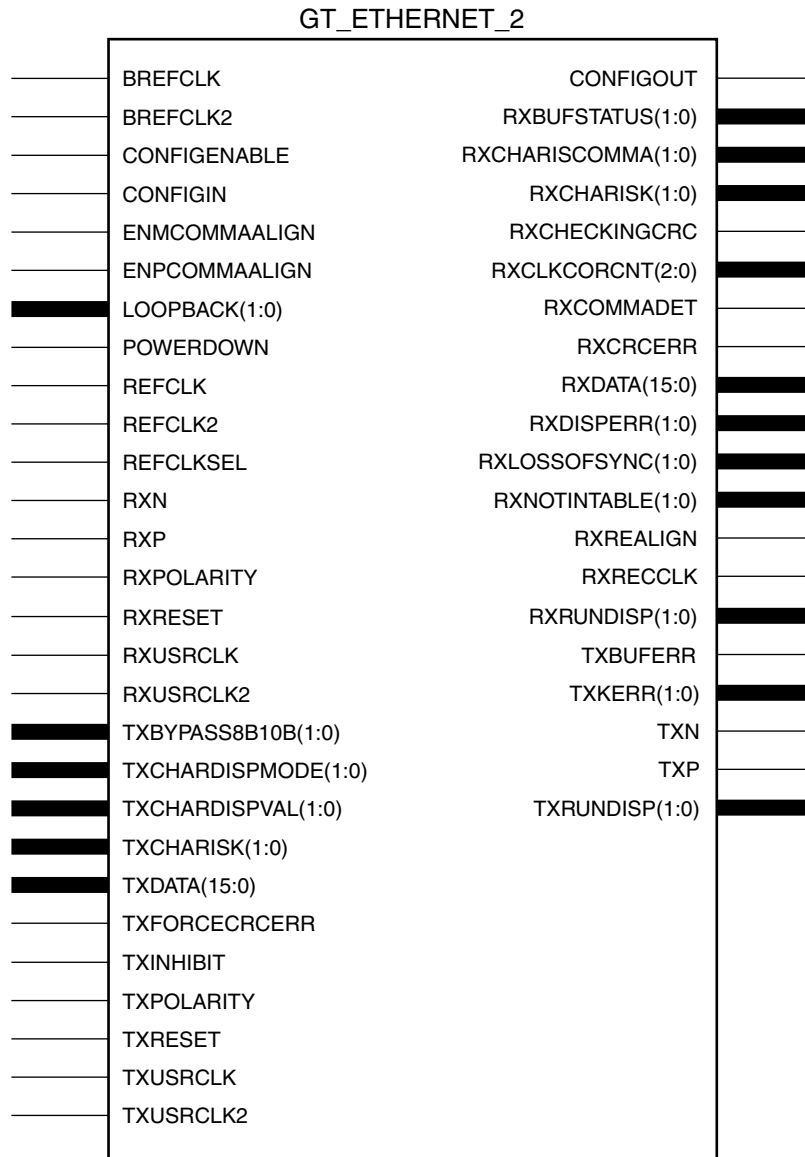
You can also set attributes for the primitives. See the *RocketIO Transceiver User Guide* for a description of these attributes and their default attribute values.

The following figures list the input and output ports for all values of *n*. For a description of each of the ports, see the *RocketIO Transceiver User Guide*.

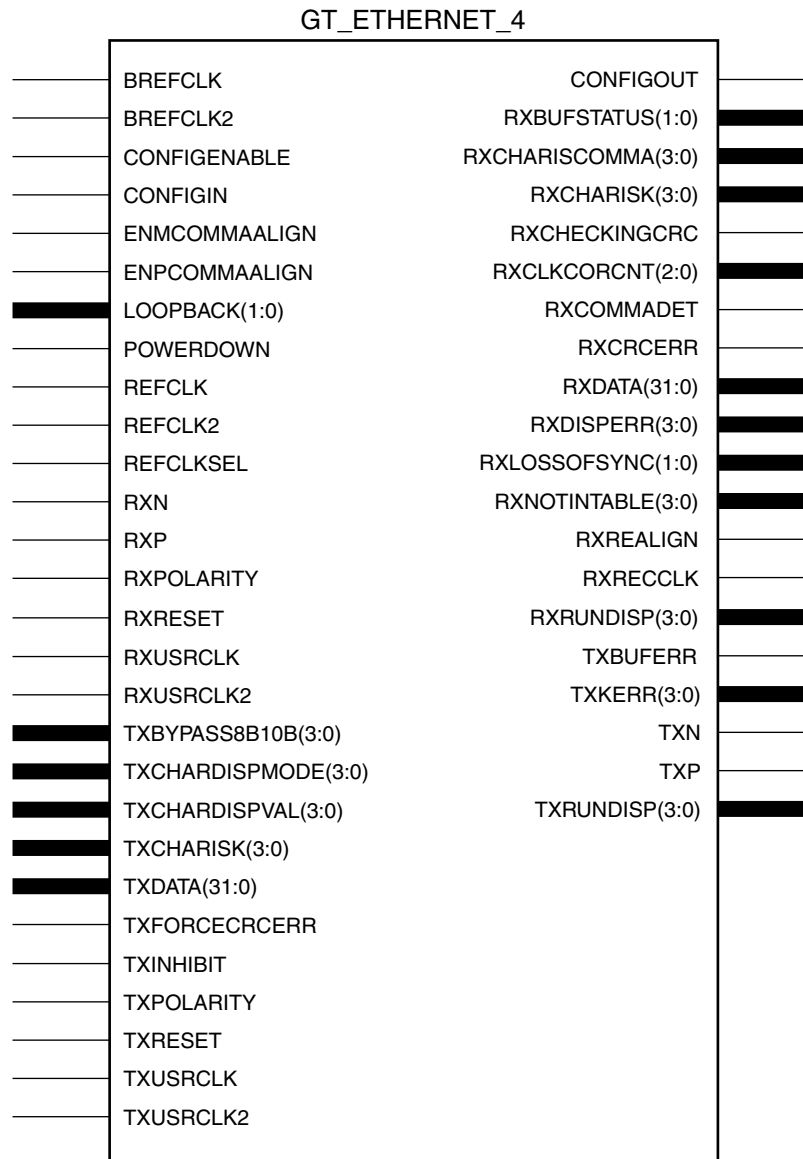
GT_ETHERNET_1



X9892



X9893



X9894

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT_FIBRE_CHAN_n

Gigabit Transceiver for High-Speed I/O

Architectures Supported

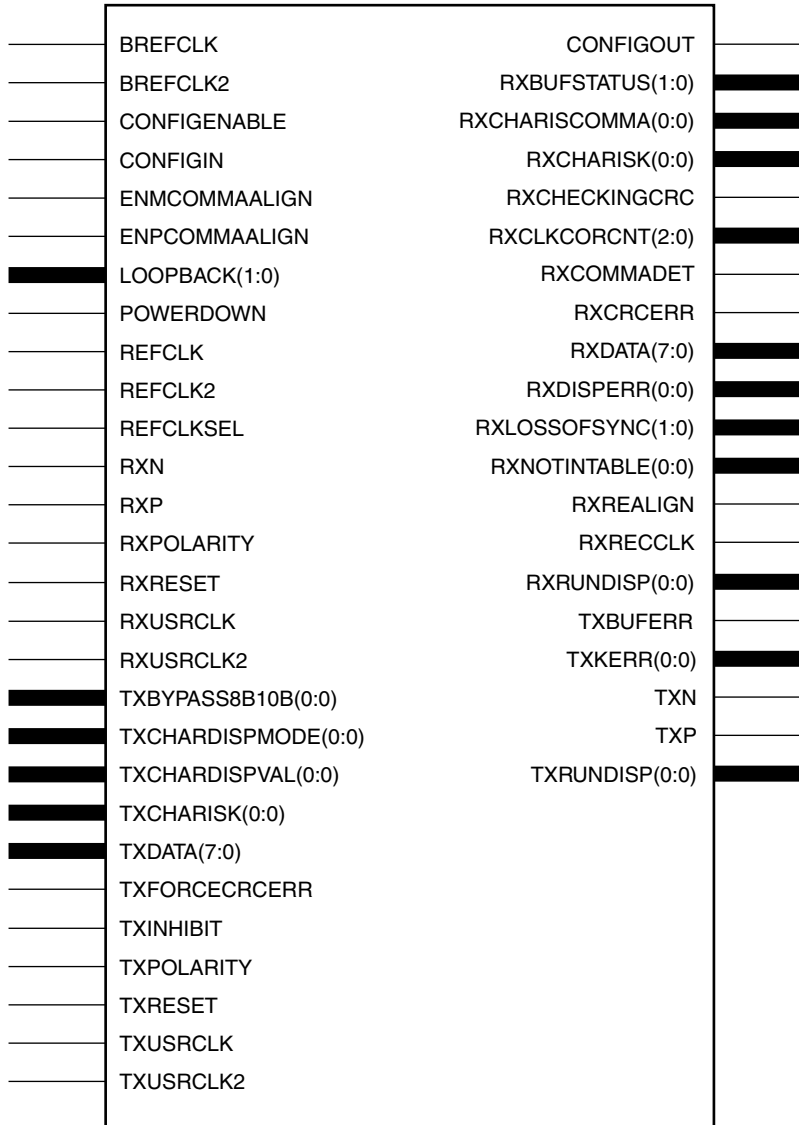
GT_FIBRE_CHAN_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

This Fibre Channel gigabit transceiver supports 1, 2, and 4-byte data paths. The letter n represents number of bytes of the data path. Valid values are 1, 2 or 4.

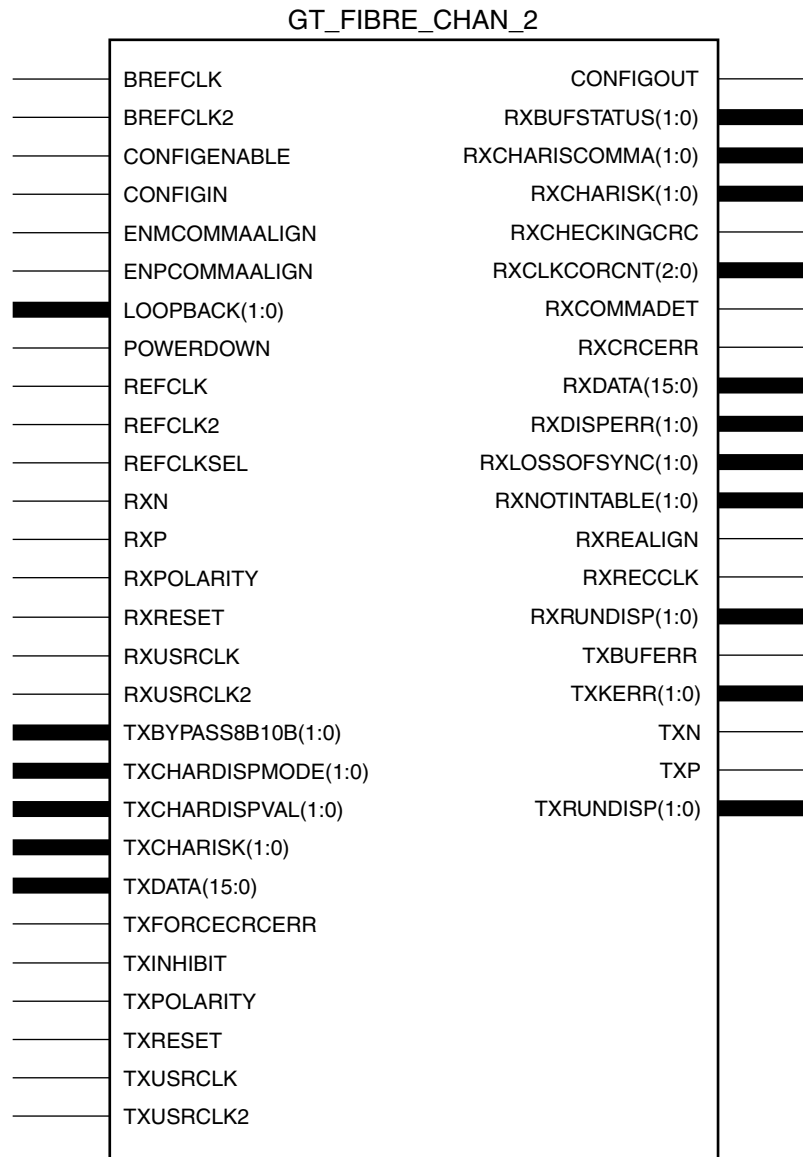
You can also set attributes for the primitives. See the *RocketIO Transceiver User Guide* for a description of these attributes and their default attribute values.

The following figures list the input and output ports for all values of n . For a description of each of the ports, see the *RocketIO Transceiver User Guide*.

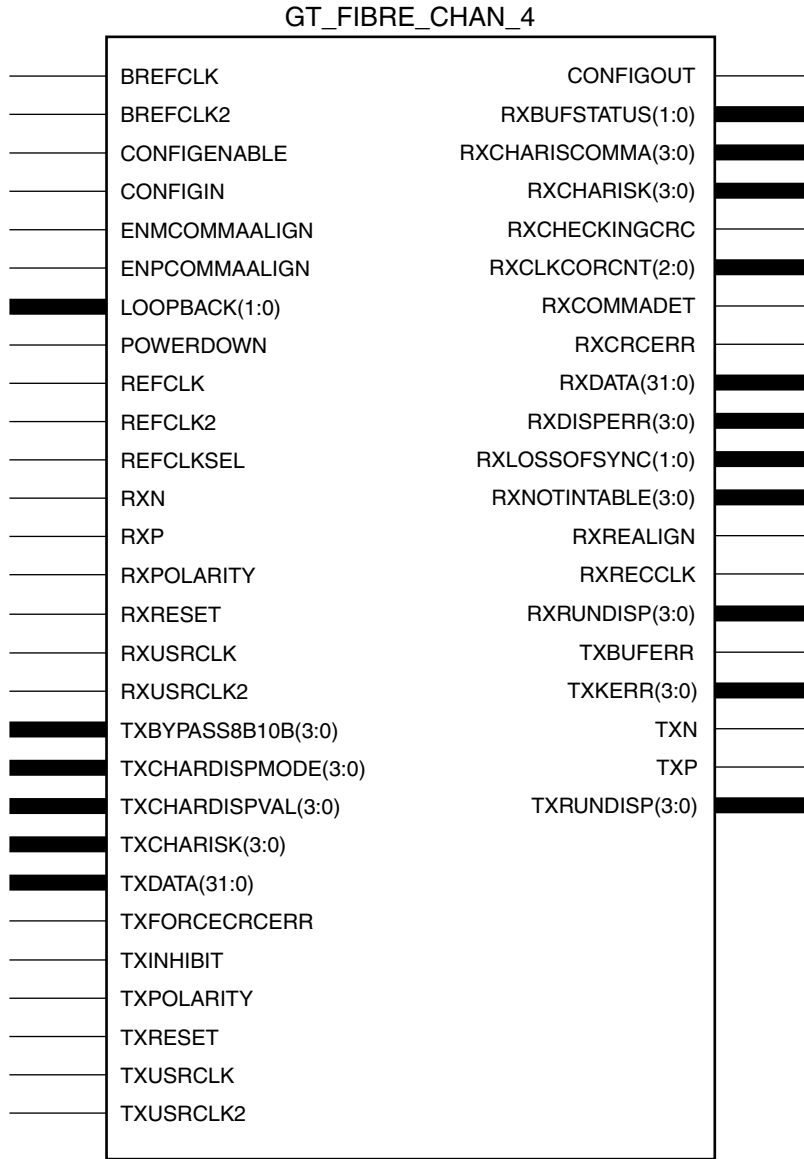
GT_FIBRE_CHAN_1



X9895



X9896



X9897

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT_INFINIBAND_n

Gigabit Transceiver for High-Speed I/O

Architectures Supported

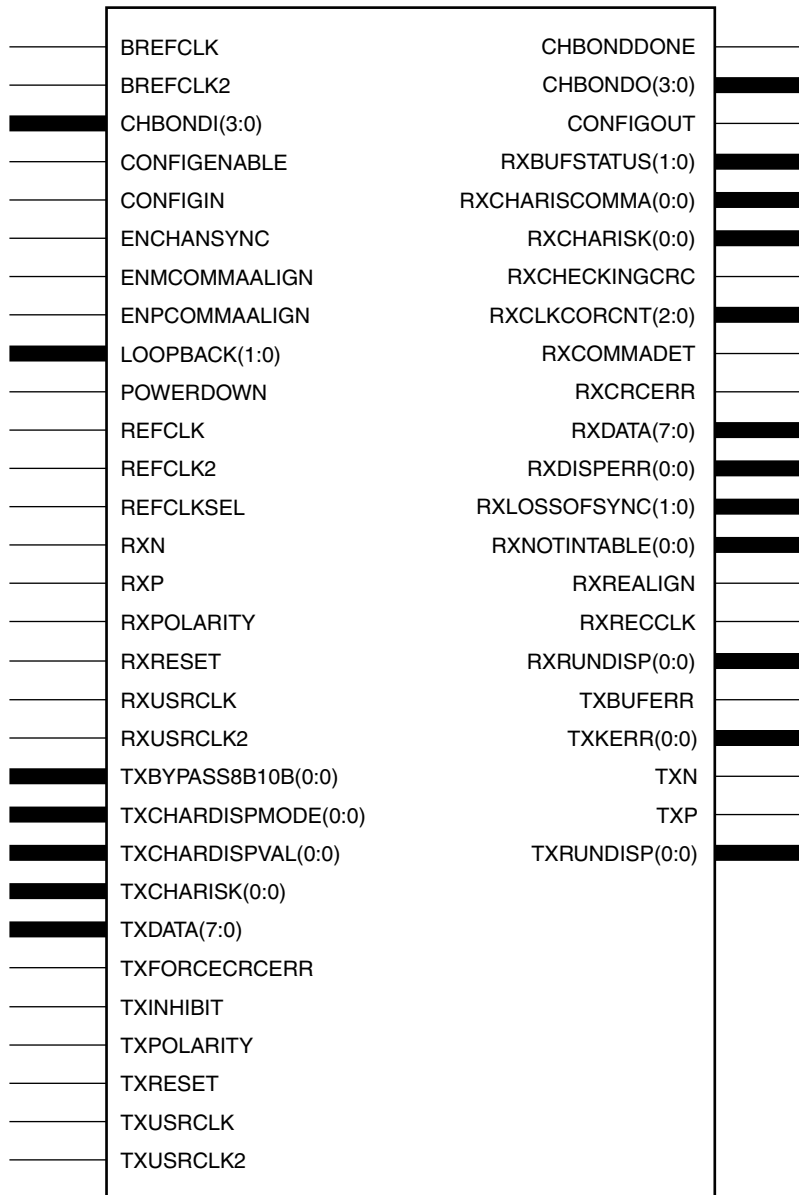
GT_INFINIBAND_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

This Infiniband gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See the *RocketIO Transceiver User Guide* for a description of these attributes and their default attribute values.

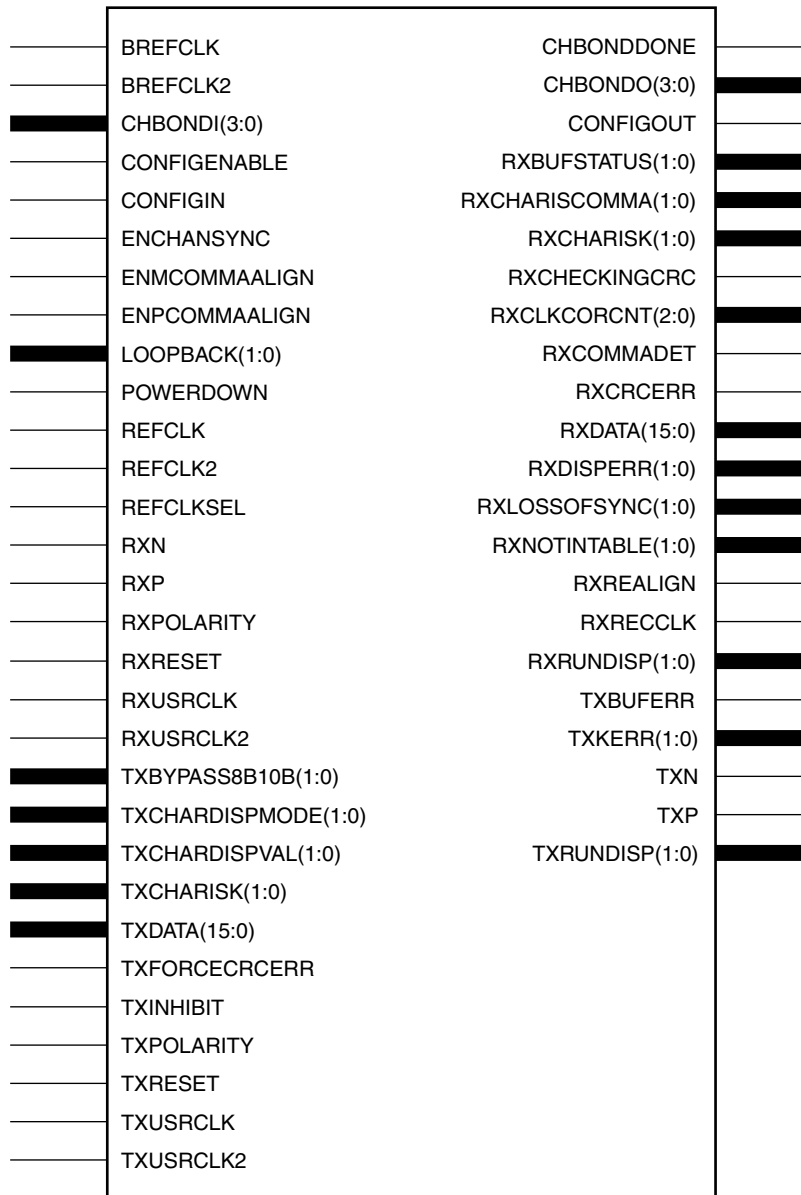
The following figures list the input and output ports for all values of *n*. For a description of each of the ports, see the *RocketIO Transceiver User Guide*.

GT_INFINIBAND_1

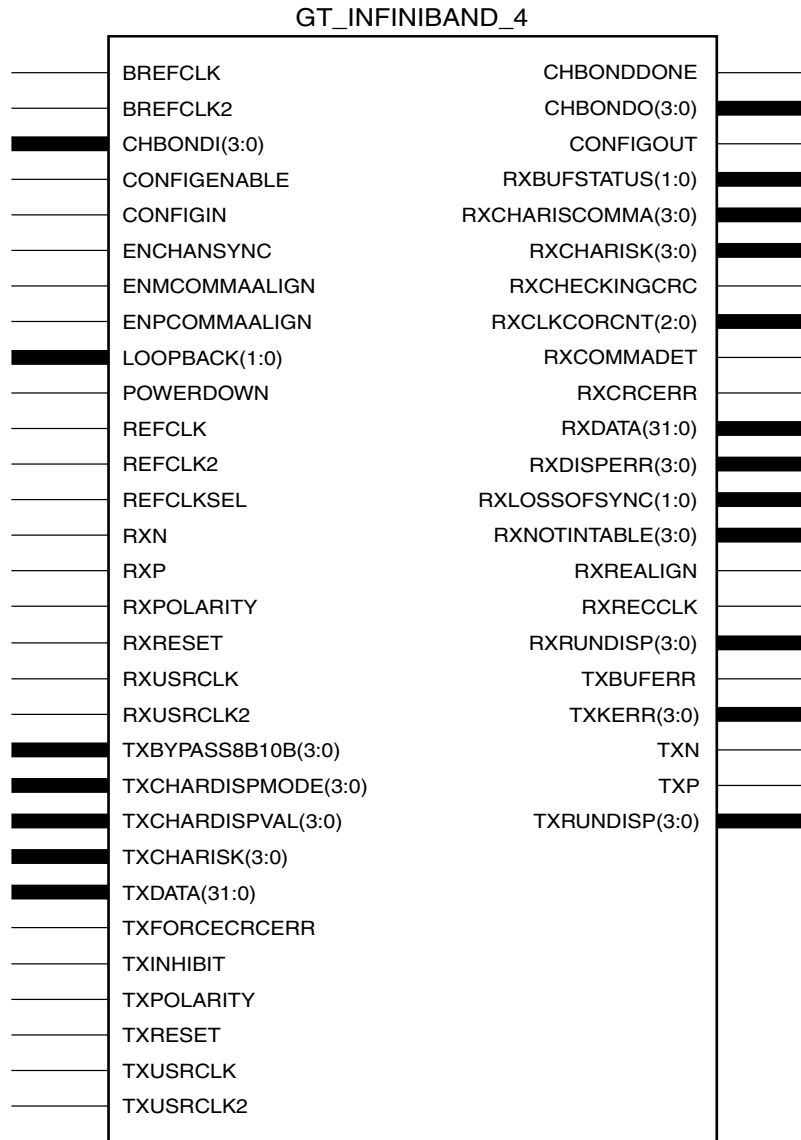


X9898

GT_INFINIBAND_2



X9899



X9900

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT_XAUI_n

Gigabit Transceiver for High-Speed I/O

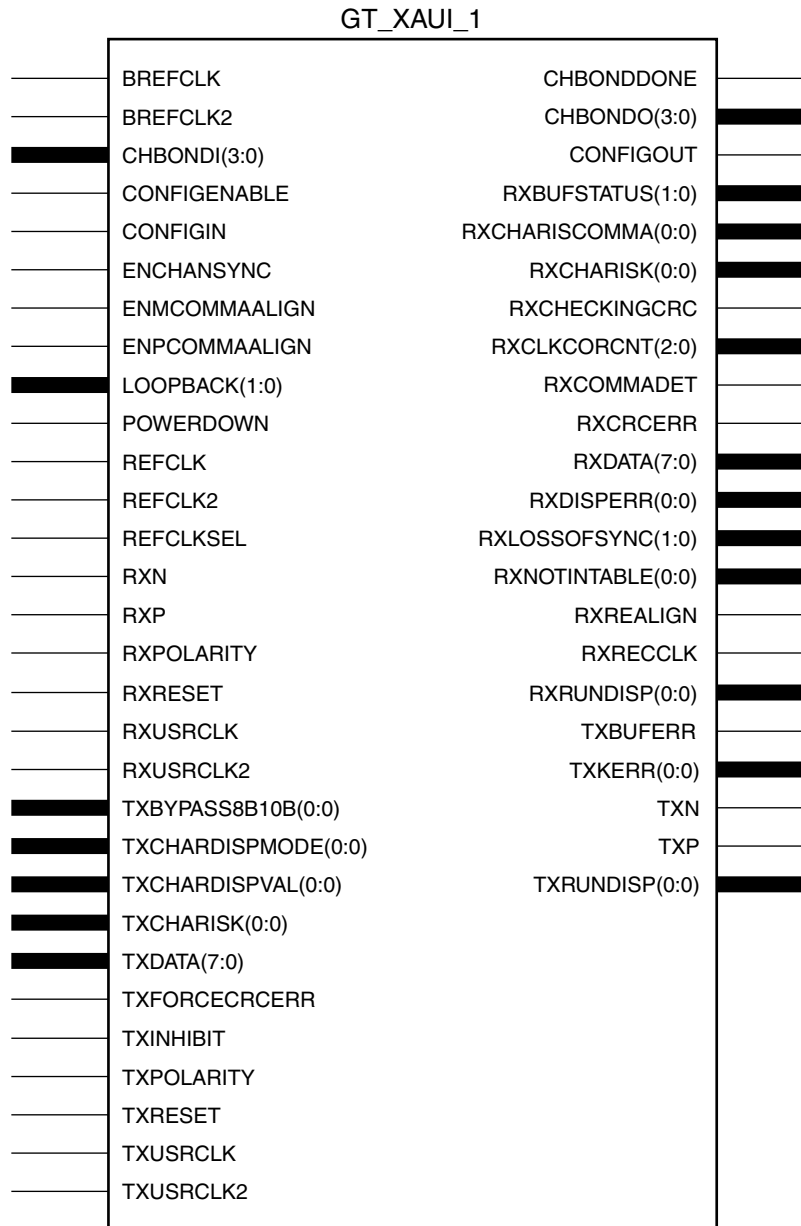
Architectures Supported

GT_XAUI_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro but not for Virtex-II or Virtex-II Pro X.	

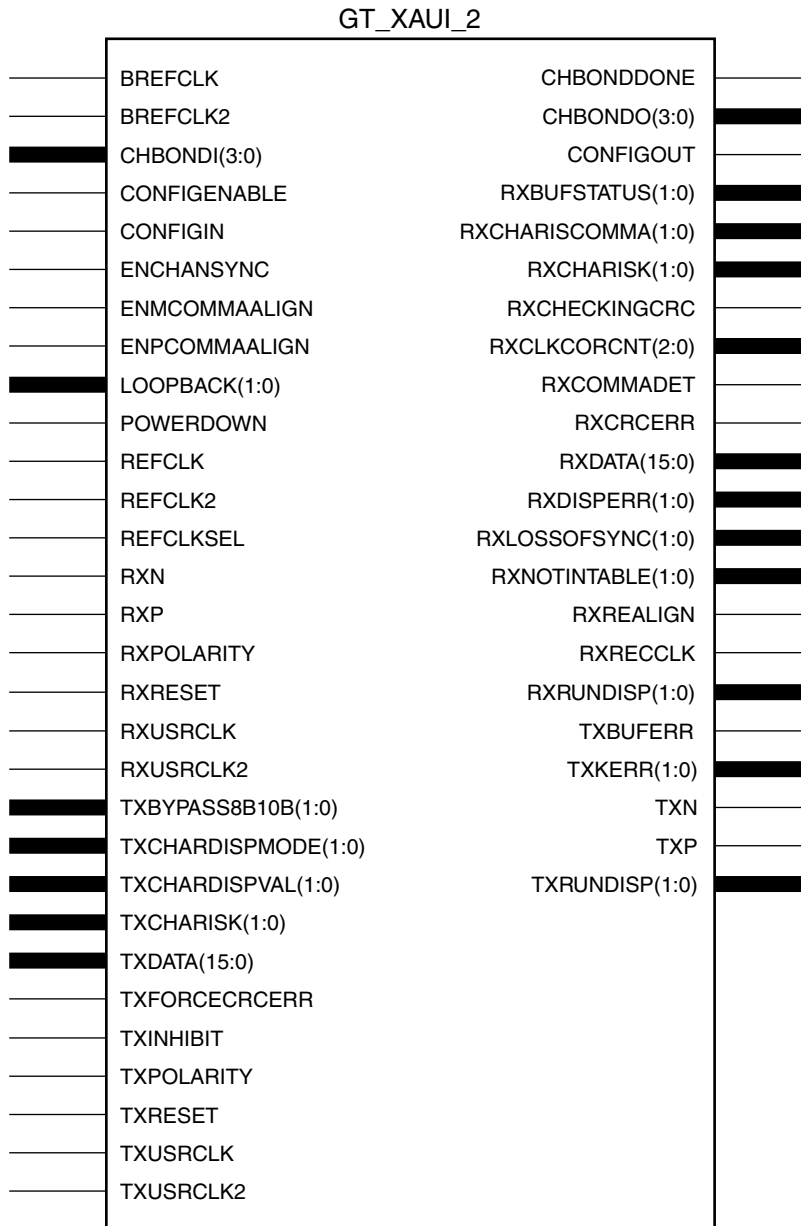
This XAUI gigabit transceiver supports 1, 2, and 4-byte data paths. The letter n represents number of bytes of the data path. Valid values are 1, 2 or 4.

You can also set attributes for the primitives. See the *RocketIO Transceiver User Guide* for a description of these attributes and their default attribute values.

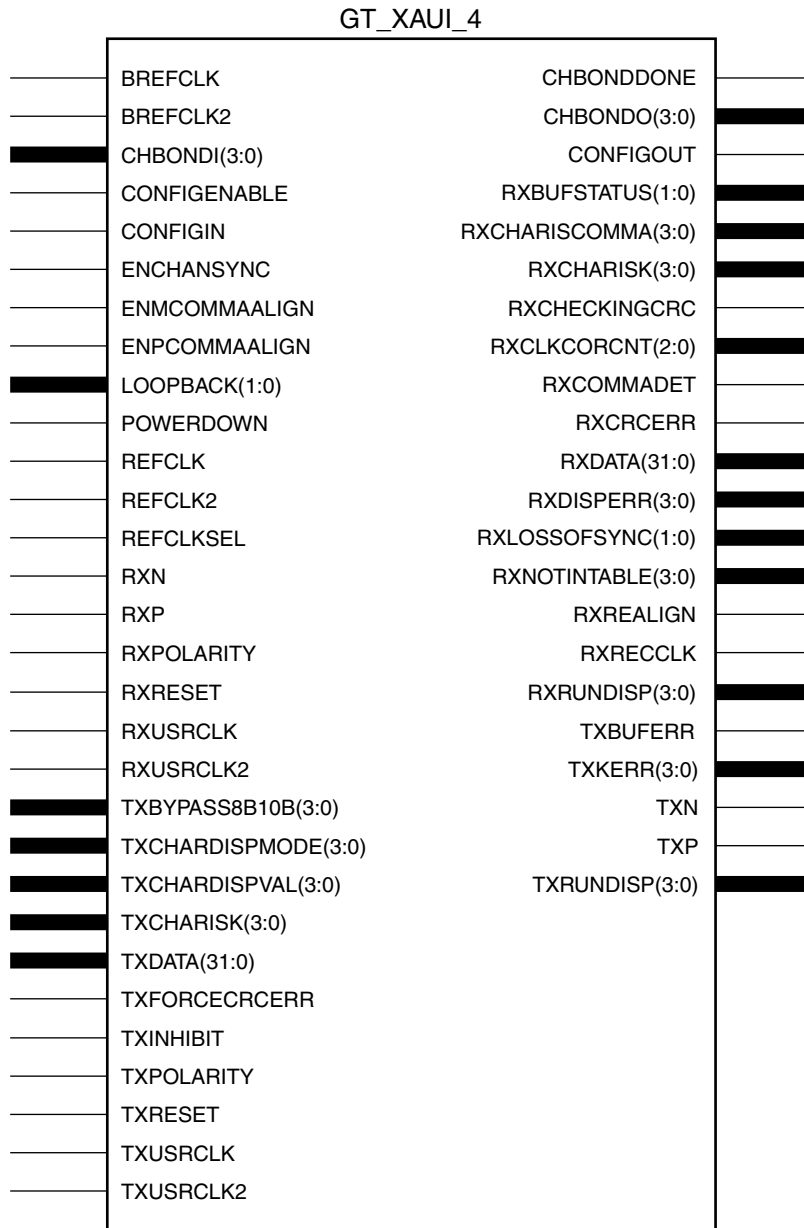
The following figures list the input and output ports for all values of n . For a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X9902



X9903



X9904

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_10GE_n

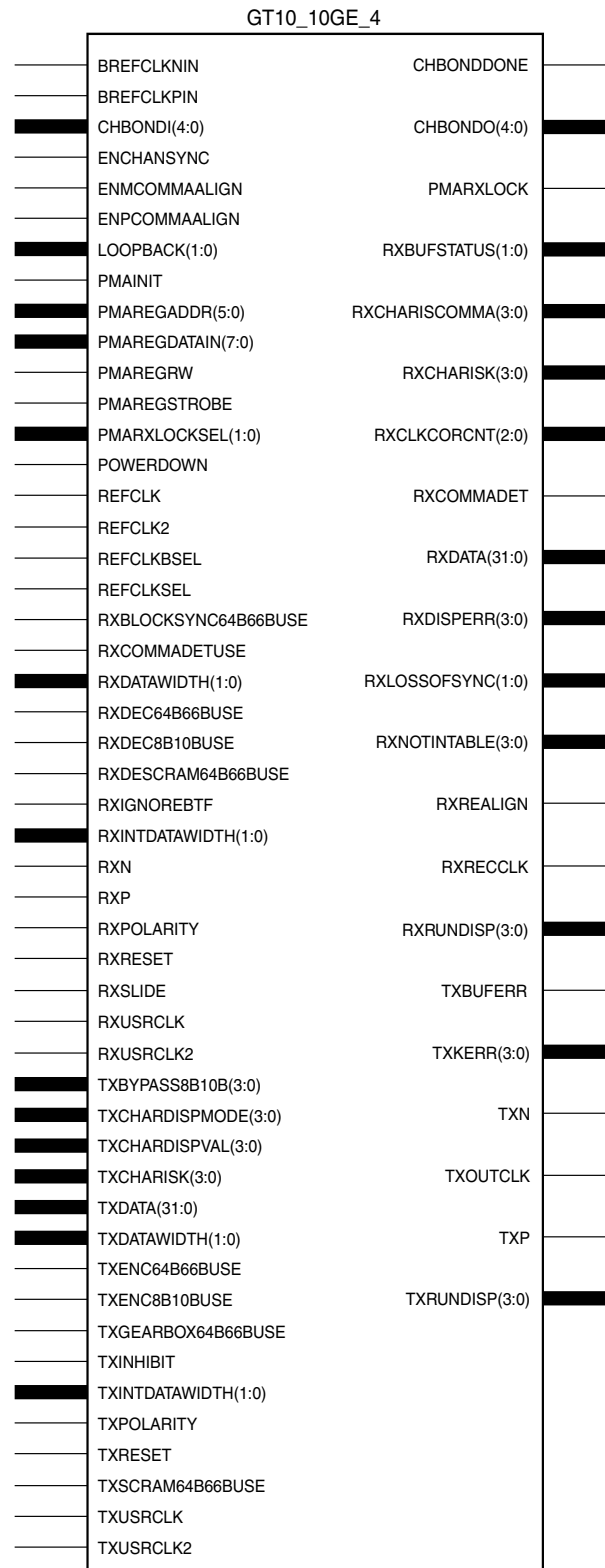
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

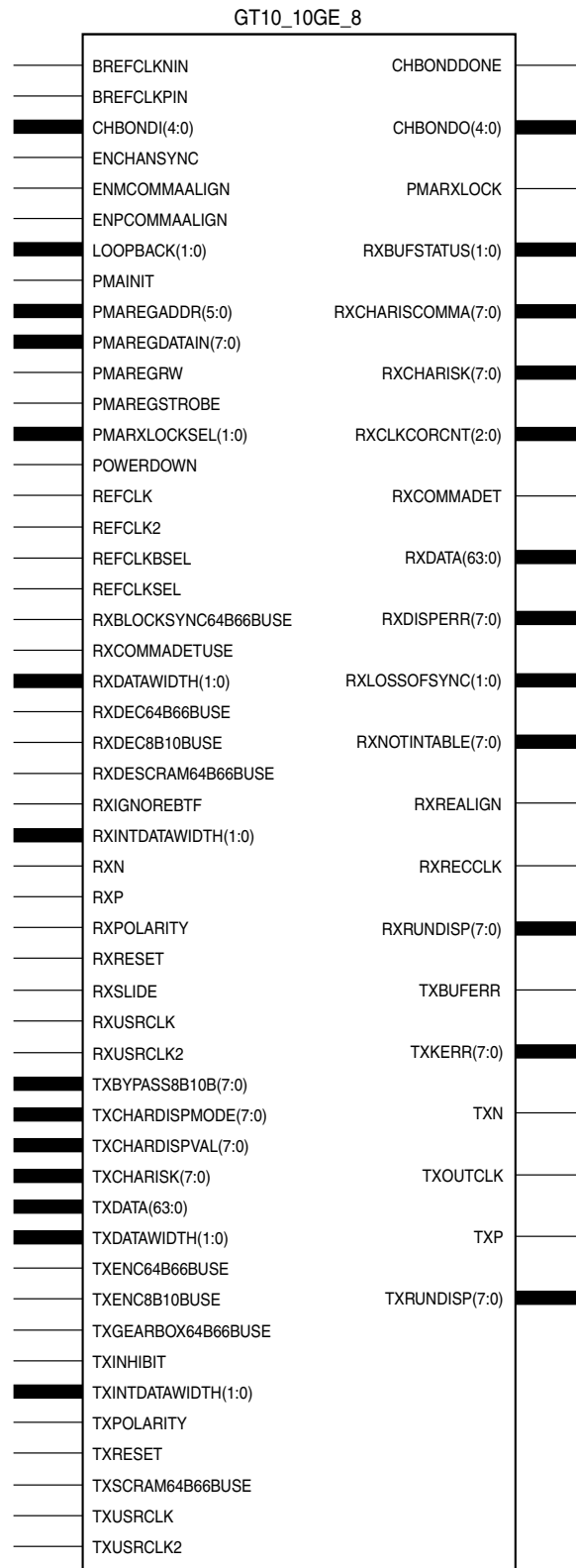
GT10_10GE_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Xilinx protocol 10-gigabit transceiver supports 4 and 8-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 4 or 8.

You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10041



X10042

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_10GFC_n

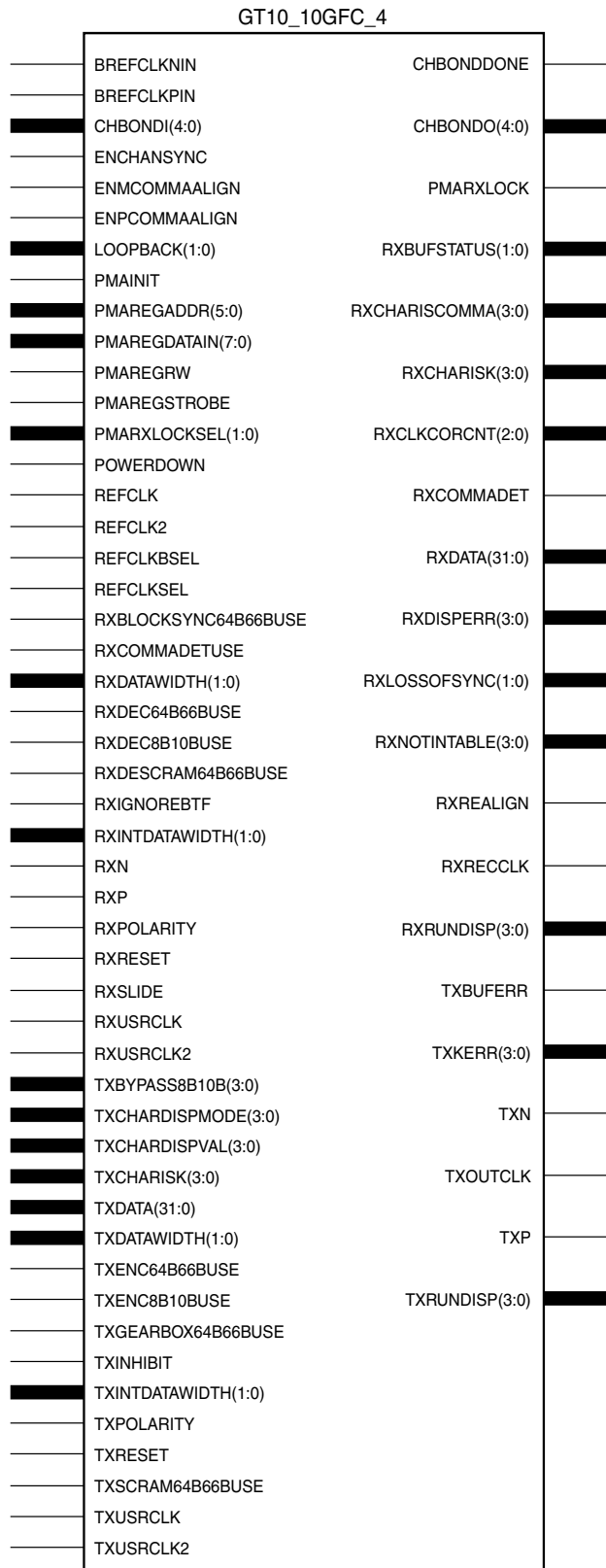
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

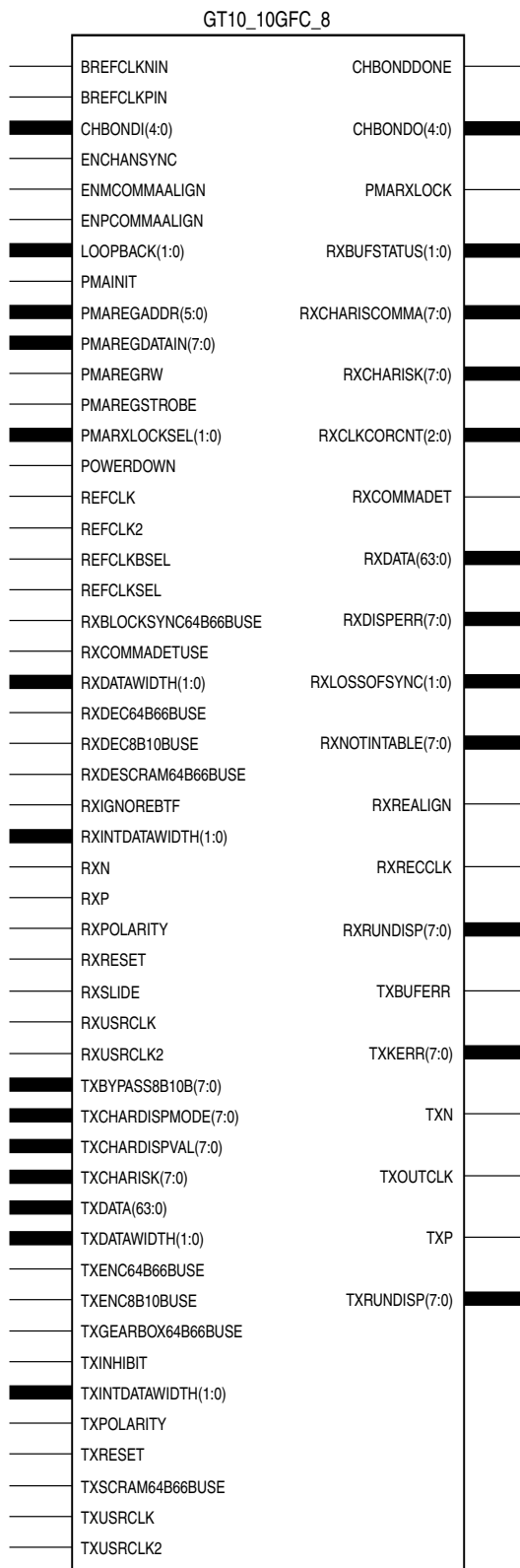
GT10_10GFC_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Xilinx protocol 10-gigabit transceiver supports 4 and 8-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 4 or 8.

You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10043



X10044

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_AURORA_n

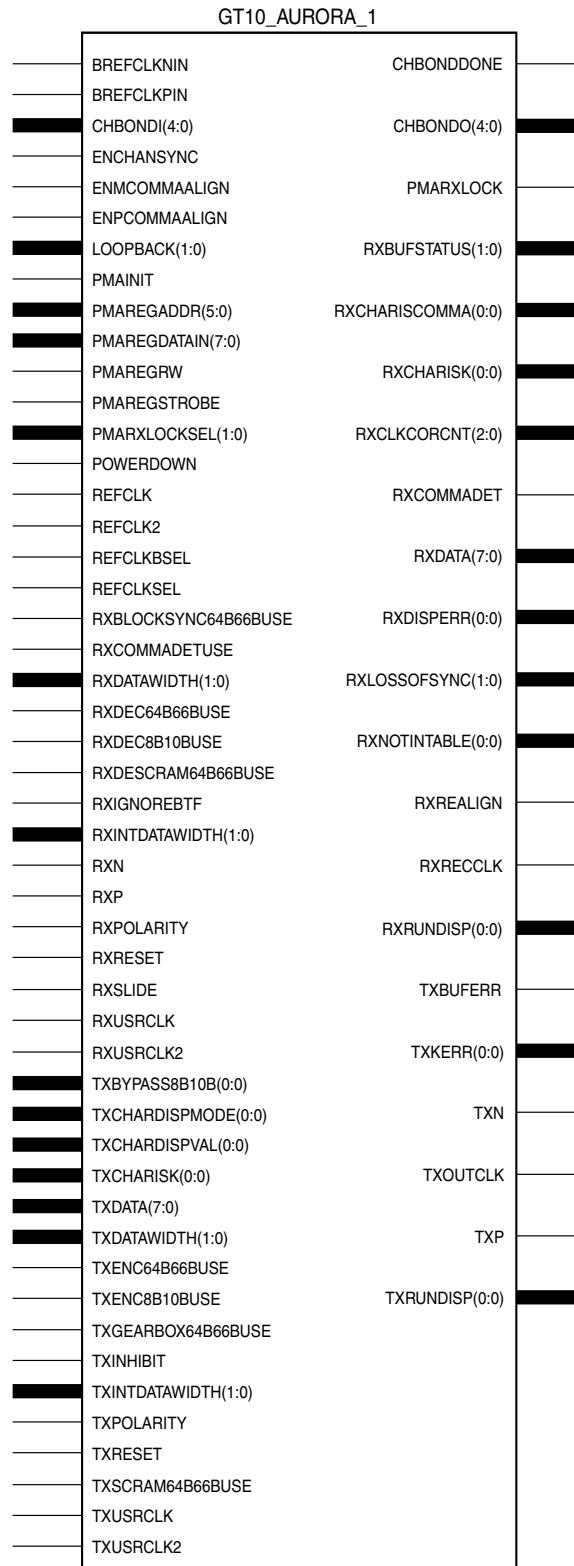
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

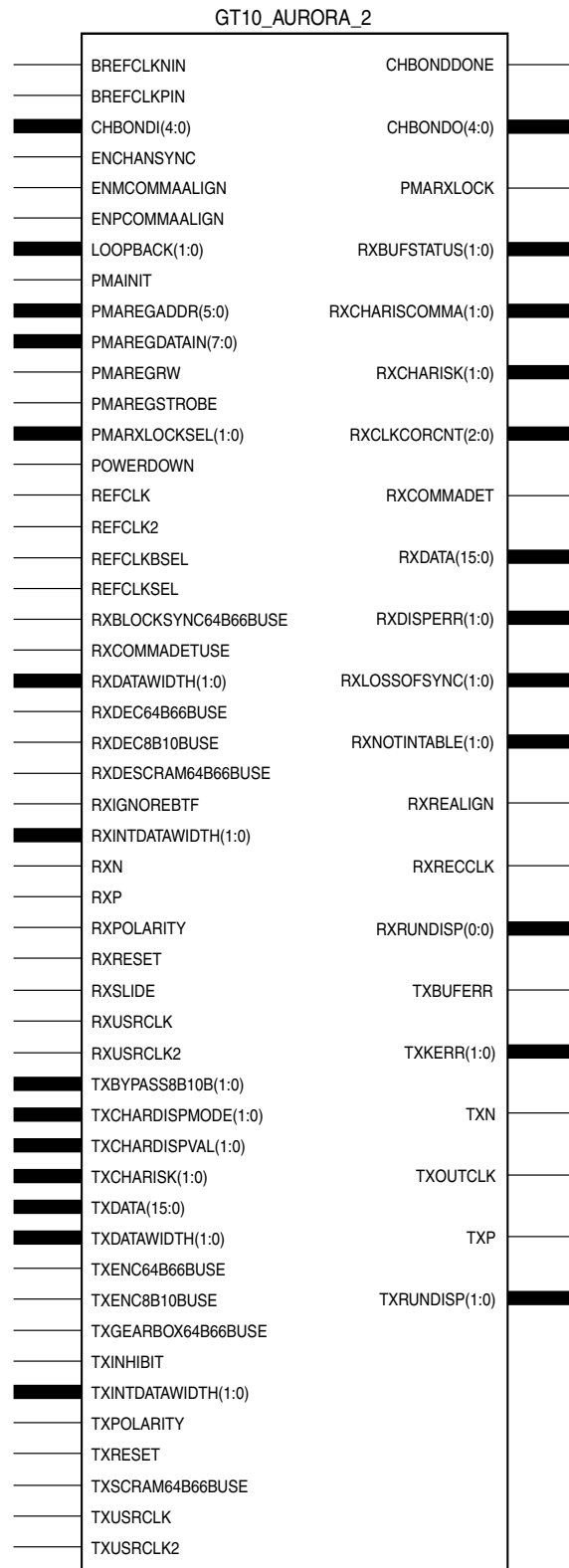
GT10_AURORA_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Xilinx protocol 10-gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2, or 4.

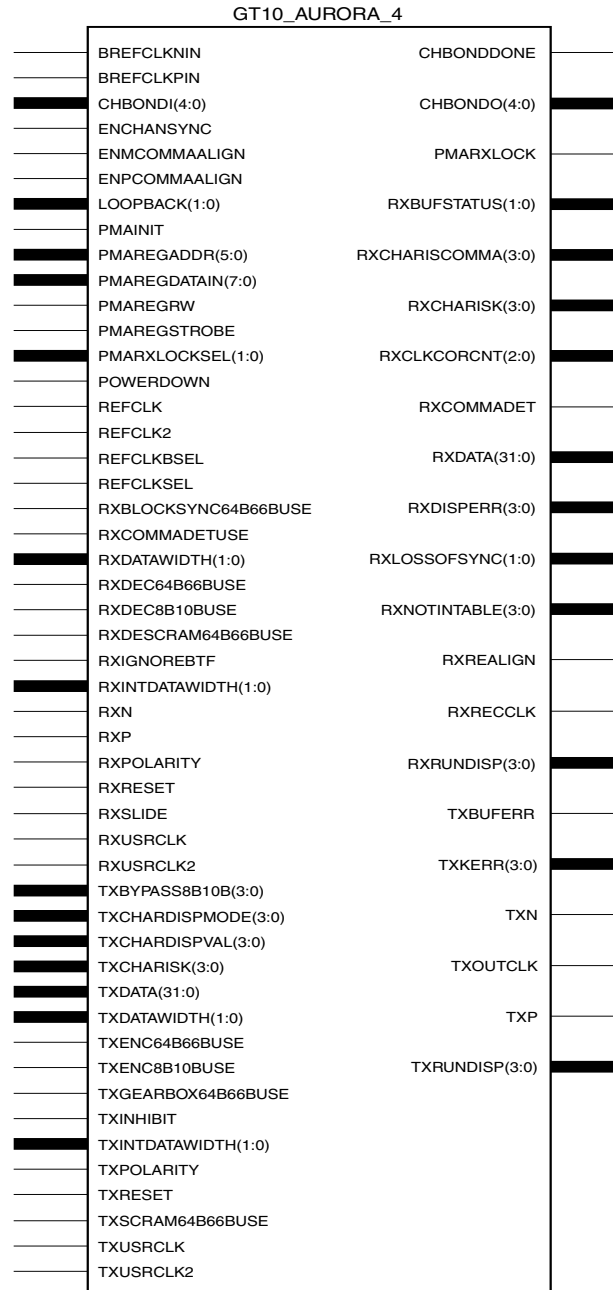
You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10045



X10046



X10047

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_AURORAX_n

10-Gigabit Transceiver for High-Speed I/O

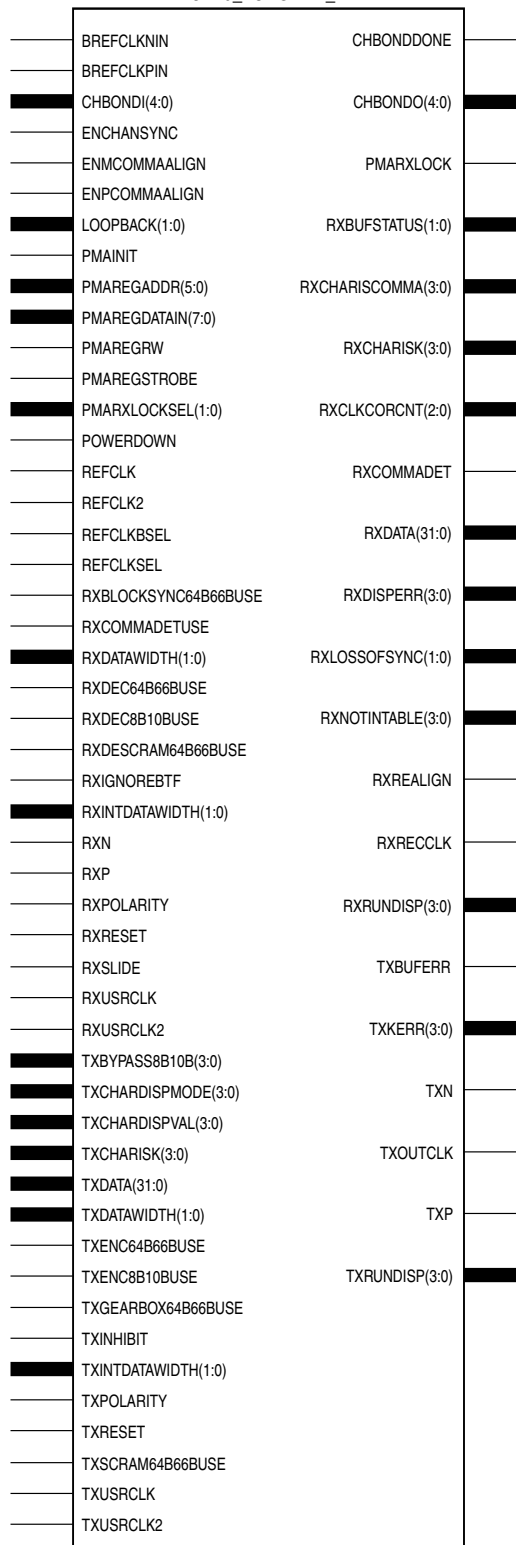
Architectures Supported

GT10_AURORAX_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

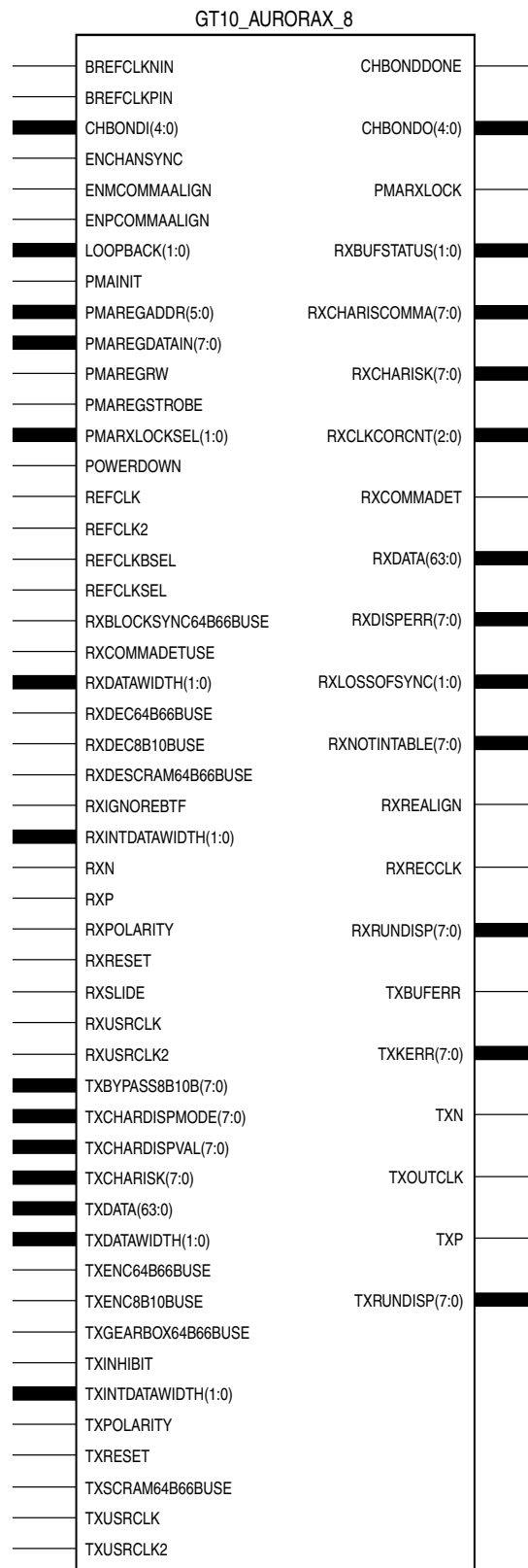
This Xilinx protocol 10-gigabit transceiver supports 4 and 8-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 4 or 8.

You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.

GT10_AURORAX_4



X10048



X10049

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

Use the Architecture Wizard in order to generate and instantiate these components.

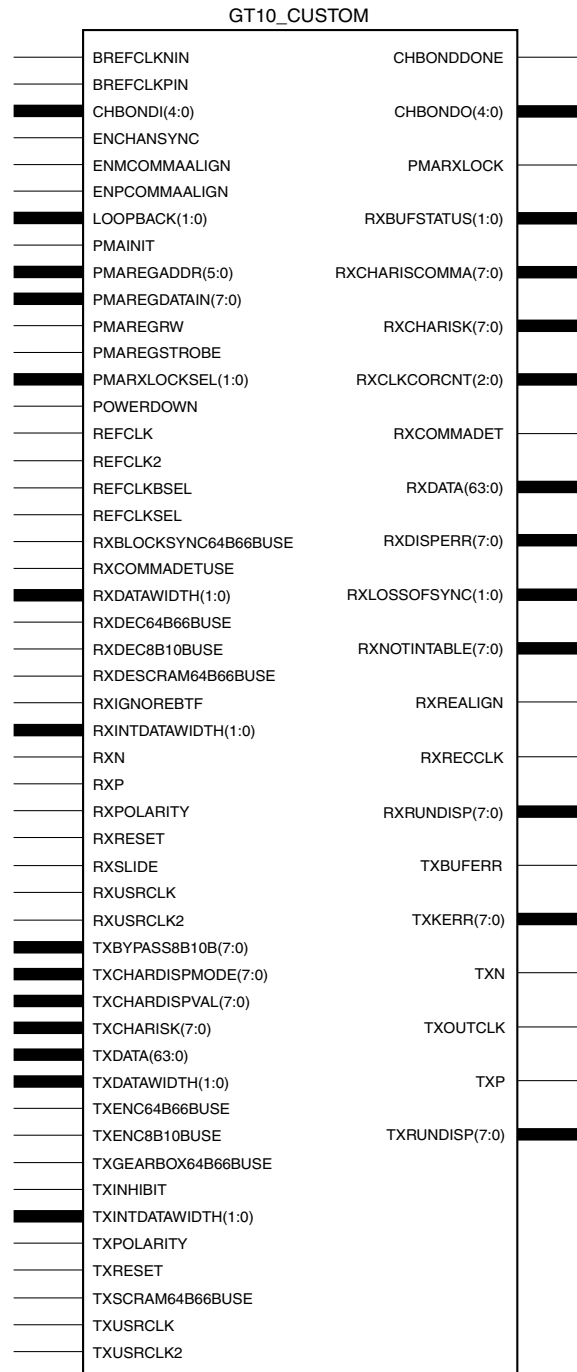
GT10_CUSTOM

10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

GT10_CUSTOM	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This 10-gigabit transceiver is fully customizable. You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of n and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10050

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Template

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_INFINIBAND_n

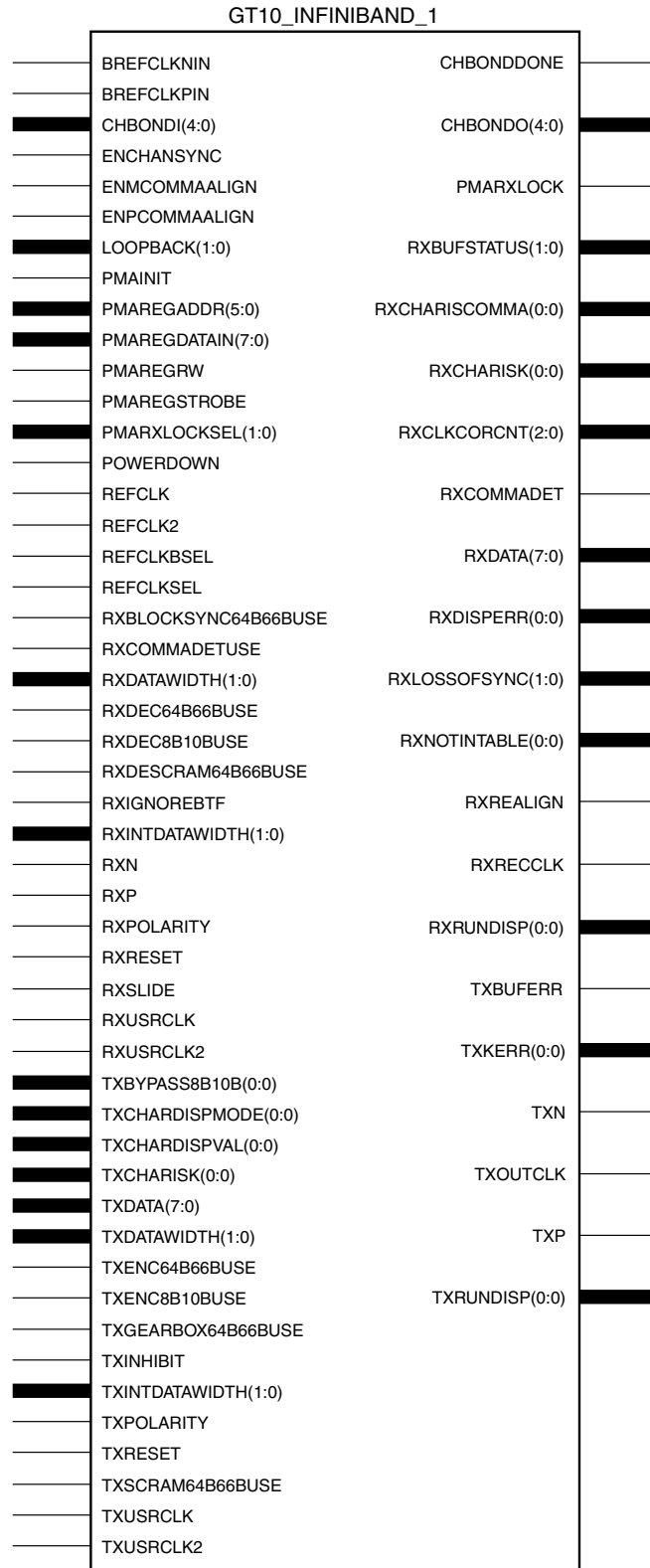
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

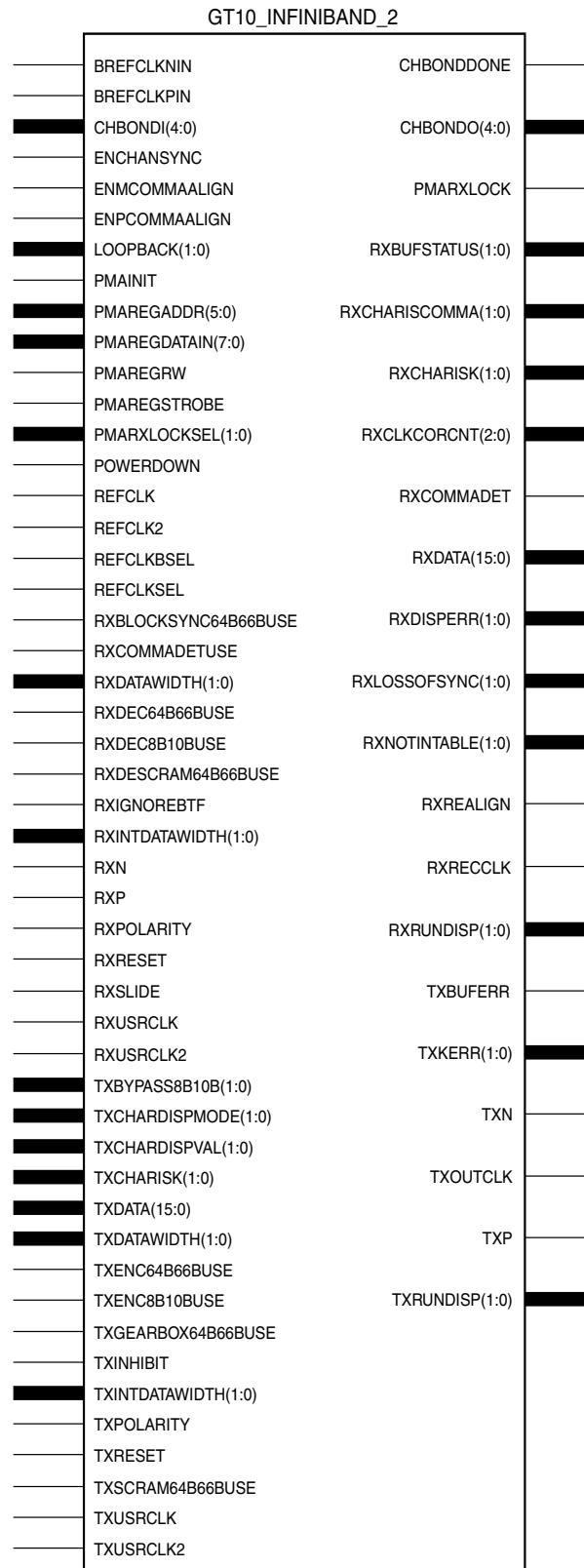
GT10_INFINIBAND_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Infiniband 10-gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

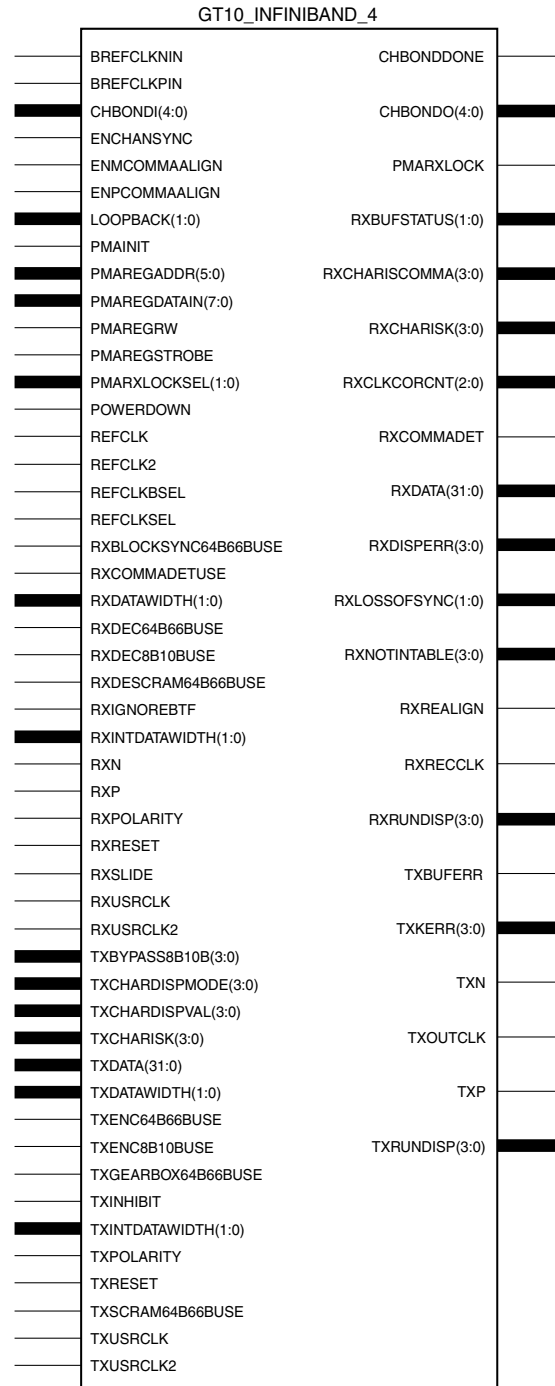
You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10051



X10052



X10053

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_OC48_n

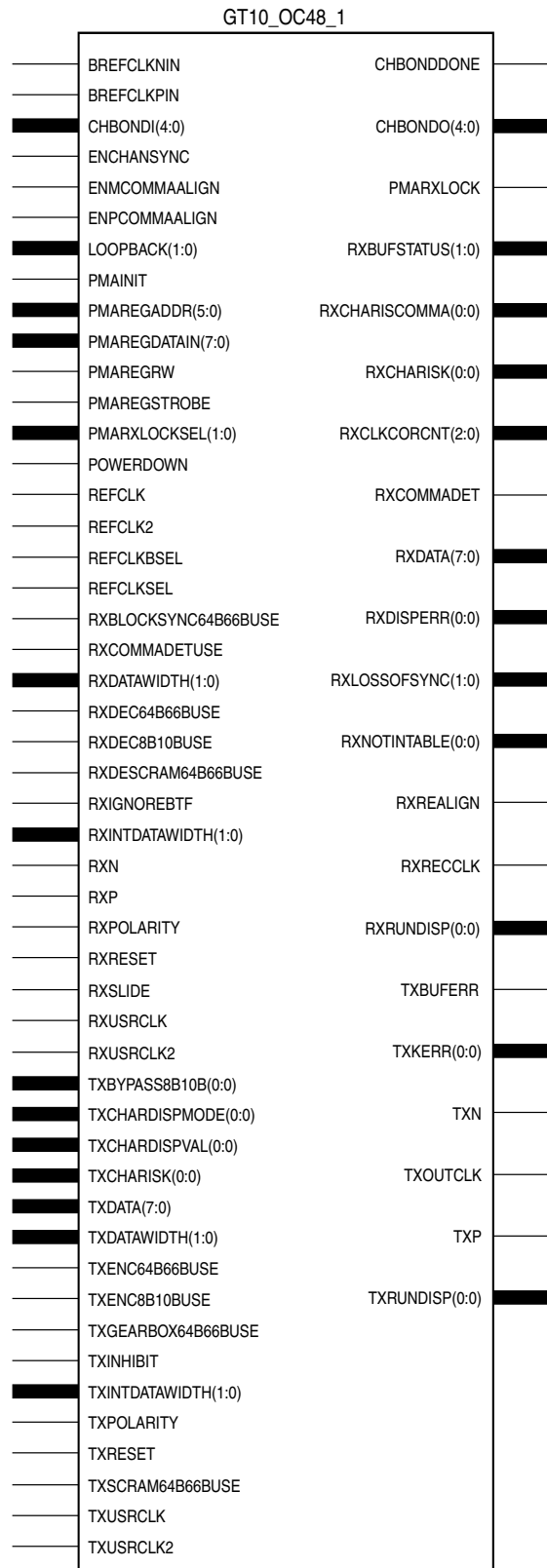
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

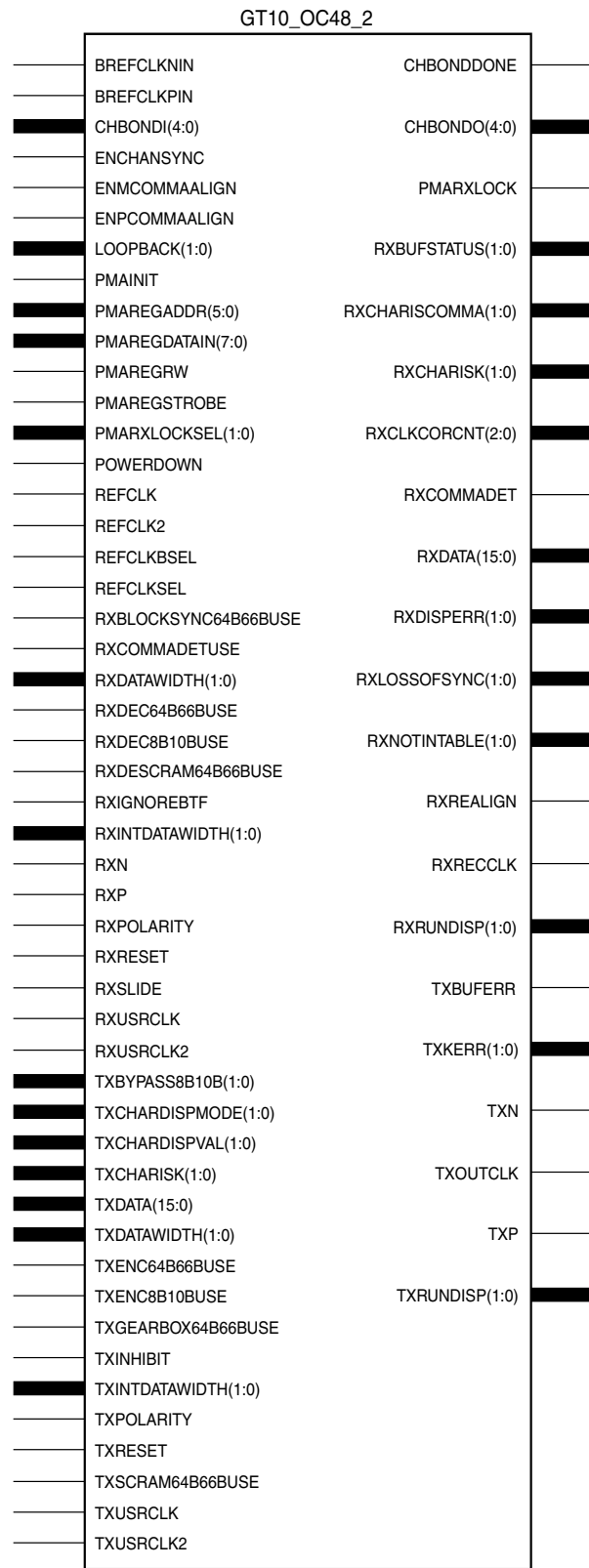
GT10_OC48_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Xilinx protocol 10-gigabit transceiver supports 1, 2 and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2, or 4.

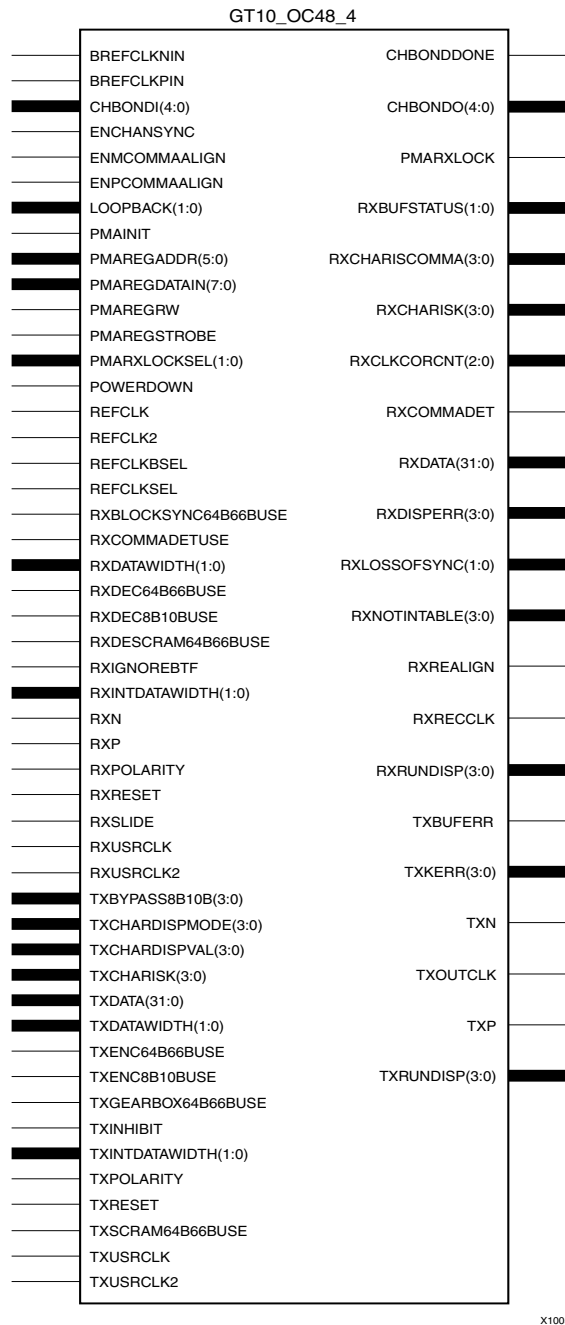
You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO X Transceiver User Guide*.



X10056



X10057



Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_OC192_n

10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

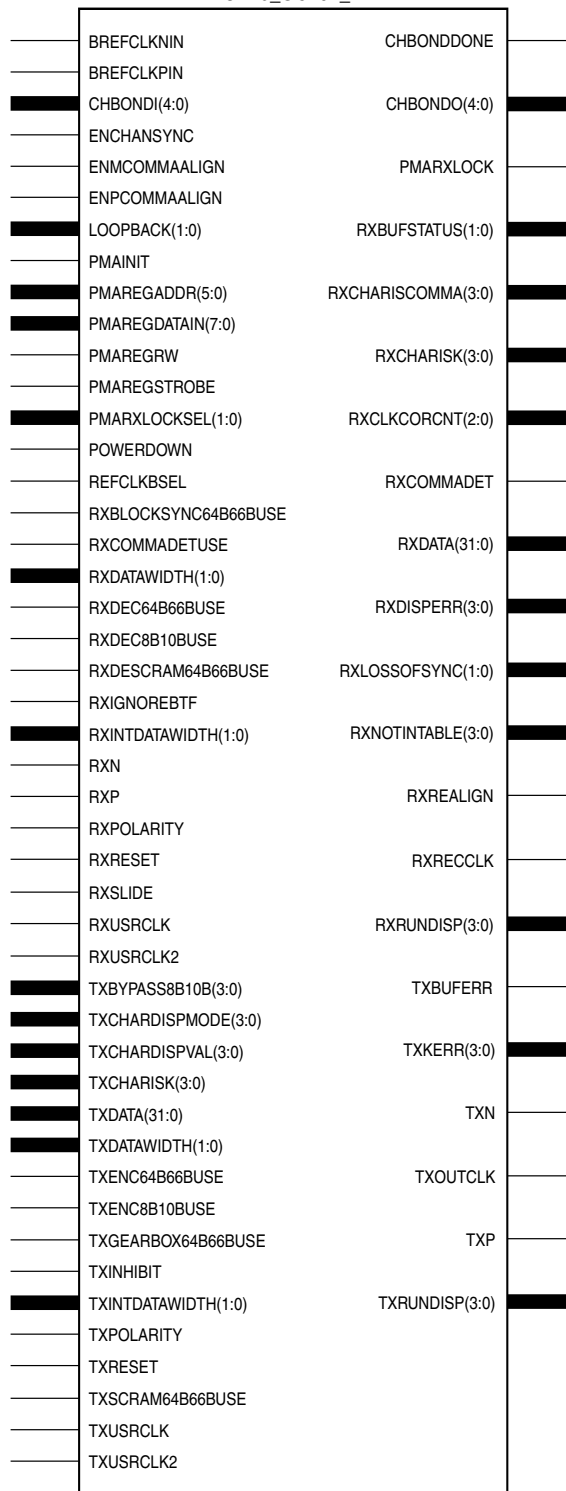
GT10_OC192_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

This Xilinx protocol 10-gigabit transceiver supports 4 and 8-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 4 or 8.

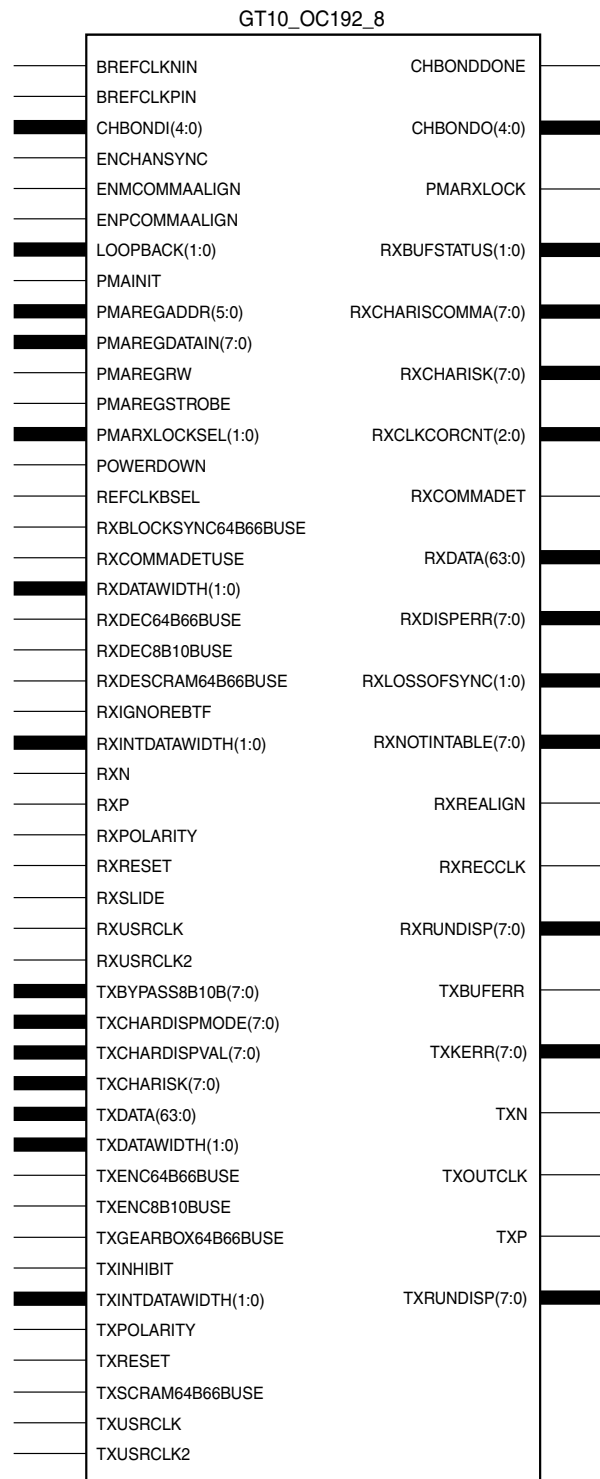
You can also set attributes for the primitives. See the *RocketIO X Transceiver User Guide* for a description of these attributes and their default attribute values.

The following figures list the input and output ports for all values of *n*. For a description of each of the ports, see the *RocketIO X Transceiver User Guide*.

