Chapter 1

DESIGN METHODS IN SUB-MICRON TECHNOLOGIES

Yan Lin and Lei He
Electrical Engineering Department
University of California, Los Angeles, CA 90095

Field Programmable Gate Arrays (FPGA) provides an attractive design platform with low NRE (non-recurring engineering) cost and short time-to-market. Due to a large number of transistors for field programmability and the low utilization rate of FPGA resources, existing FPGAs consume more power compared to Specific Integrated Circuits (ASICs). For example, [1] compared an 8-bit adder implemented in a Xilinx XC4003A FPGA with the same adder implemented in a fully customized CMOS ASIC, and showed a 100X difference in energy consumption (4.2mW/MHz at 5V for FPGA versus 5.5uW/Mhz at 3.3V for ASIC counterpart). As the process advances to nanometer technology and low-energy embedded applications are explored for FPGAs, power consumption becomes a crucial design constraint for FPGAs.

Meanwhile, modern VLSI designs see a large impact from process variation as devices scale down to nanometer technologies. Variability in device parameters such as effective channel length, threshold voltage and gate oxide thickness incurs uncertainties in both chip performance and power consumption. For example, measured variation in chip-level leakage can be as high as 20X compared to the nominal value for high performance microprocessors [2]. In addition to meeting the performance constraint under timing variation, dies with excessively large leakage due to such a high variation have to be rejected to meet the given power budget. Although FPGA has a regular fabric with replicated layout tiles, the design-dependent systematical variation can also be significant in advanced technologies such as 65nm and below. Meanwhile, it suffers from the increasingly large random variation like ASIC does.

This chapter addresses the design methods in sub-micron technologies considering optimization for power and process variation. The remaining of this chapter is organized as the following. Section 1 presents architecture optimization techniques for power and variation. Section 2 and Section 3 present power and variation aware synthesis techniques, respectively.

1. Architecture Optimization

1.1 Power Optimization

1.1.1 Power Modeling and Evaluation Framework. In order to perform architecture optimization for low-power FPGAs, Several recent papers have studied FPGA power modeling [3, 4, 5, 6]. Parameterized power models were proposed for generic FPGA architectures first in [3] and [4]. However, both [3] and [4] over-simplified the models for short-circuit and leakage power, and verification by measurement or circuit-level simulation was not reported in [3] [4]. One of the evaluation framework, *fpgaEVA-LP2* [5, 6], improved the power model in [4] and achieved high fidelity and accuracy in chip-level power consumption compared to SPICE simulation.

fpgaEVA-LP2 includes a *BC-netlist* generator and a cycle-accurate power simulator *Psim*. The BC-netlist generator takes placement and routing results by VPR [7] and generates the Basic Circuit netlist (BC-netlist) annotated with post-layout capacitance and delay. Psim then performs cycle-accurate simulation on the BC-netlist to obtain FPGA power consumption. There are three types of power sources in FPGAs, switching power, short-circuit power and static power. The first two types of power contribute to the dynamic power and can only occur when a signal transition happens at the gate output. There are two types of signal transitions. *Functional transition* is the necessary signal transition to perform the required logic functions between two consecutive clock ticks. *Spurious transition* or *glitch* is the unnecessary signal transition due to the imbalanced path delays to the inputs of a gate. Glitch power can be a significant portion of the dynamic power. The short-circuit power is consumed when both PMOS and NMOS transistors are turned on in a gate. The third type of power, static (leakage) power, is the power consumed when there is no signal transition for a gate or a circuit element.

In fpgaEVA-LP2, the power components of switching power, short-circuit power and static power for logic blocks are pre-calculated per switch or per unit time by SPICE simulation, and so is the leakage power for interconnects. The interconnect switching power is calculated by a switch-level model with extracted parasitics, and its short-circuit power is calculated as a portion of switching power. This portion can be pre-calculated by SPICE simulation for a variety of input signal transition time and load capacitances. It has been shown in [5, 6] that fpgaEVA-LP2 achieves high fidelity as well as high accuracy. The average absolute error is 8.26% compared to SPICE simulation.

1.1.2 Vdd-Programmable FPGA Circuits. To reduce FPGA power consumption, power-efficient FPGA circuits and architectures have been studied. [8] studied region-based power-gating and placement to reduce leakage power of unused FPGA logic blocks. [9] applied fine-grained power-gating

to FPGA interconnects. [10, 11] proposed dual-Vdd and Vdd-programmable FPGA logic blocks and simple yet effective CAD algorithms to reduce both dynamic and leakage power. Field programmability of supply voltage was shown to be necessary to reduce FPGA power using dual-Vdd technique. Later on, the concept of programmable-Vdd introduced in [11] was further extended to FPGA interconnects in [12, 13, 14, 5, 15].

Figure 1.1 shows the Vdd-programmable logic block [11]. Two extra PMOS transistors, called *power switches* or *power transistors* [1], are inserted between the conventional logic block and the dual-Vdd power rails for Vdd selection and power-gating. Normal-Vt power transistors with gate-boosting [11] are used to reduce area overhead and achieve effective leakage reduction.

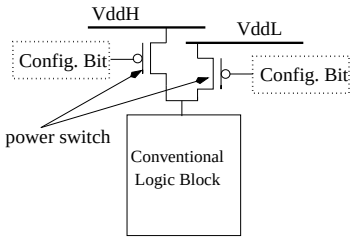


Figure 1.1. Vdd-programmable logic block.

Vdd-programmability introduces extra configuration SRAM cells. The authors in [5, 15] designed a new Vdd-programmable routing switch (see in Figure 1.2 (b)). One extra SRAM is introduced for this Vdd-programmable routing switch. $Pass\_En$ can be generated by $VddH\_En$ and $VddL\_En$ with a NAND2 gate. Similarly, Figure 1.2 (c) presents a new design of Vdd-programmable connection block with reduced configuration SRAM cells. For a connection block containing N connection switches, we use a $\lceil log_2 N \rceil : N$ decoder and $2N$ NAND2 gates as the control logic. There is a disable signal $Dec\_Disable$ for the decoder. Each decoder output is connected to $Pass\_En$ of one connection switch. Setting $Pass\_En$ of a connection switch to '0' can power-gate this switch by setting both $VddH\_En$ and $VddL\_En$ to '1' with NAND2 gates. When the whole connection block is not used, all N outputs of the decoder are set to '0' to power-gate all the connection switches by asserting $Dec\_Disable$. When the connection block is in use, $Dec\_Disable$ is not asserted. By using $\lceil log_2 N \rceil$ configuration bits for the decoder, only one $Pass\_En$ is set to '1' and others are set to '0', i.e., only one connection switch inside the connection block is selected and connects the one track to the logic block input, and other unused connection switches are power-gated. Another

[1] The terms power switch and power transistor are used interchangeably in this chapter.

configuration bit Vdd_Sel is used to select the Vdd-level for the selected connection switch. Compared to a conventional connection block, only two extra configuration SRAM cells are introduced. Power-gating unused switches can reduce leakage power by more than 1000X [12].

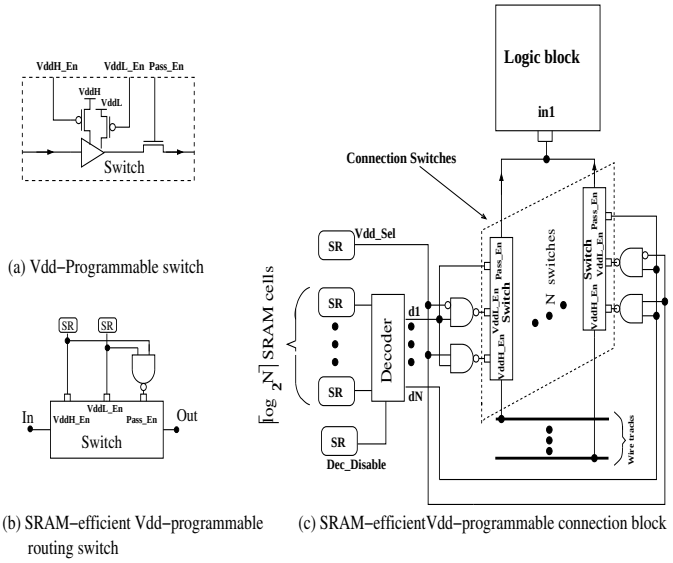


Figure 1.2. (a) Vdd-programmable switch (b) SRAM-efficient Vdd-programmable routing switch; (c) SRAM-efficient Vdd-programmable connection block. (SR stands for SRAM cell)

Compared to Vdd-programmable switch, Vdd-gateable interconnect switch proposed in [5] only provides two power states between a pre-determined Vdd-level and power-gating, but it can dramatically reduce the number of extra SRAM cells for Vdd programmability. Figure 1.3 (a) shows the circuit design for a Vdd-gateable switch. Based on a conventional tri-state buffer, a PMOS transistor M2 is inserted between the power rail and the tri-state buffer to provide the power-gating capability. When a switch is not used, transistor M1 is turned off by the configuration cell SR. At the same time, we can turn off M2 to power-gate the unused switch. Similarly, both M1 and M2 are turned on by the configuration cell SR when the switch is used. Thus, we do not need to introduce an extra SRAM cell for power-gating capability. Figure 1.3 (b) presents Vdd-gateable routing switches. We can reduce leakage power by a factor of over 1000 for an unused switch when it is power-gated. With the power transistor properly sized, the delay overhead for a Vdd-programmable or Vdd-gateable switch is bounded by 16%.

