TPR -- User Manual (Version 1.0 alpha)

Cristinel Ababei (*ababei@ece.umn.edu*)

June 19, 2004

1 Overview

TPR (Three-dimensional Place and Route for FPGAs [6]) is an FPGA placement and routing tool for 3D FPGA architectures. It was developed with the idea of exploring potential benefits, such as better performance, less routing resources, smaller total and average-net wirelength, area, and power, which 3D architecture can offer. Because the whole infrastructure of TPR is based on VPR, the reader is assumed to be very familiar with VPR [1] [2] as well as with T-VPack [4] in all regards.

Invoking TPR is as follows (see Section 4 for a concrete example):

```
tpr netlist.net architecture.arch placement.p routing.r [-options]
```

Netlist.net is the netlist describing the circuit to be placed and/or routed, while architecture.arch describes the architecture of the FPGA in which the circuit is to be implemented. The final placement will be written to placement.p. If TPR has to route a previously placed circuit, the placement is read from placement.p. The final routing is written to routing.r. The format of each of these files is basically along the same line as the VPR formats.

In its default mode, TPR places a circuit on a 3D FPGA (which in the particular case of *num_layers = 1*, the placement becomes 2D) and then repeatedly attempts to route it in order to find the minimum number of tracks required by the specified FPGA architecture to route this circuit. If a routing is unsuccessful, TPR increases the number of tracks in each *xchan*, *ychan* routing channels and tries again. If the number of tracks is increased three times in a row (this number can be changed internally) and the routing is still unsuccessful, the number of vias in each vertical channel is incremented and the routing is restarted. In this way TPR seeks the min number of vertical vias per *zchan* and the min number of horizontal tracks per *x/ychan*, which lead to successful routing of the placed circuit.

Once the minimum number of tracks (and vias) required to route the circuit is found, TPR exits. The other mode of TPR router is invoked when a user specifies a specific *x*, *y* channel width for routing. In this case, the circuit is placed and then routing is attempted only once, with the specified channel widths. Channel width for *xchan* and *ychan* is specified similarly to VPR. Channel width for *zchan* (i.e., number of vias coming vertically out of a switch box and going to another switch box in a different layer) is specified initially in the architecture file. If the circuit cannot be routed at the specified channel widths, TPR simply reports that it is unroutable.

TPR can currently perform detailed routing. The routing algorithm is breadth-first-search oriented. The placement algorithm first partitions the netlist into a number of partitions which equals the number of layers. The initial partitioning uses hMetis min-cut partitioner [3].

Typing TPR with no parameters will print out a list of all the available command line options/parameters.

2 Compiling TPR

TPR was developed in *C++* on a machine running Linux Pink Tie (Red Hat clone) but still has a lot of *C* flavor due to the fact that all(most) VPR code was (im)ported to and integrated into TPR. It successfully compiles with g++ compiler. Currently there is no sophisticated GUI supported.
In order to compile TPR you first edit the file *Makefile* to specify the path of your home directory and then type *make*.

3 Description of the TPR flow

The design flow (see Figure 1) starts with a technology-mapped netlist in .blif format, which can be generated using SIS. Then, the .blif netlist is converted into a netlist composed of more complex logic blocks with T-VPack [4]. The .net netlist as well as the architecture description file are the inputs to the placement algorithm. The placement algorithm first partitions the circuit into a number of balanced partitions equal to the number of layers for the 3D integration. The goal of this first min-cut partitioning is to minimize the connections between layers, which translates into minimum number of vertical (i.e., inter-layer) wires. Then it continues with the placement of each layer in a top-down fashion.