GT10_OC192_4



X10054



X10055

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_PCI_EXPRESS_n

10-Gigabit Transceiver for High-Speed I/O

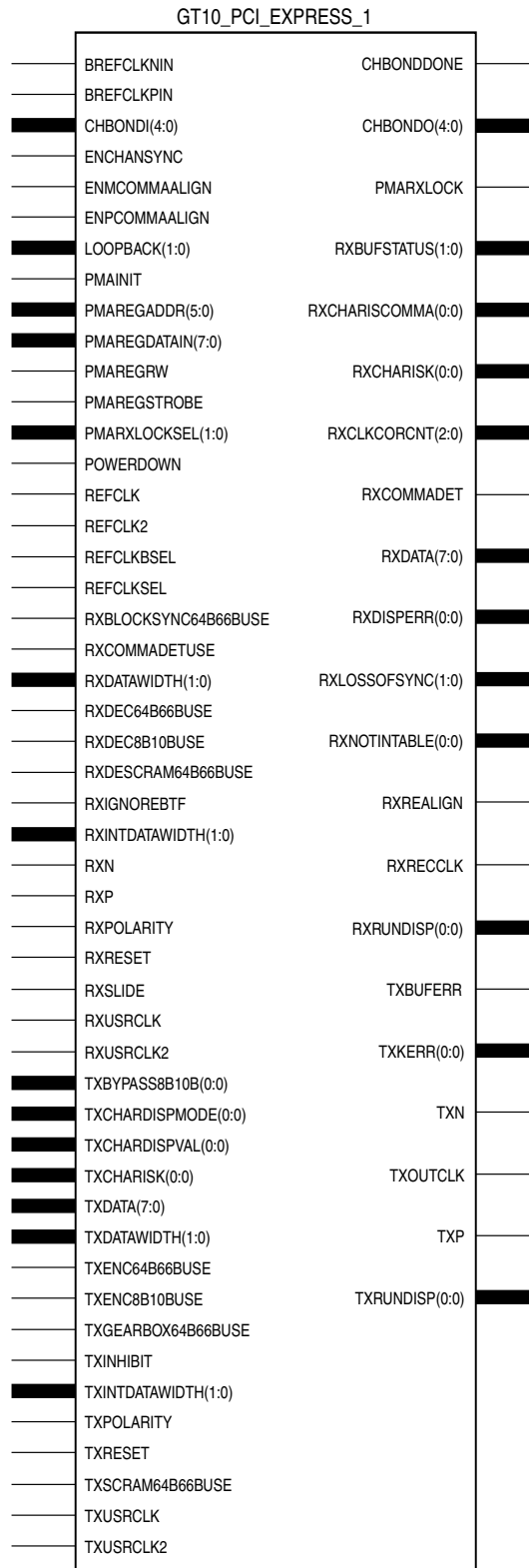
Architectures Supported

GT10_PCI_EXPRESS_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II or Virtex-II Pro.	

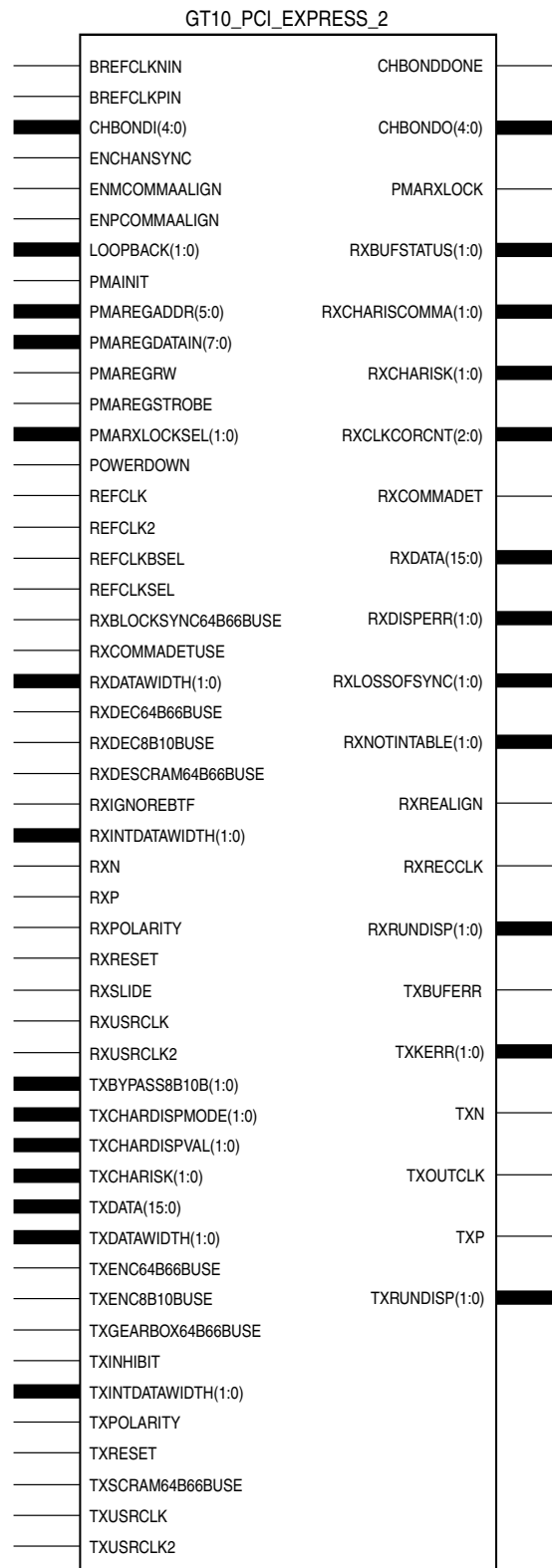
This Xilinx protocol 10-gigabit transceiver supports 1, 2 and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2, or 4.

You can also set attributes for the primitives. See the *RocketIO X Transceiver User Guide* for a description of these attributes and their default attribute values.

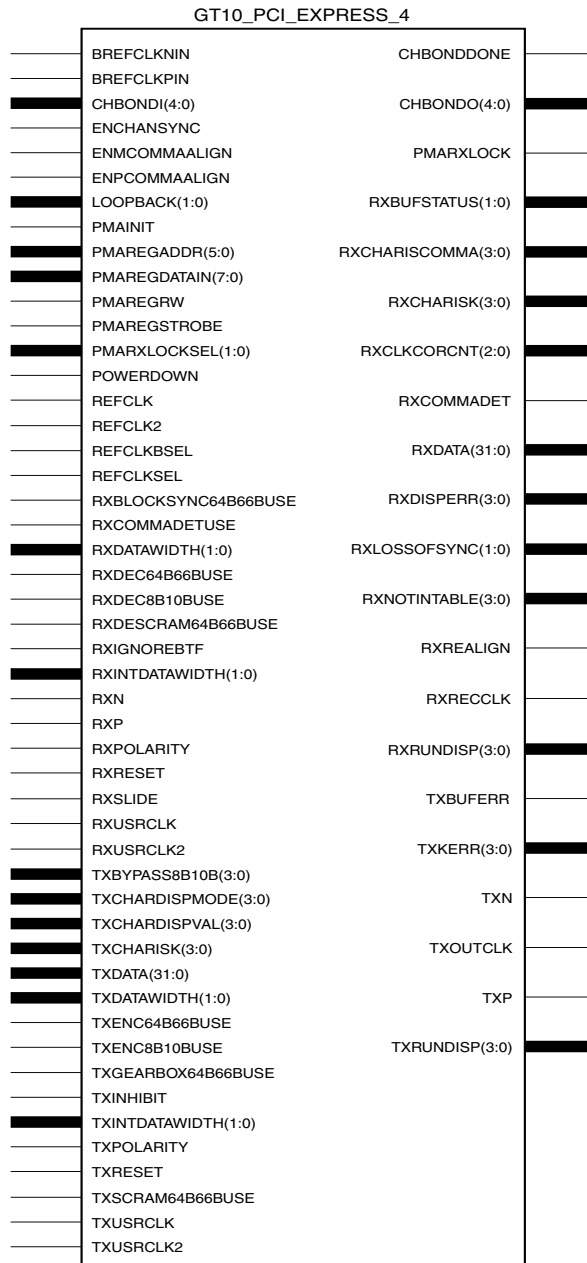
The following figures list the input and output ports for all values of *n*. For a description of each of the ports, see the *RocketIO X Transceiver User Guide*.



X10059



X10060



X10061

Usage

This design element is instantiated rather than inferred in the design code.

VHDL Start

VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

GT10_XAUI_n

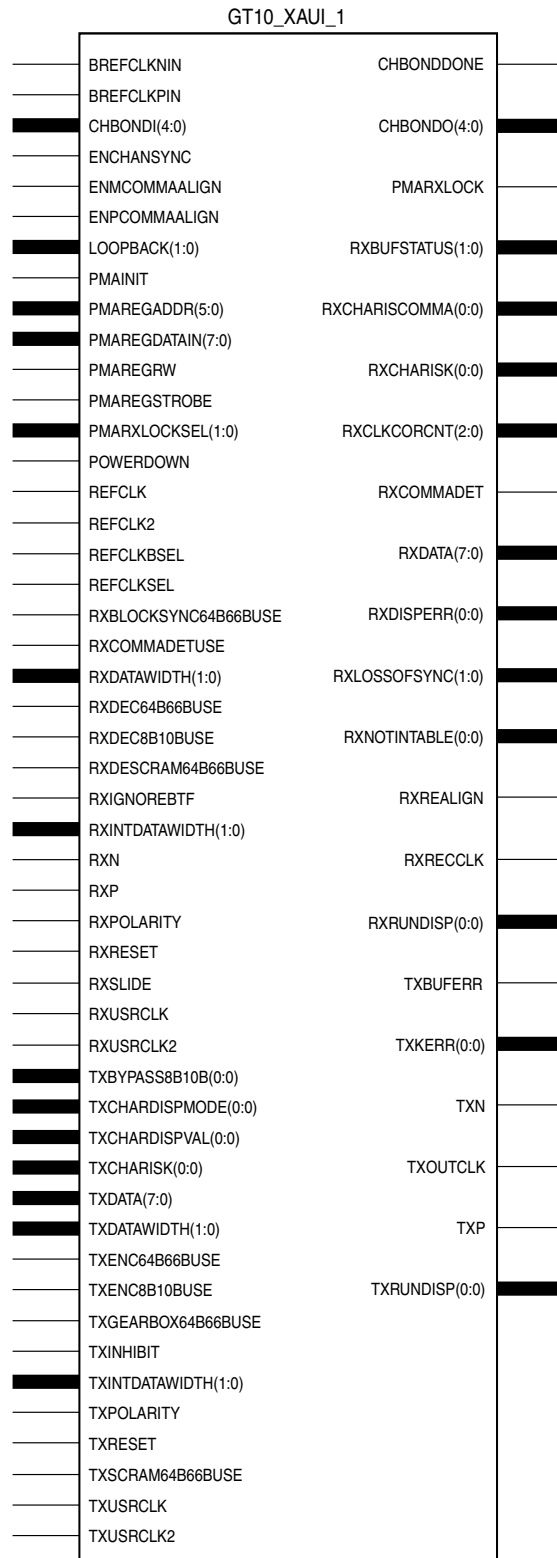
10-Gigabit Transceiver for High-Speed I/O

Architectures Supported

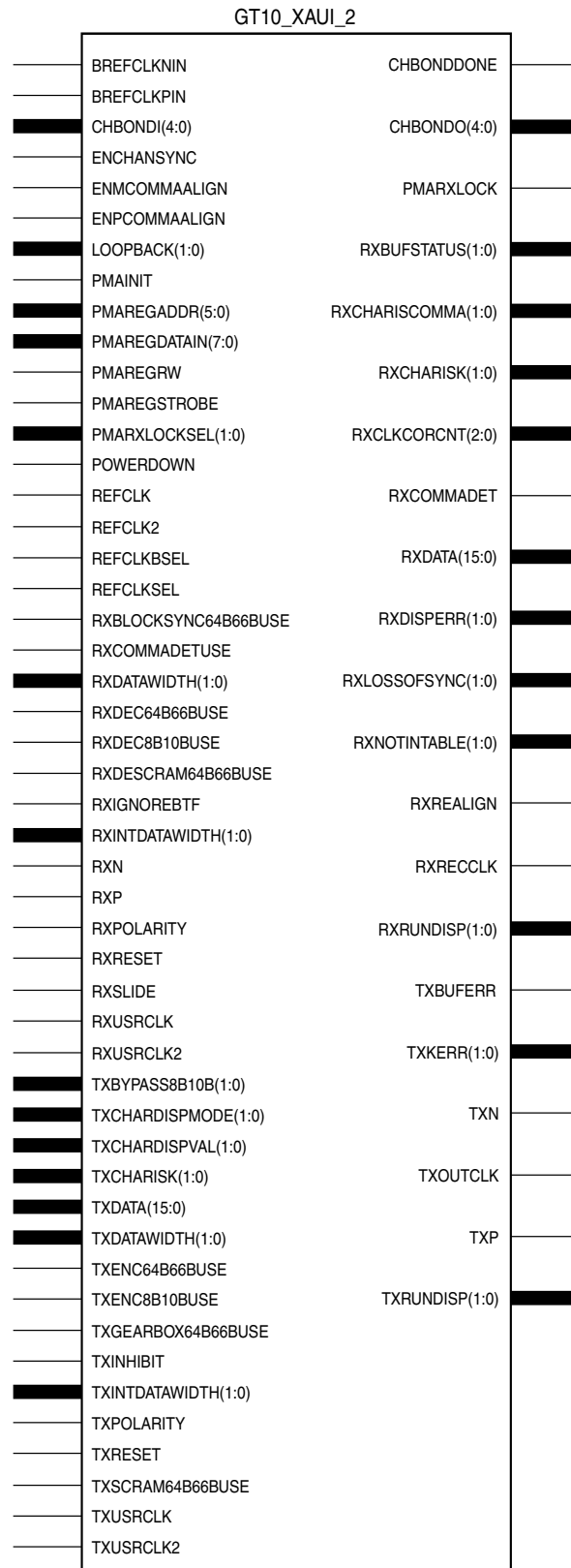
GT10_XAUI_n	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
*Supported for Virtex-II Pro X but not for Virtex-II nor Virtex-II Pro.	

This XAUI 10-gigabit transceiver supports 1, 2, and 4-byte data paths. The letter *n* represents number of bytes of the data path. Valid values are 1, 2 or 4.

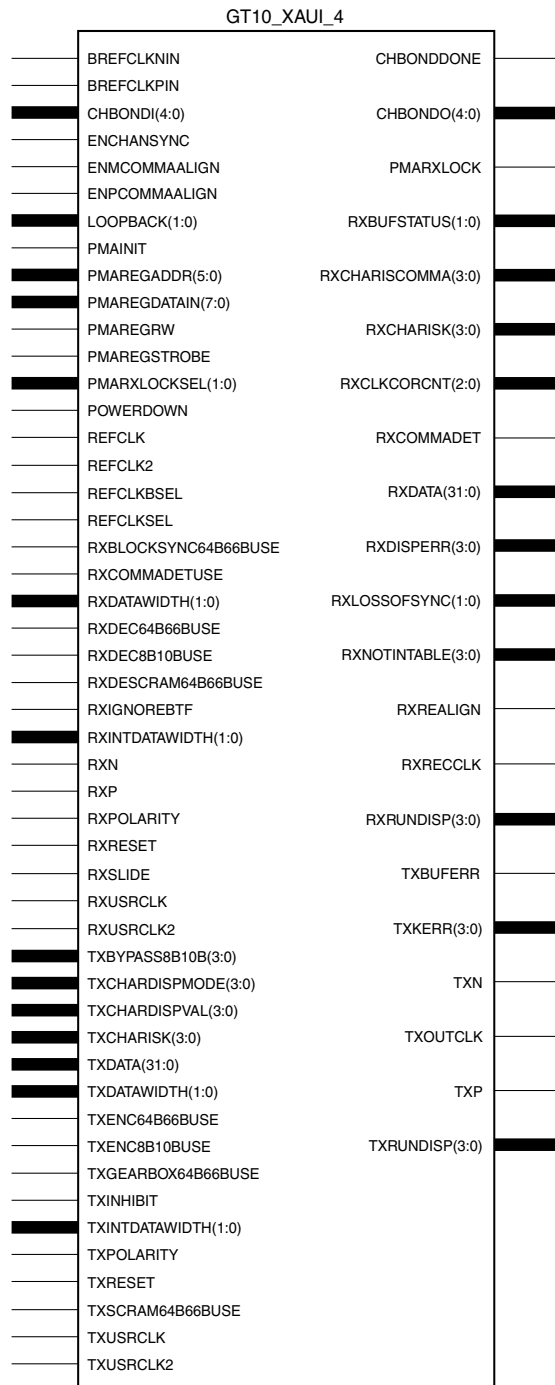
You can also set attributes for the primitives. For a description of these attributes and their default attribute values, or to see a list the input and output ports for all values of *n* and a description of each of the ports, see the *RocketIO Transceiver User Guide*.



X10062



X10063



X10064

Usage

This design element is instantiated rather than inferred in the design code.

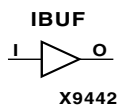
VHDL Instantiation Templates

Use the Architecture Wizard in order to generate and instantiate these components.

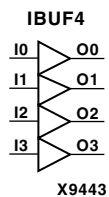
IBUF, 4, 8, 16

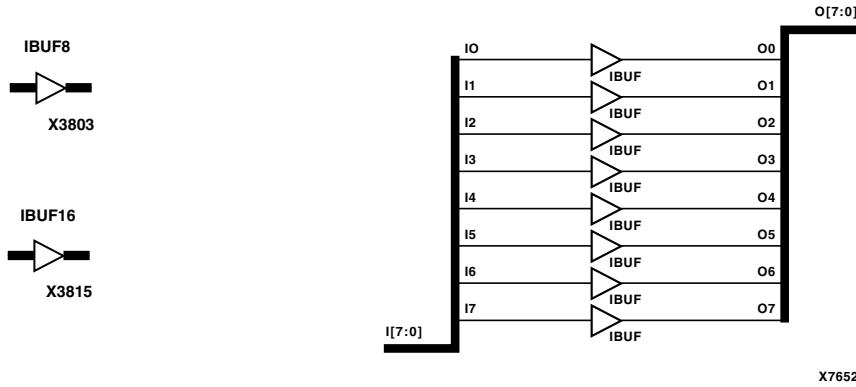
Single- and Multiple-Input Buffers

IBUF	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
IBUF4	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
IBUF8, IBUF16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

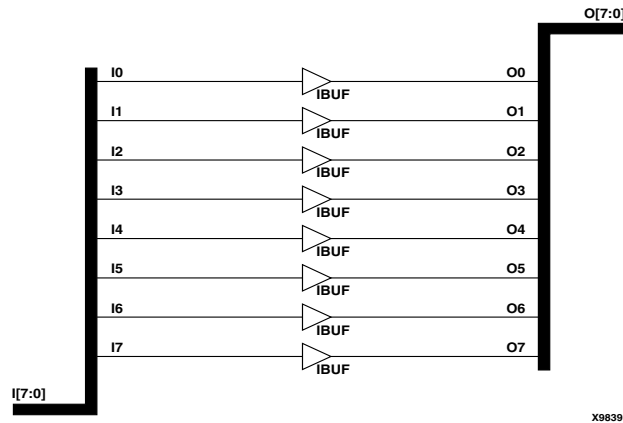


IBUF, IBUF4, IBUF8, and IBUF16 are single- and multiple-input buffers. An IBUF isolates the internal circuit from the signals coming into a chip. IBUFs are contained in input/output blocks (IOBs). IBUF inputs (I) are connected to an IPAD or an IOPAD. IBUF outputs (O) are connected to the internal circuit.





IBUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-II E, Virtex, Virtex-E



IBUF8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage Rules

The Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X architectures include a versatile interface to multiple voltage and drive standards. To select an I/O standard, you must choose the appropriate component from the library or add an IOSTANDARD attribute to the appropriate buffer component. For example, for an input buffer that uses the GTL standard, you would choose the IBUF_GTL component or choose the IBUF component and attach the IOSTANDARD=GTL attribute to it.

See the following sections for information on the various input/output buffer components and attributes available to implement the desired standard:

- "IBUFG"
- "IOBUF"
- "OBUF, 4, 8, 16"
- "OBUFT, 4, 8, 16"

The hardware implementation of the various I/O standards requires that certain usage rules be followed. Each Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X device has eight banks (two on each edge). Each

bank has voltage sources shared by all I/O in the bank. Therefore, in a particular bank, the voltage source (for either input or output) must be of the same type.

- For Spartan-II, Spartan-IIE, Virtex, and Virtex-E, see “[Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules](#)” below. Virtex-E follows the same banking rules as Virtex with a few additions.
- See “[Additional Banking Rules for Virtex-E and Spartan-IIE](#)” below for the additional Virtex-E rules. Virtex-II, Virtex-II Pro, and Virtex-II Pro X have their own set of banking rules.
- See “[Virtex-II, Virtex-II Pro, Virtex-II Pro X Banking Rules](#)” below for Virtex-II, Virtex-II Pro, and Virtex-II Pro X rules.

Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules

The hardware implementation of the various I/O standards requires that certain usage rules be followed. Each Spartan-II, Spartan-IIE, Virtex, and Virtex-E device has eight banks (two on each edge). Each bank has voltage sources shared by all I/O in the bank. Therefore, in a particular bank, the voltage source (for either input or output) must be of the same type. The Input Banking (VREF) Rules section and the Output Banking (VCCO) Rules section below summarize the component usage rules based on the hardware implementation

Input Banking (VREF) Rules

The low-voltage I/O standards that have a differential amplifier input require a voltage reference input (VREF). The VREF voltage source is provided as an external signal to the chip.

- Any input buffer component that does not require a VREF source (LVTTTL, LVCMOS2, PCI) can be placed in any bank.
- All input buffer components that require a VREF source (GTL*, HSTL*, SSTL*, CTT, AGP) must be of the same I/O standard in a particular bank. For example, IBUF with I/O standard (SSTL2_I) and IBUFG with I/O standard (SSTL2_I) are compatible since they are the same I/O standard.
- If the bank contains any input buffer component that requires a VREF source, the following conditions apply.
 - ◆ One or more VREF sources must be connected to the bank via an IOB.
 - ◆ The number of VREF sources is dependent on the device and package.
 - ◆ The locations of the VREF sources are fixed for each device/package.
 - ◆ All VREF sources must be used in that bank.
- If the bank contains no input buffer component that requires a VREF source, the IOBs for VREF sources can be used for general I/O.
- Output buffer components of any type can be placed in the bank.

Output Banking (VCCO) Rules

Because Virtex, Virtex-E, Spartan-II, and Spartan-IIE have multiple low-voltage standards, some control is required over the distribution of VCCO, the drive source voltage for output pins. To provide for maximum flexibility, the output pins are

banked. In comparison to the VREF sources described above, the VCCO voltage sources are dedicated pins on the device and do not consume valuable IOBs.

- Any output buffer component that does not require a VCCO source (GTL, GTL+) can be placed in any bank.
- To be placed in a particular bank, all output buffer components that require VCCO must have the same supply voltage (VCCO). For example, OBUF with I/O standard (SSTL3_I) and OBUF with I/O standard (PCI33_3) are compatible in the same output bank since VCCO=3.3 for both.
- Input buffer components of any type can be placed in the bank.
- The configuration pins on a Virtex, Virtex-E, Spartan-II, and Spartan-IIE device are on the right side of the chip. When configuring the device through a serial ProM, the user is required to use a VCCO of 3.3V in the two banks on the right hand side of the chip. If the user is not configuring the device through a serial ProM, the VCCO requirement is dependent upon the configuration source.

Banking Rules for KEEPER

If a KEEPER symbol is attached to a component (3-state output buffer) for an I/O standard that requires a VREF (for example, OBUFT with I/O standard (GTL), OBUFT with I/O standard (SSTL3_I)), the KEEPER element requires that the VREF be properly driven.

Additional Banking Rules for Virtex-E and Spartan-IIE

The Virtex-E and Spartan-IIE architectures requires the same banking rules as described in the “Virtex, Virtex-E, Spartan-II, and Spartan-IIE Banking Rules” section.

Additional I/O standards are supported as indicated in the following table.

Virtex-II, Virtex-II Pro, Virtex-II Pro X Banking Rules

The hardware implementation of the various I/O standards requires that certain usage rules be followed. Virtex-II, Virtex-II Pro, and Virtex-II Pro X devices have eight banks (two on each edge), numbered from 0 through 7. Each bank has voltage sources shared by all I/O in the bank.

The following table summarizes the values that you need to check for compatibility for each combination of I/O buffer programming (input, output, or bidirectional buffer). For example, the table shows that if you configure an output buffer as LVCMOS25, which has an output voltage of 2.5V, and an input buffer as LVCMOS15, which as an input voltage of 1.5V, the Out/In Voltage is checked. Because they have different voltages, this combination would not be allowed in a particular I/O bank.

IOB Programming Combinations		VREF	Output VCCO	Input VCCO	Out/In Voltage
Input	Input	Check		Check	
Input	Output				Check
Input	Bidirectional	Check		Check	Check
Output	Input				Check
Output	Output		Check		
Output	Bidirectional		Check		Check
Bidirectional	Input	Check		Check	Check

IOB Programming Combinations		VREF	Output VCCO	Input VCCO	Out/In Voltage
Bidirectional	Output		Check		Check
Bidirectional	Bidirectional	Check	Check	Check	Check

Usage

IBUFs are typically inferred for all top level input ports, but they can also be instantiated if necessary.

VHDL Instantiation Template

```
-- Component Declaration for IBUF should be placed
-- after architecture statement but before begin keyword

component IBUF
  -- synthesis translate_off
  generic (
    IOSTANDARD: bit_vector := "LVTTTL");
  -- synthesis translate_on
  port (O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for IBUF
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here
attribute IOSTANDARD : string;
attribute IOSTANDARD of IBUF_instance_name : label is "LVTTTL";

-- Component Instantiation for IBUF should be placed
-- in architecture after the begin keyword

IBUF_INSTANCE_NAME : IBUF
  -- synthesis translate_off
  generic map (
    IOSTANDARD => "string_value")
  -- synthesis translate_on
  port map (O => user_O,
            I => user_I);
```

Verilog Instantiation Template

```
IBUF instance_name (.O (user_O),
                   .I (user_I));
defparam IBUF_instance_name.IOSTANDARD = "string_value";
```


IBUFDS

Differential Signaling Input Buffer with Selectable I/O Interface

Architectures Supported

IBUFDS	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9255

IBUFDS is an input buffer that supports low-voltage, differential signaling. In IBUFDS, a design level interface signal is represented as two distinct ports (I and IB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs		Outputs
I	IB	O
0	0	- *
0	1	0
1	0	1
1	1	- *

* The dash (-) means No Change.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- IBUFDS: Differential Input Buffer
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

IBUFDS_inst : IBUFDS
-- Edit the following generic to specify the I/O standard for this
port.
generic map (
    IOSTANDARD => "LVDS_25")
port map (
    O => O, -- Clock buffer output
    I => I, -- Diff_p clock buffer input (connect to top-level port)
    IB => IB -- Diff_n clock buffer input (connect directly to top-
level port)
```

```
);  
-- End of IBUFDS_inst instantiation
```

Verilog Instantiation Template

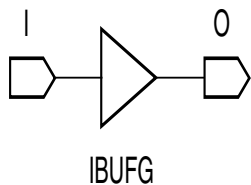
```
// IBUFDS: Differential Input Buffer  
//      Virtex-II/II-Pro, Spartan-3  
// The current version of the Xilinx HDL Libraries Guide  
  
IBUFDS IBUFDS_inst (  
    .O(O), // Clock buffer output  
    .I(I), // Diff_p clock buffer input (connect directly to  
           // top-level port)  
    .IB(IB) // Diff_n clock buffer input (connect directly to  
           // top-level port)  
);  
  
// Edit the following defparam to specify the I/O standard for this  
// port. If the instance name is change, that change needs to be  
// reflecting the this defparam.  
  
    defparam IBUFDS_inst.IOSTANDARD = "LVDS_25";  
  
// End of IBUFDS_inst instantiation
```

IBUFG

Dedicated Input Buffer with Selectable I/O Interface

Architectures Supported

IBUFG	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X10181

IBUFG is dedicated to dedicated input buffers for connecting to the clock buffer BUFG or CLKDLL. You can attach an IOSTANDARD attribute to an IBUFG instance.

The Xilinx implementation software converts each BUFG to an appropriate type of global buffer for the target PLD device. The IBUFG input can only be driven by the global clock pins. The IBUFG output can drive CLKIN of a DLL/DCM, BUFG, or user logic. IBUFG can be routed to user logic and does not have to be routed to a DLL. The IBUFG can only be driven by an IPAD.

Usage

This design element is supported for schematic and instantiation. Synthesis tools usually infer a BUFGP on any clock net. If there are more clock nets than BUFGPs, the synthesis tool usually instantiates BUFGPs for the clocks that are most utilized. The BUFGP contains both a BUFG and an IBUFG.

VHDL Instantiation Template

```
-- IBUFG: Single-ended global clock input buffer
--           All FPGA
-- The current version of the Xilinx HDL Libraries Guide

IBUFG_inst : IBUFG
-- Edit the following generic to specify the I/O standard for this
port.
generic map (
    IOSTANDARD => "LVCMOS25")
port map (
    O => O, -- Clock buffer output
    I => I -- Clock buffer input (connect directly to top-level
port)
);

-- End of IBUFG_inst instantiation
```

Verilog Instantiation Template

```
// IBUFG: Single-ended global clock input buffer
```

```
//          All FPGA
// The current version of the Xilinx HDL Libraries Guide

IBUFG IBUFG_inst (
    .O(O), // Clock buffer output
    .I(I)  // Clock buffer input (connect directly to
           // top-level port)
);

// Edit the following defparam to specify the I/O standard for this
// port. If the instance name is change, that change needs to be
// reflecting the this defparam.

defparam IBUFG_inst.IOSTANDARD = "LVCMOS25";

// End of IBUFG_inst instantiation
```


IBUFGDS

Dedicated Differential Signaling Input Buffer with Selectable I/O Interface

Architectures Supported

IBUFGDS	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9255

IBUFGDS is a dedicated differential signaling input buffer for connection to the clock buffer (BUFG) or DCM. In IBUFGDS, a design level interface signal is represented as two distinct ports (I and IB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs		Outputs
I	IB	O
0	0	- *
0	1	0
1	0	1
1	1	- *

* The dash (-) means No Change.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- IBUFGDS: Differential Global Clock Input Buffer
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

IBUFGDS_inst : IBUFGDS
-- Edit the following generic to specify the I/O standard for this
port.
generic map (
  IOSTANDARD => "LVDS_25")
port map (
  O => O,  -- Clock buffer output
  I => I,  -- Diff_p clock buffer input (connect to top-level port)
```

```
        IB => IB -- Diff_n clock buffer input (connect to top-level port)
    );

-- End of IBUFGDS_inst instantiation
```

Verilog Instantiation Template

```
// IBUFGDS: Differential Global Clock Input Buffer
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

IBUFGDS IBUFGDS_inst (
    .O(O), // Clock buffer output
    .I(I), // Diff_p clock buffer input (connect directly to
           // top-level port)
    .IB(IB) // Diff_n clock buffer input (connect directly to
           // top-level port)
);

// Edit the following defparam to specify the I/O standard for this
// port. If the instance name is change, that change needs to be
// reflecting the this defparam.

defparam IBUFGDS_inst.IOSTANDARD = "LVDS_25";

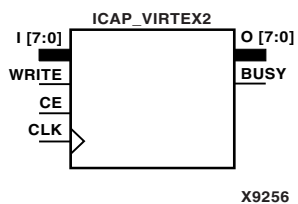
// End of IBUFGDS_inst instantiation
```

ICAP_VIRTEX2

User Interface to Virtex-II, Virtex-II Pro, and Virtex-II Pro X Internal Configuration Access Port

Architectures Supported

ICAP_VIRTEX, ICAP_VIRTEX2	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ICAP_VIRTEX2 provides user access to the Virtex-II, Virtex-II Pro, and Virtex-II Pro X internal configuration access port (ICAP).

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- ICAP_VIRTEX2: Internal Configuration Access Port
--                               Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

ICAP_VIRTEX2_inst : ICAP_VIRTEX2
port map (
    BUSY => BUSY,    -- Busy output
    O => O,          -- 8-bit data output
    CE => CE,        -- Clock enable input
    CLK => CLK,      -- Clock input
    I => I,          -- 8-bit data input
    WRITE => WRITE  -- Write input
);
```

Verilog Instantiation Template

```
// ICAP_VIRTEX2: Internal Configuration Access Port
//                               Virtex-II/II-Pro
// The current version of the Xilinx HDL Libraries Guide

ICAP_VIRTEX2 ICAP_VIRTEX2_inst (
    .BUSY(BUSY),    // Busy output
    .O(O),         // 8-bit data output
    .CE(CE),       // Clock enable input
    .CLK(CLK),     // Clock input
    .I(I),         // 8-bit data input
);
```

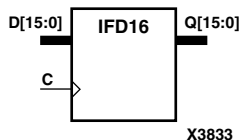
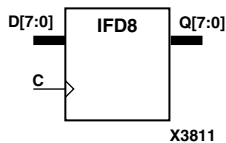
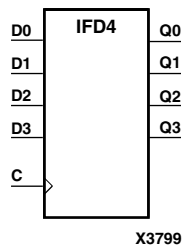
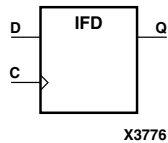
```
        .WRITE(WRITE) // Write input  
    );
```

IFD, 4, 8, 16

Single- and Multiple-Input D Flip-Flops

Architectures Supported

IFD, IFD4, IFD8, IFD16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The IFD D-type flip-flop is contained in an input/output block (IOB), except for XC9500/XV/XL, CoolRunner XPLA3. The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

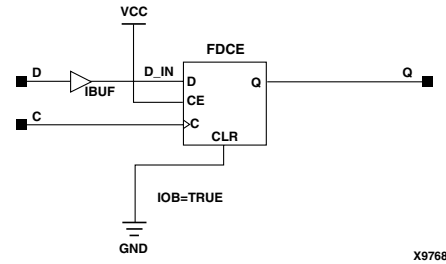
The flip-flops are asynchronously cleared with Low outputs when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

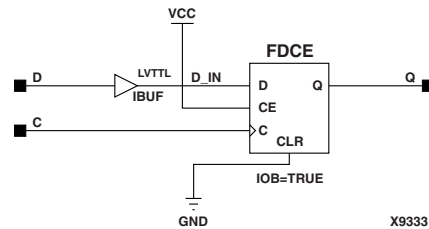
Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

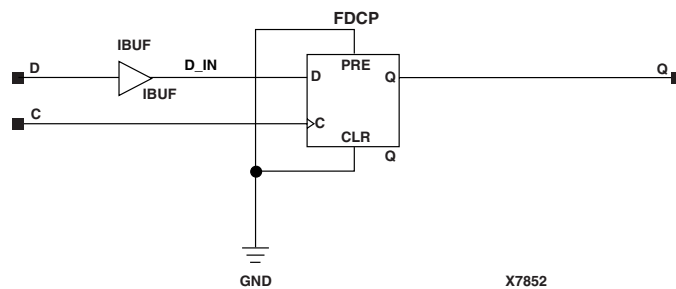
Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



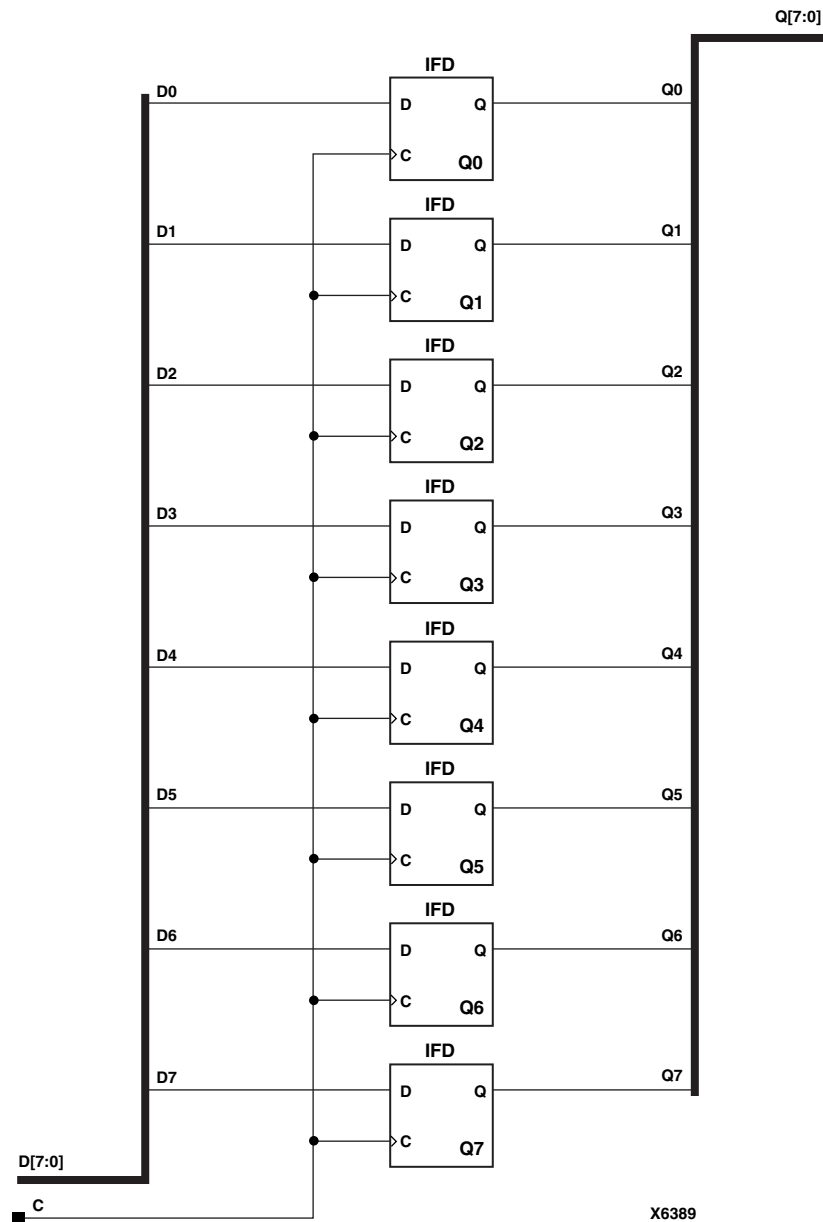
IFD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFD Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



IFD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



IFD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

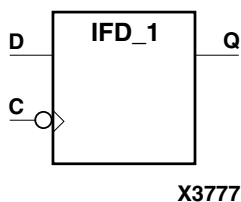
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFD, you would infer an FD and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFD_1

Input D Flip-Flop with Inverted Clock

Architectures Supported

IFD_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



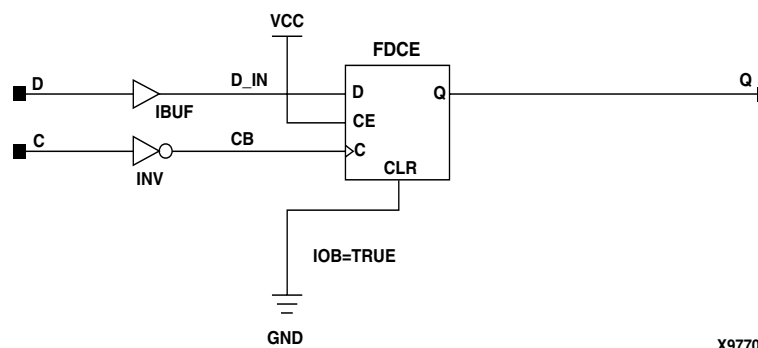
The IFD_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. The D input data is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously cleared with Low output when power is applied.

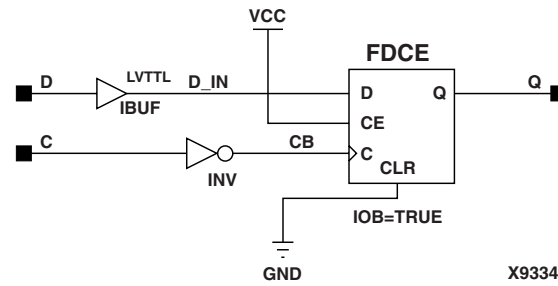
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1



IFD_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFD_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

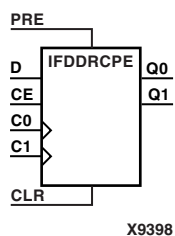
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFD_1, you would infer an FD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFDDRCPE

Dual Data Rate Input D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Architectures Supported

IFDDRCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



IFDDRCPE is a dual data rate (DDR) input D flip-flop with clock enable (CE) and asynchronous preset (PRE) and clear (CLR). It consists of one input buffer and two identical flip-flops (FDCPE).

When the asynchronous PRE is High and CLR is Low, both the Q0 and Q1 outputs are set High. When CLR is High, both outputs are reset Low. When PRE and CLR are Low and CE is High, data on the D input is loaded into the Q0 output on the Low-to-High C0 clock transition, and into the Q1 output on the Low-to-High C1 clock transition.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to IFDDRCPE components.

Inputs						Outputs	
C0	C1	CE	D	CLR	PRE	Q0	Q1
X	X	X	X	1	0	0	0
X	X	X	X	0	1	1	1
X	X	X	X	1	1	0	0
X	X	0	X	0	0	No Chg	No Chg
↑	X	1	D	0	0	D	No Chg
X	↑	1	D	0	0	No Chg	D

Usage

This component is inside the IOB and can be inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDDRCPE, you would infer an FDDRCPE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- IFDDRCPE: Double Data Rate Input Register with Async. Clear,
--           Async. Preset
--           and Clock Enable. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

IFDDRCPE_inst : IFDDRCPE
port map (
    Q0 => Q0,    -- Posedge data output
    Q1 => Q1,    -- Negedge data output
    C0 => C0,    -- 0 degree clock input
    C1 => C1,    -- 180 degree clock input
    CE => CE,    -- Clock enable input
    CLR => CLR,  -- Asynchronous reset input
    D => D,      -- Data input (connect directly to top-level port)
    PRE => PRE   -- Asynchronous preset input
);

-- End of IFDDRCPE_inst instantiation
```

Verilog Instantiation Template

```
// IFDDRCPE: Double Data Rate Input Register with Async. Clear, Async.
//           Preset and Clock Enable. Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

IFDDRCPE IFDDRCPE_inst (
    .Q0(Q0),    // Posedge data output
    .Q1(Q1),    // Negedge data output
    .C0(C0),    // 0 degree clock input
    .C1(C1),    // 180 degree clock input
    .CE(CE),    // Clock enable input
    .CLR(CLR),  // Asynchronous reset input
    .D(D),      // Data input (connect directly to top-level port)
    .PRE(PRE)   // Asynchronous preset input
);

// End of IFDDRCPE_inst instantiation
```

Commonly Used Constraints

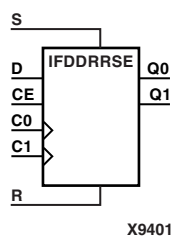
LOC, RLOC, and INIT

IFDDRSE

Dual Data Rate Input D Flip-Flop with Synchronous Reset and Set and Clock Enable

Architectures Supported

IFDDRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



IFDDRSE is a dual data rate (DDR) input D flip-flop with synchronous reset (R), synchronous set (S), and clock enable (CE). It consists of one input buffer and two identical flip-flops (FDRSE).

For the C0 input and Q0 output, reset (R) has precedence. The R input, when High, resets the Q0 output Low during the Low-to-High C0 clock transition. When S is High and R is Low, the Q0 output is set High during the Low-to-High C0 clock transition. For the C1 input and Q1 output, set (S) has precedence. The R input, when High, resets the Q1 output Low during the Low-to-High C1 clock transition. When S is High and R is Low, the Q0 output is set to High during the Low-to-High C1 clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

The INIT attribute does not apply to IFDDRSE components.

Inputs						Outputs	
C0	C1	CE	D	R	S	Q0	Q1
↑	X	X	X	1	0	0	No Chg
↑	X	X	X	0	1	1	No Chg
↑	X	X	X	1	1	0	No Chg
X	↑	X	X	1	0	No Chg	0
X	↑	X	X	0	1	No Chg	1
X	↑	X	X	1	1	No Chg	0
X	X	0	X	0	0	No Chg	No Chg
↑	X	1	D	0	0	D	No Chg
X	↑	1	D	0	0	No Chg	D

Usage

This component must be instantiated.

VHDL Instantiation Template

```
-- IFDDRSE: Double Data Rate Input Register with Sync. Clear,
--           Sync. Preset
--           and Clock Enable. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

IFDDRSE_inst : IFDDRSE
port map (
    Q0 => Q0,    -- Posedge data output
    Q1 => Q1,    -- Negedge data output
    C0 => C0,    -- 0 degree clock input
    C1 => C1,    -- 180 degree clock input
    CE => CE,    -- Clock enable input
    D => D,      -- Data input (connect directly to top-level port)
    R => R,      -- Synchronous reset input
    S => S       -- Synchronous preset input
);

-- End of IFDDRSE_inst instantiation
```

Verilog Instantiation Template

```
// IFDDRSE: Double Data Rate Input Register with Sync. Clear, Sync.
//           Preset and Clock Enable. Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

IFDDRSE IFDDRSE_inst (
    .Q0(Q0),    // Posedge data output
    .Q1(Q1),    // Negedge data output
    .C0(C0),    // 0 degree clock input
    .C1(C1),    // 180 degree clock input
    .CE(CE),    // Clock enable input
    .D(D),      // Data input (connect directly to top-level port)
    .R(R),      // Synchronous reset input
    .S(S)       // Synchronous preset input
);

// End of IFDDRSE_inst instantiation
```

Commonly Used Constraints

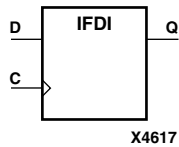
LOC, RLOC, INIT

IFDI

Input D Flip-Flop (Asynchronous Preset)

Architectures Supported

IFDI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



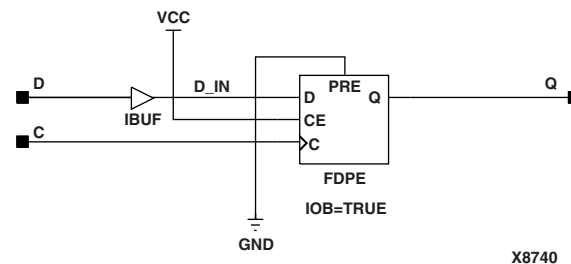
The IFDI D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied.

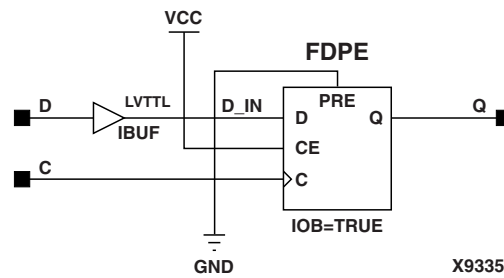
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



IFDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

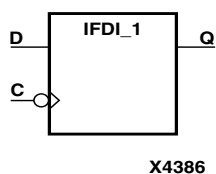
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDI, you would infer an FDP and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFDI_1

Input D Flip-Flop with Inverted Clock (Asynchronous Preset)

Architectures Supported

IFDI_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



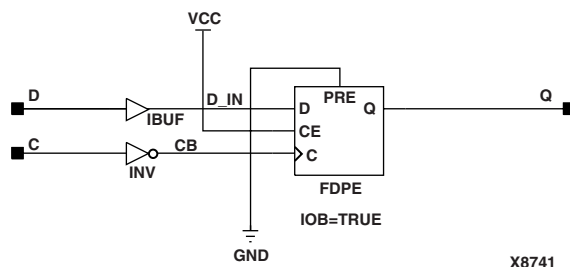
The IFDI_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. The data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin.

The flip-flop is asynchronously preset, output High, when power is applied.

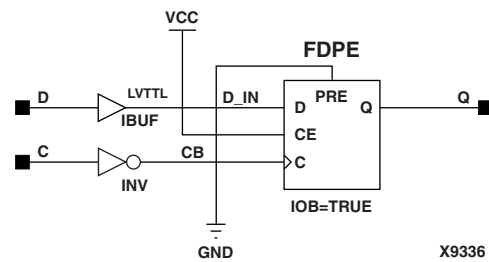
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
0	↓	0
1	↓	1



IFDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

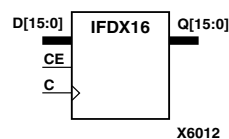
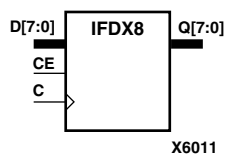
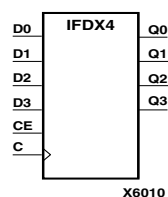
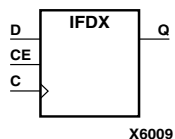
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDI_1, you would infer an FDP_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFDX, 4, 8, 16

Single- and Multiple-Input D Flip-Flops with Clock Enable

Architectures Supported

IFDX	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
IFDX4, IFDX8, IFDX16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



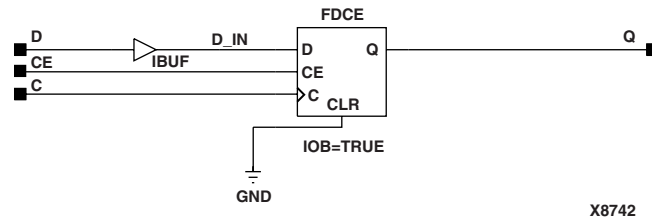
The IFDX D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD (without using an IBUF). The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When CE is Low, flip-flop outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

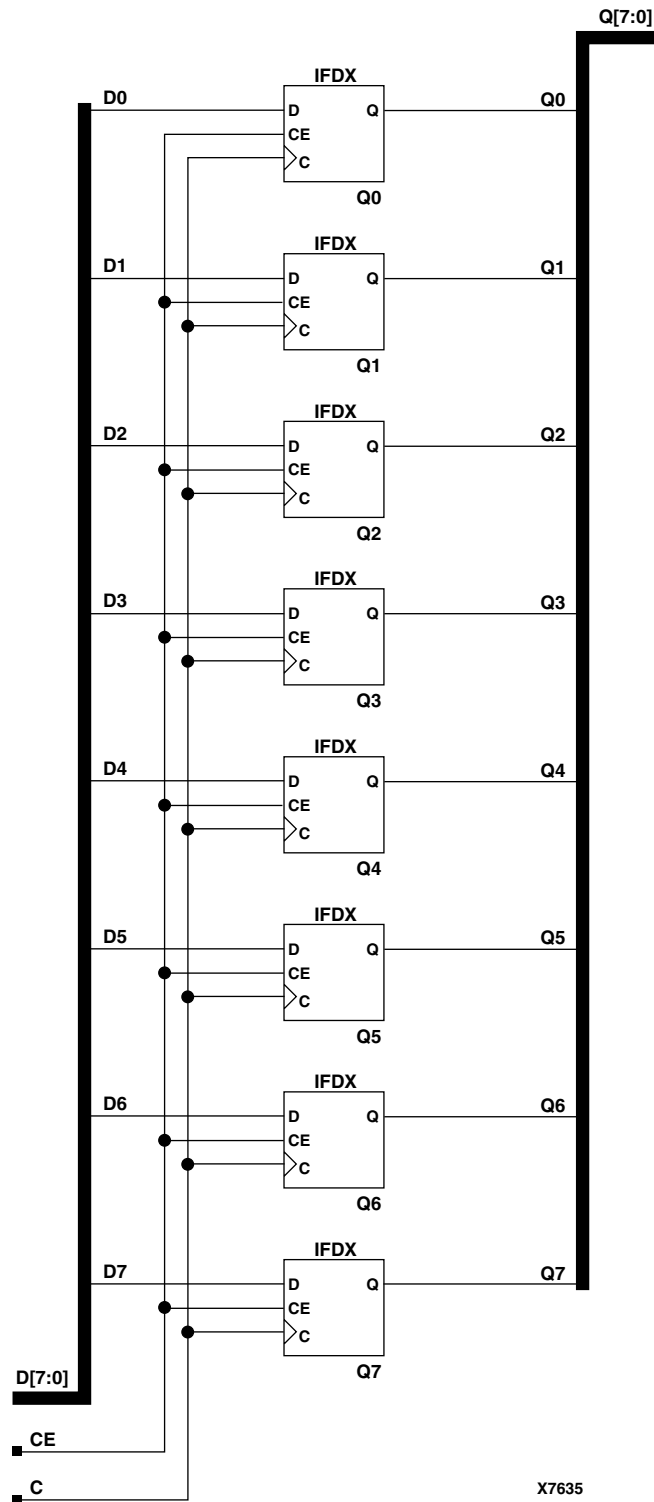
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	Dn	C	Qn
1	Dn	↑	Dn
0	X	X	No Chg



**IFDX Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E,
Virtex-II, Virtex-II Pro, Virtex-II Pro X**



IFDX8 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDX, you would infer an FDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for IFDX should be placed
-- after architecture statement but before begin keyword

component IFDX
  port (Q : out STD_ULOGIC;
        C : in STD_ULOGIC;
        CE : in STD_ULOGIC;
        D : in STD_ULOGIC);
end component;

-- Component Attribute specification for IFDX
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for IFDX should be placed
-- in architecture after the begin keyword

IFDX_INSTANCE_NAME : IFDX
  port map (Q => user_Q,
           C => user_C,
           CE => user_CE,
           D => user_D);
```

Verilog Instantiation Template

```
IFDX IFDX_instance_name (.Q (user_Q),
                        .C (user_C),
                        .CE (user_CE),
                        .D (user_D));
```

Commonly Used Constraints

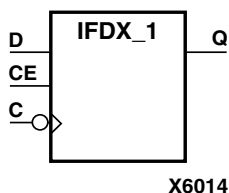
IOB

IFDX_1

Input D Flip-Flop with Inverted Clock and Clock Enable

Architectures Supported

IFDX_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The IFDX_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input also provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

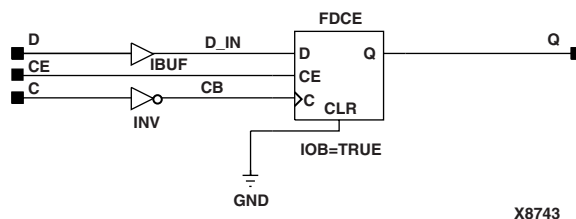
The flip-flop is asynchronously cleared with Low output, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

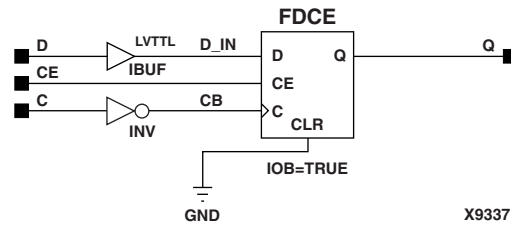
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For more information on IFDX_1, see “[ILDX, 4, 8, 16](#)”.

Inputs			Outputs
CE	D	C	Q
1	D	↓	D
0	X	X	No Chg



IFDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDX_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

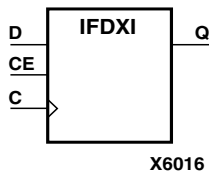
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDX_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFDXI

Input D Flip-Flop with Clock Enable (Asynchronous Preset)

Architectures Supported

IFDXI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The IFDXI D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

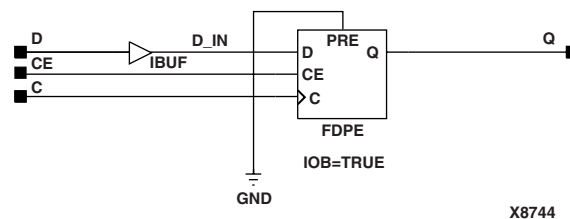
The flip-flop is asynchronously preset with High output, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

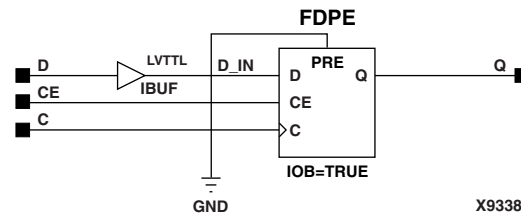
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see “ILDXI”.

Inputs			Outputs
CE	D	C	Q
1	D	↑	D
0	X	X	No Chg



IFDXI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDXI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

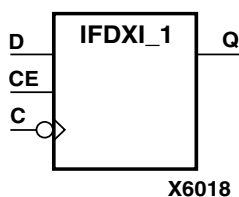
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDXI, you would infer an FDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

IFDXI_1

Input D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

Architectures Supported

IFDXI_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The IFDXI_1 D-type flip-flop is contained in an input/output block (IOB). The input (D) of the flip-flop is connected to an IPAD or an IOPAD. The D input provides data input for the flip-flop, which synchronizes data entering the chip. When CE is High, the data on input D is loaded into the flip-flop during the High-to-Low clock (C) transition and appears at the output (Q). The clock input can be driven by internal logic or through another external pin. When the CE pin is Low, the output (Q) does not change.

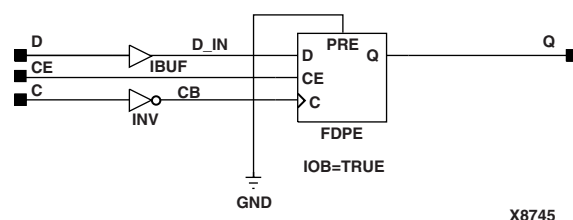
The flip-flop is asynchronously preset with High output when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

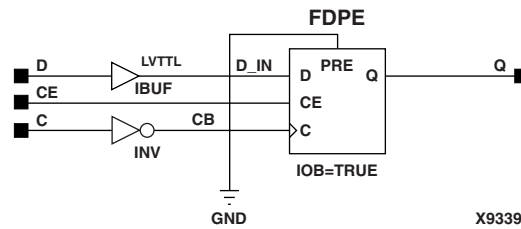
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see “ILDXI”.

Inputs			Outputs
CE	D	C	Q
1	D	↓	D
0	X	X	No Chg



IFDXI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



IFDXI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

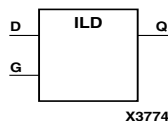
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an IFDXI_1, you would infer an FDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

ILD, 4, 8, 16

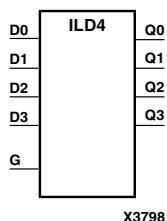
Transparent Input Data Latches

Architectures Supported

ILD	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
ILD4, ILD8, ILD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ILD, ILD4, ILD8, and ILD16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The ILD latch is contained in an input/output block (IOB), except for XC9500/XV/XL. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF). When the gate input (G) is High, data on the inputs (D) appears on the outputs (Q). Data on the D inputs during the High-to-Low G transition is stored in the latch.

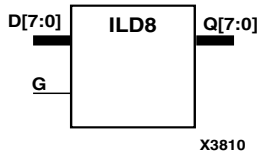


The latch is asynchronously cleared with Low output when power is applied.

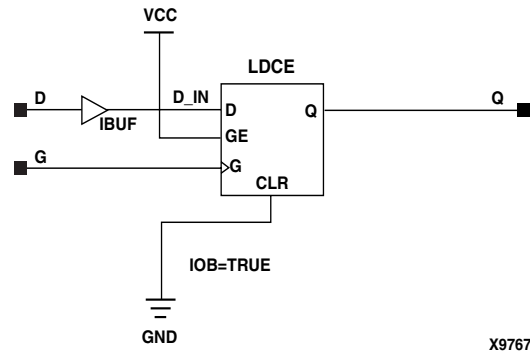
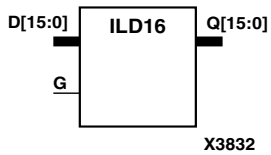
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

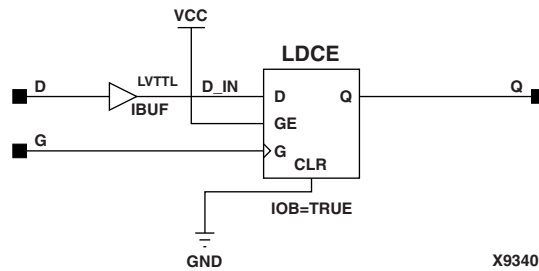
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



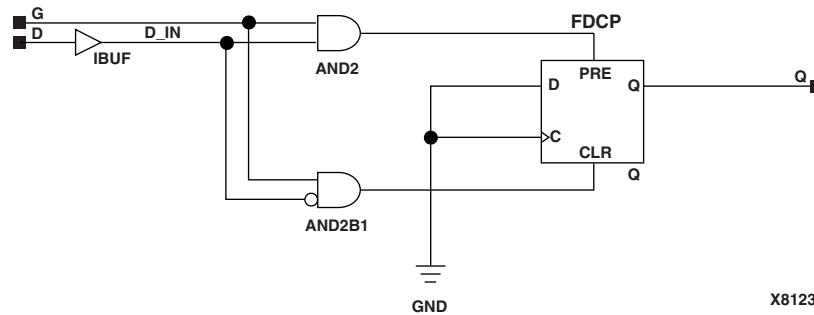
Inputs		Outputs
G	D	Q
1	1	1
1	0	0
0	X	No Chg
↓	D	D



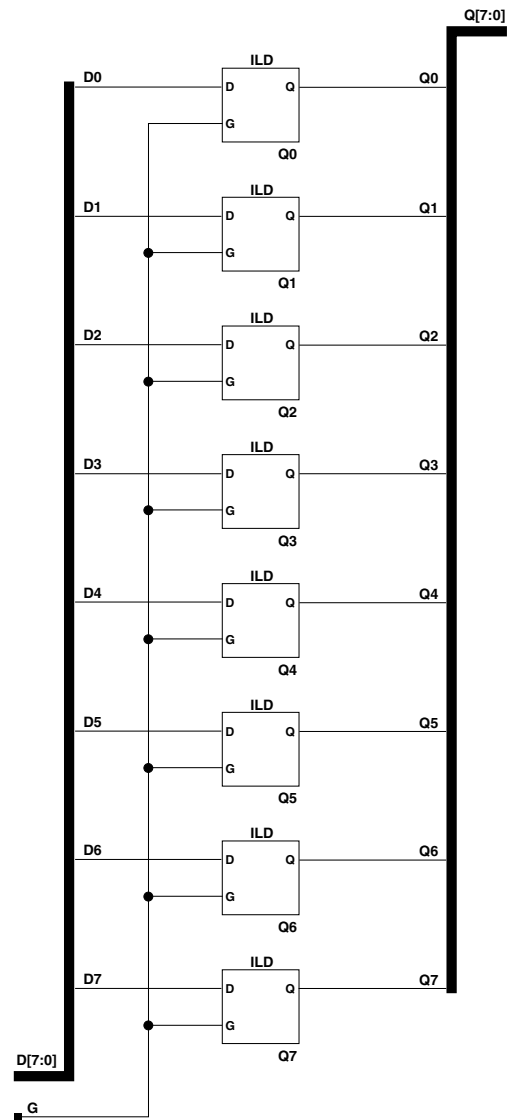
ILD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



ILD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



X7853

ILD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an `IOB=TRUE` attribute on the component in the UCF file or in the code. For instance, to get an ILD, you would infer an LD and put the `IOB = TRUE` attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for ILD should be placed
-- after architecture statement but before begin keyword

component ILD
  port (Q : out STD_ULOGIC;
        D : in STD_ULOGIC;
        DG : in STD_ULOGIC);
end component;

-- Component Attribute specification for ILD
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for ILD should be placed
-- in architecture after the begin keyword

ILD_INSTANCE_NAME : ILD
  port map (Q => user_Q,
            D => user_D,
            G => user_G);
```

Verilog Instantiation Template

```
ILD ILD_instance_name (.Q (user_Q),
                       .D (user_D),
                       .G (user_G));
```

Commonly Used Constraints

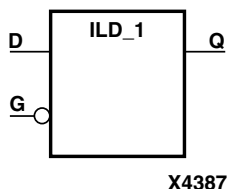
INIT

ILD_1

Transparent Input Data Latch with Inverted Gate

Architectures Supported

ILD_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



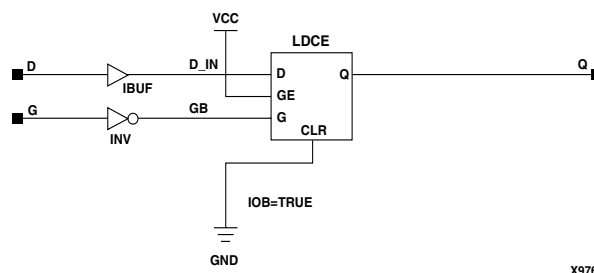
ILD_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

The latch is asynchronously cleared with Low output when power is applied.

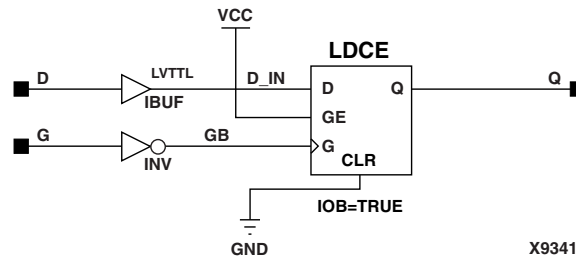
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	D
↑	D	D



ILD_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILD_1, you would infer an LD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

VHDL Instantiation Template

```
-- Component Declaration for ILD_1 should be placed
-- after architecture statement but before begin keyword

component ILD_1
    port (Q : out STD_ULOGIC;
          D : in  STD_ULOGIC;
          G : in  STD_ULOGIC);
end component;

-- Component Attribute specification for ILD_1
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for ILD_1 should be placed
-- in architecture after the begin keyword

ILD_1_INSTANCE_NAME : ILD_1
port map (Q => user_Q,
          D => user_D,
          G => user_G);
```

Verilog Instantiation Template

```
ILD_1 ILD_1_instance_name (.Q (user_Q),
                           .D (user_D),
                           .G (user_G));
```

Commonly Used Constraints

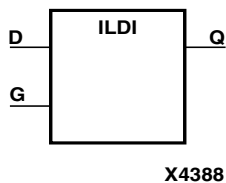
INIT

ILDI

Transparent Input Data Latch (Asynchronous Preset)

Architectures Supported

ILDI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ILDI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

The latch is asynchronously preset, output High, when power is applied.

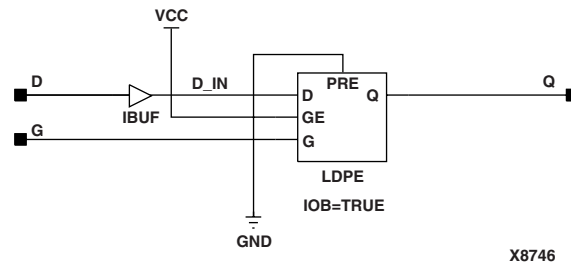
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

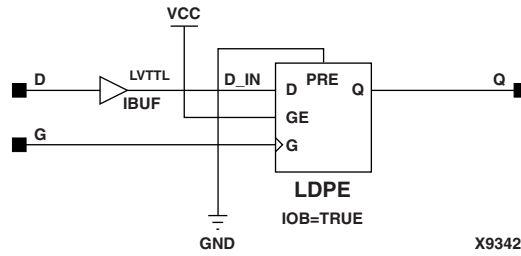
ILDIs and IFDIs

The ILDI is actually the input flip-flop master latch. It is possible to access two different outputs from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDI) corresponds to a falling edge-triggered flip-flop (IFDI_1). Similarly, a transparent Low latch (ILDI_1) corresponds to a rising edge-triggered flip-flop (IFDI).

Inputs		Outputs
G	D	Q
1	1	1
1	0	0
0	X	D
↓	D	D



ILDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

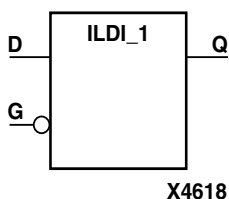
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDI, you would infer an LDP and put the IOB = TRUE attribute on the component. Or, you could use the map option -pr i to pack all input registers into the IOBs.

ILDI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

Architectures Supported

ILDI_1	
Spartan-II, Spartan-IIE	Macro*
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Macros cannot be implemented for Spartan-IIE.	



ILDI_1 is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

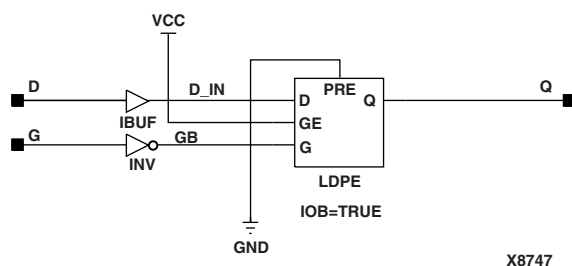
The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

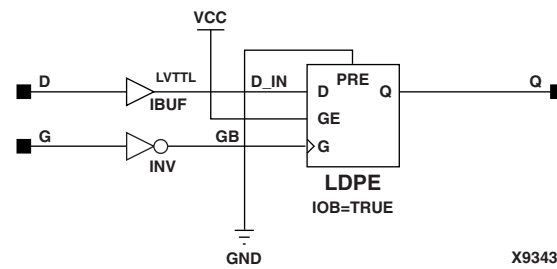
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For information on ILDI_1, see "ILDI".

Inputs		Outputs
G	D	Q
0	1	1
0	0	0
1	X	D
↑	D	D



ILDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

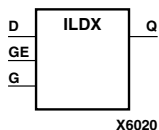
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDI_1, you would infer an LDP_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

ILDX, 4, 8, 16

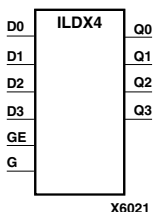
Transparent Input Data Latches

Architectures Supported

ILDX	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
ILDX4, ILDX8, ILDX16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



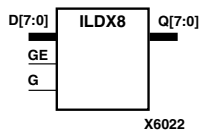
ILDX, ILDX4, ILDX8, and ILDX16 are single or multiple transparent data latches, which can be used to hold transient data entering a chip. The latch input (D) is connected to an IPAD or an IOPAD (without using an IBUF).



The latch is asynchronously cleared, output Low, when power is applied.

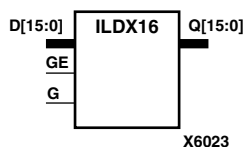
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

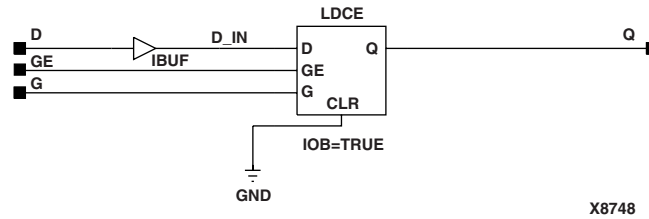


ILDXs and IFDXs

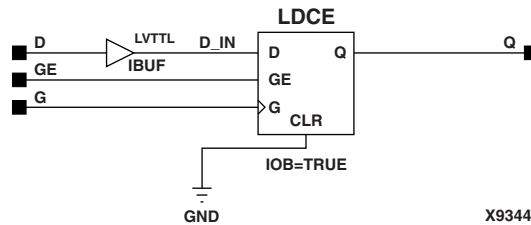
The ILDX is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILDX) corresponds to a falling edge-triggered flip-flop (IFDX_1). Similarly, a transparent Low latch (ILDX_1) corresponds to a rising edge-triggered flip-flop (IFDX).



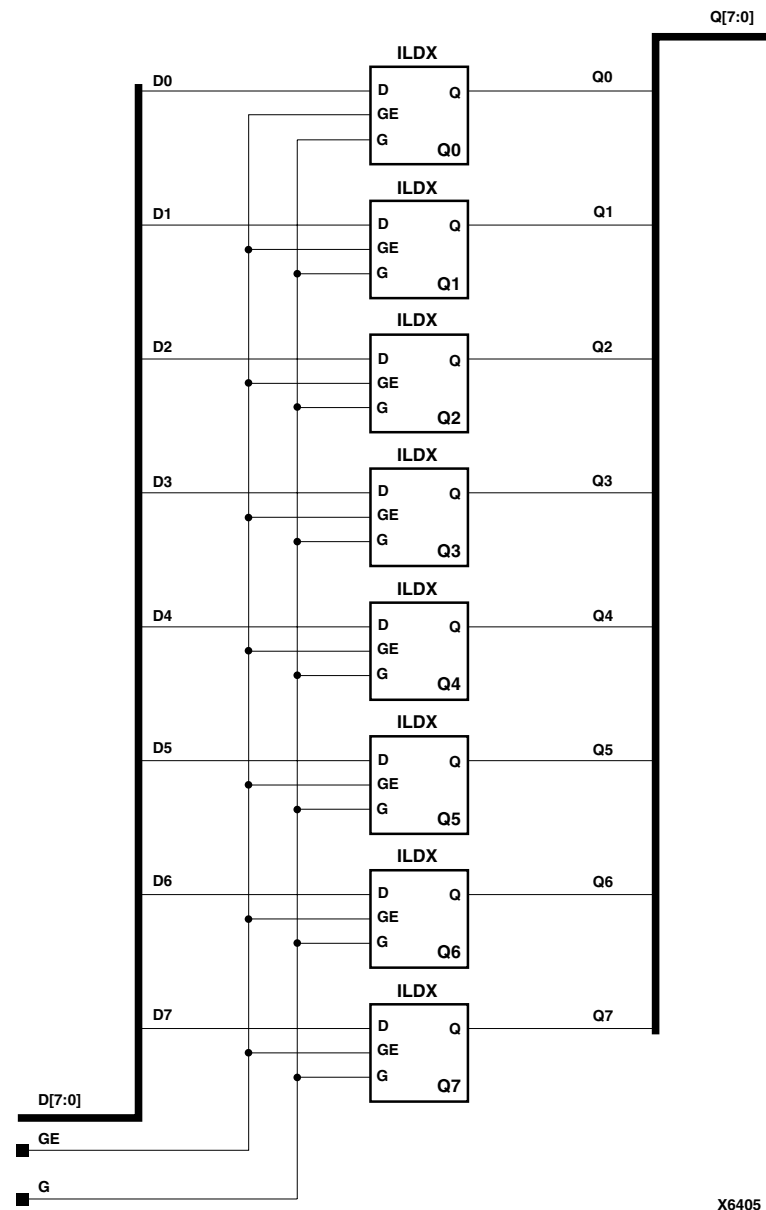
Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	D



ILDx Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDx Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X6405

ILD_X8 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

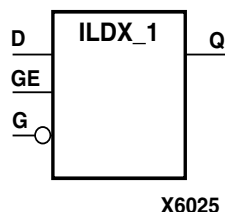
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDX, you would infer an LDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

ILDX_1

Transparent Input Data Latch with Inverted Gate

Architectures Supported

ILDX_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ILDX_1 is a transparent data latch, which can be used to hold transient data entering a chip. When the gate input (G) is Low, data on the data input (D) appears on the data output (Q). Data on D during the Low-to-High G transition is stored in the latch.

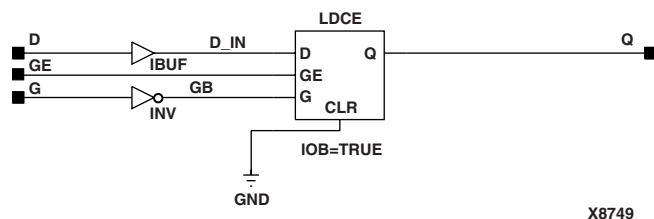
The latch is asynchronously cleared with Low output, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

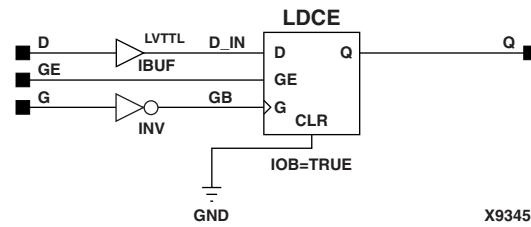
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For more information on ILDX_1, see “ILDX, 4, 8, 16”.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	D



ILDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILDX_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

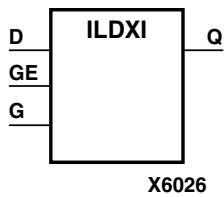
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDX_1, you would infer an LDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

ILD XI

Transparent Input Data Latch (Asynchronous Preset)

Architectures Supported

ILD XI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ILD XI is a transparent data latch, which can hold transient data entering a chip. When the gate input (G) is High, data on the input (D) appears on the output (Q). Data on the D input during the High-to-Low G transition is stored in the latch.

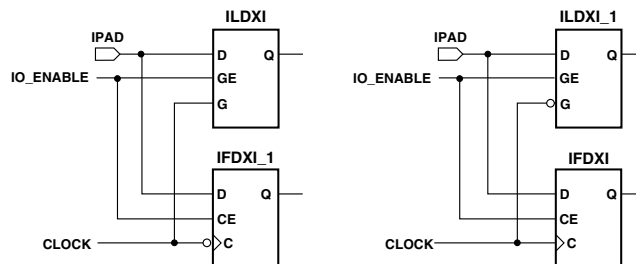
The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

ILD XIs and IFDXIs

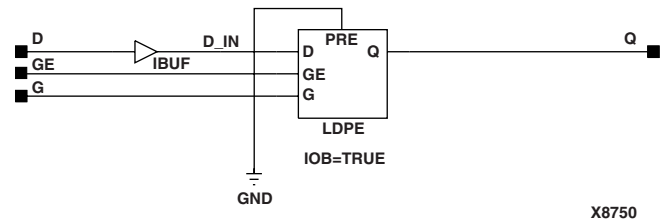
The ILDXI is actually the input flip-flop master latch. Two different outputs can be accessed from the input flip-flop: one that responds to the level of the clock signal and another that responds to an edge of the clock signal. When using both outputs from the same input flip-flop, a transparent High latch (ILD XI) corresponds to a falling edge-triggered flip-flop (IFDXI_1). Similarly, a transparent Low latch (ILD XI_1) corresponds to a rising edge-triggered flip-flop (IFDXI). See the following figure for legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations.



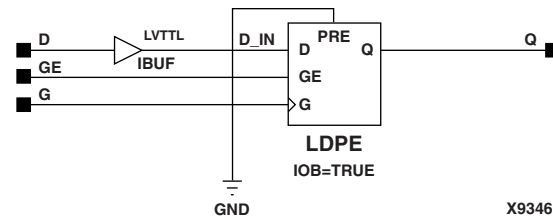
X6027

Legal Combinations of IFDXI and ILDXI for a Single IOB

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	X	No Chg
1	1	1	1
1	1	0	0
1	↓	D	D



ILD XI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD XI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

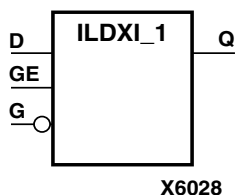
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDXI, you would infer an LDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option -pr i to pack all input registers into the IOBs.

ILD XI_1

Transparent Input Data Latch with Inverted Gate (Asynchronous Preset)

Architectures Supported

ILD XI_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



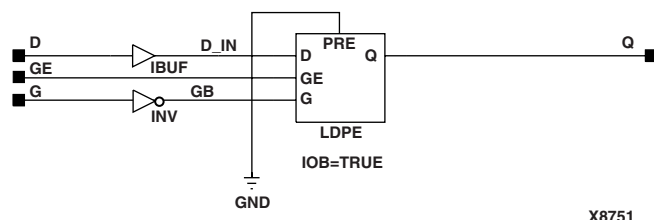
ILD XI_1 is a transparent data latch, which can hold transient data entering a chip. The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

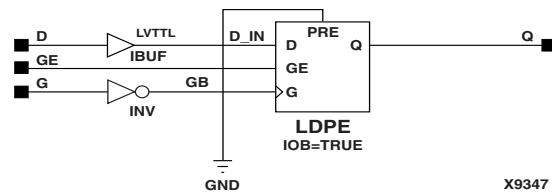
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

For information on legal IFDXI, IFDXI_1, ILDXI, and ILDXI_1 combinations, see "ILD XI".

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	X	No Chg
1	0	1	1
1	0	0	0
1	↑	D	D



ILD XI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



ILD XI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

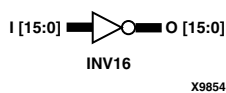
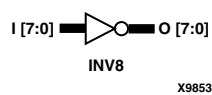
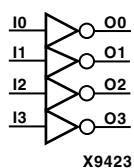
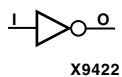
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an ILDXI_1, you would infer an LDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr i` to pack all input registers into the IOBs.

INV, 4, 8, 16

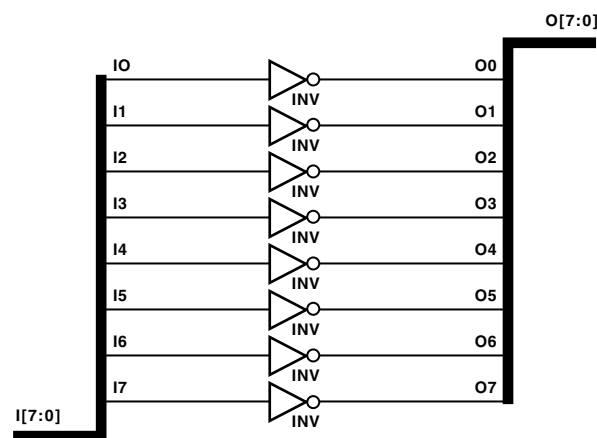
Single and Multiple Inverters

Architectures Supported

INV	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
INV4, INV8, INV16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



INV, INV4, INV8, and INV16 are single and multiple inverters that identify signal inversions in a schematic.



INV8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, this design element can be instantiated or inferred.

VHDL Instantiation Template

```
-- Component Declaration for INV should be placed
-- after architecture statement but before begin keyword

component INV
    port (O : out STD_ULOGIC;
          I : in STD_ULOGIC);
end component;

-- Component Attribute specification for INV
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for INV should be placed
-- in architecture after the begin keyword

INV_INSTANCE_NAME : INV
    port map (O => user_O,
              I => user_I);
```

Verilog Instantiation Template

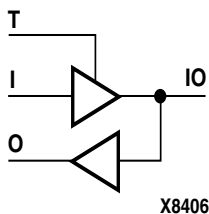
```
INV instance_name (.O (user_O),
                  .I (user_I));
```

IOBUF

Bi-Directional Buffer with Selectable I/O Interface

Architectures Supported

IOBUF	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



For Spartan-II, Spartan-III, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, IOBUF are bi-directional buffers whose I/O interface corresponds to a specific I/O standard. You can attach an IOSTANDARD attribute to an IOBUF instance.

IOBUFs are composites of IBUF and OBUFT elements. The O output is X (unknown) when IO (input/output) is Z. IOBUFs can be implemented as interconnections of their component elements.

The hardware implementation of the I/O standards requires that you follow a set of usage rules for the buffers. See the ["Usage Rules"](#) section.

Inputs		Bidirectional	Outputs
T	I	IO	O
1	X	Z	X
0	1	1	1
0	0	0	0

In the above table, where T = 1, data can be driven onto O.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- IOBUF: Single-ended Bi-directional Buffer
--       All devices
-- The current version of the Xilinx HDL Libraries Guide

IOBUF_inst : IOBUF
-- Edit the following generics to specify the I/O standard, drive and
slew rate.
generic map (
    DRIVE => 12,
    IOSTANDARD => "LVCMOS25",
```

```
        SLEW => "SLOW")  
port map (  

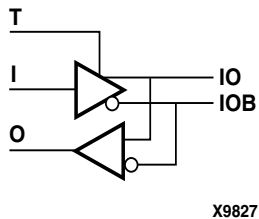
```

IOBUFDS

3-State Differential Signaling I/O Buffer with Active Low Output Enable

Architectures Supported

IOBUFDS	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9827

IOBUFDS is a single 3-state, differential signaling input/output buffer with active Low output enable.

Inputs		Bidirectional		Outputs
I	T	IO	IOB	O
X	1	Z	Z	- *
0	0	0	1	0
1	0	1	0	1

* The dash (-) means No Change.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- IOBUFDS: Differential Bi-directional Buffer
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

IOBUFDS_inst : IOBUFDS
-- Edit the following generics to specify the I/O standard, drive and
slew rate.
generic map (
    DRIVE => 12,
    IOSTANDARD => "LVDS_25",
    SLEW => "SLOW")
port map (
    O => O,      -- Buffer output
    IO => IO,    -- Diff_p inout (connect directly to top-level port)
    IOB => IOB,  -- Diff_n inout (connect directly to top-level port)
    I => I,      -- Buffer input
    T => T       -- 3-state enable input
);
```

Verilog Instantiation Template

```
// IOBUFDS: Differential Bi-directional Buffer
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

IOBUFDS IOBUFDS_inst (
    .O(O),      // Buffer output
    .IO(IO),    // Diff_p inout (connect directly to
                // top-level port)
    .IOB(IOB), // Diff_n inout (connect directly to
                // top-level port)
    .I(I),      // Buffer input
    .T(T)       // 3-state enable input
);

// Edit the following defparams to specify the I/O standard, drive and
// slew rate. If the instance name is change, that change needs to be
// reflecting the this defparam.

defparam IOBUFDS_inst.DRIVE = 12;
defparam IOBUFDS_inst.IOSTANDARD = "LVDS_25";
defparam IOBUFDS_inst.SLEW = "SLOW";

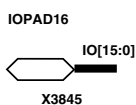
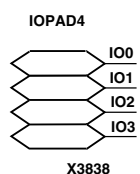
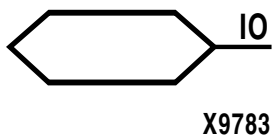
// End of IOBUFDS_inst instantiation
```

IOPAD, 4, 8, 16

Single- and Multiple-Input/Output Pads

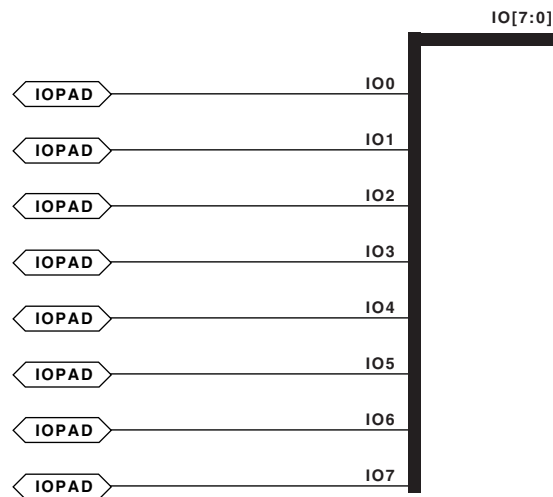
Architectures Supported

IOPAD	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
IOPAD4, IOPAD8, IOPAD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



IOPAD, IOPAD4, IOPAD8, and IOPAD16 are single and multiple input/output pads. The IOPAD is a connection point from a device pin, used as a bidirectional signal, to a PLD device. The IOPAD is connected internally to an input/output block (IOB), which is configured by the software as a bidirectional block. Bidirectional blocks can consist of any combination of a 3-state output buffer (such as OBUFT or OFDE) and any available input buffer (such as IBUF or IFD). See the appropriate CAE tool interface user guide for details on assigning pin location and identification.

Note: The LOC attribute cannot be used on IOPAD multiples.



X7854

IOPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

Commonly Used Constraints

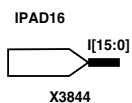
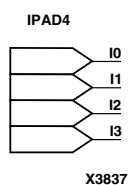
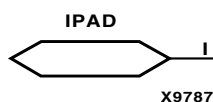
IOBDELAY, PULLDOWN

IPAD, 4, 8, 16

Single- and Multiple-Input Pads

Architectures Supported

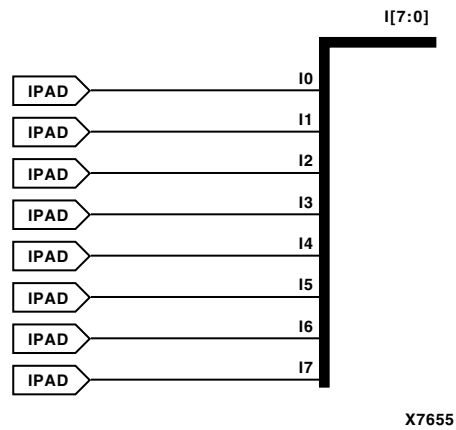
IPAD	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
IPAD4, IPAD8, IPAD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



IPAD, IPAD4, IPAD8, and IPAD16 are single and multiple input pads. The IPAD is a connection point from a device pin used for an input signal to the PLD device. It is connected internally to an input/output block (IOB), which is configured by the software as an IBUF, IFD, or ILD. See the appropriate CAE tool interface user guide for details on assigning pin location and identification.

For Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, pads must be used to drive IBUF and IBUFG inputs. An IPAD can be inferred by NGDBUILD if one is missing on an IBUF or IBUFG input.

Note: The LOC attribute cannot be used on IPAD multiples.



IPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

Commonly Used Constraints

IOBDELAY, PULLDOWN

JTAGPPC

JTAG Primitive for the Power PC

Architectures Supported

JTAGPPC	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Not supported for Virtex-II. Supported for Virtex-II Pro and Virtex-II Pro X only.	

The JTAGPPC block allows connection from the JTAG logic in the PPC405 core to the JTAG logic of Virtex-II Pro and Virtex-II Pro X devices. The connections are made through programmable routing and so the connection only exists after configuration. Following is an example instantiation of the JTAGPPC block in Verilog:

```
JTAGPPC IJTAGPPC ( .TDOTSPPC (TDO_TS_PPC) ,
                  .TDOPPC (TDO_PPC) , .TMS (TMS_PPC) ,
                  .TDIPPC (TDI_PPC) , .TCK (TCK_PPC) );

PPC405 IPPC405 (
    ...
    .JTGC405TCK (TCK_PPC) ,
    .JTGC405TDI (TDI_PPC) ,
    .JTGC405TMS (TMS_PPC) ,
    .C405JTGTDO (TDO_PPC) ,
    .C405JTGTDOEN (TDO_TS_PPC) ,
    ...
)
```

When the block is instantiated in this fashion, the instruction registers of the PPC405 and the Virtex-II Pro and Virtex-II Pro X devices are linked in series.

The following table lists the input and output pins for JTAGPPC.

Inputs	Outputs
TDOPPC	TCK
TDOTSPPC	TDIPPC
	TMS

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for JTAGPPC should be placed
-- after architecture statement but before begin keyword

component JTAGPPC
  port (TCK : out STD_ULOGIC;
        TDIPPC : out STD_ULOGIC;
        TMS : out STD_ULOGIC;
        TDOPPC : in STD_ULOGIC;
        TDOTSPPC : in STD_ULOGIC);
end component;

-- Component Attribute specification for JTAGPPC
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for JTAGPPC should be placed
-- in architecture after the begin keyword

JTAGPPC_INSTANCE_NAME : JTAGPPC
  port map (TCK => user_TCK,
           TDIPPC => user_TDIPPC,
           TMS => user_TMS,
           TDOPPC => user_TDOPPC,
           TDOTSPPC => user_TDOTSPPC);
```

Verilog Instantiation Template

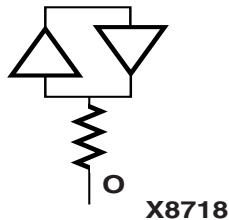
```
JTAGPPC JTAGPPC_instance_name (.TCK (user_TCK),
                               .TDIPPC (user_TDIPPC),
                               .TMS (user_TMS),
                               .TDOPPC (user_TDOPPC),
                               .TDOTSPPC (user_TDOTSPPC));
```

KEEPER

KEEPER Symbol

Architectures Supported

KEEPER	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	No
CoolRunner-II	Primitive



KEEPER is a weak keeper element used to retain the value of the net connected to its bidirectional O pin. For example, if a logic 1 is being driven onto the net, KEEPER drives a weak/resistive 1 onto the net. If the net driver is then 3-stated, KEEPER continues to drive a weak/resistive 1 onto the net.

For additional information on using a KEEPER element with SelectIO components, see the [“Usage Rules”](#) section in [“IBUF, 4, 8, 16.”](#)

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- KEEPER: I/O Buffer Weak Keeper
-- All FPGA, CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

KEEPER_inst : KEEPER
port map (
    O => O    -- Keeper output (connect directly to top-level port)
);

-- End of KEEPER_inst instantiation
```

Verilog Instantiation Template

```
// KEEPER: I/O Buffer Weak Keeper
// All FPGA, CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

KEEPER KEEPER_inst (
    .O(O),    // Keeper output (connect directly to top-level port)
);

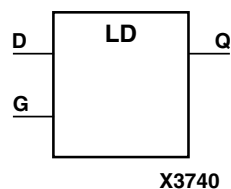
// End of KEEPER_inst instantiation
```


LD

Transparent Data Latch

Architectures Supported

LD	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



LD is a transparent data latch. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

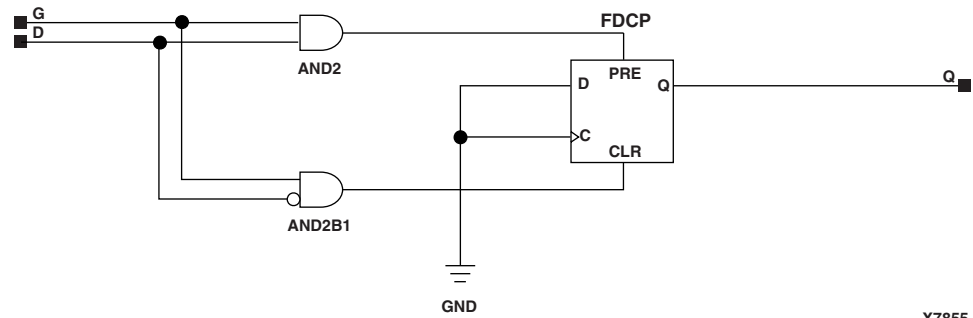
The latch is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	D



LD Implementation XC9500/XV/XL

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LD should be placed
-- after architecture statement but before begin keyword
```

```
component LD
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for LD
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of LD_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for LD should be placed
-- in architecture after the begin keyword
```

```
LD_INSTANCE_NAME : LD
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            D => user_D,
            G => user_G);
```


Verilog Instantiation Template

```
LD LD_instance_name (.Q (user_Q),  
                    .D (user_D),  
                    .G (user_G));  
  
defparam LD_instance_name.INIT = bit_value;
```

Commonly Used Constraints

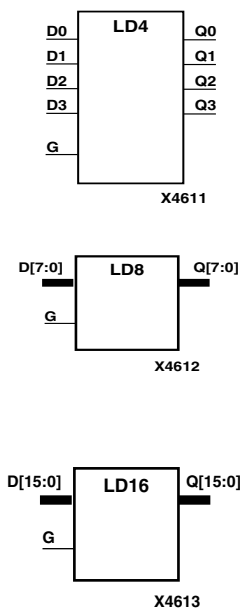
INIT

LD4, 8, 16

Multiple Transparent Data Latches

Architectures Supported

LD4, LD8, LD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



LD4, LD8, and LD16 have, respectively, 4, 8, and 16 transparent data latches with a common gate enable (G). The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

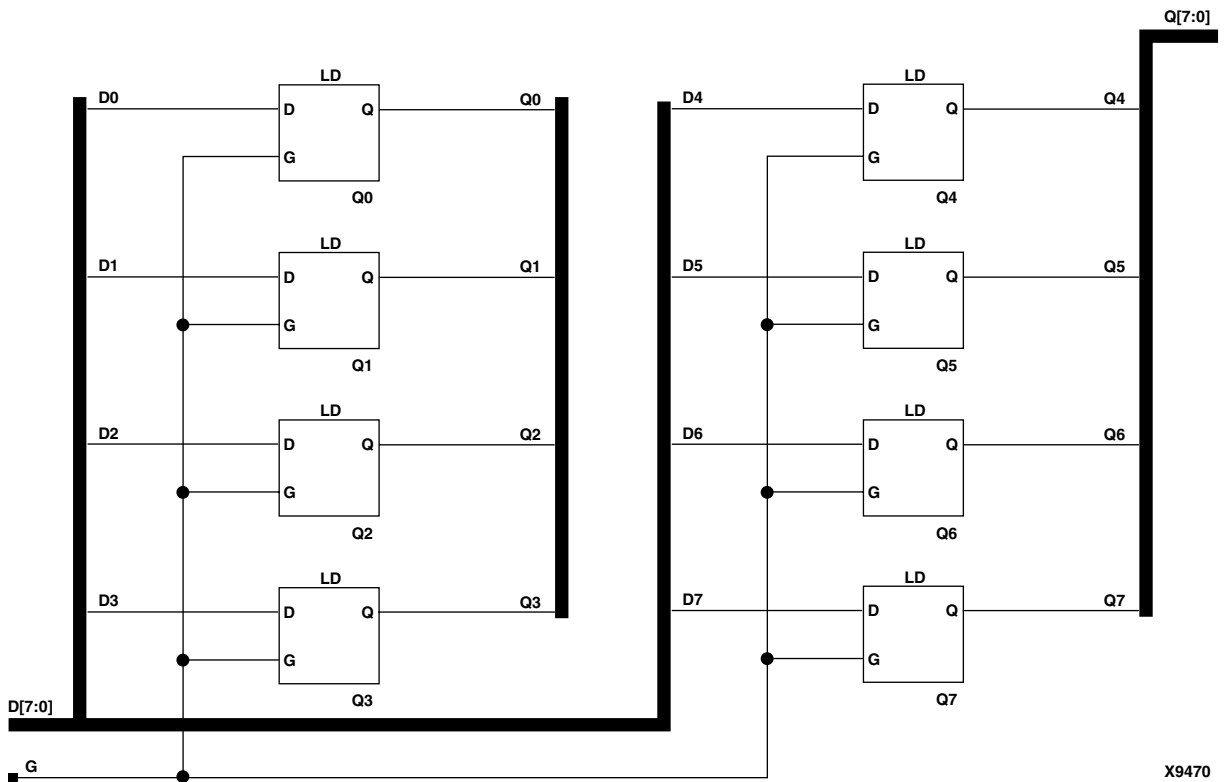
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

See "LD" for information on single transparent data latches.

Inputs		Outputs
G	D	Q
1	0	0
1	1	1
0	X	No Chg
↓	D	D



LD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

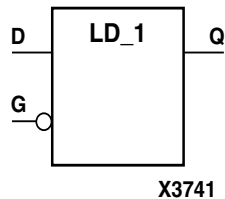
For HDL, these design elements are inferred rather than instantiated.

LD_1

Transparent Data Latch with Inverted Gate

Architectures Supported

LD_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LD_1 is a transparent data latch with an inverted gate. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LD_1 should be placed
-- after architecture statement but before begin keyword
```

```
component LD_1
```

```
-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      D : in STD_ULOGIC;
      G : in STD_ULOGIC);
end component;

-- Component Attribute specification for LD_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LD_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LD_1 should be placed
-- in architecture after the begin keyword

LD_1_INSTANCE_NAME : LD_1
-- synthesis translate_off
generic map(
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          D => user_D,
          G => user_G);
```

Verilog Instantiation Template

```
LD_1 LD_1_instance_name (.Q (user_Q),
                        .D (user_D),
                        .G (user_G));

defparam LD_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

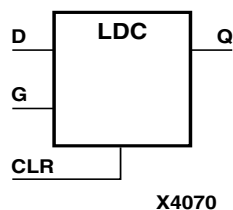
INIT

LDC

Transparent Data Latch with Asynchronous Clear

Architectures Supported

LDC	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Macro
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



LDC is a transparent data latch with asynchronous clear. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input is High and CLR is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDC should be placed
-- after architecture statement but before begin keyword
```

```
component LDC
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULONGIC;
        CLR : in STD_ULONGIC;
        D : in STD_ULONGIC;
        G : in STD_ULONGIC);
end component;

-- Component Attribute specification for LDC
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDC_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDC should be placed
-- in architecture after the begin keyword

LDC_INSTANCE_NAME : LDC
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G);
```

Verilog Instantiation Template

```
LDC LDC_instance_name (.Q (user_Q),
                       .CLR (user_CLR),
                       .D (user_D),
                       .G (user_G));

defparam LDC_instance_name.INIT = bit_value;
```

Commonly Used Constraints

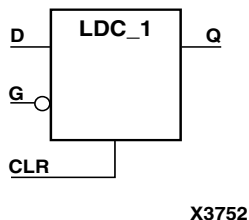
INIT

LDC_1

Transparent Data Latch with Asynchronous Clear and Inverted Gate

Architectures Supported

LDC_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDC_1 is a transparent data latch with asynchronous clear and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs (D and G) and resets the data (Q) output Low. Q reflects the data (D) input while the gate enable (G) input and CLR are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CLR	G	D	Q
1	X	X	0
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDC_1 should be placed  
-- after architecture statement but before begin keyword
```

```
component LDC_1  
  -- synthesis translate_off  
  generic (  
    INIT : bit := '1');  
  -- synthesis translate_on  
  port (Q : out STD_ULOGIC;  
        CLR : in STD_ULOGIC;  
        D : in STD_ULOGIC;  
        G : in STD_ULOGIC);  
end component;
```

```
-- Component Attribute specification for LDC_1  
-- should be placed after architecture declaration but  
-- before the begin keyword
```

```
attribute INIT : string;  
attribute INIT of LDC_1_instance_name : label is "0";  
-- values can be (0 or 1)
```

```
-- Component Instantiation for LDC_1 should be placed  
-- in architecture after the begin keyword
```

```
LDC_1_INSTANCE_NAME : LDC_1  
  -- synthesis translate_off  
  generic map(  
    INIT => bit_value)  
  -- synthesis translate_on  
  port map (Q => user_Q,  
            CLR => user_CLR,  
            D => user_D,  
            G => user_G);
```

Verilog Instantiation Template

```
LDC_1 LDC_1_instance_name (.Q (user_Q),  
                          .CLR (user_CLR),  
                          .D (user_D),  
                          .G (user_G));
```

```
defparam LDC_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

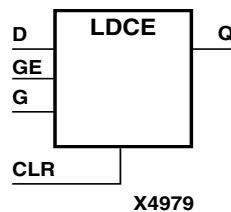
INIT

LDCE

Transparent Data Latch with Asynchronous Clear and Gate Enable

Architectures Supported

LDCE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDCE is a transparent data latch with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR is Low. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDCE should be placed
-- after architecture statement but before begin keyword

component LDCE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDCE should be placed
-- in architecture after the begin keyword

LDCE_INSTANCE_NAME : LDCE
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            GE => user_GE);
```

Verilog Instantiation Template

```
LDCE LDCE_instance_name (.Q (user_Q),
                          .CLR (user_CLR),
                          .D (user_D),
                          .G (user_G),
                          .GE (user_GE));

defparam LDCE_instance_name.INIT = bit_value;
```

Commonly Used Constraints

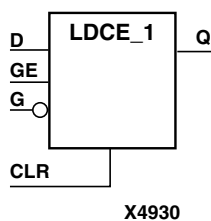
INIT

LDCE_1

Transparent Data Latch with Asynchronous Clear, Gate Enable, and Inverted Gate

Architectures Supported

LDCE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDCE_1 is a transparent data latch with asynchronous clear, gate enable, and inverted gate. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) output Low. Q reflects the data (D) input while the gate (G) input and CLR are Low and gate enable (GE) is High. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	GE	G	D	Q
1	X	X	X	0
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

-- Component Declaration for LDCE_1 should be placed
-- after architecture statement but before begin keyword

```
component LDCE_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for LDCE_1
-- should be placed after architecture declaration but
-- before the begin keyword

```
attribute INIT : string;
attribute INIT of LDCE_1_instance_name : label is "0";
-- values can be (0 or 1)
```

-- Component Instantiation for LDCE_1 should be placed
-- in architecture after the begin keyword

```
LDCE_1_INSTANCE_NAME : LDCE_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            GE => user_GE);
```

Verilog Instantiation Template

```
LDCE_1 LDCE_1_instance_name (.Q (user_Q),
                             .CLR (user_CLR),
                             .D (user_D),
                             .G (user_G),
                             .GE (user_GE));
```

```
defparam LDCE_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

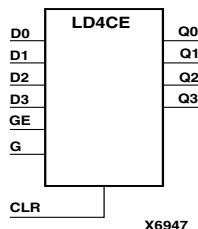
INIT

LD4CE, LD8CE, LD16CE

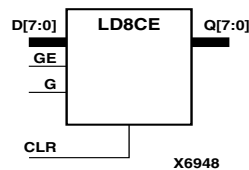
Transparent Data Latches with Asynchronous Clear and Gate Enable

Architectures Supported

LD4CE, LD8CE, LD16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



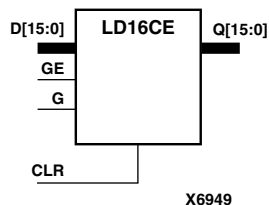
LD4CE, LD8CE, and LD16CE have, respectively, 4, 8, and 16 transparent data latches with asynchronous clear and gate enable. When the asynchronous clear input (CLR) is High, it overrides the other inputs and resets the data (Q) outputs Low. Q reflects the data (D) inputs while the gate (G) and gate enable (GE) are High, and CLR is Low. If GE is Low, data on D cannot be latched. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.



The latch is asynchronously cleared with Low output when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

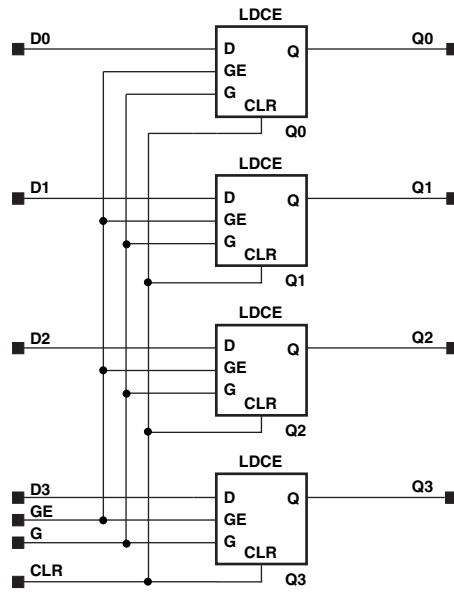
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



Inputs				Outputs
CLR	GE	G	Dn	Qn
1	X	X	X	0
0	0	X	X	No Chg
0	1	1	1	1
0	1	1	0	0
0	1	0	X	No Chg
0	1	↓	Dn	Dn

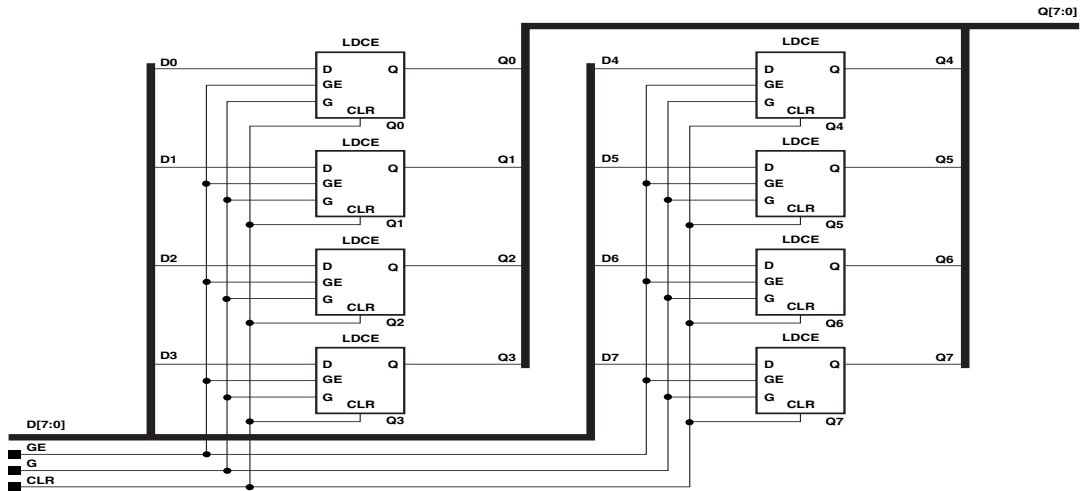
Dn = referenced input, for example, D0, D1, D2

Qn = referenced output, for example, Q0, Q1, Q2



X6538

LD4CE Implementation Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X6385

LD8CE Implementation Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

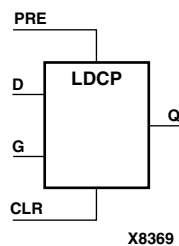
For HDL, these design elements are supported for inference only.

LDCP

Transparent Data Latch with Asynchronous Clear and Preset

Architectures Supported

LDCP	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Macro
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



LDCP is a transparent data latch with data (D), asynchronous clear (CLR) and preset (PRE) inputs. When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input is High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	1	1	1
0	0	1	0	0
0	0	0	X	No Chg
0	0	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDCP should be placed
-- after architecture statement but before begin keyword

component LDCP
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCP_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDCP should be placed
-- in architecture after the begin keyword

LDCP_INSTANCE_NAME : LDCP
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCP LDCP_instance_name (.Q (user_Q),
                        .CLR (user_CLR),
                        .D (user_D),
                        .G (user_G),
                        .PRE (user_PRE));

defparam LDCP_instance_name.INIT = bit_value;
```

Commonly Used Constraints

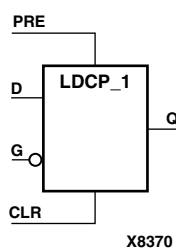
INIT

LDCP_1

Transparent Data Latch with Asynchronous Clear and Preset and Inverted Gate

Architectures Supported

LDCP_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDCP_1 is a transparent data latch with data (D), asynchronous clear (CLR), preset (PRE) inputs, and inverted gate (G). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate (G) input, CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
CLR	PRE	G	D	Q
1	X	X	X	0
0	1	X	X	1
0	0	0	1	1
0	0	0	0	0
0	0	1	X	No Chg
0	0	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

-- Component Declaration for LDCP_1 should be placed
-- after architecture statement but before begin keyword

```
component LDCP_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for LDCP_1
-- should be placed after architecture declaration but
-- before the begin keyword

```
attribute INIT : string;
attribute INIT of LDCP_1_instance_name : label is "0";
-- values can be (0 or 1)
```

-- Component Instantiation for LDCP_1 should be placed
-- in architecture after the begin keyword

```
LDCP_1_INSTANCE_NAME : LDCP_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCP_1 LDCP_1_instance_name (.Q (user_Q),
                             .CLR (user_CLR),
                             .D (user_D),
                             .G (user_G),
                             .PRE (user_PRE));
```

```
defparam LDCP_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

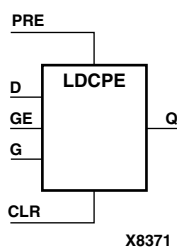
INIT

LDCPE

Transparent Data Latch with Asynchronous Clear and Preset and Gate Enable

Architectures Supported

LDCPE	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDCPE is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), and gate enable (GE). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High and CLR and PRE are Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	1	0	0
0	0	1	1	1	1
0	0	1	0	X	No Chg
0	0	1	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDCPE should be placed
-- after architecture statement but before begin keyword
```

```
component LDCPE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for LDCPE
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of LDCPE_instance_name : label is "0";
-- values can be (0 or 1)
```

```
-- Component Instantiation for LDCPE should be placed
-- in architecture after the begin keyword
```

```
LDCPE_INSTANCE_NAME : LDCPE
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            GE => user_GE,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCPE LDCPE_instance_name (.Q (user_Q),
                           .CLR (user_CLR),
                           .D (user_D),
                           .G (user_G),
                           .GE (user_D),
                           .PRE (user_PRE));
defparam LDCPE_instance_name.INIT = bit_value;
```

Commonly Used Constraints

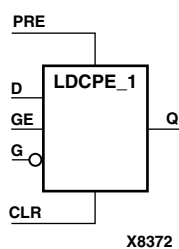
INIT

LDCPE_1

Transparent Data Latch with Asynchronous Clear and Preset, Gate Enable, and Inverted Gate

Architectures Supported

LDCPE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDCPE_1 is a transparent data latch with data (D), asynchronous clear (CLR), asynchronous preset (PRE), gate enable (GE), and inverted gate (G). When CLR is High, it overrides the other inputs and resets the data (Q) output Low. When PRE is High and CLR is Low, it presets the data (Q) output High. Q reflects the data (D) input while gate enable (GE) is High and gate (G), CLR, and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs					Outputs
CLR	PRE	GE	G	D	Q
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	X	No Chg
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	X	No Chg
0	0	1	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDCPE_1 should be placed
-- after architecture statement but before begin keyword

component LDCPE_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        CLR : in STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDCPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDCPE_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDCPE_1 should be placed
-- in architecture after the begin keyword

LDCPE_1_INSTANCE_NAME : LDCPE_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            CLR => user_CLR,
            D => user_D,
            G => user_G,
            GE => user_GE,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
LDCPE_1 LDCPE_1_instance_name (.Q (user_Q),
                               .CLR (user_CLR),
                               .D (user_D),
                               .G (user_G),
                               .GE (user_D),
                               .PRE (user_PRE));

defparam LDCPE_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

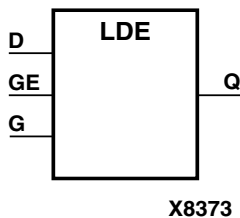
INIT

LDE

Transparent Data Latch with Gate Enable

Architectures Supported

LDE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDE is a transparent data latch with data (D) and gate enable (GE) inputs. Output Q reflects the data (D) while the gate (G) input and gate enable (GE) are High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	1	0	0
1	1	1	1
1	0	X	No Chg
1	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDE should be placed
-- after architecture statement but before begin keyword
```

```
component LDE
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULONGIC;
      D : in STD_ULONGIC;
      G : in STD_ULONGIC;
      GE : in STD_ULONGIC);
end component;

-- Component Attribute specification for LDE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDE should be placed
-- in architecture after the begin keyword

LDE_INSTANCE_NAME : LDE
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          D => user_D,
          G => user_G,
          GE => user_GE);

```

Verilog Instantiation Template

```

LDE LDE_instance_name (.Q (user_Q),
                      .D (user_D),
                      .G (user_G),
                      .GE (user_GE));

defparam LDE_instance_name.INIT = bit_value;

```

Commonly Used Constraints

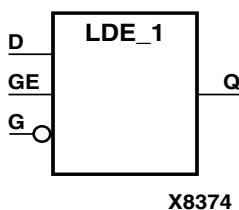
INIT

LDE_1

Transparent Data Latch with Gate Enable and Inverted Gate

Architectures Supported

LDE_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDE_1 is a transparent data latch with data (D), gate enable (GE), and inverted gate (G). Output Q reflects the data (D) while the gate (G) input is Low and gate enable (GE) is High. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G is High or GE is Low.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
GE	G	D	Q
0	X	X	No Chg
1	0	0	0
1	0	1	1
1	1	X	No Chg
1	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDE_1 should be placed
-- after architecture statement but before begin keyword
```

```

component LDE_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDE_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDE_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDE_1 should be placed
-- in architecture after the begin keyword

LDE_1_INSTANCE_NAME : LDE_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            D => user_D,
            G => user_G,
            GE => user_GE);

```

Verilog Instantiation Template

```

LDE_1 LDE_1_instance_name (.Q (user_Q),
                          .D (user_D),
                          .G (user_G),
                          .GE (user_GE));

```

```

defparam LDE_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

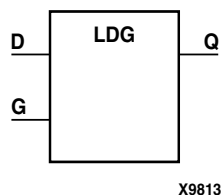
INIT

LDG

Transparent Datagate Latch

Architectures Supported

LDG	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



LDG is a transparent DataGate latch used for gating input signals to decrease power dissipation. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The D input(s) of the LDG must be connected to a device input pad(s) and must have no other fan-outs (must not branch). The CPLD fitter maps the G input to the device's DataGate Enable control pin (DGE). There must be no more than one DataGate Enable signal in the design. The DataGate Enable signal may be driven either by a device input pin or any on-chip logic source. The DataGate Enable signal may be reused by other ordinary logic in the design.

The latch is asynchronously cleared, output Low, when power is applied, or when global reset is active. For CPLDs, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net. See “LDG4, 8, 16” for information on multiple transparent datagate latches for the CoolRunner-II series.

Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	D

Usage

For HDL, these design elements are supported for inference *and* instantiation.

VHDL Instantiation Template

```
component LDG
  port (Q : out STD_ULOGIC;
        D : in  STD_ULOGIC;
```

```
        G : in STD_ULOGIC);
end component;
...
begin
...
INSTANCE_NAME : LDG
    port map (Q => user_Q,
              D => user_D,
              G => user_G);
```

Verilog Instantiation Template

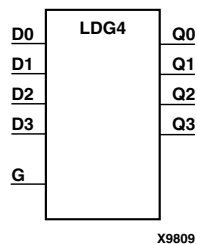
```
LDG instance_name (.Q (user_Q),
                   .D (user_D),
                   .G (user_G));
```

LDG4, 8, 16

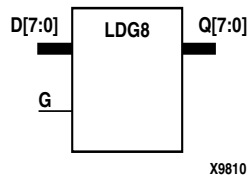
Multiple Transparent Datagate Latches

Architectures Supported

LDG4, LDG8, LDG16	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Macro

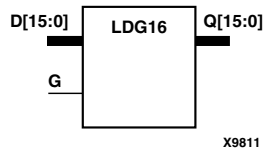


LDG4, LDG8, and LDG16 have, respectively, 4, 8, and 16 transparent DataGate latches with a common gate enable (G). These latches are used to gate input signals in order to decrease power dissipation during periods when activity on the input pins is not of interest to the CPLD. The data output (Q) of the latch reflects the data (D) input while the gate enable (G) input is Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

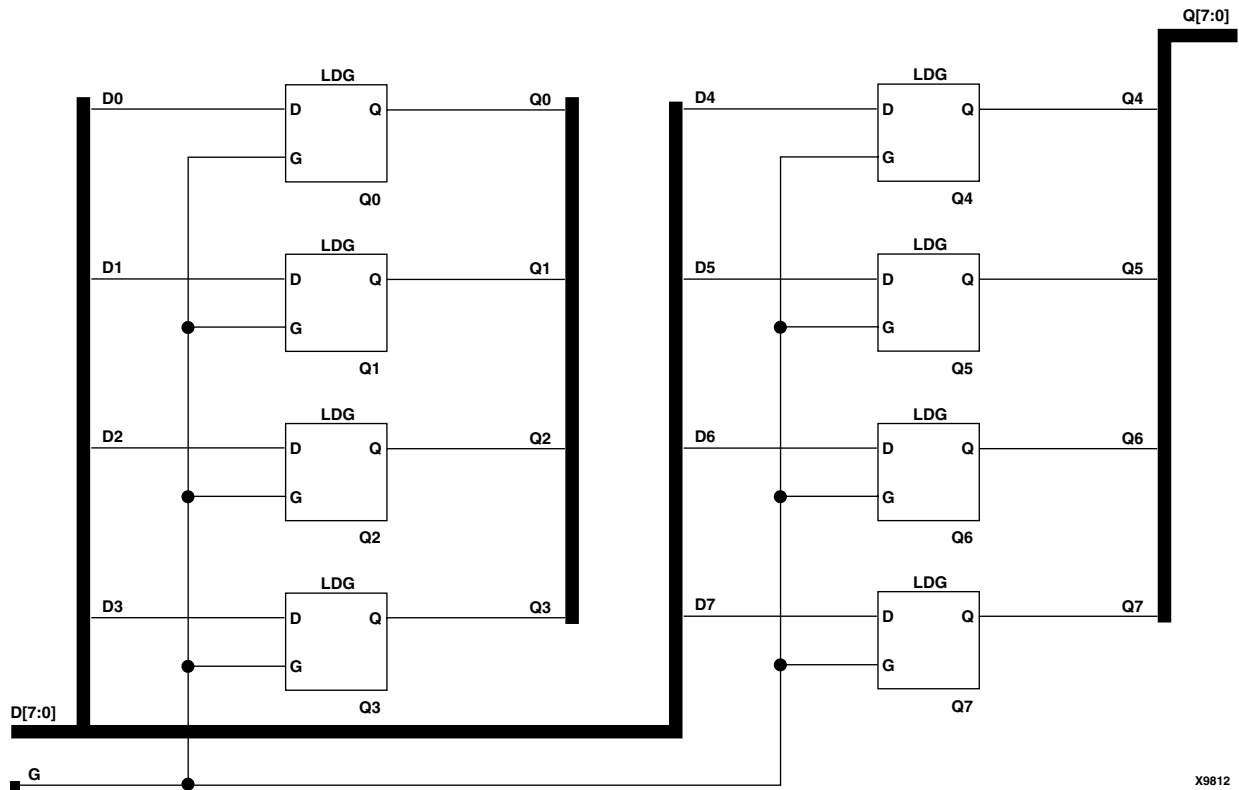


The D input(s) of the LDG must be connected to a device input pad(s) and must have no other fan-outs (must not branch). The CPLD fitter maps the G input to the device's DataGate Enable control pin (DGE). There must be no more than one DataGate Enable signal in the design. The DataGate Enable signal may be driven either by a device input pin or any on-chip logic source. The DataGate Enable signal may be reused by other ordinary logic in the design.

The latch is asynchronously cleared, output Low, when power is applied. See "LDG" for information on single transparent data latches.



Inputs		Outputs
G	D	Q
0	0	0
0	1	1
1	X	No Chg
↑	D	D



X9812

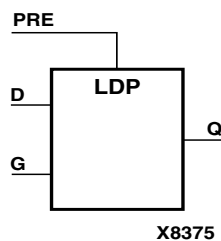
LDG8 Implementation

LDP

Transparent Data Latch with Asynchronous Preset

Architectures Supported

LDP	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Macro
CoolRunner XPLA3	Macro
CoolRunner-II	Macro



LDP is a transparent data latch with asynchronous preset (PRE). When the PRE input is High, it overrides the other inputs and presets the data (Q) output High. Q reflects the data (D) input while gate (G) input is High and PRE is Low. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	1	0	0
0	1	1	1
0	0	X	No Chg
0	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDP should be placed
-- after architecture statement but before begin keyword
```

```
component LDP
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULONGIC;
      D : in STD_ULONGIC;
      G : in STD_ULONGIC;
      PRE : in STD_ULONGIC);
end component;

-- Component Attribute specification for LDP
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDP_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDP should be placed
-- in architecture after the begin keyword

LDP_INSTANCE_NAME : LDP
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          D => user_D,
          G => user_G,
          PRE => user_PRE);

```

Verilog Instantiation Template

```

LDP LDP_instance_name (.Q (user_Q),
                      .D (user_D),
                      .G (user_G),
                      .PRE (user_PRE));

defparam LDP_instance_name.INIT = bit_value;

```

Commonly Used Constraints

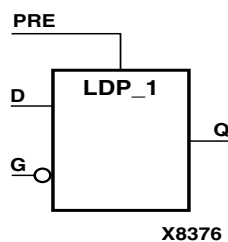
INIT

LDP_1

Transparent Data Latch with Asynchronous Preset and Inverted Gate

Architectures Supported

LDP_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDP_1 is a transparent data latch with asynchronous preset (PRE) and inverted gate (G). When the PRE input is High, it overrides the other inputs and presets the data (Q) output High. Q reflects the data (D) input while gate (G) input and PRE are Low. The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
PRE	G	D	Q
1	X	X	1
0	0	0	0
0	0	1	1
0	1	X	No Chg
0	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDP_1 should be placed
-- after architecture statement but before begin keyword
```

```
component LDP_1
```

```

-- synthesis translate_off
generic (
    INIT : bit := '1');
-- synthesis translate_on
port (Q : out STD_ULOGIC;
      D : in STD_ULOGIC;
      G : in STD_ULOGIC;
      PRE : in STD_ULOGIC);
end component;

-- Component Attribute specification for LDP_1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDP_1_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDP_1 should be placed
-- in architecture after the begin keyword

LDP_1_INSTANCE_NAME : LDP_1
-- synthesis translate_off
generic map (
    INIT => bit_value)
-- synthesis translate_on
port map (Q => user_Q,
          D => user_D,
          G => user_G,
          PRE => user_PRE);

```

Verilog Instantiation Template

```

LDP_1 LDP_1_instance_name (.Q (user_Q),
                          .D (user_D),
                          .G (user_G),
                          .PRE (user_PRE));

defparam LDP_1_instance_name.INIT = bit_value;

```

Commonly Used Constraints

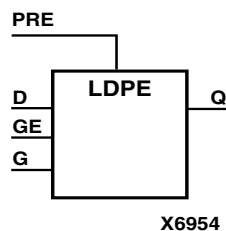
INIT

LDPE

Transparent Data Latch with Asynchronous Preset and Gate Enable

Architectures Supported

LDPE	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDPE is a transparent data latch with asynchronous preset and gate enable. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) input and gate enable (GE) are High. The data on the D input during the High-to-Low gate transition is stored in the latch. The data on the Q output remains unchanged as long as G or GE remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	1	0	0
0	1	1	1	1
0	1	0	X	No Chg
0	1	↓	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for LDPE should be placed
-- after architecture statement but before begin keyword
```

```

component LDPE
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULONGIC;
        D : in STD_ULONGIC;
        G : in STD_ULONGIC;
        GE : in STD_ULONGIC;
        PRE : in STD_ULONGIC);
end component;

-- Component Attribute specification for LDPE
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LDPE_instance_name : label is "0";
-- values can be (0 or 1)

-- Component Instantiation for LDPE should be placed
-- in architecture after the begin keyword

LDPE_INSTANCE_NAME : LDPE
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            D => user_D,
            G => user_G,
            GE => user_GE,
            PRE => user_PRE);

```

Verilog Instantiation Template

```

LDPE LDPE_instance_name (.Q (user_Q),
                        .D (user_D),
                        .G (user_G),
                        .GE (user_GE),
                        .PRE (user_PRE));

defparam LDPE_instance_name.INIT = bit_value;

```

Commonly Used Constraints

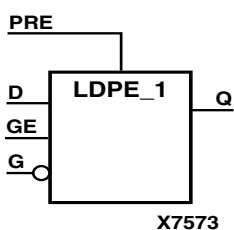
INIT

LDPE_1

Transparent Data Latch with Asynchronous Preset, Gate Enable, and Inverted Gate

Architectures Supported

LDPE_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LDPE_1 is a transparent data latch with asynchronous preset, gate enable, and inverted gate. When the asynchronous preset (PRE) is High, it overrides the other input and presets the data (Q) output High. Q reflects the data (D) input while the gate (G) and PRE are Low and gate enable (GE) is High.

The data on the D input during the Low-to-High gate transition is stored in the latch. The data on the Q output remains unchanged as long as G remains High or GE remains Low.

The latch is asynchronously preset, output High, when power is applied.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs
PRE	GE	G	D	Q
1	X	X	X	1
0	0	X	X	No Chg
0	1	0	0	0
0	1	0	1	1
0	1	1	X	No Chg
0	1	↑	D	D

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

-- Component Declaration for LDPE_1 should be placed
-- after architecture statement but before begin keyword

```
component LDPE_1
  -- synthesis translate_off
  generic (
    INIT : bit := '1');
  -- synthesis translate_on
  port (Q : out STD_ULOGIC;
        D : in STD_ULOGIC;
        G : in STD_ULOGIC;
        GE : in STD_ULOGIC;
        PRE : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for LDPE_1
-- should be placed after architecture declaration but
-- before the begin keyword

```
attribute INIT : string;
attribute INIT of LDPE_1_instance_name : label is "0";
-- values can be (0 or 1)
```

-- Component Instantiation for LDPE_1 should be placed
-- in architecture after the begin keyword

```
LDPE_1_INSTANCE_NAME : LDPE_1
  -- synthesis translate_off
  generic map (
    INIT => bit_value)
  -- synthesis translate_on
  port map (Q => user_Q,
            D => user_D,
            G => user_G,
            GE => user_GE,
            PRE => user_PRE);
```

Verilog Instantiation Template

```
LDPE_1 LDPE_1_instance_name (.Q (user_Q),
                             .D (user_D),
                             .G (user_G),
                             .GE (user_GE),
                             .PRE (user_PRE));
```

```
defparam LDPE_1_instance_name.INIT = bit_value;
```

Commonly Used Constraints

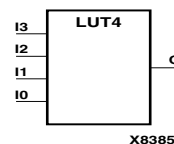
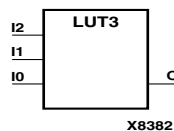
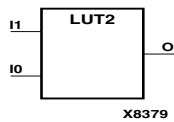
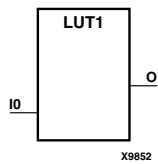
INIT

LUT1, 2, 3, 4

1-, 2-, 3-, 4-Bit Look-Up Table with General Output

Architectures Supported

LUT1, LUT2, LUT3, LUT4	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LUT1, LUT2, LUT3, and LUT4 are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with general output (O).

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1 provides a look-up-table version of a buffer or inverter.

LUTs are the basic Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, Spartan-II, Spartan-III, and Spartan-3 building blocks. Two LUTs are available in each CLB slice; four LUTs are available in each CLB. The variants, “LUT1_D, LUT2_D, LUT3_D, LUT4_D” and “LUT1_L, LUT2_L, LUT3_L, LUT4_L” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

LUT3 Function Table

Inputs			Outputs
I2	I1	I0	O
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1

```
-- Component Declaration for LUT1 should be placed
-- after architecture statement but before begin keyword

component LUT1
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"2");
  -- synthesis translate_on
  port (O : out STD_ULOGIC;
        IO : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT1
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT1_instance_name : label is "2";
-- values can be 0, 1, 2, or 3

-- Component Instantiation for LUT1 should be placed
-- in architecture after the begin keyword

LUT1_INSTANCE_NAME : LUT1
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (O => user_O,
            IO => user_IO);
```

Verilog Instantiation Template For LUT1

```
LUT1 LUT1_instance_name (.O (user_O),
                        .IO (user_IO));

defparam LUT1_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2

```
-- Component Declaration for LUT2 should be placed
-- after architecture statement but before begin keyword

component LUT2
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"4");
  -- synthesis translate_on
  port (O : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT2
-- should be placed after architecture declaration but
-- before the begin keyword
```

```

attribute INIT : string;
attribute INIT of LUT2_instance_name : label is "4";
-- values can be 0, 1, 2,3, or 4

-- Component Instantiation for LUT2 should be placed
-- in architecture after the begin keyword

LUT2_INSTANCE_NAME : LUT2
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (O => user_O,
    IO => user_IO,
    I1 => user_I1);

```

Verilog Instantiation Template For LUT2

```

LUT2 LUT2_instance_name (.O (user_O),
                        .IO (user_IO),
                        .I1 (user_I1));

defparam LUT2_instance_name.INIT = hex_value;

```

VHDL Instantiation Template for LUT3

```

-- Component Declaration for LUT3 should be placed
-- after architecture statement but before begin keyword

component LUT3
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"8");
  -- synthesis translate_on
  port (O : out STD_ULOGIC;
    IO : in STD_ULOGIC;
    I1 : in STD_ULOGIC;
    I2 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_instance_name : label is "8";
-- values can be 0, 1, 2, 3, 4, 5, 6, 7, 8

-- Component Instantiation for LUT3 should be placed
-- in architecture after the begin keyword --

LUT3_INSTANCE_NAME : LUT3
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (O => user_O,

```

```
I0 => user_I0,
I1 => user_I1,
I2 => user_I2);
```

Verilog Instantiation Template For LUT3

```
LUT3 LUT3_instance_name (.O (user_O),
                        .I0 (user_I0),
                        .I1 (user_I1),
                        .I2 (user_I2));
```

```
defparam LUT4_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT4

```
-- Component Declaration for LUT4 should be placed
-- after architecture statement but before begin keyword
```

```
component LUT4
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"16");
  -- synthesis translate_on
  port (O : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC;
        I3 : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for LUT4
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of LUT4_instance_name : label is "16";
-- values can be 0 through 16
```

```
-- Component Instantiation for LUT4 should be placed
-- in architecture after the begin keyword
```

```
LUT4_INSTANCE_NAME : LUT4
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (O => user_O,
            IO => user_I0,
            I1 => user_I1,
            I2 => user_I2,
            I3 => user_I3);
```

Verilog Instantiation Template For LUT4

```
LUT4 LUT4_instance_name (.O (user_O),
                        .I0 (user_I0),
                        .I1 (user_I1),
                        .I2 (user_I2),
```

```
.I3 (user_I3));
```

```
defparam LUT4_instance_name.INIT = hex_value;
```

Commonly Used Constraints

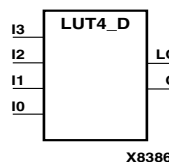
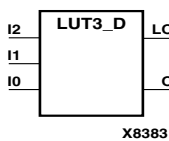
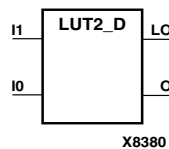
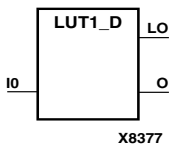
BEL, INIT, LOC, U_SET

LUT1_D, LUT2_D, LUT3_D, LUT4_D

1-, 2-, 3-, 4-Bit Look-Up Table with Dual Output

Architectures Supported

LUT1_D, LUT2_D, LUT3_D, LUT4_D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LUT1_D, LUT2_D, LUT3_D, and LUT4_D are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_D provides a look-up-table version of a buffer or inverter.

See also “LUT1, 2, 3, 4” and “LUT1_L, LUT2_L, LUT3_L, LUT4_L.”

LUT3_D Function Table

Inputs			Outputs	
I2	I1	I0	O	LO
0	0	0	INIT[0]	INIT[0]
0	0	1	INIT[1]	INIT[1]
0	1	0	INIT[2]	INIT[2]
0	1	1	INIT[3]	INIT[3]
1	0	0	INIT[4]	INIT[4]
1	0	1	INIT[5]	INIT[5]
1	1	0	INIT[6]	INIT[6]
1	1	1	INIT[7]	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1_D

```
-- Component Declaration for LUT1_D should be placed
-- after architecture statement but before begin keyword
```

```
component LUT1_D
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"2");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        O  : out STD_ULOGIC;
        IO : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for LUT1_D
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
attribute INIT : string;
attribute INIT of LUT1_D_instance_name : label is "2";
-- values can be 0, 1, 2, or 3
```

```
-- Component Instantiation for LUT1_D should be placed
-- in architecture after the begin keyword
```

```
LUT1_D_INSTANCE_NAME : LUT1_D
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (LO => user_LO,
            O  => user_O,
            IO => user_IO);
```

Verilog Instantiation Template For LUT1_D

```
LUT1_D LUT1_D_instance_name (.LO (user_LO),
                             .O (user_O),
                             .IO (user_IO));
```

```
defparam LUT1_D_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2_D

```
-- Component Declaration for LUT2_D should be placed
-- after architecture statement but before begin keyword
```

```
component LUT2_D
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"4");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        O  : out STD_ULOGIC;
        IO : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC);
```



```

end component;

-- Component Attribute specification for LUT2_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT2_D_instance_name : label is "4";

-- Component Instantiation for LUT2_D should be placed
-- in architecture after the begin keyword

LUT2_D_INSTANCE_NAME : LUT2_D
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (LO => user_LO,
            O => user_O,
            IO => user_IO,
            I1 => user_I1);

```

Verilog Instantiation Template For LUT2_D

```

LUT2_D LUT2_D_instance_name (.LO (user_O),
                             .O (user_O),
                             .IO (user_IO),
                             .I1 (user_I1));

defparam LUT2_D_instance_name.INIT = hex_value;

```

VHDL Instantiation Template for LUT3_D

```

-- Component Declaration for LUT3_D should be placed
-- after architecture statement but before begin keyword

component LUT3_D
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"8");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        O : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_D_instance_name : label is "8";

-- Component Instantiation for LUT3_D should be placed
-- in architecture after the begin keyword

```

```

LUT3_D_INSTANCE_NAME : LUT3_D
-- synthesis translate_off
generic map (
    INIT => hex_value)
-- synthesis translate_on
port map (LO => user_LO,
          O => user_O,
          IO => user_IO,
          I1 => user_I1,
          I2 => user_I2);

```

Verilog Instantiation Template For LUT3_D

```

LUT3_D LUT3_D_instance_name (.LO (user_LO),
                             .O (user_O),
                             .IO (user_IO),
                             .I1 (user_I1),
                             .I2 (user_I2));

defparam LUT3_D_instance_name.INIT = hex_value;

```

VHDL Instantiation Template for LUT4_D

```

-- Component Declaration for LUT4_D should be placed
-- after architecture statement but before begin keyword

```

```

component LUT4_D
-- synthesis translate_off
generic (
    INIT : bit_vector := X"16");
-- synthesis translate_on
port (LO : out STD_ULOGIC;
      O : out STD_ULOGIC;
      IO : in STD_ULOGIC;
      I1 : in STD_ULOGIC;
      I2 : in STD_ULOGIC;
      I3 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT4_D
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT4_D_instance_name : label is "16";
-- values can be 0 through 16

```

```

-- Component Instantiation for LUT4_D should be placed
-- in architecture after the begin keyword

```

```

LUT4_D_INSTANCE_NAME : LUT4_D
-- synthesis translate_off
generic map (
    INIT => hex_value)
-- synthesis translate_on
port map (LO => user_LO,
          O => user_O,
          IO => user_IO,

```

```
I1 => user_I1,  
I2 => user_I2,  
I3 => user_I3);
```

Verilog Instantiation Template For LUT4_D

```
LUT4_D LUT4_D_instance_name (.LO (user_LO),  
                             .O (user_O),  
                             .I0 (user_I0),  
                             .I1 (user_I1),  
                             .I2 (user_I2),  
                             .I3 (user_I3));  
  
defparam LUT4_D_instance_name.INIT = hex_value;
```

Commonly Used Constraints

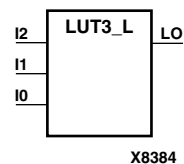
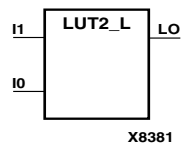
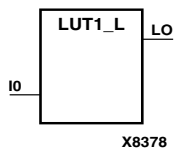
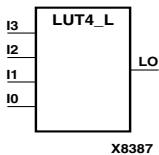
INIT, LOC, RLOC, BEL, U_SET

LUT1_L, LUT2_L, LUT3_L, LUT4_L

1-, 2-, 3-, 4-Bit Look-Up Table with Local Output

Architectures Supported

LUT1_L, LUT2_L, LUT3_L, LUT4_L	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



LUT1_L, LUT2_L, LUT3_L, and LUT4_L are, respectively, 1-, 2-, 3-, and 4-bit look-up-tables (LUTs) with a local output (LO) that is used to connect to another output within the same CLB slice and to the fast connect buffer.

A mandatory INIT attribute, with an appropriate number of hexadecimal digits for the number of inputs, must be attached to the LUT to specify its function.

LUT1_L provides a look-up-table version of a buffer or inverter.

See also “LUT1, 2, 3, 4” and “LUT1_D, LUT2_D, LUT3_D, LUT4_D.”

LUT3_L Function Table

Inputs			Outputs
I2	I1	I0	LO
0	0	0	INIT[0]
0	0	1	INIT[1]
0	1	0	INIT[2]
0	1	1	INIT[3]
1	0	0	INIT[4]
1	0	1	INIT[5]
1	1	0	INIT[6]
1	1	1	INIT[7]

INIT = binary equivalent of the hexadecimal number assigned to the INIT attribute

Usage

LUTs are generally inferred with the logic portions of the HDL code. Xilinx suggests that you instantiate LUTs only if you have a need to implicitly specify the logic mapping, or if you need to manually place or relationally place the logic.

VHDL Instantiation Template for LUT1_L

```
-- Component Declaration for LUT1_L should be placed
-- after architecture statement but before begin keyword

component LUT1_L
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"2");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        IO : inSTD_ULOGIC);
end component;

-- Component Attribute specification for LUT1_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT1_L_instance_name : label is "2";
-- values can be 0, 1, 2, or 3

-- Component Instantiation for LUT1_L should be placed
-- in architecture after the begin keyword

LUT1_L_INSTANCE_NAME : LUT1_L
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (LO => user_LO,
            IO => user_IO);
```

Verilog Instantiation Template For LUT1_L

```
LUT1_L LUT1_L_instance_name (.LO (user_LO),
                             .IO (user_IO));

defparam LUT1_L_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT2_L

```
-- Component Declaration for LUT2_L should be placed
-- after architecture statement but before begin keyword

component LUT2_L
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"4");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT2_L
-- should be placed after architecture declaration but
-- before the begin keyword
```

```

attribute INIT : string;
attribute INIT of LUT2_L_instance_name : label is "4";

-- Component Instantiation for LUT2_L should be placed
-- in architecture after the begin keyword

LUT2_L_INSTANCE_NAME : LUT2_L
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (LO => user_LO,
            IO => user_IO,
            I1 => user_I1);

```

Verilog Instantiation Template For LUT2_L

```

LUT2_L LUT2_L_instance_name (.LO (user_O),
                             .IO (user_IO),
                             .I1 (user_I1));

defparam LUT2_L_instance_name.INIT = hex_value;

```

VHDL Instantiation Template for LUT3_L

```

-- Component Declaration for LUT3_L should be placed
-- after architecture statement but before begin keyword

component LUT3_L
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"8");
  -- synthesis translate_on
  port (LO : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC);
end component;

-- Component Attribute specification for LUT3_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT3_L_instance_name : label is "8";

-- Component Instantiation for LUT3_L should be placed
-- in architecture after the begin keyword --
LUT3_L_INSTANCE_NAME : LUT3_L
  -- synthesis translate_off
  generic map (
    INIT => hex_value)
  -- synthesis translate_on
  port map (LO => user_LO,
            IO => user_IO,
            I1 => user_I1,
            I2 => user_I2);

```

Verilog Instantiation Template For LUT3_L

```
LUT3_L LUT3_L_instance_name (.LO (user_LO),
                             .IO (user_IO),
                             .I1 (user_I1),
                             .I2 (user_I2));

defparam LUT3_L_instance_name.INIT = hex_value;
```

VHDL Instantiation Template for LUT4_L

```
-- Component Declaration for LUT4_L should be placed
-- after architecture statement but before begin keyword --
```

```
component LUT4_L
-- synthesis translate_off
  generic (
    INIT : bit_vector := X"16");
-- synthesis translate_on
  port (LO : out STD_ULOGIC;
        IO : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        I2 : in  STD_ULOGIC;
        I3 : in  STD_ULOGIC);
end component;

-- Component Attribute specification for LUT4_L
-- should be placed after architecture declaration but
-- before the begin keyword

attribute INIT : string;
attribute INIT of LUT4_L_instance_name : label is "16";
-- values can be 0 through 16

-- Component Instantiation for LUT4_L should be placed
-- in architecture after the begin keyword --

LUT4_L_INSTANCE_NAME : LUT4_L
-- synthesis translate_off
  generic map (
    INIT => hex_value)
-- synthesis translate_on
  port map (LO => user_LO,
            IO => user_IO,
            I1 => user_I1,
            I2 => user_I2,
            I3 => user_I3);
```

Verilog Instantiation Template For LUT4_L

```
LUT4_L LUT4_L_instance_name (.LO (user_LO),
                             .IO (user_IO),
                             .I1 (user_I1),
                             .I2 (user_I2),
                             .I3 (user_I3));

defparam LUT4_L_instance_name.INIT = hex_value;
```


Commonly Used Constraints

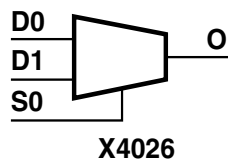
INIT, LOC, RLOC, BEL, U_SET

M2_1

2-to-1 Multiplexer

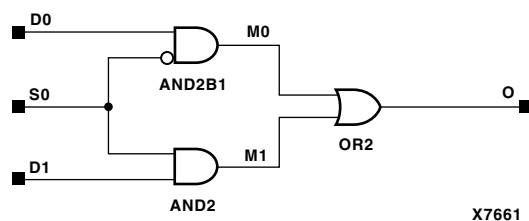
Architectures Supported

M2_1	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



The M2_1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of the select input (S0). The output (O) reflects the state of the selected data input. When Low, S0 selects D0 and when High, S0 selects D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	1
0	X	0	0



M2_1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-3, Spartan-III, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

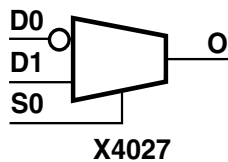
For HDL, this design element is inferred rather than instantiated.

M2_1B1

2-to-1 Multiplexer with D0 Inverted

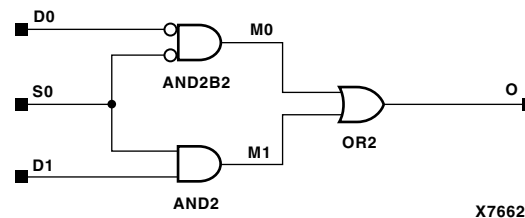
Architectures Supported

M2_1B1	
Spartan-II, Spartan-IIIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



The M2_1B1 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the inverted value of D0. When S0 is High, O reflects the state of D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	1
1	0	X	0
0	X	1	0
0	X	0	1



M2_1B1 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-3, Spartan-IIIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

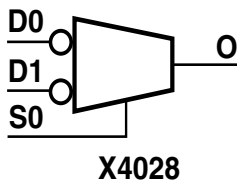
For HDL, this design element is inferred rather than instantiated.

M2_1B2

2-to-1 Multiplexer with D0 and D1 Inverted

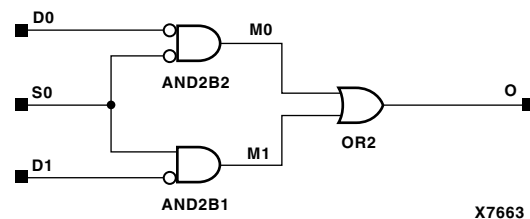
Architectures Supported

M2_1B2	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



The M2_1B2 multiplexer chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When S0 is Low, the output (O) reflects the inverted value of D0. When S0 is High, O reflects the inverted value of D1.

Inputs			Outputs
S0	D1	D0	O
1	1	X	0
1	0	X	1
0	X	1	0
0	X	0	1



M2_1B2 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

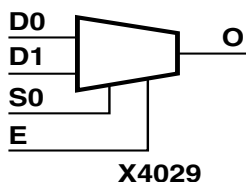
For HDL, this design element is inferred rather than instantiated.

M2_1E

2-to-1 Multiplexer with Enable

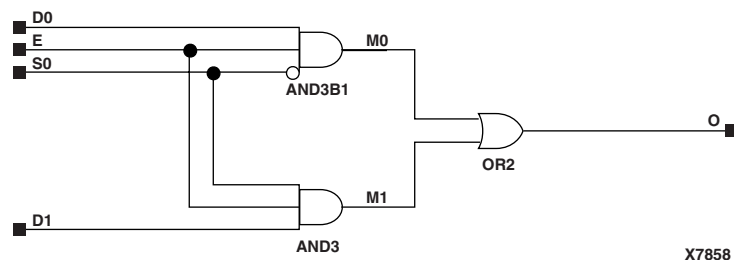
Architectures Supported

M2_1E	
Spartan-II, Spartan-IIIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



M2_1E is a 2-to-1 multiplexer with enable. When the enable input (E) is High, the M2_1E chooses one data bit from two sources (D1 or D0) under the control of select input (S0). When Low, S0 selects D0 and when High, S0 selects D1. When E is Low, the output is Low.

Inputs				Outputs
E	S0	D1	D0	O
0	X	X	X	0
1	0	X	1	1
1	0	X	0	0
1	1	1	X	1
1	1	0	X	0



M2_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro X

Usage

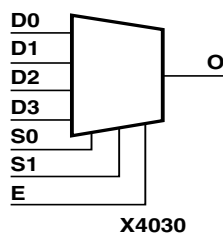
For HDL, this design element is inferred rather than instantiated.

M4_1E

4-to-1 Multiplexer with Enable

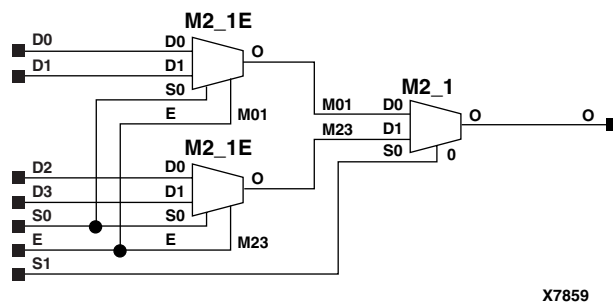
Architectures Supported

M4_1E	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

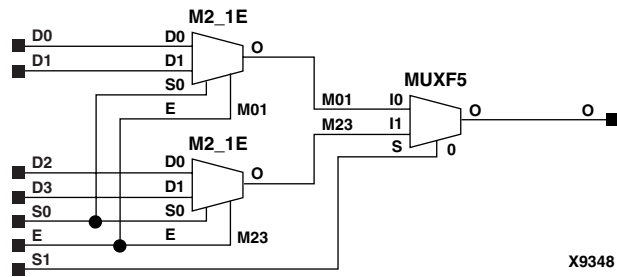


M4_1E is an 4-to-1 multiplexer with enable. When the enable input (E) is High, the M4_1E multiplexer chooses one data bit from four sources (D3, D2, D1, or D0) under the control of the select inputs (S1 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs							Outputs
E	S1	S0	D0	D1	D2	D3	O
0	X	X	X	X	X	X	0
1	0	0	D0	X	X	X	D0
1	0	1	X	D1	X	X	D1
1	1	0	X	X	D2	X	D2
1	1	1	X	X	X	D3	D3



M4_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



M4_1E Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

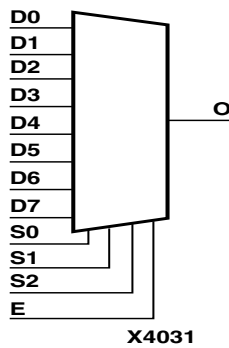
For HDL, this design element is inferred rather than instantiated.

M8_1E

8-to-1 Multiplexer with Enable

Architectures Supported

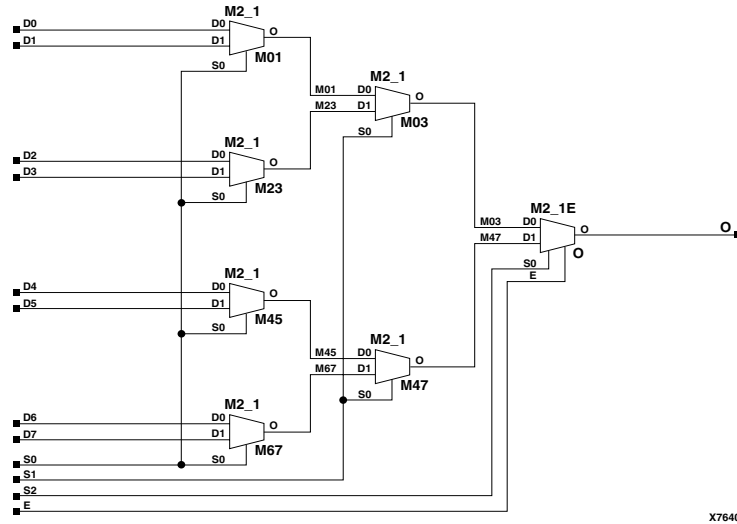
M8_1E	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



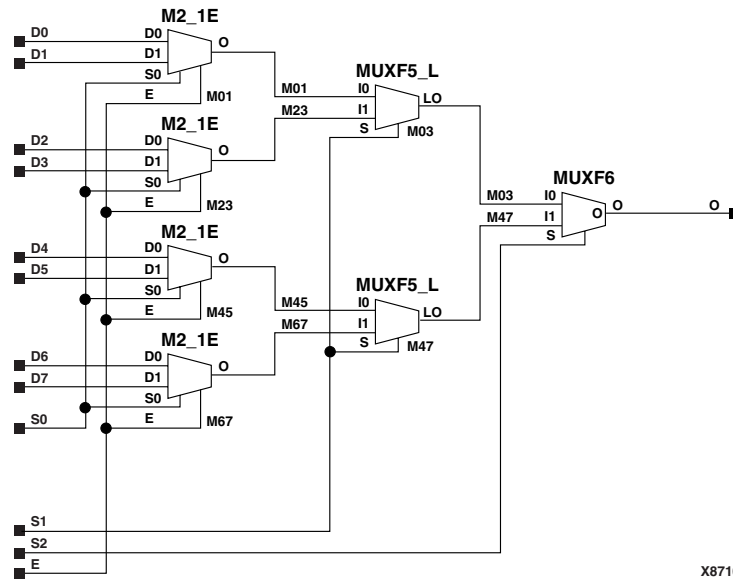
M8_1E is an 8-to-1 multiplexer with enable. When the enable input (E) is High, the M8_1E multiplexer chooses one data bit from eight sources (D7 – D0) under the control of the select inputs (S2 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs					Outputs
E	S2	S1	S0	D7 – D0	O
0	X	X	X	X	0
1	0	0	0	D0	D0
1	0	0	1	D1	D1
1	0	1	0	D2	D2
1	0	1	1	D3	D3
1	1	0	0	D4	D4
1	1	0	1	D5	D5
1	1	1	0	D6	D6
1	1	1	1	D7	D7

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).



M8_1E Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



M8_1E Implementation Spartan-II, Spartan-II E, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

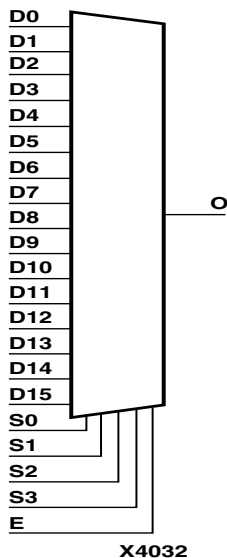
For HDL, this design element is inferred rather than instantiated.

M16_1E

16-to-1 Multiplexer with Enable

Architectures Supported

M16_1E	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



M16_1E is a 16-to-1 multiplexer with enable. When the enable input (E) is High, the M16_1E multiplexer chooses one data bit from 16 sources (D15 – D0) under the control of the select inputs (S3 – S0). The output (O) reflects the state of the selected input as shown in the truth table. When E is Low, the output is Low.

Inputs						Outputs
E	S3	S2	S1	S0	D15 – D0	O
0	X	X	X	X	X	0
1	0	0	0	0	D0	D0
1	0	0	0	1	D1	D1
1	0	0	1	0	D2	D2
1	0	0	1	1	D3	D3
.
.
.
1	1	1	0	0	D12	D12
1	1	1	0	1	D13	D13
1	1	1	1	0	D14	D14
1	1	1	1	1	D15	D15

Dn represents signal on the Dn input; all other data inputs are don't-cares (X).

Usage

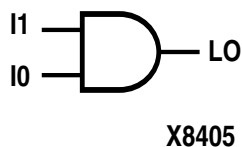
For HDL, this design element is inferred rather than instantiated.

MULT_AND

Fast Multiplier AND

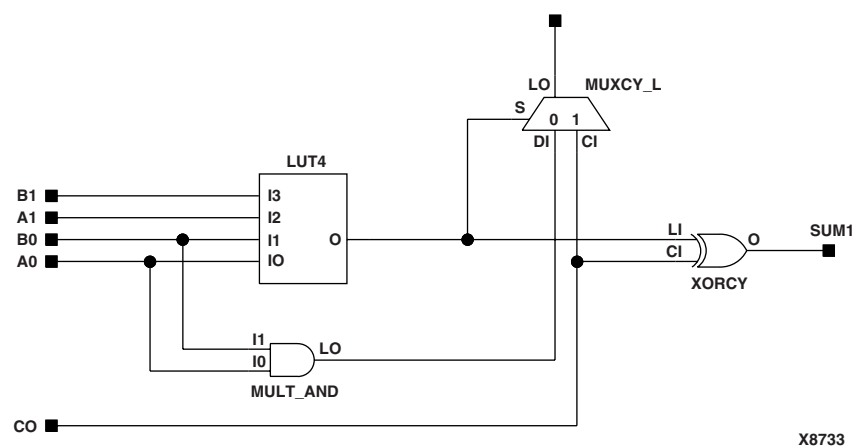
Architectures Supported

MULT_AND	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MULT_AND is an AND component used exclusively for building fast and smaller multipliers. The I1 and I0 inputs *must* be connected to the I1 and I0 inputs of the associated LUT. The LO output *must* be connected to the DI input of the associated MUXCY, MUXCY_D, or MUXCY_L.

Inputs		Output
I1	I0	LO
0	0	0
0	1	0
1	0	0
1	1	1



Example Multiplier Using MULT_AND

Usage

For HDL, this design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for MULT_AND should be placed
-- after architecture statement but before begin keyword

component MULT_AND
  port (LO : out STD_ULOGIC;
        IO : in STD_ULOGIC;
        I1 : in STD_ULOGIC);
end component;

-- Component Attribute specification for MULT_AND
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MULT_AND should be placed
-- in architecture after the begin keyword

MULT_AND_INSTANCE_NAME : MULT_AND
  port map (LO => user_LO,
           IO => user_IO,
           I1 => user_I1);
```

Verilog Instantiation Template

```
MULT_AND MULT_AND_instance_name (.LO (user_LO),
                                  .IO (user_IO),
                                  .I1 (user_I1));
```

Commonly Used Constraints

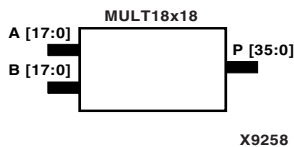
U_SET, MULT_STYLE

MULT18X18

18 x 18 Signed Multiplier

Architectures Supported

MULT18, MULT18X	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MULT18X18 is a combinational signed 18-bit by 18-bit multiplier. The value represented in the 18-bit input A is multiplied by the value represented in the 18-bit input B. Output P is the 36-bit product of A and B.

A, B, and P are two 's complement.

Inputs		Output
A	B	P
A	B	A * B

XST, Synplify, Exemplar and Synopsys all have the ability to infer the MULT18X18.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- MULT18X18: 18 x 18 signed asynchronous multiplier
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

MULT18X18_inst : MULT18X18
port map (
    P => P,      -- 36-bit multiplier output
    A => A,      -- 18-bit multiplier input
    B => B       -- 18-bit multiplier input
);

-- End of MULT18X18_inst instantiation
```

Verilog Instantiation Template

```
// MULT18X18: 18 x 18 signed asynchronous multiplier
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide
```

```
MULT18X18 MULT18X18_inst (  
  .P(P),      // 36-bit multiplier output  
  .A(A),      // 18-bit multiplier input  
  .B(B)       // 18-bit multiplier input  
);  
  
// End of MULT18X18_inst instantiation
```

Commonly Used Constraints

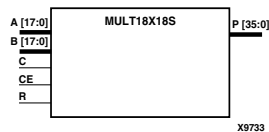
MULT_STYLE

MULT18X18S

18 x 18 Signed Multiplier -- Registered Version

Architectures Supported

MULT18X, MULT18S	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MULT18X18S is the registered version of the 18 x 18 signed multiplier with output P and inputs A, B, C, CE, and R. The registers are initialized to 0 after the GSR pulse.

The value represented in the 18-bit input A is multiplied by the value represented in the 18-bit input B. Output P is the 36-bit product of A and B.

A, B, and P are two 's complement.

Inputs					Output
C	CE	Am	Bn	R	P
↑	X	X	X	1	0
↑	1	Am	Bn	0	A * B
X	0	X	X	0	No Chg

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- MULT18X18S: 18 x 18 signed synchronous multiplier
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide
```

```
MULT18X18S_inst : MULT18X18S
port map (
    P => P,      -- 36-bit multiplier output
    A => A,      -- 18-bit multiplier input
    B => B,      -- 18-bit multiplier input
    C => C,      -- Clock input
    CE => CE,    -- Clock enable input
    R => R       -- Synchronous reset input
);
```

```
-- End of MULT18X18S_inst instantiation
```

Verilog Instantiation Template

```
// MULT18X18S: 18 x 18 signed synchronous multiplier
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

MULT18X18S MULT18X18S_inst (
    .P(P),      // 36-bit multiplier output
    .A(A),      // 18-bit multiplier input
    .B(B),      // 18-bit multiplier input
    .C(C),      // Clock input
    .CE(CE),    // Clock enable input
    .R(R)       // Synchronous reset input
);

// End of MULT18X18S_inst instantiation
```

Commonly Used Constraints

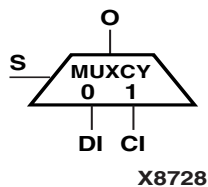
HU_SET, MULT_STYLE

MUXCY

2-to-1 Multiplexer for Carry Logic with General Output

Architectures Supported

MUXCY	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXCY is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of:

- 2-bits per CLB for Virtex, Virtex-E, Spartan-II, and Spartan-III
- 4-bits per configurable logic block (CLB) for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X

The direct input (DI) of an LC is connected to the DI input of the MUXCY. The carry in (CI) input of an LC is connected to the CI input of the MUXCY. The select input (S) of the MUXCY is driven by the output of the lookup table (LUT) and configured as a MUX function. The carry out (O) of the MUXCY reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

The variants, “MUXCY_D” and “MUXCY_L” provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	DI	CI	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY should be placed
-- after architecture statement but before begin keyword

component MUXCY
  port (O : out STD_ULOGIC;
        CI : in STD_ULOGIC;
        DI : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;

-- Component Attribute specification for MUXCY
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY should be placed
-- in architecture after the begin keyword

MUXCY_INSTANCE_NAME : MUXCY
  port map (O => user_O,
           CI => user_CI,
           DI => user_DI,
           S => user_S);
```

Verilog Instantiation Template

```
MUXCY MUXCY_instance_name (.O (user_O),
                           .CI (user_CI),
                           .DI (user_DI),
                           .S (user_S));
```

Commonly Used Constraints

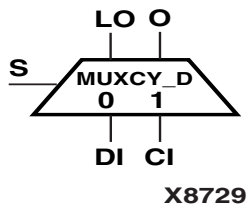
U_SET

MUXCY_D

2-to-1 Multiplexer for Carry Logic with Dual Output

Architectures Supported

MUXCY_D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXCY_D is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_D. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_D. The select input (S) of the MUX is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (O and LO) of the MUXCY_D reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

Outputs O and LO are functionally identical. The O output is a general interconnect.

See also [“MUXCY”](#) and [“MUXCY_L”](#)

Inputs			Outputs	
S	DI	CI	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY_D should be placed
-- after architecture statement but before begin keyword

component MUXCY_D
  port (LO : out STD_ULOGIC;
        O  : out STD_ULOGIC;
        CI : in  STD_ULOGIC;
        DI : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;

-- Component Attribute specification for MUXCY_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY_D should be placed
-- in architecture after the begin keyword

MUXCY_D_INSTANCE_NAME : MUXCY_D
  port map (LO => user_LO,
           O  => user_O,
           CI => user_CI,
           DI => user_DI,
           S  => user_S);
```

Verilog Instantiation Template

```
MUXCY_D MUXCY_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .CI (user_CI),
                               .DI (user_DI),
                               .S (user_S));
```

Commonly Used Constraints

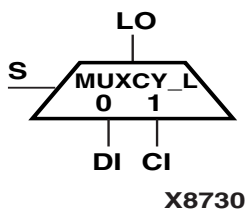
U_SET

MUXCY_L

2-to-1 Multiplexer for Carry Logic with Local Output

Architectures Supported

MUXCY_L	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXCY_L is used to implement a 1-bit high-speed carry propagate function. One such function can be implemented per logic cell (LC), for a total of 4-bits per configurable logic block (CLB). The direct input (DI) of an LC is connected to the DI input of the MUXCY_L. The carry in (CI) input of an LC is connected to the CI input of the MUXCY_L. The select input (S) of the MUXCY_L is driven by the output of the lookup table (LUT) and configured as an XOR function. The carry out (LO) of the MUXCY_L reflects the state of the selected input and implements the carry out function of each LC. When Low, S selects DI; when High, S selects CI.

See also “MUXCY” and “MUXCY_D”

Inputs			Outputs
S	DI	CI	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXCY_L should be placed
-- after architecture statement but before begin keyword
```

```
component MUXCY_L
  port (LO : out STD_ULOGIC;
        CI : in  STD_ULOGIC;
        DI : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXCY_L
```

```
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXCY_L should be placed
-- in architecture after the begin keyword

MUXCY_L_INSTANCE_NAME : MUXCY_L
    port map (LO => user_O,
              CI => user_CI,
              DI => user_DI,
              S => user_S);
```

Verilog Instantiation Template

```
MUXCY_L MUXCY_L_instance_name (.LO (user_LO),
                                .CI (user_CI),
                                .DI (user_DI),
                                .S (user_S));
```

Commonly Used Constraints

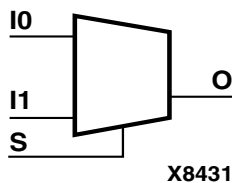
U_SET

MUXF5

2-to-1 Lookup Table Multiplexer with General Output

Architectures Supported

MUXF5	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF5 provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF5_D" and "MUXF5_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5 should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF5
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF5
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF5 should be placed
-- in architecture after the begin keyword

MUXF5_INSTANCE_NAME : MUXF5
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

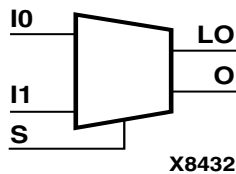
```
MUXF5 MUXF5_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF5_D

2-to-1 Lookup Table Multiplexer with Dual Output

Architectures Supported

MUXF5_D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF5_D provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output connects to other inputs in the same CLB slice.

See also “MUXF5” and “MUXF5_L”

Inputs			Outputs	
S	I0	I1	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5_D should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF5_D
  port (LO : out STD_ULOGIC;
        O  : out STD_ULOGIC;
        I0 : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF5_D
```

```
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF5_D should be placed
-- in architecture after the begin keyword

MUXF5_D_INSTANCE_NAME : MUXF5_D
    port map (LO => user_LO,
              O  => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S  => user_S);
```

Verilog Instantiation Template

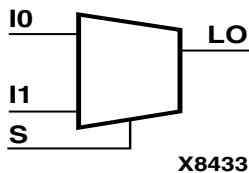
```
MUXF5_D MUXF5_D_instance_name (.LO (user_LO),
                                .O (user_O),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```


MUXF5_L

2-to-1 Lookup Table Multiplexer with Local Output

Architectures Supported

MUXF5_L	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF5_L provides a multiplexer function in a CLB slice for creating a function-of-5 lookup table or a 4-to-1 multiplexer in combination with the associated lookup tables. The local outputs (LO) from the two lookup tables are connected to the I0 and I1 inputs of the MUXF5. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output connects to other inputs in the same CLB slice.

See also “MUXF5” and “MUXF5_D”.

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF5_L should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF5_L
  port (LO : out STD_ULOGIC;
        I0 : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF5_L
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Attributes should be placed here

-- Component Instantiation for MUXF5_L should be placed
-- in architecture after the begin keyword

MUXF5_L_INSTANCE_NAME : MUXF5_L
    port map (LO => user_LO,
              I0 => user_I0,
              I1 => user_I1,
              S  => user_S);
```

Verilog Instantiation Template

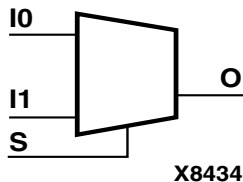
```
MUXF5_L MUXF5_L_instance_name (.LO (user_LO),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```

MUXF6

2-to-1 Lookup Table Multiplexer with General Output

Architectures Supported

MUXF6	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF6 provides a multiplexer function in a full Virtex, Virtex-E, Spartan-II, or Spartan-III CLB, or one half of a Spartan-3, Virtex-II, Virtex-II Pro, or Virtex-II Pro X CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, “MUXF6_D” and “MUXF6_L”, provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6 should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF6
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF6
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF6 should be placed
-- in architecture after the begin keyword

MUXF6_INSTANCE_NAME : MUXF6
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

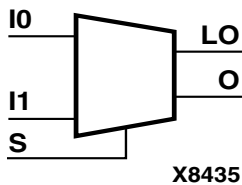
```
MUXF6 MUXF6_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF6_D

2-to-1 Lookup Table Multiplexer with Dual Output

Architectures Supported

MUXF6_D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF6_D provides a multiplexer function in a full Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, Spartan-II, Spartan-III, or Spartan-3 CLB, or one half of a Spartan-3, Virtex-II, Virtex-II Pro, or Virtex-II Pro X CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output connects to other inputs in the same CLB slice.

See also “MUXF6” and “MUXF6_L”

Inputs			Outputs	
S	I0	I1	O	LO
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6_D should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF6_D
  port (LO : out STD_ULOGIC;
        O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF6_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF6_D should be placed
-- in architecture after the begin keyword

MUXF6_D_INSTANCE_NAME : MUXF6_D
    port map (LO => user_LO,
              O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

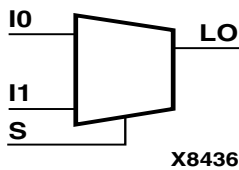
```
MUXF6_D MUXF6_D_instance_name (.LO (user_LO),
                                .O (user_O),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```

MUXF6_L

2-to-1 Lookup Table Multiplexer with Local Output

Architectures Supported

MUXF6_L	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF6_L provides a multiplexer function in a full Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, Spartan-II, Spartan-II E, or Spartan-3 CLB, or one half of a Spartan-3, Virtex-II, Virtex-II Pro, or Virtex-II Pro X CLB (two slices) for creating a function-of-6 lookup table or an 8-to-1 multiplexer in combination with the associated four lookup tables and two MUXF5s. The local outputs (LO) from the two MUXF5s in the CLB are connected to the I0 and I1 inputs of the MUXF6. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output connects to other inputs in the same CLB slice.

See also “MUXF6” and “MUXF6_D”.

Inputs			Output
S	I0	I1	LO
0	1	X	1
0	0	X	0
1	X	1	1
1	X	0	0

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF6_L should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF6_L
  port (LO : out STD_ULOGIC;
        I0 : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF6_L
```

```
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF6_L should be placed
-- in architecture after the begin keyword

MUXF6_L_INSTANCE_NAME : MUXF6_L
    port map (LO => user_LO,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

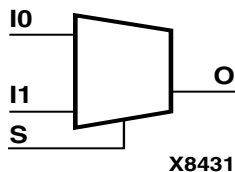
```
MUXF6_L MUXF6_L_instance_name (.LO (user_LO),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```


MUXF7

2-to-1 Lookup Table Multiplexer with General Output

Architectures Supported

MUXF7	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF7 provides a multiplexer function in a full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The variants, "MUXF7_D" and "MUXF7_L", provide additional types of outputs that can be used by different timing models for more accurate pre-layout timing estimation.

Inputs			Outputs
S	I0	I1	O
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7 should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF7
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF7
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF7 should be placed
-- in architecture after the begin keyword

MUXF7_INSTANCE_NAME : MUXF7
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

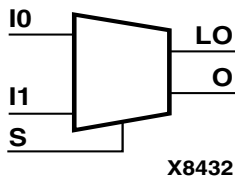
```
MUXF7 MUXF7_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .S (user_S));
```

MUXF7_D

2-to-1 Lookup Table Multiplexer with Dual Output

Architectures Supported

MUXF7_D	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF7_D provides a multiplexer function in one full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output connects to other inputs in the same CLB slice.

See also “MUXF7” and “MUXF7_L”.

Inputs			Outputs	
S	I0	I1	O	LO
0	I0	X	I0	I0
1	X	I1	I1	I1
X	0	0	0	0
X	1	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7_D should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF7_D
  port (LO : out STD_ULOGIC;
        O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF7_D
```

```
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF7_D should be placed
-- in architecture after the begin keyword

MUXF7_D_INSTANCE_NAME : MUXF7_D
    port map (LO => user_LO,
              O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

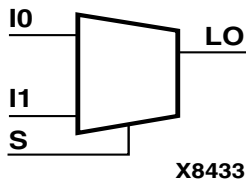
```
MUXF7_D MUXF7_D_instance_name (.LO (user_LO),
                                .O (user_O),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```

MUXF7_L

2-to-1 Lookup Table Multiplexer with Local Output

Architectures Supported

MUXF7_L	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF7_L provides a multiplexer function in a full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output connects to other inputs in the same CLB slice.

See also “MUXF7” and “MUXF7_D”.

Inputs			Output
S	I0	I1	LO
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF7_L should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF7_L
  port (LO : out STD_ULOGIC;
        I0 : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF7_L
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Attributes should be placed here

-- Component Instantiation for MUXF7_L should be placed
-- in architecture after the begin keyword
```

```
MUXF7_L_INSTANCE_NAME : MUXF7_L
    port map (LO => user_LO,
              I0 => user_I0,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

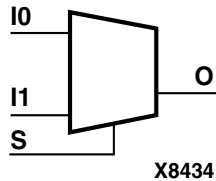
```
MUXF7_L MUXF7_L_instance_name (.LO (user_LO),
                                .I0 (user_I0),
                                .I1 (user_I1),
                                .S (user_S));
```

MUXF8

2-to-1 Lookup Table Multiplexer with General Output

Architectures Supported

MUXF8	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No



MUXF8 provides a multiplexer function in two full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated lookup tables, MUXF5s, MUXF6s, and MUXF7s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Inputs			Outputs
S	I0	I1	O
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8 should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF8
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF8
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
-- Attributes should be placed here
```

```
-- Component Instantiation for MUXF8 should be placed  
-- in architecture after the begin keyword
```

```
MUXF8_INSTANCE_NAME : MUXF8  
    port map (O => user_O,  
              I0 => user_I0,  
              I1 => user_I1,  
              S => user_S);
```

Verilog Instantiation Template

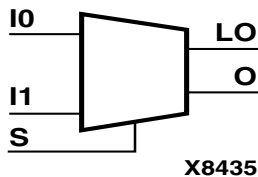
```
MUXF8 MUXF8_instance_name (.O (user_O),  
                           .I0 (user_I0),  
                           .I1 (user_I1),  
                           .S (user_S));
```


MUXF8_D

2-to-1 Lookup Table Multiplexer with Dual Output

Architectures Supported

MUXF8_D	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF8_D provides a multiplexer function in two full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated four lookup tables and two MUXF8s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

Outputs O and LO are functionally identical. The O output is a general interconnect. The LO output connects to other inputs in the same CLB slice.

See also “MUXF8” and “MUXF8_L”.