1.1.3 Architecture Optimization for Power Efficient FPGA. Previously, conventional FPGA architecture evaluation has been performed using

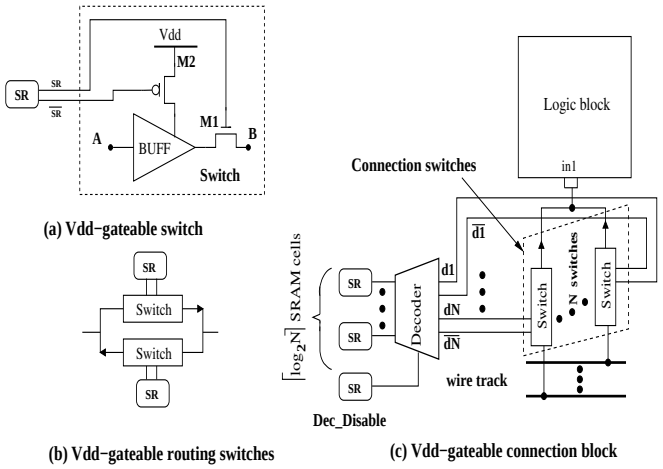


Figure 1.3. (a) Vdd-gateable switch; (b) Vdd-gateable routing switches; (c) Vdd-gateable connection switches. (SR stands for SRAM cell)

metrics of area, delay and energy. [16] showed that LUT size 4 obtains the smallest area, and [17] showed that LUT size 5 or 6 leads to the best performance in non-clustered FPGAs. [18] evaluated cluster-based island style FPGAs using the metric of area-delay product in $0.35\mu m$ technology, and showed that the range of LUT sizes from 4 to 6 and cluster sizes between 4 and 10 can produce the best area-delay product. The following work further extended FPGA architecture evaluation considering energy. [3] showed that LUT size 3 consumes the smallest energy in $0.35\mu m$ technology. [4, 6] showed that LUT size 4 consumes the smallest energy and LUT size 7 leads to the best performance in $100nm$ technology.

However, the emerging power-efficient circuits and architectures, e.g. the Vdd-programmable FPGA circuits, may lead to different FPGA power characteristics, and therefore call for an architecture evaluation considering these power optimization techniques. The authors in [5, 15] evaluated three architecture classes *Class1*, *Class2* and *Class3* for Vdd-programmable FPGAs at 100nm technology. Class1 applies programmable dual-Vdd to each logic block and each interconnect segment, and inserts a configurable level conversion circuit in front of each routing/connection switch as well as at the inputs/outputs of the logic blocks. Class2 applies programmable dual-Vdd only to logic blocks, and uses Vdd-gateable routing/connection switches in FPGA interconnects. Class3 uses the same SRAM-efficient interconnect switches as FPGA architecture Class1, but inserts level converters only at logic block inputs and outputs. All these architecture classes are summarized in Table 1.1.

Architecture Class	Logic block	Interconnect
Class0 (baseline)	single Vdd	single Vdd
Class1	programmable dual-Vdd	programmable dual-Vdd w/ level converters
Class2	programmable dual-Vdd	VddH and Vdd-gateable
Class3	programmable dual-Vdd	programmable dual-Vdd w/o level converters

Table 1.1. Summary of baseline architecture class and Vdd-programmable architecture classes (LC denotes the level converter).

A simple yet practical design flow from [11] was applied for these architecture classes. After placement and routing, a greedy algorithm is performed for Vdd assignment considering iteratively updated timing slack. The assignment unit is a logic block or an interconnect switch for Class1, or a logic block for Class2. Since there is no level converter in the routing for Class3, the assignment unit is a logic block or an interconnect routing tree to avoid low-Vdd switch driving high-Vdd switches.

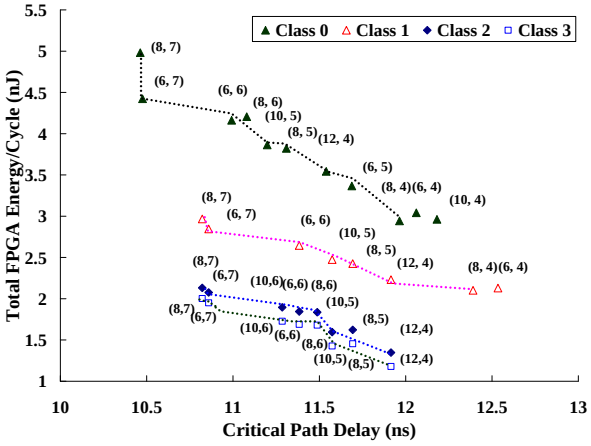


Figure 1.4. Energy and delay tradeoff for all FPGA architecture classes. The figure only shows relaxed energy-delay dominant solutions and the strictly dominant solutions are represented by polylines.

Figure 1.4 presents the energy-delay tradeoff in terms of different architectures, i.e., different combinations of cluster size N and LUT size k, for three FPGA classes: Class0, Class1, Class2 and Class3. Similar to the baseline FPGA Class0, the min-delay architecture is $(N = 8, k = 7)$ for both Class1 and Class2. The min-energy architecture is $(N = 8, k = 4)$ for Class1 and

$(N = 12, k = 4)$ for Class2. This shows that LUT size 7 gives the best performance and LUT size 4 leads to the lowest energy consumption for these Vdd-programmable FPGAs. FPGA Class1 and Class2 reduce energy-delay, ED, product by 25.97% and 54.39%, respectively. Class3 achieves better energy-delay tradeoff than architecture Class1, and is even better than Class2. This is because FPGA Class3 removes the level converters in routing channels, but still can reduce interconnect dynamic energy. This reduction is not available in architecture Class2 which uses Vdd-gateable interconnect switches.

Considering area, energy and delay tradeoff, the best architecture class in [5, 15] is Class2, which uses Vdd-programmable logic blocks and Vdd-gateable interconnects It reduces the energy-delay product over the MCNC benchmark set by 54.39% with 17% area increase and 3% more configuration SRAM cells compared to the baseline architecture class using high-Vdd for both logic blocks and interconnects.

1.1.4 Device and Architecture Concurrent Optimization. Vdd programmability introduces area overhead compared to single-Vdd architecture, e.g. 17% area overhead with the best architecture in Section 1.1.3 [5, 15]. Meanwhile, device optimization considering supply voltage Vdd and threshold voltage Vt has little chip area increase, but a great impact on power and performance in the nanometer technology. The authors in [19, 20] studied simultaneous evaluation of device and architecture optimization for FPGAs. As shown in Figure 1.5, an efficient yet accurate timing and power evaluation method, called trace-based model, was developed. By collecting trace infor-

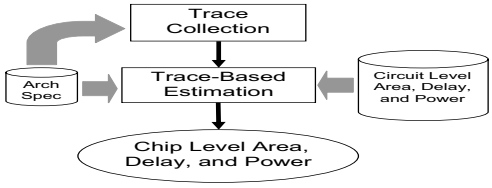


Figure 1.5. The novel trace-based evaluation flow.

mation from cycle-accurate simulation of placed and routed FPGA benchmark circuits and re-using the trace for different Vdd and Vt, device and architecture co-optimization considering hundreds of device and architecture combinations was enabled. Compared to the baseline FPGA architecture, which uses the VPR architecture model and the same LUT and cluster sizes as those used by the Xilinx Virtex-II, Vdd suggested by ITRS, and Vt optimized with respect to the above architecture and Vdd, architecture and device co-optimization can reduce energy-delay product by 20.5% and chip area by 23.3%. Furthermore, considering power-gating of unused logic blocks and interconnect switches (in this

case sleep transistor size is a parameter of device tuning), our co-optimization reduces energy-delay product by 55.0% and chip area by 8.2% compared to the baseline FPGA architecture.

