CAD and Routing Architecture for Interposer-Based Multi-FPGA Systems

Andre Hahn Pereira
Computer and Digital Systems Engineering Department - Escola Politécnica
University of São Paulo
São Paulo, SP
Brazil
andre.hahn@usp.br

Vaughn Betz
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON
Canada
vaughn@eecg.utoronto.ca

ABSTRACT

Interposer-based multi-FPGA systems are composed of multiple FPGA dice connected through a silicon interposer. Such devices allow larger FPGA systems to be built than one monolithic die can accomodate and are now commercially available. An open question, however, is how efficient such systems are compared to a monolithic FPGA, as the number of signals passing between dice is reduced and the signal delay between dice is increased in an interposer system vs. a monolithic FPGA.

We create a new version of VPR to investigate the architecture of such systems, and show that by modifying the placement cost function to minimize the number of signals that must cross between dice we can reduce routing demand by 18% and delay by 2%. We also show that the signal count between dice and the signal delay between dice are key architecture parameters for interposer-based FPGA systems. We find that if an interposer supplies (between dice) 60% of the routing capacity that the normal (within-die) FPGA routing channels supply, there is little impact on the routability of circuits. Smaller routing capacities in the interposer do impact routability however: minimum channel width increases by 20% and 50% when an interposer supplies only 40% and 30% of the within-die routing, respectively. The interposer also impacts delay, increasing circuit delay by 34% on average for a 1 ns interposer signal delay and a four-die system. Reducing the interposer delay has a greater benefit in improving circuit speed than does reducing the number of dice in the system.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Style—*VLSI*; B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*

FPGA'14, February 26–28, 2014, Monterey, CA, USA.

ACM 978-1-4503-2671-1/14/02 ...$15.00.
http://dx.doi.org/10.1145/2554688.2554776.

General Terms

Algorithms, Design

Keywords

FPGA; Silicon interposer; 2.5D ICs

1. INTRODUCTION

Interposer-based multi-FPGA systems are composed of multiple FPGA dice, which are connected through a silicon interposer. The interposer is fabricated with an older technology than that used for the dies, and links between dice include both a micro-bump on each FPGA die and a metal wire on the interposer [15]. This results in a reduced connectivity between dice and increased delay for connections between dice, as compared to the total routing connectivity and the delay one can achieve within a single die. In this paper we present an architecture study of silicon interposer-based FPGAs that analyzes how the reduced connectivity and increased delay impact their performance.

The silicon interposer FPGA technology is interesting because it enables the creation of large FPGAs composed of small dies and also of very large FPGAs, with higher logic capacity than one can achieve with a single die. Such FPGA systems are sometimes called 2.5D FPGAs, since they make use of vertical stacking of dies on an interposer to enable higher integration levels.

Being able to make large FPGAs with multiple smaller dies is particularly interesting at the beginning of a new manufacturing process, when defect densities are high. In such a case, good-die yield drops dramatically as the die size increases, and this drastically impacts the availability of large FPGAs early in the process lifetime.

The idea of using 3 dimensions to design FPGAs is not new. [1] proposed the creation of 3D FPGAs by stacking 2D FPGAs and connecting them with solder bumps. Lin et al also studied 3D FPGAs, but using a different approach [10], proposing multiple active layers, with different FPGA functions in each. While promising, both approaches have manufacturing challenges: [1] requires a higher density of through-silicon vias than can currently be manufactured and the multiple active layers required by [10] are not yet widely available. In contrast, silicon interposers linked to dies via microbumps are now manufacturable.

Chaware et al presented Xilinx's approach to silicon interposer FPGAs [5]. They describe the physical characteristics

of their implementation, including the bump pitch and estimates of the amount of die-to-die connectivity and the die-to-die delay. However [5] does not analyze the architecture question of the routability of the resulting system, nor describe possible CAD optimizations, which are the questions we investigate.

The main contributions of our work are as follows:

- A detailed study of how the multi-FPGA system routability is impacted by the amount of connectivity between dice provided by the interposer.
- An Analysis of the impact of the interposer on timing.
- Modification of the VPR CAD tool to model and target 2.5D silicon interposer-based FPGAs, as well as CAD changes to optimize for 2.5D FPGAs.
- Analysis of the commercially available Xilinx Virtex-7 silicon interposer FPGAs [17][18]. We measure the connectivity reduction and delay increase between dies caused by the interposer on such FPGAs in order to see where these devices lie within the architcture space we explore.

The paper is organized as follows. Section 2 gives more information about silicon interposer technology. Section 3 describes the changes made to the FPGA architecture description and to VPR to target 2.5D FPGAs. Section 4 describes optimizations made to the VPR placement algorithm to improve quality on such FPGAs and Section 5 presents architectural results for 2.5D FPGAs.

2. SILICON INTERPOSER BACKGROUND

As mentioned in the previous section, interposer-based FPGAs allow the creation of chips larger than a single die, making a "More than Moore" improvement on the size and number of logic elements possible, and with chips combined far more tightly and with more connectivity than if they were on separate boards connected through conventional means.

The improvement in the number of logic elements of 2.5D FPGAs over conventional ones is very significant. Xilinx's largest interposer-based FPGA, the Virtex-7 XC7V2000T, has 4 dies (which Xilinx calls Super Logic Regions) and 1.954 million logic elements [18], while the largest non-interposer Virtex-7 die (the XC7VX980T), has 979k logic elements and Altera's largest FPGA, the Stratix V 5SGXBB, has 952k logic elements [2]. Even though all these FPGAs use a 28 nm process, silicon interposer technology allows the creation of FPGAs with twice the resources possible on even an extremely large single die.

Another major advantage of interposer-based FPGAs comes at the beginning of a new manufacturing process, when the defect density is high [15]. Bigger dice suffer a much reduced yield compared to smaller dice, and this greatly affects the supply and cost of top-of-the-line FPGAs to early adopters.