Inputs			Outputs	
S	I0	I1	O	LO
0	I0	X	I0	I0
1	X	I1	I1	I1
X	0	0	0	0
X	1	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8_D should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF8_D
  port (LO : out STD_ULOGIC;
        O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        S : in STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF8_D
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF8_D should be placed
-- in architecture after the begin keyword

MUXF8_D_INSTANCE_NAME : MUXF8_D
    port map (LO => user_LO,
              O => user_O,
              IO => user_IO,
              I1 => user_I1,
              S => user_S);
```

Verilog Instantiation Template

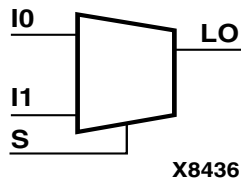
```
MUXF8_D MUXF8_D_instance_name (.LO (user_LO),
                               .O (user_O),
                               .IO (user_IO),
                               .I1 (user_I1),
                               .S (user_S));
```

MUXF8_L

2-to-1 Lookup Table Multiplexer with Local Output

Architectures Supported

MUXF8_L	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



MUXF8_L provides a multiplexer function in two full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLBs for creating a function-of-8 lookup table or a 32-to-1 multiplexer in combination with the associated four lookup tables and two MUXF8s. Local outputs (LO) of MUXF7 are connected to the I0 and I1 inputs of the MUXF8. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1.

The LO output connects to other inputs in the same CLB slice.

See also “MUXF8” and “MUXF8_D”.

Inputs			Output
S	I0	I1	LO
0	I0	X	I0
1	X	I1	I1
X	0	0	0
X	1	1	1

Usage

For HDL, this design element can only be instantiated.

VHDL Instantiation Template

```
-- Component Declaration for MUXF8_L should be placed
-- after architecture statement but before begin keyword
```

```
component MUXF8_L
  port (LO : out STD_ULOGIC;
        I0 : in  STD_ULOGIC;
        I1 : in  STD_ULOGIC;
        S  : in  STD_ULOGIC);
end component;
```

```
-- Component Attribute specification for MUXF8_L
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for MUXF8_L should be placed
-- in architecture after the begin keyword

MUXF8_L_INSTANCE_NAME : MUXF8_L
    port map (LO => user_LO,
             I0 => user_I0,
             I1 => user_I1,
             S => user_S);
```

Verilog Instantiation Template

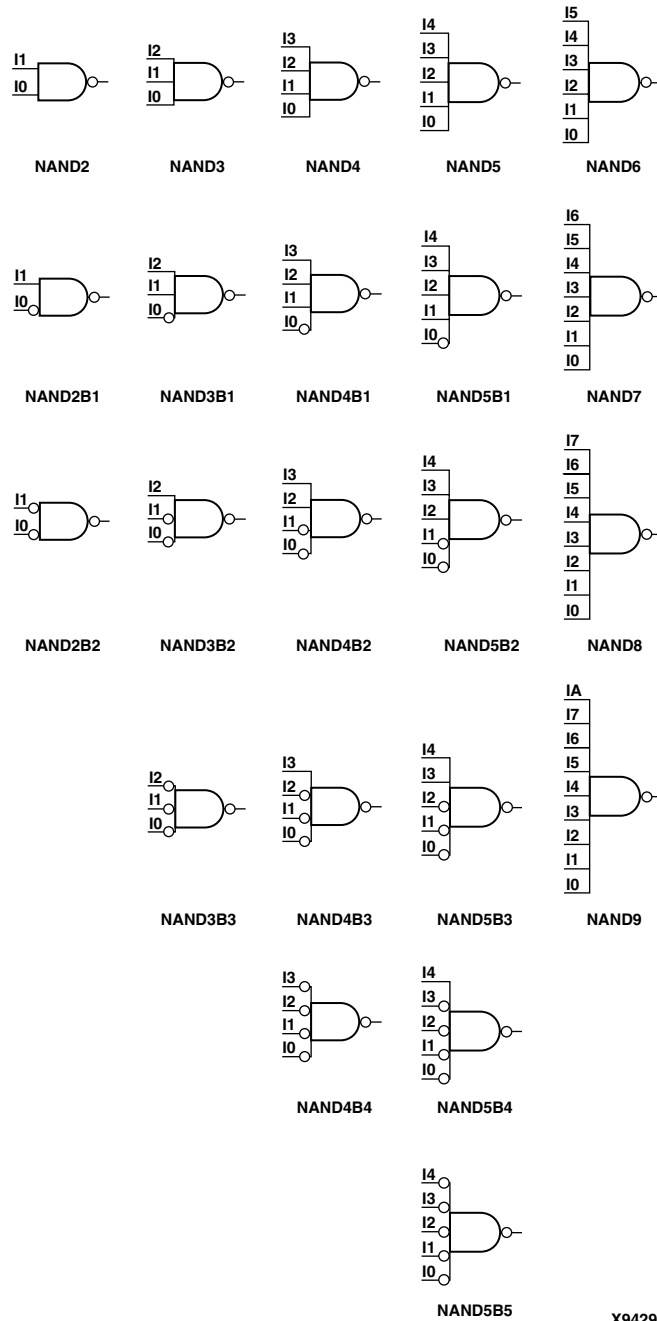
```
MUXF8_L MUXF8_L_instance_name (.LO (user_LO),
                               .I0 (user_I0),
                               .I1 (user_I1),
                               .S (user_S));
```

NAND2-9

2- to 9-Input NAND Gates with Inverted and Non-Inverted Inputs

Architectures Supported

NAND2, NAND3, NAND4, NAND5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
NAND2B1, NAND2B2, NAND3B1, NAND3B2, NAND3B3, NAND4B1, NAND4B2, NAND4B3, NAND4B4, NAND5B1, NAND5B2, NAND5B3, NAND5B4, NAND5B5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
NAND6, NAND7, NAND8, NAND9	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

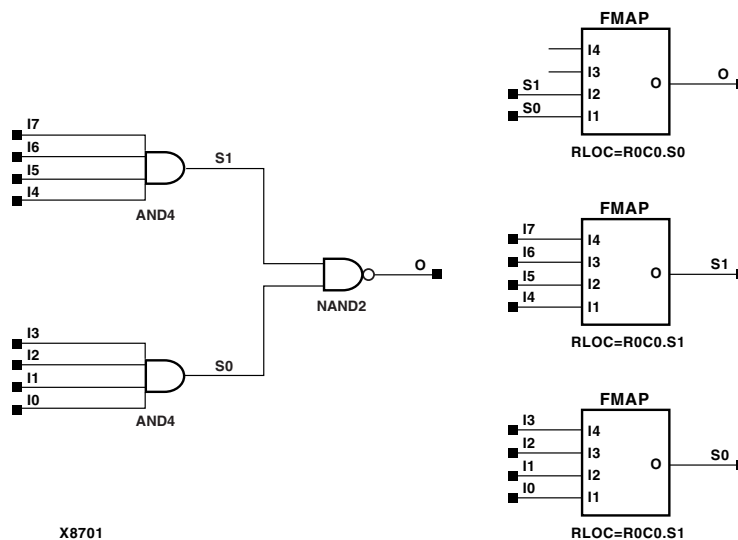


X9429

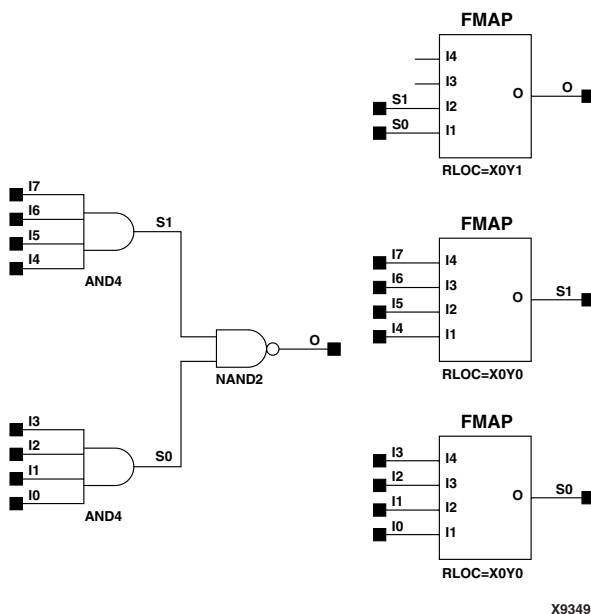
NAND Gate Representations

NAND gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NAND gates of six to nine inputs are available with only non-inverting inputs. To invert inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

See “NAND12, 16” for information on additional NAND functions.



NAND8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

NAND2 through NAND5 are primitives that can be inferred or instantiated. NAND6 through NAND9 are macros which can be inferred.

VHDL Instantiation Template for NAND5

Following is the VHDL code for NAND5. To instantiate NAND2, remove I2, I3, and I4. To instantiate NAND3, remove I3 and I4, For NAND4, remove I4. NAND2B1, and NAND2B2 have the same code as NAND2. NAND3B1, 3B2, and 3B3 have the same code as NAND3 etc..

```
-- Component Declaration for NAND5 should be placed
-- after architecture statement but before begin keyword

component NAND5
    port (O : out STD_ULONGIC;
          I0 : in STD_ULONGIC;
          I1 : in STD_ULONGIC;
          I2 : in STD_ULONGIC;
          I3 : in STD_ULONGIC;
          I4 : in STD_ULONGIC);
end component;

-- Component Attribute specification for NAND5
-- should be placed after architecture declaration but
-- before the begin keyword
-- Attributes should be placed here

-- Component Instantiation for NAND5 should be placed
-- in architecture after the begin keyword

NAND5_INSTANCE_NAME : NAND5
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2,
              I3 => user_I3,
              I4 => user_I4);
```

Verilog Instantiation Template for NAND5

```
NAND5 NAND5_instance_name (.O (user_O),
                            .I0 (user_I0),
                            .I1 (user_I1),
                            .I2 (user_I2),
                            .I3 (user_I3),
                            .I4 (user_I4));
```


NAND12, 16

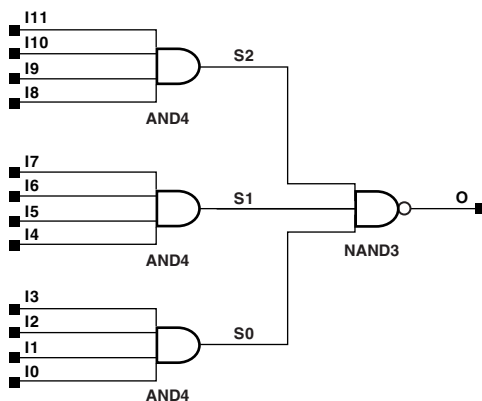
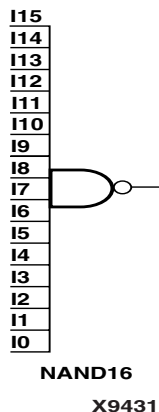
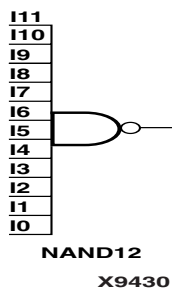
12- and 16-Input NAND Gates with Non-Inverted Inputs

Architectures Supported

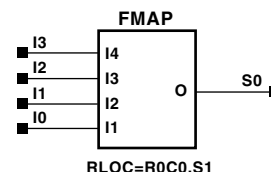
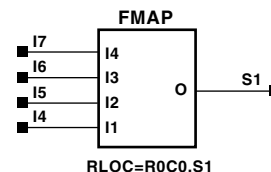
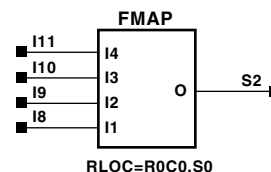
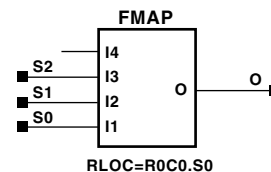
NAND12, NAND16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

The NAND function is performed in the Configurable Logic Block (CLB) function generators for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X. The 12- and 16-input NAND functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

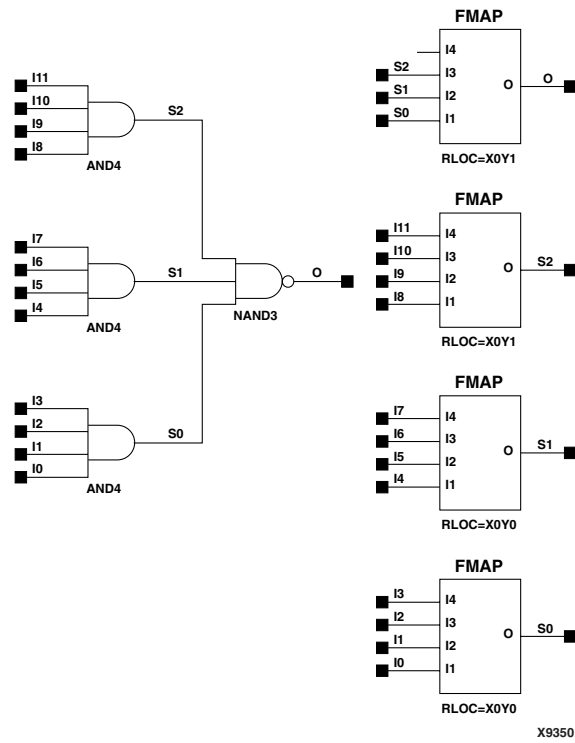
See “NAND2-9” for more information on NAND functions.



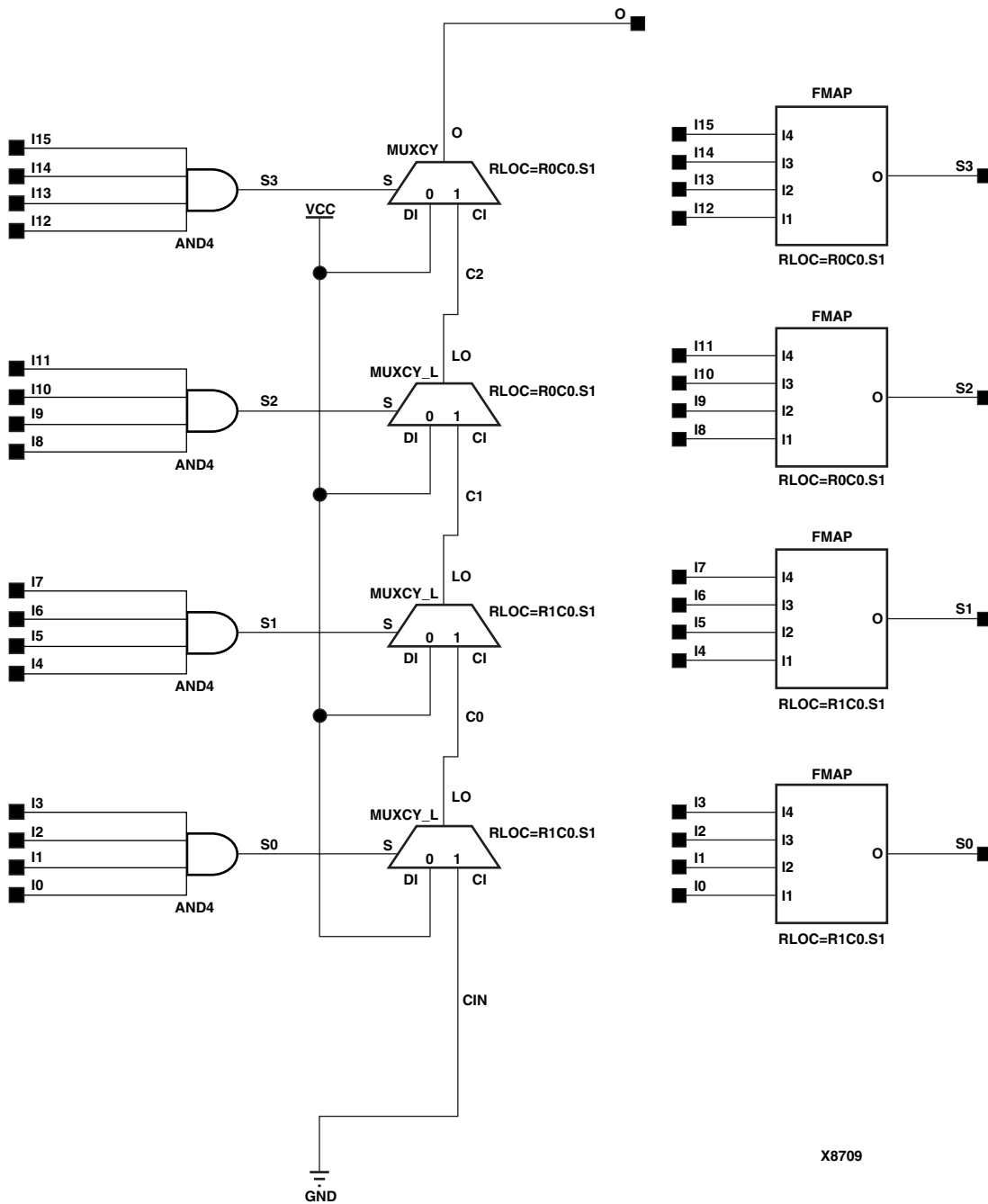
X8704



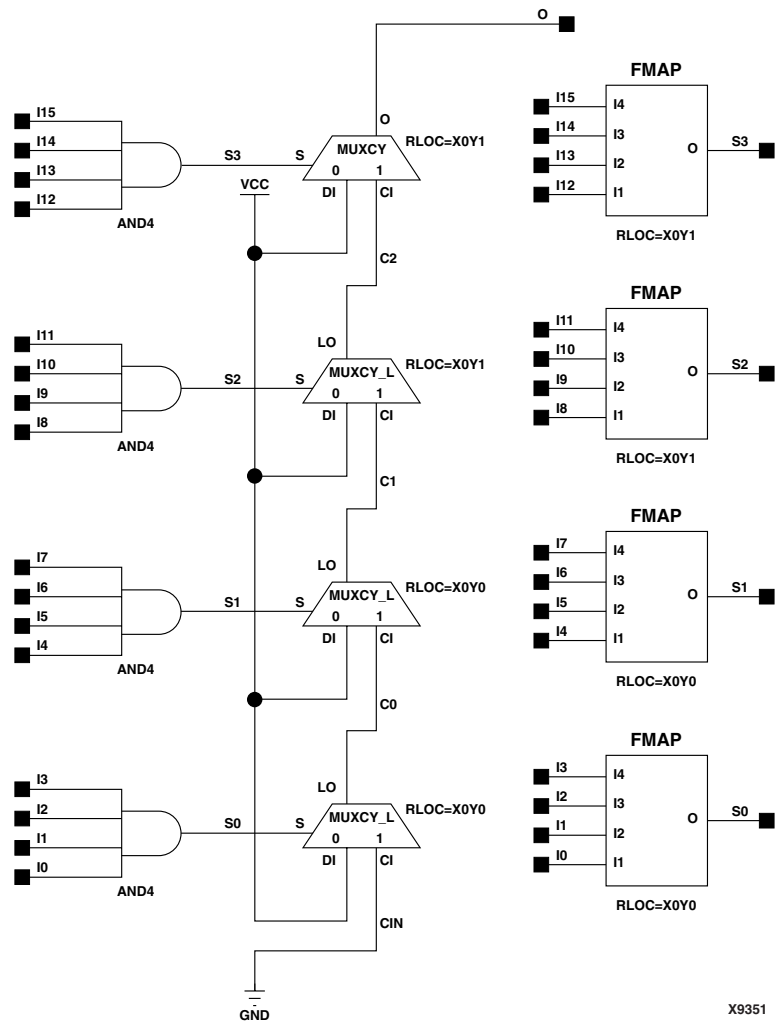
NAND12 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND12 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



NAND16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NAND16 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

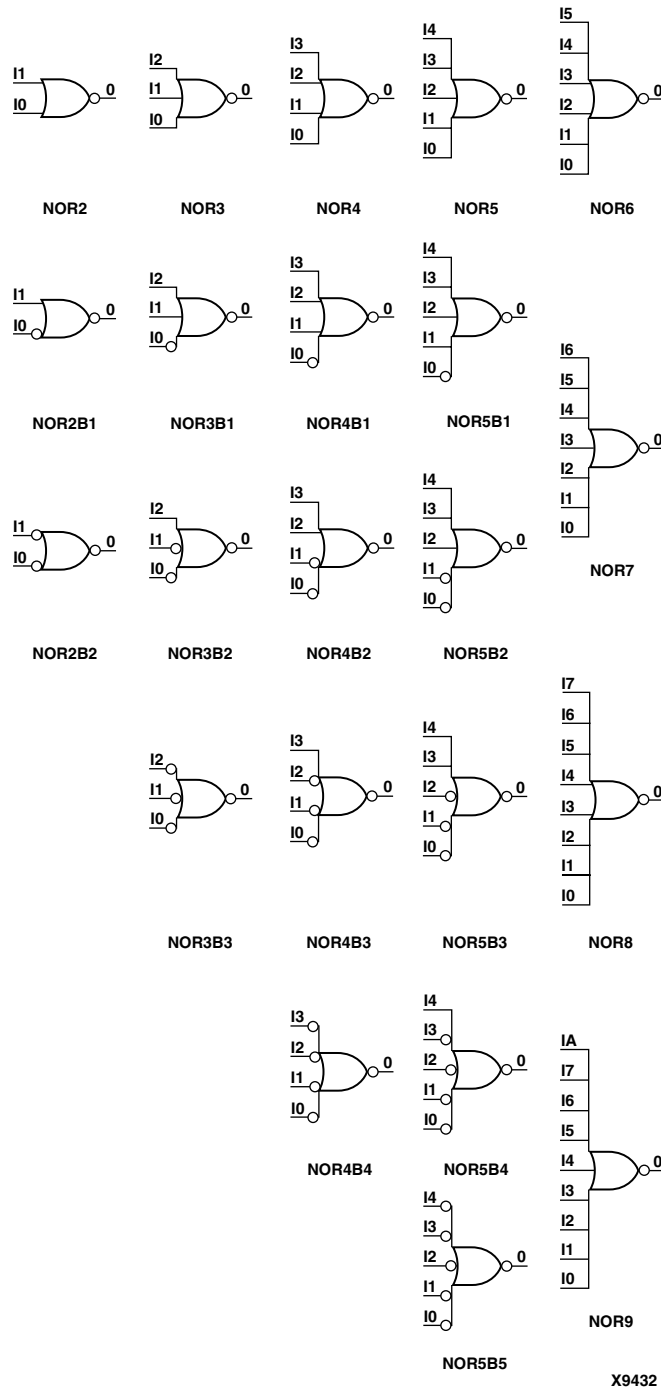
For HDL, NAND12 and NAND16 are macros that are inferred. See “NAND2-9” for more information about inferring NAND gates.

NOR2-9

2- to 9-Input NOR Gates with Inverted and Non-Inverted Inputs

Architectures Supported

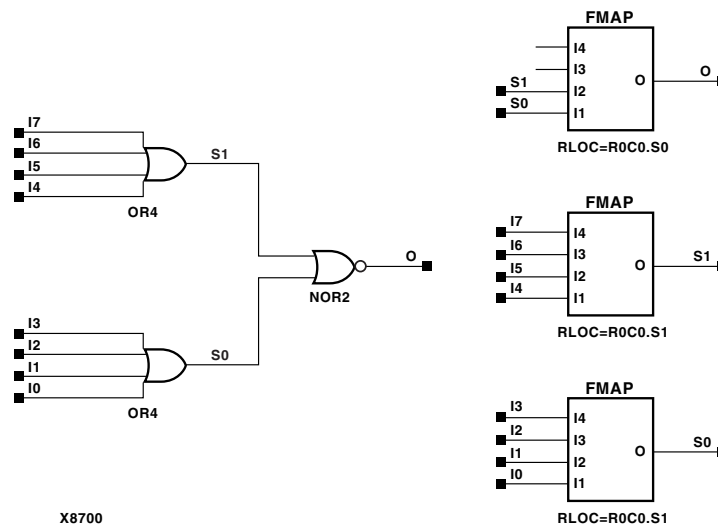
NOR2, NOR3, NOR4, NOR5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
NOR2B1, NOR2B2, NOR3B1, NOR3B2, NOR3B3, NOR4B1, NOR4B2, NOR4B3, NOR4B4, NOR5B1, NOR5B2, NOR5B3, NOR5B4, NOR5B5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
NOR6, NOR7	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
NOR8, NOR9	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



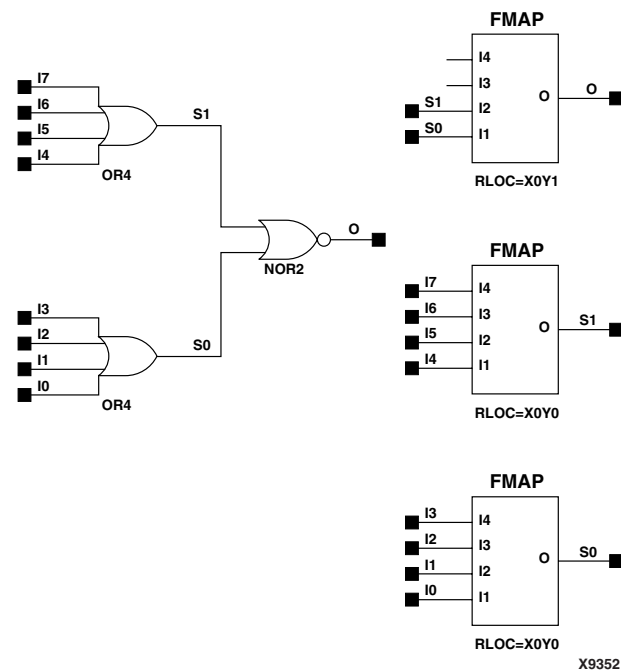
NOR Gate Representations

NOR gates of up to five inputs are available in any combination of inverting and non-inverting inputs. NOR gates of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace gates with unused inputs with gates having the necessary number of inputs.

See “NOR12, 16” for information on additional NOR functions.



NOR8 Implementation Spartan-II, Spartan-II E, Virtex, Virtex-E



NOR8 Implementation Spartan-3, Virtex-II, Virtex-II Pro , Virtex-II Pro X

Usage

NOR2 through NOR5 are primitives that can be inferred or instantiated. NOR6 through NOR9 are macros which can be inferred.

VHDL Instantiation Template for NOR5

Following is the VHDL code for NOR5. To instantiate NOR2, remove I2, I3, and I4. To instantiate NOR3, remove I3 and I4, For NOR4, remove I4. NOR2B1, and NOR2B2

have the same code as NOR2. NOR3B1, 3B2, and 3B3 have the same code as NOR3 etc..

```
-- Component Declaration for NOR5 should be placed  
-- after architecture statement but before begin keyword
```

```
component NOR5  
  port (O : out STD_ULONGIC;  
        I0 : in STD_ULONGIC;  
        I1 : in STD_ULONGIC;  
        I2 : in STD_ULONGIC;  
        I3 : in STD_ULONGIC;  
        I4 : in STD_ULONGIC);  
end component;
```

```
-- Component Attribute specification for NOR5  
-- should be placed after architecture declaration but  
-- before the begin keyword
```

```
-- Attributes should be placed here
```

```
-- Component Instantiation for NOR5 should be placed  
-- in architecture after the begin keyword
```

```
NOR5_INSTANCE_NAME : NOR5  
  port map (O => user_O,  
            I0 => user_I0,  
            I1 => user_I1,  
            I2 => user_I2,  
            I3 => user_I3,  
            I4 => user_I4);
```

Verilog Instantiation Template for NOR5

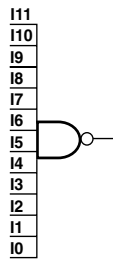
```
NOR5 NOR5_instance_name (.O (user_O),  
                        .I0 (user_I0),  
                        .I1 (user_I1),  
                        .I2 (user_I2),  
                        .I3 (user_I3),  
                        .I4 (user_I4));
```


NOR12, 16

12- and 16-Input NOR Gates with Non-Inverted Inputs

Architectures Supported

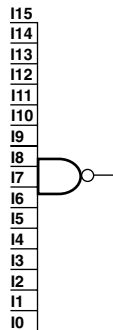
NOR12, NOR16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



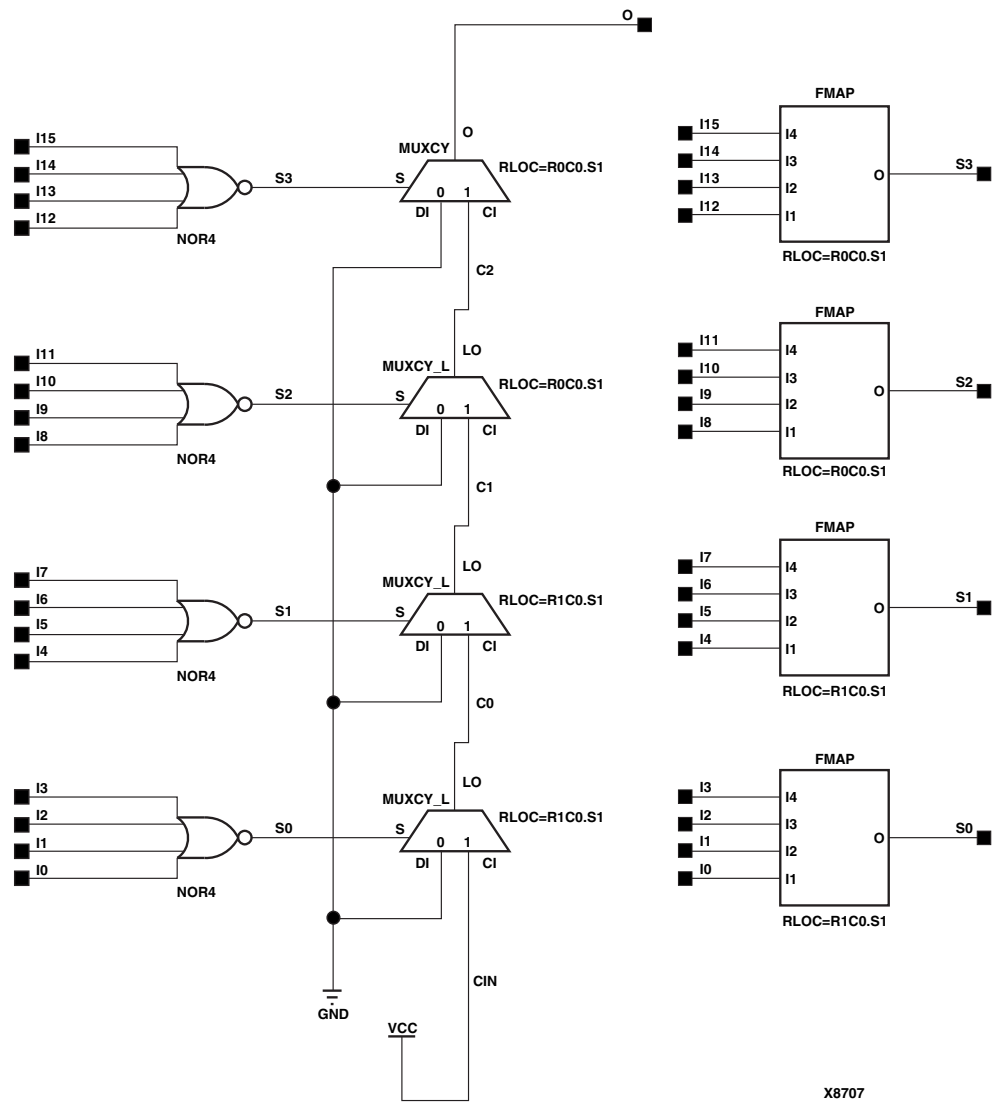
NOR12
X9433

The 12- and 16-input NOR functions are available only with non-inverting inputs. To invert some or all inputs, use external inverters.

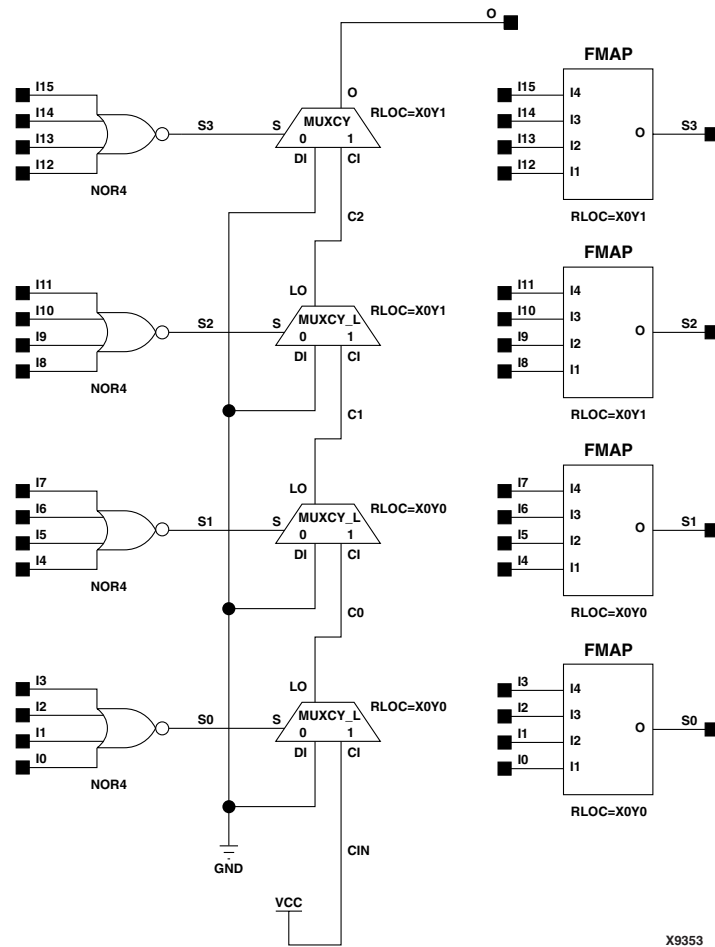
See “NOR2-9” for more information on NOR functions.



NOR16
X9434



NOR16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



NOR16 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

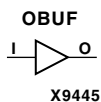
For HDL, NOR12 and NOR16 are macros that can be inferred. See “NOR2-9” for more information about inferring NOR gates.

OBUF, 4, 8, 16

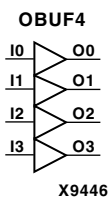
Single- and Multiple-Output Buffers

Architectures Supported

OBUF	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OBUF4, OBUF8, OBUF16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

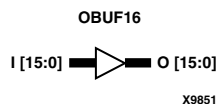
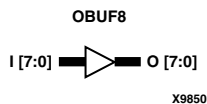


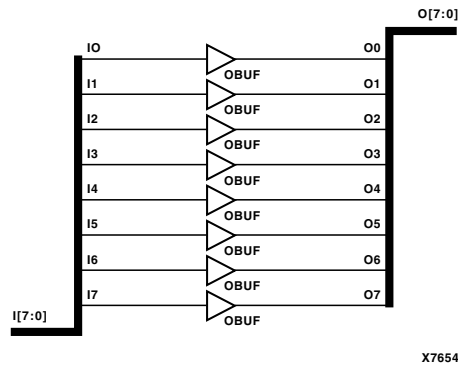
OBUF, OBUF4, OBUF8, and OBUF16 are single and multiple output buffers. An OBUF isolates the internal circuit and provides drive current for signals leaving a chip. OBUFs exist in input/output blocks (IOB). The output (O) of an OBUF is connected to an OPAD or an IOPAD.



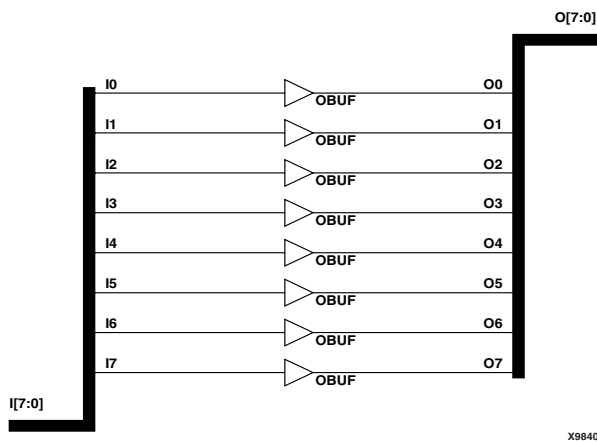
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, if a high impedance (Z) signal from an on-chip 3-state buffer (like BUFE) is applied to the input of an OBUF, it is propagated to the CPLD device output pin.

The interface standard used by OBUF, 4, 8, and 16 is LVTTTL. Also, Virtex, Virtex-E, Spartan-II, and Spartan-IIE OBUF, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.





OBUF8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Virtex, Virtex-E



OBUF8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

OBUFs are typically inferred for all top level input ports, but they can also be instantiated if necessary.

VHDL Instantiation Template

```
-- Component Declaration for OBUF should be placed
-- after architecture statement but before begin keyword

component OBUF
  -- synthesis translate_off
  generic (
    IOSTANDARD : string:= "LVTTTL";
    DRIVE       : integer := 2;
    SLEW        : string := "FAST");
  -- synthesis translate_on
  port (O : out STD_ULONGIC;
        I : in  STD_ULONGIC);
end component;

-- Component Attribute specification for OBUF
-- should be placed after architecture declaration but
```

```
-- before the begin keyword

-- Enter attributes here
attribute IOSTANDARD : string;
attribute DRIVE : integer;
attribute SLEW : string;
attribute IOSTANDARD of OBUF_instance_name : label is "LVTTTL";
attribute DRIVE of OBUF_instance_name : label is 2;
attribute SLEW of OBUF_instance_name : label is "FAST";

-- Component Instantiation for OBUF should be placed
-- in architecture after the begin keyword

OBUF_INSTANCE_NAME : OBUF
  -- synthesis translate_off
  generic map (
    IOSTANDARD => "string_value",
    DRIVE => integer := integer_value,
    SLEW => string := "string_value");
  -- synthesis translate_on
  port map (O => user_O,
            I => user_I);
```

Verilog Instantiation Template

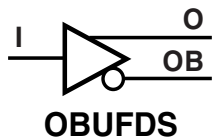
```
OBUF_instance_name (.O (user_O),
                    .I (user_I));
defparam OBUF_instance_name.IOSTANDARD => "string_value";
defparam OBUF_instance_name.DRIVE => integer_value;
defparam OBUF_instance_name.SLEW => "string_value";
```


OBUFDS

Differential Signaling Output Buffer with Selectable I/O Interface

Architectures Supported

OBUFDS	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9259

OBUFDS is a single output buffer that supports low-voltage, differential signaling (1.8v CMOS). OBUFDS isolates the internal circuit and provides drive current for signals leaving the chip. Its output is represented as two distinct ports (O and OB), one deemed the "master" and the other the "slave." The master and the slave are opposite phases of the same logical signal (for example, MYNET and MYNETB).

Inputs	Outputs	
	O	OB
0	0	1
1	1	0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OBUFDS: Differential Output Buffer
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OBUFDS_inst : OBUFDS
-- Edit the following generics to specify the I/O standard, drive and
-- slew rate.
generic map (
    DRIVE => 12,
    IOSTANDARD => "LVDS_25",
    SLEW => "SLOW")
port map (
    O => O,    -- Diff_p output (connect to top-level port)
    OB => OB,  -- Diff_n output (connect to top-level port)
    I => I     -- Buffer input
);

-- End of OBUFDS_inst instantiation
```

Verilog Instantiation Template

```
// OBUFDS: Differential Output Buffer
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OBUFDS OBUFDS_inst (
    .O(O), // Diff_p output(connect directly to top-level port)
    .OB(OB), // Diff_n output (connect directly to top-level port)
    .I(I) // Buffer input
);

// Edit the following defparams to specify the I/O standard, drive and
// slew rate. If the instance name is change, that change needs to be
// reflecting the this defparam.

defparam OBUFDS_inst.DRIVE = 12;
defparam OBUFDS_inst.IOSTANDARD = "LVDS_25";
defparam OBUFDS_inst.SLEW = "SLOW";

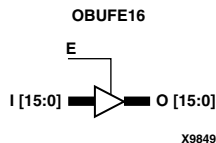
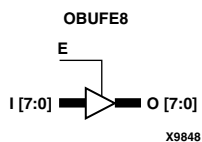
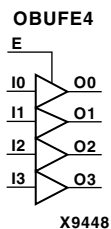
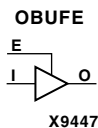
// End of OBUFDS_inst instantiation
```

OBUFE, 4, 8, 16

3-State Output Buffers with Active-High Output Enable

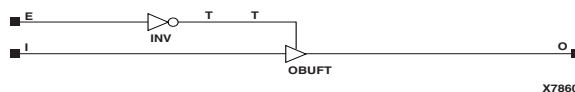
Architectures Supported

OBUFE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OBUFE4, OBUFE8, OBUFE16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

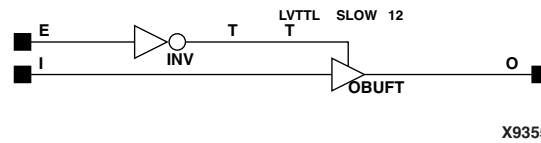


OBUFE, OBUFE4, OBUFE8, and OBUFE16 are 3-state buffers with inputs I, I3 – I0, I7 – I0, and I15-I0, respectively; outputs O, O3 – O0, O7 – O0, and O15-O0, respectively; and active-High output enable (E). When E is High, data on the inputs of the buffers is transferred to the corresponding outputs. When E is Low, the output is High impedance (off or Z state). An OBUFE isolates the internal circuit and provides drive current for signals leaving a chip. An OBUFE output is connected to an OPAD or an IOPAD. An OBUFE input is connected to the internal circuit.

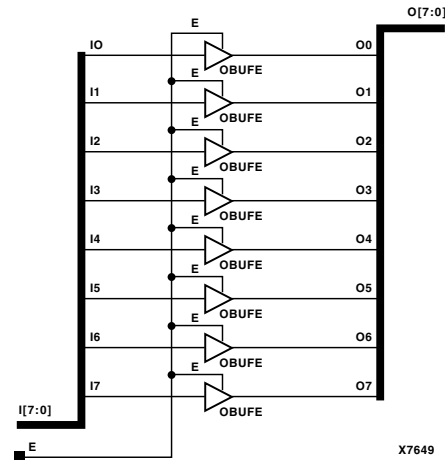
Inputs		Outputs
E	I	O
0	X	Z
1	1	1
1	0	0



OBUFE Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OBUFE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



OBUFE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFE should be placed
-- after architecture statement but before begin keyword

component OBUFE
  port (O : out STD_ULOGIC;
        E : in  STD_ULOGIC;
        I : in  STD_ULOGIC);
end component;

-- Component Attribute specification for OBUFE
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for OBUFE should be placed
-- in architecture after the begin keyword

OBUFE_INSTANCE_NAME : OBUFE
  port map (O => user_O,
            E => user_E,
            I => user_I);
```

Verilog Instantiation Template

```
OBUFE_instance_name (.O (user_O),  
                    .E (user_E),  
                    .I (user_I));
```

Commonly Used Constraints

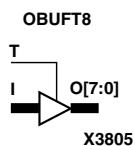
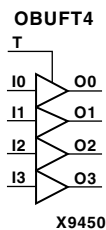
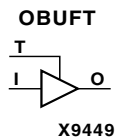
IOBDELAY

OBUFT, 4, 8, 16

Single and Multiple 3-State Output Buffers with Active-Low Output Enable

Architectures Supported

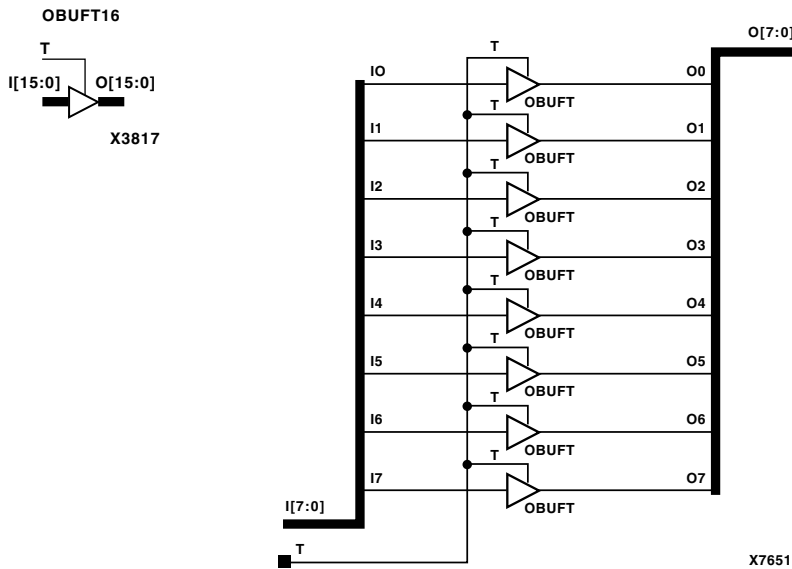
OBUFT	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OBUFT4, OBUFT8, OBUFT16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



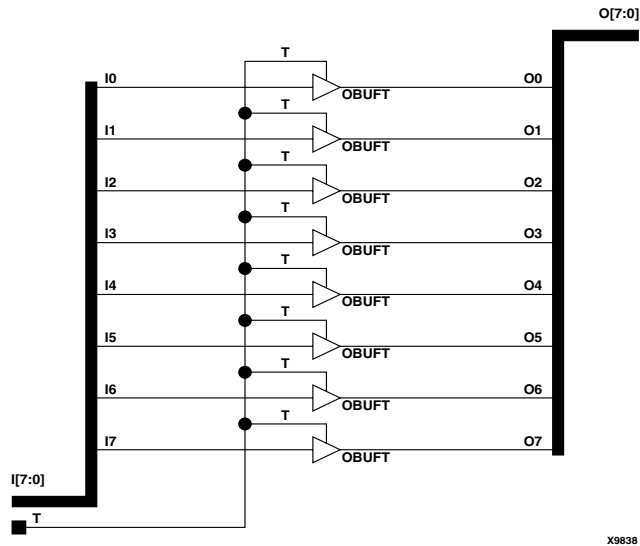
OBUFT, OBUFT4, OBUFT8, and OBUFT16 are single and multiple 3-state output buffers with inputs I, I3 – I0, I7 – I0, I15 – I0, outputs O, O3 – O0, O7 – O0, O15 – O0, and active-Low output enables (T). When T is Low, data on the inputs of the buffers is transferred to the corresponding outputs. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip. An OBUFT output is connected to an OPAD or an IOPAD.

OBUFT, 4, 8, and 16 use the LVTTTL standard. Also, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X, Spartan-II, Spartan-IIE, and Spartan-3 OBUFT, 4, 8, and 16 have selectable drive and slew rates using the DRIVE and SLOW or FAST constraints. The defaults are DRIVE=12 mA and SLOW slew.

Inputs		Outputs
T	I	O
1	X	Z
0	1	1
0	0	0



OBUFT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Virtex, Virtex-E



OBUFT8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

OBUFT and its variants have selectable drive and slew rates using the DRIVE and FAST or SLOW constraints. The defaults are DRIVE=12 mA and SLOW slew.

When T is Low, data on the input of the buffer is transferred to the output. When T is High, the output is high impedance (off or Z state). OBUFTs isolate the internal circuit and provide extra drive current for signals leaving a chip.

Usage

For HDL, these design elements are instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for OBUFT should be placed
-- after architecture statement but before begin keyword

component OBUFT
  -- synthesis translate_off
  generic (
    IOSTANDARD : string:= "LVTTTL";
    DRIVE : integer := 2;
    SLEW : string := "FAST");
  -- synthesis translate_on
  port (O : out STD_ULONGIC;
        I : in STD_ULONGIC;
        T : in STD_ULONGIC);
end component;

-- Component Attribute specification for OBUFT
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

attribute IOSTANDARD : string;
attribute DRIVE : integer;
attribute SLEW : string;
attribute IOSTANDARD of OBUFT_instance_name : label is "LVTTTL";
attribute DRIVE of OBUFT_instance_name : label is 2;
attribute SLEW of OBUFT_instance_name : label is "FAST";

-- Component Instantiation for OBUFT should be
-- placed in architecture after the begin keyword

OBUFT_INSTANCE_NAME : OBUFT
  -- synthesis translate_off
  generic map (
    IOSTANDARD => "string_value",
    DRIVE => integer := integer_value,
    SLEW => string := "string_value");
  -- synthesis translate_on
  port map (O => user_O,
            I => user_I,
            T => user_T);
```

Verilog Instantiation Template

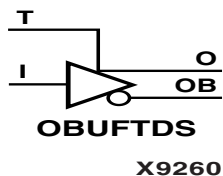
```
OBUFT instance_name (.O (user_O),
                    .I (user_I),
                    .T (user_T));
defparam OBUFT_instance_name.IOSTANDARD => "string_value";
defparam OBUFT_instance_name.DRIVE => integer_value;
defparam OBUFT_instance_name.SLEW => "string_value";
```


OBUFTDS

3-State Differential Signaling Output Buffer with Active Low Output Enable and Selectable I/O Interface

Architectures Supported

OBUFTDS	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OBUFTDS is a single 3-state, differential signaling output buffer with active Low enable and a Select I/O interface.

When T is Low, data on the input of the buffer is transferred to the output (O) and inverted output (OB). When T is High, both outputs are high impedance (off or Z state).

Inputs		Outputs	
I	T	O	OB
X	1	Z	Z
0	0	0	1
1	0	1	0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OBUFTDS: Differential 3-state Output Buffer
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OBUFTDS_inst : OBUFTDS
-- Edit the following generics to specify the I/O standard, drive and
-- slew rate.
generic map (
    DRIVE => 12,
    IOSTANDARD => "LVDS_25",
    SLEW => "SLOW")
port map (
    O => O,      -- Diff_p output (connect directly to top-level port)
    OB => OB,    -- Diff_n output (connect directly to top-level port)
```

```
        I => I,      -- Buffer input
        T => T      -- 3-state enable input
    );

-- End of OBUFTDS_inst instantiation
```

Verilog Instantiation Template

```
// OBUFTDS: Differential 3-state Output Buffer
//           Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OBUFTDS OBUFTDS_inst (
    .O(O),      // Diff_p output (connect directly to top-level port)
    .OB(OB),    // Diff_n output (connect directly to top-level port)
    .I(I),      // Buffer input
    .T(T)       // 3-state enable input
);

// Edit the following defparams to specify the I/O standard, drive and
// slew rate. If the instance name is change, that change needs to be
// reflecting the this defparam.

defparam OBUFTDS_inst.DRIVE = 12;
defparam OBUFTDS_inst.IOSTANDARD = "LVDS_25";
defparam OBUFTDS_inst.SLEW = "SLOW";

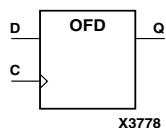
// End of OBUFTDS_inst instantiation
```

OFD, 4, 8, 16

Single- and Multiple-Output D Flip-Flops

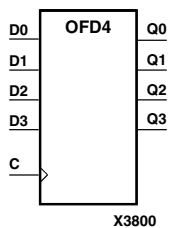
Architectures Supported

OFD, OFD4, OFD8, OFD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OFD, OFD4, OFD8, and OFD16 are single and multiple output D flip-flops.

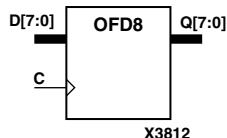
The outputs (for example, Q3 – Q0) are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs.



The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

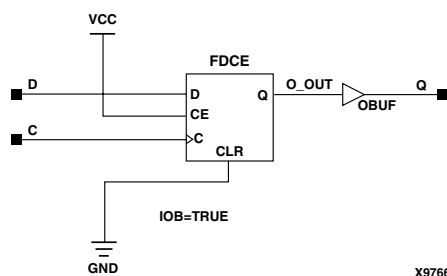
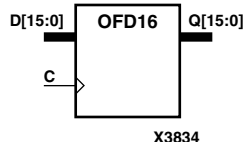
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

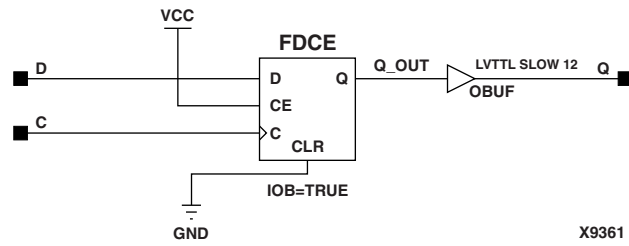


GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

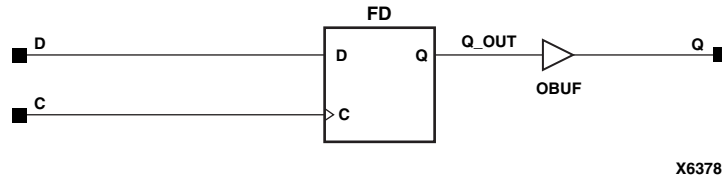
Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1



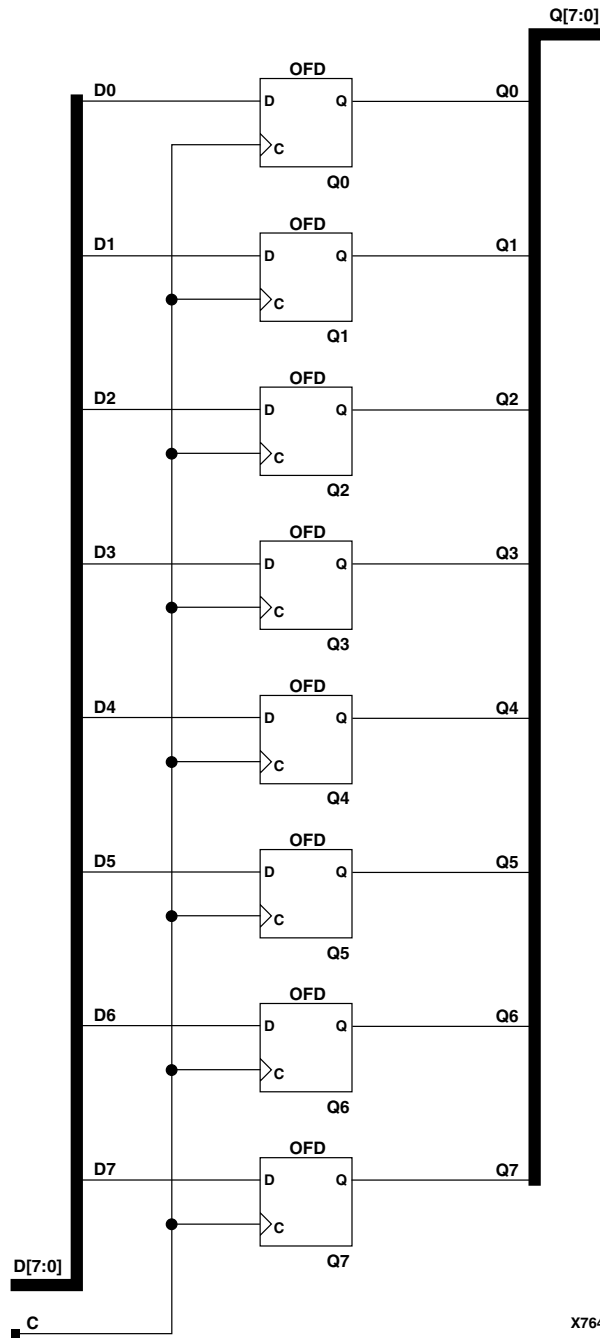
OFD Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



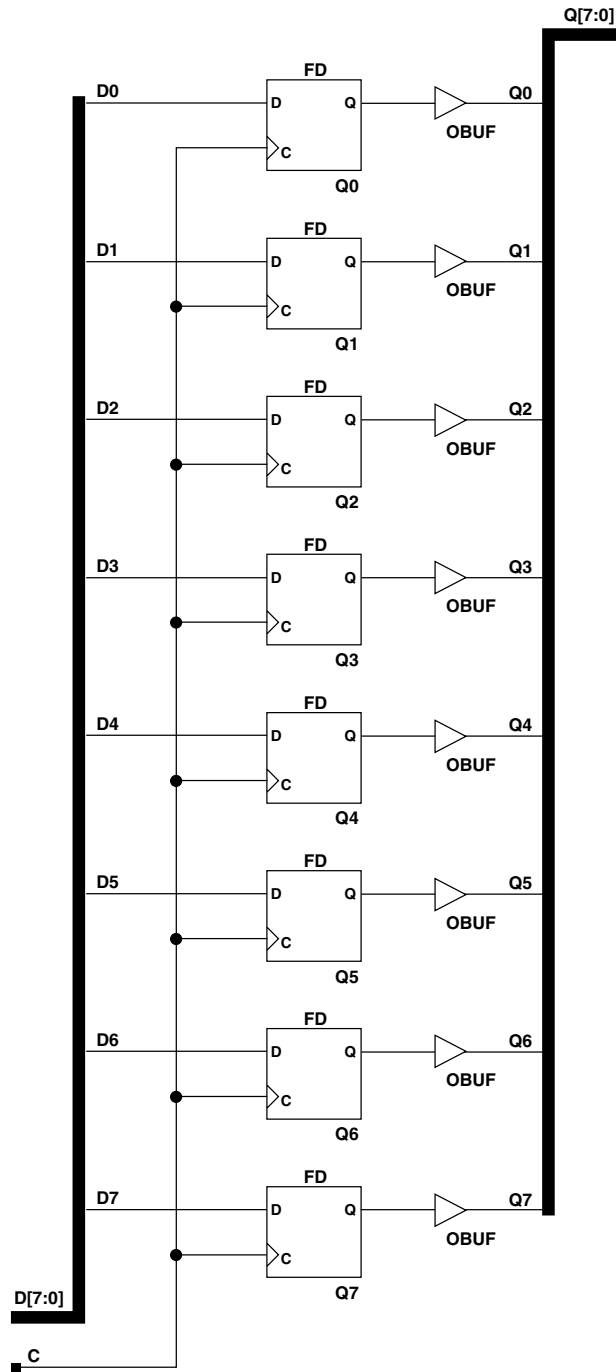
OFD Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



OFD Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFD8 Implementation Spartan-II, Spartan-II-E, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X7648

OFD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

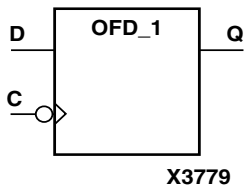
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFD, you would infer an FD and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFD_1

Output D Flip-Flop with Inverted Clock

Architectures Supported

OFD_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



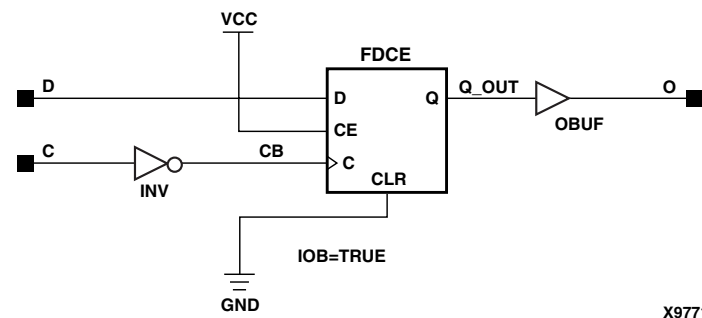
OFD_1 is located in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

The flip-flop is asynchronously cleared, output Low, when power is applied, or when global reset is active.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↓	D



OFD_1 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

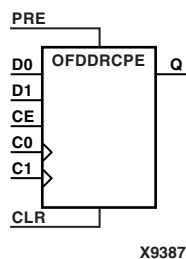
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFD_1, you would infer an FD_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDDRCPE

Dual Data Rate Output D Flip-Flop with Clock Enable and Asynchronous Preset and Clear

Architectures Supported

OFDDRCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



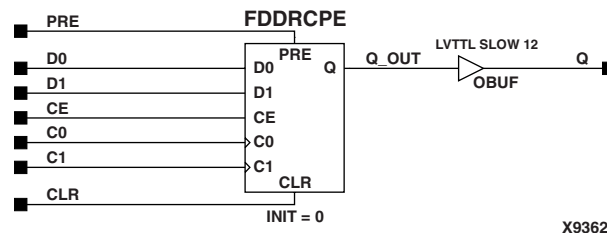
OFDDRCPE is a dual data rate (DDR) output D flip-flop with clock enable (CE) and asynchronous preset (PRE) and clear (CLR). It consists of one output buffer and one dual data rate flip-flop (FDDRCPE).

When the asynchronous PRE is High and CLR is Low, the Q output is preset High. When CLR is High, Q is set Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition.

The INIT attribute does not apply to OFDDRCPE components.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1



OFDDRCPE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OFDDRCPE: Double Data Rate Output Register with Async. Clear,
--           Async. Preset
--           and Clock Enable. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OFDDRCPE_inst : OFDDRCPE
port map (
    Q => Q,      -- Data output (connect directly to top-level port)
    C0 => C0,    -- 0 degree clock input
    C1 => C1,    -- 180 degree clock input
    CE => CE,    -- Clock enable input
    CLR => CLR,  -- Asynchronous reset input
    D0 => D0,    -- Posedge data input
    D1 => D1,    -- Negedge data input
    PRE => PRE   -- Asynchronous preset input
);

-- End of OFDDRCPE_inst instantiation
```

Verilog Instantiation Template

```
// OFDDRCPE: Double Data Rate Output Register with Async. Clear,
//           Async. Preset and Clock Enable. Virtex-II/II-Pro, Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OFDDRCPE OFDDRCPE_inst (
    .Q(Q),      // Data output (connect directly to top-level port)
    .C0(C0),   // 0 degree clock input
    .C1(C1),   // 180 degree clock input
    .CE(CE),   // Clock enable input
    .CLR(CLR), // Asynchronous reset input
    .D0(D0),   // Posedge data input
    .D1(D1),   // Negedge data input
    .PRE(PRE)  // Asynchronous preset input
);

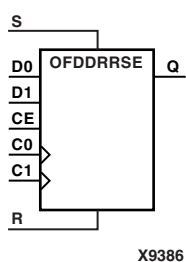
// End of OFDDRCPE_inst instantiation
```

OFDDRSE

Dual Data Rate Output D Flip-Flop with Synchronous Reset and Set and Clock Enable

Architectures Supported

OFDDRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



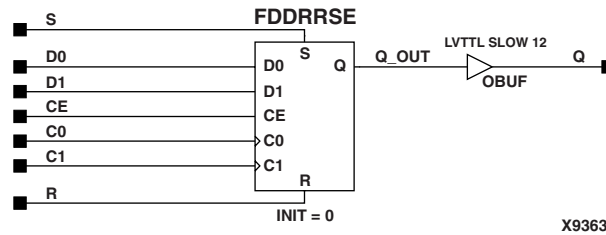
OFDDRSE is a dual data rate (DDR) output D flip-flop with synchronous reset (R) and set (S) and clock enable (CE). It consists of one output buffer and one dual data rate flip-flop (FDDRSE).

On a Low-to-High clock transition (C0 or C1), a High R input resets the Q output Low; a Low R input with a High S input sets Q High. When both R and S are Low and clock enable is High, data on the D0 input is loaded into the flip-flop on a Low-to-High C0 clock transition and data on the D1 input is loaded into the flip-flop on a Low-to-High C1 clock transition.

The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

The INIT attribute does not apply to OFDDRSE components.

Inputs							Outputs
C0	C1	CE	D0	D1	R	S	Q
↑	X	X	X	X	1	0	0
↑	X	X	X	X	0	1	1
↑	X	X	X	X	1	1	0
X	↑	X	X	X	1	0	0
X	↑	X	X	X	0	1	1
X	↑	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1



OFDDRSE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OFDDRSE: Double Data Rate Input Register with Sync. Clear,
--           Sync. Preset
--           and Clock Enable. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OFDDRSE_inst : OFDDRSE
port map (
    Q => Q,      -- Data output (connect directly to top-level port)
    C0 => C0,    -- 0 degree clock input
    C1 => C1,    -- 180 degree clock input
    CE => CE,    -- Clock enable input
    D0 => D0,    -- Posedge data input
    D1 => D1,    -- Negedge data input
    R => R,      -- Synchronous reset input
    S => S       -- Synchronous preset input
);

-- End of OFDDRSE_inst instantiation
```

Verilog Instantiation Template

```
// OFDDRSE: Double Data Rate Output Register with Async. Clear,
//           Async.
//           Preset and Clock Enable with 3-state. Virtex-II/II-Pro,
//           Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OFDDRSE OFDDRSE_inst (
    .Q(Q),      // Data output (connect directly to top-level port)
    .C0(C0),    // 0 degree clock input
    .C1(C1),    // 180 degree clock input
    .CE(CE),    // Clock enable input
    .D0(D0),    // Posedge data input
    .D1(D1),    // Negedge data input
    .R(R),      // Synchronous reset input
    .S(S)       // Synchronous preset input
);

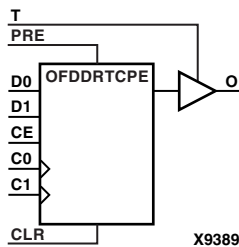
// End of OFDDRSE_inst instantiation
```


OFDDRTCPE

Dual Data Rate D Flip-Flop with Active-Low 3-State Output Buffer, Clock Enable, and Asynchronous Preset and Clear

Architectures Supported

OFDDRTCPE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OFDDRTCPE is a dual data rate (DDR) D flip-flop with clock enable (CE) and asynchronous preset and clear whose output is enabled by a 3-state buffer. It consists of a dual data rate flip-flop (FDDRCPE) and a 3-state output buffer (OBUFT). The data output (O) of the flip-flop is connected to the input of the output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or IOPAD.

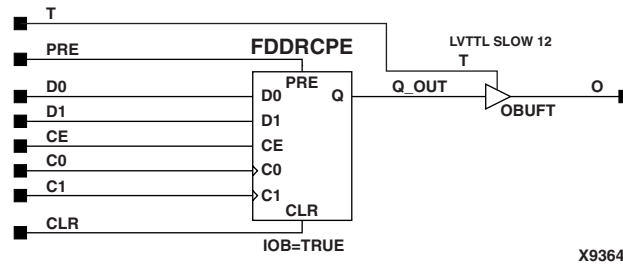
When the active-Low enable input (T) is Low, output is enabled and the data on the flip-flop's Q output appears on the OBUFT's O output. When the asynchronous PRE is High and CLR is Low, the O output is preset High. When CLR is High, O is set Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition.

When T is High, outputs are high impedance (Off). When CE is Low and T is Low, the outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to OFDDRTCPE components.

Inputs								Outputs
C0	C1	CE	D0	D1	CLR	PRE	T	O
X	X	X	X	X	X	X	1	Z
X	X	X	X	X	1	0	0	0
X	X	X	X	X	0	1	0	1
X	X	X	X	X	1	1	0	0
X	X	0	X	X	0	0	0	No Chg
↑	X	1	D0	X	0	0	0	D0
X	↑	1	X	D1	0	0	0	D1



OFDDRTCPE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OFDDRTCPE: Double Data Rate Output Register with Async. Clear,
--           Async. Preset
--           and Clock Enable with 3-state. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OFDDRTCPE_inst : OFDDRTCPE
port map (
    Q => Q,          -- Data output (connect directly to top-level port)
    C0 => C0,        -- 0 degree clock input
    C1 => C1,        -- 180 degree clock input
    CE => CE,        -- Clock enable input
    CLR => CLR,      -- Asynchronous reset input
    D0 => D0,        -- Posedge data input
    D1 => D1,        -- Negedge data input
    PRE => PRE,      -- Asynchronous preset input
    T => T           -- 3-state enable input
);

-- End of OFDDRTCPE_inst instantiation

-- Component Instantiation for OFDDRTCPE should be placed
-- in architecture after the begin keyword
```

Verilog Instantiation Template

```
// OFDDRTCPE: Double Data Rate Output Register with Async. Clear,
//           Async. Preset and Clock Enable with 3-state. Virtex-II/II-Pro,
//           Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OFDDRTCPE OFDDRTCPE_inst (
    .Q(Q),          // Data output (connect directly to top-level port)
    .C0(C0),        // 0 degree clock input
    .C1(C1),        // 180 degree clock input
    .CE(CE),        // Clock enable input
    .CLR(CLR),      // Asynchronous reset input
    .D0(D0),        // Posedge data input
```

```
.D1(D1),    // Negedge data input
.PRE(PRE),  // Asynchronous preset input
.T(T)       // 3-state enable input
);

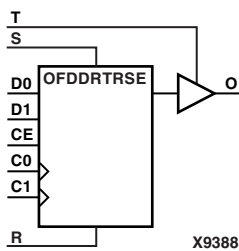
// End of OFDDRTCPE_inst instantiation
```


OFDDRTRSE

Dual Data Rate D Flip-Flop with Active-Low 3-State Output Buffer, Synchronous Reset and Set, and Clock Enable

Architectures Supported

OFDDRTRSE	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OFDDRTRSE is a dual data rate (DDR) D flip-flop with clock enable (CE) and synchronous reset and set whose output is enabled by a 3-state buffer. It consists of a dual data rate flip-flop (FDDRSE) and a 3-state output buffer (OBUFT). The data output (O) of the flip-flop is connected to the input of the output buffer (OBUFT). The output of the OBUFT is connected to an OPAD or IOPAD.

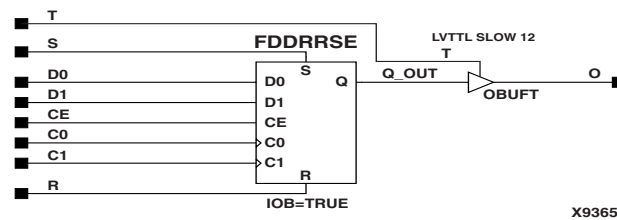
When the active-Low enable input (T) is Low, output is enabled and the data on the flip-flop's Q output appears on the OBUFT's O output. On a Low-to-High clock transition (C0 or C1), a High R input resets the Q output Low; a Low R input with a High S input sets O High. When both R and S are Low and clock enable is High, data on the D0 input is loaded into the flip-flop on a Low-to-High C0 clock transition and data on the D1 input is loaded into the flip-flop on a Low-to-High C1 clock transition.

When T is High, outputs are high impedance (Off). When CE is Low and T is Low, the outputs do not change.

The flip-flops are asynchronously cleared with Low outputs when power is applied.

The INIT attribute does not apply to OFDDRTRSE components.

Inputs								Outputs
C0	C1	CE	D0	D1	R	S	T	O
X	X	X	X	X	X	X	1	Z
↑	X	X	X	X	1	0	0	0
↑	X	X	X	X	0	1	0	1
↑	X	X	X	X	1	1	0	0
X	↑	X	X	X	1	0	0	0
X	↑	X	X	X	0	1	0	1
X	↑	X	X	X	1	1	0	0
X	X	0	X	X	0	0	0	No Chg
↑	X	1	D0	X	0	0	0	D0
X	↑	1	X	D1	0	0	0	D1



OFDDRTRSE Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- OFDDRTRSE: Double Data Rate Input Register with Sync. Clear,
--           Sync. Preset
--           and Clock Enable with 3-state. Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

OFDDRTRSE_inst : OFDDRTRSE
port map (
    Q => Q,      -- Data output (connect directly to top-level port)
    C0 => C0,    -- 0 degree clock input
    C1 => C1,    -- 180 degree clock input
    CE => CE,    -- Clock enable input
    D0 => D0,    -- Posedge data input
    D1 => D1,    -- Negedge data input
    R => R,      -- Synchronous reset input
    S => S,      -- Synchronous preset input
    T => T      -- 3-state enable input
);

-- End of OFDDRTRSE_inst instantiation
```

Verilog Instantiation Template

```
// OFDDRTRSE: Double Data Rate Input Register with Sync. Clear, Sync.
//           Preset and Clock Enable with 3-state. Virtex-II/II-Pro,
//           Spartan-3
// The current version of the Xilinx HDL Libraries Guide

OFDDRTRSE OFDDRTRSE_inst (
    .Q(Q),      // Data output (connect directly to top-level port)
    .C0(C0),    // 0 degree clock input
    .C1(C1),    // 180 degree clock input
    .CE(CE),    // Clock enable input
    .D0(D0),    // Posedge data input
    .D1(D1),    // Negedge data input
    .R(R),      // Synchronous reset input
    .S(S),      // Synchronous preset input
    .T(T)       // 3-state enable input
);