1.1.5 Architecture Optimization for Glitch Power Minimization. It has been shown that the dynamic power contributes 62% amount of total power for a commercial 90nm FPGA [21] and 30.8% of the switching is due to glitch [22]. The authors in [22] described a technique that reduces dynamic power in FPGAs by reducing the number of glitches in the global routing resources. The technique involves adding programmable delay elements within the logic blocks of an FPGA to align the arrival times of early-arriving signals to the inputs of the lookup tables and to filter out glitches generated by earlier circuitry. On average, the proposed technique eliminates 91% of the glitching, which reduces overall FPGA power by 18%. The added circuitry increases overall area by 5% and critical-path delay by less than 1%. Furthermore, since it is applied after routing, the proposed technique requires no modifications to the existing FPGA routing architecture or CAD flow.

1.2 Process Variation Optimization

1.2.1 Device and Architecture Concurrent Optimization. Existing FPGA architecture evaluation has considered performance, area, and power [7, 23, 18, 4]. Section 1.1.3 [5] evaluated new FPGA architectures considering field programmable supply voltage including dual-Vdd and power-gating. Section 1.1.4 [19] showed that device and architecture co-optimization is able to obtain the largest improvement in FPGA performance and power efficiency. However, all these evaluation work did not consider process variations.

The authors in [24] proposed an analytical model for chip-level leakage power and delay considering within-die and die-to-die variations. Leveraging this analytical model, the authors in [24] also extended $Ptrace$ in [19] and performed device and architecture concurrent optimization considering process variation. Similar to [25], the leakage current of one circuit element is modeled as a lognormal variable. The chip-level leakage current is a sum of numerous lognormal variables and is modeled as another lognormal variable. The delay of one circuit element is modeled as a Gaussian variable. The chip-level delay involves maximum and addition operations between Gaussian variables, and is modeled as another Gaussian variable.

Figure 1.6 shows the full chip leakage power and delay variations in the presence of inter-die and intra-die variations. 10% of the nominal value is assumed as 3σ for all the process variation in gate length L_{eff}, oxide thickness T_{ox} and threshold voltage V_{th}. Leakage may change significantly due to process variations. When there is a $\pm 3\sigma$ inter-die variation of L_{eff}, the leakage power has a $3X$ span. When no inter-die variation is present, there is still a $2X$ span in

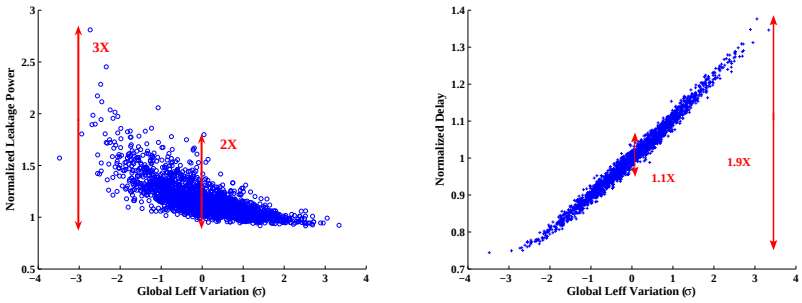


Figure 1.6. Chip-level Leakage power and delay variations of architecture (N=8, K=4) with ITRS 65nm device setting under intra-die and inter-die variations.

leakage power due to within-die variation in L_{eff} and inter-die variations in T_{ox} and V_{th}. It is important to consider the impact of process variations on leakage when determining the yield. On the other hand, there is a 1.9X span with $\pm 3\sigma$ inter-die L_g variation, and a $1.1X$ span without L_g variation. Clearly, delay is more sensitive to inter-die variation than within-die variation. This is because of the independence of local L_{eff} variation between each element. Therefore the effect of within-die L_{eff} variation tends to average out when the critical path is long enough.

Figure 1.7 presents the leakage and delay variation for the baseline case using Monte Carlo simulation with $Ptrace$. It can be seen that a smaller delay leads to a larger leakage in general. This is because of the inverse correlation between circuit delay and leakage. A device with short channel length has a small delay and consumes large leakage, which may lead to a high leakage. To calculate the leakage and delay combined yield, the cutoff leakage is set as the nominal leakage plus 30% that of the baseline, while the cutoff delay is $1.2X$ of each architecture's nominal delay.

It has been shown that heterogeneous-V_t and power-gating may have great impact on energy delay tradeoff [19]. The impact of heterogeneous-V_t and power-gating on the yield is further considered by comparing *Homo*-V_t (homogeneous Vt), *Hetero*-V_t (heterogeneous Vt) and *Homo*-V_t+*G* (homogeneous Vt with power-gating) architecture classes in min-ED device setting. Table 1.2 presents the combined yield for *Homo*-V_t with ITRS device setting and all classes with min-ED device setting. The area overhead introduced by power-gating is also presented in the table. Comparing *Homo*-V_t with ITRS device setting and min-ED device setting, the combined yield is improved by 21%. Comparing the classes using min-ED device setting, *Hetero*-V_t has a 3% higher yield than *Homo*-V_t due to heterogeneous-V_t while *Homo*-V_t+*G* has a 8% higher yield than *Homo*-V_t due to power-gating. *Homo*-V_t+*G* has the highest combined yield with an average of 16% area overhead. Device tuning and power-gating

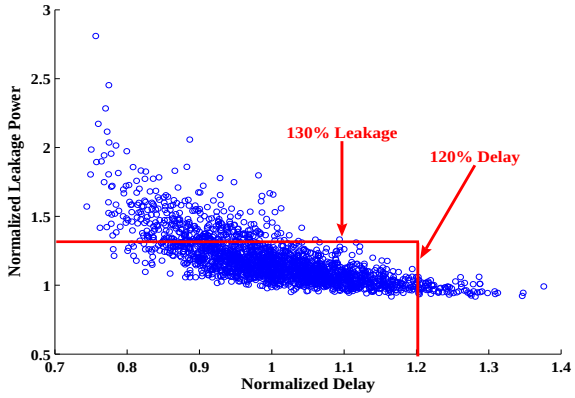


Figure 1.7. Leakage and delay of baseline architecture (N=8, K=4) with ITRS setting under process variations.

(N,K)	ITRS	Min-ED			
	Homo-V_t	Homo-V_t	Hetero-V_t	Homo-V_t+G	
	Y(%)	Y(%)	Y(%)	Y(%)	Area Inc(%)
(6,4)	71	83	83	86	18
(8,4)	67	81	81	86	14
(10,4)	65	81	81	86	17
(12,4)	48	77	81	87	20
(6,5)	79	85	84	90	14
(8,5)	55	81	86	89	15
(10,5)	55	81	86	89	19
(6,6)	49	77	82	88	15
(8,6)	49	75	80	88	16
(6,7)	45	73	77	86	10
Avg	58	79	82	87	16

Table 1.2. Combined Leakage-delay yield between FPGA Classes.

improve yield by 29% comparing *Homo-V_t+G* with min-ED setting to *Homo-V_t* with ITRS setting. This table also shows that architectures with LUT size 5 gives the highest yield within each class. This is because it has both a relatively high leakage yield as well as timing yield.

1.2.2 An Adaptive FPGA Architecture. It has been shown in Section 1.2.1 that process induced threshold voltage variations bring about fluctuations in circuit delay, that act the FPGA timing yield. The authors in [26] proposed an adaptive FPGA architecture that compensates for these fluctuations. The architecture includes an additional characterizer circuit that classi-

fies logic and routing blocks on each die according to their performance. Based on this classification, the architecture adaptively body-biases these resources by either speeding up the slow blocks or by slowing down the leaky ones. This procedure mitigates the effect of the variations and provides a better yield. We further diminish leakage by slowing down areas of the FPGA that have a positive slack. Overall, this architecture minimizes the timing variance of within-die and die-to-die Vth variations by up to 3.45X and reduces leakage power in the non-critical areas of the FPGA by 3X with no ect on frequency.