To illustrate this impact, consider a new process in which the defect density is $1/cm^2$, which is a reasonable value early in the process lifecycle, and the die area is $6cm^2$, which roughly matches the size of the largest member of a high-end FPGA family such as Virtex 7. Using the Poisson Yield Model [7], the yield is only 0.25% of die. If instead the chip is composed of four $1.5cm^2$ dies, the yield is 22%. This means that a 12 inch silicon wafer with $730cm^2$ of area would produce on average 0.3 working $6cm^2$ dies, while the same wafer would produce on average 107 working $1.5cm^2$ dies. Therefore, as a $6cm^2$ chip would be composed of four $1.5cm^2$ dies, the wafer would yield 26.75 systems on average, as the "assembly yield" of placing these four die on an interposer is very high [5]. Hence the number of interposer-based FPGAs created from the same silicon wafer would be almost 100× greater than a monolithic FPGA of the same size.

When the process matures and the defect density decreases this advantage drops significantly. Consider a mature process with defect density of $0.1/cm^2$. The yield for a $6cm^2$ die is 55% and the yield for a $1.5cm^2$ die is 86%. Hence the number of single die FPGAs created from a $730cm^2$ silicon wafer would be 66.9 and the number of interposer-based FPGAs created would be 104.6. While the interposer-based FPGA still has a yield advantage it might not lead to a major cost advantage, particularly when the cost of the interposer and assembling the die to it are included. For the large, state-of-the-art FPGAs that are built early in a process cycle and heavily used for prototyping, however, there is clearly a compelling cost advantage to an interposer-based solution.

2.1 Virtex-7 Interposer-based FPGAs

The Xilinx 2.5D FPGAs from the Virtex-7 family are currently the only commercially available silicon interposer-based FPGAs. As described in Section 3 we are studying the impact of several key interposer parameters on the performance of the multi-FPGA system, including (i) the percentage of the wiring normally present between rows of the FPGA which are cut when crossing between dice, and (ii) the extra delay (vs. a normal connection between adjacent rows) added when one must traverse the interposer. To locate where Virtex-7 lies in this architecture space, we combined published information on the implementation of Xilinx's interposer-based FPGAs [5] with a detailed analysis of the *XC7V2000T* FPGA routing resources visible in the *Vivado Device View*.

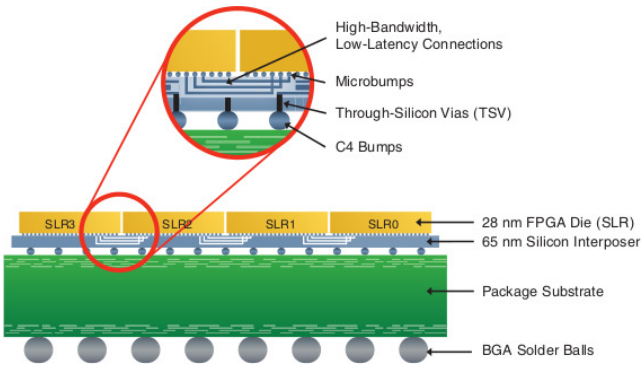


Figure 1: Lateral view of an interposer-based FPGA[17]. The FPGA dice are at the top, and are connected to the silicon interposer through microbumps. The interposer is then connected to the substrate through C4 bumps.

The *XC7V2000T* is composed of four identical dice arranged such that the vertical routing crosses between the dice. Each horizontal edge of each die has 280 groups of

48 length-12 wires crossing the interposer, which sums to a total of 13440 wires between dice. There are also 40 clock wires crossing the interposer. The average number of wires per vertical channel of this FPGA is 210 and there are approximately 280 vertical channels on the FPGA, resulting in approximately 58800 vertical wires crossing a horizontal cutline within a die. Hence the number of wires which cross the interposer is about 23% of the total number of within-die vertical wires.

The 28nm dies are connected to the 65nm silicon interposer through microbumps with a 45μm pitch. Hence the area occupied by microbumps at one edge of one die is 13440 $\times$ (45μm)^2 = 27mm^2. If we assume each die is $7 \times 12mm$, as presented by Chaware et al. in [5], the bumps have to be spread out near the edge and need to go as far as $2.25mm$ away from the edge of the die. This greater distance from the border increaes the length of the inter-die connections, and along with the presence of the micro-bumps and their capacitance, leads to an increased delay for these crossing wires vs. that of a typical on-die routing wire. Chaware et al. state that the latency to cross the interposer is approximately $1ns$. For comparison, a typical medium length 28 nm FPGA routing wire (e.g. spanning four logic blocks) has a delay of approximately 125 ps, while a longer wire (e.g. spanning 12 logic blocks) has a delay of approximately 250 ps.

Overall, these interposer-based FPGAs have increased delay and reduced connectivity between dies, with approximately 23% of the usual number of vertical wires crossing between dies and approximately $1ns$ of increased delay to cross the interposer.