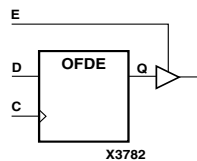
// End of OFDDRTRSE_inst instantiation
```

OFDE, 4, 8, 16

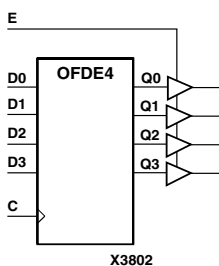
D Flip-Flops with Active-High Enable Output Buffers

Architectures Supported

OFDE, OFDE4, OFDE8, OFDE16	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OFDE, OFDE4, OFDE8, and OFDE16 are single or multiple D flip-flops whose outputs are enabled by 3-state buffers. The flip-flop data outputs (Q) are connected to the inputs of output buffers (OBUFE). The OBUFE outputs (O) are connected to OPADs or IOPADs. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-High enable inputs (E) are High, the data on the flip-flop outputs (Q) appears on the O outputs. When E is Low, outputs are high impedance (Z state or Off).

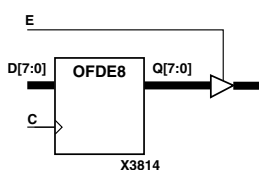


The flip-flops are asynchronously cleared with Low outputs when power is applied, or when global reset is active.

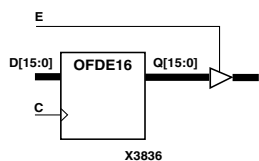
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

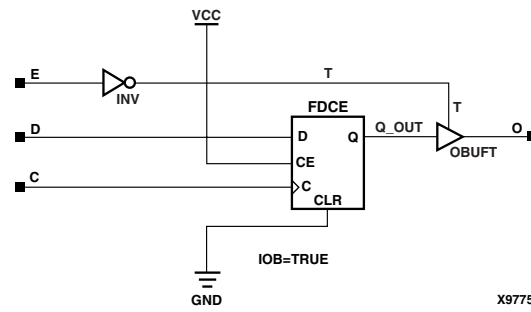
Spartan-II, Spartan-III, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

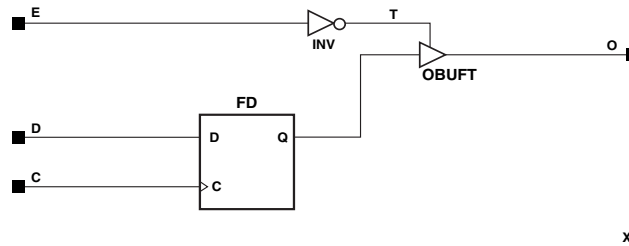


Inputs			Outputs
E	D	C	O
0	X	X	Z
1	1	↑	1
1	0	↑	0

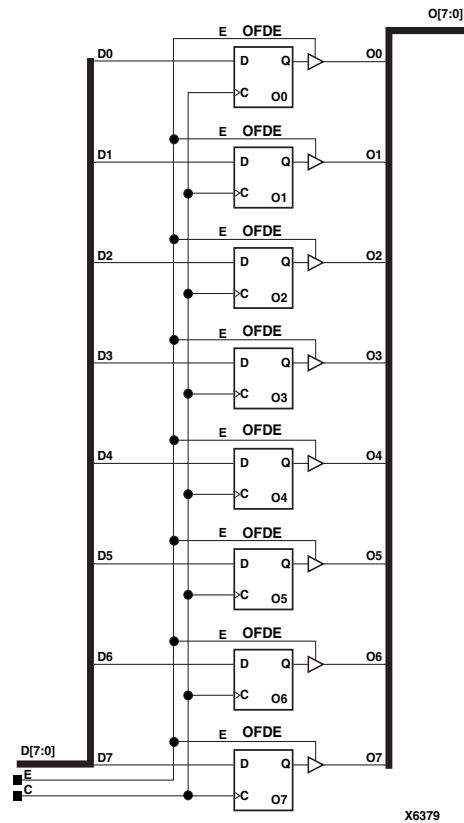




OFDE Implementation Spartan-II, Spartan-IIIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



OFDE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFDE8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

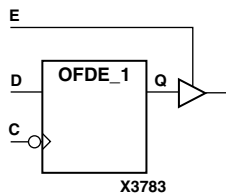
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDE, you would infer an FDE and put the IOB = TRUE attribute on the component. Or, you could use the map option -pr o to pack all output registers into the IOBs.

OFDE_1

D Flip-Flop with Active-High Enable Output Buffer and Inverted Clock

Architectures Supported

OFDE_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



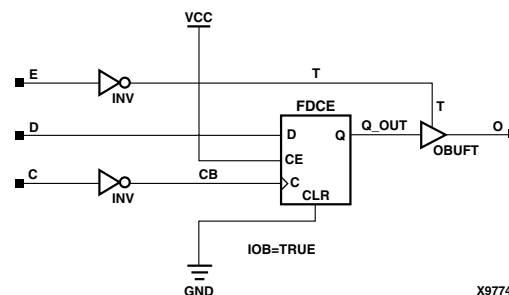
OFDE_1 and its output buffer are located in an input/output block (IOB). The data output of the flip-flop (Q) is connected to the input of an output buffer or OBUFE. The output of the OBUFE is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-High enable input (E) is High, the data on the flip-flop output (Q) appears on the O output. When E is Low, the output is high impedance (Z state or Off).

The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

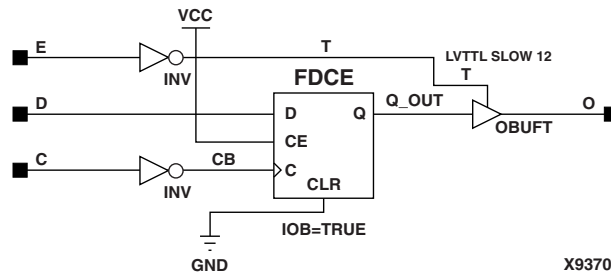
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
E	D	C	O
0	X	X	Z
1	1	↓	1
1	0	↓	0



OFDE_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDE_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

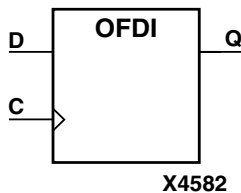
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDE_1, you would infer an FDE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDI

Output D Flip-Flop (Asynchronous Preset)

Architectures Supported

OFDI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



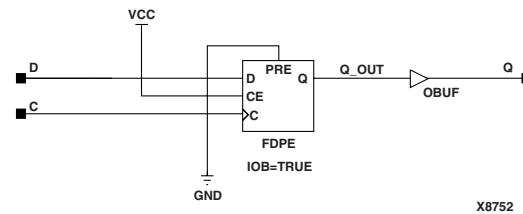
OFDI is contained in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q).

The flip-flop is asynchronously preset, output High, when power is applied.

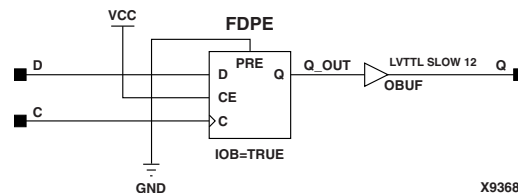
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↑	D



OFDI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

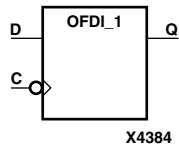
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDI, you would infer an FDP and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDI_1

Output D Flip-Flop with Inverted Clock (Asynchronous Preset)

Architectures Supported

OFDI_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



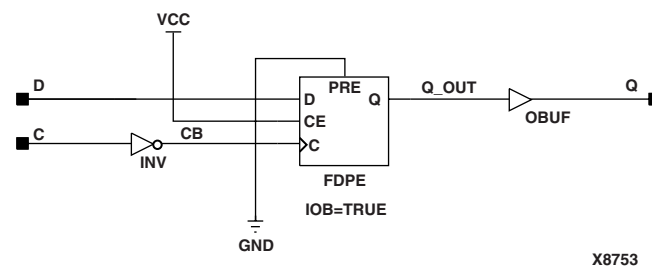
OFDI_1 exists in an input/output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output.

The flip-flop is asynchronously preset, output High, when power is applied.

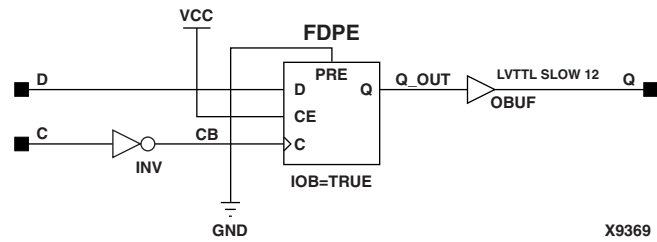
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs		Outputs
D	C	Q
D	↓	D



OFDI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

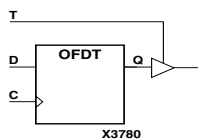
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an `IOB=TRUE` attribute on the component in the UCF file or in the code. For instance, to get an `OFDI_1`, you would infer an `FDP_1` and put the `IOB = TRUE` attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDT, 4, 8, 16

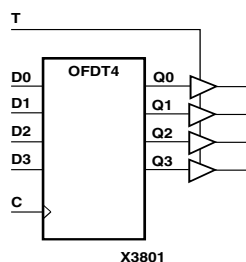
Single and Multiple D Flip-Flops with Active-Low 3-State Output Enable Buffers

Architectures Supported

OFDT, OFDT4, OFDT8, OFDT16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



OFDT, OFDT4, OFDT8, and OFDT16 are single or multiple D flip-flops whose outputs are enabled by a 3-state buffers. The data outputs (Q) of the flip-flops are connected to the inputs of output buffers (OBUFT). The outputs of the OBUFTs (O) are connected to OPADs or IOPADs. The data on the data inputs (D) is loaded into the flip-flops during the Low-to-High clock (C) transition. When the active-Low enable inputs (T) are Low, the data on the flip-flop outputs (Q) appears on the O outputs. When T is High, outputs are high impedance (Off).

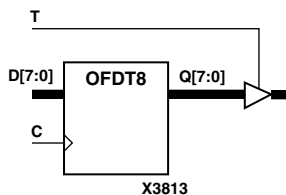


The flip-flops are asynchronously cleared with Low outputs, when power is applied, or when global reset is active.

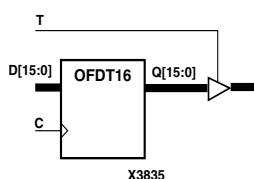
For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

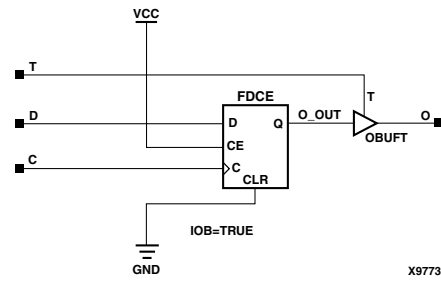
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

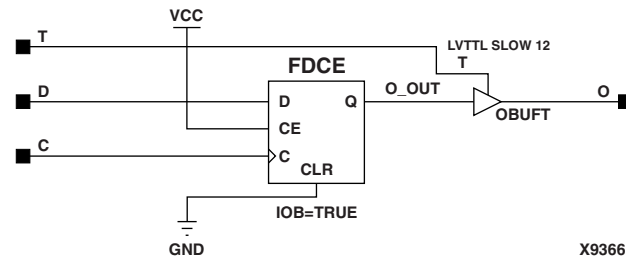


Inputs			Outputs
T	D	C	O
1	X	X	Z
0	D	↑	D

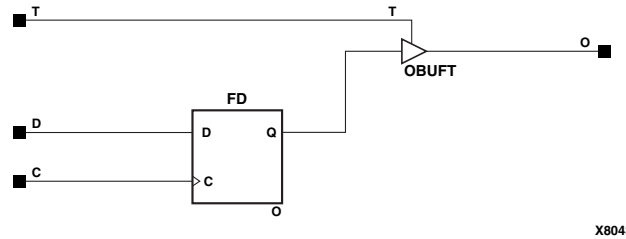




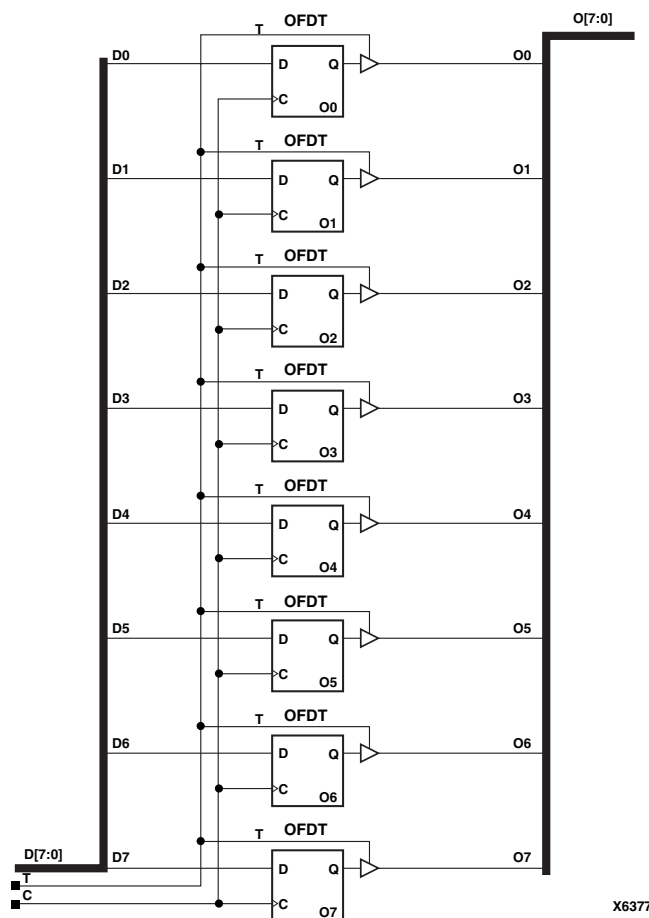
OFDT Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDT Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



OFDT Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



OFDT8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

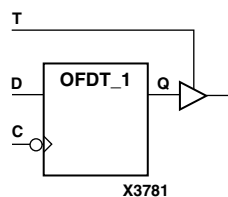
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an `IOB=TRUE` attribute on the component in the UCF file or in the code. For instance, to get an OFDT, you would infer an FDCE and put the `IOB = TRUE` attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDT_1

D Flip-Flop with Active-Low 3-State Output Buffer and Inverted Clock

Architectures Supported

OFDT_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



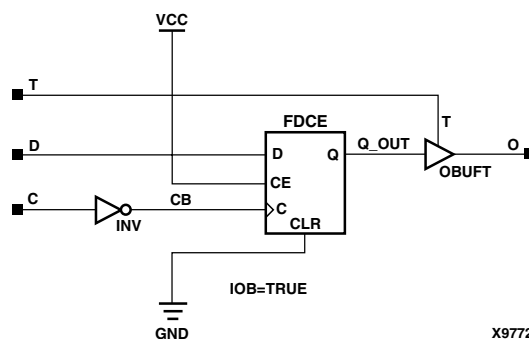
OFDT_1 and its output buffer are located in an input/output block (IOB). The flip-flop data output (Q) is connected to the input of an output buffer (OBUFT). The OBUFT output is connected to an OPAD or an IOPAD. The data on the data input (D) is loaded into the flip-flop on the High-to-Low clock (C) transition. When the active-Low enable input (T) is Low, the data on the flip-flop output (Q) appears on the O output. When T is High, the output is high impedance (Off).

The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

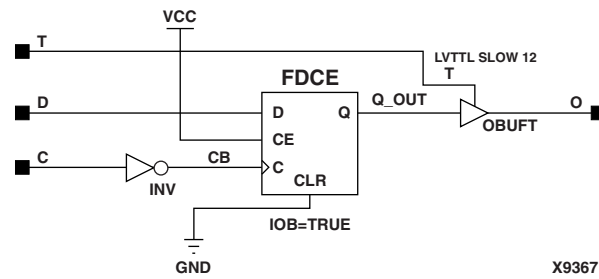
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
T	D	C	O
1	X	X	Z
0	1	↓	1
0	0	↓	0



OFDT_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDT_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

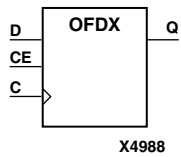
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDT_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDX, 4, 8, 16

Single- and Multiple-Output D Flip-Flops with Clock Enable

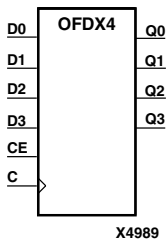
Architectures Supported

OFDX, OFDX4, OFDX8, OFDX16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



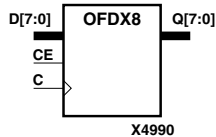
OFDX, OFDX4, OFDX8, and OFDX16 are single and multiple output D flip-flops. The Q outputs are connected to OPADs or IOPADs. The data on the D inputs is loaded into the flip-flops during the Low-to-High clock (C) transition and appears on the Q outputs. When CE is Low, flip-flop outputs do not change.

The flip-flops are asynchronously cleared with Low outputs, when power is applied, or when global reset is active.

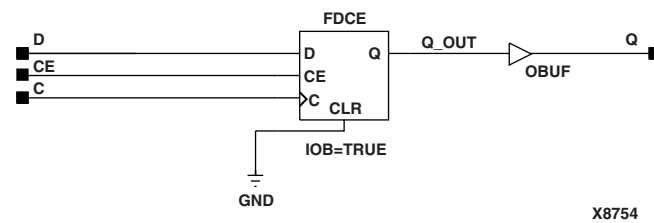
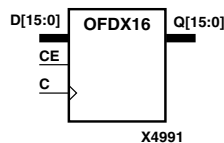


Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

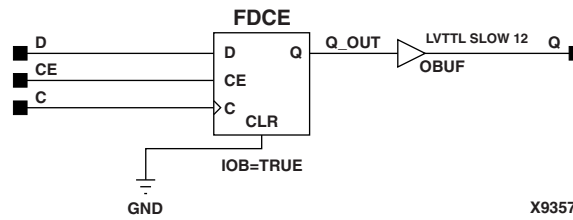
GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



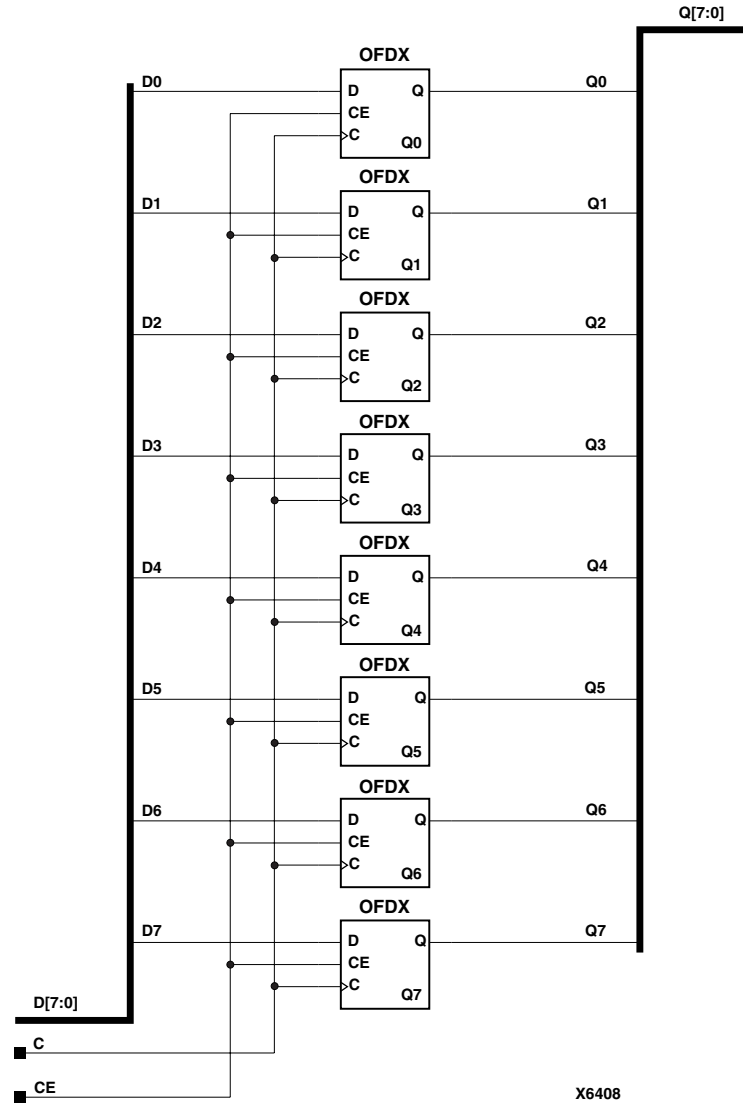
Inputs			Outputs
CE	D	C	Q
1	D _n	↑	D _n
0	X	X	No Chg



OFDX Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDX Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



OFDX8 Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

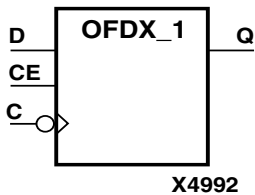
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDX, you would infer an FDCE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDX_1

Output D Flip-Flop with Inverted Clock and Clock Enable

Architectures Supported

OFDX_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



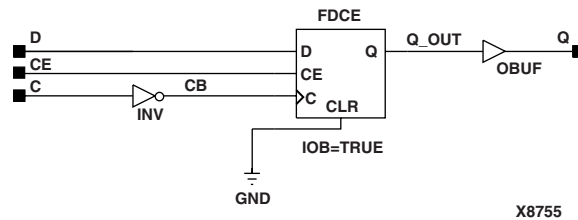
OFDX_1 is located in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When the CE pin is Low, the output (Q) does not change.

The flip-flop is asynchronously cleared with Low output when power is applied, or when global reset is active.

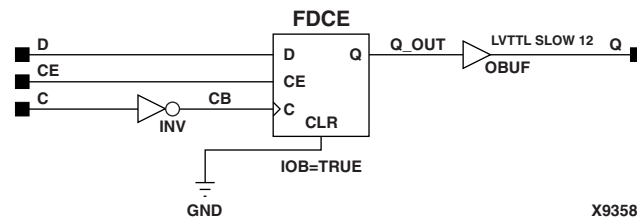
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	D
0	X	X	No Chg



OFDX_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDX_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

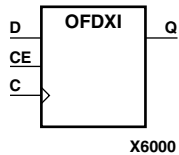
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDX_1, you would infer an FDCE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDXI

Output D Flip-Flop with Clock Enable (Asynchronous Preset)

Architectures Supported

OFDXI	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



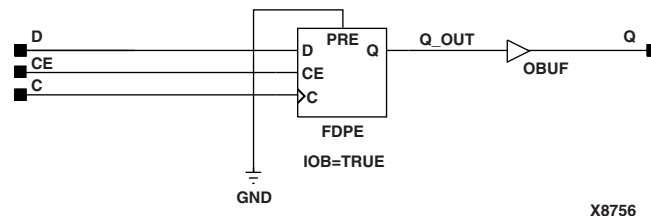
OFDXI is contained in an input/output block (IOB). The output (Q) of the D flip-flop is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the Low-to-High clock (C) transition and appears at the output (Q). When CE is Low, the output does not change.

The flip-flop is asynchronously preset with High output when power is applied, or when global reset is active.

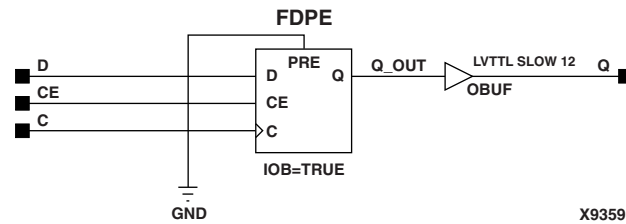
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↑	D
0	X	X	No Chg



OFDXI Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDXI Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

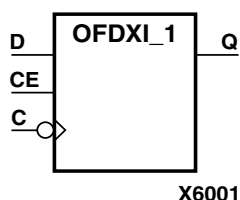
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDXI, you would infer an FDPE and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr o` to pack all output registers into the IOBs.

OFDXI_1

Output D Flip-Flop with Inverted Clock and Clock Enable (Asynchronous Preset)

Architectures Supported

OFDXI_1	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



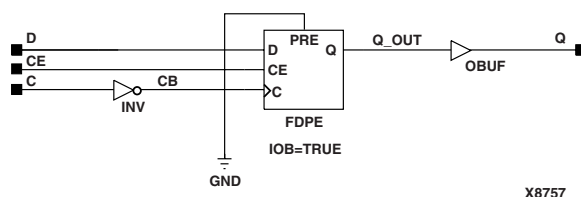
OFDXI_1 is located in an input/output block (IOB). The D flip-flop output (Q) is connected to an OPAD or an IOPAD. The data on the D input is loaded into the flip-flop during the High-to-Low clock (C) transition and appears on the Q output. When CE is Low, the output (Q) does not change.

The flip-flop is asynchronously preset with High output when power is applied, or when global reset is active.

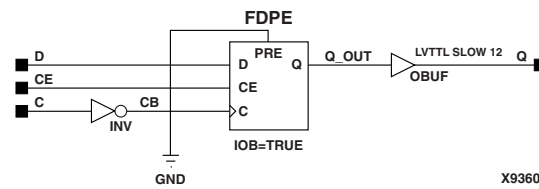
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
CE	D	C	Q
1	D	↓	D
0	X	X	No Chg



OFDXI_1 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OFDXI_1 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

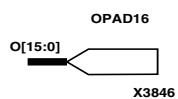
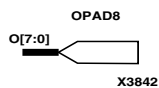
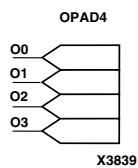
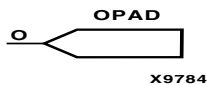
This component is inside the IOB. It cannot be directly inferred. The most common design practice is to infer a regular component and put an IOB=TRUE attribute on the component in the UCF file or in the code. For instance, to get an OFDXI_1, you would infer an FDPE_1 and put the IOB = TRUE attribute on the component. Or, you could use the map option `-pr 0` to pack all output registers into the IOBs.

OPAD, 4, 8, 16

Single- and Multiple-Output Pads

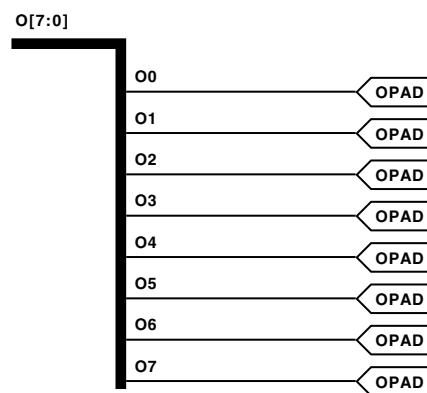
Architectures Supported

OPAD	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OPAD4, OPAD8, OPAD16	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



OPAD, OPAD4, OPAD8, and OPAD16 are single and multiple output pads. An OPAD connects a device pin to an output signal of a PLD. It is internally connected to an input/output block (IOB), which is configured by the software as an OBUF, an OBUFT, an OBUFE, an OFD, or an OFDT.

See the appropriate CAE tool interface user guide for details on assigning pin location and identification.



OPAD8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, it is not necessary to use these elements in the design. They will be added automatically.

Commonly Used Constraints

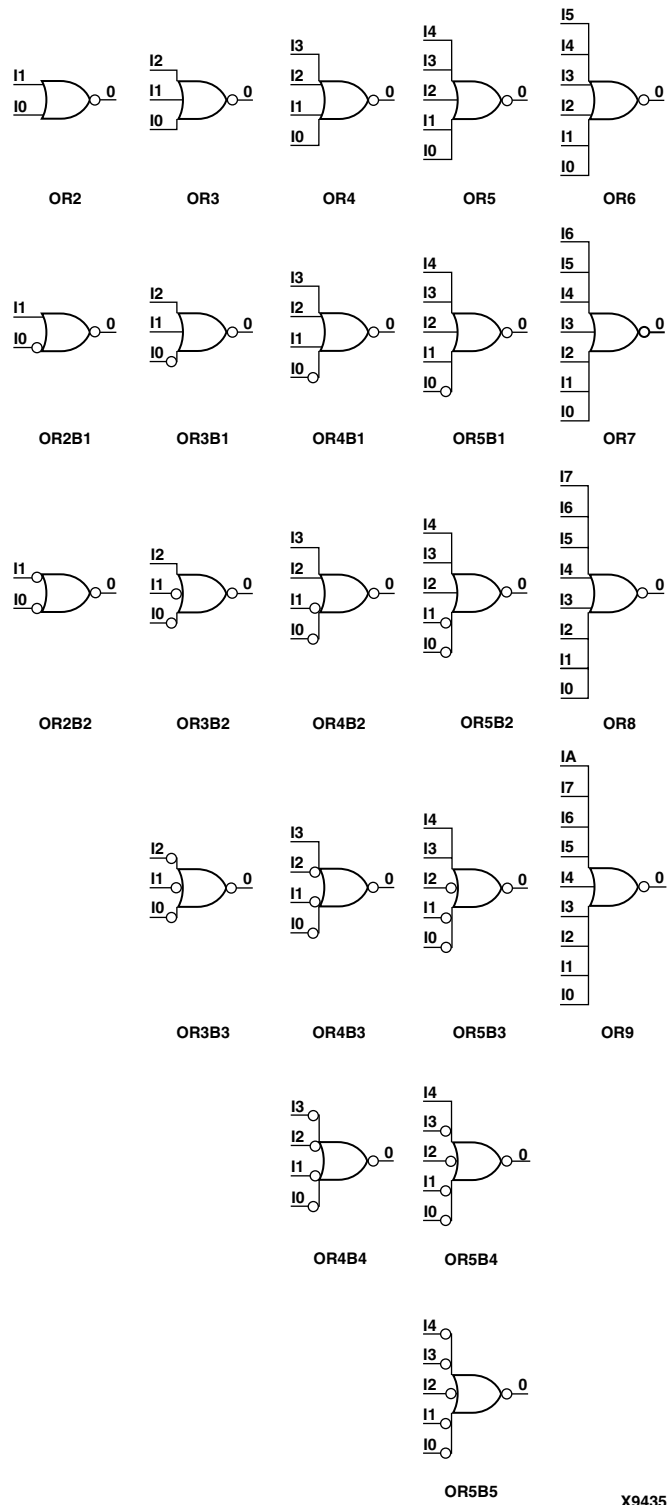
IOBDELAY, PULLDOWN

OR2-9

2- to 9-Input OR Gates with Inverted and Non-Inverted Inputs

Architectures Supported

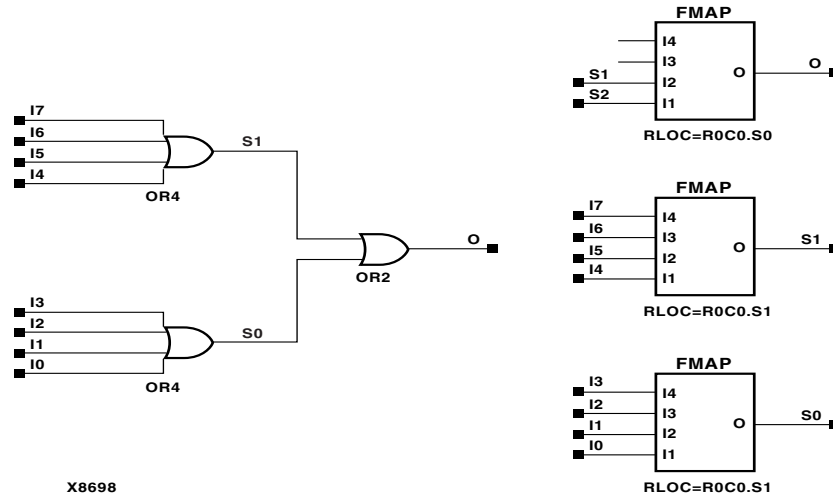
OR2, OR2B1, OR2B2, OR3, OR3B1, OR3B2, OR3B3, OR4, OR4B1, OR4B2, OR4B3, OR4B4	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OR5, OR5B1, OR5B2, OR5B3, OR5B4, OR5B5	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
OR6, OR7, OR8, OR9	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



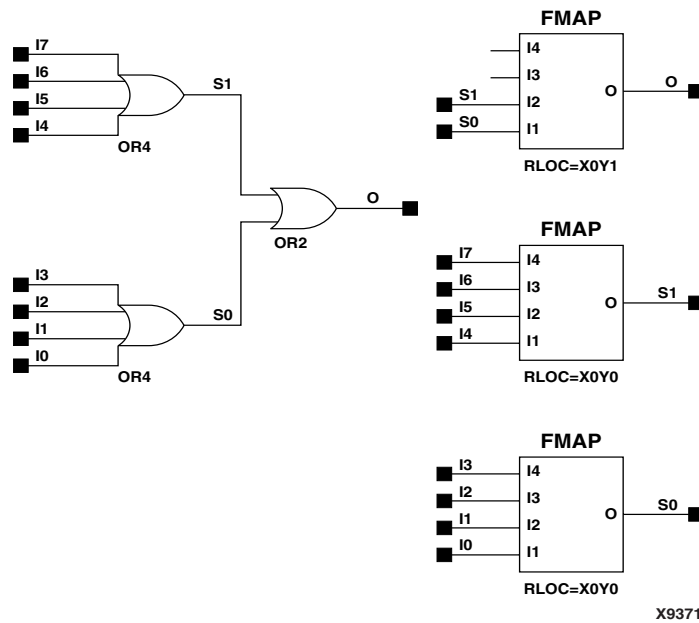
OR Gate Representations

The OR function is performed in the Configurable Logic Block (CLB) function generators for Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II Pro, and Virtex-II Pro X. OR functions of up to five inputs are available in any combination of inverting and non-inverting inputs. OR functions of six to nine inputs are available with only non-inverting inputs. To invert some or all inputs, use external inverters. Since each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.

See "OR2-9" for information on additional OR functions for the Spartan-II, Spartan-IIE, Virtex, and Virtex-E.



OR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OR8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

OR2 through OR5 are primitives that can be inferred or instantiated. OR6 through OR9 are macros which can be inferred.

VHDL Instantiation Template for OR5

Following is the VHDL code for OR5. To instantiate OR2, remove I2, I3, and I4. To instantiate OR3, remove I3 and I4. For OR4, remove I4. OR2B1, and OR2B2 have the same code as OR2. OR3B1, 3B2, and 3B3 have the same code as OR3 etc..

```
-- Component Declaration for OR5 should be placed
-- after architecture statement but before begin keyword

component OR5
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC;
        I3 : in STD_ULOGIC;
        I4: in STD_ULOGIC);
end component;

-- Component Attribute specification for OR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for OR5 should be placed
-- in architecture after the begin keyword

OR5_INSTANCE_NAME : OR5
  port map (O => user_O,
            I0 => user_I0,
            I1 => user_I1,
            I2 => user_I2,
            I3 => user_I3,
            I4 => user_I4);
```

Verilog Instantiation Template for OR5

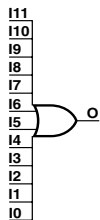
```
OR5 OR5_instance_name (.O (user_O),
                       .I0 (user_I0),
                       .I1 (user_I1),
                       .I2 (user_I2),
                       .I3 (user_I3),
                       .I4 (user_I4));
```

OR12, 16

12- and 16-Input OR Gates with Non-Inverted Inputs

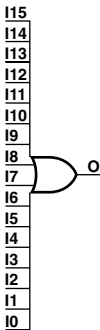
Architectures Supported

OR12, OR16	
Spartan-II, Spartan-IIIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

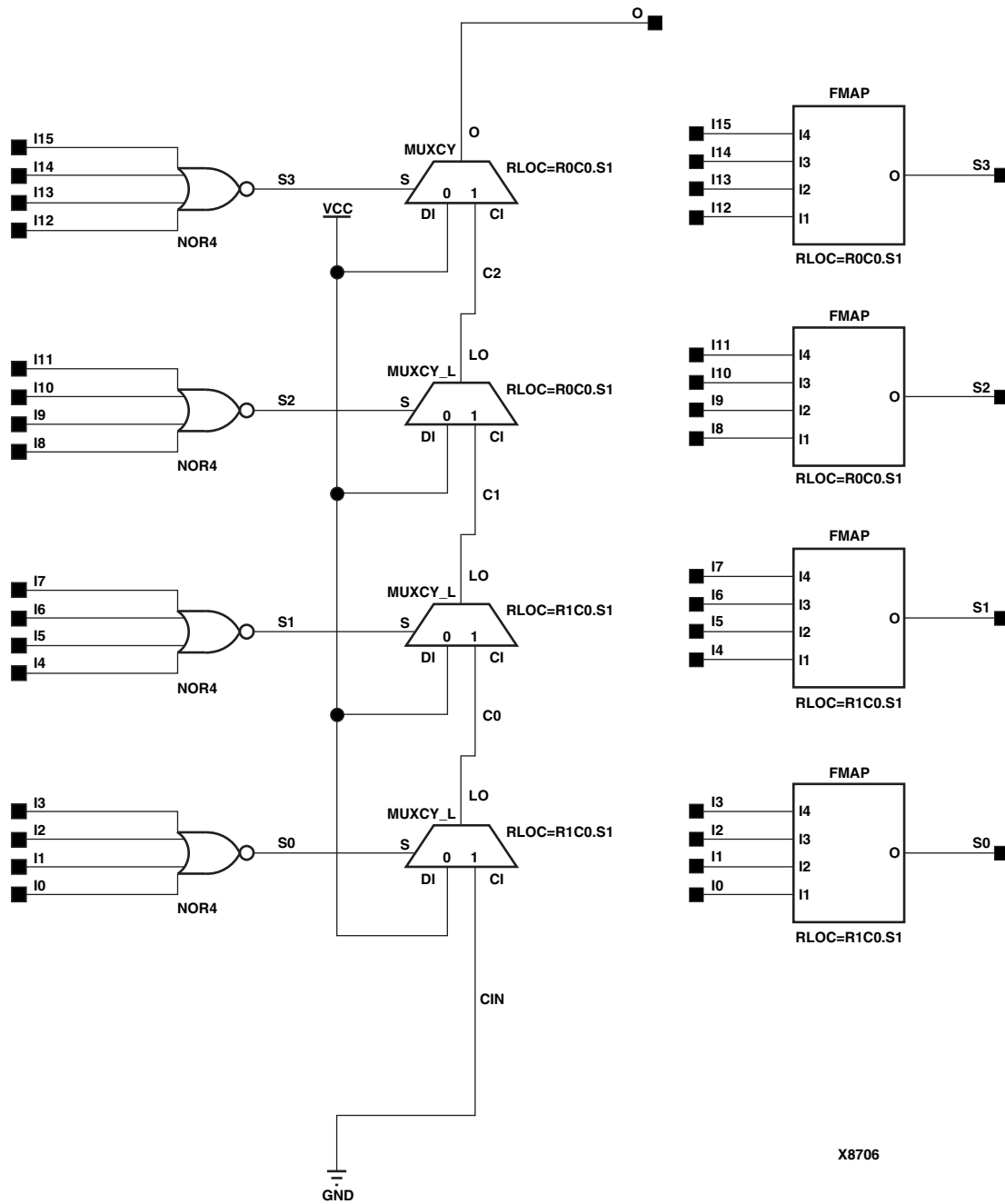


OR12
X9436

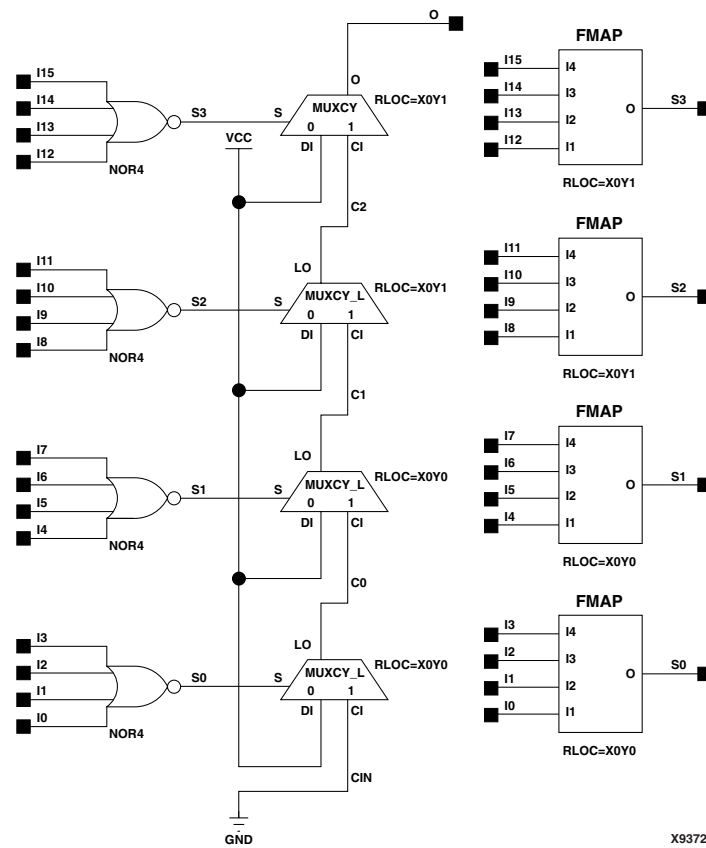
See “OR2-9” for information on OR functions.



OR16
X9437



OR16 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



OR16 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

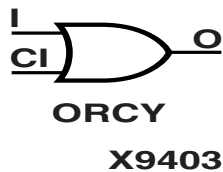
For HDL, OR12 and OR16 are macros that are inferred. See “OR2-9” for information about inferring OR gates.

ORCY

OR with Carry Logic

Architectures Supported

ORCY	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ORCY is a special OR with general O output that generates faster and smaller arithmetic functions.

Each Virtex-II, Virtex-II Pro, and Virtex-II Pro X slice contains a dedicated 2-input OR gate that ORs together carry out values for a series of horizontally adjacent carry chains. The OR gate gets one input external to the slice and the other input from the output of the high order carry mux. The OR gate's output drives the next slice's OR gate horizontally across the die.

Only MUXCY outputs can drive the signal on the CI pin. Only ORCY outputs or logic zero can drive the I pin.

Usage

For HDL, the ORCY design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for ORCY should be placed
-- after architecture statement but before begin keyword

component ORCY
  port (O : out STD_ULOGIC;
        CI : in STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

-- Component Attribute specification for ORCY
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for ORCY should be placed
-- in architecture after the begin keyword

ORCY_INSTANCE_NAME : ORCY
```

```
port map (O => user_O,  
         CI => user_CI,  
         I => user_I);
```

Verilog Instantiation Template

```
ORCY instance_name (.O (user_O),  
                   .CI (user_CI),  
                   .I (user_I));
```

PPC405

Primitive for the Power PC Core

Architectures Supported

PPC405	
Spartan-II, Spartan-III	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive*
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Not supported for Virtex-II. Supported for Virtex-II Pro and Virtex-II Pro X only.	

The PowerPC 405 embedded core is a 32-bit RISC core integrating a PowerPC 405 CPU, separate instruction and data caches, a JTAG port, trace FIFO, multiple timers, and a memory management unit (MMU). Integrated on-chip memory (OCM) controllers provide dedicated interfaces between Block SelectRAM memory and the processor core instruction and data paths for high-speed access. The PowerPC 405 core implements the PowerPC User Instruction Set.

For complete information about the PowerPC 405, see the following documents:

- *Virtex-II Pro Datasheet*
- *Virtex-II Pro Handbook*
- *The PowerPC 405 Core Processor Block Manual*
- *The PowerPC 405 User Guide*

The following table lists the inputs and outputs of the primitive. For detailed information about the pinouts, see the *DS083 Virtex-II Pro Data Sheet*.

Inputs	Outputs
BRAMDSOCMCLK	C405CPMCORESLEEPREQ
BRAMDSOCMRDDBUS [0:31]	C405CPMMSRCE
BRAMISOCMCLK	C405CPMMSREE
BRAMISOCMRDDBUS [0:63]	C405CPMTIMERIRQ
CPMC405CLOCK	C405CPMTIMERRESETRREQ
CPMC405CORECLKINACTIVE	C405DBGMSRWE
CPMC405CPUCLKEN	C405DBGSTOPACK
CPMC405JTAGCLKEN	C405DBGWBCOMPLETE
CPMC405TIMERCLKEN	C405DBGWBFULL
CPMC405TIMERTICK	C405DBGWBIA[0:29]

Inputs	Outputs
DBG405DEBUGHALT	C405DCRABUS [0:9]
DBG405EXTBUSHOLDACK	C405DCRDBUSOUT [0:31]
DBG405UNCONDDEBUGEVENT	C405DCRREAD
DCRC405ACK	C405DCRWRITE
DCRC405DBUSIN [0:31]	C405JTG CAPTUREDR
DSARCVALUE [0:7]	C405JTGEXTTEST
DSCNTLVALUE [0:7]	C405JTGPGMOUT
EICC405CRITINPUTIRQ	C405JTGSHIFTDR
EICC405EXTINPUTIRQ	C405JTGTD0
ISARCVALUE [0:7]	C405JTGTD0EN
ISCNTLVALUE [0:7]	C405JTGUPDATER
JTGC405BNDSCANTDO	C405PLBDCUABORT
JTGC405TCK	C405PLBDCUABUS [0:31]
JTGC405TDI	C405PLBDCUBE [0:7]
JTGC405TMS	C405PLBDCUCACHEABLE
JTGC405TRSTNEG	C405PLBDCUGUARDED
MCBCPUCLKEN	C405PLBDCUPRIORITY [0:1]
MCBJTAGEN	C405PLBDCUREQUEST
MCBTIMEREN	C405PLBDCURNW
MCPPCRST	C405PLBDCUSIZE2
PLBC405DCUADDRACK	C405PLBDCUU0ATTR
PLBC405DCUBUSY	C405PLBDCUWRDBUS [0:63]
PLBC405DCUERR	C405PLBDCUWRITETHRU
PLBC405DCURDDACK	C405PLBICUABORT
PLBC405DCURDDBUS [0:63]	C405PLBICUABUS [0:29]
PLBC405DCURDWDADDR [1:3]	C405PLBICUCACHEABLE
PLBC405DCUSSIZE1	C405PLBICUPRIORITY [0:1]
PLBC405DCUWRDACK	C405PLBICUREQUEST
PLBC405ICUADDRACK	C405PLBICUSIZE [2:3]
PLBC405ICUBUSY	C405PLBICUU0ATTR
PLBC405ICUERR	C405RSTCHIPPRESETREQ
PLBC405ICURDDACK	C405RSTCORERESETREQ
PLBC405ICURDDBUS [0:63]	C405RSTSYSRESETREQ
PLBC405ICURDWDADDR [1:3]	C405TRCCYCLE
PLBC405ICUSSIZE1	C405TRCEVENEXECUTIONSTATUS [0:1]
PLBCLK	C405TRCODDEXECUTIONSTATUS [0:1]
RSTC405RESECHIP	C405TRCTRACESTATUS [0:3]
RSTC405RESECORE	C405TRCTRIGGEREVENTOUT
RSTC405RESESYS	C405TRCTRIGGEREVENTTYPE [0:10]
TIEC405DETERMINISTICMULT	C405XXXMACHINECHECK
TIEC405DISOPERANDFWD	DSOCMBRAMABUS [8:29]

Inputs	Outputs
TIEC405MMUEN	DSOCMBRAMBYTEWRITE [0:3]
TIEDSOCMDCRADDR [0:7]	DSOCMBRAMEN
TIEISOCMDCRADDR [0:7]	DSOCMBRAMWRDBUS [0:31]
TRCC405TRACEDISABLE	DSOCMBUSY
TRCC405TRIGGEREVENTIN	ISOCMBRAMEN
	ISOCMBRAMEVENWRITEEN
	ISOCMBRAMODDWRITEEN
	ISOCMBRAMRDABUS [8:28]
	ISOCMBRAMWRABUS [8:28]
	ISOCMBRAMWRDBUS [0:31]

PPC405

BRAMDSOCCLK	C405CPMCORESLEEPREQ
BRAMDSOCMRDDBUS(0:31)	C405CPMMSRCE
BRAMISOCCLK	C405CPMMSREE
BRAMISOCMRDDBUS(0:63)	C405CPMTIMERIRQ
CPMC405CLOCK	C405CPMTIMERRESETRQ
CPMC405CORECLKINACTIVE	C405DBGMSRWE
CPMC405CPUCLKEN	C405DBGSTOPACK
CPMC405JTAGCLKEN	C405DBGWBCOMplete
CPMC405TIMERCLKEN	C405DBGWBFULL
CPMC405TIMERTICK	C405DBGWBIAR(0:29)
DBGC405DEBUGHALT	C405DCRABUS(0:9)
DBGC405EXTBUSHOLDACK	C405DCRDBUSOUT(0:31)
DBGC405UNCONDDEBUgeVENT	C405DCRREAD
DCRC405ACK	C405DCRWRITE
DCRC405DBUSIN(0:31)	C405JTGCAPTUREDR
DSARCVALUE(0:7)	C405JTGEXTEST
DSCNTLVALUE(0:7)	C405JTGPGMOUT
EICC405CRITINPUTIRQ	C405JTGSHIFTR
EICC405EXTINPUTIRQ	C405JTGDO
ISARCVALUE(0:7)	C405JTGDOEN
ISCNTLVALUE(0:7)	C405JTGUPDATER
JTGC405BNDSCANTDO	C405PLBDCUABORT
JTGC405TCK	C405PLBDCUABUS(0:31)
JTGC405TDI	C405PLBDCUBE(0:7)
JTGC405TMS	C405PLBDCUCACHEABLE
JTGC405TRSTNEG	C405PLBDCUGUARDED
MCBCPUCLKEN	C405PLBDCUPRIORITY(0:1)
MCBJTAGEN	C405PLBDCUREQUEST
MCBTIMEREN	C405PLBDCURNW
MCPPCRST	C405PLBDCUSIZE2
PLBC405DCUADDRACK	C405PLBDCUUOATTR
PLBC405DCUBUSY	C405PLBDCUWRDDBUS(0:63)
PLBC405DCUERR	C405PLBDCUWRITETHRU
PLBC405DCURDDACK	C405PLBICUABORT
PLBC405DCURDDBUS(0:63)	C405PLBICUABUS(0:29)
PLBC405DCURDWDADDR(1:3)	C405PLBICUACHEABLE
PLBC405DCUSSIZE1	C405PLBICUPRIORITY(0:1)
PLBC405DCUWRDACK	C405PLBICUREQUEST
PLBC405SICUADDRACK	C405PLBICUSIZE(2:3)
PLBC405SICUBUSY	C405PLBICUUOATTR
PLBC405SICUERR	C405RSTCHIPPRESETRQ
PLBC405SICURDDACK	C405RSTCORERESETRQ
PLBC405SICURDDBUS(0:63)	C405RSTSYSRESETRQ
PLBC405SICURDWDADDR(1:3)	C405TRCCYCLE
PLBC405SICUSSIZE1	C405TRCEVENEXECUTIONSTATUS(0:1)
PLBCLK	C405TRCODDEXECUTIONSTATUS(0:1)
RSTC405RESETRCHIP	C405TRCTRACESTATUS(0:3)
RSTC405RESETRCORE	C405TRCTRIGGEREVENTOUT
RSTC405RESETRSYS	C405TRCTRIGGEREVENTTYPE(0:10)
TIEC405DETERMINISTICMULT	C405XXXMACHINECHECK
TIEC405DISOPERANDFWD	DSOCMBRAMABUS(8:29)
TIEC405MMUEN	DSOCMBRAMBYTEWRITE(0:3)
TIEDSOCMDCRADDR(0:7)	DSOCMBRAMEN
TIEISOCMDCRADDR(0:7)	DSOCMBRAMWRDDBUS(0:31)
TRCC405TRACEDISABLE	DSOCMBUSY
TRCC405TRIGGEREVENTIN	ISOCMBRAMEN
	ISOCMBRAMEVENWRITEEN
	ISOCMBRAMODDWRITEEN
	ISOCMBRAMRDABUS(8:28)
	ISOCMBRAMWRABUS(8:28)
	ISOCMBRAMWRDDBUS(0:31)

X9929

Usage

For HDL, the PPC405 design element is instantiated rather than inferred.

VHDL Instantiation Template

```
-- Component Declaration for PPC405 should be placed
-- after architecture statement but before begin keyword

component PPC405

    port (C405CPMCORESLEEPPREQ      : out STD_ULONGIC;
          C405CPMMSRCE              : out STD_ULONGIC;
          C405CPMMSREE              : out STD_ULONGIC;
          C405CPMTIMERIRQ           : out STD_ULONGIC;
          C405CPMTIMERRESETREQ      : out STD_ULONGIC;
          C405DBGMSRWE              : out STD_ULONGIC;
          C405DBGSTOPACK            : out STD_ULONGIC;
          C405DBGWBCOMPLETE         : out STD_ULONGIC;
          C405DBGWBFULL             : out STD_ULONGIC;
          C405DBGWBIAR              : out STD_LOGIC_VECTOR (29 downto
0);
          C405DCRABUS                : out STD_LOGIC_VECTOR (9 downto 0);
          C405DCRDBUSOUT            : out STD_LOGIC_VECTOR (31 downto
0);
          C405DCRREAD                : out STD_ULONGIC;
          C405DCRWRITE              : out STD_ULONGIC;
          C405JTGCAPTUREDR          : out STD_ULONGIC;
          C405JTGEXTTEST            : out STD_ULONGIC;
          C405JTGPGMOUT             : out STD_ULONGIC;
          C405JTGSHIFTDR            : out STD_ULONGIC;
          C405JTGTD0                : out STD_ULONGIC;
          C405JTGTD0EN              : out STD_ULONGIC;
          C405JTGUPDATEDR           : out STD_ULONGIC;
          C405PLBDCUABORT           : out STD_ULONGIC;
          C405PLBDCUABUS            : out STD_LOGIC_VECTOR (31 downto
0);
          C405PLBDCUBE              : out STD_LOGIC_VECTOR (7 downto 0);
          C405PLBDCUCACHEABLE       : out STD_ULONGIC;
          C405PLBDCUGUARDED         : out STD_ULONGIC;
          C405PLBDCUPRIORITY        : out STD_LOGIC_VECTOR (1 downto 0);
          C405PLBDCUREQUEST         : out STD_ULONGIC;
          C405PLBDCURNW             : out STD_ULONGIC;
          C405PLBDCUSIZE2           : out STD_ULONGIC;
          C405PLBDCUU0ATTR          : out STD_ULONGIC;
          C405PLBDCUWRDBUS          : out STD_LOGIC_VECTOR (63 downto
0);
          C405PLBDCUWRITETHRU       : out STD_ULONGIC;
          C405PLBICUABORT           : out STD_ULONGIC;
          C405PLBICUABUS            : out STD_LOGIC_VECTOR (29 downto
0);
          C405PLBICUCACHEABLE       : out STD_ULONGIC;
          C405PLBICUPRIORITY        : out STD_LOGIC_VECTOR (1 downto 0);
          C405PLBICUREQUEST         : out STD_ULONGIC;
          C405PLBICUSIZE            : out STD_LOGIC_VECTOR (3 downto 2);
          C405PLBICUU0ATTR          : out STD_ULONGIC;
          C405RSTCHIPRESETREQ       : out STD_ULONGIC;
          C405RSTCORERESETREQ       : out STD_ULONGIC;
```

```

C405RSTSYSRESETREQ      : out STD_ULONGIC;
C405TRCCYCLE            : out STD_ULONGIC;
C405TRCEVENEXECUTIONSTATUS: out STD_LOGIC_VECTOR (1 downto
0);
C405TRCODDEXECUTIONSTATUS : out STD_LOGIC_VECTOR (1 downto
0);
C405TRCTRACESTATUS      : out STD_LOGIC_VECTOR (3 downto
0);
C405TRCTRIGGEREVENTOUT  : out STD_ULONGIC;
C405TRCTRIGGEREVENTTYPE : out STD_LOGIC_VECTOR (10 downto
0);
C405XXXMACHINECHECK     : out STD_ULONGIC;
DSOCMBRAMABUS           : out STD_LOGIC_VECTOR (29 downto
8);
DSOCMBRAMBYTEWRITE      : out STD_LOGIC_VECTOR (3 downto
0);
DSOCMBRAMEN             : out STD_ULONGIC;
DSOCMBRAMWRDBUS        : out STD_LOGIC_VECTOR (31 downto
0);
DSOCMBUSY               : out STD_ULONGIC;
ISOCMBRAMEN             : out STD_ULONGIC;
ISOCMBRAMEVENWRITEEN    : out STD_ULONGIC;
ISOCMBRAMODDWRITEEN     : out STD_ULONGIC;
ISOCMBRAMRDABUS        : out STD_LOGIC_VECTOR (28 downto
8);
ISOCMBRAMWRABUS        : out STD_LOGIC_VECTOR (28 downto
8);
ISOCMBRAMWRDBUS        : out STD_LOGIC_VECTOR (31 downto
0);
BRAMDSOCCLK             : in STD_ULONGIC;
BRAMDSOCMRDDBUS        : in STD_LOGIC_VECTOR (31 downto
0);
BRAMISOCCLK             : in STD_ULONGIC;
BRAMISOCMRDDBUS        : in STD_LOGIC_VECTOR (63 downto
0);
CPMC405CLOCK            : in STD_ULONGIC;
CPMC405CORECLKINACTIVE  : in STD_ULONGIC;
CPMC405CPUCLKEN         : in STD_ULONGIC;
CPMC405JTAGCLKEN        : in STD_ULONGIC;
CPMC405TIMERCLKEN       : in STD_ULONGIC;
CPMC405TIMERTICK        : in STD_ULONGIC;
DBG405DEBUGHALT         : in STD_ULONGIC;
DBG405EXTBUSHOLDACK     : in STD_ULONGIC;
DBG405UNCONDDEBUGEVENT  : in STD_ULONGIC;
DCRC405ACK               : in STD_ULONGIC;
DCRC405DBUSIN           : in STD_LOGIC_VECTOR (31 downto
0);
DSARCVALUE              : in STD_LOGIC_VECTOR (7 downto 0);
DSCNTLVALUE             : in STD_LOGIC_VECTOR (7 downto 0);
EICC405CRITINPUTIRQ     : in STD_ULONGIC;
EICC405EXTINPUTIRQ      : in STD_ULONGIC;
ISARCVALUE              : in STD_LOGIC_VECTOR (7 downto 0);
ISCNTLVALUE             : in STD_LOGIC_VECTOR (7 downto 0);
JTGC405BNDSCANTDO       : in STD_ULONGIC;
JTGC405TCK              : in STD_ULONGIC;
JTGC405TDI              : in STD_ULONGIC;
JTGC405TMS              : in STD_ULONGIC;
JTGC405TRSTNEG          : in STD_ULONGIC;

```

```

MCBCPUCLKEN           : in STD_ULOGIC;
MCBJTAGEN             : in STD_ULOGIC;
MCBTIMEREN           : in STD_ULOGIC;
MCPPCRST             : in STD_ULOGIC;
PLBC405DCUADDRACK    : in STD_ULOGIC;
PLBC405DCUBUSY       : in STD_ULOGIC;
PLBC405DCUERR        : in STD_ULOGIC;
PLBC405DCURDDACK     : in STD_ULOGIC;
PLBC405DCURDDBUS     : in STD_LOGIC_VECTOR (63 downto
0);

PLBC405DCURDWDADDR   : in STD_LOGIC_VECTOR (3 downto 1);
PLBC405DCUSSIZE1     : in STD_ULOGIC;
PLBC405DCUWRDACK     : in STD_ULOGIC;
PLBC405ICUADDRACK    : in STD_ULOGIC;
PLBC405ICUBUSY       : in STD_ULOGIC;
PLBC405ICUERR        : in STD_ULOGIC;
PLBC405ICURDDACK     : in STD_ULOGIC;
PLBC405ICURDDBUS     : in STD_LOGIC_VECTOR (63 downto
0);

PLBC405ICURDWDADDR   : in STD_LOGIC_VECTOR (3 downto 1);
PLBC405ICUSSIZE1     : in STD_ULOGIC;
PLBCLK                : in STD_ULOGIC;
RSTC405RESETCHIP     : in STD_ULOGIC;
RSTC405RESETCORE     : in STD_ULOGIC;
RSTC405RESETSYS      : in STD_ULOGIC;
TIEC405DETERMINISTICMULT : in STD_ULOGIC;
TIEC405DISOPERANDFWD : in STD_ULOGIC;
TIEC405MMUEN         : in STD_ULOGIC;
TIEDSOCMDCRADDR      : in STD_LOGIC_VECTOR (7 downto 0);
TIEISOCMDCRADDR      : in STD_LOGIC_VECTOR (7 downto 0);
TRCC405TRACEDISABLE  : in STD_ULOGIC;
TRCC405TRIGGEREVENTINE : in STD_ULOGIC);
end component;

-- Component Attribute specification for PPC405
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for PPC405 should be placed
-- in architecture after the begin keyword

PPC405_INSTANCE_NAME : PPC405
  port map (C405CPMCORESLEEPREQ => user_C405CPMCORESLEEPREQ,
            C405CPMMSRCE       => user_C405CPMMSRCE,
            C405CPMMSREE       => user_C405CPMMSREE,
            C405CPMTIMERIRQ    => user_C405CPMTIMERIRQ,
            C405CPMTIMERRESETREQ => user_C405CPMTIMERRESETREQ,
            C405DBGMSRWE       => user_C405DBGMSRWE,
            C405DBGSTOPACK     => user_C405DBGSTOPACK,
            C405DBGWBCOMPLETE  => user_C405DBGWBCOMPLETE,
            C405DBGWBFULL      => user_C405DBGWBFULL,
            C405DBGWBIAR       => user_C405DBGWBIAR,
            C405DCRABUS        => user_C405DCRABUS,
            C405DCRDBUSOUT     => user_C405DCRDBUSOUT,
            C405DCRREAD        => user_C405DCRREAD,

```

```

C405DCRWRITE => user_C405DCRWRITE,
C405JTGCAPTUREDR => user_C405JTGCAPTUREDR,
C405JTGEXTTEST => user_C405JTGEXTTEST,
C405JTGPGMOUT => user_C405JTGPGMOUT,
C405JTGSHIFTDR => user_C405JTGSHIFTDR,
C405JTGTDO => user_C405JTGTDO,

C405JTGTDOEN => user_C405JTGTDOEN,
C405JTGUPDATER => user_C405JTGUPDATER,
C405PLBDCUABORT => user_C405PLBDCUABORT,
C405PLBDCUABUS => user_C405PLBDCUABUS,
C405PLBDCUBE => user_C405PLBDCUBE,
C405PLBDCUCACHEABLE => user_C405PLBDCUCACHEABLE,
C405PLBDCUGUARDED => user_C405PLBDCUGUARDED,
C405PLBDCUPRIORITY => user_C405PLBDCUPRIORITY,
C405PLBDCUREQUEST => user_C405PLBDCUREQUEST,
C405PLBDCURNW => user_C405PLBDCURNW,
C405PLBDCUSIZE2 => user_C405PLBDCUSIZE2,
C405PLBDCUU0ATTR => user_C405PLBDCUU0ATTR,
C405PLBDCUWRDBUS => user_C405PLBDCUWRDBUS,
C405PLBDCUWRITETHRU => user_C405PLBDCUWRITETHRU,
C405PLBICUABORT => user_C405PLBICUABORT,
C405PLBICUABUS => user_C405PLBICUABUS,
C405PLBICUCACHEABLE => user_C405PLBICUCACHEABLE,
C405PLBICUPRIORITY => user_C405PLBICUPRIORITY,
C405PLBICUREQUEST => user_C405PLBICUREQUEST,
C405PLBICUSIZE => user_C405PLBICUSIZE,
C405PLBICUU0ATTR => user_C405PLBICUU0ATTR,
C405RSTCHIPRESETREQ => user_C405RSTCHIPRESETREQ,
C405RSTCORERESETREQ => user_C405RSTCORERESETREQ,
C405RSTSYSRESETREQ => user_C405RSTSYSRESETREQ,
C405TRCCYCLE => user_C405TRCCYCLE,
C405TRCEVENEXECUTIONSTATUS =>
user_C405TRCEVENEXECUTIONSTATUS,
C405TRCODDEXECUTIONSTATUS =>
user_C405TRCODDEXECUTIONSTATUS,
C405TRCTRACESTATUS => user_C405TRCTRACESTATUS,
C405TRCTRIGGEREVENTOUT =>
user_C405TRCTRIGGEREVENTOUT,
C405TRCTRIGGEREVENTTYPE =>
user_C405TRCTRIGGEREVENTTYPE,
C405XXXMACHINECHECK => user_C405XXXMACHINECHECK,
DSOCMBRAMABUS => user_DSOCMBRAMABUS,
DSOCMBRAMBYTEWRITE => user_DSOCMBRAMBYTEWRITE,
DSOCMBRAMEN => user_DSOCMBRAMEN,
DSOCMBRAMWRDBUS => user_DSOCMBRAMWRDBUS,
DSOCMBUSY => user_DSOCMBUSY,
ISOCMBRAMEN => user_ISOCMBRAMEN,
ISOCMBRAMEVENWRITEEN => user_ISOCMBRAMEVENWRITEEN,
ISOCMBRAMODDWRITEEN => user_ISOCMBRAMODDWRITEEN,
ISOCMBRAMRDABUS => user_ISOCMBRAMRDABUS,
ISOCMBRAMWRABUS => user_ISOCMBRAMWRABUS,
ISOCMBRAMWRDBUS => user_ISOCMBRAMWRDBUS,
BRAMDSOCCLK => user_BRAMDSOCCLK,
BRAMDSOCMRDDBUS => user_BRAMDSOCMRDDBUS,
BRAMISOCCLK => user_BRAMISOCCLK,
BRAMISOCMRDDBUS => user_BRAMISOCMRDDBUS,
CPMC405CLOCK => user_CPMC405CLOCK,

```


Verilog Instantiation Template

```

PPC405 instance_name (.C405CPMCORESLEEPREQ
(user_C405CPMCORESLEEPREQ),

    .C405CPMMSRCE (user_C405CPMMSRCE),
    .C405CPMMSREE (user_C405CPMMSREE),
    .C405CPMTIMERIRQ (user_C405CPMTIMERIRQ),
    .C405CPMTIMERRESETRREQ
(user_ C405CPMTIMERRESETRREQ),
    .C405DBGMSRWE (user_C405DBGMSRWE),
    .C405DBGSTOPACK (user_C405DBGSTOPACK),
    .C405DBGWBCOMPLETE (user_C405DBGWBCOMPLETE),
    .C405DBGWBFULL (user_C405DBGWBFULL),
    .C405DBGWBIAR (user_C405DBGWBIAR),
    .C405DCRABUS (user_C405DCRABUS),
    .C405DCRDBUSOUT (user_C405DCRDBUSOUT),
    .C405DCRREAD (user_C405DCRREAD),
    .C405DCRWRITE (user_C405DCRWRITE),
    .C405JTG CAPTUREDR (user_C405JTG CAPTUREDR),
    .C405JTGEXTTEST (user_C405JTGEXTTEST),
    .C405JTGPGMOUT (user_C405JTGPGMOUT),
    .C405JTGSHIFTR (user_C405JTGSHIFTR),
    .C405JTGTD0 (user_C405JTGTD0),
    .C405JTGTD0EN (user_C405JTGTD0EN),
    .C405JTGUPDR (user_C405JTGUPDR),
    .C405PLBDCUABORT (user_C405PLBDCUABORT),
    .C405PLBDCUABUS (user_C405PLBDCUABUS),
    .C405PLBDCUCACHEABLE (user_C405PLBDCUCACHEABLE),
    .C405PLBDCUGUARDED (user_C405PLBDCUGUARDED),
    .C405PLBDCUPRIORITY (user_C405PLBDCUPRIORITY),
    .C405PLBDCUREQUEST (user_C405PLBDCUREQUEST),
    .C405PLBDCURNW (user_C405PLBDCURNW),
    .C405PLBDCUSIZE2 (user_C405PLBDCUSIZE2),
    .C405PLBDCUU0ATTR (user_C405PLBDCUU0ATTR),
    .C405PLBDCUWRDBUS (user_C405PLBDCUWRDBUS),
    .C405PLBDCUWRITETHRU (user_C405PLBDCUWRITETHRU),
    .C405PLBICUABORT (user_C405PLBICUABORT),
    .C405PLBICUABUS (user_C405PLBICUABUS),
    .C405PLBICUCACHEABLE (user_C405PLBICUCACHEABLE),
    .C405PLBICUPRIORITY (user_C405PLBICUPRIORITY),
    .C405PLBICUREQUEST (user_C405PLBICUREQUEST),
    .C405PLBICUSIZE (user_C405PLBICUSIZE),
    .C405PLBICUU0ATTR (user_C405PLBICUU0ATTR),
    .C405RSTCHIPPRESETREQ (user_C405RSTCHIPPRESETREQ),
    .C405XXXMACHINECHECK (user_C405XXXMACHINECHECK),
    .DSOCMBRAMABUS (user_DSOCMBRAMABUS),
    .DSOCMBRAMBYTEWRITE (user_DSOCMBRAMBYTEWRITE),
    .DSOCMBRAMEN (user_DSOCMBRAMEN),
    .DSOCMBRAMWRDBUS (user_DSOCMBRAMWRDBUS),
    .DSOCMBUSY (user_DSOCMBUSY),
    .ISOCMBRAMEN (user_ISOCMBRAMEN),
    .ISOCMBRAMEVENWRITEEN
(user_ISOCMBRAMEVENWRITEEN),
    .ISOCMBRAMODDWRITEEN (user_ISOCMBRAMODDWRITEEN),
    .ISOCMBRAMRDABUS (user_ISOCMBRAMRDABUS),
    .ISOCMBRAMWRABUS (user_ISOCMBRAMWRABUS),

```

```

        .ISOCMBRAMWRDBUS (user_ISOCMBRAMWRDBUS),
        .BRAMDSOCMCLK (user_BRAMDSOCMCLK),
        .BRAMDSOCMRDDBUS (user_BRAMDSOCMRDDBUS),
        .BRAMISOCMCLK (user_BRAMISOCMCLK),
        .BRAMISOCMRDDBUS (user_BRAMISOCMRDDBUS),
        .CPMC405CLOCK (user_CPMC405CLOCK),
        .CPMC405CORECLKINACTIVE
(user_CPMC405CORECLKINACTIVE),
        .CPMC405CPUCLKEN (user_CPMC405CPUCLKEN),
        .CPMC405JTAGCLKEN (user_CPMC405JTAGCLKEN),
        .CPMC405TIMERCLKEN (user_CPMC405TIMERCLKEN),
        .CPMC405TIMERTICK (user_CPMC405TIMERTICK),
        .DBG405DEBUGHALT (user_DBG405DEBUGHALT),
        .DBG405EXTBUSHOLDACK
(user_DBG405EXTBUSHOLDACK),
        .DBG405UNCONDDEBUGEVENT
(user_DBG405UNCONDDEBUGEVENT),
        .DCRC405ACK (user_DCRC405ACK),
        .DCRC405DBUSIN (user_DSARCVALUE),
        .DSCNTLVALUE (user_DSCNTLVALUE),
        .EICC405CRITINPUTIRQ (user_EICC405CRITINPUTIRQ),
        .EICC405EXTINPUTIRQ (user_EICC405EXTINPUTIRQ),
        .ISARCVALUE (user_ISARCVALUE),
        .ISCNTLVALUE (user_ISCNTLVALUE),
        .JTGC405BNDESCANTDO (user_JTGC405BNDESCANTDO),
        .JTGC405TCK (user_JTGC405TCK),
        .JTGC405TDI (user_JTGC405TDI),
        .JTGC405TMS (user_JTGC405TMS),
        .JTGC405TRSTNEG (user_JTGC405TRSTNEG),
        .MCBCPUCLKEN (user_MCBCPUCLKEN),
        .MCBJTAGEN (user_MCBJTAGEN),
        .MCBTIMEREN (user_MCBTIMEREN),
        .MCPPCRST (user_MCPPCRST),
        .PLBC405DCUADDRACK (user_PLBC405DCUADDRACK),
        .PLBC405DCUBUSY (user_PLBC405DCUBUSY),
        .PLBC405DCUERR (user_PLBC405DCUERR),
        .PLBC405DCURDDACK (user_PLBC405DCURDDACK),
        .PLBC405DCURDDBUS (user_PLBC405DCURDDBUS),
        .PLBC405DCURDWDADDR (user_PLBC405DCURDWDADDR),
        .PLBC405DCUSSIZE1 (user_PLBC405DCUSSIZE1),
        .PLBC405DCUWRDACK (user_PLBC405DCUWRDACK),
        .PLBC405ICUADDRACK (user_PLBC405ICUADDRACK),
        .PLBC405ICUBUSY (user_PLBC405ICUBUSY),
        .PLBC405ICUERR (user_PLBC405ICUERR),
        .PLBC405ICURDDACK (user_PLBC405ICURDDACK),
        .PLBC405ICURDDBUS (user_PLBC405ICURDDBUS),
        .PLBC405ICURDWDADDR (user_PLBC405ICURDWDADDR),
        .PLBC405ICUSSIZE1 (user_PLBC405ICUSSIZE1),
        .PLBCLK (user_PLBCLK),
        .RSTC405RESETCORE (user_RSTC405RESETCORE),
        .RSTC405RESETCORE (user_RSTC405RESETCORE),
        .RSTC405RESETSYS (user_RSTC405RESETSYS),
        .TIEC405DETERMINISTICMULT
(user_TIEC405DETERMINISTICMULT),
        .TIEC405DISOPERANDFWD
(user_TIEC405DISOPERANDFWD),
        .TIEC405MMUEN (user_TIEC405MMUEN),
        .TIEDSOCMDCRADDR (user_TIEDSOCMDCRADDR),

```

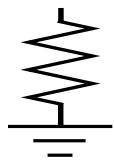
```
.TIEISOCMDCRADDR (user_TIEISOCMDCRADDR),  
.TRCC405TRACEDISABLE (user_TRCC405TRACEDISABLE),  
.TRCC405TRIGGEREVENTINE  
(user_TRCC405TRIGGEREVENTINE);
```


PULLDOWN

Resistor to GND for Input Pads

Architectures Supported

PULLDOWN	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X3860

PULLDOWN resistor elements are connected to input, output, or bidirectional pads to guarantee a logic Low level for nodes that might float.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- Component Declaration for PULLDOWN should be placed
-- after architecture statement but before begin keyword
-- <Cut code below this line and paste into the architecture body>

-- PULLDOWN: I/O Buffer Weak Pull-down
--           All FPGA, CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

PULLDOWN_inst : PULLDOWN
port map (
    O => O    -- Pulldown output (connect directly to top-level port)
);

-- End of PULLDOWN_inst instantiation
```

Verilog Instantiation Template

```
// PULLDOWN: I/O Buffer Weak Pull-down
//           All FPGA, CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide

PULLDOWN PULLDOWN_inst (
    .O(O),    // Pulldown output (connect directly to
             // top-level port)
);

// End of PULLDOWN_inst instantiation
```


PULLUP

Resistor to VCC for Input PADS, Open-Drain, and 3-State Outputs

Architectures Supported

PULLUP	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	Primitive*
CoolRunner-II	Primitive
* Supported only on input-only pins for CoolRunner XPLA3.	



X3861

The pull-up elements establish a High logic level for open-drain elements and macros (DECODE, WAND, WORAND) or 3-state nodes (TBUF) when all the drivers are off.

The buffer outputs are connected together as a wired-AND to form the output (O). When all the inputs are High, the output is off. To establish an output High level, a PULLUP resistor(s) is tied to output (O). One PULLUP resistor uses the least power, two pull-up resistors achieve the fastest Low-to-High speed.

To indicate two PULLUP resistors, append a DOUBLE parameter to the pull-up symbol attached to the output (O) node. See the appropriate CAE tool interface user guide for details.

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Template

```
-- PULLUP: I/O Buffer Weak Pull-up
--           All FPGA, CoolRunner-II
-- The current version of the Xilinx HDL Libraries Guide

PULLUP_inst : PULLUP
port map (
    O => O      -- Pullup output (connect directly to top-level port)
);

-- End of PULLUP_inst instantiation
```

Verilog Instantiation Template

```
// PULLUP: I/O Buffer Weak Pull-up
//           All FPGA, CoolRunner-II
// The current version of the Xilinx HDL Libraries Guide
```

```
PULLUP PULLUP_inst (  
    .O(0),      // Pullup output (connect directly to top-level port)  
);  
  
// End of PULLUP_inst instantiation
```

Commonly Used Constraints

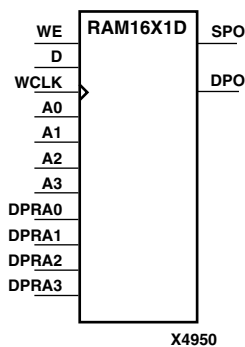
DOUBLE, HBLKNM

RAM16X1D

16-Deep by 1-Wide Static Dual Port Synchronous RAM

Architectures Supported

RAM16X1D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X1D is a 16-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Specifying Initial Contents of a RAM

You can use the INIT attribute to specify an initial value directly on the symbol if the RAM is 1 bit wide and 16, 32, 64, or 128 bits deep. The value must be a hexadecimal number, for example, INIT=ABAC. If the INIT attribute is not specified, the RAM is initialized with zero.

For Virtex, Virtex-E, Spartan-II, and Spartan-IIE, lower INIT values get mapped to the G function generator and upper INIT values get mapped to the F function generator.

See the "INIT" section of the *Constraints Guide* for more information on the INIT attribute.

For Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, wide RAMs (2, 4, and 8-bit wide single port synchronous RAMs with a WCLK) can also be initialized. These RAMs, however, require INIT_xx attributes. See [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in the RAM16X2S section for more information on initializing Virtex-II wide RAM.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X1D: 16 x 1 positive edge write, asynchronous read dual-port
-- distributed RAM
--           All FPGAs
-- The current version of the Xilinx HDL Libraries Guide

RAM16X1D_inst : RAM16X1D
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT: bit-vector := X"0000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    WCLK => WCLK,   -- Port A write clock input
    WE => WE        -- Port A write enable input
);

-- End of RAM16X1D_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM16X1D RAM16X1D_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
    .DPRA2(DPRA2), // Port B address[2] input bit
    .DPRA3(DPRA3), // Port B address[3] input bit
    .WCLK(WCLK),   // Port A write clock input
    .WE(WE)        // Port A write enable input
);
```

// The following defparam INIT declaration is only necessary if you
 // wish to change the initial contents of the RAM to anything other
 // than all zero's. If the instance name for the RAM is is changes,
 // that change needs to be reflected in the defparam statement.

```
defparam RAM16X1D_inst.INIT = 16'h0000;
```

```
// End of RAM16X1D_inst instantiation
```

Commonly Used Constraints

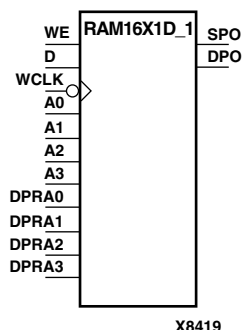
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X1D_1

16-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM16X1D_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X1D_1 is a 16-word by 1-bit static dual port random access memory with synchronous write capability and negative-edge clock. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM16X1D_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X1D_1: 16 x 1 negative edge write, asynchronous read
-- dual-port distributed RAM
--           All FPGA
-- The current version of the Xilinx HDL Libraries Guide

RAM16X1D_1_inst : RAM16X1D_1
-- The following generic INIT declaration is only necessary if
-- you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    WCLK => WCLK,   -- Port A write clock input
    WE => WE        -- Port A write enable input
);

-- End of RAM16X1D_1_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM16X1D_1 RAM16X1D_1_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
    .DPRA2(DPRA2), // Port B address[2] input bit
    .DPRA3(DPRA3), // Port B address[3] input bit
    .WCLK(WCLK),   // Port A write clock input
    .WE(WE)        // Port A write enable input
);
```

```
// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.
```

```
defparam RAM16X1D_1_inst.INIT = 16'h0000000000000000;
```

```
// End of RAM16X1D_1_inst instantiation
```

Commonly Used Constraints

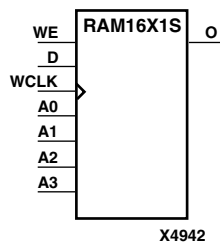
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X1S

16-Deep by 1-Wide Static Synchronous RAM

Architectures Supported

RAM16X1S	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X1S is a 16-word by 1-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- Note that the use of INIT below is for simulation only
-- For examples on how to include INIT as an implementation
-- constraint, please refer to the Constraints Guide
-- <Cut code below this line and paste into the architecture body>

-- RAM16X1S: 16 x 1 posedge write distributed => LUT RAM
--           All FPGA
-- The current version of the Xilinx HDL Libraries Guide

RAM16X1S_inst : RAM16X1S
-- Edit the following generic to change the initial contents of the RAM.
generic map (
    INIT => X"0000")
port map (
    O => O,          -- RAM output
    A0 => A0,        -- RAM address[0] input
    A1 => A1,        -- RAM address[1] input
    A2 => A2,        -- RAM address[2] input
    A3 => A3,        -- RAM address[3] input
    D => D,          -- RAM data input
    WCLK => WCLK,    -- Write clock input
    WE => WE         -- Write enable input
);

-- End of RAM16X1S_inst instantiation
```

Verilog Instantiation Template

```
// RAM16X1S: 16 x 1 posedge write distributed (LUT) RAM
//           All FPGA
// The current version of the Xilinx HDL Libraries Guide

RAM16X1S RAM16X1S_inst (
    .O(O),          // RAM output
    .A0(A0),        // RAM address[0] input
    .A1(A1),        // RAM address[1] input
    .A2(A2),        // RAM address[2] input
    .A3(A3),        // RAM address[3] input
    .D(D),          // RAM data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defpara
// statements.

defparam RAM16X1S_inst.INIT = 16'h0000;

// End of RAM16X1S_inst instantiation
```

Commonly Used Constraints

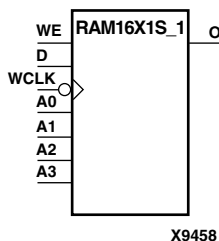
BEL, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM16X1S_1

16-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM16X1S_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X1S_1 is a 16-word by 1-bit static random access memory with synchronous write capability and negative-edge clock. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM16X1S_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE(mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X1S_1: 16 x 1 negedge write distributed => LUT RAM
--
-- All FPGA
-- The current version of the Xilinx HDL Libraries Guide

RAM16X1S_1_inst : RAM16X1S_1
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT => X"0000")
port map (
    O => O,          -- RAM output
    A0 => A0,        -- RAM address[0] input
    A1 => A1,        -- RAM address[1] input
    A2 => A2,        -- RAM address[2] input
    A3 => A3,        -- RAM address[3] input
    D => D,          -- RAM data input
    WCLK => WCLK,    -- Write clock input
    WE => WE         -- Write enable input
);

-- End of RAM16X1S_1_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
RAM16X1S_1 RAM16X1S_1_inst (
    .O(O),          // RAM output
    .A0(A0),        // RAM address[0] input
    .A1(A1),        // RAM address[1] input
    .A2(A2),        // RAM address[2] input
    .A3(A3),        // RAM address[3] input
    .D(D),          // RAM data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM16X1S_1_inst.INIT = 16'h0000;

// End of RAM16X1S_1_inst instantiation
```

Commonly Used Constraints

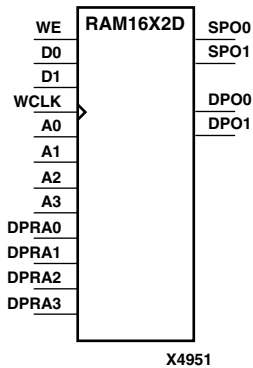
BEL, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X2D

16-Deep by 2-Wide Static Dual Port Synchronous RAM

Architectures Supported

RAM16X2D	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X2D is a 16-word by 2-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO1 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X2D cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D1-D0	SPO1-SPO0	DPO1-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D1-D0	D1-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

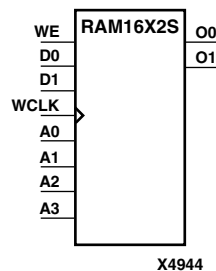
For HDL, this design element is inferred. See the *XST User Guide* for details.

RAM16X2S

16-Deep by 2-Wide Static Synchronous RAM

Architectures Supported

RAM16X2S	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X2S is a 16-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM16X2S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM16X2S as described in [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in this section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D1-D0	O1-O0
0 (read)	X	X	Data
1(read)	0	X	Data
1(read)	1	X	Data
1(write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM

You can use the INIT_xx properties to specify the initial contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X wide RAM. INIT_00 initializes the RAM cells corresponding to the O0 output, INIT_01 initializes the cells corresponding to the O1 output, etc. For example, a RAM16X2S instance is initialized by INIT_00 and INIT_01 containing 4 hex characters each. A RAM16X8S instance is initialized by eight properties INIT_00 through INIT_07 containing 4 hex characters each. A RAM64x2S instance is completely initialized by two properties INIT_00 and INIT_01 containing 16 hex characters each. See the INIT_xx section of the *Constraints Guide* for more information on the INIT_xx attribute.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X2S: 16 x 2 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

RAM16X2S_inst : RAM16X2S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"0000", -- INIT for bit 0 of RAM
    INIT_01 => X"0000", -- INIT for bit 1 of RAM")
port map (
    O0 => O0,      -- RAM data[0] output
    O1 => O1,      -- RAM data[1] output
    A0 => A0,      -- RAM address[0] input
    A1 => A1,      -- RAM address[1] input
    A2 => A2,      -- RAM address[2] input
    A3 => A3,      -- RAM address[3] input
    D0 => D0,      -- RAM data[0] input
    D1 => D1,      -- RAM data[1] input
    WCLK => WCLK,  -- Write clock input
    WE => WE       -- Write enable input
);

-- End of RAM16X2S_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

RAM16X2S RAM16X2S_inst (
    .O0(O0),      // RAM data[0] output
    .O1(O1),      // RAM data[1] output
    .A0(A0),      // RAM address[0] input
    .A1(A1),      // RAM address[1] input
```

```
.A2(A2),      // RAM address[2] input
.A3(A3),      // RAM address[3] input
.D0(D0),      // RAM data[0] input
.D1(D1),      // RAM data[1] input
.WCLK(WCLK), // Write clock input
.WE(WE)       // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM16X2S_inst.INIT_00 = 16'h0000; // INIT for bit 0 of RAM
defparam RAM16X2S_inst.INIT_01 = 16'h0000; // INIT for bit 1 of RAM

// End of RAM16X2S_inst instantiation
```

Commonly Used Constraints

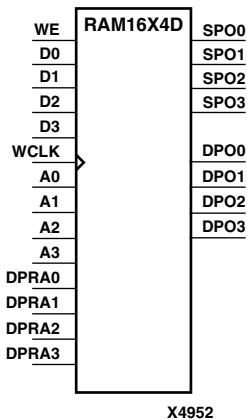
BEL, INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X4D

16-Deep by 4-Wide Static Dual Port Synchronous RAM

Architectures Supported

RAM16X4D	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X4D is a 16-word by 4-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO3 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X4D cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D3-D0	SPO3-SPO0	DPO3-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D3-D0	D3-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

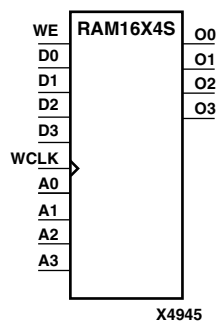
For HDL, this design element must be inferred. For information on how to infer RAM, see the *XST User Guide*.

RAM16X4S

16-Deep by 4-Wide Static Synchronous RAM

Architectures Supported

RAM16X4S	
Spartan-II, Spartan-III	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X4S is a 16-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D3 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM16X4S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use INIT_00 through INIT_03 to specify the initial contents of RAM16X4S as described in the [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) section in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D3 – D0	O3 – O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X4S: 16 x 4 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

RAM16X4S_inst : RAM16X4S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"0000", -- INIT for bit 0 of RAM
    INIT_01 => X"0000", -- INIT for bit 1 of RAM
    INIT_02 => X"0000", -- INIT for bit 2 of RAM
    INIT_03 => X"0000", -- INIT for bit 3 of RAM")
port map (
    O0 => O0,      -- RAM data[0] output
    O1 => O1,      -- RAM data[1] output
    O2 => O2,      -- RAM data[2] output
    O3 => O3,      -- RAM data[3] output
    A0 => A0,      -- RAM address[0] input
    A1 => A1,      -- RAM address[1] input
    A2 => A2,      -- RAM address[2] input
    A3 => A3,      -- RAM address[3] input
    D0 => D0,      -- RAM data[0] input
    D1 => D1,      -- RAM data[1] input
    D2 => D2,      -- RAM data[2] input
    D3 => D3,      -- RAM data[3] input
    WCLK => WCLK,  -- Write clock input
    WE => WE       -- Write enable input
);

-- End of RAM16X4S_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM16X4S RAM16X4S_inst (
    .O0(O0),      // RAM data[0] output
    .O1(O1),      // RAM data[1] output
    .O2(O2),      // RAM data[2] output
    .O3(O3),      // RAM data[3] output
    .A0(A0),      // RAM address[0] input
    .A1(A1),      // RAM address[1] input
    .A2(A2),      // RAM address[2] input
    .A3(A3),      // RAM address[3] input
    .D0(D0),      // RAM data[0] input
    .D1(D1),      // RAM data[1] input
    .D2(D2),      // RAM data[2] input
    .D3(D3),      // RAM data[3] input
    .WCLK(WCLK), // Write clock input
```

```
        .WE(WE)      // Write enable input
    );

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

    defparam RAM16X4S_inst.INIT_00 = 16'h0000; // INIT for bit 0 of RAM
    defparam RAM16X4S_inst.INIT_01 = 16'h0000; // INIT for bit 1 of RAM
    defparam RAM16X4S_inst.INIT_02 = 16'h0000; // INIT for bit 2 of RAM
    defparam RAM16X4S_inst.INIT_03 = 16'h0000; // INIT for bit 3 of RAM

// End of RAM16X4S_inst instantiation
```

Commonly Used Constraints

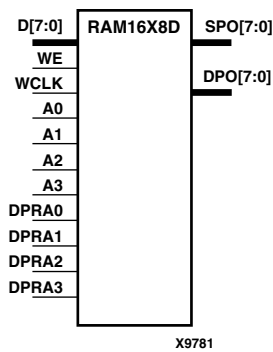
BEL, INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM16X8D

16-Deep by 8-Wide Static Dual Port Synchronous RAM

Architectures Supported

RAM16X8D	
Spartan-II, Spartan-IIE	Macro
Spartan-3	No
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X8D is a 16-word by 8-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA3 – DPRA0) and the write address (A3 – A0). These two address ports are completely asynchronous. The read address controls the location of data driven out of the output pin (DPO7 – DPO0), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D7 – D0) into the word selected by the 4-bit write address (A3 – A0). For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The initial contents of RAM16X8D cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D7-D0	SP7-SPO0	DPO7-DPO0
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D7-D0	D7-D0	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A3-A0

data_d = word addressed by bits DPRA3-DPRA0

The SPO output reflects the data in the memory cell addressed by A3 – A0. The DPO output reflects the data in the memory cell addressed by DPRA3 – DPRA0.

The write process is not affected by the address on the read address port.

Usage

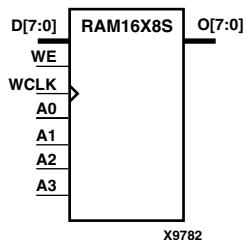
For HDL, this design element must be inferred. For information on how to infer RAM, see the *XST User Guide*.

RAM16X8S

16-Deep by 8-Wide Static Synchronous RAM

Architectures Supported

RAM16X8S	
Spartan-II, Spartan-III	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM16X8S is a 16-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on data inputs (D7 – D0) into the word selected by the 4-bit address (A3 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM16X8S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use INIT_00 through INIT_07 to specify the initial contents of RAM16X8S as described in the [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) section in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D7-D0	O7-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A3 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X8S: 16 x 8 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM16X8S_inst : RAM16X8S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"0000", -- INIT for bit 0 of RAM
    INIT_01 => X"0000", -- INIT for bit 1 of RAM
    INIT_02 => X"0000", -- INIT for bit 2 of RAM
    INIT_03 => X"0000", -- INIT for bit 3 of RAM
    INIT_04 => X"0000", -- INIT for bit 4 of RAM
    INIT_05 => X"0000", -- INIT for bit 5 of RAM
    INIT_06 => X"0000", -- INIT for bit 6 of RAM
    INIT_07 => X"0000", -- INIT for bit 7 of RAM")
port map (
    O => O,          -- 8-bit RAM data output
    A0 => A0,        -- RAM address[0] input
    A1 => A1,        -- RAM address[1] input
    A2 => A2,        -- RAM address[2] input
    A3 => A3,        -- RAM address[3] input
    D => D,          -- 8-bit RAM data input
    WCLK => WCLK,    -- Write clock input
    WE => WE         -- Write enable input
);

-- End of RAM16X8S_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the *Constraints Guide*.