2. Power Aware FPGA Synthesis

In this section, we present power aware synthesis techniques for FPGAs in sub-micron technologies. Section 2.1 targets at the power reduction of the conventional FPGAs and presents a low-power CAD flow from technology mapping to routing, and the impact of pipelining on energy at system-level. We then presents the techniques for the emerging Vdd-programmable FPGAs in Section 2.2.

2.1 Synthesis for Conventional FPGAs

2.1.1 Power-Aware FPGA CAD Flow. The authors in [27] proposed power-aware FPGA CAD algorithms including technology mapping, clustering, placement and routing. The interaction between these CAD stages has also been studied in [27].

The power-aware technology mapping algorithm in [27] reduces power by minimizing the switching activity of the wires between LUTs. In FPGAs, these wires are implemented using routing tracks with significant capacitance; charging and discharging this capacitance consumes a significant amount of power. The capacitance of high activity wires between LUTs can be minimized during technology mapping by implementing LUTs that encapsulate high activity wires, thereby removing them from the netlist. Another power reduction technique adopted in [27] is to minimize the number of wires between LUTs. This is be achieved by minimizing node duplication while still keeping logic depth optimal. The authors in [28] further improved the technology mapping algorithm in [27] by considering glitch power reduction explicitly.

Traditional clustering goals include minimizing area and delay, and maximizing routability. Based on one of the representative clustering algorithm [7], the cost function in the power-aware clustering algorithm in [27] is extended to minimize the switching activity of connections between logic blocks by attracting high activity nets inside the logic blocks. This cost function favors the LUTs that share high activity nets with the packed LUTs in the current logic block.

After being packed, the clusters are placed and routed to physical locations on the FPGA. The power-aware placement in [27] tries to place those clusters connected by high-activity nets near each other. As a result, these high-activity nets will likely be short, and thus, consume less power. Unlike technology mapping, a placement algorithm cannot eliminate high-activity nets all together, it can only make these nets shorter. In cases when there are many high-activity nets, it may not be possible to place all clusters connected by these nets close together. The power-aware router in [27] tries to reduce power by using low capacitance resource node for nets with high switching activities. The authors in [29] also proposed a leakage-aware routing algorithm to encourage more frequent use of routing resources that have low leakage power consumptions.

The experimental results in [27] showed that the individual savings of the power-aware technology-mapping, clustering, placement, and routing algorithms were 7.6%, 12.6%, 3.0%, and 2.6% respectively. The majority of the overall savings were achieved during the technology mapping and clustering stages of the power aware FPGA CAD flow. In addition, the savings were mostly cumulative when the individual power-aware CAD algorithms were applied concurrently with an overall energy reduction of 22.6%.

Motivated by the fact that the leakage consumption depends on the state of gate inputs, the authors in [29] proposed an input vector configuration method to reduce active leakage after placement and routing. This leakage reduction technique leverages a fundamental property of basic FPGA LUTs that allows a logic signal in an FPGA design to be interchanged with its complemented form without any area or delay penalty. This property is applied to select polarities for logic signals so that FPGA hardware structures spend the majority of time in low-leakage states. The experimental results showed that leakage power can be reduced by 25% for a 90nm commercial FPGA.

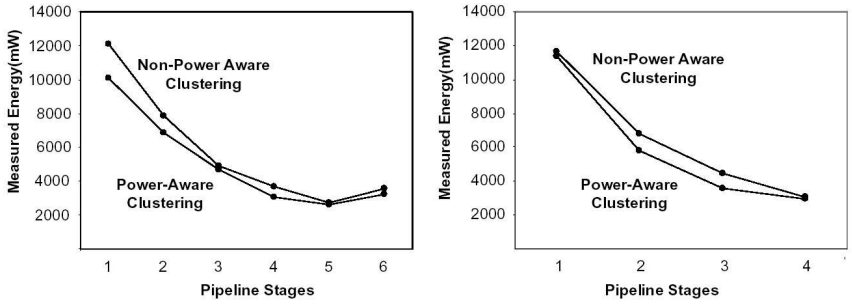
2.1.2 Impact of Pipelining. One of the simplest but effective ways of reducing the energy per operation of a circuit is pipelining. A highly pipelined circuit typically has fewer logic levels between registers, and may reduce glitches and therefore dynamic power compared to the unpipelined circuit. The author in [30] investigated experimentally the quantitative impact of pipelining on energy per operation for two representative FPGAs under 130nm and 180nm CMOS technologies.

For each circuit, several versions are created with a different degree of pipelining. For some circuits, pipeline stages are added by modifying the original hardware description code by hand. For other circuits, an automatic synthesis tool is used to generate circuits with differing degrees of pipelining. In all cases, the function of all versions of each circuit is the same, except for the additional latency imposed by pipeline stages. For each design, a version with a pipeline stage after every logic element is also created. Unlike all other

versions of the circuit, this version is likely to have a different behavior from the original circuit, since paths containing different numbers of logic elements may have different numbers of registers. The results from this version of each circuit may give an estimate of the best possible optimization achievable using pipelining.

For the 64-bit unsigned multiplier with the 130nm high performance FPGA, the difference in dynamic system energy between the most pipelined version and the least pipelined version is 81%. For the other benchmark circuits, this difference ranges from 40% to 82%. For the dynamic logic block energy, the difference is as high as 98%. In contrast, as shown in Section 2.1.1, lower-level physical design optimizations can typically reduce energy by up to 23% [27]. System-level optimizations such as pipelining can have a far more significant impact on the overall energy dissipation.

The "maximally pipelined" version of each benchmark circuit, in which a register is used at the output of every logic block, obtains the smallest energy compared to all the other versions. However, the difference in energy between the maximally pipelined version and the next-most pipelined version is small. This implies that there is little opportunity of reducing glitch energy further by increasing the number of pipeline stages in these circuits. The impact of pipelining on dynamic power reduction for a low-power 180nm FPGA is similar to that for the high-performance 130nm chip, although less pronounced.



a) 64-bit integer multiplier b) Cordic circuit

Figure 1.8. Power-aware and non-power aware clustering results.

The authors in [30] also investigated whether the effectiveness of lower-level tools is affected by pipelining at the system level. Clustering was focused, since it has been shown in Section 2.1.1 that clustering is more effective at reducing power than other low-level CAD stages [27]. Figure 1.8 shows the results for two designs. The horizontal axis on each graph is the number of pipeline stages,

and the vertical axis is the measured system (board) dynamic power, which is proportional to the energy per operation of the circuit. The top line in each graph represents the energy obtained when using the non-power-aware cluster algorithm, while the lower line represents the energy obtained when using the power-aware cluster algorithm. As the figures show, the improvements obtained by pipelining are much more significant than those obtained by making the cluster algorithm power-aware.

The figure also illustrates the interaction between the two optimization schemes: the reduction achieved by the cluster algorithm varies as the degree of pipelining changes. In general, for the array multiplier, the reduction achieved by the cluster algorithm decreases as the degree of pipelining increases. For both circuits, the power-aware cluster algorithm is ineffective at reducing power for the most heavily pipelined variants, since it has fewer high activity nets to work with.

These results are significant. They indicate that for most circuits, it does make sense to optimize for power both at the system level as well as during low-level synthesis and physical design. The results also show, however, that it is not reasonable for a designer to rely on pipeline-aware synthesis and physical design. Incorrect system-level design decisions, such as a bad choice for pipelining depth, can not be "made up for" by low-level CAD tools.

2.2 Synthesis for Vdd-Programmable FPGAs

Vdd-programmability has been proposed in [10, 11] to reduce FPGA logic block power and extended to interconnects in [12, 13]. In Section 1.1.1, we have presented FPGA Vdd-programmable circuits and architecture optimization. In this section, we present the synthesis algorithms including technology mapping, clustering, placement and post-layout physical synthesis for Vdd-programmable FPGAs.

2.2.1 Dual-Vdd Technology Mapping and Clustering. The authors in [31] studied the technology mapping problem of FPGA architectures with dual supply voltages for power optimization. This was done with the guarantee that the optimal mapping depth of the circuit with single-Vdd is not increased with dual-Vdd. Low Vdd is used for the LUTs on the non-critical paths. The experimental results in [31] showed that the dual-Vdd mapping algorithm can improve power savings by up to 11.6% over the single-Vdd mapper.