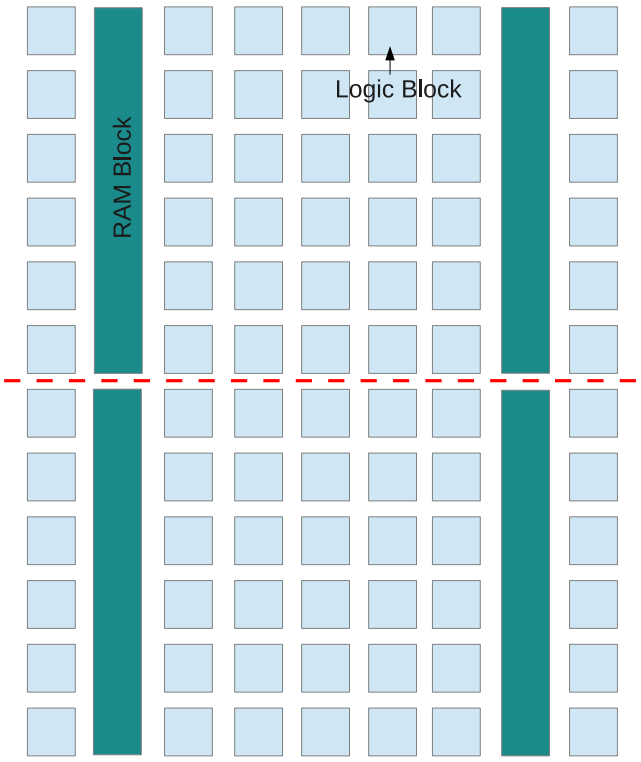


Figure 2: A sample two-die / 1-cutline architecture containing both logic blocks and RAM blocks.

3. ARCHITECTURE MODELS

To properly model a silicon interposer FPGA, we use version 7.0 of the popular FPGA exploration toolset, Verilog-to-Routing (VTR) [13]. The logic synthesis portions of the flow (ODIN II and ABC) are left untouched, while we modify the placement and routing portion of the flow (VPR [3]) to model and optimize for interposer-based FPGAs. The modifications are made in such a way that they require no changes to any of the input files, so one can experiment with interposer-based FPGAs with any existing benchmark circuits and any existing VPR-format FPGA architecture description simply by specifying appropriate command-line parameters.

We add the following parameters to VPR: *number of cuts*, *% wires cut*, *increased delay*, and *interposer-routing interface*. These parameters describe the interposer portion of the 2.5D FPGA as detailed below.

3.1 Number of cuts

Number of cuts describes how many cuts are made to the interposer-based FPGA versus a monolithic die. If *number of cuts* equals 1 then there are 2 dice, and so on. We investigate values of 1 and 3 (2 and 4 dice). The Virtex 7 family has members with *number of cuts* = 2 and 3. Figure 2 shows a sample architecture with one cutline.

3.2 % wires cut

This variable describes and models the reduced connectivity between different dice by specifying the fraction of routing wires that are removed at the border between dice. For example, if a channel had 200 wires and *% wires cut* was 70, 140 of them would be cut and only 60 would pass through the interposer. Higher values of *% wires cut* make an interposer easier to manufacture, and can reduce the interposer delay by allowing all the microbumps linking dice to be placed near the die edge. However, the higher *% wires cut* is, the less routable the multi-die system becomes. As described in 2.1, the Virtex 7 family has *% wires cut* = 77%.

3.3 Increased delay

Interposer wires are longer and wider than on-die wires and have microbumps on each end. *Increased delay* models the resulting larger delay when compared with wires which are internal to a die. A reasonable estimate for this variable is around $1ns$, as presented by Chaware et al[5].

3.4 Interposer-Routing interface

In a simple architecture, the vertical routing wires that can not connect to an interposer wire are dangling. Since interposer wires are scarce, we want to be able to study architectures in which the interposer wires are more accessible. To do so, we have enhanced the router to support the following interposer-routing interface options.

(a) Fan-in Transfer for Interposer Wires
For an architecture with *% wires cut* = 0, all vertical routing wires have a corresponding interposer wire that connects them to the adjacent die. Practical interposers, however, can only accommodate a fraction of all vertical routing wires. When *fanin transfer* option is *on*, we place a multiplexer at the input of each available interposer wire and connect some adjacent "dan-

gling" vertical routing wires to the multiplexer. This allows for additional flexibility in routing by ensuring that *every* vertical routing wire could *potentially* drive an interposer wire through a multiplexer. As an example, if *% wires cut* = 75%, 3 of every 4 interposer wires are cut. Therefore, we place a 4-to-1 multiplexer at the input of the one interposer wire that is not cut, and we connect the 3 vertical routing wires that do not have a corresponding interposer wire to the multiplexer.

(b) Fan-out Transfer for Interposer Wires
When *fanout transfer* option is *on*, we allow the interposer wire output to drive not only its own corresponding vertical wire at the other side of the cut, but also the other "dangling" vertical routing wires that are not driven by anything because their corresponding interposer wires are cut.

(c) Bidirectional Interposer Wires
Dominant routing architectures in modern FPGAs use unidirectional wires [11, 12]. Such wires can only be driven at one point, normally their start point. When this option is *on*, we allow interposer wires to be driven from one end or the other by adding tri-state buffers at both ends.

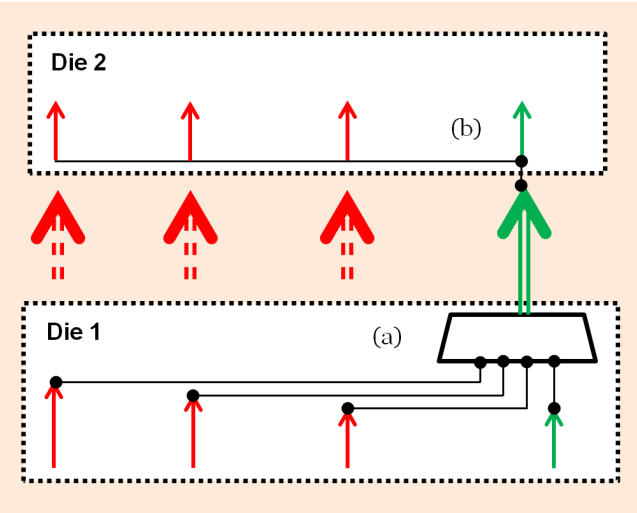


Figure 3: For a channel width of 4 and 75% wires cut, only 1 interposer wire exists (in green) and 3 interposer wires are cut (in the red). (a) Fan-in Transfer for Interposer Wires allows red routing wires in die 1 to drive a mux that allows them to cross the interposer layer. (b) Fan-out Transfer for Interposer Wires allows the output of the interposer wire to drive wires in other tracks in die 2.