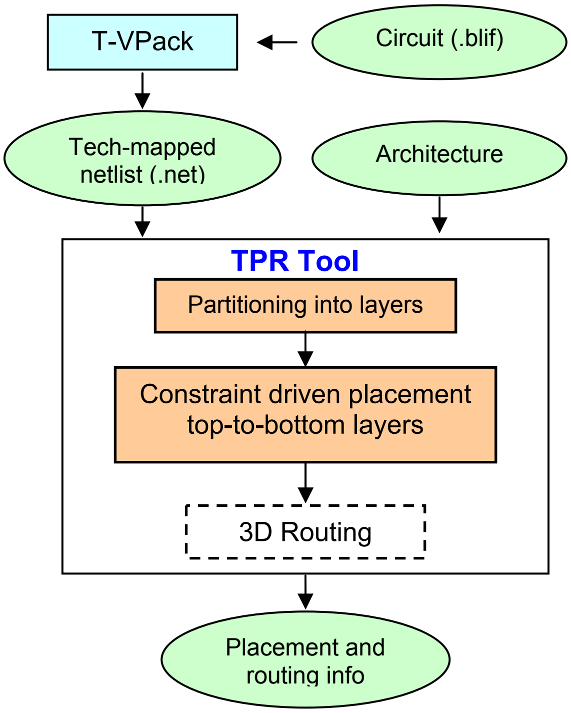
T-VPack
Circuit (.blif)
Tech-mapped netlist (.net)
Architecture
TPR Tool
Partitioning into layers
Constraint driven placement top-to-bottom layers
3D Routing
Placement and routing info

Figure 1 Flow diagram of the 3D placement and routing tool

The top layer is placed by unconstrained recursive partitioning. Then, the rest of the layers are placed in turn by recursive partitioning but constraint to reduce the delay on timing-critical nets: the terminals of the most critical nets, which span more layers, are placed on restricted placement-regions. The restricted placement-regions are defined by the projection onto the current layer of the bounding-boxes defined by the terminals placed already in the layers above. Hence, the 3D bounding-box of the critical nets is minimized. Finally, detailed routing is performed using the adapted 3D version of the breadth-first-search VPR routing algorithm. The placement algorithm has integrated a static timing analysis engine as well as a path enumeration algorithm [5]. The recursive partitioning of a given layer stops when each placement region has less than four blocks. Complete overlap removal is done using a greedy heuristic which moves non-critical blocks (i.e., not lying on critical paths) to the closest available empty locations.

4 Options of TPR

At this moment options are few. However, the whole infrastructure of VPR has been ported in and further modifications can be easily integrated (e.g., doing it all simulated annealing based; practically pure transformation of VPR to 3D, etc.). In fact, it was the initial scope of this project to build a flexible infrastructure, which shall allow research experimentation with hypothetical 3D FPGA architectures.

-route_only: Says that an existing placement (from the placement file specified on the command line) shall be routed only. Default set to off.

-place_only: Only place the circuit, without routing it. Default set to off.

-nx <int>: Number of columns in each layer of the FPGA logic array. Default: set to minimum required to fit circuit.

-ny <int>: Number of rows in each layer of the FPGA logic array. Default: set to minimum required to fit circuit.

-K <int>: Number of k-most critical paths in the circuit, which shall be enumerated and used for edge criticality computations. Default set to 100.

-vertical_delay_same <int>: Tells the place engine that all (vertical) vias have same delay irrespective of the number of spanned layers (when 1) or not (when 0; in this case a via has its delay depending on the number of spanned layers). Used only during placement. Default set to 1.

-vertical_uwl <int>: Number of times the vertical length of a via is greater than the unit length, which is the length between two adjacent CLBs (i.e., distance between two adjacent layers). Used only during placement. Default set to 1.

-num_layers <int>: Number of layers; defining the 3D architecture. Currently should be between 1-20, but this range can be changed internally. Default set to 2.

-route_chan_width <int>: Tells TPR to route the circuit with a certain *x*, *y* channel width. No binary search on channel capacity will be performed to find the minimum number of tracks required for routing - the router will simply report whether or not the circuit is routable with this channel width.

-bend_cost <float>: The cost of a bend. Larger numbers will lead to routes with fewer bends, at the cost of some increase in track count. Default set to 0 for detailed routing.