Similarly, the authors in [32] proposed delay optimal FPGA clustering algorithm with Vdd-programmability. The clustering procedure guarantees the optimal delay of the circuit obtained under the general delay model. In the meantime, logic blocks on the non-critical paths can be driven by low-Vdd (VddL) to save power. Experimental results showed that our clustering algorithm can

achieve power savings by 20.3% on average compared to the clustering result for an FPGA with a single high-Vdd (VddH).

2.2.2 Region-Constrained Placement with Power-Gating. The authors in [8] proposed region-constraint placement (RCP) to reduce leakage energy for FPGAs with power-gating capability. There are 4 to 256 basic logic elements (or logic slices) in each region. Each unused region can be power-gated with power transistor inserted to reduce leakage. A larger region may have smaller area overhead due to the inserted power transistor.

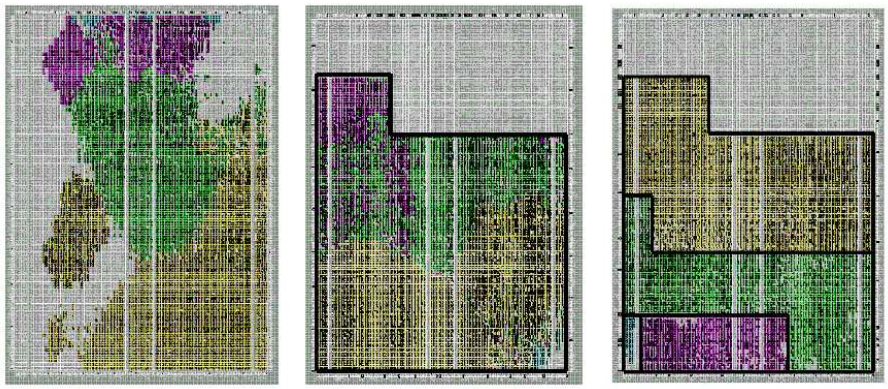


Figure 1.9. Different placements for an example design. In part (c), each module is bounded by a polygon.

The traditional placement due to lack of region concept tends to scatter the utilized slices across different regions (See Figure 1.9 (a)). Since the regions with partially used slices cannot be power-gated, the potential for leakage energy savings reduces. On the other hand, RCP takes into account the region concept explicitly. The basic principle of RCP is to constrain the placement of the design to specific regions of the FPGA (See Figure 1.9 (b)) and leave some regions of the FPGA completely unused, so that they can be power gated. This in turn helps to maximize the potential leakage savings. Experimental results in [8] showed that RCP reduces leakage energy by 11% on average with the region size of 256 slices while the normal placement only reduces leakage energy by 2%.

RCP is essentially a static technique where the unused FPGA space (regions) can be shut off at configuration time (before the execution). While it is easy to implement, it may not be as effective in designs that occupy large portion of the FPGA space (which in turn limits the potential leakage savings). However, for

the designs with modules that remain inactive over significant durations of time, a time-based control scheme can be applied, which reduces leakage even in the utilized portions of the FPGA by switching off/on the power supply, exploiting the idleness in portions of the design. Specifically, the time-based scheme turns off power supply to all regions containing only idle modules. The authors in [8] investigated the time-based control scheme with RCP. The module-level RCP places each module of the design that exhibits a distinct idleness profile using RCP individually, and turns off power supply to all regions containing only idle modules.

Figure 1.9 (c) shows an example design placement using module-level RCP for time-based leakage control. Modules of the design get placed on non-overlapping regions, thus maximizing the number of regions that can be dynamically switched-off. Note that this slightly decreases the statically unused portion on the FPGA (because in order to ensure the inter-module region exclusivity needed for module-level RCP, some regions can only be partially filled). The experiments in [8] showed a significant increase in leakage savings due to module-level RCP. Assuming that reconfiguration incurs a 10% increase in overall execution time and consequent leakage energy penalty, module- level RCP with time-based leakage control provides 19% (is 27% without reconfiguration overhead) more leakage savings compared to RCP.

2.2.3 Post-layout Synthesis. After placement and routing, Vdd-level assignment is performed to select Vdd-level for Vdd-programmable FPGAs. A straight-forward greedy assignment based on power sensitivity was proposed for logic block Vdd-level selection in [10, 11]. Before the assignment, all logic blocks use high-Vdd. In the iterative assignment, the logic block with the largest power sensitivity is assigned to low-Vdd. If the new critical path delay exceeds the user-specified delay increase bound, low-Vdd assignment is reversed. Otherwise, this assignment is accepted. In either case, the logic block selected in this iteration may not be re-visited in other iterations.

A similar assignment algorithm was proposed in [12] for Vdd-programmable interconnects, where a Vdd-level converter is inserted in front of each interconnect switch. However, it has been shown that these Vdd-level converters consume a large amount of leakage [5, 15, 33, 34]. As shown in Figure 1.10, the authors in [33, 34] proposed two ways to avoid using level converters in interconnects, *tree based level converter insertion (TLC)* and *dual-Vdd tree based level converter insertion (dTLC)*. TLC enforces that there is only one Vdd-level within each routing tree while dTLC can have different Vdd-levels within a routing tree, but no VddL switch drives VddH switches. Dual-Vdd assignment algorithms considering chip-level time slack allocation were developed for maximum power reduction. These algorithms included *TLC-S* and *dTLC-S*, two power sensitivity based algorithms with implicit time slack allocation and

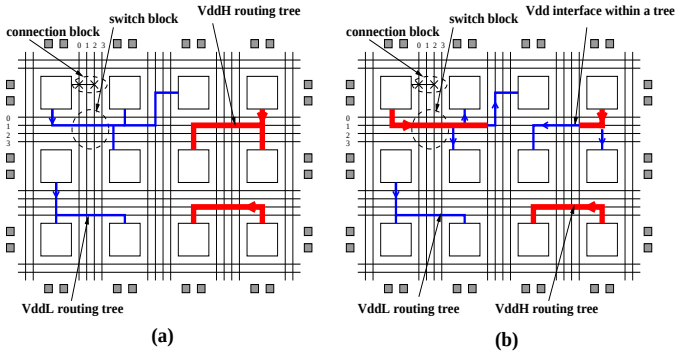


Figure 1.10. (a) Tree based level converter insertion (TLC); (b) Dual-Vdd tree based level converter insertion (dTLC).

dTLC-LP, a linear programming (LP) based algorithm with explicit time slack allocation. All allocated time slack first to interconnects with higher power sensitivity and assign low-Vdd to them for more power reduction. Experiments in [33] showed that dTLC-LP obtains the lowest power consumption. Compared to dTLC-LP, dTLC-S obtains slightly higher power consumption but runs $3X$ faster. Compared to the existing *segment-based level converter insertion (SLC)* for dual-Vdd, dTLC-LP reduces interconnect power by 52.90% without performance loss for the MCNC benchmark circuits. The authors in [35] further extended the LP based algorithm dTLC-LP to EdTLC-LP for the interconnects with wire segments of mixed lengths.

However, it takes a long time to solve the LP problem for time slack allocation. The authors in [36] reformulated the LP based problem to min-cost network flow based problem for runtime reduction. The algorithm, EdTLC-NW, was proposed in [36]. EdTLC-NW achieves as good results as EdTLC-LP but runs $8X$ faster on average. Furthermore, the speedup increases for larger circuits and EdTLC-NW is $20X$ faster for the largest circuit.

The authors in [35] also presented a Mixed Integer and Linear Programming (MILP) based approach for simultaneous retiming and slack budgeting (SRSB) to further reduce interconnect power for FPGAs. The motivation of this idea is illustrated in Figure 1.11, where circuits in (A), (B) and (C) have the same clock period of 4 units. To change a buffer from VddH to VddL, one needs a slack of 2 units. If circuit (A) is decomposed to its combinational sub-components (B) and dual-Vdd budgeting is performed, no extra buffer can be powered by VddL. On the other hand, one extra buffer can be powered by VddL in (C), which can be obtained from (A) by retiming under the same clock period. Therefore, SRSB is able to reduce more power than slack budgeting alone in [34]. To minimize the distortion of the placement and routing, the placement and flip-flop (FF) binding constraints are considered during the retiming process.