These options are independent and can be combined to achieve more flexible routing. Enabling any of these options requires adding specific circuitry (such as multiplexers) on the dice which causes very small area and delay increase that we neglect in our experiments.

3.5 Implementation

To model an interposer-based multi-FPGA system in the CAD tool, we use the existing VPR infrastructure for monolithic FPGAs and make several modifications to it.

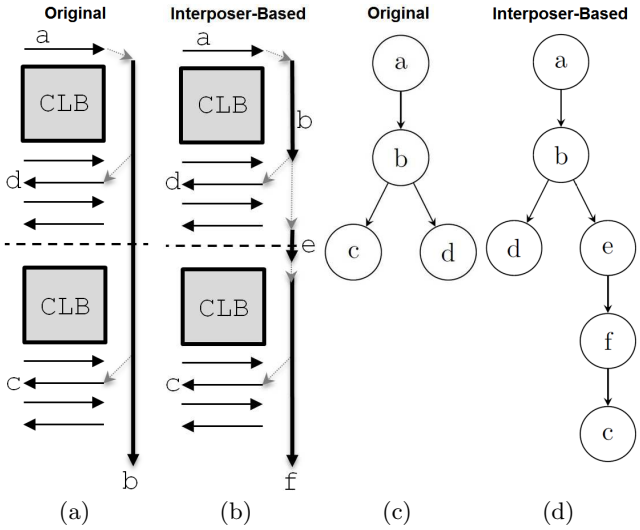


Figure 4: (a) Regular routing wire connectivity. (b) Replacing the long wire with two wires and connecting them via an interposer wire. (c) rrgraph for regular connectivity. (d) rrgraph after modifications.

First, The presence of multiple dice in an interposer-based FPGA is modeled by creating horizontal cuts in the FPGA. The *number of cuts* is specified by the user, and we use that to determine the y-location of the cuts. The cutlines "snap to the grid" such that the height of all dice are equal and no cutline cuts through a block. For instance, a RAM block of height 4 must remain a unit and can not be sliced by a cutline.

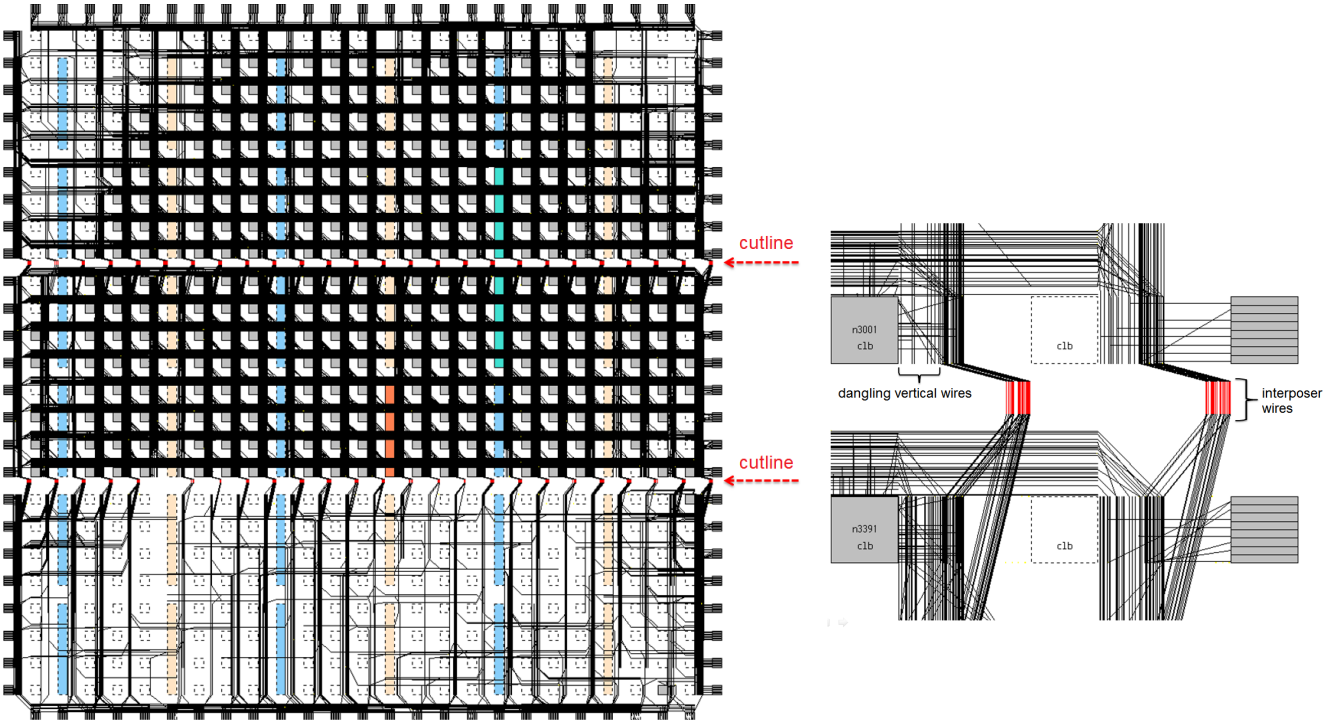
Second, knowing the location of the cutlines, the placer uses the *(x,y)* coordinate of any pin or block to determine the die to which it belongs. The placer uses this information as well as *interposer delay* to better decide where to place each netlist block in order to minimize the cost of inter-die signal transfer as described in Section 4.1.