```
RAM16X8S RAM16X8S_inst (
    .O(O),          // 8-bit RAM data output
    .A0(A0),        // RAM address[0] input
    .A1(A1),        // RAM address[1] input
    .A2(A2),        // RAM address[2] input
    .A3(A3),        // RAM address[3] input
    .D(D),          // 8-bit RAM data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.
```



```
defparam RAM16X8S_inst.INIT_00 = 16'h0000; // INIT for bit 0 of RAM
defparam RAM16X8S_inst.INIT_01 = 16'h0000; // INIT for bit 1 of RAM
defparam RAM16X8S_inst.INIT_02 = 16'h0000; // INIT for bit 2 of RAM
defparam RAM16X8S_inst.INIT_03 = 16'h0000; // INIT for bit 3 of RAM
defparam RAM16X8S_inst.INIT_04 = 16'h0000; // INIT for bit 4 of RAM
defparam RAM16X8S_inst.INIT_05 = 16'h0000; // INIT for bit 5 of RAM
defparam RAM16X8S_inst.INIT_06 = 16'h0000; // INIT for bit 6 of RAM
defparam RAM16X8S_inst.INIT_07 = 16'h0000; // INIT for bit 7 of RAM
```

```
// End of RAM16X8S_inst instantiation
```

Commonly Used Constraints

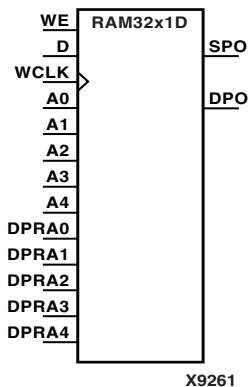
BEL, INIT_XX, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X1D

32-Deep by 1-Wide Static Dual Static Port Synchronous RAM

Architectures Supported

RAM32X1D	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X1D is a 32-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA4 – DPRA0) and the write address (A4 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM32X1D during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A4-A0

data_d = word addressed by bits DPRA4-DPRA0

The SPO output reflects the data in the memory cell addressed by A4 – A0. The DPO output reflects the data in the memory cell addressed by DPRA4 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X1D: 32 x 1 positive edge write, asynchronous read dual-port
distributed RAM
--           Viretx-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM32X1D_inst : RAM32X1D
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"00000000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    A4 => A4,        -- Port A address[4] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    DPRA4 => DPRA4, -- Port B address[4] input bit
    WCLK => WCLK,    -- Port A write clock input
    WE => WE         -- Port A write enable input
);

-- End of RAM32X1D_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM32X1D RAM32X1D_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .A4(A4),        // Port A address[4] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
    .DPRA2(DPRA2), // Port B address[2] input bit
    .DPRA3(DPRA3), // Port B address[3] input bit
    .DPRA4(DPRA4), // Port B address[4] input bit
```

```
.WCLK(WCLK), // Port A write clock input
.WE(WE)      // Port A write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM32X1D_inst.INIT = 32'h00000000;

// End of RAM32X1D_inst instantiation
```

Commonly Used Constraints

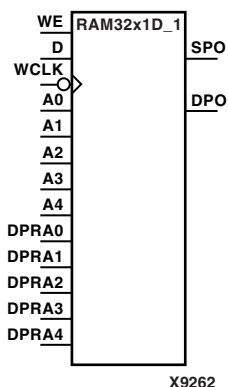
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM32X1D_1

32-Deep by 1-Wide Static Dual Port Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM32X1D_1	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X1D_1 is a 32-word by 1-bit static dual port random access memory with synchronous write capability and a negative-edge clock. The device has two separate address ports: the read address (DPRA4 – DPRA0) and the write address (A4 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM32X1D_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A4-A0

data_d = word addressed by bits DPRA4-DPRA0

The SPO output reflects the data in the memory cell addressed by A4 – A0. The DPO output reflects the data in the memory cell addressed by DPRA4 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM16X1D_1: 32 x 1 negative edge write, asynchronous read
-- dual-port distributed RAM
--           Viretx-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM32X1D_1_inst : RAM32X1D_1
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"00000000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    A4 => A4,        -- Port A address[4] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    DPRA4 => DPRA4, -- Port B address[4] input bit
    WCLK => WCLK,   -- Port A write clock input
    WE => WE        -- Port A write enable input
);

-- End of RAM32X1D_1_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM32X1D_1 RAM32X1D_1_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .A4(A4),        // Port A address[4] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
    .DPRA2(DPRA2), // Port B address[2] input bit
    .DPRA3(DPRA3), // Port B address[3] input bit
    .DPRA4(DPRA4), // Port B address[4] input bit
```



```
.WCLK(WCLK), // Port A write clock input
.WE(WE)      // Port A write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM32X1D_1_inst.INIT = 32'h00000000;

// End of RAM32X1D_1_inst instantiation
```

Commonly Used Constraints

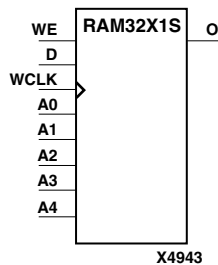
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X1S

32-Deep by 1-Wide Static Synchronous RAM

Architectures Supported

RAM32X1S	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X1S is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM32X1S: 32 x 1 posedge write distributed => LUT RAM
--           All FPGA
-- The current version of the Xilinx HDL Libraries Guide

RAM32X1S_inst : RAM32X1S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT => X"00000000")
port map (
    O => O,           -- RAM output
    A0 => A0,         -- RAM address[0] input
    A1 => A1,         -- RAM address[1] input
    A2 => A2,         -- RAM address[2] input
    A3 => A3,         -- RAM address[3] input
    A4 => A4,         -- RAM address[4] input
    D => D,           -- RAM data input
    WCLK => WCLK,     -- Write clock input
    WE => WE          -- Write enable input
);

-- End of RAM32X1S_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

RAM32X1S RAM32X1S_inst (
    .O(O),           // RAM output
    .A0(A0),        // RAM address[0] input
    .A1(A1),        // RAM address[1] input
    .A2(A2),        // RAM address[2] input
    .A3(A3),        // RAM address[3] input
    .A4(A4),        // RAM address[4] input
    .D(D),          // RAM data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM32X1S_inst.INIT = 32'h00000000;

// End of RAM32X1S_inst instantiation
```

Commonly Used Constraints

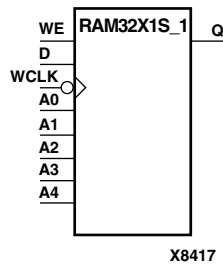
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X1S_1

32-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM32X1S_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X1S_1 is a 32-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM32X1S_1: 32 x 1 negedge write distributed => LUT RAM
--
-- All FPGA
-- The current version of the Xilinx HDL Libraries Guide

RAM32X1S_1_inst : RAM32X1S_1
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT => X"00000000")
port map (
    O => O,          -- RAM output
    A0 => A0,        -- RAM address[0] input
    A1 => A1,        -- RAM address[1] input
    A2 => A2,        -- RAM address[2] input
    A3 => A3,        -- RAM address[3] input
    A4 => A4,        -- RAM address[4] input
    D => D,          -- RAM data input
    WCLK => WCLK,    -- Write clock input
    WE => WE         -- Write enable input
);

-- End of RAM32X1S_1_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAM32X1S_1 RAM32X1S_1_inst (
    .O(O),          // RAM output
    .A0(A0),        // RAM address[0] input
    .A1(A1),        // RAM address[1] input
    .A2(A2),        // RAM address[2] input
    .A3(A3),        // RAM address[3] input
    .A4(A4),        // RAM address[4] input
    .D(D),          // RAM data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM32X1S_1_inst.INIT = 32'h00000000;

// End of RAM32X1S_1_inst instantiation
```

Commonly Used Constraints

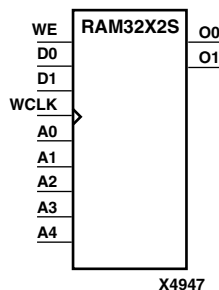
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X2S

32-Deep by 2-Wide Static Synchronous RAM

Architectures Supported

RAM32X2S	
Spartan-II, Spartan-III	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X2S is a 32-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM32X2S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM32X2S as described in [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D0-D1	O0-O1
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM32X2S: 32 x 2 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

RAM32X2S_inst : RAM32X2S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"00000000", -- INIT for bit 0 of RAM
    INIT_01 => X"00000000", -- INIT for bit 1 of RAM")
port map (
    O0 => O0,      -- RAM data[0] output
    O1 => O1,      -- RAM data[1] output
    A0 => A0,      -- RAM address[0] input
    A1 => A1,      -- RAM address[1] input
    A2 => A2,      -- RAM address[2] input
    A3 => A3,      -- RAM address[3] input
    A4 => A4,      -- RAM address[4] input
    D0 => D0,      -- RAM data[0] input
    D1 => D1,      -- RAM data[1] input
    WCLK => WCLK,  -- Write clock input
    WE => WE       -- Write enable input
);

-- End of RAM32X2S_inst instantiation
```

Verilog Instantiation Template

Note that the use of INIT below is for simulation only. For examples of how to include INIT as an implementation constraint, please refer to the *Constraints Guide*.

```
RAM32X2S RAM32X2S_inst (
    .O0(O0),      // RAM data[0] output
    .O1(O1),      // RAM data[1] output
    .A0(A0),      // RAM address[0] input
    .A1(A1),      // RAM address[1] input
    .A2(A2),      // RAM address[2] input
    .A3(A3),      // RAM address[3] input
    .A4(A4),      // RAM address[4] input
    .D0(D0),      // RAM data[0] input
    .D1(D1),      // RAM data[1] input
    .WCLK(WCLK), // Write clock input
    .WE(WE)       // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM32X2S_inst.INIT_00 = 32'h00000000; // INIT for bit 0 of RAM
```



```
defparam RAM32X2S_inst.INIT_01 = 32'h00000000; // INIT for bit 1 of RAM  
// End of RAM32X2S_inst instantiation
```

Commonly Used Constraints

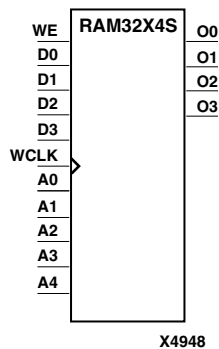
INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X4S

32-Deep by 4-Wide Static Synchronous RAM

Architectures Supported

RAM32X4S	
Spartan-II, Spartan-III	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X4S is a 32-word by 4-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D3 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O3 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

Except for Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM32X4S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use the INIT_00 through INIT_03 properties to specify the initial contents of RAM32X4S as described in [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE	WCLK	D3-D0	O3-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D3-D0	D3-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM32X4S: 32 x 4 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM32X4S_inst : RAM32X4S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"00000000", -- INIT for bit 0 of RAM
    INIT_01 => X"00000000", -- INIT for bit 1 of RAM
    INIT_02 => X"00000000", -- INIT for bit 2 of RAM
    INIT_03 => X"00000000", -- INIT for bit 3 of RAM")
port map (
    O0 => O0,      -- RAM data[0] output
    O1 => O1,      -- RAM data[1] output
    O2 => O2,      -- RAM data[2] output
    O3 => O3,      -- RAM data[3] output
    A0 => A0,      -- RAM address[0] input
    A1 => A1,      -- RAM address[1] input
    A2 => A2,      -- RAM address[2] input
    A3 => A3,      -- RAM address[3] input
    A4 => A4,      -- RAM address[4] input
    D0 => D0,      -- RAM data[0] input
    D1 => D1,      -- RAM data[1] input
    D2 => D2,      -- RAM data[2] input
    D3 => D3,      -- RAM data[3] input
    WCLK => WCLK,  -- Write clock input
    WE => WE       -- Write enable input
);

-- End of RAM32X4S_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the *Constraints Guide*.

```
RAM32X4S RAM32X4S_inst (
    .O0(O0),      // RAM data[0] output
    .O1(O1),      // RAM data[1] output
    .O2(O2),      // RAM data[2] output
    .O3(O3),      // RAM data[3] output
    .A0(A0),      // RAM address[0] input
    .A1(A1),      // RAM address[1] input
    .A2(A2),      // RAM address[2] input
    .A3(A3),      // RAM address[3] input
    .A4(A4),      // RAM address[4] input
    .D0(D0),      // RAM data[0] input
    .D1(D1),      // RAM data[1] input
    .D2(D2),      // RAM data[2] input
```

```
.D3(D3),      // RAM data[3] input
.WCLK(WCLK), // Write clock input
.WE(WE)       // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM32X4S_inst.INIT_00 = 32'h00000000; // INIT for bit 0 of RAM
defparam RAM32X4S_inst.INIT_01 = 32'h00000000; // INIT for bit 1 of RAM
defparam RAM32X4S_inst.INIT_02 = 32'h00000000; // INIT for bit 2 of RAM
defparam RAM32X4S_inst.INIT_03 = 32'h00000000; // INIT for bit 3 of RAM

// End of RAM32X4S_inst instantiation
```

Commonly Used Constraints

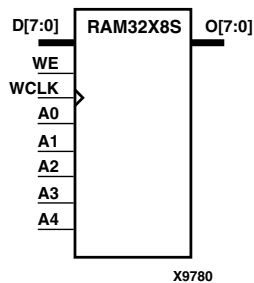
INIT_xx, BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM32X8S

32-Deep by 8-Wide Static Synchronous RAM

Architectures Supported

RAM32X8S	
Spartan-II, Spartan-III	Macro
Spartan-3	Primitive
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM32X8S is a 32-word by 8-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data inputs (D7 – D0) into the word selected by the 5-bit address (A4 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O7 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

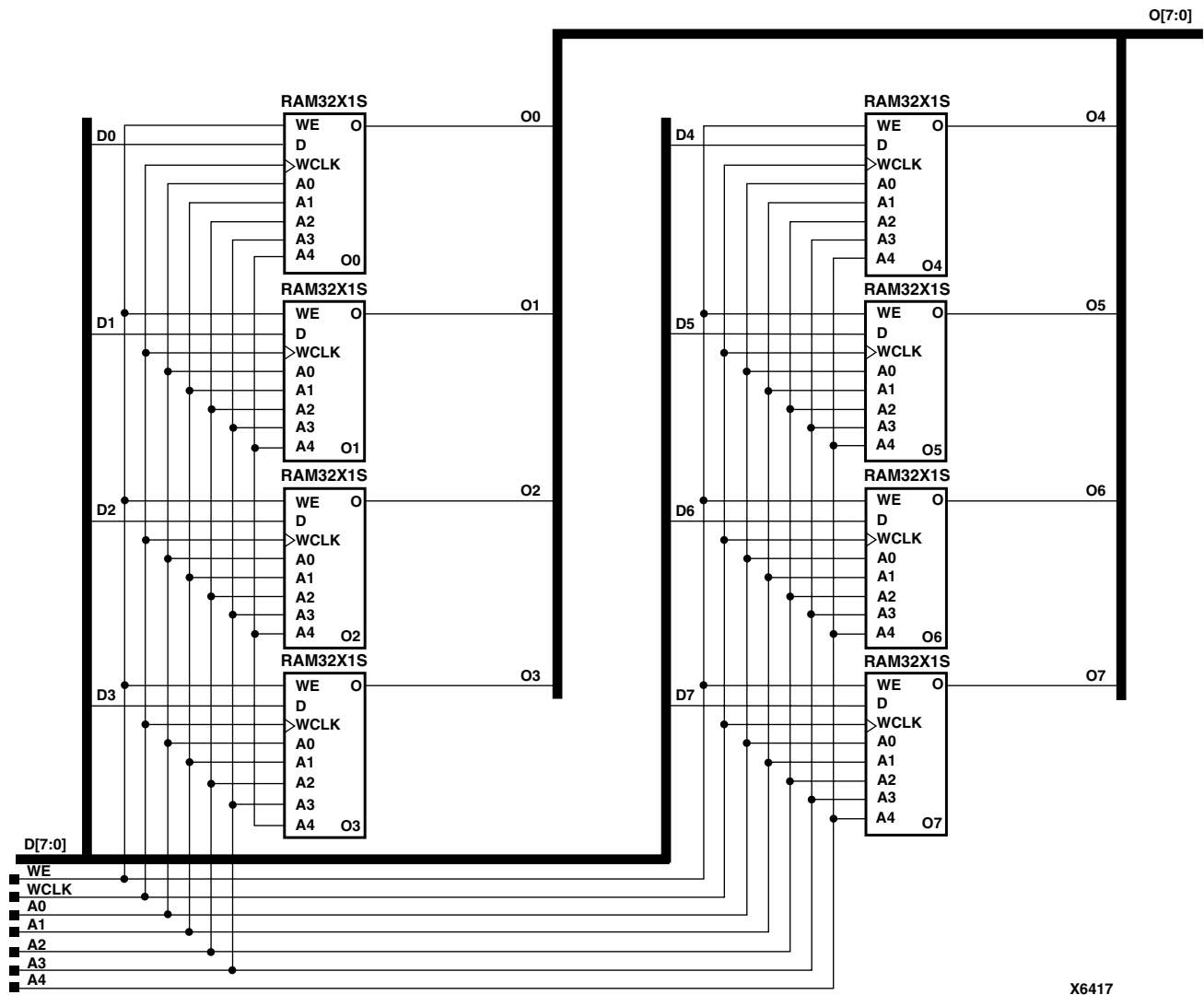
Except for Virtex-II, Virtex-II Pro, and Virtex-II Pro X, the initial contents of RAM32X8S cannot be specified directly. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

For Virtex-II, Virtex-II Pro, and Virtex-II Pro X, you can use the INIT_00 through INIT_07 properties to specify the initial contents of RAM32X8S as described in [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D7-D0	O7-O0
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D7-D0	D7-D0
1 (read)	↓	X	Data

Data = word addressed by bits A4 – A0



RAM32X8S Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```

-- RAM32X8S: 32 x 8 posedge write distributed => LUT RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM32X8S_inst : RAM32X8S
-- Edit the following generic to change the initial contents
-- of the RAM.
generic map (
    INIT_00 => X"00000000", -- INIT for bit 0 of RAM

```



```

INIT_01 => X"00000000", -- INIT for bit 1 of RAM
INIT_02 => X"00000000", -- INIT for bit 2 of RAM
INIT_03 => X"00000000", -- INIT for bit 3 of RAM
INIT_04 => X"00000000", -- INIT for bit 4 of RAM
INIT_05 => X"00000000", -- INIT for bit 5 of RAM
INIT_06 => X"00000000", -- INIT for bit 6 of RAM
INIT_07 => X"00000000", -- INIT for bit 7 of RAM")
port map (
  O => O,      -- 8-bit RAM data output
  A0 => A0,    -- RAM address[0] input
  A1 => A1,    -- RAM address[1] input
  A2 => A2,    -- RAM address[2] input
  A3 => A3,    -- RAM address[3] input
  A4 => A4,    -- RAM address[4] input
  D => D,      -- 8-bit RAM data input
  WCLK => WCLK, -- Write clock input
  WE => WE     -- Write enable input
);

-- End of RAM32X8S_inst instantiation

```

Verilog Instantiation Template

```

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

```

```

RAM32X8S RAM32X8S_inst (
  .O(O),      // 8-bit RAM data output
  .A0(A0),    // RAM address[0] input
  .A1(A1),    // RAM address[1] input
  .A2(A2),    // RAM address[2] input
  .A3(A3),    // RAM address[3] input
  .A4(A4),    // RAM address[4] input
  .D(D),      // 8-bit RAM data input
  .WCLK(WCLK), // Write clock input
  .WE(WE)     // Write enable input
);

// Edit the following defparam to change the initial contents of the RAM
// to a non-zero value if desired. If the instance name to the RAM is
// changed, that change needs to be reflected in the defparam
// statements.

defparam RAM32X8S_inst.INIT_00 = 32'h00000000; // INIT for bit 0 of RAM
defparam RAM32X8S_inst.INIT_01 = 32'h00000000; // INIT for bit 1 of RAM
defparam RAM32X8S_inst.INIT_02 = 32'h00000000; // INIT for bit 2 of RAM
defparam RAM32X8S_inst.INIT_03 = 32'h00000000; // INIT for bit 3 of RAM
defparam RAM32X8S_inst.INIT_04 = 32'h00000000; // INIT for bit 4 of RAM
defparam RAM32X8S_inst.INIT_05 = 32'h00000000; // INIT for bit 5 of RAM
defparam RAM32X8S_inst.INIT_06 = 32'h00000000; // INIT for bit 6 of RAM
defparam RAM32X8S_inst.INIT_07 = 32'h00000000; // INIT for bit 7 of RAM

// End of RAM32X8S_inst instantiation

```

Commonly Used Constraints

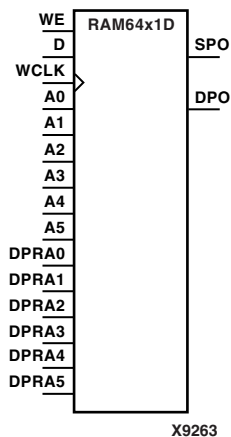
BLKNM, HBLKNM, HU_SET, INIT, INIT_xx, LOC, RLOC, U_SET, and XBLKNM

RAM64X1D

64-Deep by 1-Wide Dual Port Static Synchronous RAM

Architectures Supported

RAM64X1D	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM64X1D is a 64-word by 1-bit static dual port random access memory with synchronous write capability. The device has two separate address ports: the read address (DPRA5 – DPRA0) and the write address (A5 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit (A0 - A5) write address. For predictable performance, write address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM64X1D during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↑	D	D	data_d
1 (read)	↓	X	data_a	data_d

data_a = word addressed by bits A5-A0

data_d = word addressed by bits DPRA5-DPRA0

The SPO output reflects the data in the memory cell addressed by A5 – A0. The DPO output reflects the data in the memory cell addressed by DPRA5 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM64X1D: 64 x 1 positive edge write, asynchronous read
-- dual-port distributed RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM64X1D_inst : RAM64X1D
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000000000000000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    A4 => A4,        -- Port A address[4] input bit
    A5 => A5,        -- Port A address[5] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    DPRA4 => DPRA4, -- Port B address[4] input bit
    DPRA5 => DPRA5, -- Port B address[5] input bit
    WCLK => WCLK,   -- Port A write clock input
    WE => WE        -- Port A write enable input
);

-- End of RAM64X1D_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM64X1D RAM64X1D_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .A4(A4),        // Port A address[4] input bit
    .A5(A5),        // Port A address[5] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
```

```
.DPRA2(DPRA2), // Port B address[2] input bit
.DPRA3(DPRA3), // Port B address[3] input bit
.DPRA4(DPRA4), // Port B address[4] input bit
.DPRA5(DPRA5), // Port B address[5] input bit
.WCLK(WCLK),  // Port A write clock input
.WE(WE)       // Port A write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM64X1D_inst.INIT = 64'h0000000000000000;

// End of RAM64X1D_inst instantiation
```

Commonly Used Constraints

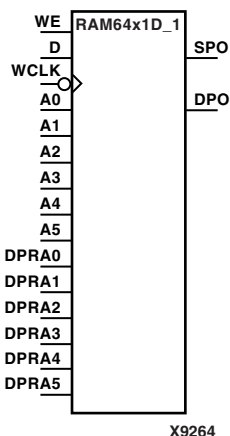
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X1D_1

64-Deep by 1-Wide Dual Port Static Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM64X1D_1	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM64X1D_1 is a 64-word by 1-bit static dual port random access memory with synchronous write capability and a negative-edge clock. The device has two separate address ports: the read address (DPRA5 – DPRA0) and the write address (A5 – A0). These two address ports are completely asynchronous. The read address controls the location of the data driven out of the output pin (DPO), and the write address controls the destination of a valid write transaction.

When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit (A0 - A5) write address. For predictable performance, write address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

You can initialize RAM64X1D_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs	
WE (mode)	WCLK	D	SPO	DPO
0 (read)	X	X	data_a	data_d
1 (read)	0	X	data_a	data_d
1 (read)	1	X	data_a	data_d
1 (write)	↓	D	D	data_d
1 (read)	↑	X	data_a	data_d

data_a = word addressed by bits A5-A0

data_d = word addressed by bits DPRA5-DPRA0

The SPO output reflects the data in the memory cell addressed by A5 – A0. The DPO output reflects the data in the memory cell addressed by DPRA5 – DPRA0.

Note: The write process is not affected by the address on the read address port.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM64X1D_1: 64 x 1 negative edge write, asynchronous read
-- dual-port distributed RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM64X1D_1_inst : RAM64X1D_1
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000000000000000")
port map (
    DPO => DPO,      -- Port A 1-bit data output
    SPO => SPO,      -- Port B 1-bit data output
    A0 => A0,        -- Port A address[0] input bit
    A1 => A1,        -- Port A address[1] input bit
    A2 => A2,        -- Port A address[2] input bit
    A3 => A3,        -- Port A address[3] input bit
    A4 => A4,        -- Port A address[4] input bit
    A5 => A5,        -- Port A address[5] input bit
    D => D,          -- Port A 1-bit data input
    DPRA0 => DPRA0, -- Port B address[0] input bit
    DPRA1 => DPRA1, -- Port B address[1] input bit
    DPRA2 => DPRA2, -- Port B address[2] input bit
    DPRA3 => DPRA3, -- Port B address[3] input bit
    DPRA4 => DPRA4, -- Port B address[4] input bit
    DPRA5 => DPRA5, -- Port B address[5] input bit
    WCLK => WCLK,    -- Port A write clock input
    WE => WE         -- Port A write enable input
);

-- End of RAM64X1D_1_inst instantiation
```

Verilog Instantiation Template

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

```
RAM64X1D_1 RAM64X1D_1_inst (
    .DPO(DPO),      // Port A 1-bit data output
    .SPO(SPO),      // Port B 1-bit data output
    .A0(A0),        // Port A address[0] input bit
    .A1(A1),        // Port A address[1] input bit
    .A2(A2),        // Port A address[2] input bit
    .A3(A3),        // Port A address[3] input bit
    .A4(A4),        // Port A address[4] input bit
    .A5(A5),        // Port A address[5] input bit
    .D(D),          // Port A 1-bit data input
    .DPRA0(DPRA0), // Port B address[0] input bit
    .DPRA1(DPRA1), // Port B address[1] input bit
```



```
.DPRA2(DPRA2), // Port B address[2] input bit
.DPRA3(DPRA3), // Port B address[3] input bit
.DPRA4(DPRA4), // Port B address[4] input bit
.DPRA5(DPRA5), // Port B address[5] input bit
.WCLK(WCLK),  // Port A write clock input
.WE(WE)       // Port A write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM64X1D_1_inst.INIT = 64'h0000000000000000;

// End of RAM64X1D_1_inst instantiation
```

Commonly Used Constraints

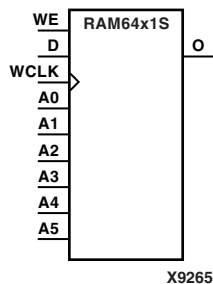
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, XBLKNM

RAM64X1S

64-Deep by 1-Wide Static Synchronous RAM

Architectures Supported

RAM64X1S	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM64X1S is a 64-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM64X1S during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A5 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM64X1S: 64 x 1 positive edge write, asynchronous read
-- single-port distributed RAM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

RAM64X1S_inst : RAM64X1S
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000000000000000")
port map (
    O => O,           -- 1-bit data output
    A0 => A0,         -- Address[0] input bit
    A1 => A1,         -- Address[1] input bit
    A2 => A2,         -- Address[2] input bit
    A3 => A3,         -- Address[3] input bit
    A4 => A4,         -- Address[4] input bit
    A5 => A5,         -- Address[5] input bit
    D => D,           -- 1-bit data input
    WCLK => WCLK,     -- Write clock input
    WE => WE          -- Write enable input
);

-- End of RAM64X1S_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAM64X1S RAM64X1S_inst (
    .O(O),           // 1-bit data output
    .A0(A0),         // Address[0] input bit
    .A1(A1),         // Address[1] input bit
    .A2(A2),         // Address[2] input bit
    .A3(A3),         // Address[3] input bit
    .A4(A4),         // Address[4] input bit
    .A5(A5),         // Address[5] input bit
    .D(D),           // 1-bit data input
    .WCLK(WCLK),     // Write clock input
    .WE(WE)          // Write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM64X1S_inst.INIT = 64'h0000000000000000;

// End of RAM64X1S_inst instantiation
```

Commonly Used Constraints

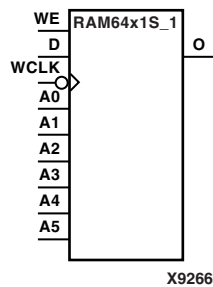
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM64X1S_1

64-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM64X1S_1	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM64X1S_1 is a 64-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM32X1S_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A5 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM64X1S_1: 64 x 1 negative edge write, asynchronous read
-- single-port distributed RAM
--                               Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

RAM64X1S_1_inst : RAM64X1S_1
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000000000000000")
port map (
    O => O,           -- 1-bit data output
    A0 => A0,         -- Address[0] input bit
    A1 => A1,         -- Address[1] input bit
    A2 => A2,         -- Address[2] input bit
    A3 => A3,         -- Address[3] input bit
    A4 => A4,         -- Address[4] input bit
    A5 => A5,         -- Address[5] input bit
    D => D,           -- 1-bit data input
    WCLK => WCLK,     -- Write clock input
    WE => WE          -- Write enable input
);

-- End of RAM64X1S_1_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAM64X1S_1 RAM64X1S_1_inst (
    .O(O),           // 1-bit data output
    .A0(A0),         // Address[0] input bit
    .A1(A1),         // Address[1] input bit
    .A2(A2),         // Address[2] input bit
    .A3(A3),         // Address[3] input bit
    .A4(A4),         // Address[4] input bit
    .A5(A5),         // Address[5] input bit
    .D(D),           // 1-bit data input
    .WCLK(WCLK),     // Write clock input
    .WE(WE)          // Write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM64X1S_1_inst.INIT = 64'h0000000000000000;

// End of RAM64X1S_1_inst instantiation
```


Commonly Used Constraints

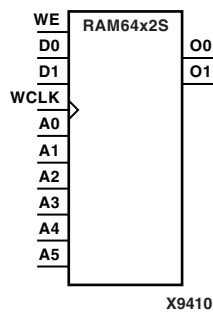
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM64X2S

64-Deep by 2-Wide Static Synchronous RAM

Architectures Supported

RAM64X2S	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM64X2S is a 64-word by 2-bit static random access memory with synchronous write capability. When the write enable (WE) is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D1 – D0) into the word selected by the 6-bit address (A5 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pins (O1 – O0) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can use the INIT_00 and INIT_01 properties to specify the initial contents of RAM64X2S as described in [“Specifying Initial Contents of a Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X Wide RAM”](#) in the RAM16X2S section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D0-D1	O0-O1
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D1-D0	D1-D0
1 (read)	↓	X	Data

Data = word addressed by bits A5 – A0

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template

```
-- RAM64X1S: 64 x 2 positive edge write, asynchronous read
-- single-port distributed RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM64X2S_inst : RAM64X2S
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"0000000000000000")
port map (
    O0 => O0,      -- Data[0] output
    O1 => O1,      -- Data[1] output bit
    A0 => A0,      -- Address[0] input bit
    A1 => A1,      -- Address[1] input bit
    A2 => A2,      -- Address[2] input bit
    A3 => A3,      -- Address[3] input bit
    A4 => A4,      -- Address[4] input bit
    A5 => A5,      -- Address[5] input bit
    D0 => D0,      -- Data[0] input
    D1 => D1,      -- Data[1] input
    WCLK => WCLK,  -- Write clock input
    WE => WE       -- Write enable input
);

-- End of RAM64X2S_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAM64X2S RAM64X2S_inst (
    .O0(O0),      // Data[0] output
    .O1(O1),      // Data[1] output bit
    .A0(A0),      // Address[0] input bit
    .A1(A1),      // Address[1] input bit
    .A2(A2),      // Address[2] input bit
    .A3(A3),      // Address[3] input bit
    .A4(A4),      // Address[4] input bit
    .A5(A5),      // Address[5] input bit
    .D0(D0),      // Data[0] input
    .D1(D1),      // Data[1] input
    .WCLK(WCLK), // Write clock input
    .WE(WE)       // Write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.

defparam RAM64X2S_inst.INIT = 64'h0000000000000000;
// End of RAM64X2S_inst instantiation
```

Commonly Used Constraints

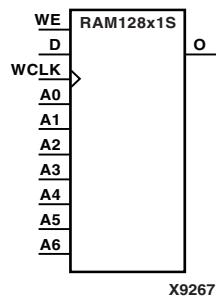
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM128X1S

128-Deep by 1-Wide Static Synchronous RAM

Architectures Supported

RAM128X1S	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM128X1S is a 128-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any positive transition on WCLK loads the data on the data input (D) into the word selected by the 7-bit address (A6 – A0). For predictable performance, address and data inputs must be stable before a Low-to-High WCLK transition. This RAM block assumes an active-High WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM128X1S during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↑	D	D
1 (read)	↓	X	Data

Data = word addressed by bits A6 – A0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- RAM128X1S: 128 x 1 positive edge write, asynchronous read
-- single-port distributed RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM128X1S_inst : RAM128X1S
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"00000000000000000000000000000000")
port map (
    O => O,           -- 1-bit data output
    A0 => A0,         -- Address[0] input bit
    A1 => A1,         -- Address[1] input bit
    A2 => A2,         -- Address[2] input bit
    A3 => A3,         -- Address[3] input bit
    A4 => A4,         -- Address[4] input bit
    A5 => A5,         -- Address[5] input bit
    A6 => A6,         -- Address[6] input bit
    D => D,           -- 1-bit data input
    WCLK => WCLK,     -- Write clock input
    WE => WE          -- Write enable input
);

-- End of RAM128X1S_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
// RAM128X1S: 128 x 1 positive edge write, asynchronous read
// single-port
//           distributed RAM. Virtex-II/II-Pro
// The current version of the Xilinx HDL Libraries Guide

RAM128X1S RAM128X1S_inst (
    .O(O),           // 1-bit data output
    .A0(A0),         // Address[0] input bit
    .A1(A1),         // Address[1] input bit
    .A2(A2),         // Address[2] input bit
    .A3(A3),         // Address[3] input bit
    .A4(A4),         // Address[4] input bit
    .A5(A5),         // Address[5] input bit
    .A6(A6),         // Address[6] input bit
    .D(D),           // 1-bit data input
    .WCLK(WCLK),    // Write clock input
    .WE(WE)         // Write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
```



```
// that change needs to be reflected in the defparam statement.  
defparam RAM128X1S_inst.INIT = 128'h00000000000000000000000000000000;  
// End of RAM128X1S_inst instantiation
```

Commonly Used Constraints

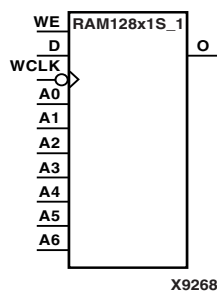
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAM128X1S_1

128-Deep by 1-Wide Static Synchronous RAM with Negative-Edge Clock

Architectures Supported

RAM128X1S_1	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAM128X1S_1 is a 128-word by 1-bit static random access memory with synchronous write capability. When the write enable is Low, transitions on the write clock (WCLK) are ignored and data stored in the RAM is not affected. When WE is High, any negative transition on WCLK loads the data on the data input (D) into the word selected by the 7-bit address (A6 – A0). For predictable performance, address and data inputs must be stable before a High-to-Low WCLK transition. This RAM block assumes an active-Low WCLK. However, WCLK can be active-High or active-Low. Any inverter placed on the WCLK input net is absorbed into the block.

The signal output on the data output pin (O) is the data that is stored in the RAM at the location defined by the values on the address pins.

You can initialize RAM128X1S_1 during configuration using the INIT attribute. See [“Specifying Initial Contents of a RAM”](#) in the RAM16X1D section.

Mode selection is shown in the following truth table.

Inputs			Outputs
WE (mode)	WCLK	D	O
0 (read)	X	X	Data
1 (read)	0	X	Data
1 (read)	1	X	Data
1 (write)	↓	D	D
1 (read)	↑	X	Data

Data = word addressed by bits A6 – A0

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user’s source code.

VHDL Instantiation Template

```
-- RAM128X1S_1: 128 x 1 negative edge write, asynchronous
-- read single-port distributed RAM
--           Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

RAM128X1S_1_inst : RAM128X1S_1
-- The following generic INIT declaration is only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
    INIT => X"00000000000000000000000000000000")
port map (
    O => O,           -- 1-bit data output
    A0 => A0,         -- Address[0] input bit
    A1 => A1,         -- Address[1] input bit
    A2 => A2,         -- Address[2] input bit
    A3 => A3,         -- Address[3] input bit
    A4 => A4,         -- Address[4] input bit
    A5 => A5,         -- Address[5] input bit
    A6 => A6,         -- Address[6] input bit
    D => D,           -- 1-bit data input
    WCLK => WCLK,     -- Write clock input
    WE => WE          -- Write enable input
);

-- End of RAM128X1S_1_inst instantiation
```

Verilog Instantiation Template

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
// RAM128X1S_1: 128 x 1 negative edge write, asynch. read
//           single-port distributed RAM.   Virtex-II/II-Pro
// The current version of the Xilinx HDL Libraries Guide

RAM128X1S_1 RAM128X1S_1_inst (
    .O(O),           // 1-bit data output
    .A0(A0),         // Address[0] input bit
    .A1(A1),         // Address[1] input bit
    .A2(A2),         // Address[2] input bit
    .A3(A3),         // Address[3] input bit
    .A4(A4),         // Address[4] input bit
    .A5(A5),         // Address[5] input bit
    .A6(A6),         // Address[6] input bit
    .D(D),           // 1-bit data input
    .WCLK(WCLK),     // Write clock input
    .WE(WE)          // Write enable input
);

// The following defparam INIT declaration is only necessary if you
// wish to change the initial contents of the RAM to anything other
// than all zero's. If the instance name for the RAM is is changes,
// that change needs to be reflected in the defparam statement.
```

```
defparam RAM128X1S_1_inst.INIT =  
    128'h00000000000000000000000000000000;  
  
// End of RAM128X1S_1_inst instantiation
```

Commonly Used Constraints

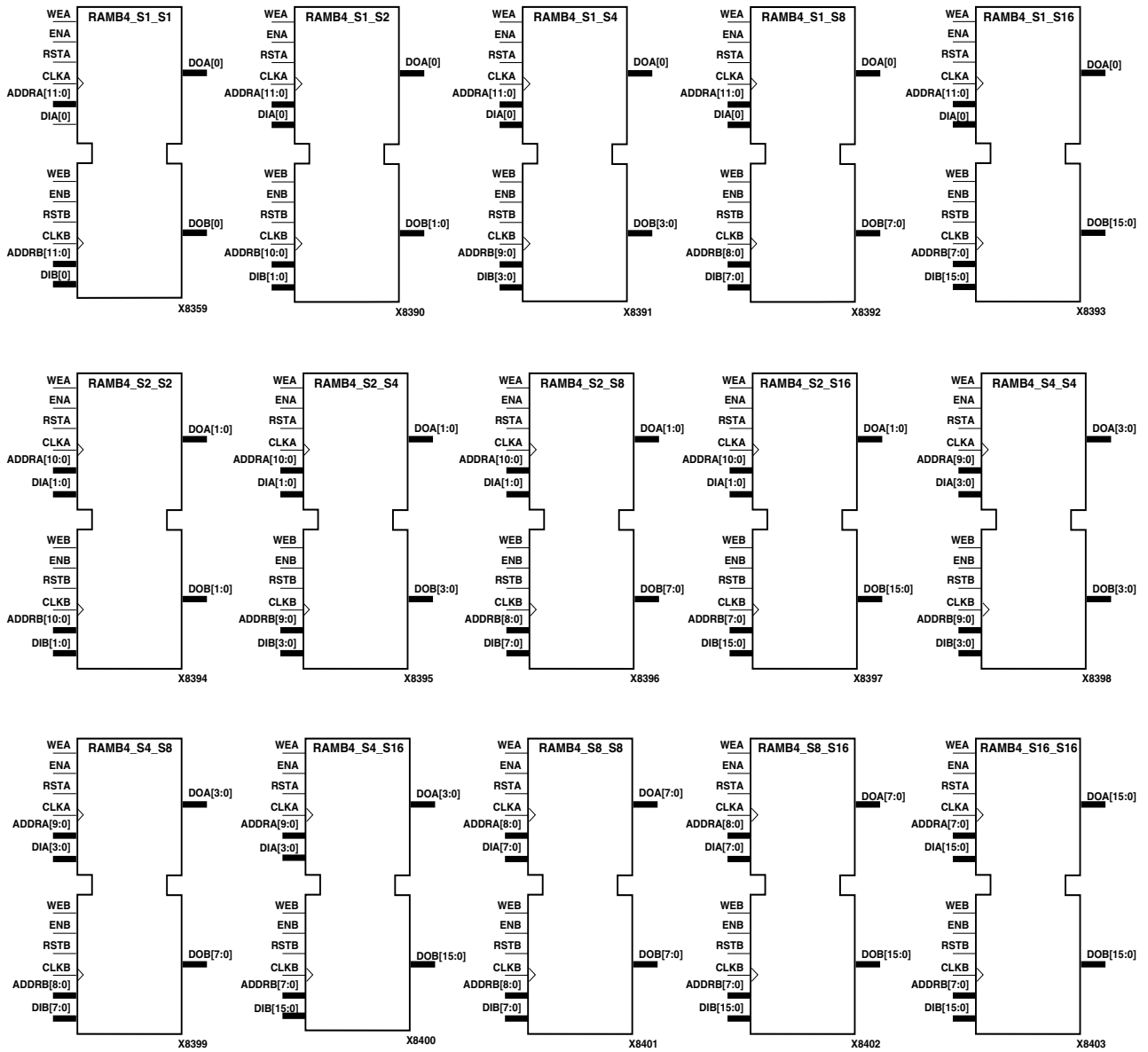
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, U_SET, and XBLKNM

RAMB4_Sm_Sn

4096-Bit Dual-Port Synchronous Block RAM with Port Width (m or n)
Configured to 1, 2, 4, 8, or 16 Bits

Architectures Supported

RAMB4_Sm_Sn	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X8727

RAMB4_Sm_Sn Representations

The RAMB4_Sm_Sn components listed in the following table are 4096-bit dual-ported dedicated random access memory blocks with synchronous write capability. Each port is independent of the other while accessing the same set of 4096 memory cells. Each port is independently configured to a specific data width.

Component	Port A Depth	Port A Width	Port A ADDR	Port A DI	Port B Depth	Port B Width	Port B ADDR	Port B DI
RAMB4_S1_S1	4096	1	(11:0)	(0:0)	4096	1	(11:0)	(0:0)
RAMB4_S1_S2	4096	1	(11:0)	(0:0)	2048	2	(10:0)	(1:0)
RAMB4_S1_S4	4096	1	(11:0)	(0:0)	1024	4	(9:0)	(3:0)
RAMB4_S1_S8	4096	1	(11:0)	(0:0)	512	8	(8:0)	(7:0)
RAMB4_S1_S16	4096	1	(11:0)	(0:0)	256	16	(7:0)	(15:0)
RAMB4_S2_S2	2048	2	(10:0)	(1:0)	2048	2	(10:0)	(1:0)
RAMB4_S2_S4	2048	2	(10:0)	(1:0)	1024	4	(9:0)	(3:0)
RAMB4_S2_S8	2048	2	(10:0)	(1:0)	512	8	(8:0)	(7:0)
RAMB4_S2_S16	2048	2	(10:0)	(1:0)	256	16	(7:0)	(15:0)
RAMB4_S4_S4	1024	4	(9:0)	(3:0)	1024	4	(9:0)	(3:0)
RAMB4_S4_S8	1024	4	(9:0)	(3:0)	512	8	(8:0)	(7:0)
RAMB4_S4_S16	1024	4	(9:0)	(3:0)	256	16	(7:0)	(15:0)
RAMB4_S8_S8	512	8	(8:0)	(7:0)	512	8	(8:0)	(7:0)
RAMB4_S8_S16	512	8	(8:0)	(7:0)	256	16	(7:0)	(15:0)
RAMB4_S16_S16	256	16	(7:0)	(15:0)	256	16	(7:0)	(15:0)

ADDR=address bus for the port

DI=data input bus for the port

Each port is fully synchronous with independent clock pins. All port A input pins have setup time referenced to the CLKA pin and its data output bus DOA has a clock-to-out time referenced to the CLKA. All port B input pins have setup time referenced to the CLKB pin and its data output bus DOB has a clock-to-out time referenced to the CLKB.

The enable ENA pin controls read, write, and reset for port A. When ENA is Low, no data is written and the output (DOA) retains the last state. When ENA is High and reset (RSTA) is High, DOA is cleared during the Low-to-High clock (CLKA) transition; if write enable (WEA) is High, the memory contents reflect the data at DIA. When ENA is High and WEA is Low, the data stored in the RAM address (ADDRA) is read during the Low-to-High clock transition. When ENA and WEA are High, the data on the data input (DIA) is loaded into the word selected by the write address (ADDRA) during the Low-to-High clock transition and the data output (DOA) reflects the selected (addressed) word.

The enable ENB pin controls read, write, and reset for port B. When ENB is Low, no data is written and the output (DOB) retains the last state. When ENB is High and reset (RSTB) is High, DOB is cleared during the Low-to-High clock (CLKB) transition; if write enable (WEB) is High, the memory contents reflect the data at DIB. When ENB is High and WEB is Low, the data stored in the RAM address (ADDRB) is read during the Low-to-High clock transition. When ENB and WEB are High, the data on the data input (DIB) is loaded into the word selected by the write address (ADDRB) during the Low-to-High clock transition and the data output (DOB) reflects the selected (addressed) word.

The above descriptions assume active High control pins (ENA, WEA, RSTA, CLKA, ENB, WEB, RSTB, and CLKB). However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB4_Sm_Sn 's may be initialized during configuration. See the following truth table.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered.

Virtex, Virtex-E, Spartan-II, and Spartan-II-E simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN(A/B)	RST(A/B)	WE(A/B)	CLK(A/B)	ADDR(A/B)	DI(A/B)	DO(A/B)	RAM Contents
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) =>data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) =>data

addr=RAM address of port A/B

RAM(addr)=RAM contents at address ADDR_A/ADDR_B

data=RAM input data at pins DIA/DIB

Address Mapping

Each port accesses the same set of 4096 memory cells using an addressing scheme that is dependent on the width of the port. The physical RAM location that is addressed for a particular width is determined from the following formula.

$$\text{Start} = ((\text{ADDR}_{\text{port}} + 1) * (\text{Width}_{\text{port}})) - 1$$

$$\text{End} = (\text{ADDR}_{\text{port}}) * (\text{Width}_{\text{port}})$$

The following table shows address mapping for each port width.

Port Address Mapping

Port Width	Port Addresses																	
1	4096	<----	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
2	2048	<----	07		06		05		04		03		02		01		00	
4	1024	<----	03				02				01				00			
8	512	<----	01								00							
16	256	<----	00															

Port A and Port B Conflict Resolution

A RAMB4_Sm_Sn component is a true dual-ported RAM in that it allows simultaneous reads of the same memory cell. When one port is performing a write to a given memory cell, the other port should not address that memory cell (for a write or a read) within the clock-to-clock setup window.

If both ports write to the same memory cell simultaneously, violating the clock-to-setup requirement, the data stored will be invalid.

If one port attempts to read from the same memory cell that the other is simultaneously writing to, violating the clock setup requirement, the write will be successful but the data read will be invalid.

Specifying Initial Contents of a Block RAM

You can use the INIT_0x attributes to specify an initial value during device configuration. The initialization of each RAMB4_Sm_Sn is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the INIT_xx section of the *Constraints Guide* for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template for RAMB4_Sm_Sn

```
-- RAMB4_S1_S1: Virtex/E, Spartan-II/IIE 4k x 1 Dual-Port RAM
-- The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_S1_inst : RAMB4_S1_S1
-- The following generic INIT_xx declarations are only necessary if you
-- wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
INIT_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0A => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0B => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0C => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0D => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0E => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0F => X"0000000000000000000000000000000000000000000000000000000000000000")
port map (
DOA => DOA,      -- Port A 1-bit data output
DOB => DOB,      -- Port B 1-bit data output
ADDRA => ADDR,   -- Port A 12-bit address input
```

```

ADDRB => ADDR_B, -- Port B 12-bit address input
CLKA => CLK_A, -- Port A clock input
CLKB => CLK_B, -- Port B clock input
DIA => DIA, -- Port A 1-bit data input
DIB => DIB, -- Port B 1-bit data input
ENA => ENA, -- Port A RAM enable input
ENB => ENB, -- Port B RAM enable input
RSTA => RSTA, -- Port A Synchronous reset input
RSTB => RSTB, -- Port B Synchronous reset input
WEA => WEA, -- Port A RAM write enable input
WEB => WEB, -- Port B RAM write enable input
);

-- End of RAMB4_S1_S1_inst instantiation

```

Verilog Instantiation Template for RAMB4_Sm_Sn

All RAMB4_Sm_Sn templates are listed below. Find and cut the specific configuration you need for your design.

-- Note that the use of INIT below is for simulation only. For examples of how to include INIT as an implementation constraint, please refer to the *Constraints Guide*.

// RAMB4_S1_S1: Virtex/E, Spartan-II/IIE 4k x 1 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

```

RAMB4_S1_S1 RAMB4_S1_S1_inst (
.DOA(DOA), // Port A 1-bit data output
.DOB(DOB), // Port B 1-bit data output
.ADDRA(ADDR_A), // Port A 12-bit address input
.ADDRB(ADDR_B), // Port B 12-bit address input
.CLKA(CLK_A), // Port A clock input
.CLKB(CLK_B), // Port B clock input
.DIA(DIA), // Port A 1-bit data input
.DIB(DIB), // Port B 1-bit data input
.ENA(ENA), // Port A RAM enable input
.ENB(ENB), // Port B RAM enable input
.RSTA(RSTA), // Port A Synchronous reset input
.RSTB(RSTB), // Port B Synchronous reset input
.WEA(WEA), // Port A RAM write enable input
.WEB(WEB) // Port B RAM write enable input
);

```

// The following defparam INIT_xx declarations are only necessary if you wish to change the initial contents of the RAM to anything other than all zero's. If the instance name to the RAM is changed, that change needs to be reflected in the defparam statements.

```

defparam RAMB4_S1_S1_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```

defparam RAMB4_S1_S1_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S1_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S1_S1_inst instantiation

// RAMB4_S1_S2: Virtex/E, Spartan-II/IIE 4k/2k x 1/2 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_S2 RAMB4_S1_S2_inst (
    .DOA(DOA),      // Port A 1-bit data output
    .DOB(DOB),      // Port B 2-bit data output
    .ADDRA(ADDRA), // Port A 12-bit address input
    .ADDRB(ADDRB), // Port B 11-bit address input
    .CLKA(CLKA),   // Port A clock input
    .CLKB(CLKB),   // Port B clock input
    .DIA(DIA),     // Port A 1-bit data input
    .DIB(DIB),     // Port B 2-bit data input
    .ENA(ENA),     // Port A RAM enable input
    .ENB(ENB),     // Port B RAM enable input
    .RSTA(RSTA),   // Port A Synchronous reset input
    .RSTB(RSTB),   // Port B Synchronous reset input
    .WEA(WEA),     // Port A RAM write enable input
    .WEB(WEB)      // Port B RAM write enable input
);

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S1_S2_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S2_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S1_S2_inst instantiation

// RAMB4_S1_S4: Virtex/E, Spartan-II/IIE 4k/1k x 1/4 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_S4 RAMB4_S1_S4_inst (
    .DOA(DOA),      // Port A 1-bit data output
    .DOB(DOB),      // Port B 4-bit data output
    .ADDRA(ADDRA), // Port A 12-bit address input
    .ADDRB(ADDRB), // Port B 10-bit address input

```

```

        .CLKA(CLKA), // Port A clock input
        .CLKB(CLKB), // Port B clock input
        .DIA(DIA), // Port A 1-bit data input
        .DIB(DIB), // Port B 4-bit data input
        .ENA(ENA), // Port A RAM enable input
        .ENB(ENB), // Port B RAM enable input
        .RSTA(RSTA), // Port A Synchronous reset input
        .RSTB(RSTB), // Port B Synchronous reset input
        .WEA(WEA), // Port A RAM write enable input
        .WEB(WEB) // Port B RAM write enable input
    );

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S1_S4_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S4_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S1_S4_inst instantiation

// RAMB4_S1_S8: Virtex/E, Spartan-II/IIE 4k/512 x 1/8 Dual-Port RAM
//The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_S8 RAMB4_S1_S8_inst (
    .DOA(DOA), // Port A 1-bit data output
    .DOB(DOB), // Port B 8-bit data output
    .ADDRA(ADDRA), // Port A 12-bit address input
    .ADDRB(ADDRB), // Port B 9-bit address input
    .CLKA(CLKA), // Port A clock input
    .CLKB(CLKB), // Port B clock input
    .DIA(DIA), // Port A 1-bit data input
    .DIB(DIB), // Port B 8-bit data input
    .ENA(ENA), // Port A RAM enable input
    .ENB(ENB), // Port B RAM enable input
    .RSTA(RSTA), // Port A Synchronous reset input
    .RSTB(RSTB), // Port B Synchronous reset input
    .WEA(WEA), // Port A RAM write enable input
    .WEB(WEB) // Port B RAM write enable input
);

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,

```

```

// that change needs to be reflected in the defparam statements.

defparam RAMB4_S1_S8_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S8_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S1_S8_inst instantiation

// RAMB4_S1_S16: Virtex/E, Spartan-II/IIE 4k/256 x 1/16 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_S16 RAMB4_S1_S16_inst (
    .DOA(DOA),      // Port A 1-bit data output
    .DOB(DOB),      // Port B 16-bit data output
    .ADDRA(ADDRA), // Port A 12-bit address input
    .ADDRB(ADDRB), // Port B 8-bit address input
    .CLKA(CLKA),   // Port A clock input
    .CLKB(CLKB),   // Port B clock input
    .DIA(DIA),     // Port A 1-bit data input
    .DIB(DIB),     // Port B 16-bit data input
    .ENA(ENA),     // Port A RAM enable input
    .ENB(ENB),     // Port B RAM enable input
    .RSTA(RSTA),   // Port A Synchronous reset input
    .RSTB(RSTB),   // Port B Synchronous reset input
    .WEA(WEA),     // Port A RAM write enable input
    .WEB(WEB)      // Port B RAM write enable input
);

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S1_S16_inst.INIT_00=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_01=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_02=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_03=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_04=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_05=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_06=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_07=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_08=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_09=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_0A=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_0B=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_0C=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_0D=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_S16_inst.INIT_0E=256'h0000000000000000000000000000000000000000000000000000000000000000;

```



```

defparam RAMB4_S2_S8_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S8_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```
// End of RAMB4_S2_S8_inst instantiation
```

```
// RAMB4_S2_S16: Virtex/E, Spartan-II/IIE 2k/256 x 2/16 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide
```

```

RAMB4_S2_S16 RAMB4_S2_S16_inst (
    .DOA(DOA),      // Port A 2-bit data output
    .DOB(DOB),      // Port B 16-bit data output
    .ADDRA(ADDRA), // Port A 11-bit address input
    .ADDRB(ADDRB), // Port B 8-bit address input
    .CLKA(CLKA),   // Port A clock input
    .CLKB(CLKB),   // Port B clock input
    .DIA(DIA),     // Port A 2-bit data input
    .DIB(DIB),     // Port B 16-bit data input
    .ENA(ENA),     // Port A RAM enable input
    .ENB(ENB),     // Port B RAM enable input
    .RSTA(RSTA),   // Port A Synchronous reset input
    .RSTB(RSTB),   // Port B Synchronous reset input
    .WEA(WEA),     // Port A RAM write enable input
    .WEB(WEB),     // Port B RAM write enable input
);

```

```
// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.
```

```

defparam RAMB4_S2_S16_inst.INIT_00=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_01=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_02=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_03=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_04=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_05=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_06=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_07=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_08=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_09=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0A=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0B=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0C=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0D=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0E=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S2_S16_inst.INIT_0F=256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```
// End of RAMB4_S2_S16_inst instantiation
```



```

        .RSTB(RSTB), // Port B Synchronous reset input
        .WEA(WEA), // Port A RAM write enable input
        .WEB(WEB) // Port B RAM write enable input
    );

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S4_S8_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S8_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S4_S8_inst instantiation

// RAMB4_S4_S16: Virtex/E, Spartan-II/IIE 1k/256 x 4/16 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB4_S4_S16 RAMB4_S4_S16_inst (
    .DOA(DOA), // Port A 4-bit data output
    .DOB(DOB), // Port B 16-bit data output
    .ADDRA(ADDRA), // Port A 10-bit address input
    .ADDRB(ADDRB), // Port B 8-bit address input
    .CLKA(CLKA), // Port A clock input
    .CLKB(CLKB), // Port B clock input
    .DIA(DIA), // Port A 4-bit data input
    .DIB(DIB), // Port B 16-bit data input
    .ENA(ENA), // Port A RAM enable input
    .ENB(ENB), // Port B RAM enable input
    .RSTA(RSTA), // Port A Synchronous reset input
    .RSTB(RSTB), // Port B Synchronous reset input
    .WEA(WEA), // Port A RAM write enable input
    .WEB(WEB) // Port B RAM write enable input
);

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S4_S16_inst.INIT_00=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_01=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_02=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_03=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_04=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_05=256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```

defparam RAMB4_S4_S16_inst.INIT_06=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_07=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_08=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_09=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0A=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0B=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0C=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0D=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0E=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_S16_inst.INIT_0F=256'h0000000000000000000000000000000000000000000000000000000000000000;

    // End of RAMB4_S4_S16_inst instantiation

// RAMB4_S8_S8: Virtex/E, Spartan-II/IIE 512 x 8 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

    RAMB4_S8_S8 RAMB4_S8_S8_inst (
        .DOA(DOA),      // Port A 8-bit data output
        .DOB(DOB),      // Port B 8-bit data output
        .ADDRA(ADDRA),  // Port A 9-bit address input
        .ADDRB(ADDRB),  // Port B 9-bit address input
        .CLKA(CLKA),    // Port A clock input
        .CLKB(CLKB),    // Port B clock input
        .DIA(DIA),      // Port A 8-bit data input
        .DIB(DIB),      // Port B 8-bit data input
        .ENA(ENA),      // Port A RAM enable input
        .ENB(ENB),      // Port B RAM enable input
        .RSTA(RSTA),    // Port A Synchronous reset input
        .RSTB(RSTB),    // Port B Synchronous reset input
        .WEA(WEA),      // Port A RAM write enable input
        .WEB(WEB)       // Port B RAM write enable input
    );

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S8_S8_inst.INIT_00 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_01 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_02 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_03 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_04 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_05 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_06 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_07 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_08 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_09 =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0A =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0B =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0C =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0D =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0E =256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_S8_inst.INIT_0F =256'h0000000000000000000000000000000000000000000000000000000000000000;

    // End of RAMB4_S8_S8_inst instantiation

// RAMB4_S8_S16: Virtex/E, Spartan-II/IIE 512/256 x 8/16 Dual-Port RAM
// The current version of the Xilinx HDL Libraries Guide

```



```

);

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

defparam RAMB4_S16_S16_inst.INIT_00
=256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_02 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_03 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_04 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_05 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_06 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_07 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_08 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_09 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0A =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0B =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0C =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0D =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0E =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_S16_inst.INIT_0F =
256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB4_S16_S16_inst instantiation

```

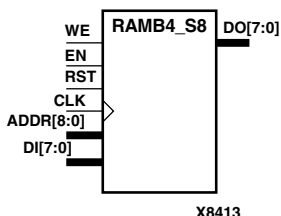
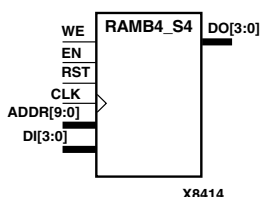
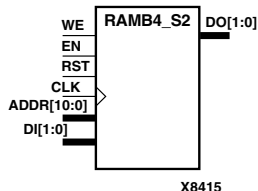
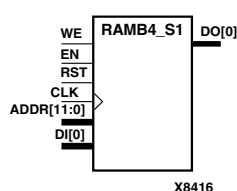
Commonly Used Constraints

INIT_xx

RAMB4_Sn

4096-Bit Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 8, or 16 Bits

RAMB4_Sn	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



RAMB4_S1, RAMB4_S2, RAMB4_S4, RAMB4_S8, and RAMB4_S16 are dedicated random access memory blocks with synchronous write capability. They provide the capability for fast, discrete, large blocks of RAM in each Virtex, Virtex-E, Spartan-II, and Spartan-IIE device. The RAMB4_Sn cell configurations are listed in the following table.

Component	Depth	Width	Address Bus	Data Bus
RAMB4_S1	4096	1	(11:0)	(0:0)
RAMB4_S2	2048	2	(10:0)	(1:0)
RAMB4_S4	1024	4	(9:0)	(3:0)
RAMB4_S8	512	8	(8:0)	(7:0)
RAMB4_S16	256	16	(7:0)	(15:0)

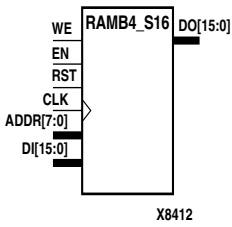
The enable (EN) pin controls read, write, and reset. When EN is Low, no data is written and the output (DO) retains the last state. When EN is High and reset (RST) is High, DO is cleared during the Low-to-High clock (CLK) transition; if write enable (WE) is High, the memory contents reflect the data at DI. When EN is High and WE is Low, the data stored in the RAM address (ADDR) is read during the Low-to-High clock transition. When EN and WE are High, the data on the data input (DI) is loaded into the word selected by the write address (ADDR) during the Low-to-High clock transition and the data output (DO) reflects the selected (addressed) word.

The above description assumes an active High EN, WE, RST, and CLK. However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB4 port is absorbed into the block and does not use a CLB resource.

RAMB4_Sn 's may be initialized during configuration. See [“Specifying Initial Contents of a Block RAM”](#) below.

Block RAM output registers are asynchronously cleared, output Low, when power is applied. The initial contents of the block RAM are not altered.

Virtex, Virtex-E, Spartan-II, and Spartan-IIE simulate power-on when global set/reset (GSR) is active.



GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Mode selection is shown in the following truth table.

Inputs						Outputs	
EN	RST	WE	CLK	ADDR	DI	DO	RAM Contents
0	X	X	X	X	X	No Chg	No Chg
1	1	0	↑	X	X	0	No Chg
1	1	1	↑	addr	data	0	RAM(addr) =>data
1	0	0	↑	addr	X	RAM(addr)	No Chg
1	0	1	↑	addr	data	data	RAM(addr) =>data

addr=RAM address

RAM(addr)=RAM contents at address ADDR

data=RAM input data

Specifying Initial Contents of a Block RAM

You can use the INIT_xx attributes to specify an initial value during device configuration. The initialization of each RAMB4_Sn is set by 16 initialization attributes (INIT_00 through INIT_0F) of 64 hex values for a total of 4096 bits. See the INIT_xx section of the *Constraints Guide* for more information on these attributes.

If any INIT_0x attribute is not specified, it is configured as zeros. Partial initialization strings are padded with zeros to the left.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template for RAMB4_Sn

```

-- RAMB4_S1: Virtex/E, Spartan-II/IIE 4k x 1 Single-Port RAM
-- The current version of the Xilinx HDL Libraries Guide

RAMB4_S1_inst : RAMB4_S1
-- The following generic INIT_xx declarations are only necessary
-- if you wish to change the initial
-- contents of the RAM to anything other than all zero's.
generic map (
INIT_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0A => X"0000000000000000000000000000000000000000000000000000000000000000",

```

```

INIT_0B => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0C => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0D => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0E => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0F => X"0000000000000000000000000000000000000000000000000000000000000000")

port map (
    DO => DO,      -- 1-bit data output
    ADDR => ADDR, -- 12-bit address input
    CLK => CLK,    -- Clock input
    DI => DI,      -- 1-bit data input
    EN => EN,      -- RAM enable input
    RST => RST,    -- Synchronous reset input
    WE => WE       -- RAM write enable input
);

-- End of RAMB4_S1_inst instantiation

```

Verilog Instantiation Template for RAMB4_Sn

-- Note that the use of INIT below is for simulation only. For examples
 -- of how to include INIT as an implementation constraint,
 -- please refer to the *Constraints Guide*.

ALL RAMB4_Sn TEMPLATES ARE LISTED BELOW. FIND AND CUT THE SPECIFIC
 CONFIGURATION YOU NEED FOR YOUR DESIGN:

```
// RAMB4_S1: Virtex/E, Spartan-II/IIE 4k x 1 Single-Port RAM
// The current version of the Xilinx HDL Libraries Guide
```

```

RAMB4_S1 RAMB4_S1_inst (
    .DO(DO),      // 1-bit data output
    .ADDR(ADDR), // 12-bit address input
    .CLK(CLK),   // Clock input
    .DI(DI),     // 1-bit data input
    .EN(EN),     // RAM enable input
    .RST(RST),   // Synchronous reset input
    .WE(WE)      // RAM write enable input
);

```

```
// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.
```

```

defparam RAMB4_S1_inst.INIT_00 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_01 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_02 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_03 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_08 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_09 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S1_inst.INIT_0F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

```



```

defparam RAMB4_S4_inst.INIT_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_08 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_09 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S4_inst.INIT_0F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam RAMB4_S8_inst.INIT_00 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_01 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_02 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_03 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_08 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_09 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S8_inst.INIT_0F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```
// End of RAMB4_S8_inst instantiation
```

```
// RAMB4_S16: Virtex/E, Spartan-II/IIE 256 x 16 Single-Port RAM
// The current version of the Xilinx HDL Libraries Guide
```

```

RAMB4_S16 RAMB4_S16_inst (
    .DO(DO), // 16-bit data output
    .ADDR(ADDR), // 8-bit address input
    .CLK(CLK), // Clock input
    .DI(DI), // 16-bit data input
    .EN(EN), // RAM enable input
    .RST(RST), // Synchronous reset input
    .WE(WE) // RAM write enable input
);

```

```

// The following defparam INIT_xx declarations are only necessary if
// you wish to change the initial contents of the RAM to anything
// other than all zero's. If the instance name to the RAM is changed,
// that change needs to be reflected in the defparam statements.

```

```

defparam RAMB4_S16_inst.INIT_00 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_01 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_02 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_03 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_08 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_09 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB4_S16_inst.INIT_0A = 256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```
defparam RAMB4_S16_inst.INIT_0B =
```

Commonly Used Constraints

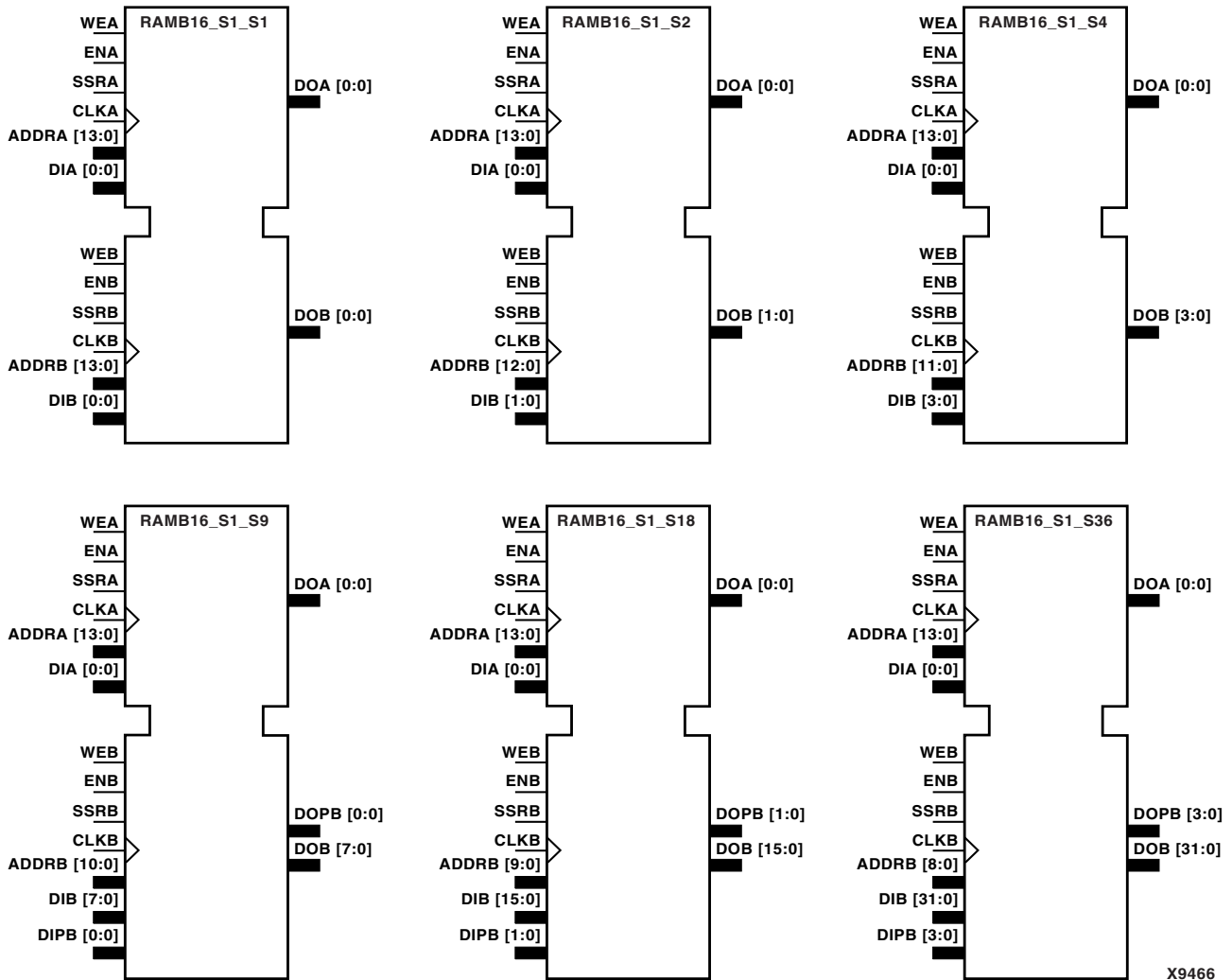
INIT_xx

RAMB16_Sm_Sn

16384-Bit Data Memory and 2048-Bit Parity Memory, Dual-Port Synchronous Block RAM with Port Width (m or n) Configured to 1, 2, 4, 9, 18, or 36 Bits

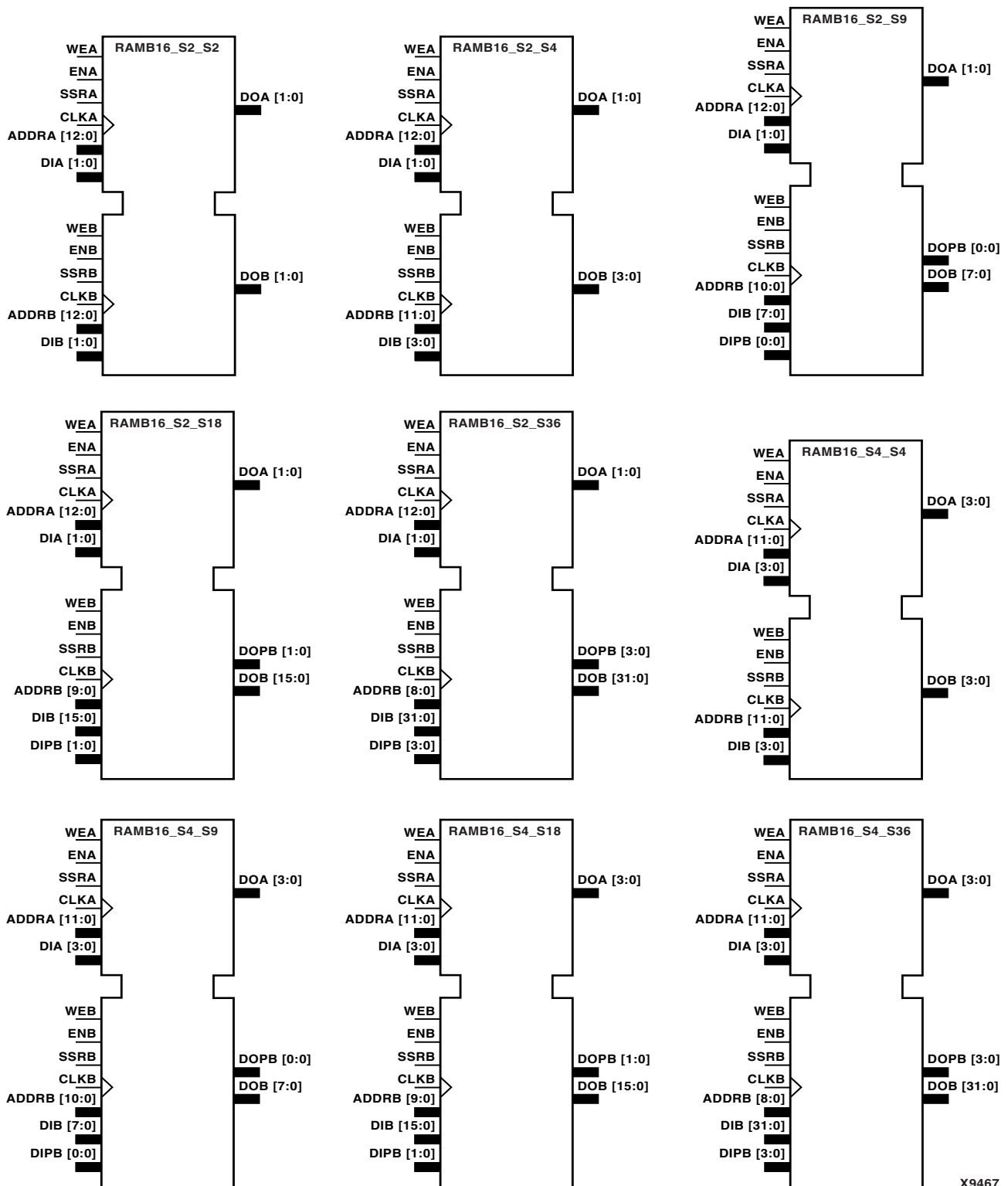
Architectures Supported

RAMB16_Sm_Sn	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



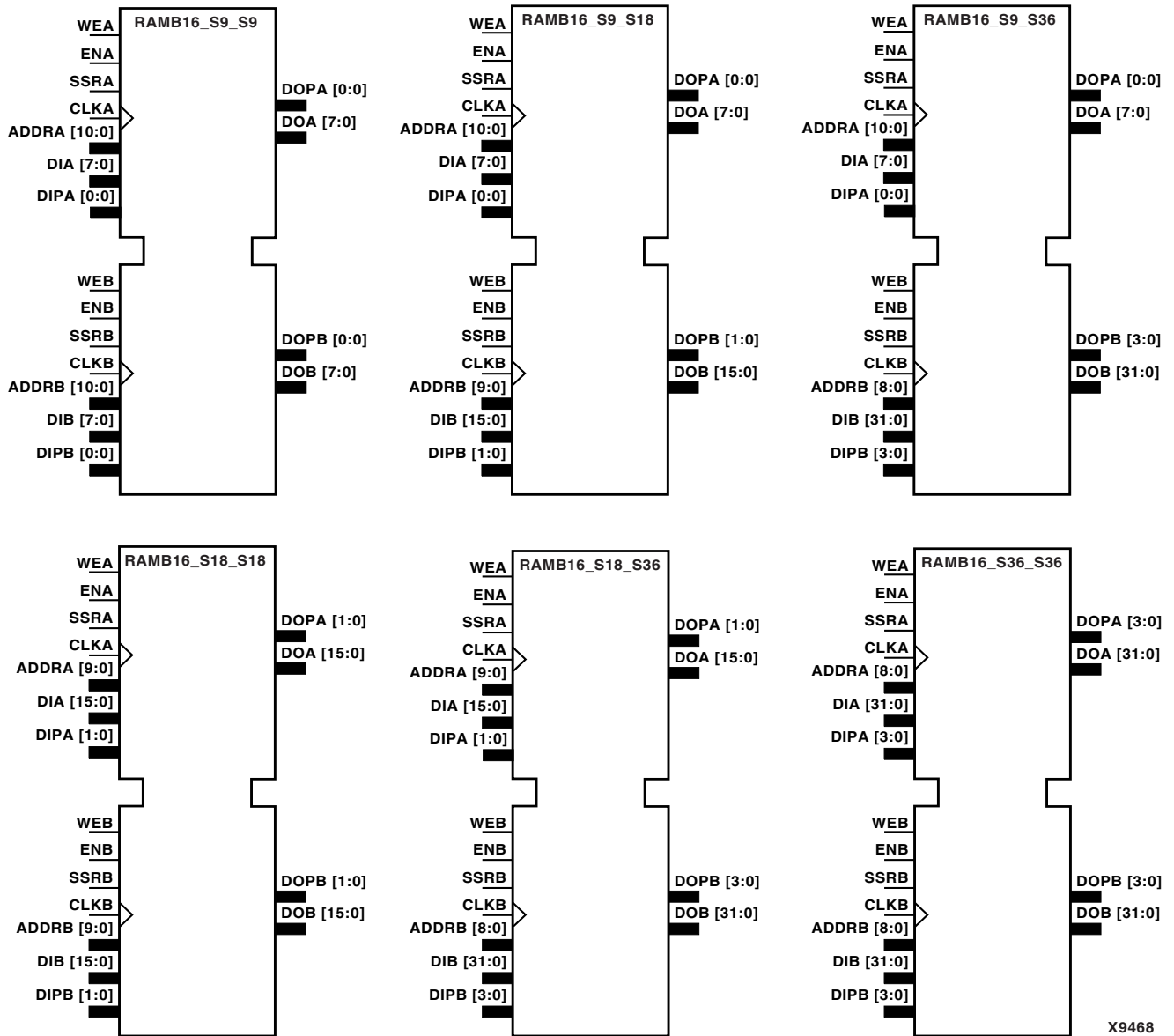
X9466

RAMB16_S1_S1 through RAMB16_S1_S36 Representations



X9467

RAMB16_S2_S2 through RAMB16_S4_S36 Representations



RAMB16_S9_S9 through RAMB16_S36_S36 Representations

The RAMB16_Sm_Sn components listed in the following table are dual-ported dedicated random access memory blocks with synchronous write capability. Each block RAM port has 16384 bits of data memory. Ports configured as 9, 18, or 36-bits wide have an additional 2048 bits of parity memory. Each port is independent of the other while accessing the same set of 16384 data memory cells. Each port is independently configured to a specific data width. The possible port and cell configurations are listed in the following table.

Component	Port A					Port B				
	Data Cells ^d	Parity Cells ^d	Address Bus	Data Bus	Parity Bus	Data Cells ^d	Parity Cells ^d	Address Bus	Data Bus	Parity Bus
RAMB16_S1_S1	16384 x 1	-	(13:0)	(0:0)	-	16384 x 1	-	(13:0)	(0:0)	-
RAMB16_S1_S2	16384 x 1	-	(13:0)	(0:0)	-	8192 x 2	-	(12:0)	(1:0)	-
RAMB16_S1_S4	16384 x 1	-	(13:0)	(0:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S1_S9	16384 x 1	-	(13:0)	(0:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S1_S18	16384 x 1	-	(13:0)	(0:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S1_S36	16384 x 1	-	(13:0)	(0:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S2_S2	8192 x 2	-	(12:0)	(1:0)	-	8192 x 2	-	(12:0)	(1:0)	-
RAMB16_S2_S4	8192 x 2	-	(12:0)	(1:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S2_S9	8192 x 2	-	(12:0)	(1:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S2_S18	8192 x 2	-	(12:0)	(1:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S2_S36	8192 x 2	-	(12:0)	(1:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S4_S4	4096 x 4	-	(11:0)	(3:0)	-	4096 x 4	-	(11:0)	(3:0)	-
RAMB16_S4_S9	4096 x 4	-	(11:0)	(3:0)	-	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S4_S18	4096 x 4	-	(11:0)	(3:0)	-	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S4_S36	4096 x 4	-	(11:0)	(3:0)	-	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S9_S9	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)
RAMB16_S9_S18	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S9_S36	2048 x 8	2048 x 1	(10:0)	(7:0)	(0:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S18_S18	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)
RAMB16_S18_S36	1024 x 16	1024 x 2	(9:0)	(15:0)	(1:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)
RAMB16_S36_S36	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)	512 x 32	512 x 4	(8:0)	(31:0)	(3:0)

^dDepth x Width

Each port is fully synchronous with independent clock pins. All port A input pins have setup time referenced to the CLKA pin and its data output bus DOA has a clock-to-out time referenced to the CLKA. All port B input pins have setup time referenced to the CLKB pin and its data output bus DOB has a clock-to-out time referenced to the CLKB.

The enable ENA pin controls read, write, and reset for port A. When ENA is Low, no data is written and the outputs (DOA and DOPA) retain the last state. When ENA is High and reset (SSRA) is High, DOA and DOPA are set to SRVAL_A during the Low-to-High clock (CLKA) transition; if write enable (WEA) is High, the memory contents reflect the data at DIA and DIPA. When ENA is High and WEA is Low, the data stored in the RAM address (ADDRA) is read during the Low-to-High clock transition. By default, WRITE_MODE_A=WRITE_FIRST, when ENA and WEA are High, the data on the data inputs (DIA and DIPA) is loaded into the word selected by the write address (ADDRA) during the Low-to-High clock transition and the data outputs (DOA and DOPA) reflect the selected (addressed) word.

The enable ENB pin controls read, write, and reset for port B. When ENB is Low, no data is written and the outputs (DOB and DOPB) retain the last state. When ENB is High and reset (SSRB) is High, DOB and DOPB are set to SRVAL_B during the Low-to-High clock (CLKB) transition; if write enable (WEB) is High, the memory contents reflect the data at DIB and DIPB. When ENB is High and WEB is Low, the data stored in the RAM address (ADDRB) is read during the Low-to-High clock transition. By default, WRITE_MODE_B=WRITE_FIRST, when ENB and WEB are High, the data on the data inputs (DIB and DIPB) are loaded into the word selected by the write address (ADDRB) during the Low-to-High clock transition and the data outputs (DOB and DOPB) reflect the selected (addressed) word.

The above descriptions assume active High control pins (ENA, WEA, SSRA, CLKA, ENB, WEB, SSRB, and CLKB). However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB16 port is absorbed into the block and does not use a CLB resource.

Port A Truth Table

Inputs								Outputs			
GS R	ENA	SSR A	WE A	CLK A	ADD RA	DIA	DIPA	DOA	DOPA	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT_A	INIT_A	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL_A	SRVAL_A	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL_A	SRVAL_A	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg

Port A Truth Table

Inputs								Outputs			
GS R	ENA	SSR A	WE A	CLK A	ADD RA	DIA	DIPA	DOA	DOPA	RAM Contents	
0	1	0	1	↑	addr	data	pdata	No Chg ¹ RAM (addr) ² data ³	No Chg ¹ RAM(addr) ² pdata ³	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset

INIT_A=Value specified by the INIT_A attribute for output register. Default is all zeros.

SRVAL_A=register value

addr=RAM address

RAM(addr)=RAM contents at address ADDR

data=RAM input data

pdata=RAM parity data

¹WRITE_MODE_A=NO_CHANGE

²WRITE_MODE_A=READ_FIRST

³WRITE_MODE_A=WRITE_FIRST

Port B Truth Table

Inputs								Outputs			
GS R	ENB	SSR B	WE B	CLK B	ADD RB	DIB	DIPB	DOB	DOPB	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT_B	INIT_B	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL_B	SRVAL_B	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL_B	SRVAL_B	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg
0	1	0	1	↑	addr	data	pdata	No Chg ¹ RAM (addr) ² data ³	No Chg ¹ RAM(addr) ² pdata ³	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset

INIT_B=Value specified by the INIT_B attribute for output registers. Default is all zeros.

SRVAL_B=register value

addr=RAM address

RAM(addr)=RAM contents at address ADDR

data=RAM input data

pdata=RAM parity data

¹WRITE_MODE_B=NO_CHANGE

²WRITE_MODE_B=READ_FIRST

³WRITE_MODE_B=WRITE_FIRST

Address Mapping

Each port accesses the same set of 18432 memory cells using an addressing scheme that is dependent on the width of the port. For all port widths, 16384 memory cells are available for data as shown in the “Port Address Mapping for Data” table. For 9-, 18-, and 36-bit wide ports, 2408 parity memory cells are also available as shown in “Port Address Mapping for Parity” table. The physical RAM location that is addressed for a particular width is determined from the following formula.

$$\text{Start} = (\text{ADDR}_{\text{port}} + 1) * (\text{Width}_{\text{port}}) - 1$$

$$\text{End} = \text{ADDR}_{\text{port}} * (\text{Width}_{\text{port}})$$

The following tables shows address mapping for each port width.

Port Address Mapping for Data

Data Width	Port Data Addresses																																	
1	16384	←	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
2	8192	←	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00																
4	4096	←	07	06	05	04	03	02	01	00																								
8	2048	←	03	02	01	00																												
16	1024	←	01	00																														
32	512	←	00																															

Port Address Mapping for Parity

Parity Width	Port Parity Addresses					
1	2048	←	03	02	01	00
2	1024	←	01	00		
4	512	←	00			

Initializing Memory Contents of a Dual-Port RAMB16

You can use the INIT_xx attributes to specify an initialization value for the memory contents of a RAMB16 during device configuration. The initialization of each RAMB16_Sm_Sn is set by 64 initialization attributes (INIT_00 through INIT_3F) of 64 hex values for a total of 16384 bits.

You can use the INITP_xx attributes to specify an initial value for the parity memory during device configuration or assertion. The initialization of the parity memory for ports configured for 9, 18, or 36 bits is set by 8 initialization attributes (INITP_00 through INITP_07) of 64 hex values for a total of 2048 bits.

If any INIT_xx or INITP_xx attribute is not specified, it is configured as zeros. Partial strings are padded with zeros to the left.

See the *Constraints Guide* for more information on these attributes.

Initializing the Output Register of a Dual-Port RAMB16

In Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, each bit in an output register can be initialized at power on (when GSR is high) to either a 0 or 1. In addition, the initial state specified for power on can be different than the state that results from assertion of a set/reset. Four properties control initialization of the output register for a dual-port RAMB16: INIT_A, INIT_B, SRVAL_A, and SRVAL_B. The INIT_A attribute specifies the output register value at power on for port A and the INIT_B attribute specifies the value for port B. You can use the SRVAL_A attribute to define the state resulting from assertion of the SSR (set/reset) input on port A. You can use the SRVAL_B attribute to define the state resulting from assertion of the SSR input on port B.

The INIT_A, INIT_B, SRVAL_A, and SRVAL_B attributes specify the initialization value as a hexadecimal string. The value is dependent upon the port width. For example, for a RAMB16_S1_S4 with port A width equal to 1 and port B width equal to 4, the port A output register contains 1 bit and the port B output register contains 4 bits. Therefore, the INIT_A or SRVAL_A value can only be specified as a 1 or 0. For port B, the output register contains 4 bits. In this case, you can use INIT_B or SRVAL_B to specify a hexadecimal value from 0 through F to initialize the 4 bits of the output register.

For those ports that include parity bits, the parity portion of the output register is specified in the high order bit position of the INIT_A, INIT_B, SRVAL_A, or SRVAL_B value.

The INIT and SRVAL attributes default to zero if they are not set by the user.

See the *Constraints Guide* for more information on these attributes.

Write Mode Selection

The WRITE_MODE_A attribute controls the memory and output contents of port A for a dual-port RAMB16. The WRITE_MODE_B attribute does the same for port B. By default, both WRITE_MODE_A and WRITE_MODE_B are set to WRITE_FIRST. This means that input is read, written to memory, and then passed to output. You can set the write mode for port A and/or port B to READ_FIRST to read the memory contents, pass the memory contents to the outputs, and then write the input to memory. Or, you can set the write mode to NO_CHANGE to have the input written to memory without changing the output. The “Port A and Port B Conflict Resolution” section describes how read/write conflicts are resolved when both port A and port B are attempting to read/write to the same memory cells.

Port A and Port B Conflict Resolution

Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X block SelectRAM is True Dual-Port RAM that allows both ports to simultaneously access the same memory cell. When one port writes to a given memory cell, the other port must not address that memory cell (for a write or a read) within the clock-to-clock setup window. For a list of specifics of conflict resolution for port and memory cell write operations that have either a clock common to both ports or synchronous clocks on each port, see *Virtex-II Handbook, Chapter 2, Design Considerations, Using BlockSelectRAM Memory, Conflict Resolution*.

The following tables summarize the collision detection behavior of the dual-port RAMB16 based on the WRITE_MODE_A and WRITE_MODE_B settings.

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=NO_CHANGE

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	No Chg	X	No Chg	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	No Chg	No Chg	No Chg	X	X

WRITE_MODE_A=READ_FIRST and WRITE_MODE_B=READ_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	X	X

WRITE_MODE_A= WRITE_FIRST and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	DIA	X	DIPA	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	X	X	X	X	X	X

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=READ_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIB	DIPB

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	DIA	DIPA

WRITE_MODE_A=NO_CHANGE and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	No Chg	X	No Chg	X	X	X

WRITE_MODE_A=READ_FIRST and WRITE_MODE_B=WRITE_FIRST

WEA	WEB	CLKA	CLKB	DIA	DIB	DIPA	DIPB	DOA	DOB	DOPA	DOPB	Data RAM	Parity Ram
0	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	No Chg	No Chg
1	0	↑	↑	DIA	DIB	DIPA	DIPB	RAM	RAM	RAM	RAM	DIA	DIPA
0	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIB	DIPB
1	1	↑	↑	DIA	DIB	DIPA	DIPB	X	DIB	X	DIPB	DIA	DIPA

Usage

For HDL, these design elements can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template for RAMB16_S1_S1, RAMB16_S1_S2, RAMB16_S1_S4, RAMB16_S2_S2, RAMB16_S2_S4, and RAMB16_S4_S4

```
-- RAMB16_S1: Virtex-II/II-Pro, Spartan-3 16kx1 Single-Port RAM
-- The current version of the Xilinx HDL Libraries Guide

RAMB16_S1_inst : RAMB16_S1
generic map (
  -- The following generics are only necessary if you wish to
  change the default behavior.
  INIT => "0", -- Value of output RAM registers at startup
  SRVAL => "0", -- Ouput value upon SSR assertion
  WRITE_MODE => "WRITE_FIRST", -- WRITE_FIRST, READ_FIRST or
  NO_CHANGE
  -- The following generic INIT_xx declarations are only necessary
  if you wish to change the initial
  -- contents of the RAM to anything other than all zero's.
  INIT_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0A => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0B => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0C => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0D => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0E => X"0000000000000000000000000000000000000000000000000000000000000000",
```



```
-- Component Attribute specification for RAM16X1D
-- should be placed after architecture declaration but
-- before the begin keyword
```

```
Put attributes if necessary
```

```
-- Component Instantiation for RAMB16_Sm_Sn should be placed
-- in architecture after the begin keyword
```

```
-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn
```

```
-- synthesis translate_off
generic map (
    INIT_00 => vector_value,
    INIT_01 => vector_value,
    INIT_02 => vector_value,
    INIT_03 => vector_value,
    INIT_04 => vector_value,
    INIT_05 => vector_value,
    INIT_06 => vector_value,
    INIT_07 => vector_value,
    INIT_08 => vector_value,
    INIT_09 => vector_value,
    INIT_0A => vector_value,
    INIT_0B => vector_value,
    INIT_0C => vector_value,
    INIT_0D => vector_value,
    INIT_0E => vector_value,
    INIT_0F => vector_value,
    INIT_10 => vector_value,
    INIT_11 => vector_value,
    INIT_12 => vector_value,
    INIT_13 => vector_value,
    INIT_14 => vector_value,
    INIT_15 => vector_value,
    INIT_16 => vector_value,
    INIT_17 => vector_value,
    INIT_18 => vector_value,
    INIT_19 => vector_value,
    INIT_1A => vector_value,
    INIT_1B => vector_value,
    INIT_1C => vector_value,
    INIT_1D => vector_value,
    INIT_1E => vector_value,
    INIT_1F => vector_value,
    INIT_20 => vector_value,
    INIT_21 => vector_value,
    INIT_22 => vector_value,
    INIT_23 => vector_value,
    INIT_24 => vector_value,
    INIT_25 => vector_value,
    INIT_26 => vector_value,
    INIT_27 => vector_value,
    INIT_28 => vector_value,
    INIT_29 => vector_value,
```

```

INIT_2A => vector_value,
INIT_2B => vector_value,
INIT_2C => vector_value,
INIT_2D => vector_value,
INIT_2E => vector_value,
INIT_2F => vector_value,
INIT_30 => vector_value,
INIT_31 => vector_value,
INIT_32 => vector_value,
INIT_33 => vector_value,
INIT_34 => vector_value,
INIT_35 => vector_value,
INIT_36 => vector_value,
INIT_37 => vector_value,
INIT_38 => vector_value,
INIT_39 => vector_value,
INIT_3A => vector_value,
INIT_3B => vector_value,
INIT_3C => vector_value,
INIT_3D => vector_value,
INIT_3E => vector_value,
INIT_3F => vector_value,
INIT_A => bit_value,
INIT_B => bit_value,
INITP_00 => vector_value,
INITP_01 => vector_value,
INITP_02 => vector_value,
INITP_03 => vector_value,
INITP_04 => vector_value,
INITP_05 => vector_value,
INITP_06 => vector_value,
INITP_07 => vector_value,
SRVAL_A => bit_value,
SRVAL_B => bit_value,
WRITE_MODE_A => string_value,
WRITE_MODE_B => string_value)
-- synopsys translate_on
port map (DOA => user_DOA,
         DOB => user_DOB,
         ADDRA => user_ADDRA,
         ADDR1 => user_ADDR1,
         ADDR2 => user_ADDR2,
         CLKA => user_CLKA,
         CLKB => user_CLKB,
         DIA => user_DIA,
         DIB => user_DIB,
         ENA => user_ENA,
         ENB => user_ENB,
         SSRA => user_SSRA,
         SSRB => user_SSRB,
         WEA => user_WEA,
         WEB => user_WEB);

```

Verilog Instantiation Template for RAMB16_S1_S1, RAMB16_S1_S2, RAMB16_S1_S4, RAMB16_S2_S2, RAMB16_S2_S4, and RAMB16_S4_S4

-- Note that the use of INIT below is for simulation only. For examples

```
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.
```

```
RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                  .DOB (user_DOB),
                                  .ADDRA (user_ADDRA),
                                  .ADDRB (user_ADDRB),
                                  .CLKA (user_CLKA),
                                  .CLKB (user_CLKB),
                                  .DIA (user_DIA),
                                  .DIB (user_DIB),
                                  .ENA (user_ENA),
                                  .ENB (user_ENB),
                                  .SSRA (user_SSRA),
                                  .SSRB (user_SSRB),
                                  .WEA (user_WEA),
                                  .WEB (user_WEB));

defparam user_instance_name.INIT_00 = 256_bit_hex_value;
defparam user_instance_name.INIT_01 = 256_bit_hex_value;
defparam user_instance_name.INIT_02 = 256_bit_hex_value;
defparam user_instance_name.INIT_03 = 256_bit_hex_value;
defparam user_instance_name.INIT_04 = 256_bit_hex_value;
defparam user_instance_name.INIT_05 = 256_bit_hex_value;
defparam user_instance_name.INIT_06 = 256_bit_hex_value;
defparam user_instance_name.INIT_07 = 256_bit_hex_value;
defparam user_instance_name.INIT_08 = 256_bit_hex_value;
defparam user_instance_name.INIT_09 = 256_bit_hex_value;
defparam user_instance_name.INIT_0A = 256_bit_hex_value;
defparam user_instance_name.INIT_0B = 256_bit_hex_value;
defparam user_instance_name.INIT_0C = 256_bit_hex_value;
defparam user_instance_name.INIT_0D = 256_bit_hex_value;
defparam user_instance_name.INIT_0E = 256_bit_hex_value;
defparam user_instance_name.INIT_0F = 256_bit_hex_value;
defparam user_instance_name.INIT_10 = 256_bit_hex_value;
defparam user_instance_name.INIT_11 = 256_bit_hex_value;
defparam user_instance_name.INIT_12 = 256_bit_hex_value;
defparam user_instance_name.INIT_13 = 256_bit_hex_value;
defparam user_instance_name.INIT_14 = 256_bit_hex_value;
defparam user_instance_name.INIT_15 = 256_bit_hex_value;
defparam user_instance_name.INIT_16 = 256_bit_hex_value;
defparam user_instance_name.INIT_17 = 256_bit_hex_value;
defparam user_instance_name.INIT_18 = 256_bit_hex_value;
defparam user_instance_name.INIT_19 = 256_bit_hex_value;
defparam user_instance_name.INIT_1A = 256_bit_hex_value;
defparam user_instance_name.INIT_1B = 256_bit_hex_value;
defparam user_instance_name.INIT_1C = 256_bit_hex_value;
defparam user_instance_name.INIT_1D = 256_bit_hex_value;
defparam user_instance_name.INIT_1E = 256_bit_hex_value;
defparam user_instance_name.INIT_1F = 256_bit_hex_value;
defparam user_instance_name.INIT_20 = 256_bit_hex_value;
defparam user_instance_name.INIT_21 = 256_bit_hex_value;
defparam user_instance_name.INIT_22 = 256_bit_hex_value;
defparam user_instance_name.INIT_23 = 256_bit_hex_value;
defparam user_instance_name.INIT_24 = 256_bit_hex_value;
defparam user_instance_name.INIT_25 = 256_bit_hex_value;
defparam user_instance_name.INIT_26 = 256_bit_hex_value;
defparam user_instance_name.INIT_27 = 256_bit_hex_value;
```