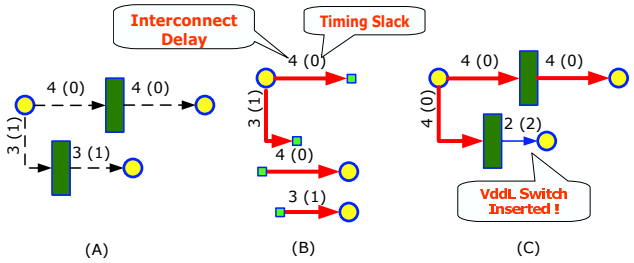


Figure 1.11. Comparison of effectiveness for power reduction by sequential approach and simultaneous approach. (A) original circuit, (B) sequential approach (retiming followed by budgeting), (C) simultaneous retiming and dual-Vdd budgeting. The number outside (inside) the brackets associated in each edge denotes the interconnect delay (timing slack) of that edge, respectively. The thick, red lines are interconnects driven by VddH while the thin, blue lines are interconnects driven by VddL.

Compared to the sequential approach (min-clock retiming followed by dual-Vdd budgeting), experimental results in [35] showed that the MILP based SRSB approach achieves 7.7% and 3.8% interconnect power reduction on average for MCNC and industrial circuits, respectively.

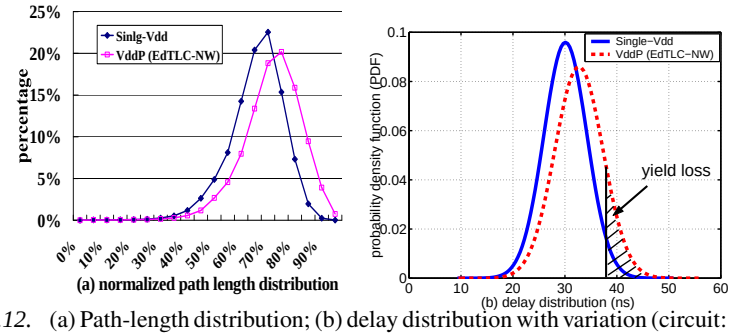


Figure 1.12. (a) Path-length distribution; (b) delay distribution with variation (circuit: s38417).

All the above Vdd-level assignment algorithms are deterministic. However, these deterministic Vdd assignment algorithms leverage timing slack exhaustively and significantly increases the number of near-critical paths, which results in a degraded timing yield with process variation. As shown in Figure 1.12, the usage of programmable Vdd increases the percentage of near-critical paths (with delay larger than 90% critical path delay) from 0.26% to 4.7%, which degrades the timing yield from 97.7% to 89.2%. The authors in [37] presented two statistical Vdd assignment algorithms. The first greedy algorithm is based on sensitivity while the second one is based on timing slack budgeting. Both minimize chip-level interconnect power without degrading timing yield. Evaluated with MCNC circuits, the statistical algorithms reduce interconnect power

by 40% compared to the single-Vdd FPGA with power gating. In contrast, the deterministic algorithm reduces interconnect power by 51% but degrades timing yield from 97.7% to 87.5%.

3. Variation Aware FPGA Synthesis

It has been shown that the FPGA chip-level variations in leakage and delay could be 3X and 1.9X respectively in Figure 1.6. We have presented architecture level optimization to combat with process variation in Section 1.2. In this section, we present variation aware FPGA synthesis algorithms. These algorithms could be categorized into two main groups, stochastic and chipwise synthesis algorithms. The stochastic algorithms apply the same synthesis results across different chips and optimize timing statistically considering process variation. On the other hand, the chipwise algorithms apply different synthesis results and optimize timing for each chip or each bin of chips individually. We present the stochastic and chipwise synthesis in Section 3.1 and Section 3.2, respectively.

3.1 Stochastic Synthesis

Process variation and pre-routing interconnect delay uncertainty affect timing and power for modern VLSI designs in nanometer technologies. The authors in [38] presented the first in-depth study on stochastic physical synthesis algorithms leveraging statistical static timing analysis (SSTA) with process variation and pre-routing interconnect delay uncertainty for FPGAs Interconnect delay is modeled as a Gaussian random variable considering interconnect uncertainty and process variation. SSTA instead of static timing analysis (STA) is performed to analyze statistical criticality, i.e. the probability for each timing edge/node to be timing critical with variation. Statistical criticality is then leveraged for statistical timing optimization.

3.1.1 Stochastic Clustering. The stochastic clustering, *ST-VPack* in [38] is extended based on the deterministic timing driven clustering, T-VPack [7]. The deterministic timing model with constant interconnect delay used in T-VPack leads to some inaccuracy in estimation of where the critical path lies. T-VPack may try to shorten a path which is not part of the post-routing critical path due to this inaccurate estimation. Furthermore, any near-critical paths may become critical considering process variation. ST-VPack, leverages a statistical timing model and optimizes timing statistically. It has been shown in [38] that the timing gain in stochastic clustering is mainly due to modeling pre-routing interconnect uncertainty instead of process variation. This can be explained by Figure 1.13, which compares the probability density functions (PDF) for post-routing delay normalized with respect to the estimated one during clustering and post-routing delay with process variation normalized with respect to the

nominal one. The statistics are based on all global interconnects of all designs. Clearly, interconnect uncertainty leads to a more significant delay variance in clustering stage.

On average, ST-VPack reduces the nominal, mean and standard deviation of delay by 3.7%, 5.0%, and 6.4%, respectively. The impact of ST-VPack on timing distribution (mean and standard deviation) is larger than that on the nominal delay due to the fact that we are aiming to optimize timing statistically for ST-VPack. In addition, ST-VPack does not have area, post-routing wire length and runtime overhead compared to T-VPack. Clearly, ST-VPack is able to effectively improve timing statistically and therefore improve yield without routability, area and runtime overhead compared to T-VPack.

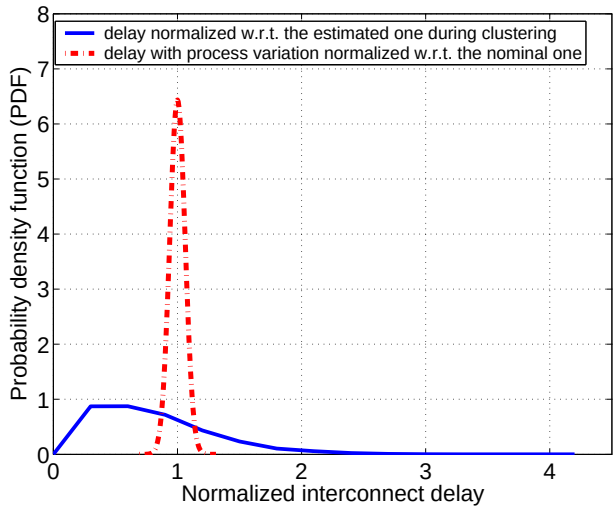


Figure 1.13. Comparison between probability density functions (PDF) for (i) post-routing delay normalized with respect to (w.r.t.) the estimated delay during clustering, and (ii) post-routing delay with process variation normalized w.r.t. the nominal one.

3.1.2 Stochastic Placement. After packing, clusters are placed to physical locations on the FPGA chip. For FPGAs, the typical placement algorithm is simulated annealing as in the timing-driven algorithm, *T-VPlace* [39], in VPR [7]. The authors in [40, 38] extended T-VPlace to the stochastic placement *ST-VPlace* for process variation and interconnect uncertainty. The interconnect delay estimated in T-VPlace is based on 2-pin net routing for each pair of locations without considering congestion. The actual delay after routing may differ from the estimated delay in placement, mainly due to the impact of congestion and multi-terminal nets. This introduces interconnect delay uncertainty in placement. In addition, any near-critical paths may become critical with process

variation. Figure 1.14 compares the PDFs for post-routing delay normalized with respect to the estimated one during placement and post-routing delay with process variation normalized with respect to the nominal one. The statistics are based on near-critical interconnects (static criticality greater than 0.9 after routing) of all designs. As shown in this figure, process variation leads to a much wider delay spread compared to interconnect uncertainty. The statistics show that more than 70% of interconnects have an estimation error within 1% while the relative standard deviation is 6% due to process variation. It is clear that process variation leads to a more significant delay variance and needs to be considered in placement. Interconnect uncertainty can be modeled as an independent Gaussian with a small relative standard deviation with respect to the estimated delay. Such a small relative standard deviation, e.g. 0.5%, however has little impact on the timing.