Third, the Routing Resource Graph (*rrgraph*) must be modified. The rrgraph is the data structure that defines all the available routing wires (*rrnodes*) and switches in the FPGA, as well as the delay of each [4]. Given a suitable rrgraph, the VPR router can implement circuits in the desired FPGA, and the VPR timing analyzer can estimate their delay. In the experiments below, we use an architecture that has only unidirectional wires.

Previous work by Pereira et al [8] modeled the interposer indirectly: they did not introduce new rrnodes in the rrgraph to represent the interposer wires. Rather, they found the rrnodes that cross a cutline. Then they either removed all outgoing edges of these rrnodes (if the routing node was part of the *% wires cut*), or, they increased the switch delay of the rrnode fan-outs (if the wire was not cut) to account for the additional interposer delay.

While this is a reasonable method to model such architectures, it makes it difficult to experiment with different routing-interposer interfaces such as placing input muxes for interposer wires. Therefore, we modify the rrgraph differently to more accurately reflect the interposer layer directly:

For every vertical routing wire in the channel, we add a new routing node ("*interposer node*") into the rrgraph.



(a) (b)

Figure 5: VPR Graphical User Interface. (a) Full chip view. (b) Connectivity at the cutline. Due to the limited signal capacity of the interposer, some vertical routing wires are left dangling.

Then, we determine rrnodes (e.g. vertical routing wires) that cross a cutline in the rrgraph, and we replace them with 2 rrnodes: one that is above the cutline and one that is below the cutline. We properly transfer fan-ins and fan-outs of the original node onto the two newly created nodes and we connect the two nodes using an interposer node. Finally, we remove *all* edges from *%wires cut* percent of interposer nodes and increase the switch delay of output edges of the remaining interposer nodes by *increased delay*. Figure 4 shows an example of these operations.

As illustrated in Figure 5, no vertical routing wires cross a cutline boundary and a fraction of them connect to other dies via interposer wires.

4. CAD ENHANCEMENTS

4.1 Placer optimization

Placement is crucial in mitigating the impact of the reduced wiring and increased delay when crossing between dice; a good placement should minimize the number of signals crossing between dice, particularly time-critical ones. VPR uses 2 different costs as the metrics for its placer algorithm: the timing cost and the bounding box (wiring) cost. We modified the placer's bounding box and timing costs to account for the increased latency to cross the interposer and for the reduced wire capacity close to the cutlines, as the wire availability becomes more sparse.

4.1.1 Placer timing cost

The usual VPR timing cost is a (criticality-weighted) summation of the estimated delay (given the current placement) of every connection required by the circuit [14]:

$$Timing\_Cost = \sum_{\forall i,j \subset circuit} delay(\Delta x_{ij}, \Delta y_{ij}) \times criticality(i,j) \quad (1)$$

where ij denotes a connection from block i to block j that exists in the circuit netlist.

This cost function assumes that the FPGA is homogeneous and consequently the delay between 2 points (x_1, y_1) and (x_2, y_2) only depends on $(\Delta x, \Delta y)$. This is obviously not true for interposer-based FPGAs, as the cutlines make them heterogeneous in the y direction. To solve this problem and improve the quality of the results, we add an extra term to the delay function:

$$delay(i,j) = delay(\Delta x_{ij}, \Delta y_{ij}) + times\_crossed(i,j) \times delay\_increase \quad (2)$$

where $times\_crossed(i,j)$ is the number of times this path has to cross the interposer to go from (x_i, y_i) to (x_j, y_j) and $delay\_increase$ is the timing penalty of crossing the interposer.

4.1.2 Placer wiring cost

The bounding box cost estimates the amount of wiring required for a net, based on the number of pins and size of the net's bounding box. VPR's original formulation is [3]:

$$wiring\_cost_{orig} = \sum_{n=1}^{N_{nets}} q(n) \times [\frac{bb_x(n)}{avg\_chanx\_W(n)} + \frac{bb_y(n)}{avg\_chany\_W(n)}] \quad (3)$$

where $bb_x(n)$ and $bb_y(n)$ are the dimensions of the bounding box of net n in the x and y directions, respectively. avg_chanx_W and avg_chany_W are the average x-directed and y-directed channel widths over this bounding box. Finally, $q(n)$ is a function obtained from [6] which models the fact that bounding boxes underpredict the required routing for high fanout nets. $q(n)$ slowly increases with the fanout of net n.

VPR's wiring cost also considers the FPGA to be homogeneous, and uses only the number of nets, the size of the net's bounding box and the average number of wires per channel to calculate the cost. Thus, to account for the reduced connectivity near the cutlines we create an extra cost term, *cut_cost*. This new cost is added to (3) to create the total wiring cost.

$$wiring\_cost = wiring\_cost_{orig} + cut\_cost \quad (4)$$

In [8], Pereira et al tested several different *cut_cost* formulations, as well as different weighting for each. They found that the following *cut_cost* results in the best quality of results in terms of minimum routable channel width and critical path delay.

$$cut\_cost = \sum_{n=1}^{N_{nets}} C' \times bbHeight(i) \times times\_crossed(n) \quad (5)$$

where

$$C' = \frac{ratio\_wires\_cut}{avg\_chany\_W(n)} \quad (6)$$

bbHeight(n) is the height of the bounding box of the net n and *ratio_wires_cut* is the ratio of wires cut at the cutline. This formulation of C' ensures that the new cost term is of roughly the same magnitude as $wire\_cost_{orig}$ in (3) and that it is weighted more heavily for interposer architectures in which the wiring between dice is more scarce.

4.2 Router optimization

Once the Routing Resource Graph is modified as detailed in Section 3, the VPR router adapted automatically to the interposer architecture. The router starts at a net source node and expands by discovering adjacent routing nodes until it reaches the destination node. At any point, the router knows the cost of the path taken to the current point and it uses a *lookahead cost function* to estimate the resource and delay cost from the current node to the destination. Using this lookahead cost, the router can properly sort routing alternatives. The lookahead cost function in VPR is not aware of the interposer, and therefore it takes a long time to converge to a routing decision. To make the lookahead aware of the interposer, we modify the lookahead cost function by adding terms that account for extra *interposer delay* and extra *interposer nodes* that may exist on the route from the current node to the destination node.