```

defparam user_instance_name.INIT_28 = 256_bit_hex_value;
defparam user_instance_name.INIT_29 = 256_bit_hex_value;
defparam user_instance_name.INIT_2A = 256_bit_hex_value;
defparam user_instance_name.INIT_2B = 256_bit_hex_value;
defparam user_instance_name.INIT_2C = 256_bit_hex_value;
defparam user_instance_name.INIT_2D = 256_bit_hex_value;
defparam user_instance_name.INIT_2E = 256_bit_hex_value;
defparam user_instance_name.INIT_2F = 256_bit_hex_value;
defparam user_instance_name.INIT_30 = 256_bit_hex_value;
defparam user_instance_name.INIT_31 = 256_bit_hex_value;
defparam user_instance_name.INIT_32 = 256_bit_hex_value;
defparam user_instance_name.INIT_33 = 256_bit_hex_value;
defparam user_instance_name.INIT_34 = 256_bit_hex_value;
defparam user_instance_name.INIT_35 = 256_bit_hex_value;
defparam user_instance_name.INIT_36 = 256_bit_hex_value;
defparam user_instance_name.INIT_37 = 256_bit_hex_value;
defparam user_instance_name.INIT_38 = 256_bit_hex_value;
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.INIT_A = bit_value;
defparam user_instance_name.INIT_B = bit_value;
defparam user_instance_name.INITP_00 = 256_bit_hex_value;
defparam user_instance_name.INITP_01 = 256_bit_hex_value;
defparam user_instance_name.INITP_02 = 256_bit_hex_value;
defparam user_instance_name.INITP_03 = 256_bit_hex_value;
defparam user_instance_name.INITP_04 = 256_bit_hex_value;
defparam user_instance_name.INITP_05 = 256_bit_hex_value;
defparam user_instance_name.INITP_06 = 256_bit_hex_value;
defparam user_instance_name.INITP_07 = 256_bit_hex_value;
defparam user_instance_name.SRVAL_A = bit_value;
defparam user_instance_name.SRVAL_B = bit_value;
defparam user_instance_name.WRITE_MODE_A = string_value;
defparam user_instance_name.WRITE_MODE_B = string_value;

```

VHDL Instantiation Template for RAMB16_S1_S9, RAMB16_S1_S18, RAMB16_S1_S36, RAMB16_S2_S9, RAMB16_S2_S18, RAMB16_S2_S36, RAMB16_S4_S9, RAMB16_S4_S18, and RAMB16_S4_S36

```

-- Component Declaration for RAMB16_S1_{S9 | S18 | S36}, RAMB16_S2_{S9 |
  S18 | S36}, or RAMB16_S4_{S9 | S18 | S36} should be placed
-- after architecture statement but before begin keyword

```

```

-- Note that the use of INIT below is for simulation only
-- For examples on how to include INIT as an implementation
-- constraint, please refer to the Constraints Guide

```

```

component RAMB16_Sm_Sn
  -- synthesis translate_off
  generic (
    INIT_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_01 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";

```



```

INITP_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_01 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        SRVAL_A : bit_vector := X"0";
        SRVAL_B : bit_vector := X"0";
        WRITE_MODE_A : string := "WRITE_FIRST";
        WRITE_MODE_B : string := "WRITE_FIRST";
    );
    -- synthesis translate_on

    port (DOA : out STD_LOGIC_VECTOR (n downto 0);
          DOB : out STD_LOGIC_VECTOR (n downto 0);
          DOPB : out STD_LOGIC_VECTOR (n downto 0);
          ADDRA : in STD_LOGIC_VECTOR (n downto 0);
          ADDR_B : in STD_LOGIC_VECTOR (n downto 0);
          CLKA : in STD_ULOGIC;
          CLKB : in STD_ULOGIC;
          DIA : in STD_LOGIC_VECTOR (n downto 0);
          DIB : in STD_LOGIC_VECTOR (n downto 0);
          DIPB : in STD_LOGIC_VECTOR (n downto 0);
          ENA : in STD_ULOGIC;
          ENB : in STD_ULOGIC;
          SSRA : in STD_ULOGIC;
          SSRB : in STD_ULOGIC;
          WEA : in STD_ULOGIC;
          WEB : in STD_ULOGIC);

end component;

-- Component Attribute specification for RAMB16_Sm_Sn
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for RAMB16_Sm_Sn should be placed
-- in architecture after the begin keyword

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn

-- synthesis translate_off
generic map (
    INIT_00 => vector_value,
    INIT_01 => vector_value,
    INIT_02 => vector_value,
    INIT_03 => vector_value,
    INIT_04 => vector_value,
    INIT_05 => vector_value,
    INIT_06 => vector_value,

```



```
INIT_07 => vector_value,  
INIT_08 => vector_value,  
INIT_09 => vector_value,  
INIT_0A => vector_value,  
INIT_0B => vector_value,  
INIT_0C => vector_value,  
INIT_0D => vector_value,  
INIT_0E => vector_value,  
INIT_0F => vector_value,  
INIT_10 => vector_value,  
INIT_11 => vector_value,  
INIT_12 => vector_value,  
INIT_13 => vector_value,  
INIT_14 => vector_value,  
INIT_15 => vector_value,  
INIT_16 => vector_value,  
INIT_17 => vector_value,  
INIT_18 => vector_value,  
INIT_19 => vector_value,  
INIT_1A => vector_value,  
INIT_1B => vector_value,  
INIT_1C => vector_value,  
INIT_1D => vector_value,  
INIT_1E => vector_value,  
INIT_1F => vector_value,  
INIT_20 => vector_value,  
INIT_21 => vector_value,  
INIT_22 => vector_value,  
INIT_23 => vector_value,  
INIT_24 => vector_value,  
INIT_25 => vector_value,  
INIT_26 => vector_value,  
INIT_27 => vector_value,  
INIT_28 => vector_value,  
INIT_29 => vector_value,  
INIT_2A => vector_value,  
INIT_2B => vector_value,  
INIT_2C => vector_value,  
INIT_2D => vector_value,  
INIT_2E => vector_value,  
INIT_2F => vector_value,  
INIT_30 => vector_value,  
INIT_31 => vector_value,  
INIT_32 => vector_value,  
INIT_33 => vector_value,  
INIT_34 => vector_value,  
INIT_35 => vector_value,  
INIT_36 => vector_value,  
INIT_37 => vector_value,  
INIT_38 => vector_value,  
INIT_39 => vector_value,  
INIT_3A => vector_value,  
INIT_3B => vector_value,  
INIT_3C => vector_value,  
INIT_3D => vector_value,  
INIT_3E => vector_value,  
INIT_3F => vector_value,  
INIT_A => bit_value,
```

```

INIT_B => bit_value,
INITP_00 => vector_value,
INITP_01 => vector_value,
INITP_02 => vector_value,
INITP_03 => vector_value,
INITP_04 => vector_value,
INITP_05 => vector_value,
INITP_06 => vector_value,
INITP_07 => vector_value,
SRVAL_A => bit_value,
SRVAL_B => bit_value,
WRITE_MODE_A => string_value,
WRITE_MODE_B => string_value)
-- synopsys translate_on
port map (DOA => user_DOA,
         DOB => user_DOB,
         DOPB => user_DOPB,
         ADDRA => user_ADDRA,
         ADDR8 => user_ADDR8,
         CLKA => user_CLKA,
         CLKB => user_CLKB,
         DIA => user_DIA,
         DIB => user_DIB,
         DIPB => user_DIPB,
         ENA => user_ENA,
         ENB => user_ENB,
         SSRA => user_SSRA,
         SSRB => user_SSRB,
         WEA => user_WEA,
         WEB => user_WEB);

```

Verilog Instantiation Template for RAMB16_S1_S9, RAMB16_S1_S18, RAMB16_S1_S36, RAMB16_S2_S9, RAMB16_S2_S18, RAMB16_S2_S36, RAMB16_S4_S9, RAMB16_S4_S18, and RAMB16_S4_S36

```

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

```

```

RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                .DOB (user_DOB),
                                .DOPB (user_DOPB),
                                .ADDRA (user_ADDRA),
                                .ADDR8 (user_ADDR8),
                                .CLKA (user_CLKA),
                                .CLKB (user_CLKB),
                                .DIA (user_DIA),
                                .DIB (user_DIB),
                                .DIPB (user_DIPB),
                                .ENA (user_ENA),
                                .ENB (user_ENB),
                                .SSRA (user_SSRA),
                                .SSRB (user_SSRB),
                                .WEA (user_WEA),

```



```

defparam user_instance_name.INIT_38 = 256_bit_hex_value;
defparam user_instance_name.INIT_39 = 256_bit_hex_value;
defparam user_instance_name.INIT_3A = 256_bit_hex_value;
defparam user_instance_name.INIT_3B = 256_bit_hex_value;
defparam user_instance_name.INIT_3C = 256_bit_hex_value;
defparam user_instance_name.INIT_3D = 256_bit_hex_value;
defparam user_instance_name.INIT_3E = 256_bit_hex_value;
defparam user_instance_name.INIT_3F = 256_bit_hex_value;
defparam user_instance_name.INIT_A = bit_value;
defparam user_instance_name.INIT_B = bit_value;
defparam user_instance_name.INITP_00 = 256_bit_hex_value;
defparam user_instance_name.INITP_01 = 256_bit_hex_value;
defparam user_instance_name.INITP_02 = 256_bit_hex_value;
defparam user_instance_name.INITP_03 = 256_bit_hex_value;
defparam user_instance_name.INITP_04 = 256_bit_hex_value;
defparam user_instance_name.INITP_05 = 256_bit_hex_value;
defparam user_instance_name.INITP_06 = 256_bit_hex_value;
defparam user_instance_name.INITP_07 = 256_bit_hex_value;
defparam user_instance_name.SRVAL_A = bit_value;
defparam user_instance_name.SRVAL_B = bit_value;
defparam user_instance_name.WRITE_MODE_A = string_value;
defparam user_instance_name.WRITE_MODE_B = string_value;

```

VHDL Instantiation Template RAMB16_S9_S9, RAMB16_S9_S18, RAMB16_S9_S36, RAMB16_S18_S18, RAMB16_S18_S36, and RAMB16_S36_S36

```

-- RAMB16_S18: Virtex-II/II-Pro, Spartan-3 1k x 16 + 2 Parity bits
-- Single-Port RAM
-- The current version of the Xilinx HDL Libraries Guide

RAMB16_S18_inst : RAMB16_S18
generic map (
  -- The following generics are only necessary if you wish to
  -- change the default behavior.
  INIT => X"00000", -- Value of output RAM registers at startup
  SRVAL => X"00000", -- Ouput value upon SSR assertion
  WRITE_MODE => "WRITE_FIRST", -- WRITE_FIRST, READ_FIRST or
  NO_CHANGE
  -- The following generic INIT_xx declarations are only necessary
  -- if you wish to change the initial
  -- contents of the RAM to anything other than all zero's.
  INIT_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0A => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0B => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0C => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0D => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0E => X"0000000000000000000000000000000000000000000000000000000000000000",
  INIT_0F => X"0000000000000000000000000000000000000000000000000000000000000000",

```



```

        DIPA => DIPA,      -- Port A 2-bit parity Input
        DIPB => DIPB,      -- Port-B 4-bit parity Input
        ENA => ENA,        -- Port A RAM Enable Input
    ENB => ENB,          -- PortB RAM Enable Input
        SSRA => SSRA,      -- Port A Synchronous Set/Reset Input
        SSRB => SSRB,      -- Port B Synchronous Set/Reset Input
        WEA => WEA,        -- Port A Write Enable Input
        WEB => WEB         -- Port B Write Enable Input
    );

-- End of RAMB16_S18_S36_inst instantiation

-- Component Attribute specification for RAM16X1D
-- should be placed after architecture declaration but
-- before the begin keyword

Put attributes if necessary

-- Component Instantiation for RAMB16_Sm_Sn should be placed
-- in architecture after the begin keyword

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

RAMB16_Sm_Sn INSTANCE_NAME : RAMB16_Sm_Sn

-- synthesis translate_off
generic map (
    INIT_00 => vector_value,
    INIT_01 => vector_value,
    INIT_02 => vector_value,
    INIT_03 => vector_value,
    INIT_04 => vector_value,
    INIT_05 => vector_value,
    INIT_06 => vector_value,
    INIT_07 => vector_value,
    INIT_08 => vector_value,
    INIT_09 => vector_value,
    INIT_0A => vector_value,
    INIT_0B => vector_value,
    INIT_0C => vector_value,
    INIT_0D => vector_value,
    INIT_0E => vector_value,
    INIT_0F => vector_value,
    INIT_10 => vector_value,
    INIT_11 => vector_value,
    INIT_12 => vector_value,
    INIT_13 => vector_value,
    INIT_14 => vector_value,
    INIT_15 => vector_value,
    INIT_16 => vector_value,
    INIT_17 => vector_value,
    INIT_18 => vector_value,
    INIT_19 => vector_value,
    INIT_1A => vector_value,
    INIT_1B => vector_value,

```



```
INIT_1C => vector_value,  
INIT_1D => vector_value,  
INIT_1E => vector_value,  
INIT_1F => vector_value,  
INIT_20 => vector_value,  
INIT_21 => vector_value,  
INIT_22 => vector_value,  
INIT_23 => vector_value,  
INIT_24 => vector_value,  
INIT_25 => vector_value,  
INIT_26 => vector_value,  
INIT_27 => vector_value,  
INIT_28 => vector_value,  
INIT_29 => vector_value,  
INIT_2A => vector_value,  
INIT_2B => vector_value,  
INIT_2C => vector_value,  
INIT_2D => vector_value,  
INIT_2E => vector_value,  
INIT_2F => vector_value,  
INIT_30 => vector_value,  
INIT_31 => vector_value,  
INIT_32 => vector_value,  
INIT_33 => vector_value,  
INIT_34 => vector_value,  
INIT_35 => vector_value,  
INIT_36 => vector_value,  
INIT_37 => vector_value,  
INIT_38 => vector_value,  
INIT_39 => vector_value,  
INIT_3A => vector_value,  
INIT_3B => vector_value,  
INIT_3C => vector_value,  
INIT_3D => vector_value,  
INIT_3E => vector_value,  
INIT_3F => vector_value,  
INIT_A => bit_value,  
INIT_B => bit_value,  
INITP_00 => vector_value,  
INITP_01 => vector_value,  
INITP_02 => vector_value,  
INITP_03 => vector_value,  
INITP_04 => vector_value,  
INITP_05 => vector_value,  
INITP_06 => vector_value,  
INITP_07 => vector_value,  
SRVAL_A => bit_value,  
SRVAL_B => bit_value,  
WRITE_MODE_A => string_value,  
WRITE_MODE_B => string_value)  
-- synopsys translate_on  
port map (DOA => user_DOA,  
         DOB => user_DOB,  
         DOPA => user_DOPA,  
         DOPB => user_DOPB,  
         ADDRA => user_ADDRA,  
         ADDR_B => user_ADDRB,  
         CLKA => user_CLKA,
```

```

CLKB => user_CLKB,
DIA => user_DIA,
DIB => user_DIB,
DIPA => user_DIPA,
DIPB => user_DIPB,
ENA => user_ENA,
ENB => user_ENB,
SSRA => user_SSRA,
SSRB => user_SSRB,
WEA => user_WEA,
WEB => user_WEB);

```

Verilog Instantiation Template for RAMB16_S9_S9, RAMB16_S9_S18, RAMB16_S9_S36, RAMB16_S18_S18, RAMB16_S18_S36, and RAMB16_S36_S36

```

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the Constraints Guide.

```

```

RAMB16_Sm_Sn user_instance_name (.DOA (user_DOA),
                                .DOB (user_DOB),
                                .DOPA (user_DOPA),
                                .DOPB (user_DOPB),
                                .ADDRA (user_ADDRA),
                                .ADDRB (user_ADDRB),
                                .CLKA (user_CLKA),
                                .CLKB (user_CLKB),
                                .DIA (user_DIA),
                                .DIB (user_DIB),
                                .DIPA (user_DIPA),
                                .DIPB (user_DIPB),
                                .ENA (user_ENA),
                                .ENB (user_ENB),
                                .SSRA (user_SSRA),
                                .SSRB (user_SSRB),
                                .WEA (user_WEA),
                                .WEB (user_WEB));

```

```

defparam user_instance_name.INIT_00 = 256_bit_hex_value;
defparam user_instance_name.INIT_01 = 256_bit_hex_value;
defparam user_instance_name.INIT_02 = 256_bit_hex_value;
defparam user_instance_name.INIT_03 = 256_bit_hex_value;
defparam user_instance_name.INIT_04 = 256_bit_hex_value;
defparam user_instance_name.INIT_05 = 256_bit_hex_value;
defparam user_instance_name.INIT_06 = 256_bit_hex_value;
defparam user_instance_name.INIT_07 = 256_bit_hex_value;
defparam user_instance_name.INIT_08 = 256_bit_hex_value;
defparam user_instance_name.INIT_09 = 256_bit_hex_value;
defparam user_instance_name.INIT_0A = 256_bit_hex_value;
defparam user_instance_name.INIT_0B = 256_bit_hex_value;
defparam user_instance_name.INIT_0C = 256_bit_hex_value;
defparam user_instance_name.INIT_0D = 256_bit_hex_value;
defparam user_instance_name.INIT_0E = 256_bit_hex_value;
defparam user_instance_name.INIT_0F = 256_bit_hex_value;
defparam user_instance_name.INIT_10 = 256_bit_hex_value;
defparam user_instance_name.INIT_11 = 256_bit_hex_value;

```



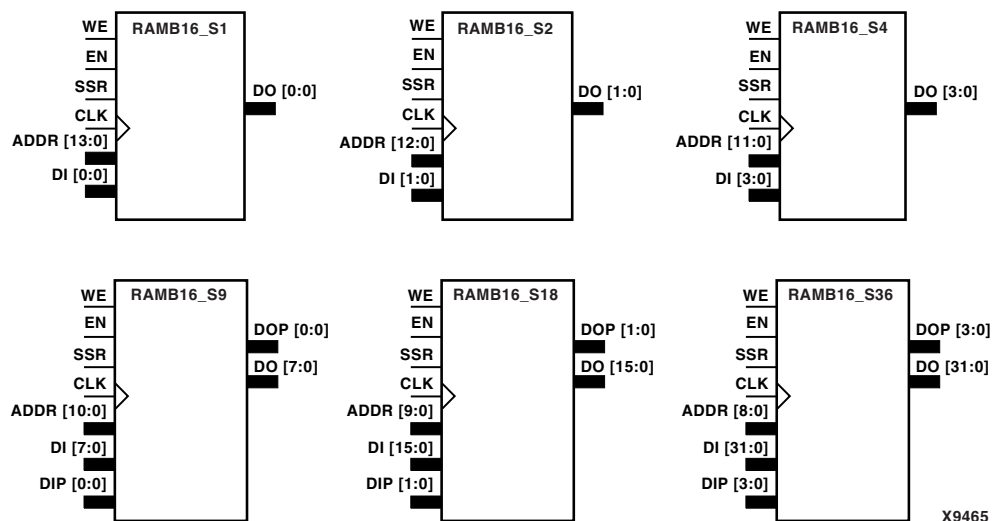
```
defparam user_instance_name.WRITE_MODE_A = string_value;  
defparam user_instance_name.WRITE_MODE_B = string_value;
```

RAMB16_Sn

16384-Bit Data Memory and 2048-Bit Parity Memory, Single-Port Synchronous Block RAM with Port Width (n) Configured to 1, 2, 4, 9, 18, or 36 Bits

Architectures Supported

RAMB16_Sn	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9465

RAMB16_S1 through RAMB16_S36 Representations

RAMB16_S1, RAMB16_S2, RAMB16_S4, RAMB16_S9, RAMB16_S18, and RAMB16_S36 are dedicated random access memory blocks with synchronous write capability. The block RAM port has 16384 bits of data memory. RAMB16_S9, RAMB16_S18, and RAMB16_S36 have an additional 2048 bits of parity memory. The RAMB16_Sn cell configurations are listed in the following table.

The enable (EN) pin controls read, write, and reset. When EN is Low, no data is

Component	Data Cells		Parity Cells		Address Bus	Data Bus	Parity Bus
	Depth	Width	Depth	Width			
RAMB16_S1	16384	1	-	-	(13:0)	(0:0)	-
RAMB16_S2	8192	2	-	-	(12:0)	(1:0)	-
RAMB16_S4	4096	4	-	-	(11:0)	(3:0)	-
RAMB16_S9	2048	8	2048	1	(10:0)	(7:0)	(0:0)
RAMB16_S18	1024	16	1024	2	(9:0)	(15:0)	(1:0)
RAMB16_S36	512	32	512	4	(8:0)	(31:0)	(3:0)

written and the outputs (DO and DOP) retain the last state. When EN is High and reset (SSR) is High, DO and DOP are set to SRVAL during the Low-to-High clock (CLK) transition; if write enable (WE) is High, the memory contents reflect the data at DI and DIP. When SSR is Low, EN is High, and WE is Low, the data stored in the RAM address (ADDR) is read during the Low-to-High clock transition. The output value depends on the mode. By default WRITE_MODE=WRITE_FIRST, when EN and WE are High and SSR is Low, the data on the data inputs (DI and DIP) is loaded into the word selected by the write address (ADDR) during the Low-to-High clock transition. See ["Write Mode Selection"](#) for information on setting the WRITE_MODE.

The above description assumes an active High EN, WE, SSR, and CLK. However, the active level can be changed by placing an inverter on the port. Any inverter placed on a RAMB16 port is absorbed into the block and does not use a CLB resource.

GS R	Inputs							Outputs			
	EN	SSR	WE	CLK	ADD R	DI	DIP	DO	DOP	RAM Contents	
										Data RAM	Parity RAM
1	X	X	X	X	X	X	X	INIT	INIT	No Chg	No Chg
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
0	1	1	0	↑	X	X	X	SRVAL	SRVAL	No Chg	No Chg
0	1	1	1	↑	addr	data	pdata	SRVAL	SRVAL	RAM(addr) =>data	RAM(addr) =>pdata
0	1	0	0	↑	addr	X	X	RAM(addr)	RAM(addr)	No Chg	No Chg

Inputs								Outputs			
GS R	EN	SSR	WE	CLK	ADD R	DI	DIP	DO	DOP	RAM Contents	
0	1	0	1	↑	addr	data	pdata	No Chg ^a RAM (addr) ^b data ^c	No Chg ^a RAM(addr) ^b pdata ^c	RAM(addr) =>data	RAM(addr) =>pdata

GSR=Global Set Reset signal

INIT=Value specified by the INIT attribute for data memory. Default is all zeros.

SRVAL=Value after assertion of SSR as specified by the SRVAL attribute.

addr=RAM address

RAM(addr)=RAM contents at address ADDR

data=RAM input data

pdata=RAM parity data

^aWRITE_MODE=NO_CHANGE

^bWRITE_MODE=READ_FIRST

^cWRITE_MODE=WRITE_FIRST

Initializing Memory Contents of a Single-Port RAMB16

You can use the INIT_xx attributes to specify an initialization value for the memory contents of a RAMB16 during device configuration. The initialization of each RAMB16_Sn is set by 64 initialization attributes (INIT_00 through INIT_3F) of 64 hex values for a total of 16384 bits.

You can use the INITP_xx attributes to specify an initial value for the parity memory during device configuration or assertion. The initialization of the parity memory for ports configured for 9, 18, or 36 bits is set by 8 initialization attributes (INITP_00 through INITP_07) of 64 hex values for a total of 2048 bits.

If any INIT_xx or INITP_xx attribute is not specified, it is configured as zeros. Partial strings are padded with zeros to the left.

See the *Constraints Guide* for more information on these attributes.

Initializing the Output Register of a Single-Port RAMB16

In Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X, each bit in the output register can be initialized at power on to either a 0 or 1. In addition, the initial state specified for power on can be different than the state that results from assertion of a set/reset. Two types of properties control initialization of the output register for a single-port RAMB16: INIT and SRVAL. The INIT attribute specifies the output register value at power on. You can use the SRVAL attribute to define the state resulting from assertion of the SSR (set/reset) input.

The INIT and SRVAL attributes specify the initialization value as a hexadecimal string. The value is dependent upon the port width. For example, for a RAMB16_S1 with port width equal to 1, the output register contains 1 bit. Therefore, the INIT or SRVAL value can only be specified as a 1 or 0. For RAMB16_S4 with port width equal to 4, the output register contains 4 bits. In this case, you can specify a hexadecimal value from 0 through F to initialize the 4 bits of the output register.

For those ports that include parity bits, the parity portion of the output register is specified in the high order bit position of the INIT or SRVAL value.

The INIT and SRVAL attributes default to zero if they are not set by the user.

See the *Constraints Guide* for more information on these attributes.

Write Mode Selection

The WRITE_MODE attribute controls RAMB16 memory and output contents. By default, the WRITE_MODE is set to WRITE_FIRST. This means that input is read, written to memory, and then passed to output. You can set the WRITE_MODE to READ_FIRST to read the memory contents, pass the memory contents to the outputs, and then write the input to memory. Or, you can set the WRITE_MODE to NO_CHANGE to have the input written to memory without changing the output.

Usage

For HDL, this design element can be inferred or instantiated. The instantiation code is shown below. For information on how to infer RAM, see the *XST User Guide*.

VHDL Instantiation Template for RAMB16_S1, S2, and S4

```
RAMB16_S4
-- RAMB16_S4: Virtex-II/II-Pro, Spartan-3 4k x 4 Single-Port RAM
-- The current version of the Xilinx HDL Libraries Guide

RAMB16_S4_inst : RAMB16_S4
generic map (
  -- The following generics are only necessary if you
  -- wish to change the default behavior.
  INIT => X"0", -- Value of output RAM registers at startup
  SRVAL => X"0", -- Ouput value upon SSR assertion
  WRITE_MODE => "WRITE_FIRST", -- WRITE_FIRST, READ_FIRST or
  NO_CHANGE
  -- The following generic INIT_xx declarations are only
  -- necessary if you wish to change the initial
  -- contents of the RAM to anything other than all zero's.
INIT_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0A => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0B => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0C => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0D => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0E => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_0F => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_10 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_11 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_12 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_13 => X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_14 => X"0000000000000000000000000000000000000000000000000000000000000000",
```



```

DOB => DOB,      -- Port B 32-bit Data Output
DOPB => DOPB,    -- Port B 4-bit Parity Output
ADDRA => ADDR A 12-bit Address Input
ADDRB => ADDR B 9-bit Address Input
CLKA => CLKA,    -- Port A Clock
CLKB => CLKB,    -- Port B Clock
DIA => DIA,      -- Port A 4-bit Data Input
DIB => DIB,      -- Port B 32-bit Data Input
DIPB => DIPB,    -- Port-B 4-bit parity Input
ENA => ENA,      -- Port A RAM Enable Input
ENB => ENB,      -- PortB RAM Enable Input
SSRA => SSRA,    -- Port A Synchronous Set/Reset Input
SSRB => SSRB,    -- Port B Synchronous Set/Reset Input
WEA => WEA,      -- Port A Write Enable Input
WEB => WEB,      -- Port B Write Enable Input
);

-- End of RAMB16_S4_S36_inst instantiation

```

Verilog Instantiation Template for RAMB16_Sn

All RAMB16_Sn templates are listed below. Find and cut the specific configuration you need for your design.

RAMB16_S1

-- Note that the use of INIT below is for simulation only. For examples
-- of how to include INIT as an implementation constraint,
-- please refer to the *Constraints Guide*.

```

// RAMB16_S1: Virtex-II/II-Pro,
// Spartan-3 16kx1 Single-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB16_S1 RAMB16_S1_inst (
    .DO(DO),      // 1-bit Data Output
    .ADDR(ADDR), // 14-bit Address Input
    .CLK(CLK),   // Clock
    .DI(DI),     // 1-bit Data Input
    .EN(EN),     // RAM Enable Input
    .SSR(SSR),   // Synchronous Set/Reset Input
    .WE(WE)      // Write Enable Input
);

// The following defparam declarations are only necessary if you
// wish to change the default behavior of the RAM. If the instance
// name is changed, these defparams need to be updated accordingly.

defparam RAMB16_S1_inst.INIT = 1'h0;
    // Value of output RAM registers at startup
defparam RAMB16_S1_inst.SRVAL = 1'h0;
    // Output value upon SSR assertion
defparam RAMB16_S1_inst.WRITE_MODE = "WRITE_FIRST";
    // WRITE_FIRST, READ_FIRST or NO_CHANGE

// The following defparam INIT_xx declarations are only
// necessary if you wish to change the initial
// contents of the RAM to anything other than all zero's.

```



```

defparam RAMB16_S2_inst.INIT_1D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_1E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_1F = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_20 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_21 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_22 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_23 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_24 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_25 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_26 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_27 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_28 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_29 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_2F = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_30 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_31 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_32 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_33 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_34 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_35 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_36 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_37 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_38 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_39 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S2_inst.INIT_3F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

```

```
// End of RAMB16_S2_inst instantiation
```

RAMB16_S4

```

// RAMB16_S4: Virtex-II/II-Pro,
// Spartan-3 4k x 4 Single-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB16_S4 RAMB16_S4_inst (
    .DO(DO),          // 4-bit Data Output
    .ADDR(ADDR),     // 12-bit Address Input
    .CLK(CLK),       // Clock
    .DI(DI),         // 4-bit Data Input
    .EN(EN),         // RAM Enable Input
    .SSR(SSR),       // Synchronous Set/Reset Input
    .WE(WE)          // Write Enable Input
);

// The following defparam declarations are only necessary if you
// wish to change the default behavior of the RAM. If the instance
// name is changed, these defparams need to be updated accordingly.

defparam RAMB16_S4_inst.INIT = 1'h0;
// Value of output RAM registers at startup
defparam RAMB16_S4_inst.SRVAL = 1'h0;
// Output value upon SSR assertion
defparam RAMB16_S4_inst.WRITE_MODE = "WRITE_FIRST";

```



```
defparam RAMB16_S18_inst.INIT_27 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_28 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_29 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_2F = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_30 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_31 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_32 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_33 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_34 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_35 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_36 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_37 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_38 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_39 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INIT_3F = 256'h0000000000000000000000000000000000000000000000000000000000000000;
// The next set of INITP_xx are for the parity bits
defparam RAMB16_S18_inst.INITP_00= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_01= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_02= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_03= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_04= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_05= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_06= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S18_inst.INITP_07= 256'h0000000000000000000000000000000000000000000000000000000000000000;
```

```
// End of RAMB16_S18_inst instantiation
```

RAMB16_S36

```
// RAMB16_S36: Virtex-II/II-Pro,
// Spartan-3 512 x 32 + 4 Parity bits Single-Port RAM
// The current version of the Xilinx HDL Libraries Guide

RAMB16_S36 RAMB16_S36_inst (
    .DO(DO),        // 32-bit Data Output
    .DOP(DOP),      // 4-bit parity Output
    .ADDR(ADDR),    // 9-bit Address Input
    .CLK(CLK),      // Clock
    .DI(DI),        // 32-bit Data Input
    .DIP(DIP),      // 4-bit parity Input
    .EN(EN),        // RAM Enable Input
    .SSR(SSR),      // Synchronous Set/Reset Input
    .WE(WE)         // Write Enable Input
);

// The following defparam declarations are only necessary if you
// wish to change the default behavior of the RAM. If the instance
// name is changed, these defparams need to be updated accordingly.

defparam RAMB16_S36_inst.INIT = 1'h0;
// Value of output RAM registers at startup
defparam RAMB16_S36_inst.SRVAL = 1'h0;
// Ouput value upon SSR assertion
```



```

defparam RAMB16_S36_inst.INIT_39 = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3A = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3B = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3C = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3D = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3E = 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INIT_3F = 256'h0000000000000000000000000000000000000000000000000000000000000000;
// The next set of INITP_xx are for the parity bits
defparam RAMB16_S36_inst.INITP_00= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_01= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_02= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_03= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_04= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_05= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_06= 256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam RAMB16_S36_inst.INITP_07= 256'h0000000000000000000000000000000000000000000000000000000000000000;

// End of RAMB16_S36_inst instantiation

```

VHDL Instantiation Template for RAMB16_S9, S18 and S36

```

-- Component Declaration for RAMB16_{S9 | S18 | S36} should be placed
-- after architecture statement but before begin keyword

-- Note that the use of INIT below is for simulation only
-- For examples on how to include INIT as an implementation
-- constraint, please refer to the Constraints Guide

```

```

component RAMB16_{S9 | S18 | S36}
  -- synthesis translate_off
  generic (
    INIT : bit_vector := X"0";
    INIT_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_01 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_08 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_09 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0A : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0B : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0C : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0D : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0E : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_0F : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_10 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_11 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_12 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_13 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_14 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_15 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_16 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_17 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_18 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_19 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1A : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1B : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1C : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
    INIT_1D : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";

```

```

INIT_1E : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_1F : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_20 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_21 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_22 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_23 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_24 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_25 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_26 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_27 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_28 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_29 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2A : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2B : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2C : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2D : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2E : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_2F : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_30 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_31 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_32 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_33 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_34 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_35 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_36 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_37 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_38 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_39 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3A : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3B : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3C : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3D : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3E : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_3F : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_01 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
INITP_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
SRVAL : bit_vector := X"0";
WRITE_MODE : string := "WRITE_FIRST";
);

-- synthesis translate_on

port (DO : out STD_LOGIC_VECTOR (0 downto 0);
      DOP : out STD_LOGIC_VECTOR (1 downto 0);
      ADDR : in STD_LOGIC_VECTOR (13 downto 0);
      CLK : in STD_ULONGIC;
      DI : in STD_LOGIC_VECTOR (0 downto 0);
      DIP : in STD_LOGIC_VECTOR (0 downto 0);
      EN : in STD_ULONGIC;
      SSR : in STD_ULONGIC;
      WE : in STD_ULONGIC);

end component;

-- Component Attribute specification for RAMB16_{S9 | S18 | S36}
-- should be placed after architecture declaration but
-- before the begin keyword

```

Enter attributes here

```
-- Component Instantiation for RAMB16_{S9 | S18 | S36} should be placed  
-- in architecture after the begin keyword
```

```
-- Note that the use of INIT below is for simulation only. For examples  
-- of how to include INIT as an implementation constraint,  
-- please refer to the Constraints Guide.
```

```
RAMB16_{S9 | S18 | S36}_INSTANCE_NAME : RAMB16_S1
```

```
-- synthesis translate_off  
generic map (  
    INIT => bit_value,  
    INIT_00 => vector_value,  
    INIT_01 => vector_value,  
    INIT_02 => vector_value,  
    INIT_03 => vector_value,  
    INIT_04 => vector_value,  
    INIT_05 => vector_value,  
    INIT_06 => vector_value,  
    INIT_07 => vector_value,  
    INIT_08 => vector_value,  
    INIT_09 => vector_value,  
    INIT_0A => vector_value,  
    INIT_0B => vector_value,  
    INIT_0C => vector_value,  
    INIT_0D => vector_value,  
    INIT_0E => vector_value,  
    INIT_0F => vector_value,  
    INIT_10 => vector_value,  
    INIT_11 => vector_value,  
    INIT_12 => vector_value,  
    INIT_13 => vector_value,  
    INIT_14 => vector_value,  
    INIT_15 => vector_value,  
    INIT_16 => vector_value,  
    INIT_17 => vector_value,  
    INIT_18 => vector_value,  
    INIT_19 => vector_value,  
    INIT_1A => vector_value,  
    INIT_1B => vector_value,  
    INIT_1C => vector_value,  
    INIT_1D => vector_value,  
    INIT_1E => vector_value,  
    INIT_1F => vector_value,  
    INIT_20 => vector_value,  
    INIT_21 => vector_value,  
    INIT_22 => vector_value,  
    INIT_23 => vector_value,  
    INIT_24 => vector_value,  
    INIT_25 => vector_value,  
    INIT_26 => vector_value,  
    INIT_27 => vector_value,  
    INIT_28 => vector_value,  
    INIT_3A => vector_value,  
    INIT_3B => vector_value,  
    INIT_3C => vector_value,
```



```
INIT_3D => vector_value,  
INIT_3E => vector_value,  
INIT_3F => vector_value,  
INITP_00 => vector_value,  
INITP_01 => vector_value,  
INITP_02 => vector_value,  
INITP_03 => vector_value,  
INITP_04 => vector_value,  
INITP_05 => vector_value,  
INITP_06 => vector_value,  
INITP_07 => vector_value  
SRVAL => bit_value,  
WRITE_MODE => user_WRITE_MODE)  
-- synopsys translate_on  
port map (DO => user_DO,  
          DOP => user_DOP,  
          ADDR => user_ADDR,  
          CLK => user_CLK,  
          DI => user_DI,  
          DIP => user_DIP,  
          EN => user_EN,  
          SSR => user_SSR,  
          WE => user_WE);
```

Commonly Used Constraints

```
INIT, INIT_xx, SRVAL, WRITE_MODE, HU_SET, INITP_xx, SRVAL,  
WRITE_MODE
```


ROC

Reset On Configuration

Architectures Supported

ROC	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

The ROC is a component used for VHDL simulation of FPGA designs. This component should not be used for Verilog or schematic entry. The ROC's function is to mimic the function of the internal reset signal during the FPGA configuration process. In order to use ROC, it must be connected to the reset/preset signal for all inferred and instantiated registers in the design. During synthesis and implementation, this reset signal will use the dedicated global set/reset network and will not use local routing resources. During simulation, ROC will emit a one-shot pulse for the amount of time specified by the WIDTH generic (default is 100 ns). This one-shot pulse is intended to reset all registers so that at the beginning of operation, all registers are at a known value as would happen in the real silicon during configuration of the device.

For more information, see the *Xilinx Synthesis and Verification Design Guide*.

Port O will be high at simulation time 0 for the amount of time specified by the WIDTH generic attribute. After that time, it will be 0. This will not affect implementation in any way.

VHDL Instantiation Code

```
component ROC
-- synthesis translate_off
  generic (WIDTH : Time := 100 ns);
-- synthesis translate_on
  port (O : out std_ulogic := '1');
end component;
```

Commonly Used Constraints

For simulation, the WIDTH generic can be modified to change the amount of time the one-shot pulse is applied for.

There are no supported constraints for this component for implementation.

ROCBUF

Reset On Configuration Buffer

Architectures Supported

ROCBUF	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

The ROCBUF is a component used for VHDL simulation of FPGA designs that is similar to the ROC component except that it contains an input for controlling the global set/reset function rather than a one-shot. This component should not be used for Verilog or schematic entry. The ROCBUF's function allows user control of the function of the global set/reset signal as done during the FPGA configuration process. In order to use the ROCBUF, the input should be connected to a top-level port in the design and the output must be connected to the reset/preset signal for all inferred and instantiated registers in the design.

During simulation, the input to the ROCBUF can be toggled by the testbench in order to activate the global set/reset signal in the design. This should be done at the beginning of the simulation as is done in the real silicon after configuration to get the design in a known state. The signal may also be pulsed during simulation to simulate a reconfiguration (ProG pin high) of the device. During synthesis and implementation, this reset signal will use the dedicated global set/reset network and will not use local routing resources. The port connected to this component will be optimized out of the design and not use any pin resources.

If you want to have the port implemented in the design, a `STARTBUF_architecture` should be used. In order to replace this port during back-end simulation the `-gp` switch should be used when invoking the netgen. If using the ISE GUI, use the "Bring Out Global Set/Reset Net as a Port" option in the Simulation Model Properties window.

For more information, see the *Xilinx Synthesis and Verification Design Guide*.

The value at port O will always be the value at port I (it is a buffer).

VHDL Instantiation Code

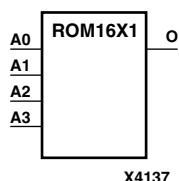
```
component ROCBUF
  port( I : in  std_ulogic;
        O : out std_ulogic);
end component;
```


ROM16X1

16-Deep by 1-Wide ROM

Architectures Supported

ROM16X1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ROM16X1 is a 16-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 4-bit address (A3 – A0). The ROM is initialized to a known value during configuration with the `INIT=value` parameter. The value consists of four hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H. For example, the `INIT=10A7` parameter produces the data stream:

```
0001 0000 1010 0111
```

An error occurs if the `INIT=value` is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM16X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- ROM16X1: 16 x 1 Asynchronous Distributed => LUT ROM
-- The current version of the Xilinx HDL Libraries Guide

ROM16X1_inst : ROM16X1
-- Edit the following generic to define the contents of the ROM.
generic map (
    INIT => X"0000")
port map (
    O => O,    -- ROM output
    A0 => A0,  -- ROM address[0]
    A1 => A1,  -- ROM address[1]
    A2 => A2,  -- ROM address[2]
    A3 => A3  -- ROM address[3]
);

-- End of ROM16X1_inst instantiation
```

Verilog Instantiation Template

```
ROM16X1 ROM16X1_inst (  
    .O(O), // ROM output  
    .A0(A0), // ROM address[0]  
    .A1(A1), // ROM address[1]  
    .A2(A2), // ROM address[2]  
    .A3(A3) // ROM address[3]  
);  
  
// Edit the following defparam to define the contents of the ROM.  
// If the instance name to the ROM is changed, that change needs to  
// be reflected in the defparam statements.  
  
    defparam ROM16X1_inst.INIT = 16'h0000;  
  
// End of ROM16X1_inst instantiation
```

Commonly Used Constraints

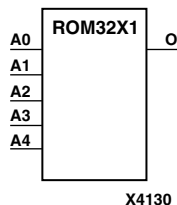
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKNM

ROM32X1

32-Deep by 1-Wide ROM

Architectures Supported

ROM32X1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ROM32X1 is a 32-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 5-bit address (A4 – A0). The ROM is initialized to a known value during configuration with the `INIT=value` parameter. The value consists of eight hexadecimal digits that are written into the ROM from the most-significant digit A=1FH to the least-significant digit A=00H. For example, the `INIT=10A78F39` parameter produces the data stream:

```
0001 0000 1010 0111 1000 1111 0011 1001
```

An error occurs if the `INIT=value` is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM32X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- ROM32X1: 32 x 1 Asynchronous Distributed => LUT ROM
-- The current version of the Xilinx HDL Libraries Guide

ROM32X1_inst : ROM32X1
-- Edit the following generic to define the contents of the ROM.
generic map (
    INIT => X"00000000")
port map (
    O => O,    -- ROM output
    A0 => A0,  -- ROM address[0]
    A1 => A1,  -- ROM address[1]
    A2 => A2,  -- ROM address[2]
    A3 => A3,  -- ROM address[3]
    A4 => A4   -- ROM address[4]
);
-- End of ROM32X1_inst instantiation
```

Verilog Instantiation Template

```
ROM32X1 ROM32X1_inst (  
    .O(O), // ROM output  
    .A0(A0), // ROM address[0]  
    .A1(A1), // ROM address[1]  
    .A2(A2), // ROM address[2]  
    .A3(A3), // ROM address[3]  
    .A4(A4) // ROM address[4]  
);  
// Edit the following defparam to define the contents of the ROM.  
// If the instance name to the ROM is changed, that change needs to  
// be reflected in the defparam statements.  
  
    defparam ROM32X1_inst.INIT = 32'h00000000;  
  
// End of ROM32X1_inst instantiation
```

Commonly Used Constraints

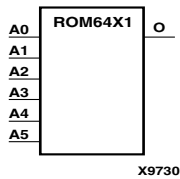
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKNM

ROM64X1

64-Deep by 1-Wide ROM

Architectures Supported

ROM64X1	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ROM64X1 is a 64-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 6-bit address (A5 – A0). The ROM is initialized to a known value during configuration with the `INIT=value` parameter. The value consists of 16 hexadecimal digits that are written into the ROM from the most-significant digit A=FH to the least-significant digit A=0H.

An error occurs if the `INIT=value` is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM64X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- ROM64X1: 64 x 1 Asynchronous Distributed => LUT ROM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

ROM64X1_inst : ROM64X1
-- Edit the following generic to define the contents of the ROM.
generic map (
  INIT => X"0000000000000000")
port map (
  O => O,    -- ROM output
  A0 => A0,  -- ROM address[0]
  A1 => A1,  -- ROM address[1]
  A2 => A2,  -- ROM address[2]
  A3 => A3,  -- ROM address[3]
  A4 => A4,  -- ROM address[4]
  A5 => A5   -- ROM address[5]
);

-- End of ROM64X1_inst instantiation
```

Verilog Instantiation Template

```
ROM64X1 ROM64X1_inst (  
    .O(O), // ROM output  
    .A0(A0), // ROM address[0]  
    .A1(A1), // ROM address[1]  
    .A2(A2), // ROM address[2]  
    .A3(A3), // ROM address[3]  
    .A4(A4), // ROM address[4]  
    .A5(A5) // ROM address[5]  
);  
  
// Edit the following defparam to define the contents of the ROM.  
// If the instance name to the ROM is changed, that change needs to  
// be reflected in the defparam statements.  
  
    defparam ROM64X1_inst.INIT = 64'h0000000000000000;  
  
// End of ROM64X1_inst instantiation
```

Commonly Used Constraints

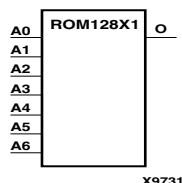
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKNM

ROM128X1

128-Deep by 1-Wide ROM

Architectures Supported

ROM128X1	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



ROM128X1 is a 128-word by 1-bit read-only memory. The data output (O) reflects the word selected by the 7-bit address (A6 – A0). The ROM is initialized to a known value during configuration with the `INIT=value` parameter. The value consists of 32 hexadecimal digits that are written into the ROM from the most-significant digit A=0H to the least-significant digit A=0H.

An error occurs if the `INIT=value` is not specified. See the appropriate CAE tool interface user guide for details.

Usage

For HDL, the ROM128X1 design element should be instantiated rather than inferred.

VHDL Instantiation Template

```
-- ROM128X1: 128 x 1 Asynchronous Distributed => LUT ROM
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

ROM128X1_inst : ROM128X1
-- Edit the following generic to define the contents of the ROM.
generic map (
  INIT => X"00000000000000000000000000000000"
port map (
  O => O,    -- ROM output
  A0 => A0,  -- ROM address[0]
  A1 => A1,  -- ROM address[1]
  A2 => A2,  -- ROM address[2]
  A3 => A3,  -- ROM address[3]
  A4 => A4,  -- ROM address[4]
  A5 => A5,  -- ROM address[5]
  A6 => A6   -- ROM address[6]
);

-- End of ROM128X1_inst instantiation
```

Verilog Instantiation Template

```
ROM128X1 ROM128X1_inst (  
    .O(O), // ROM output  
    .A0(A0), // ROM address[0]  
    .A1(A1), // ROM address[1]  
    .A2(A2), // ROM address[2]  
    .A3(A3), // ROM address[3]  
    .A4(A4), // ROM address[4]  
    .A5(A5), // ROM address[5]  
    .A6(A6) // ROM address[6]  
);  
  
// Edit the following defparam to define the contents of the ROM.  
// If the instance name to the ROM is changed, that change needs to  
// be reflected in the defparam statements.  
  
defparam ROM128X1_inst.INIT =  
    128'h00000000000000000000000000000000;  
  
// End of ROM128X1_inst instantiation
```

Commonly Used Constraints

BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKNM

Verilog Instantiation Template

```
ROM256X1 ROM256X1_inst (  
    .O(O), // ROM output  
    .A0(A0), // ROM address[0]  
    .A1(A1), // ROM address[1]  
    .A2(A2), // ROM address[2]  
    .A3(A3), // ROM address[3]  
    .A4(A4), // ROM address[4]  
    .A5(A5), // ROM address[5]  
    .A6(A6) // ROM address[6]  
    .A7(A7) // ROM address[7]  
);  
  
// Edit the following defparam to define the contents of the ROM.  
// If the instance name to the ROM is changed, that change needs to  
// be reflected in the defparam statements.  
  
defparam ROM256X1_inst.INIT =  
    256'h0000000000000000000000000000000000000000000000000000000000000000;  
  
// End of ROM256X1_inst instantiation
```

Commonly Used Constraints

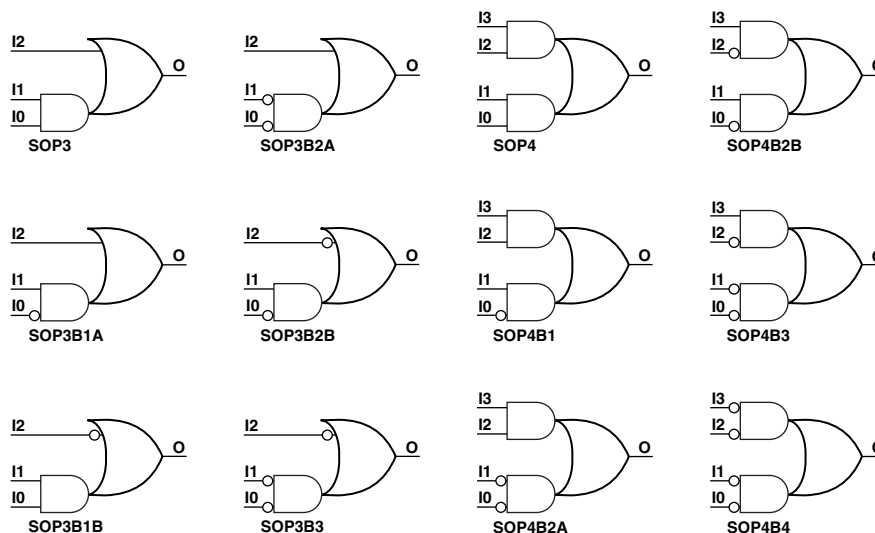
BLKNM, HBLKNM, HU_SET, INIT, LOC, RLOC, ROM_EXTRACT, U_SET, XBLKNM

SOP3-4

Sum of Products

Architectures Supported

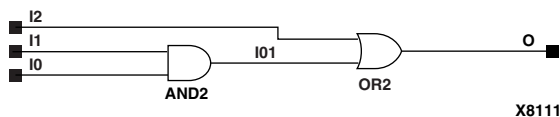
SOP3, SOP3B1A, SOP3B1B, SOP3B2A, SOP3B2B, SOP3B3 SOP4, SOP4B1, SOP4B2A, SOP4B2B, SOP4B3, SOP4B4	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



X9421

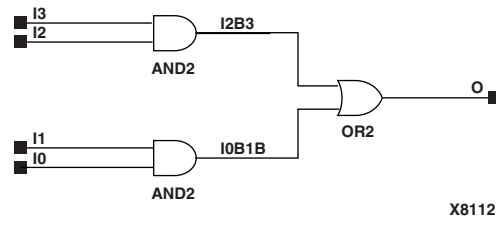
SOP Gate Representations

Sum Of Products (SOP) macros provide common logic functions by OR gating the outputs of two AND functions or the output of one AND function with one direct input. Variations of inverting and non-inverting inputs are available.

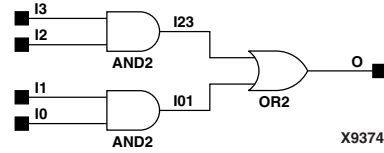


X8111

SOP3 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



SOP4 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Virtex, Virtex-E



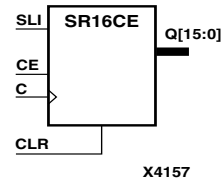
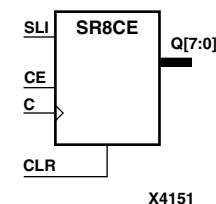
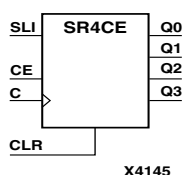
SOP4 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

SR4CE, SR8CE, SR16CE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SR4CE, SR8CE, SR16CE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4CE, SR8CE, and SR16CE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel outputs (Q), and clock enable (CE) and asynchronous clear (CLR) inputs. The CLR input, when High, overrides all other inputs and resets the data outputs (Q) Low. When CE is High and CLR is Low, the data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and CLR is Low, data shifts to the next highest bit position as new data is loaded into Q0 (SLI-Q0, Q0-Q1, Q1-Q2, etc.). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4CE, Q7 for SR8CE, or Q15 for SR16CE) of one stage to the SLI input of the next stage and connecting clock, CE, and CLR in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

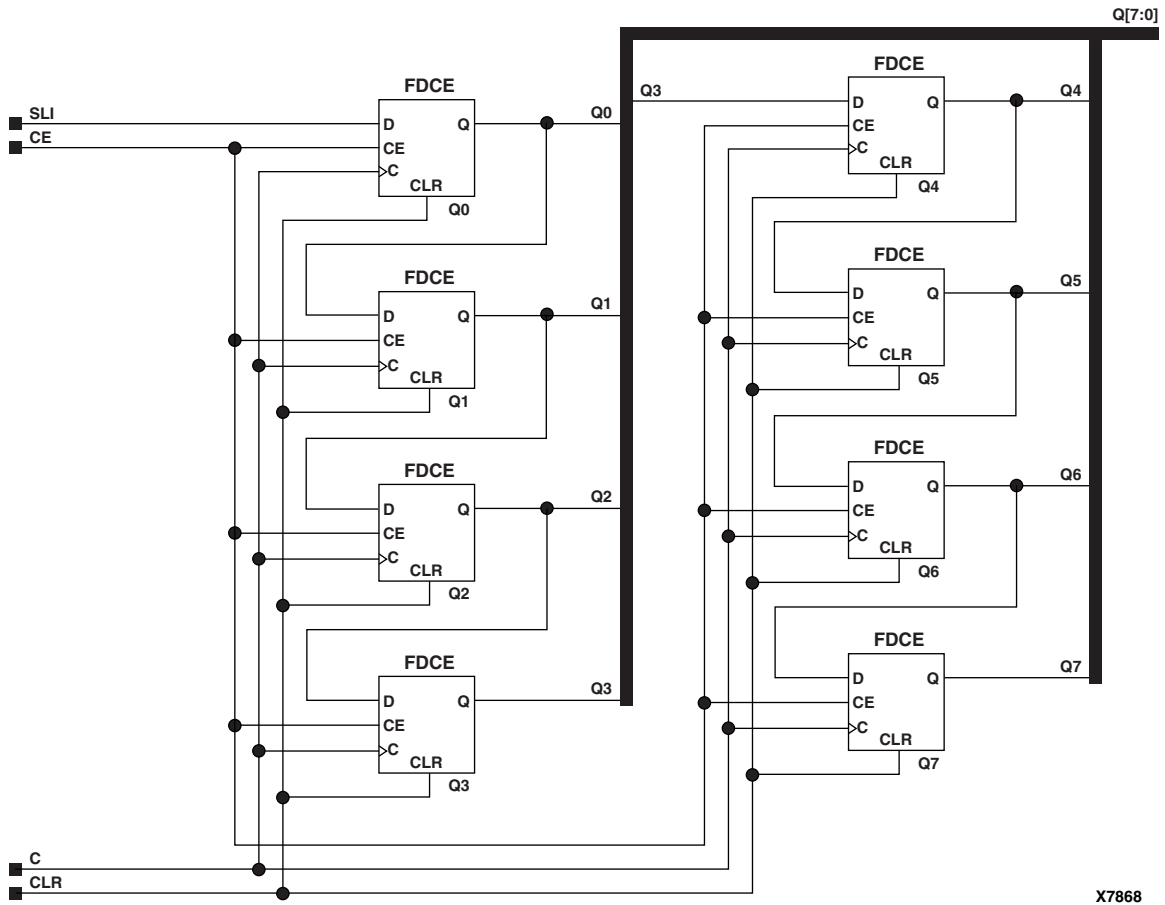
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz - Q1
1	X	X	X	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4CE; z = 7 for SR8CE; z = 15 for SR16CE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8CE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

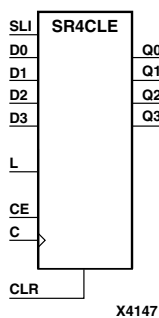
For HDL, these design elements are inferred rather than instantiated.

SR4CLE, SR8CLE, SR16CLE

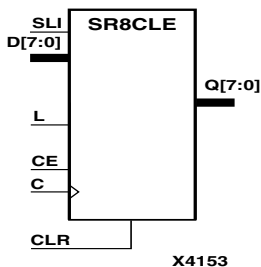
4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SR4CLE, SR8CLE, SR16CLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4CLE, SR8CLE, and SR16CLE are 4-, 8-, and 16-bit shift registers, respectively, with a shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and asynchronous clear (CLR). The register ignores clock transitions when L and CE are Low. The asynchronous CLR, when High, overrides all other inputs and resets the data outputs (Q) Low. When L is High and CLR is Low, data on the Dn – D0 inputs is loaded into the corresponding Qn – Q0 bits of the register. When CE is High and L and CLR are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and CLR are Low, the data shifts to the next highest bit position as new data is loaded into Q0 (SLI–Q0, Q0–Q1, Q1–Q2, etc.).



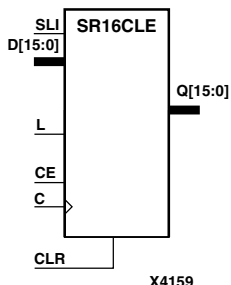
Registers can be cascaded by connecting the last Q output (Q3 for SR4CLE, Q7 for SR8CLE, or Q15 for SR16CLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and CLR inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

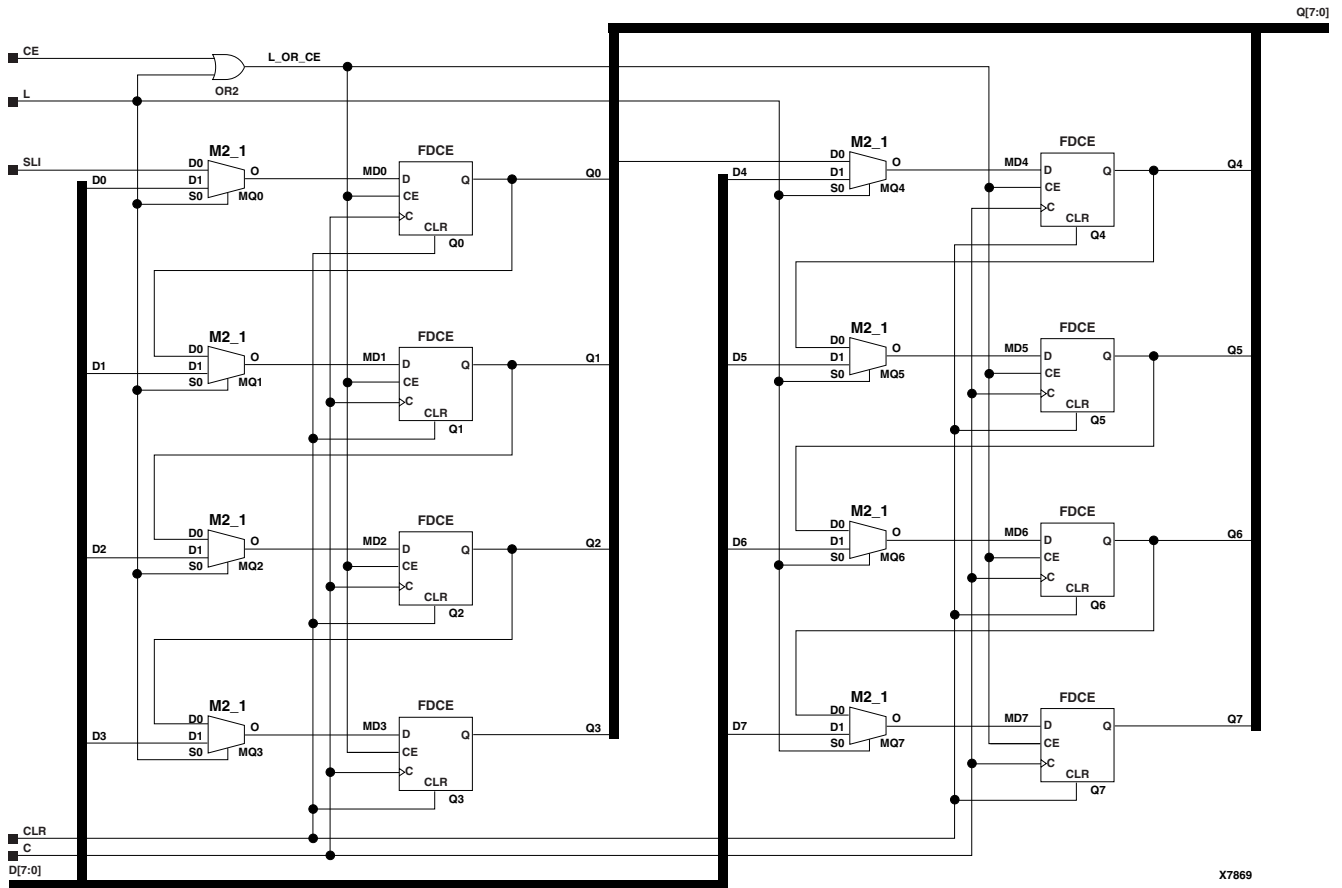


Inputs						Outputs	
CLR	L	CE	SLI	Dn – D0	C	Q0	Qz – Q1
1	X	X	X	X	X	0	0
0	1	X	X	Dn – D0	↑	D0	Dn
0	0	1	SLI	X	↑	SLI	qn-1

Inputs						Outputs	
CLR	L	CE	SLI	Dn – D0	C	Q0	Qz – Q1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4CLE; z = 7 for SR8CLE; z = 15 for SR16CLE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8CLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

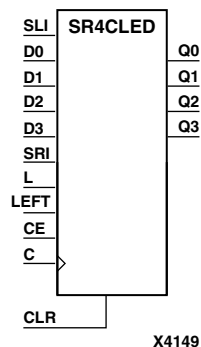
For HDL, this design element is inferred rather than instantiated.

SR4CLED, SR8CLED, SR16CLED

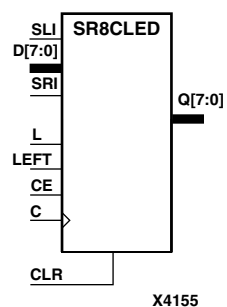
4-, 8-, 16-Bit Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SR4CLED, SR8CLED, SR16CLED	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4CLED, SR8CLED, and SR16CLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs: clock enable (CE), load enable (L), shift left/right (LEFT), and asynchronous clear (CLR). The register ignores clock transitions when CE and L are Low. The asynchronous clear, when High, overrides all other inputs and resets the data outputs (Qn) Low. When L is High and CLR is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and CLR are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on the SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, etc.) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4CLED, Q7 for SR8CLED, or Q15 for SR16CLED) during the Low-to-High clock transition and shifted right (to Q2, Q1, ... for SR4CLED; to Q6, Q5, ... for SR8CLED; and to Q14, Q13, ... for SR16CLED) during subsequent clock transitions. The truth tables for SR4CLED, SR8CLED, and SR16CLED indicate the state of the Q outputs under all input conditions for SR4CLED, SR8CLED, and SR16CLED.

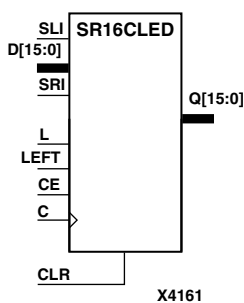


The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.



SR4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D3– D0	↑	D0	D3	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg

SR4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 and qn+1 = state of referenced output one setup time prior to active clock transition

SR8CLED Truth Table

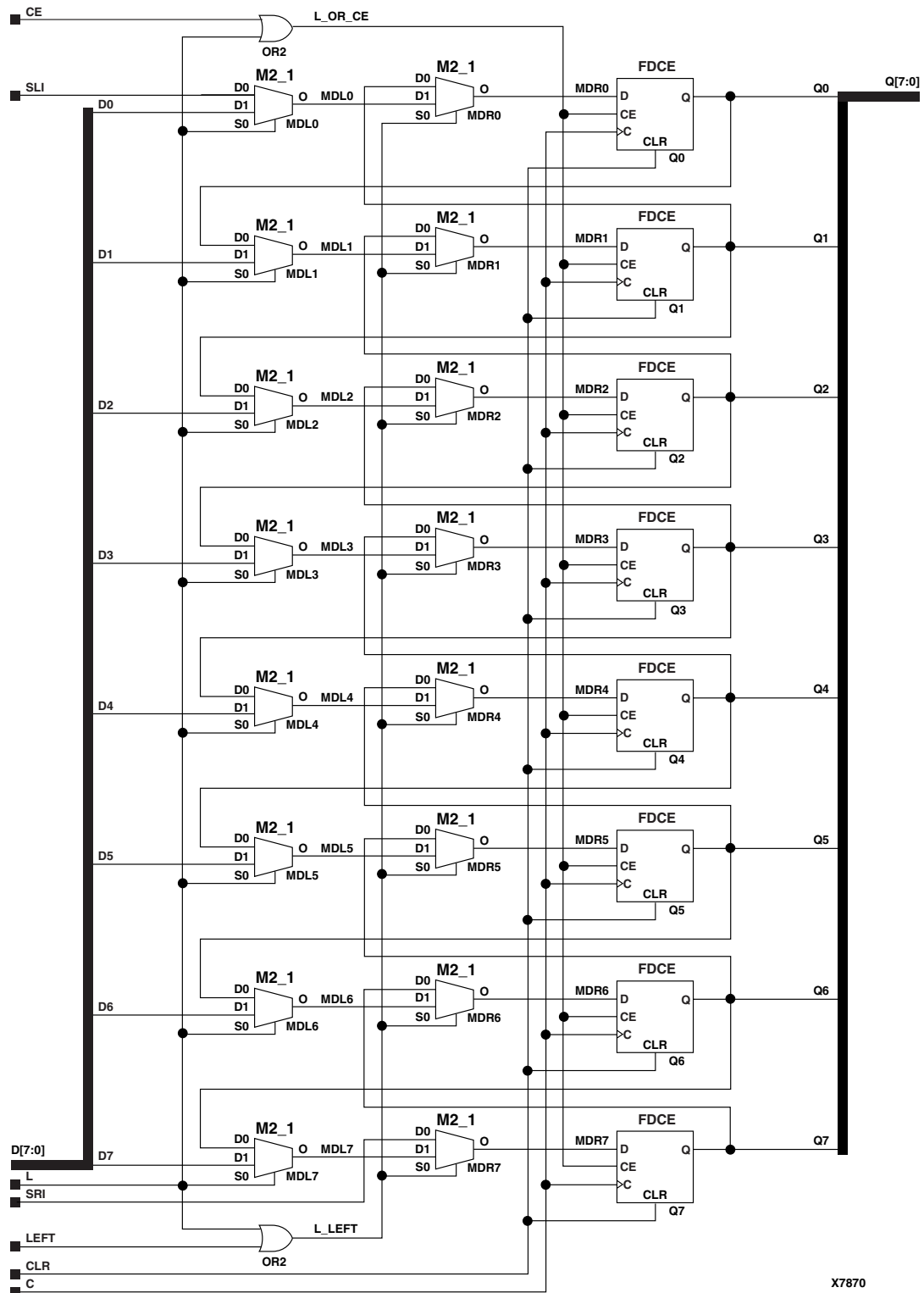
Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D7 – D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D7 – D0	↑	D0	D7	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR16CLED Truth Table

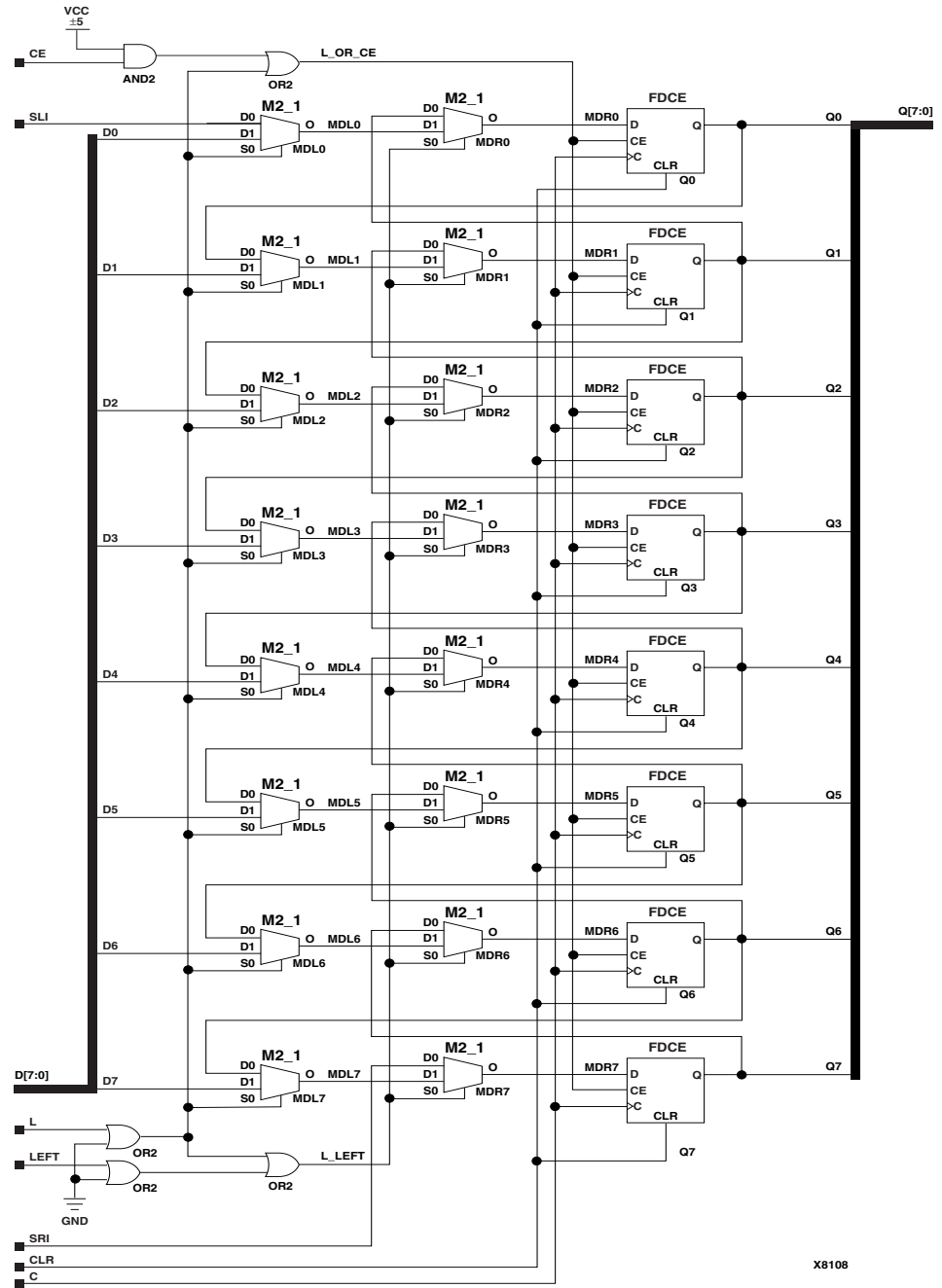
Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D15 – D0	↑	D0	D15	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



X7870

SR8CLED Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



X8108

SR8CLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

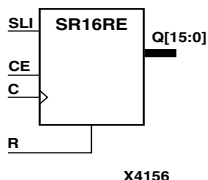
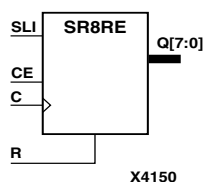
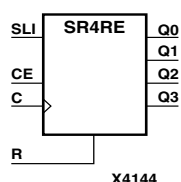
For HDL, this design element is inferred rather than instantiated.

SR4RE, SR8RE, SR16RE

4-, 8-, 16-Bit Serial-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SR4RE, SR8RE, SR16RE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4RE, SR8RE, and SR16RE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel outputs (Qn), clock enable (CE), and synchronous reset (R) inputs. The R input, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent Low-to-High clock transitions, when CE is High and R is Low, data shifts to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, etc.). The register ignores clock transitions when CE is Low.

Registers can be cascaded by connecting the last Q output (Q3 for SR4RE, Q7 for SR8RE, or Q15 for SR16RE) of one stage to the SLI input of the next stage and connecting clock, CE, and R in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

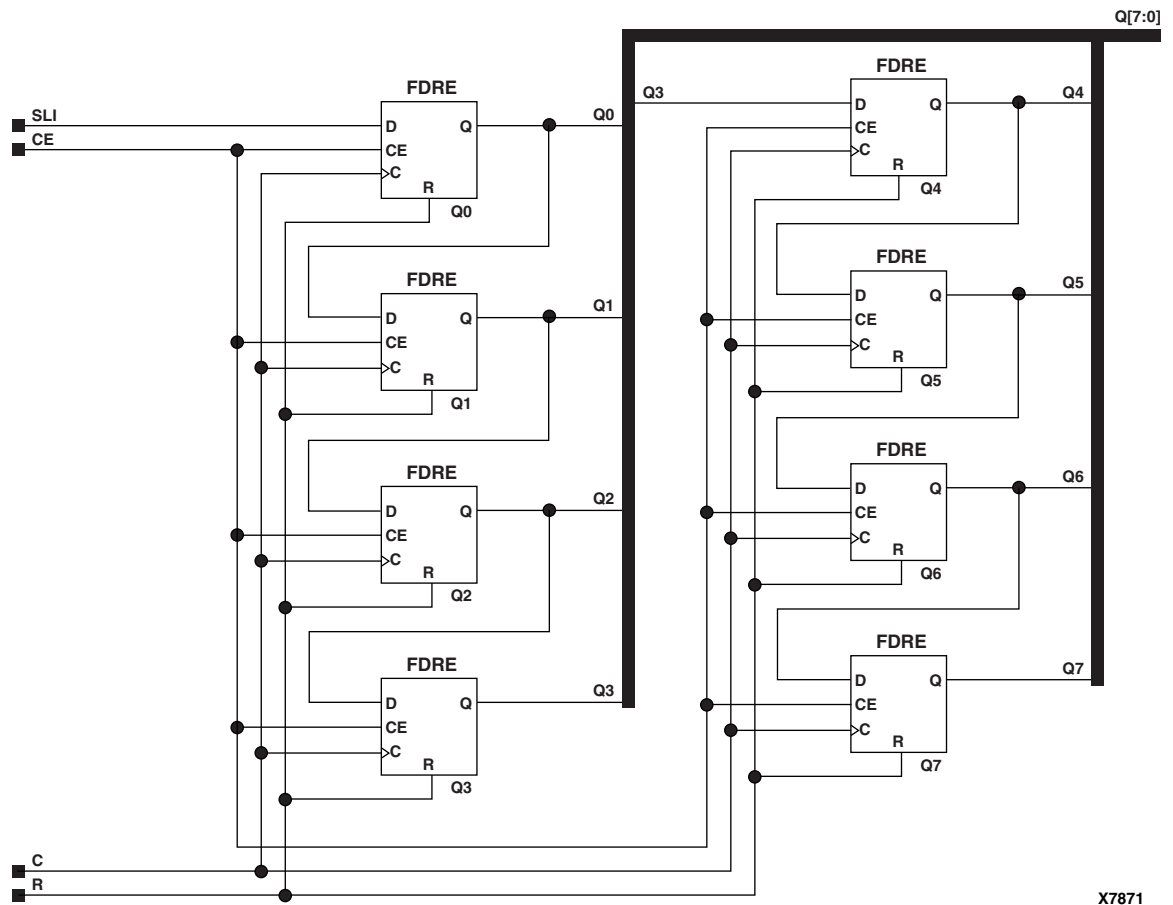
Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs				Outputs	
R	CE	SLI	C	Q0	Qz – Q1
1	X	X	↑	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	0	↑	0	qn-1

z = 3 for SR4RE; z = 7 for SR8RE; z = 15 for SR16RE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8RE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

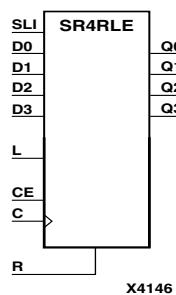
For HDL, this design element is inferred rather than instantiated.

SR4RLE, SR8RLE, SR16RLE

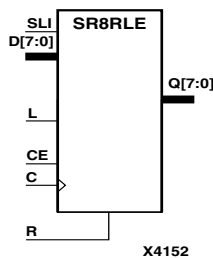
4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SR4RLE, SR8RLE, SR16RLE	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4RLE, SR8RLE, and SR16RLE are 4-, 8-, and 16-bit shift registers, respectively, with shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and synchronous reset (R). The register ignores clock transitions when L and CE are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low during the Low-to-High clock transition, data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and R are Low, the data shifts to the next highest bit position as new data is loaded into Q0 (SLI-Q0, Q0-Q1, Q1-Q2, etc.).



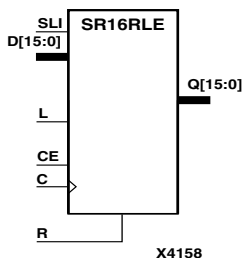
Registers can be cascaded by connecting the last Q output (Q3 for SR4RLE, Q7 for SR8RLE, or 15 for SR16RLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and R inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

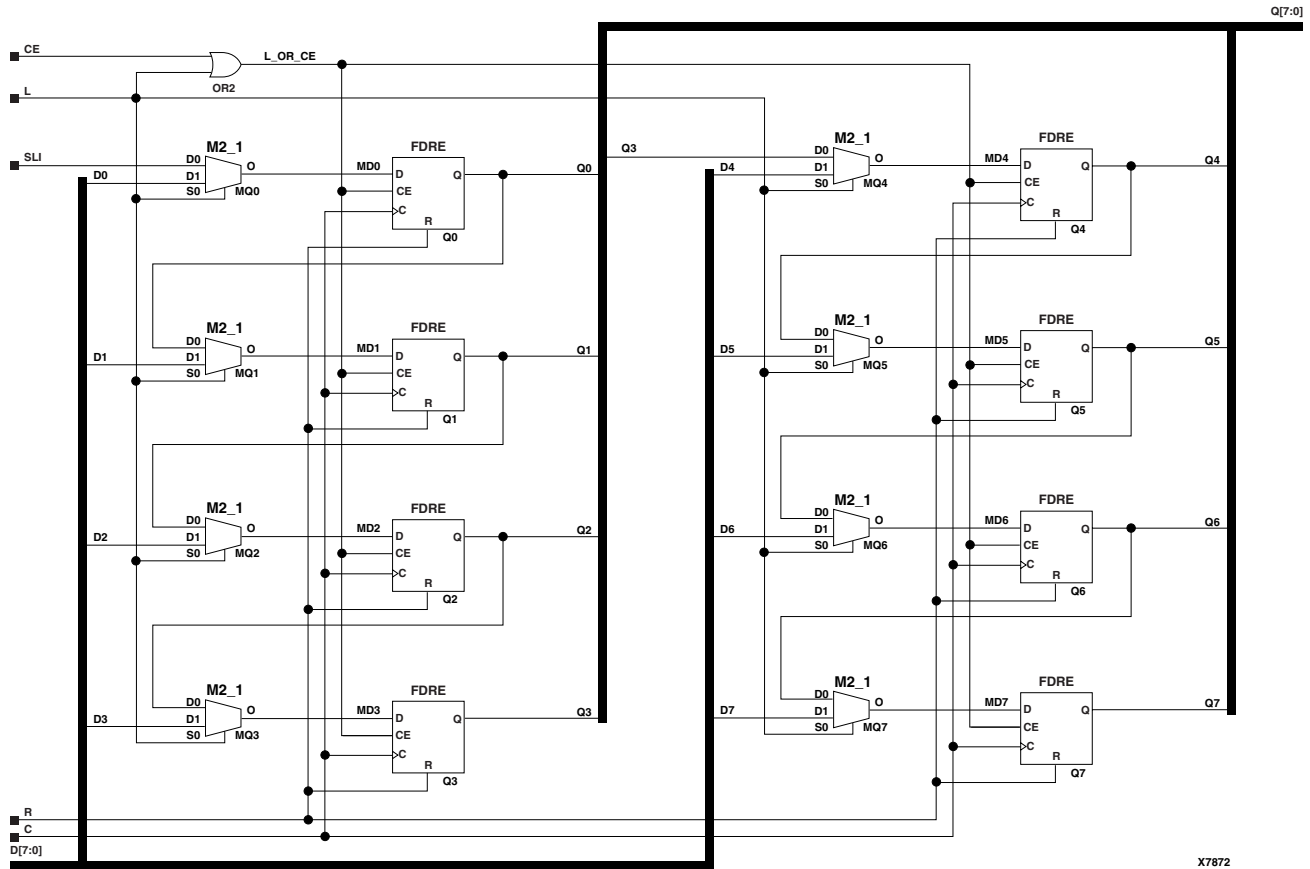


Inputs						Outputs	
R	L	CE	SLI	Dz - D0	C	Q0	Qz - Q1
1	X	X	X	X	↑	0	0
0	1	X	X	Dz - D0	↑	D0	Dn
0	0	1	SLI	X	↑	SLI	qn-1

Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SR4RLE; z = 7 for SR8RLE; z = 15 for SR16RLE

qn-1 = state of referenced output one setup time prior to active clock transition



SR8RLE Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

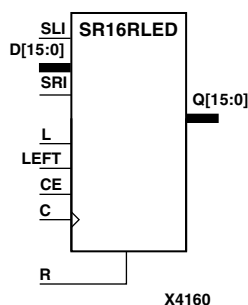
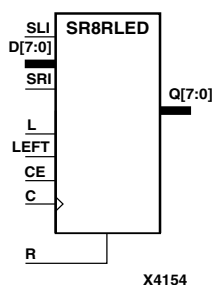
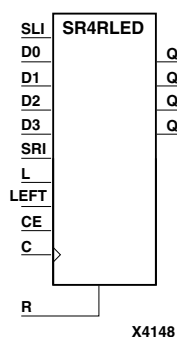
For HDL, these design elements are inferred rather than instantiated.

SR4RLED, SR8RLED, SR16RLED

4-, 8-, 16-Bit Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SR4RLED, SR8RLED, SR16RLED	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



SR4RLED, SR8RLED, and SR16RLED are 4-, 8-, and 16-bit shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q) and four control inputs — clock enable (CE), load enable (L), shift left/right (LEFT), and synchronous reset (R). The register ignores clock transitions when CE and L are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low during the Low-to-High clock transition, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on SLI is loaded into Q0 during the Low-to-High clock transition and shifted left (to Q1, Q2, etc.) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q3 for SR4RLED, Q7 for SR8RLED, or Q15 for SR16RLED) during the Low-to-High clock transition and shifted right (to Q2, Q1,... for SR4RLED; to Q6, Q5,... for SR8RLED; or to Q14, Q13,... for SR16RLED) during subsequent clock transitions. The truth table indicates the state of the Q outputs under all input conditions.

The register is asynchronously cleared, outputs Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

SR4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D3 – D0	↑	D0	D3	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg

SR4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR8RLED Truth Table

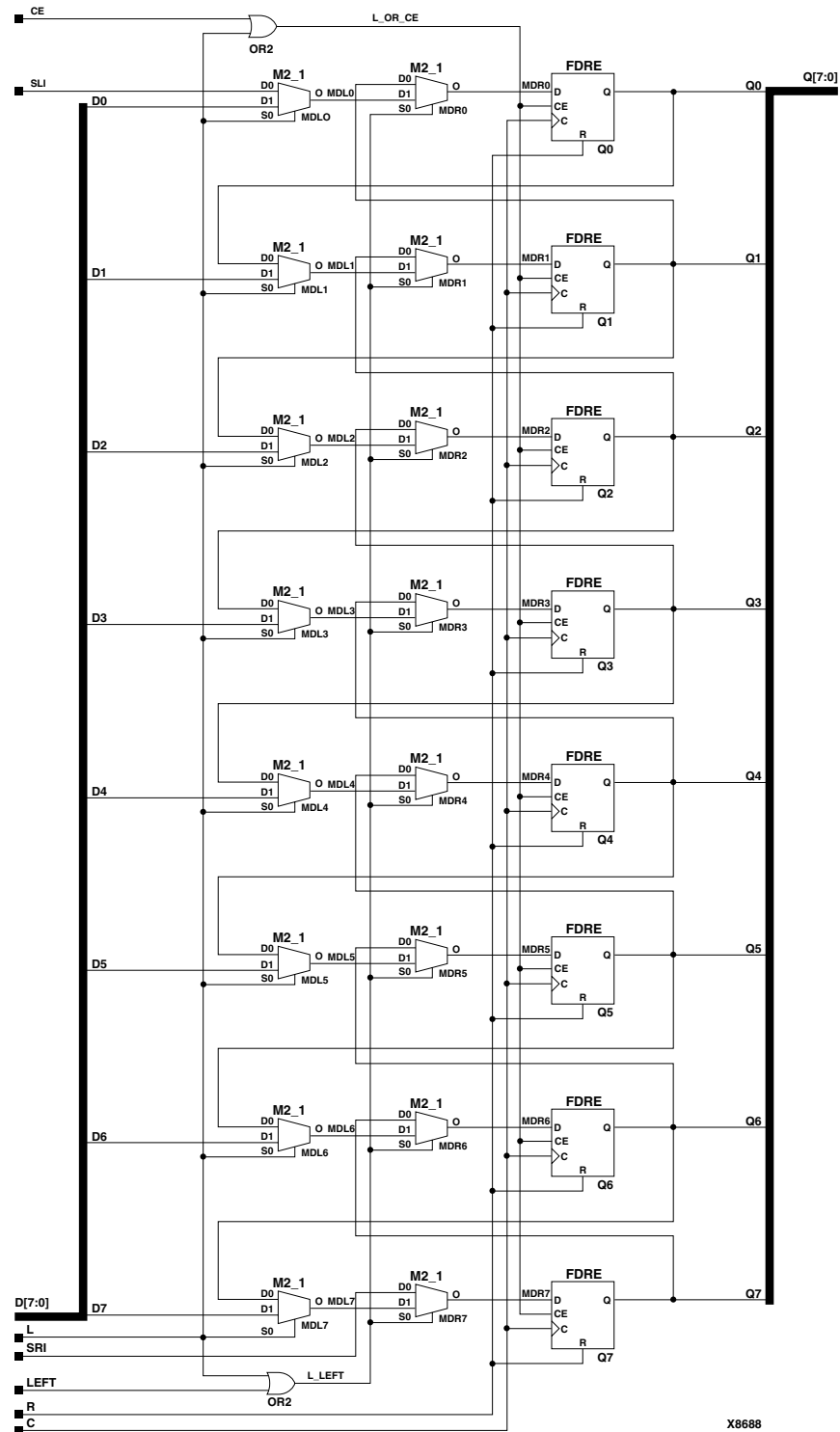
Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D7 – D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D7 – D0	↑	D0	D7	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SR16RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	↑	0	0	0
0	1	X	X	X	X	D15 – D0	↑	D0	D15	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



SR8RLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II, Spartan-II, Spartan-IIe, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

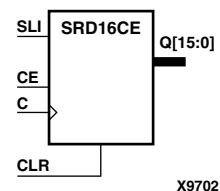
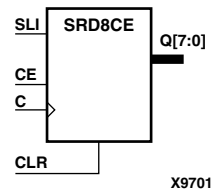
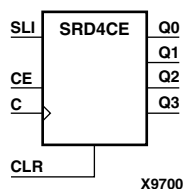
For HDL, these design elements are inferred rather than instantiated.