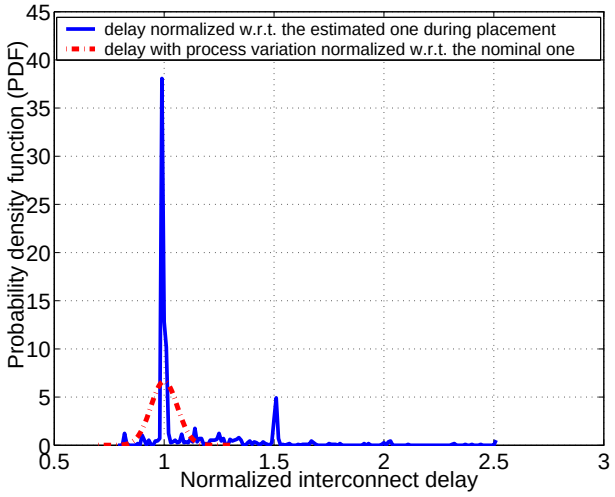


Figure 1.14. Comparison between probability density functions (PDF) for (i) post-routing delay normalized w.r.t. the estimated delay during placement, and (ii) post-routing delay with process variation normalized w.r.t. the nominal one.

On average, ST-VPlace reduces the nominal, mean and standard deviation of delay by 3.3%, 4.0%, and 6.1%, respectively. Similar to the stochastic clustering ST-VPack, ST-VPlace has larger impact on timing distribution than that on the nominal delay due to its statistical fashion. The impact of ST-VPlace on timing is similar to that of ST-VPack. Nevertheless, the gain of ST-VPlace mainly comes from considering process variation, different from ST-VPack where the gain is mainly due to modeling interconnect uncertainty. On the other hand, ST-VPlace increases the total wire length after routing by 1.3% and takes $3.1X$ runtime compared to T-VPlace.

3.1.3 Stochastic Routing. After clusters are placed to physical locations on the FPGA, routing is performed to determine which programmable interconnect switches should be turned on to connect required interconnects. The stochastic routing algorithm, *ST-PathFinder* in [38, 41] is extended based on the deterministic routing algorithm, *PathFinder* [42, 7]. In the routing stage, the interconnect estimation occurs when predicting delay from the current partial routing to the target sink, and has the highest accuracy within all design stages. On the other hand, timing analysis is only performed after an entire routing iteration. Interconnect uncertainty has little impact on one of the key parameters, static criticality, in PathFinder. Therefore ST-PathFinder only considers process variation in SSTA.

On average, ST-PathFinder reduces the nominal, mean and standard deviation of delay by 1.4%, 1.4% , and 0.7%, respectively. Compared to clustering and placement stages, the impact of stochastic routing on timing is much smaller due to the fact that routing stage has the smallest design flexibility. In addition, ST-PathFinder reduces the total wire length after routing and overall runtime by 4.5% and 4.2%, respectively. Although SSTA is more expensive than STA, SSTA or STA is only performed once after one entire routing iteration. ST-PathFinder reduces the average number of routing iterations required for a successful routing from 22 to 15 compared to PathFinder and therefore consumes less runtime. It is due to the fact that ST-PathFinder uses statistical criticality to achieve a better balance between weights of timing and wire lengths in the cost function for each net. Besides of a 4.2% of runtime reduction, ST-PathFinder also reduces the total wire length after routing by 4.5% due to this better balanced cost function.

3.1.4 Interaction Between Clustering, Placement and Routing. In addition, the authors in [38] studied the interaction between each individual design stage. Table 1.3 presents the results including the nominal, mean and standard deviation of circuit delay, runtime and average yield improvement for all combinations of algorithms in The delay values are presented in the difference compared to the flow consisting of all deterministic algorithms. The runtime is also normalized with respect to the deterministic flow. The yield improvement is calculated as the difference between the yields obtained by the deterministic flow and each stochastic flow when each stochastic flow achieves a yield of 90% or 95%.

When applying stochastic clustering and placement concurrently, we can achieve a smaller nominal, mean and standard deviation of delay than applying any one of them alone. However, there exists some overlap between gains in clustering and placement. On the other hand, the routing stage has less improvement. Compared to the deterministic flow, the stochastic clustering and placement increase total wire length by 0.8% and 1.3%, respectively. When

clustering	D	S	D	D	S	S	D	S
placement	D	D	S	D	S	D	S	S
routing	D	D	D	S	D	S	S	S
Tnom(ns)	21.2	-3.7%	-3.3%	-1.4%	-6.4%	-4.1%	-3.6%	-6.3%
Tmean(ns)	22.9	-5.0%	-4.0%	-1.4%	-5.9%	-4.7%	-4.0%	-6.2%
Tsigma(ns)	3.4	-6.4%	-6.1%	-0.7%	-8.8%	-6.1%	-6.3%	-7.5%
runtime	1.0X	0.99X	3.1X	0.96X	3.0X	0.97X	3.1X	3.0X
wire length	27610	0.8%	1.3%	-4.5%	3.2%	-3.4%	-3.4%	-1.6%
average yield improvement								
90% yield	-	9.9%	8.0%	2.3%	12.3%	9.3%	8.7%	12.6%
95% yield	-	7.0%	5.6%	1.5%	8.9%	6.5%	6.2%	9.1%

Table 1.3. Combined results. 'D' and 'S' stand for deterministic and stochastic, respectively (based on the results of 20 MCNC designs).

applying stochastic clustering and placement concurrently with deterministic routing, the wire length overhead increases to 3.2%. On the other hand, the stochastic routing reduces wire length by 4.5%. When applying stochastic routing with stochastic placement, clustering or both concurrently, the wire length is reduced by 3.2%, 3.4% and 1.6% respectively compared to the deterministic flow.

When all stochastic algorithms are applied concurrently, the stochastic flow reduces the nominal, mean and standard deviation of delay by 6.3%, 6.2% and 7.5% respectively but takes $3.0X$ runtime compared to the deterministic flow. In addition, the stochastic flow can improve the yield by 12.6% (or 9.1%) when a yield of 90% (or 95%) is obtained by the stochastic flow. For a good gain with less runtime, only stochastic clustering with deterministic placement and routing may be applied. This flow reduces the nominal, mean and standard deviation of delay by 3.7%, 5.0% and 6.4% respectively, and reduces runtime slightly.

3.2 Chipwise Synthesis

The programmability of FPGAs offers a unique opportunity to leverage process variation and improve circuit performance. For custom ICs, physical design for a targeted circuit must be the same for all chips. For FPGAs, however, we can potentially place (and route) each pre-fabricated FPGA chip differently for the same application. Compared to manufacturing level process control, this chipwise physical design technique only involves post-silicon design optimization and is more cost-effective.

The authors in [43] considered placement and proposed the following variation aware chipwise design flow (see Fig. 1.15). For a given set of FPGA chips, the variation map for each chip is first generated, which may be obtained by syn-

thesizing test circuits for each chip. Based on the variation map, the potential delay improvement of chipwise placement is then estimated. If the improvement is large, it is worthwhile to perform placement for each chip; otherwise, the conventional design flow is adopted, which uses the same placement and route for all chips. A similar approach leveraging multiple routing configurations has been presented in [44]. For each application, multiple configurations are first generated. Each FPGA chip is then tested using each individual configuration. The best implementation is then selected for each particular chip.

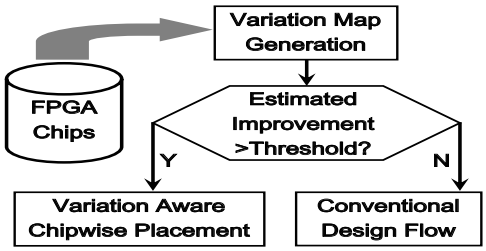


Figure 1.15. Design flow of variation aware chipwise placement.

The authors in [43] first proposed an efficient high-level trace-based estimation method to evaluate the potential performance gain achievable through chipwise FPGA placement without detailed placement. Such estimation may provide a lower bound of the performance gain of detailed placement. The authors in [43] then developed a variation-aware detailed placement algorithm $vaPL$ within the VPR framework [7] to leverage process variation and optimize performance for each chip. Chipwise placement $vaPL$ is deterministic for each chip when the variation map of the chip is known, and leads to different placements for different chips of the same application.

Table 1.4 compares the circuit performance between $vaPL$ and the deterministic placement T-VPlace (or $dtPL$) [7] for all benchmarks. The chip size for each benchmark is decided so that the utilization rate is about 90%. For the chips obtained from $dtPL$, the 3-sigma timing, according to the existing practice for FPGA designs without considering process variation, is obtained by taking the worst-case delay for all circuit elements on the critical path. Results from this approach is reported under column 3 in Table 1.4. This approach is apparently too pessimistic, as it is very unlikely for all circuit elements on the critical paths to have the worst-case delay at the same time because of the spatial correlation. To reduce pessimism and consider correlated process variation, the true 3-sigma timing as a measure of chip performance can be used. The true 3-sigma timing can be approximated by using the maximum delay among all tested variation maps. The approximated true 3-sigma timing for chips from both $dtPL$ and $vaPL$ are reported under columns 4 and 5 in Table 1.4.