The VPR router uses the following to calculate a cost for a routing node r on the route from i to j:

$$Cost(r) = Criticality(i,j) \times Delay_{Elmore}(r, topology) + [1 - Criticality(i,j)] \times Congestion(r) \quad (7)$$

Given this cost function, the router's lookahead algorithm determines the total cost of the route from i to j when routing node r is used:

$$Total\_Cost(r) = \sum_{\forall l \in RT(i,r)} Cost(l) + \alpha \times ExpectedCost(r,j) \quad (8)$$

where *RT(i,r)* is the set of all routing nodes used so far from i to r and α is a constant number that determines the aggressiveness of the directed search. α is 1.2 by default in VPR. VPR's original $ExpectedCost(n,j)$ function is [4]:

$$ExpectedCost(r,j) = Criticality_{fac} \times T_{delay} + (1 - Criticality_{fac}) \times ExpectedCongestionCost(r,j); \quad (9)$$

T_{delay} is an estimate of the delay that the router predicts it will see on the route from r to j. Since (9) assumes a homogeneous FPGA, T_{delay} does not include the additional delay some routes experience by going through the interposer. Thus, for interposer-based architectures, we replace T_{delay} in (9) with T'_{delay} to include the delay of expected interposer hops from r to j.

$$T'_{delay} = T_{delay} + T_{interposer} \times ExpectedInterposerHops \quad (10)$$

In a similar way, we modify the $ExpectedCongestionCost(r,j)$ in (9) to account for the new interposer routing nodes.

The interposer-aware router lookahead is able to make routing decisions more quickly given the large delay difference of a path that crosses the interposer as opposed to a path that doesn't.

4.3 Effectiveness of the enhancements

For all experiments in this paper, the remaining architecture parameters of the FPGA are taken from the "flagship" architecture of the VTR project (*k6_frac_N10_mem32K_40nm.xml*). The parameters of this architecture are in line with both current commercial FPGAs and academic research into best practices. It consists of logic clusters with 10 fracturable 6-LUTs per block (N=10, k=6), and also includes 32kb RAM blocks and DSP blocks configurable to perform 9x9, 18x18 or 36x36 multipliers. The delay values in the architecture are taken from 40 nm circuit simulations and 40 nm commercial FPGAs. It uses unidirectional routing, with all wire segments having length L = 4.

We used VPR 7.0, with the enhancements we detailed above, and the architecture file *k6_frac_N10_mem32K_40nm* in the experiments below. All experiments targeted the smallest FPGA (with number of rows equal to number of columns) which could accommodate a benchmark circuit; this represents a very full FPGA with little white space left, and hence presents a difficult case to an interposer-based FPGA as no die can be left mostly empty.

We have used the eight largest circuits from the VTR benchmark [16], namely: *bgm*, *LU8PEEng*, *LU32PEEng*,

mcml, *mkDelayWorker32B*, *stereovision0*, *stereovision1* and *stereovision2*. The size of these circuits ranges from 9,100 to 153,000 primitives (LUTs, FFs, etc.), with an average size of 52,600 primitives. All results are the geometric mean over all circuits for a given interposer architecture.

To obtain the *critical path delay* the circuits were run with a low stress routing with a channel width, $W = 1.3 \times minW$, where $minW$ is the minimum channel width for which the circuit is still routable.

4.3.1 *Effectiveness of placer optimizations*

Enabling the enhancements described in Section 4.1 improves the results .... haven't finished running these yet. So, <TODO EHSAN>: fill this section with actual results. For now, let's use a graph from Andre as a placeholder! The graphs show that combining the timing with the bounding box optimizations led to an average of 20% improvement on the area-delay product and the most significant contribution was made by incorporation of *cut_cost* into the wire cost function.

We believe the key to the good performance of the new placer cost functions is that it produces gradual gains as bounding boxes crossing the cutline shrink to be closer and closer to being captured entirely on one side of the cutline. Consider for example the 3 bounding boxes shown in Figure 6. Note that bounding box (a) and (b) both cross the cutline, but (b) is penalized less by (5) because bounding box (b) is mostly on the lower side of the cutline; it is more likely that later smaller placement changes will result in the bounding box moving entirely below the cutline, reducing interposer wiring demand. Cost function (5) penalize bounding box (a) more than (b) and (b) more than (c) to guide placement to gradually move bounding boxes to one side or the other of a cutline.

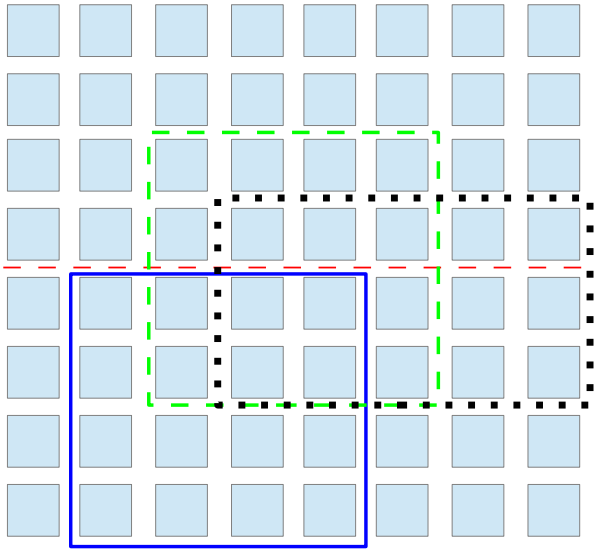


Figure 6: Illustration of three different scenarios for the wiring cost. The dashed green box shows a case where *(a)* it crosses the interposer, the dotted black box shows a case where *(b)* it barely crosses the cutline, and the solid blue one shows a case where *(c)* the bounding box does not span the cutline.