An example of how to run TPR on one of the netlists, which comes with VPR is as follows:

```
tpr TESTS/misex3.net TESTS/multi_06.arch placements/misex3.p routings/misex3.r -K 11
-vertical_delay_same 1 -vertical_uwl 1 -num_layers 3 -route_chan_width 20
```

5 File formats

File formats are inherited from VPR. The only change consists of the introduction of the third dimension, *z*, which is added to both .p and .r file formats. Also, CHANZ is one more type of a routing node (rr_node), which appears in .r format files.
The format of the architecture file is also preserved. The only changes are as follows.

chan_width_z uniform <int> Sets the number of vias (i.e., vias_per_zchan) inside a vertical *zchan* channel.

via frequency: <float> length: <int | longline> wire_switch: <int> opin_switch: <int> Frac_cb: <float> Frac_sb: <float> Rmetal: <float> Cmetal: <float>
Describes a type of via. You can specify as many types of vias as you like (as in the case of segments) -- just use one via line for each. The meaning of each value is as follows.

- *frequency:* The fraction (from 0 to 1) of tracks composed of this type of via. The sum of the frequency values for all the vias lines must be 1.
- *length:* Either the "number of layers spanned by each via – 1", or the keyword *longline*. Longline means vias of this type span all layers.
- *wire_switch:* The index of the switch type used by other wiring vias/segments to drive this type of via. That is, switches going *to* this via from other pieces of wiring will use this type of switch.
- *opin_switch:* Inherited from VPR. Not used because clbs and pads are not connected to vertical vias. Therefore always set to 0.
- *Frac_cb:* Inherited from VPR. Not used because clbs and pads are not connected to vertical vias. Therefore always set to 0.
- *Frac_sb:* Describes the internal population of the via for switch boxes (connections to other routing tracks). This number gives the fraction (from 0 to 1) of the length + 1 switch blocks which could exist along the via that do in fact exist. So, a via of length 7 that had a *Frac_sb* value of 0.5 would have 4 switch boxes along its length. Exactly which tracks a via connects to at each switch box is determined by the *switch_box_type* parameter.
- *Rmetal:* Resistance per unit length of this wiring track, in Ohms.
- *Cmetal:* Capacitance per unit length of this wiring track, in Farads.

For example, let's say an architecture file describes three types of vias (as it is the case of the "multi" architectures found in TESTS/ coming inside TPR's archive).

```
# # Vias lengths 1, 2, and long.
```

```
via frequency: 0.4 length: 1 wire_switch: 0 opin_switch: 0 Frac_cb: 0. \
        Frac_sb: 1 Rmetal: 4.16 Cmetal: 81e-15
via frequency: 0.3 length: 2 wire_switch: 0 opin_switch: 0 Frac_cb: 0. \
        Frac_sb: 1 Rmetal: 4.16 Cmetal: 81e-15
via frequency: 0.3 length: longline wire_switch: 0 opin_switch: 0 Frac_cb: 0. \
        Frac_sb: 1 Rmetal: 4.16 Cmetal: 81e-15
```

6 Switch box

The switch box is 3D and its coding is similar to the 2D switch box present in VPR. Please read our paper(s) to find out more about the 3D switch boxes.

7 References

[1] V. Betz and J. Rose, "VPR: A New Packing Placement and Routing Tool for FPGA Research", *Field-Programmable Logic App.*, 1997, pp. 213-222.

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

[3] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI domain", *Proc. ACM/IEEE DAC*, 1997, pp. 526-529.

[4] A. Marquardt, V. Betz, J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density", *Proc. ACM/IEEE FPGA*, 199, pp. 37-46.

[5] Y-C. Ju and R. A. Saleh, "Incremental Techniques for the Identification of Statically Sensitizable Critical Paths," *Proc. ACM/IEEE DAC*, 1991, pp. 541-546.

[6] C. Ababei, P. Maidee, and K. Bazargan, "Exploring Potential Benefits for 3D FPGA Integration", *Field Programmable Logic and its Applications (FPL)*, 2004.