Comparing columns 3 and 4 in Table 1.4, the worst-case timing is indeed too pessimistic compared to the true 3-sigma one, and the relative pessimism reduction by using the true 3-sigma timing is about 49.5% on average. Comparing columns 4 and 5, placement results from $vaPL$ are always better than those from $dtPL$ even when both use the true 3-sigma performance as a metric. The performance improvement for $vaPL$ is up to 10%, or 5.3% on average.

1	2	3	4	5
Bench-	Chip	$dtPL$ (ns)		$vaPL$ (ns)
mark	size	WC	3-sigma	3-sigma
alu4	13×13	35.8	19.0 (-46.9%)	18.4 (-3.2%)
apex2	15×15	43.8	24.3 (-44.5%)	21.9 (-9.9%)
apex4	12×12	35.3	19.7 (-44.3%)	17.4 (-11.6%)
clma	37×37	79.0	41.4 (-47.6%)	39.7 (-4.1%)
diffeq	14×14	54.3	24.5 (-55.0%)	23.6 (-3.7%)
elliptic	21×21	67.5	34.5 (-48.8%)	32.9 (-4.6%)
ex5p	12×12	37.4	20.8 (-44.4%)	19.9 (-4.3%)
misex3	13×13	35.9	19.8 (-44.9%)	19.4 (-2.0%)
s298	16×16	80.5	41.3 (-48.7%)	39.5 (-4.4%)
s38584.1	27×27	41.3	21.6 (-47.8%)	20.6 (-4.6%)
seq	15×15	33.1	18.5 (-44.3%)	18.0 (-2.7%)
spla	20×20	50.0	28.4 (-43.3%)	26.0 (-8.5%)
average	-	49.5	26.2 (-46.7%)	24.8 (-5.3%)

Table 1.4. Performance comparison between $vaPL$ and $dtPL$.

References

[1] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," in *Proc. Intl. Symp. Low Power Electronics and Design*, pp. 155–160, August 1998.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Design Automation Conf.*, June 2003.

[3] K. Poon, A. Yan, and S. Wilton, "A flexible power model for FPGAs," in *Proc. of 12th International conference on Field-Programmable Logic and Applications*, Sep 2002.

[4] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb 2003.

[5] Y. Lin, F. Li, and L. He, "Power modeling and architecture evaluation for FPGA with novel circuits for vdd programmability," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2005.

[6] F. Li, Y. Lin, L. He, D. Chen, and J. Congs, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 1712–1724, Nov. 2005.

[7] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Feb 1999.

[8] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2004.

[9] Y. Lin, F. Li, and L. He, "Routing track duplication with fine-grained power-gating for FPGA interconnect power reduction," in *Proc. Asia South Pacific Design Automation Conf.*, Jan 2005.

[10] F. Li, Y. Lin, L. He, and J. Cong, "Low-power FPGA using pre-defined dual-vdd/dual-vt fabrics," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2004.

[11] F. Li, Y. Lin, and L. He, "FPGA power reduction using configurable dual-vdd," in *Proc. Design Automation Conf.*, June 2004.

[12] Fei Li and Yan Lin and Lei He, "Vdd programmability to reduce FPGA interconnect power," in *Proc. Intl. Conf. Computer-Aided Design*, November 2004.

[13] Jason H. Anderson and Farid N. Najm, "Low-power programmable routing circuitry for FPGAs," in *Proc. Intl. Conf. Computer-Aided Design*, November 2004.

[14] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "A dual-vdd low power FPGA architecture," in *Proc. Intl. Conf. Field-Programmable Logic and its Application*, August 2004.

[15] F. L. Yan Lin and L. He, "Circuits and architectures for field programmable gate array with configurable supply voltage," *IEEE Trans. VLSI Syst.*, 2005.

[16] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, 1990.

[17] S. Singh, J. Rose, P. Chow, and D. Lewis, "The effect of logic block architecture on FPGA performance," *IEEE Journal of Solid-State Circuits*, 1992.

[18] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 3–12, Feb 2000.

[19] L. Cheng, P. Wong, F. Li, Y. Lin, and L. He, "Device and architecture co-optimization for FPGA power reduction," in *Proc. Design Automation Conf.*, June 2005.

[20] L. Cheng, F. Li, Y. Lin, P. Wong, and L. He, "Device and architecture cooptimization for fpga power reduction," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1211–1221, July 2007.

[21] T. Tuan and et al, "A 90nm low-power FPGA for battery-powered application," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2006.

[22] J. Lamoureux, G. Lemieux, and S. Wilton, "GlitchLess: An active glitch minimization Techniques for FPGAs," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2007.

[23] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb 1999.

[24] H.-Y. Wong, L. Cheng, Y. Lin, and L. He, "FPGA device and architecture evaluation considering process variations," in *Proc. Intl. Conf. Computer-Aided Design*, Nov 2005.

[25] R. Rao, A. Devgan, D. Blaauw, and D. Sylvester, "Parametric yield estimation considering leakage variability," in *Proc. Design Automation Conf.*, June 2004.

[26] G. Nabaa, N. Azizi, and F. Najm, "An adaptive FPGA architecture with process variation compensation and reduced leakage," in *Proc. Design Automation Conf.*, June 2006.

[27] J. Lamoureux and S. J. Wilton, "On the interaction between power-aware FPGA CAD algorithms," in *Proc. Intl. Conf. Computer-Aided Design*, pp. 701–708, November 2003.

[28] L. Cheng, D. Chen, and D. Wong, "GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches," in *Proc. Design Automation Conf.*, June 2007.

[29] J. H. Anderson and F. N. Najm, "Active leakage power optimization for FPGAs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, March 2006.

[30] S. Wilton, S. Ang, and W. Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," in *Proc. Intl. Conf. Field-Programmable Logic and its Application*, August 2004.

[31] D. Chen, J. Cong, F. Li, and L. He, "Low Power Technology Mapping for FPGA Architectures with Dual Supply Voltages," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2004.

[32] D. Chen and J. Cong, "Delay optimal low-power circuit clustering for FPGAs with dual supply voltages," in *Proc. Intl. Symp. Low Power Electronics and Design*, August 2004.

[33] Y. Lin and L. He, "Leakage efficient chip-level dual-vdd assignment with time slack allocation for FPGA power reduction," in *Proc. Design Automation Conf.*, June 2005.

[34] Y. Lin and L. He, "Dual-vdd interconnect with chip-level time slack allocation for fpga power reduction," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, Oct. 2006.

[35] Y. Hu, Y. Lin, L. He, and T. Tuan, "Simultaneous time slack budgeting and retiming for dual-vdd FPGA power reducti on," in *Proc. Design Automation Conf.*, July 2006.

[36] Y. Lin, Y. Hu, L. He and V. Raghunat, "An efficient chip-level time slack allocation algorithm for dual-vdd FPGA power reduction," in *Proc. Intl. Symp. Low Power Electronics and Design*, October 2006.

[37] Y. Lin and L. He, "Statistical Dual-Vdd Assignment for FPGA Interconnect Power Reduction," in *Design Automation and Test in Europe*, pp. 636–641, April 2007.

[38] Y. Lin and L. He, "Stochastic Physical Synthesis for FPGAs with Prerouting Interconnect Uncertainty and Process Variation," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2007.

[39] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb 2000.

[40] Y. Lin, M. Hutton, and L. He, "Placement and timing for FPGAs considering variations," in *Proc. Intl. Conf. Field-Programmable Logic and its Application*, August 2006.

[41] S. Sivaswamy and K. Bazargan, "Variation-aware routing for FPGAs," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, February 2007.

[42] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb 1995.

[43] L. Cheng, J. Xiong, L. He, and M. Hutton, "FPGA performance optimization via chipwise placement considering process variations," in *International Conference on Field-Programmable Logic and Applications*, August 2006.

[44] Y. Matsumoto and et al, "Performance and yield enhancement of fpgas with with-in die variation using multiple configurations," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb. 2007.