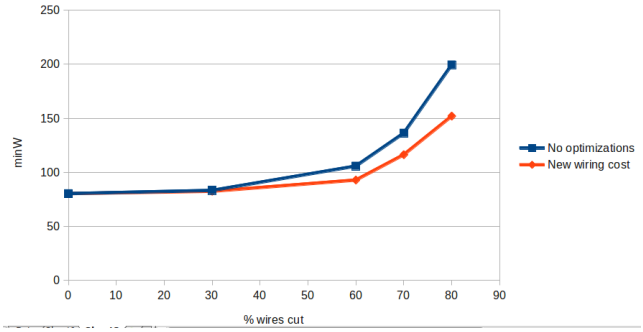


Figure 7: Minimum channel width vs. *% wires cut*.

4.3.2 *Effectiveness of router optimizations*

The router optimizations described in Section 4.2 extensively improve the runtime of the router lookahead algorithm. Figure 8 shows geomean of the router runtime for the 8 circuits for different *% wires cut* values. On average, the router runs 31 times faster and finds identical minimum channel widths for all circuits.

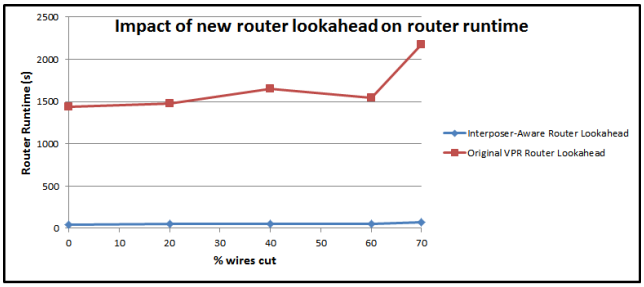


Figure 8: Router runtime improvement using updated lookahead function

The new router lookahead changes the priority of the routes discovered by the router. Using the new router lookahead function increases the critical path delay by 1.3% on average as shown in Figure 9.

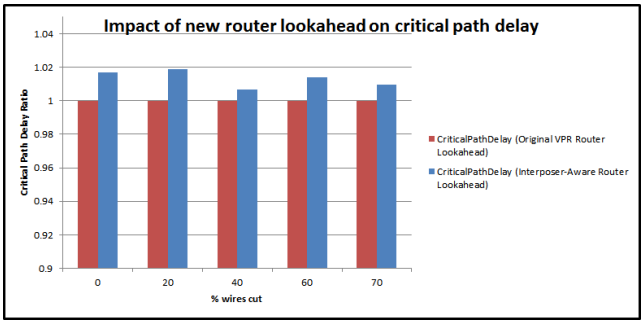


Figure 9: The new router lookahead increases the critical path delay.

5. ARCHITECTURE RESULTS

Using the best CAD settings found in Section 4 and the best CAD flow described in Section **??**, we analyze the impact of the key architecture parameters: *% wires cut*, *delay*

increase, *number of cuts*, and *interposer-routing interface*. We run all the experiments using the setup explained in Section 4.3.

5.1 Impact of Number of Dice

To examine the impact of the number of die used to construct an interposer-based FPGA, we varied the number of cuts from 1 to 3 (which varies the number of die from 2 to 4). In these experiments, the *delay increase* was kept constant at $1ns$.

As shown in Figure 10, the number of cuts does have an impact on the minimum channel width, but not a very large or constant one. At 80% of wires cut the experiments with 2 and 3 cuts had the minimum channel width only 8% and 10%, respectively, greater than the scenario with only 1 cut. At lower values of *% wires cut* the difference was even smaller.

Figure 11 shows that the number of dice in an interposer-based FPGA impacts circuit speed significantly, as the critical path delay rises significantly as systems have more cuts. Recall that a monolithic FPGA has a geometric average critical path delay of $23.7ns$ for our test architecture. With *number of cuts* $= 1$ the critical path delay increases by 3 to 4 ns, while with *number of cuts* $= 2$ the critical path delay increases by 5 to 6 ns. With *number of cuts* $= 3$, the critical path delay is typically 7 ns higher than that of a monolithic FPGA.

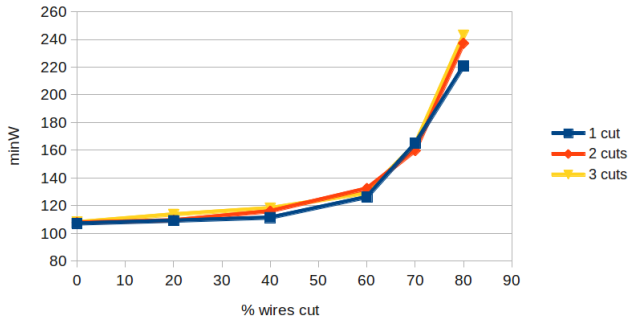


Figure 10: Minimum channel width vs. *% wires cut* for 1, 2 and 3 cuts and $1ns$ of *delay increase*.

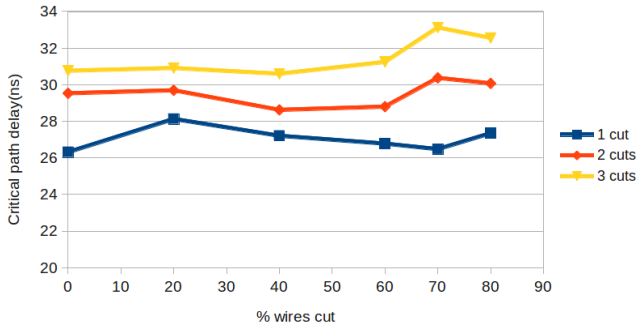


Figure 11: Critical path delay vs. *% wires cut* for 1, 2 and 3 cuts and $1ns$ of *delay increase*.