SRD4CE, SRD8CE, SRD16CE

4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SRD4CE, SRD8CE, SRD16CE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



SRD4CE, SRD8CE, and SRD16CE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with a shift-left serial input (SLI), parallel outputs (Q), clock enable (CE) and asynchronous clear (CLR) inputs. The CLR input, when High, overrides all other inputs and resets the data outputs (Q) Low. When CE is High and CLR is Low, the data on the SLI input is loaded into the first bit of the shift register during the Low-to-High (or High-to-Low) clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and CLR is Low, data shifts to the next highest bit position as new data is loaded into Q0 (SLI–Q0, Q0–Q1, Q1–Q2, etc.). The register ignores clock transitions when CE is Low.

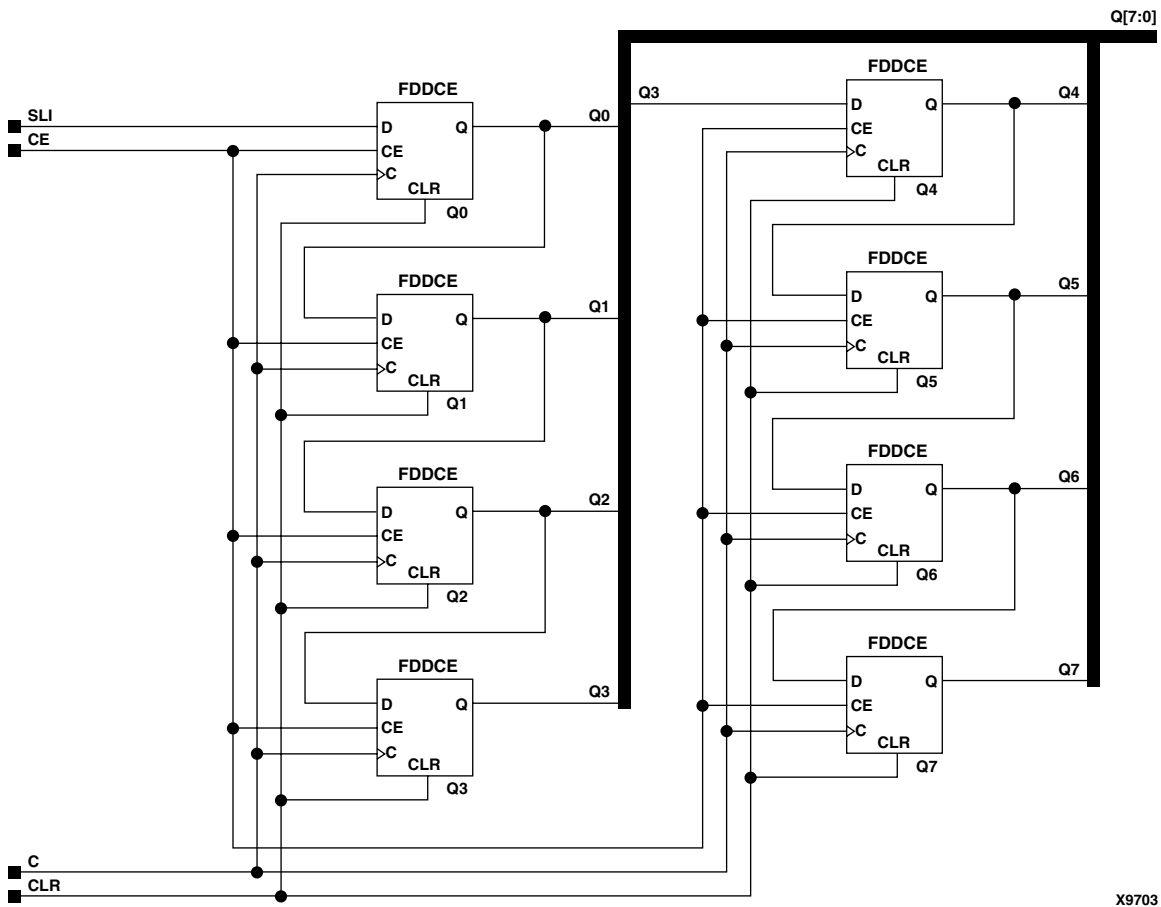
Registers can be cascaded by connecting the last Q output (Q3 for SRD4CE, Q7 for SRD8CE, or Q15 for SRD16CE) of one stage to the SLI input of the next stage and connecting clock, CE, and CLR in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz – Q1
1	X	X	X	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	1	↓	1	qn-1
0	1	0	↑	0	qn-1
0	1	0	↓	0	qn-1

z = 3 for SRD4CE; z = 7 for SRD8CE; z = 15 for SRD16CE

qn-1 = state of referenced output one setup time prior to active clock transition



X9703

SRD8CE Implementation CoolRunner-II

Usage

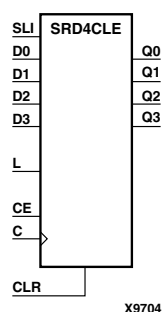
For HDL, these design elements are inferred rather than instantiated.

SRD4CLE, SRD8CLE, SRD16CLE

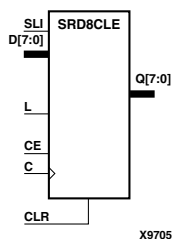
4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SRD4CLE, SRD8CLE, SRD16CLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive

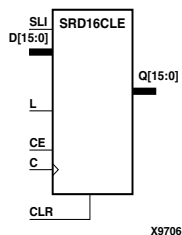


SRD4CLE, SRD8CLE, and SRD16CLE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with a shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and asynchronous clear (CLR). The register ignores clock transitions when L and CE are Low. The asynchronous CLR, when High, overrides all other inputs and resets the data outputs (Q) Low. When L is High and CLR is Low, data on the D_n – D₀ inputs is loaded into the corresponding Q_n – Q₀ bits of the register. When CE is High and L and CLR are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High (or High-to-Low) clock (C) transition and appears on the Q₀ output. During subsequent clock transitions, when CE is High and L and CLR are Low, the data shifts to the next highest bit position as new data is loaded into Q₀ (SLI → Q₀, Q₀ → Q₁, Q₁ → Q₂, etc.).



Registers can be cascaded by connecting the last Q output (Q₃ for SRD4CLE, Q₇ for SRD8CLE, or Q₁₅ for SRD16CLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and CLR inputs in parallel.

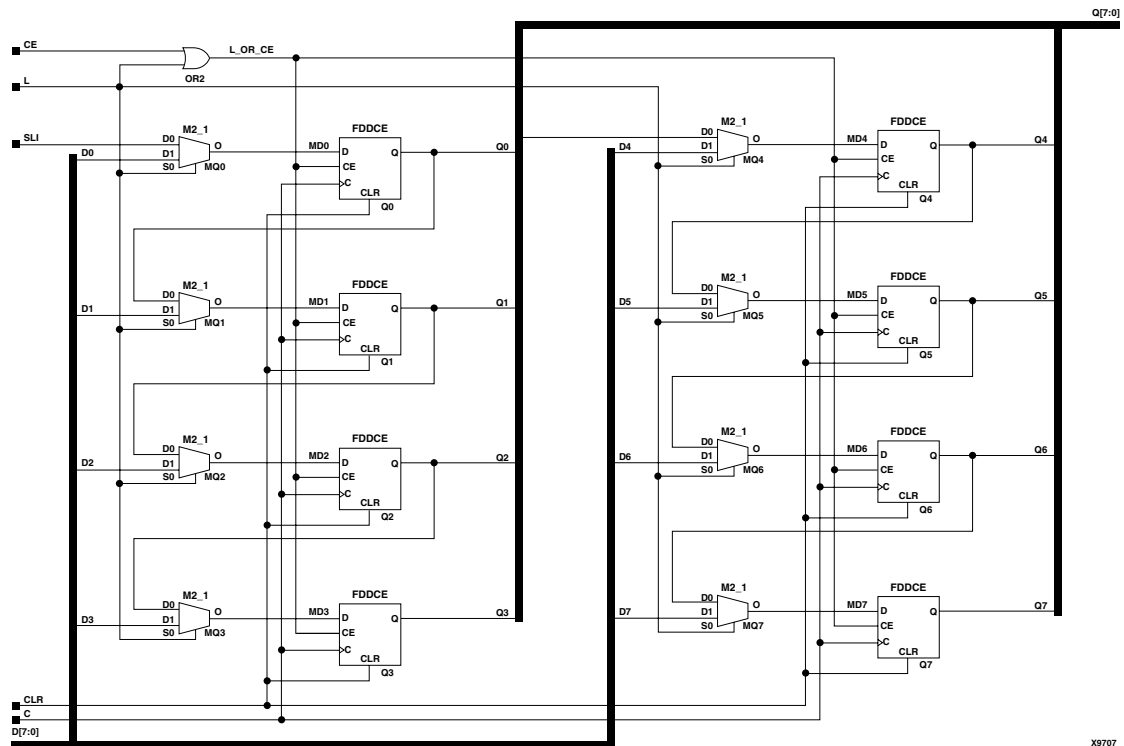
The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.



Inputs						Outputs	
CLR	L	CE	SLI	D _n – D ₀	C	Q ₀	Q _z – Q ₁
1	X	X	X	X	X	0	0
0	1	X	X	D _n – D ₀	↑	D ₀	D _n
0	1	X	X	D _n – D ₀	↓	D ₀	D _n
0	0	1	SLI	X	↑	SLI	qn-1
0	0	1	SLI	X	↓	SLI	qn-1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SRD4CLE; z = 7 for SRD8CLE; z = 15 for SRD16CLE

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8CLE Implementation CoolRunner-II

Usage

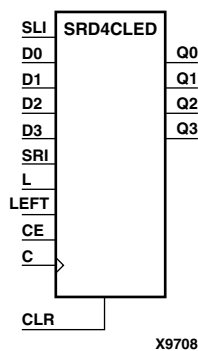
For HDL, these design elements are inferred rather than instantiated.

SRD4CLED, SRD8CLED, SRD16CLED

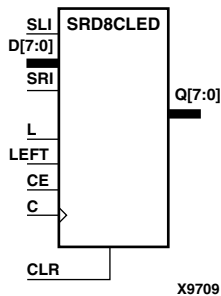
4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Asynchronous Clear

Architectures Supported

SRD4CLED, SRD8CLED, SRD16CLED	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



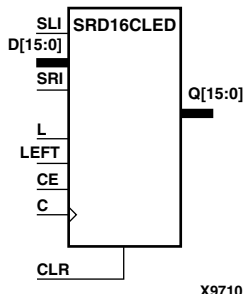
SRD4CLED, SRD8CLED, and SRD16CLED are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left (SLI) and shift-right (SRI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs: clock enable (CE), load enable (L), shift left/right (LEFT), and asynchronous clear (CLR). The register ignores clock transitions when CE and L are Low. The asynchronous clear, when High, overrides all other inputs and resets the data outputs (Q_n) Low. When L is High and CLR is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and CLR are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on the SLI is loaded into Q₀ during the Low-to-High or High-to-Low clock transition and shifted left (to Q₁, Q₂, etc.) during subsequent clock transitions. If LEFT is Low, data on the SRI is loaded into the last Q output (Q₃ for SRD4CLED, Q₇ for SRD8CLED, or Q₁₅ for SRD16CLED) during the Low-to-High or High-to-Low clock transition and shifted right (to Q₂, Q₁,... for SRD4CLED; to Q₆, Q₅,... for SRD8CLED; and to Q₁₄, Q₁₃,... for SRD16CLED) during subsequent clock transitions. The truth tables for SRD4CLED, SRD8CLED, and SRD16CLED indicate the state of the Q outputs under all input conditions for SRD4CLED, SRD8CLED, and SRD16CLED.



The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

SRD4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D3– D0	↑	D0	D3	Dn
0	1	X	X	X	X	D3– D0	↓	D0	D3	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q2	qn-1



SRD4CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D3 – D0	C	Q0	Q3	Q2 – Q1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

qn-1 and qn+1 = state of referenced output one setup time prior to active clock transition

SRD8CLED Truth Table

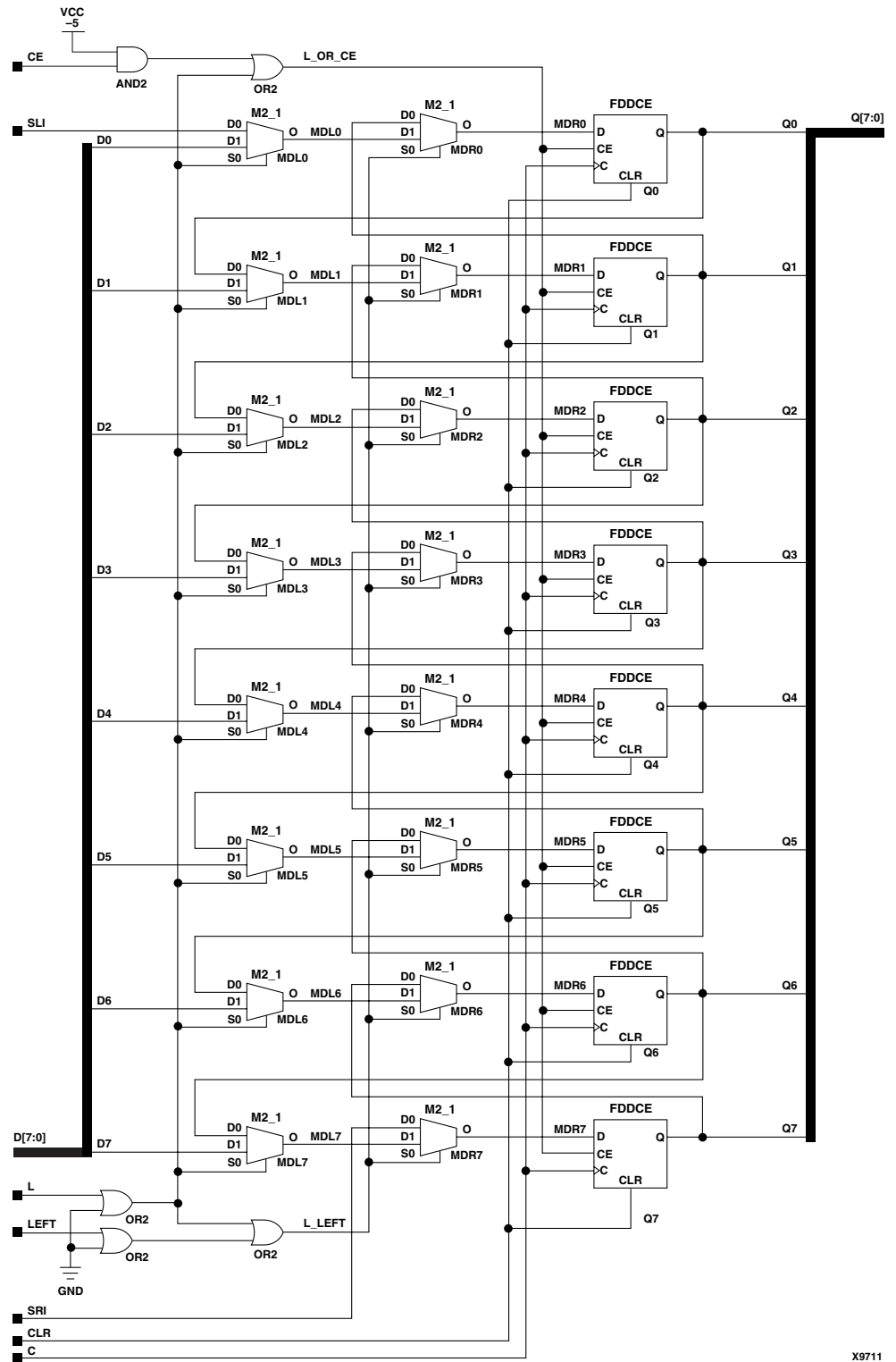
Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D7 – D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D7 – D0	↑	D0	D7	Dn
0	1	X	X	X	X	D7 – D0	↓	D0	D7	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q6	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD16CLED Truth Table

Inputs								Outputs		
CLR	L	CE	LEFT	SLI	SRI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	X	0	0	0
0	1	X	X	X	X	D15 – D0	↑	D0	D15	Dn
0	1	X	X	X	X	D15 – D0	↓	D0	D15	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q14	qn-1
0	0	1	0	X	SRI	X	↑	q1	SRI	qn+1
0	0	1	0	X	SRI	X	↓	q1	SRI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



X9711

SRD8CLED Implementation CoolRunner-II

Usage

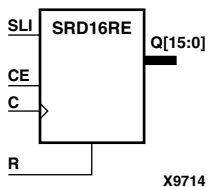
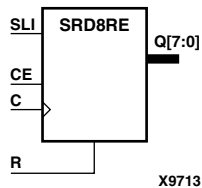
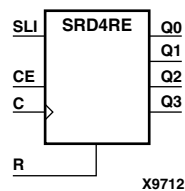
For HDL, these design elements are inferred rather than instantiated.

SRD4RE, SRD8RE, SRD16RE

4-, 8-, 16-Bit Serial-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SRD4RE, SRD8RE, SRD16RE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



SRD4RE, SRD8RE, and SRD16RE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left serial input (SLI), parallel outputs (Qn), clock enable (CE), and synchronous reset (R) inputs. The R input, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When CE is High and R is Low, the data on the SLI is loaded into the first bit of the shift register during the Low-to-High clock or High-to-Low (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and R is Low, data shifts to the next highest bit position as new data is loaded into Q0 (SLI→Q0, Q0→Q1, Q1→Q2, etc.). The register ignores clock transitions when CE is Low.

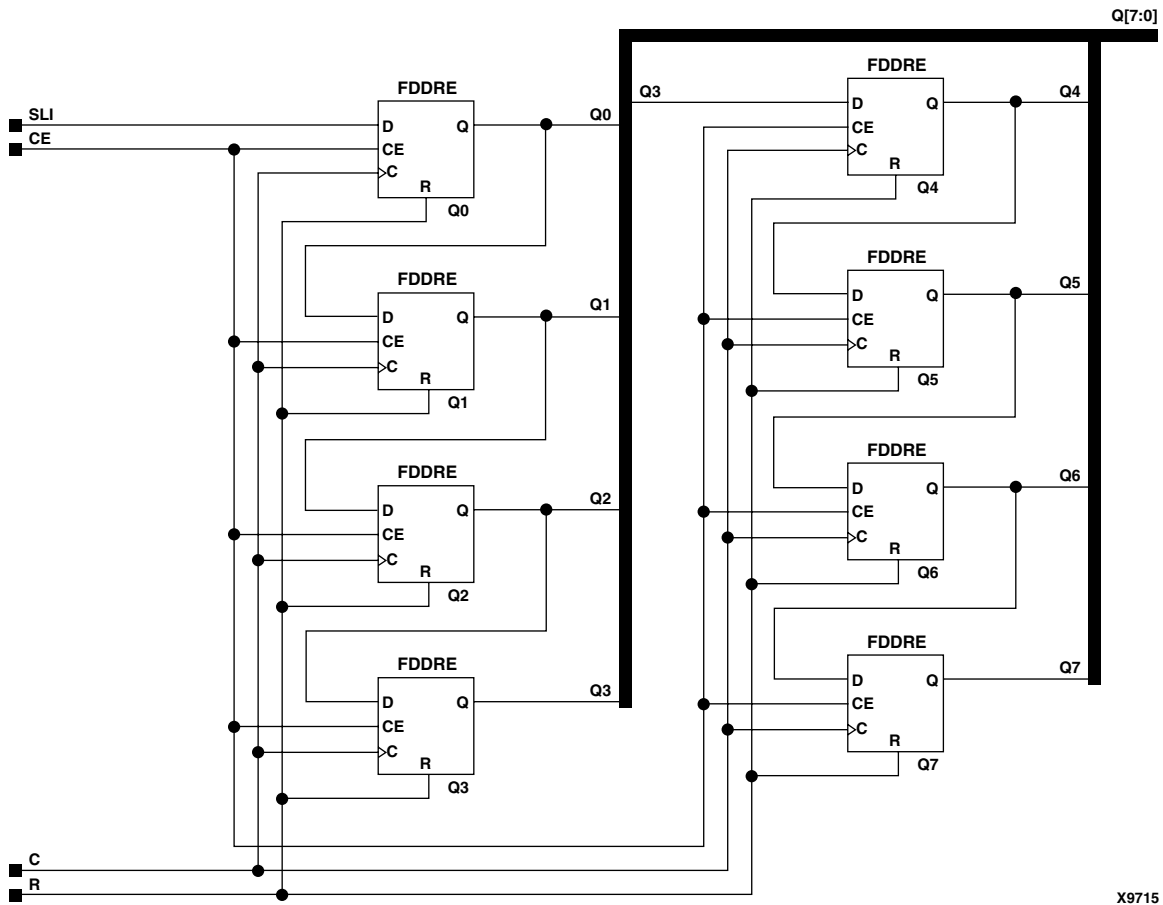
Registers can be cascaded by connecting the last Q output (Q3 for SRD4RE, Q7 for SRD8RE, or Q15 for SRD16RE) of one stage to the SLI input of the next stage and connecting clock, CE, and R in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Inputs				Outputs	
R	CE	SLI	C	Q0	Qz – Q1
1	X	X	↑	0	0
1	X	X	↓	0	0
0	0	X	X	No Chg	No Chg
0	1	1	↑	1	qn-1
0	1	1	↓	1	qn-1
0	1	0	↑	0	qn-1
0	1	0	↓	0	qn-1

z = 3 for SRD4RE; z = 7 for SRD8RE; z = 15 for SRD16RE

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8RE Implementation CoolRunner-II

Usage

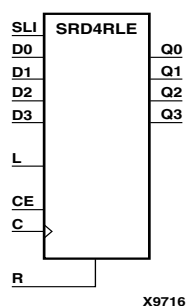
For HDL, these design elements are inferred rather than instantiated.

SRD4RLE, SRD8RLE, SRD16RLE

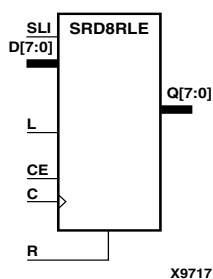
4-, 8-, 16-Bit Loadable Serial/Parallel-In Parallel-Out Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SRD4RLE, SRD8RLE, SRD16RLE	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive

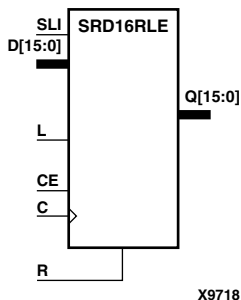


SRD4RLE, SRD8RLE, and SRD16RLE are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left serial input (SLI), parallel inputs (D), parallel outputs (Q), and three control inputs: clock enable (CE), load enable (L), and synchronous reset (R). The register ignores clock transitions when L and CE are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data on the SLI input is loaded into the first bit of the shift register during the Low-to-High or High-to-Low clock (C) transition and appears on the Q0 output. During subsequent clock transitions, when CE is High and L and R are Low, the data shifts to the next highest bit position as new data is loaded into Q0 (SLI–Q0, Q0–Q1, Q1–Q2, etc.).



Registers can be cascaded by connecting the last Q output (Q3 for SRD4RLE, Q7 for SRD8RLE, or 15 for SRD16RLE) of one stage to the SLI input of the next stage and connecting clock, CE, L, and R inputs in parallel.

The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

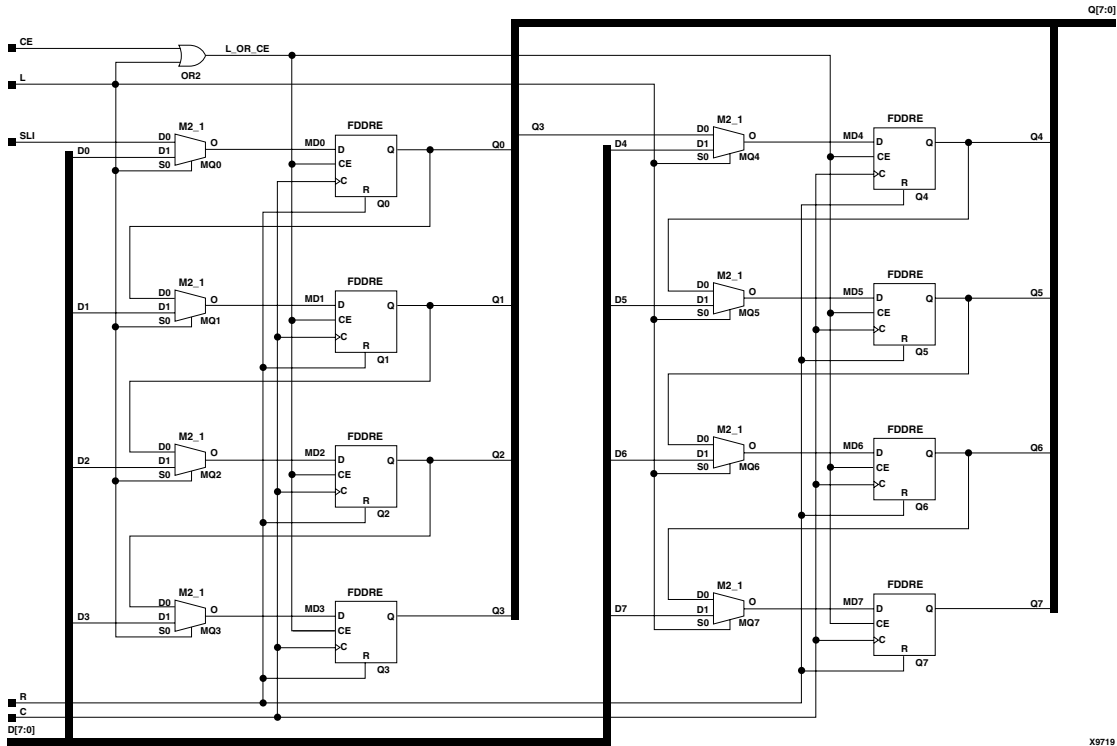


Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
1	X	X	X	X	↑	0	0
1	X	X	X	X	↓	0	0
0	1	X	X	Dz – D0	↑	D0	Dn
0	1	X	X	Dz – D0	↓	D0	Dn
0	0	1	SLI	X	↑	SLI	qn-1
0	0	1	SLI	X	↓	SLI	qn-1

Inputs						Outputs	
R	L	CE	SLI	Dz – D0	C	Q0	Qz – Q1
0	0	0	X	X	X	No Chg	No Chg

z = 3 for SRD4RLE; z = 7 for SRD8RLE; z = 15 for SRD16RLE

qn-1 = state of referenced output one setup time prior to active clock transition



SRD8RLE Implementation CoolRunner-II

Usage

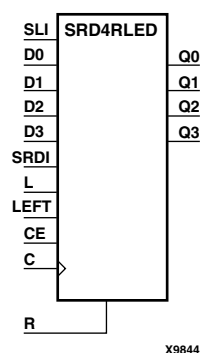
For HDL, these design elements are inferred rather than instantiated.

SRD4RLED, SRD8RLED, SRD16RLED

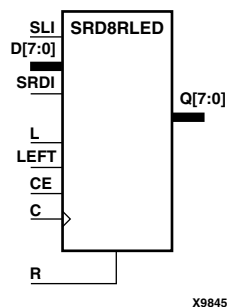
4-, 8-, 16-Bit Dual Edge Triggered Shift Registers with Clock Enable and Synchronous Reset

Architectures Supported

SRD4RLED, SRD8RLED, SRD16RLED	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Primitive



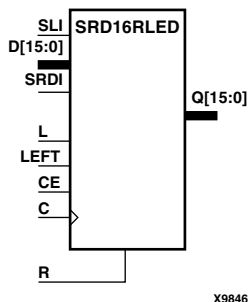
SRD4RLED, SRD8RLED, and SRD16RLED are 4-, 8-, and 16-bit dual edge triggered shift registers, respectively, with shift-left (SLI) and shift-right (SRDI) serial inputs, parallel inputs (D), parallel outputs (Q), and four control inputs — clock enable (CE), load enable (L), shift left/right (LEFT), and synchronous reset (R). The register ignores clock transitions when CE and L are Low. The synchronous R, when High, overrides all other inputs during the Low-to-High or High-to-Low clock (C) transition and resets the data outputs (Q) Low. When L is High and R is Low, the data on the D inputs is loaded into the corresponding Q bits of the register. When CE is High and L and R are Low, data is shifted right or left, depending on the state of the LEFT input. If LEFT is High, data on SLI is loaded into Q0 during the Low-to-High or High-to-Low clock transition and shifted left (to Q1, Q2, etc.) during subsequent clock transitions. If LEFT is Low, data on the SRDI is loaded into the last Q output (Q3 for SRD4RLED, Q7 for SRD8RLED, or Q15 for SRD16RLED) during the Low-to-High or High-to-Low clock transition and shifted right (to Q2, Q1,... for SRD4RLED; to Q6, Q5,... for SRD8RLED; or to Q14, Q13,... for SRD16RLED) during subsequent clock transitions. The truth table indicates the state of the Q outputs under all input conditions.



The register is asynchronously cleared, outputs Low, when power is applied. The power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

SRD4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D3 – D0	C	Q0	Q3	Q2 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D3 – D0	↑	D0	D3	Dn
0	1	X	X	X	X	D3 – D0	↓	D0	D3	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q2	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q2	qn-1



SRD4RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D3 – D0	C	Q0	Q3	Q2 – Q1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD8RLED Truth Table

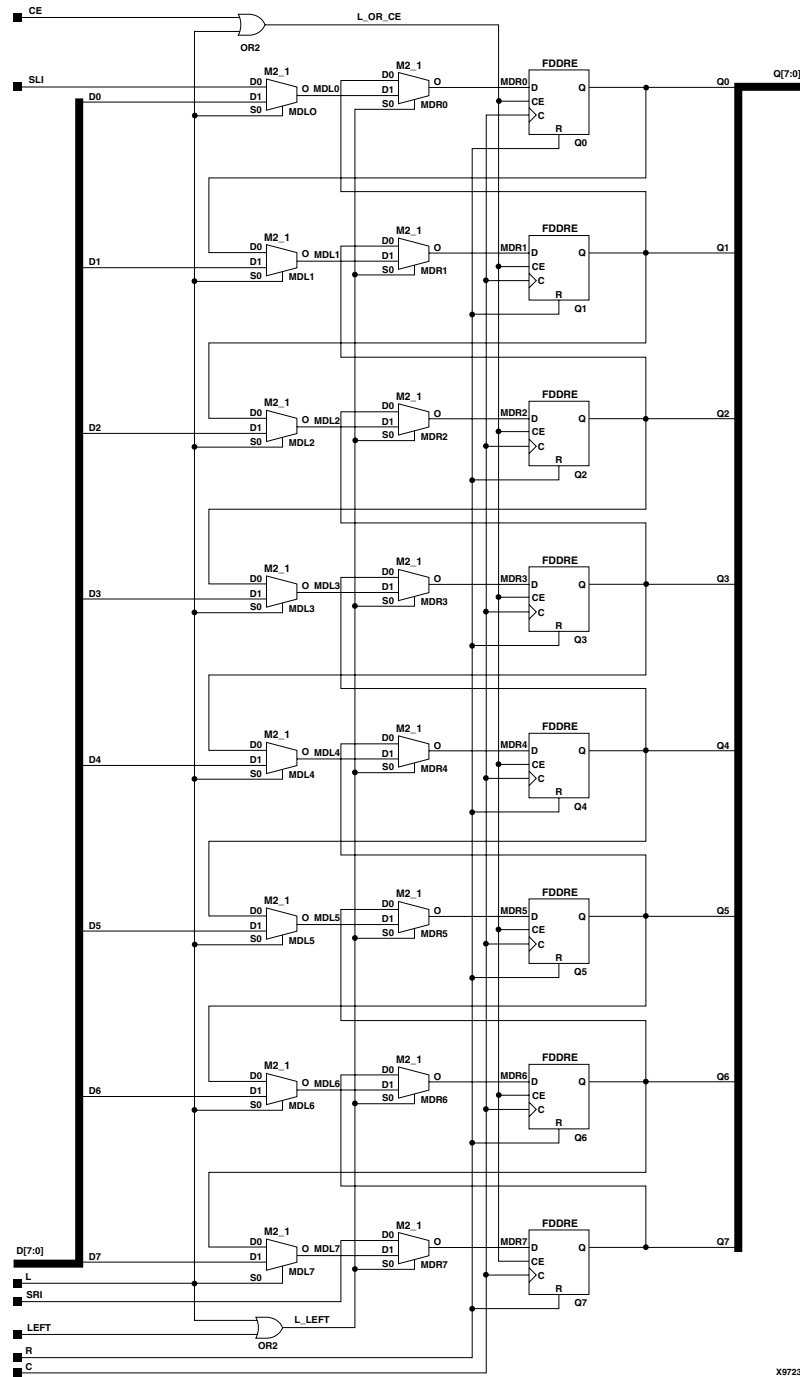
Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D7 – D0	C	Q0	Q7	Q6 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D7 – D0	↑	D0	D7	Dn
0	1	X	X	X	X	D7 – D0	↓	D0	D7	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q6	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q6	qn-1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition

SRD16RLED Truth Table

Inputs								Outputs		
R	L	CE	LEFT	SLI	SRDI	D15 – D0	C	Q0	Q15	Q14 – Q1
1	X	X	X	X	X	X	↑	0	0	0
1	X	X	X	X	X	X	↓	0	0	0
0	1	X	X	X	X	D15 – D0	↑	D0	D15	Dn
0	1	X	X	X	X	D15 – D0	↓	D0	D15	Dn
0	0	0	X	X	X	X	X	No Chg	No Chg	No Chg
0	0	1	1	SLI	X	X	↑	SLI	q14	qn-1
0	0	1	1	SLI	X	X	↓	SLI	q14	qn-1
0	0	1	0	X	SRDI	X	↑	q1	SRDI	qn+1
0	0	1	0	X	SRDI	X	↓	q1	SRDI	qn+1

qn-1 or qn+1 = state of referenced output one setup time prior to active clock transition



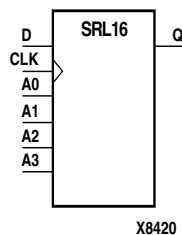
SRD8RLED Implementation CoolRunner-II

SRL16

16-Bit Shift Register Look-Up Table (LUT)

Architectures Supported

SRL16	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRL16 is a shift register look-up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

Static Length Mode

To get a fixed length shift register, drive the A3 through A0 inputs with static values. The length of the shift register can vary from 1 bit to 16 bits as determined from the following formula:

$$\text{Length} = (8 * A3) + (4 * A2) + (2 * A1) + A0 + 1$$

If A3, A2, A1, and A0 are all zeros (0000), the shift register is one bit long. If they are all ones (1111), it is 16 bits long.

Dynamic Length Mode

The length of the shift register can be changed dynamically by changing the values driving the A3 through A0 inputs. For example, if A2, A1, and A0 are all ones (111) and A3 toggles between a one (1) and a zero (0), the length of the shift register changes from 16 bits to 8 bits.

Internally, the length of the shift register is always 16 bits and the input lines A3 through A0 select which of the 16 bits reach the output.

Inputs			Output
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↑	D	Q(Am-1)

m=0,1,2,3

Usage

Below are example templates for instantiating this component into a design. These templates can be cut and pasted directly into the user's source code.

VHDL Instantiation Templates

```
-- SRL16: 16-bit shift register LUT operating on posedge of clock
--      All FPGAs
-- The current version of the Xilinx HDL Libraries Guide

SRL16_inst : SRL16
-- The following generic declaration is only necessary if you wish to
-- change the initial contents of the SRL to anything other than all
-- zero's.
generic map (
    INIT => X"0000")
port map (
    Q => Q,          -- SRL data output
    A0 => A0,        -- Select[0] input
    A1 => A1,        -- Select[1] input
    A2 => A2,        -- Select[2] input
    A3 => A3,        -- Select[3] input
    CLK => CLK,      -- Clock input
    D => D           -- SRL data input
);

-- End of SRL16_inst instantiation
```

Verilog Instantiation Template

```
-- SRL16: 16-bit shift register LUT operating on posedge of clock
--      All FPGAs
-- The current version of the Xilinx HDL Libraries Guide

SSRL16 SRL16_inst (
    .Q(Q),          // SRL data output
    .A0(A0),       // Select[0] input
    .A1(A1),       // Select[1] input
    .A2(A2),       // Select[2] input
    .A3(A3),       // Select[3] input
    .CLK(CLK),     // Clock input
    .D(D)          // SRL data input
);

// The following defparam declaration is only necessary if you wish to
```

```
// change the initial contents of the SRL to anything other than all
// zero's.  If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.
```

```
defparam SRL16_inst.INIT = 16'h0000;
```

```
// End of SRL16_inst instantiation
```

Commonly Used Constraints

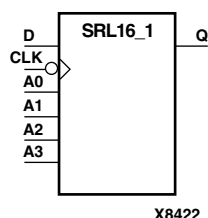
BEL, U_SET, INIT

SRL16_1

16-Bit Shift Register Look-Up Table (LUT) with Negative-Edge Clock

Architectures Supported

SRL16_1	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRL16_1 is a shift register look-up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See “Static Length Mode” and “Dynamic Length Mode” in “SRL16”.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

Inputs			Output
Am	CLK	D	Q
Am	X	X	Q(Am)
Am	↓	D	Q(Am-1)

m=0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRL16_1: 16-bit shift register LUT operating on negedge of clock
--           All FPGAs
-- The current version of the Xilinx HDL Libraries Guide

SRL16_1_inst : SRL16_1
-- The following generic declaration is only necessary if
-- you wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
```

```
        INIT => X"0000")
    port map (
Q => Q,          -- SRL data output
    A0 => A0,      -- Select[0] input
    A1 => A1,      -- Select[1] input
    A2 => A2,      -- Select[2] input
    A3 => A3,      -- Select[3] input
    CLK => CLK,    -- Clock input
    D => D         -- SRL data input
    );

-- End of SRL16_1_inst instantiation
```

Verilog Instantiation Template

```
SRL16_1 SRL16_1_inst (
    .Q(Q),          // SRL data output
    .A0(A0),        // Select[0] input
    .A1(A1),        // Select[1] input
    .A2(A2),        // Select[2] input
    .A3(A3),        // Select[3] input
    .CLK(CLK),      // Clock input
    .D(D)           // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's.  If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRL16_1_inst.INIT = 16'h0000;

// End of SRL16_1_inst instantiation
```

Commonly Used Constraints

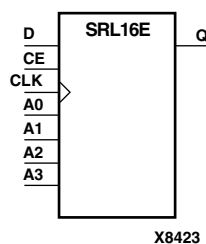
BEL, U_SET, INIT

SRL16E

16-Bit Shift Register Look-Up Table (LUT) with Clock Enable

Architectures Supported

SRL16E	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRL16E is a shift register look-up table (LUT). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. See “[Static Length Mode](#)” and “[Dynamic Length Mode](#)” in “[SRL16](#)”.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions, when CE is High, data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

Inputs				Output
Am	CE	CLK	D	Q
Am	0	X	X	Q(Am)
Am	1	↑	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRL16E: 16-bit shift register LUT with clock enable operating
-- on posedge of clock
-- All FPGAs
-- The current version of the Xilinx HDL Libraries Guide

SRL16E_inst : SRL16E
```

```

-- The following generic declaration is only necessary if
-- you wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
    INIT => X"0000")
port map (
    Q => Q,          -- SRL data output
    A0 => A0,        -- Select[0] input
    A1 => A1,        -- Select[1] input
    A2 => A2,        -- Select[2] input
    A3 => A3,        -- Select[3] input
    CE => CE,        -- Clock enable input
    CLK => CLK,      -- Clock input
    D => D           -- SRL data input
);

-- End of SRL16E_inst instantiation

```

Verilog Instantiation Template

```

SRL16E SRL16E_inst (
    .Q(Q),          // SRL data output
    .A0(A0),        // Select[0] input
    .A1(A1),        // Select[1] input
    .A2(A2),        // Select[2] input
    .A3(A3),        // Select[3] input
    .CE(CE),        // Clock enable input
    .CLK(CLK),      // Clock input
    .D(D)           // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's.  If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRL16E_inst.INIT = 16'h0000;

// End of SRL16E_inst instantiation

```

Commonly Used Constraints

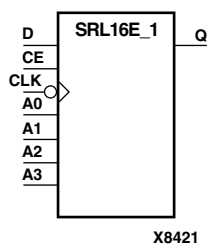
BEL, U_SET, INIT

SRL16E_1

16-Bit Shift Register Look-Up Table (LUT) with Negative-Edge Clock and Clock Enable

Architectures Supported

SRLC16E_1	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRL16E_1 is a shift register look-up table (LUT) with clock enable (CE). The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or dynamically adjusted. See “[Static Length Mode](#)” and “[Dynamic Length Mode](#)” in the “[SRL16](#)”.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions, when CE is High, data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

When CE is Low, the register ignores clock transitions.

Inputs				Output
Am	CE	CLK	D	Q
Am	0	X	X	Q(Am)
Am	1	↓	D	Q(Am-1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRL16E_1: 16-bit shift register LUT with clock enable
-- operating on negedge of clock
--           All FPGAs
-- The current version of the Xilinx HDL Libraries Guide
```

```
SRL16E_1_inst : SRL16E_1
-- The following generic declaration is only necessary if
-- you wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
  INIT => X"0000")
port map (
  Q => Q,          -- SRL data output
  A0 => A0,        -- Select[0] input
  A1 => A1,        -- Select[1] input
  A2 => A2,        -- Select[2] input
  A3 => A3,        -- Select[3] input
  CE => CE,        -- Clock enable input
  CLK => CLK,      -- Clock input
  D => D           -- SRL data input
);

-- End of SRL16E_1_inst instantiation
```

Verilog Instantiation Template

```
SRL16E_1 SRL16E_1_inst (
  .Q(Q),          // SRL data output
  .A0(A0),        // Select[0] input
  .A1(A1),        // Select[1] input
  .A2(A2),        // Select[2] input
  .A3(A3),        // Select[3] input
  .CE(CE),        // Clock enable input
  .CLK(CLK),      // Clock input
  .D(D)           // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's.  If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRL16E_1_inst.INIT = 16'h0000;

// End of SRL16E_1_inst instantiation
```

Commonly Used Constraints

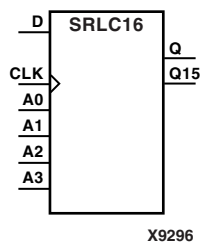
BEL, U_SET, and INIT

SRLC16

16-Bit Shift Register Look-Up Table (LUT) with Carry

Architectures Supported

SRLC16	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRLC16 is a shift register look-up table (LUT) with Carry. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length, or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. During subsequent Low-to-High clock transitions data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

For information about the static length mode, see [“Static Length Mode”](#) in [“SRL16”](#).

For information about the dynamic length mode, see [“Dynamic Length Mode”](#) in [“SRL16”](#).

Inputs			Output
A _m	CLK	D	Q
A _m	X	X	Q(A _m)
A _m	↑	D	Q(A _m -1)

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRLC16: 16-bit cascadable shift register LUT operating on
-- posedge of clock
--           Virtex-II/II-Pro, Spartan-3
```

```
-- The current version of the Xilinx HDL Libraries Guide

SRLC16_inst : SRLC16
-- The following generic declaration is only necessary if
-- you wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
    INIT => X"0000")
port map (
    Q => Q,          -- SRL data output
    Q15 => Q15,     -- Carry output (connect to next SRL)
    A0 => A0,       -- Select[0] input
    A1 => A1,       -- Select[1] input
    A2 => A2,       -- Select[2] input
    A3 => A3,       -- Select[3] input
    CLK => CLK,     -- Clock input
    D => D          -- SRL data input
);

-- End of SRLC16_inst instantiation
```

Verilog Instantiation Template

```
SRLC16 SRLC16_inst (
    .Q(Q),          // SRL data output
    .Q15(Q15),     // Carry output (connect to next SRL)
    .A0(A0),       // Select[0] input
    .A1(A1),       // Select[1] input
    .A2(A2),       // Select[2] input
    .A3(A3),       // Select[3] input
    .CLK(CLK),     // Clock input
    .D(D)          // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's. If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRLC16_inst.INIT = 16'h0000;

// End of SRLC16_inst instantiation
```

Commonly Used Constraints

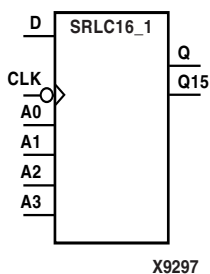
INIT

SRLC16_1

16-Bit Shift Register Look-Up Table (LUT) with Carry and Negative-Edge Clock

Architectures Supported

SRLC16_1	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRLC16_1 is a shift register look-up table (LUT) with carry and a negative-edge clock. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See “[Static Length Mode](#)” and “[Dynamic Length Mode](#)” in “SRL16”.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

Inputs			Output	
Am	CLK	D	Q	Q15
Am	X	X	Q(Am)	No Chg
Am	↓	D	Q(Am-1)	Q14

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred.

Verilog Instantiation Template

```
-- SRLC16_1: 16-bit cascadable shift register LUT operating
-- on negedge of clock
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide
```

```
SRLC16_1_inst : SRLC16_1
-- The following generic declaration is only necessary if you
-- wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
  INIT => X"0000")
port map (
  Q => Q,          -- SRL data output
  Q15 => Q15,     -- Carry output (connect to next SRL)
  A0 => A0,       -- Select[0] input
  A1 => A1,       -- Select[1] input
  A2 => A2,       -- Select[2] input
  A3 => A3,       -- Select[3] input
  CLK => CLK,     -- Clock input
  D => D          -- SRL data input
);

-- End of SRLC16_1_inst instantiation
```

Commonly Used Constraints

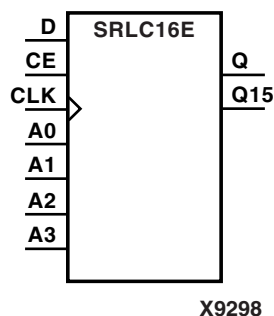
BEL, U_SET, INIT

SRLC16E

16-Bit Shift Register Look-Up Table (LUT) with Carry and Clock Enable

Architectures Supported

SRLC16E	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRLC16E is a shift register look-up table (LUT) with carry and clock enable. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

The data (D) is loaded into the first bit of the shift register during the Low-to-High clock (CLK) transition. When CE is High, during subsequent Low-to-High clock transitions, data shifts to the next highest bit position as new data is loaded. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

For information about the static length mode, see [“Static Length Mode”](#) in [“SRL16”](#).

For information about the dynamic length mode, see [“Dynamic Length Mode”](#) in [“SRL16”](#).

Inputs				Output	
Am	CLK	CE	D	Q	Q15
Am	X	0	X	Q(Am)	Q(15)
Am	X	1	X	Q(Am)	Q(15)
Am	↑	1	D	Q(Am-1)	Q15

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRLC16E: 16-bit cascable shift register LUT with clock
-- enable operating on posedge of clock
--           Virtex-II/II-Pro, Spartan-3
-- The current version of the Xilinx HDL Libraries Guide

SRLC16E_inst : SRLC16E
-- The following generic declaration is only necessary if you
-- wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
    INIT => X"0000")
port map (
    Q => Q,           -- SRL data output
    Q15 => Q15,       -- Carry output (connect to next SRL)
    A0 => A0,         -- Select[0] input
    A1 => A1,         -- Select[1] input
    A2 => A2,         -- Select[2] input
    A3 => A3,         -- Select[3] input
    CE => CE,        -- Clock enable input
    CLK => CLK,       -- Clock input
    D => D            -- SRL data input
);

-- End of SRLC16E_inst instantiation
```

Verilog Instantiation Template

```
SRLC16E SRLC16E_inst (
    .Q(Q),           // SRL data output
    .Q15(Q15),       // Carry output (connect to next SRL)
    .A0(A0),         // Select[0] input
    .A1(A1),         // Select[1] input
    .A2(A2),         // Select[2] input
    .A3(A3),         // Select[3] input
    .CE(CE),         // Clock enable input
    .CLK(CLK),       // Clock input
    .D(D)            // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's.  If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRLC16E_inst.INIT = 16'h0000;

// End of SRLC16E_inst instantiation
```

Commonly Used Constraints

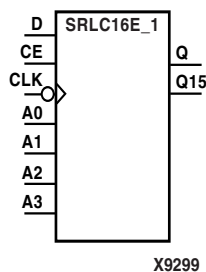
INIT

SRLC16E_1

16-Bit Shift Register Look-Up Table (LUT) with Carry, Negative-Edge Clock, and Clock Enable

Architectures Supported

SRLC16E_1	
Spartan-II, Spartan-IIE	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



SRLC16E_1 is a shift register look-up table (LUT) with carry, clock enable, and negative-edge clock. The inputs A3, A2, A1, and A0 select the output length of the shift register. The shift register may be of a fixed, static length or it may be dynamically adjusted. See “SRLC16” and “Dynamic Length Mode” in “SRL16”.

The shift register LUT contents are initialized by assigning a four-digit hexadecimal number to an INIT attribute. The first, or the left-most, hexadecimal digit is the most significant bit. If an INIT value is not specified, it defaults to a value of four zeros (0000) so that the shift register LUT is cleared during configuration.

When CE is High, the data (D) is loaded into the first bit of the shift register during the High-to-Low clock (CLK) transition. During subsequent High-to-Low clock transitions data shifts to the next highest bit position as new data is loaded when CE is High. The data appears on the Q output when the shift register length determined by the address inputs is reached.

The Q15 output is available for the user to cascade multiple shift register LUTs to create larger shift registers.

Inputs				Output	
Am	CE	CLK	D	Q	Q15
Am	0	X	X	Q(Am)	No Chg
Am	1	X	X	Q(Am)	No Chg
Am	1	↓	D	Q(Am-1)	Q14

m= 0, 1, 2, 3

Usage

For HDL, this design element can be inferred or instantiated.

VHDL Instantiation Template

```
-- SRLC16E_1: 16-bit shift register LUT with clock enable
-- operating on negedge of clock
```

```

--          Virtex-II/II-Pro, Spartan-3
--The current version of the Xilinx HDL Libraries Guide

SRLC16_1_inst : SRLC16_1
-- The following generic declaration is only necessary if you
-- wish to change the initial.
-- contents of the SRL to anything other than all zero's.
generic map (
    INIT => X"0000")
port map (
    Q => Q,          -- SRL data output
    Q15 => Q15,     -- Carry output (connect to next SRL)
    A0 => A0,       -- Select[0] input
    A1 => A1,       -- Select[1] input
    A2 => A2,       -- Select[2] input
    A3 => A3,       -- Select[3] input
    CE => CE,       -- Clock enable input
    CLK => CLK,     -- Clock input
    D => D          -- SRL data input
);

-- End of SRLC16E_1_inst instantiation

```

Verilog Instantiation Template

```

SRLC16E_1 SRLC16E_1_inst (
    .Q(Q),          // SRL data output
    .Q15(Q15),     // Carry output (connect to next SRL)
    .A0(A0),       // Select[0] input
    .A1(A1),       // Select[1] input
    .A2(A2),       // Select[2] input
    .A3(A3),       // Select[3] input
    .CE(CE),       // Clock enable input
    .CLK(CLK),     // Clock input
    .D(D)          // SRL data input
);

// The following defparam declaration is only necessary if you wish to
// change the initial contents of the SRL to anything other than all
// zero's. If the instance name to the SRL is changed, that change
// needs to be reflected in the defparam statements.

defparam SRLC16E_1_inst.INIT = 16'h0000;

// End of SRLC16E_1_inst instantiation

```

Commonly Used Constraints

INIT

STARTBUF_architecture

VHDL Simulation of FPGA Designs

Architectures Supported

STARTBUF_architecture	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

STARTBUF_architecture is used for VHDL simulation of FPGA designs that require the use of the STARTUP block. The difference between the STARTBUF_architecture and the STARTUP block is that the STARTBUF_architecture contains output ports which may be connected to all register set/resets in the design (GSR0UT) or to all I/O three-state controls (GTS0UT) so that these functions may be functionally simulated. This design element should not be used for Verilog or schematic entry. In order to use the STARTBUF_architecture, the desired input(s) should be connected to a top-level port in the design and the corresponding output(s) must be connected to either the three-state control signal for all inferred and instantiated output buffers in the design (GTS0UT) or all inferred or instantiated register set/resets in the design.

During simulation, the inputs to the STARTBUF_architecture can be toggled by the testbench in order to activate the global three-state or global set/reset signal in the design. This should be done at the beginning of the simulation to simulate the behavior of the registers and I/O during configuration. It may also be applied during simulation to simulate a reconfiguration (ProG pin high) of the device. During synthesis and implementation, this component will be treated as a STARTUP block. The connected input ports to this component should remain in the design and be connected to the correct corresponding global resource.

For more information, see the *Xilinx Synthesis and Verification Design Guide*.

The value at port GSR0UT will be always the be value at port GSRIN. The value at port GTS0UT will always be the value at port GTSIN. CLKIN has no effect on simulation.

Usage

This block cannot be realized in hardware and is thus for behavioral simulation only.

VHDL Instantiation Code

Note: This device cannot be realized in hardware and is thus for behavioral simulation only.

Following are four examples:

```
component STARTBUF_SPARTAN2
  port (GSR0UT      : out std_ulogic;
```

```
        GTSOUT      : out std_ulogic;
        CLKIN       : in  std_ulogic;
        GSRIN       : in  std_ulogic;
        GTSIN       : in  std_ulogic);
end component;

component STARTBUF_VIRTEX
  port (GTSOUT      : out std_ulogic;
        GSROUT     : out std_ulogic;
        CLKIN      : in  std_ulogic;
        GSRIN      : in  std_ulogic;
        GTSIN      : in  std_ulogic);
end component;

component STARTBUF_VIRTEX2
  port (GSROUT     : out std_ulogic;
        GTSOUT     : out std_ulogic;
        CLKIN      : in  std_ulogic;
        GSRIN      : in  std_ulogic;
        GTSIN      : in  std_ulogic);
end component;

component STARTBUF_VIRTEX4
  port(
    EOSOUT : out std_ulogic;
    GSROUT : out std_ulogic;
    GTSOUT : out std_ulogic;

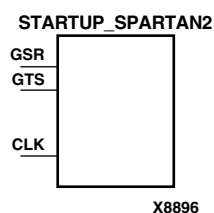
    CLKIN  : in  std_ulogic := 'X';
    GSRIN  : in  std_ulogic := 'X';
    GTSIN  : in  std_ulogic := 'X'
  );
end component;
```

STARTUP_SPARTAN2

Spartan-II User Interface to Global Clock, Reset, and 3-State Controls

Architectures Supported

STARTUP_SPARTAN2	
Spartan-II, Spartan-IIE	Primitive*
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No
* Supported for Spartan-II but not for Spartan-IIE, which is supported by STARTUP_VIRTEX.	



The STARTUP_SPARTAN2 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component.

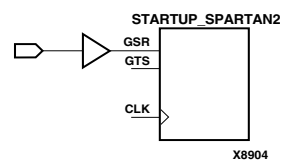
Note: Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

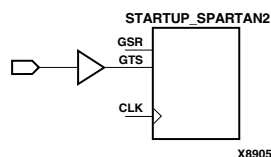
Note: GTS= Global 3-State

Including the STARTUP_SPARTAN2 symbol in a design is optional. You must include the symbol under the following conditions.

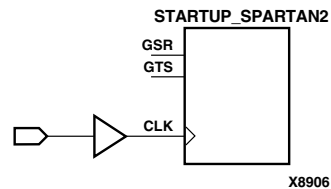
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- STARTUP_SPARTAN2: Startup primitive for GSR, GTS or
-- startup sequence
--                               control. Spartan-II
-- The current version of the Xilinx HDL Libraries Guide

STARTUP_SPARTAN2_inst : STARTUP_SPARTAN2
port map (
    CLK => CLK,    -- Clock input for start-up sequence
    GSR => GSR,    -- Global Set/Reset input
    GTS => GTS     -- Global 3-state input
);

-- End of STARTUP_SPARTAN2_inst instantiation
```

Verilog Instantiation Template

```
STARTUP_SPARTAN2 STARTUP_SPARTAN2_inst (
    .CLK(CLK),    // Clock input for start-up sequence
    .GSR(GSR),    // Global Set/Reset input
    .GTS(GTS)     // Global 3-state input
);

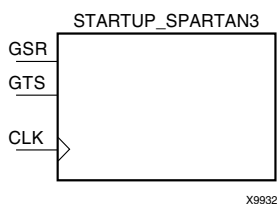
// End of STARTUP_SPARTAN2_inst instantiation
```


STARTUP_SPARTAN3

Spartan-3 User Interface to Global Clock, Reset, and 3-State Controls

Architectures Supported

STARTUP_SPARTAN3	
Spartan-II, Spartan-III	No
Spartan-3	Primitive
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The STARTUP_SPARTAN3 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAMB16 output register in the device, depending on the initialization state (INIT=1 or 0) of the component.

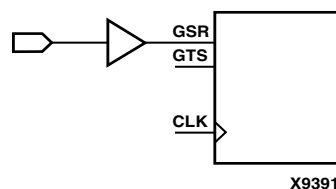
Note: Block RAM content, LUT RAMs, the Digital Clock Manager (DCM), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1, SRLC16, SRLC16_1, SRLC16E, and SRLC16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

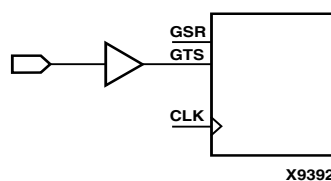
Note: GTS= Global 3-State

Including the STARTUP_SPARTAN3 symbol in a design is optional. You must include the symbol under the following conditions.

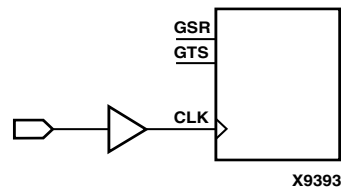
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- STARTUP_SPARTAN3: Startup primitive for GSR, GTS or startup sequence
--                               control. Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

STARTUP_SPARTAN3_inst : STARTUP_SPARTAN3
port map (
    CLK => CLK,    -- Clock input for start-up sequence
    GSR => GSR,    -- Global Set/Reset input
    GTS => GTS     -- Global 3-state input
);

-- End of STARTUP_SPARTAN3_inst instantiation
```

Verilog Instantiation Template

```
STARTUP_SPARTAN3 STARTUP_SPARTAN3_inst (
    .CLK(CLK),    // Clock input for start-up sequence
    .GSR(GSR),   // Global Set/Reset input
    .GTS(GTS)    // Global 3-state input
);

// End of STARTUP_SPARTAN3_inst instantiation
```

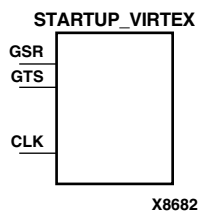
STARTUP_VIRTEX

Virtex and Virtex-E User Interface to Global Clock, Reset, and 3-State Controls

Architectures Supported

STARTUP_VIRTEX	
Spartan-II, Spartan-IIE	Primitive*
Spartan-3	No
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

* Supported for Spartan-IIE but not for Spartan-II which is supported by STARTUP_SPARTAN2..



The STARTUP_VIRTEX primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAM (RAMB4) output register in the device, depending on the initialization state (S or R) of the component. For Virtex-II, Virtex-II Pro, and Virtex-II Pro X, see “STARTUP_VIRTEX2”.

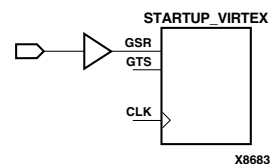
Note: Block RAMB4 content, LUT RAMs, delay locked loop elements (CLKDLL, CLKDLLHF, BUFGDLL), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

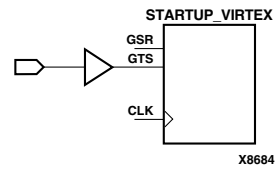
Note: GTS= Global 3-State

Including the STARTUP_VIRTEX symbol in a design is optional. You must include the symbol under the following conditions.

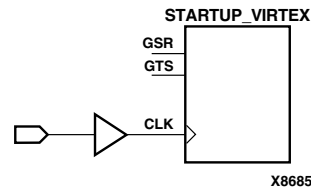
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- STARTUP_VIRTEX: Startup primitive for GSR, GTS or startup sequence
--                   control. Virtex/E, Spartan-IIE
-- The current version of the Xilinx HDL Libraries Guide

STARTUP_VIRTEX_inst : STARTUP_VIRTEX
port map (
    CLK => CLK,    -- Clock input for start-up sequence
    GSR => GSR,    -- Global Set/Reset input
    GTS => GTS     -- Global 3-state input
);

-- End of STARTUP_VIRTEX_inst instantiation
```

Verilog Instantiation Template

```
STARTUP_VIRTEX STARTUP_VIRTEX_inst (
    .CLK(CLK),    // Clock input for start-up sequence
    .GSR(GSR),   // Global Set/Reset input
    .GTS(GTS)    // Global 3-state input
);

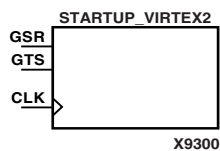
// End of STARTUP_VIRTEX_inst instantiation
```

STARTUP_VIRTEX2

Virtex-II, Virtex-II Pro, and Virtex-II Pro X User Interface to Global Clock, Reset, and 3-State Controls

Architectures Supported

STARTUP_VIRTEX2	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



The STARTUP_VIRTEX2 primitive is used for Global Set/Reset, global 3-state control, and the user configuration clock. The Global Set/Reset (GSR) input, when High, sets or resets all flip-flops, all latches, and every block RAMB16 output register in the device, depending on the initialization state (INIT=1 or 0) of the component. For Virtex and Virtex-E, see “STARTUP_VIRTEX”.

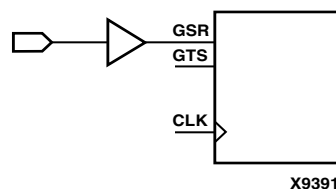
Note: Block RAM content, LUT RAMs, the Digital Clock Manager (DCM), and shift register LUTs (SRL16, SRL16_1, SRL16E, SRL16E_1, SRLC16, SRLC16_1, SRLC16E, and SRLC16E_1) are not set/reset.

Following configuration, the global 3-state control (GTS), when High—and BSCAN is not enabled and executing an EXTEST instruction—forces all the IOB outputs into high impedance mode, which isolates the device outputs from the circuit but leaves the inputs active.

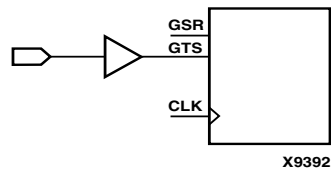
Note: GTS= Global 3-State

Including the STARTUP_VIRTEX2 symbol in a design is optional. You must include the symbol under the following conditions.

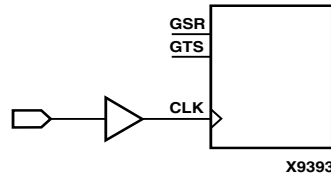
- To exert external control over global set/reset, connect the GSR pin to a top level port and an IBUF, as shown here.



- To exert external control over global 3-state, connect the GTS pin to a top level port and IBUF, as shown here.



- To synchronize startup to a user clock, connect the user clock signal to the CLK input, as shown here. Furthermore, “user clock” must be selected in the BitGen program.



You can use location constraints to specify the pin from which GSR or GTS (or both) is accessed.

Usage

For HDL, this design element typically is instantiated rather than inferred.

VHDL Instantiation Template

```
-- STARTUP_VIRTEX2: Startup primitive for GSR, GTS or startup sequence
--                   control. Virtex-II/II-Pro
-- The current version of the Xilinx HDL Libraries Guide

STARTUP_VIRTEX2_inst : STARTUP_VIRTEX2
port map (
    CLK => CLK,    -- Clock input for start-up sequence
    GSR => GSR,    -- Global Set/Reset input
    GTS => GTS     -- Global 3-state input
);

-- End of STARTUP_VIRTEX2_inst instantiation
```

Verilog Instantiation Template

```
STARTUP_VIRTEX2 STARTUP_VIRTEX2_inst (
    .CLK(CLK),    // Clock input for start-up sequence
    .GSR(GSR),   // Global Set/Reset input
    .GTS(GTS)    // Global 3-state input
);

// End of STARTUP_VIRTEX2_inst instantiation
```

TOC

Three-State On Configuration

Architectures Supported

TOC	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

The TOC is a component used for VHDL simulation of FPGA designs. This component should not be used for Verilog or schematic entry. The TOC's function is to mimic the function of the internal three-state signal during the FPGA configuration process. In order to use TOC, it must be connected to the three-state signal for all inferred and instantiated output buffers in the design. During synthesis and implementation, this three-state signal will use the dedicated global three-state network and will not use local routing resources. During simulation, TOC will emit a one-shot pulse for the amount of time specified by the WIDTH generic (default is 100 ns). This one-shot pulse is intended to three-state all outputs so that at the beginning of operation, all outputs are not being driven as would happen in the real silicon during configuration of the device.

For more information, see the *Xilinx Synthesis and Verification Design Guide*.

Port O will be high at simulation time 0 for the amount of time specified by the WIDTH generic attribute. After that time, it will be 0. This will not affect implementation in any way.

VHDL Instantiation Code

```
component TOC
-- synthesis translate_off
  generic (WIDTH : Time := 100 ns);
-- synthesis translate_on
  port (O : out std_ulogic := '0');
end component;
```

Commonly Used Constraints

For simulation, the WIDTH generic can be modified to change the amount of time the one-shot pulse is applied for.

There are no supported constraints for this component for implementation.

TOCBUF

Three-State On Configuration Buffer

Architectures Supported

TOCBUF	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No

The TOCBUF is a component used for VHDL simulation of FPGA designs. It is similar to the TOC component except that it contains an input for controlling the global I/O three-state function rather than a one-shot. This component should not be used for Verilog or schematic entry. The TOCBUF's function allows user control of the function of the global I/O three-state signal as done during the FPGA configuration process. In order to use the TOCBUF, the input should be connected to a top-level port in the design and the output must be connected to the three-state control signal for all inferred and instantiated output buffers in the design.

During simulation, the input to the TOCBUF can be toggled by the testbench in order to activate the global three-state signal in the design. This should be done at the beginning of the simulation to simulate the behavior of the I/O during configuration. It may also be applied during simulation to simulate a reconfiguration (ProG pin high) of the device. During synthesis and implementation, this three-state signal uses the dedicated global three-state network and does not use local routing resources. The port connected to this component is optimized out of the design and does not use any pin resources. If you want to have the port implemented in the design, a `STARTBUF_architecture` should be used. In order to replace this port during back-end simulation, the `-tp` switch should be used when invoking the `NGD2VER` or `NGD2VHDL` netlister. If using the ISE GUI, use the Bring Out Global Three-state Net as a Port option in the Simulation Model Properties window.

For more information, see the *Xilinx Synthesis and Verification Design Guide*.

The value at port O will be always be the value at port I (it is a buffer).

VHDL Instantiation Code

```
component TOCBUF
  port (I : in std_ulogic;
        O : out std_ulogic);
end component;
```

Commonly Used Constraints

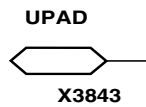
None

UPAD

Connects the I/O Node of an IOB to the Internal PLD Circuit

Architectures Supported

UPAD	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



A UPAD allows the use of any unbonded IOBs in a device. It is used the same way as an IOPAD except that the signal output is not visible on any external device pins.

VCC

VCC-Connection Signal Tag

Architectures Supported

VCC	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



The VCC signal tag or parameter forces a net or input function to a logic High level. A net tied to VCC cannot have any other source.

When the placement and routing software encounters a net or input function tied to VCC, it removes any logic that is disabled by the VCC signal. The VCC signal is only implemented when the disabled logic cannot be removed.

Usage

VHDL Instantiation Template

```
-- Component Declaration for VCC should be placed
-- after architecture statement but before begin keyword

component VCC
  port (P : out STD_ULOGIC);
end component;

-- Component Attribute specification for VCC
-- should be placed after architecture declaration but
-- before the begin keyword

-- Enter attributes here

-- Component Instantiation for VCC should be placed
-- in architecture after the begin keyword

VCC_INSTANCE_NAME : VCC
  port map (P => user_P);
```

Verilog Instantiation Template

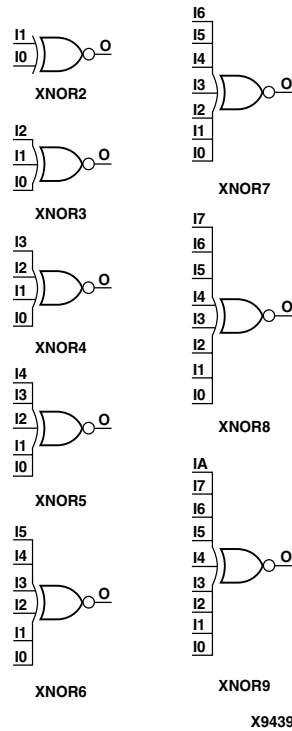
```
VCC instance_name (.P (user_P));
```

XNOR2-9

2- to 9-Input XNOR Gates with Non-Inverted Inputs

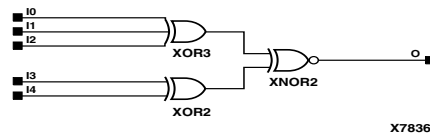
Architectures Supported

XNOR2, XNOR3, XNOR4	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XNOR5	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XNOR6, XNOR7, XNOR8	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XNOR9	
Spartan-II, Spartan-IIE	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

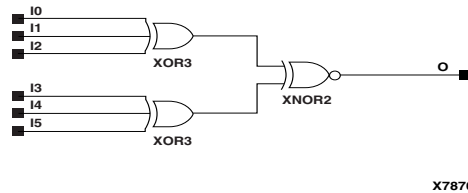


XNOR Gate Representations

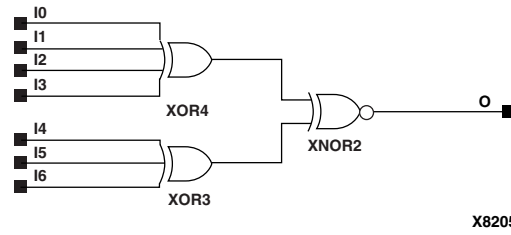
XNOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.



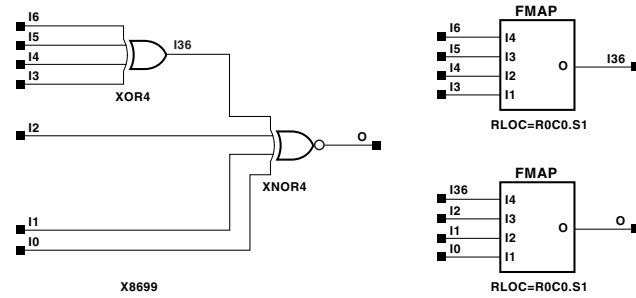
XNOR5 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



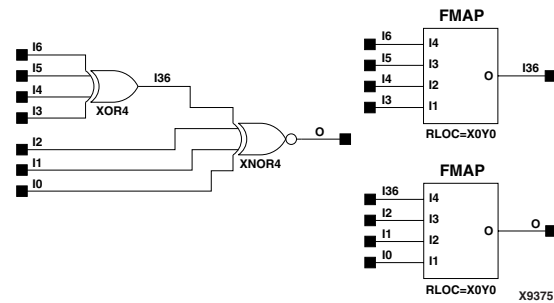
XNOR6 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



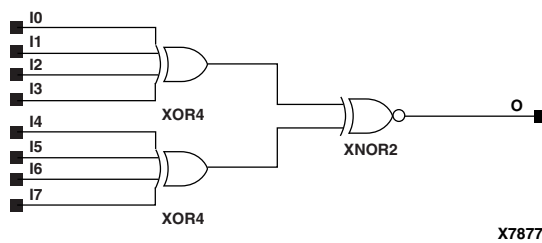
XNOR7 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



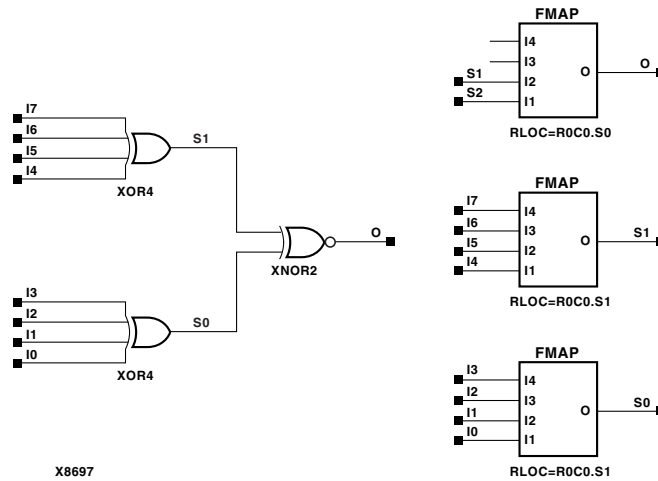
XNOR7 Implementation Spartan-II, Spartan-II E, Virtex, Virtex-E



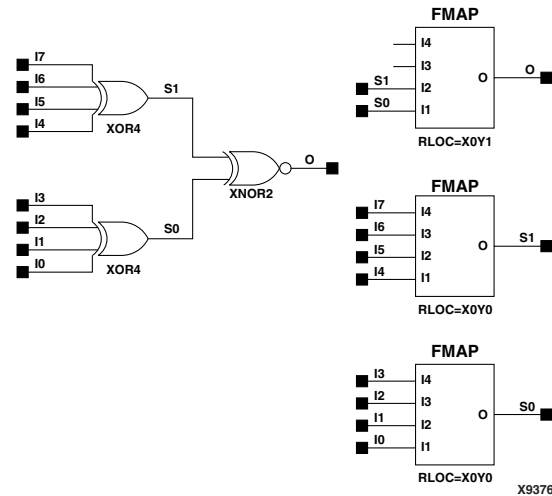
XNOR7 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



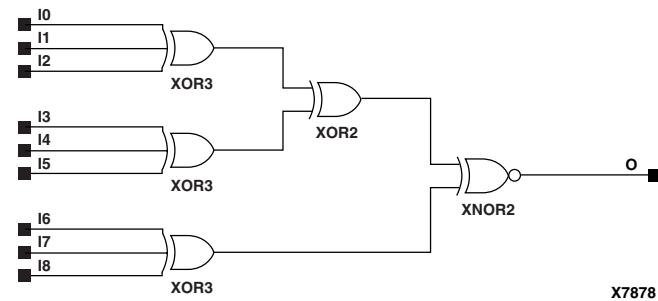
XNOR8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



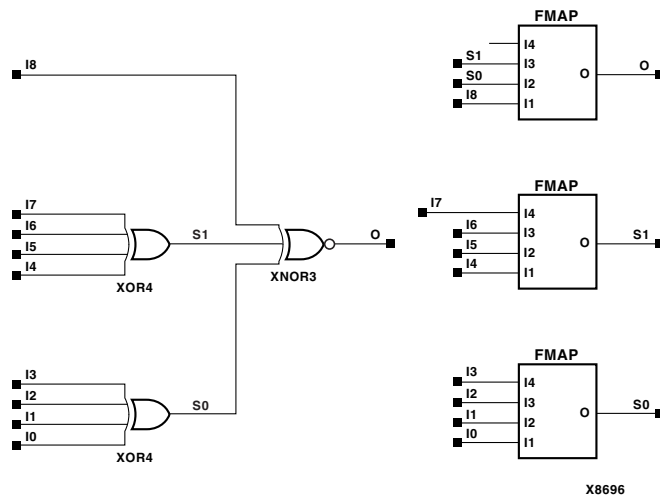
XNOR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



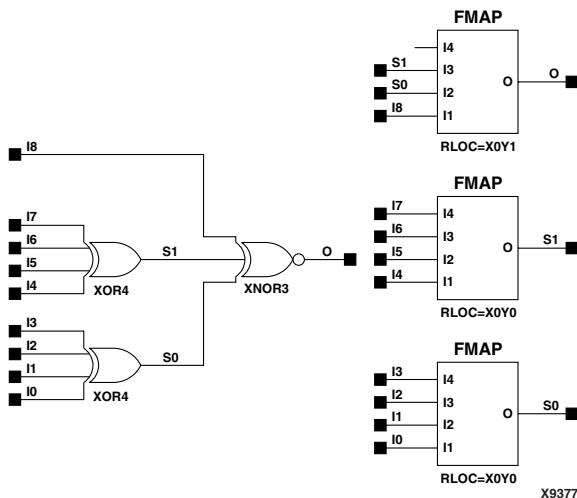
XNOR8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



XNOR9 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



XNOR9 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



XNOR9 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X

Usage

For HDL, these design elements can be inferred or instantiated.

VHDL Instantiation Template for XNOR5

Following is the VHDL code for XNOR5. To instantiate XNOR2, remove I2, I3, and I4. To instantiate XNOR3, remove I3 and I4, For XNOR4, remove I4. XNOR2B1, and XNOR2B2 have the same code as XNOR2. XNOR3B1, 3B2, and 3B3 have the same code as XNOR3 etc..

```
-- Component Declaration for XNOR5 should be placed
-- after architecture statement but before begin keyword
```

```
component XNOR5
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
```

```
        I2 : in STD_ULOGIC;
        I3 : in STD_ULOGIC;
        I4 : in STD_ULOGIC);
end component;

-- Component Attribute specification for XNOR5
-- should be placed after architecture declaration but
-- before the begin keyword

-- Attributes should be placed here

-- Component Instantiation for XNOR5 should be placed
-- in architecture after the begin keyword

XNOR5_INSTANCE_NAME : XNOR5
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2,
              I3 => user_I3,
              I4 => user_I4);
```

Verilog Instantiation Template for XNOR5

```
XNOR5 XNOR5_instance_name (.O (user_O),
                           .I0 (user_I0),
                           .I1 (user_I1),
                           .I2 (user_I2),
                           .I3 (user_I3),
                           .I4 (user_I4));
```

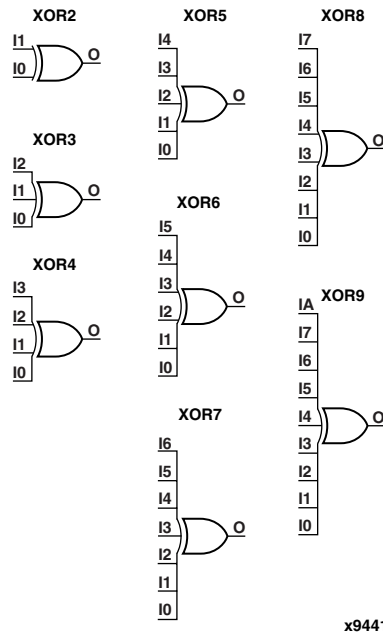
XOR2-9

2- to 9-Input XOR Gates with Non-Inverted Inputs

Architectures Supported

XOR2, XOR3	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XOR4	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XOR5	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive
XOR6, XOR7, XOR9	
Spartan-II, Spartan-III	Macro
Spartan-3	Macro
Virtex, Virtex-E	Macro
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Macro
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive

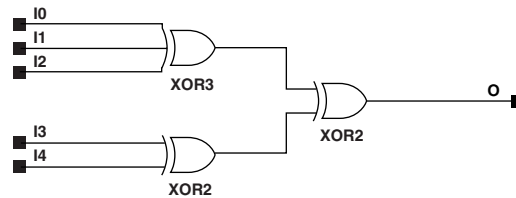
XOR8	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



x9441

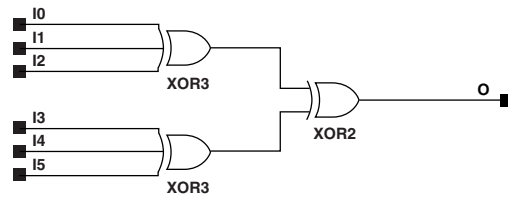
XOR Gate Representations

XOR functions of up to nine inputs are available. All inputs are non-inverting. Because each input uses a CLB resource, replace functions with unused inputs with functions having the necessary number of inputs.



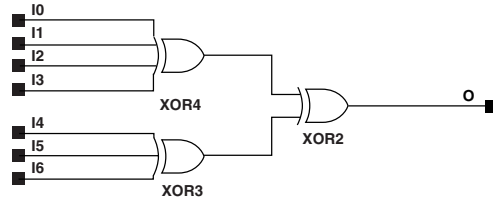
X7882

XOR5 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



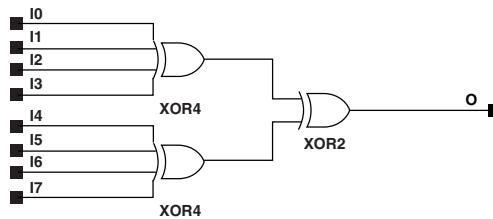
X7883

XOR6 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



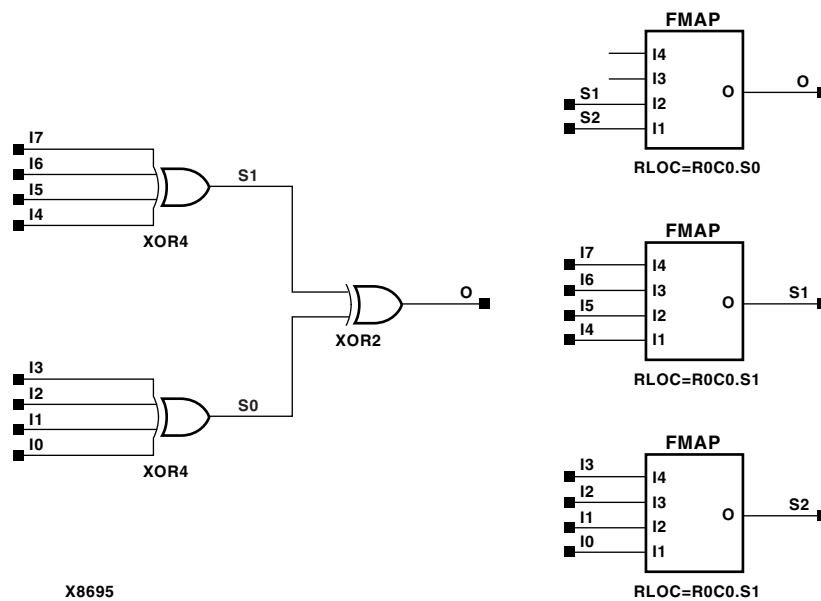
X7884

XOR7 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II



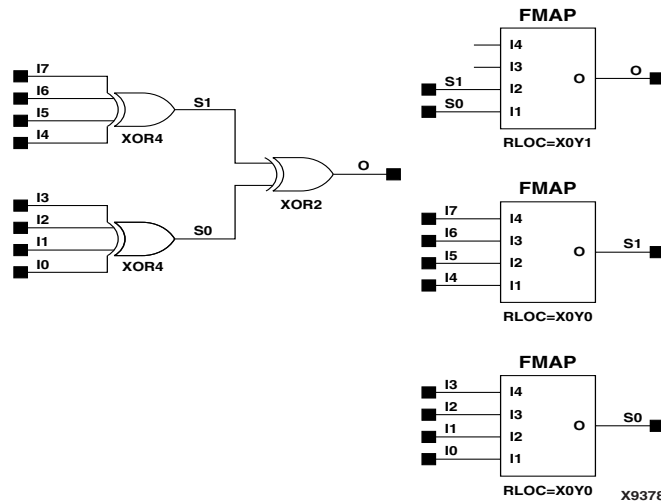
X7885

XOR8 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

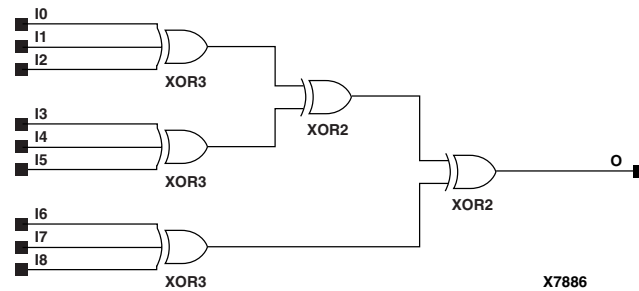


X8695

XOR8 Implementation Spartan-II, Spartan-IIE, Virtex, Virtex-E



XOR8 Implementation Spartan-3, Virtex-II, Virtex-II Pro, Virtex-II Pro X



XOR9 Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements can be inferred or instantiated.

VHDL Instantiation Template for XOR5

Following is the VHDL code for XOR5. To instantiate XOR2, remove I2, I3, and I4. To instantiate XOR3, remove I3 and I4, For XOR4, remove I4. XOR2B1, and XOR2B2 have the same code as XOR2. XOR3B1, 3B2, and 3B3 have the same code as XOR3 etc..

-- Component Declaration for XOR5 should be placed
 -- after architecture statement but before begin keyword

```

component XOR5
  port (O : out STD_ULOGIC;
        I0 : in STD_ULOGIC;
        I1 : in STD_ULOGIC;
        I2 : in STD_ULOGIC;
        I3 : in STD_ULOGIC;
        I4 : in STD_ULOGIC);
end component;
```

-- Component Attribute specification for XOR5
 -- should be placed after architecture declaration but
 -- before the begin keyword


```
-- Attributes should be placed here

-- Component Instantiation for XOR5 should be placed
-- in architecture after the begin keyword

XOR5_INSTANCE_NAME : XOR5
    port map (O => user_O,
              I0 => user_I0,
              I1 => user_I1,
              I2 => user_I2,
              I3 => user_I3,
              I4 => user_I4);
```

Verilog Instantiation Template for XOR5

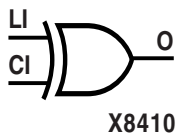
```
XOR5 XOR5_instance_name (.O (user_O),
                          .I0 (user_I0),
                          .I1 (user_I1),
                          .I2 (user_I2),
                          .I3 (user_I3),
                          .I4 (user_I4));
```


XORCY

XOR for Carry Logic with General Output

Architectures Supported

XORCY	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



XORCY is a special XOR with general O output that generates faster and smaller arithmetic functions.

Its O output is a general interconnect. See also ["XORCY_D"](#) and ["XORCY_L"](#).

Commonly Used Constraints

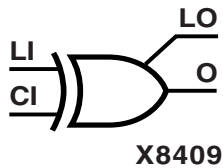
BEL

XORCY_D

XOR for Carry Logic with Dual Output

Architectures Supported

XORCY_D	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



XORCY_D is a special XOR that generates faster and smaller arithmetic functions.

XORCY_D has two functionally identical outputs, O and LO. The O output is a general interconnect. The LO output is used to connect to another output within the same CLB slice.

See also ["XORCY"](#) and ["XORCY_L."](#)

Commonly Used Constraints

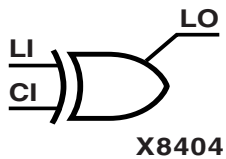
BEL

XORCY_L

XOR for Carry Logic with Local Output

Architectures Supported

XORCY_L	
Spartan-II, Spartan-III	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	No



XORCY_L is a special XOR with local LO output that generates faster and smaller arithmetic functions. The LO output is used to connect to another output within the same CLB slice.

See also ["XORCY"](#) and ["XORCY_D."](#)

Commonly Used Constraints

BEL