5.2 Interposer Wiring Capacity (% wires cut)

To analyze the impact of the number of cut wires we ran experiments varying only this parameter while leaving the other parameters constant. We use *number of cuts* $= 1$ (corresponding to two dice), *increased delay* $= 1ns$, *bidirectional interposer wires* $=$ *on*, *fanin transfer* $=$ *on*, and *fanout transfer* $=$ *on*.

Figure 12 shows the graph of minimum channel width versus *% wires cut*. It can be noted from this graph that the minimum channel width is almost steady up to 60% of wires cut, indicating that the placement engine is able to avoid saturating the interposer routing until that point. When more than 60% of the wires are cut however (i.e. the interposer provides less than 40% of the usual within-die routing capacity), the minimum channel width grows rapidly indicating that the interposer routing bandwidth has become a limiting factor.

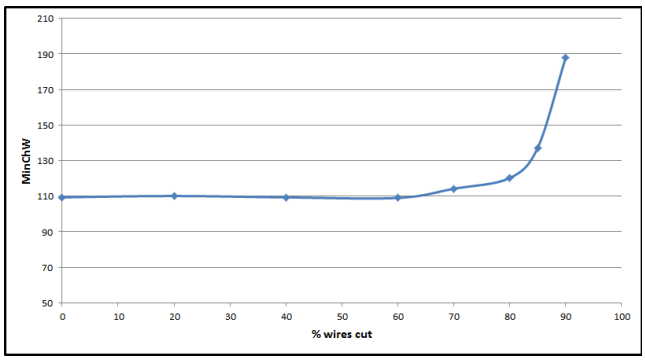


Figure 12: Minimum channel width vs. *% wires cut* for 2 dice and $1ns$ of *delay increase*.

The critical path delay, depicted in Figure 13, on the other hand, is not strongly influenced by the percentage of wires cut, as the critical path delay at 80% of the wires cut is essentially the same as at 0% wires cut. Note, however, that this same critical path delay is achieved at a much higher channel width.

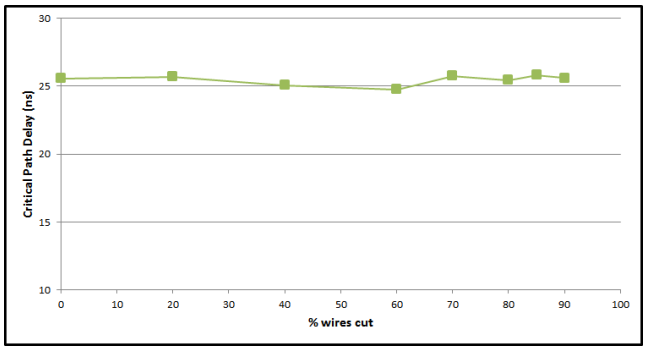


Figure 13: Critical path delay vs. *% wires cut* for 2 dice and $1ns$ of *delay increase*.

Figure 14 provides an alternative way to visualize the relationship between interposer routing supply and routability. This figure shows how the geometric average minimum channel width required within the FPGA dice varies as the geometric average of the *absolute* number of wires crossing the interposer in each channel varies, again for 2-die system.

When 110 tracks cross the interposer, the interposer channels have the same capacity as the vertical routing channels within each FPGA die. As fewer wires cross the interposer, the channel width required within the FPGA dice increases to compensate for the routing difficulty in crossing the interposer. The increase is gradual as the interposer routing is reduced from 110 tracks per channel to 34 tracks per channel; over this range the routing per channel required in the FPGA dice increases from 110 tracks per channel to 114 tracks per channel. As the routing crossing the interposer is further reduced however, it becomes very difficult to increase the within-die routing sufficiently to compensate. At 18 tracks crossing the interposer channels, for example, the within-die routing must have a channel width of *188* tracks to successfully route the designs. Clearly the CAD tools have the ability to trade-off interposer routing for within-die routing over a reasonable range but below a certain level (40% of the original within-die minimum channel width in our experiments) routability becomes almost solely limited by the wiring crossing the interposer and further reduction in interposer routing is not productive.

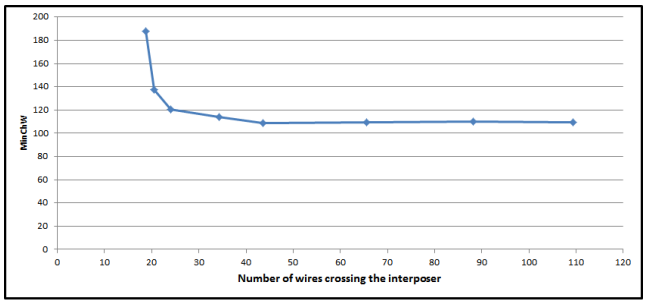


Figure 14: Geometric mean of required intra-die minimum channel width vs. geometric mean of the number of wires crossing the interposer for 2 dice and $1ns$ of *delay increase*.

5.3 Circuit Speed vs. Interposer Delay

To investigate the impact of the interposer delay (*delay increase*), we keep all parameters constant and we sweep the *delay increase* from 0 to 1.5 *ns*. We use *number of cuts* = 1 (corresponding to two dice), *bidirectional interposer wires* = *on*, *fanin transfer* = *on*, and *fanout transfer* = *on*.

Figure 15 shows critical path delay versus *% wires cut* for 4 different values of *delay increase*. The penalty in critical path delay is significant, ranging between 1 and 3.7 times the interposer *delay increase*, when compared to the case where the interposer adds no delay. Note that the 0% wires cut with a 0 ns *delay increase* in Figure 15 corresponds to a traditional monolithic FPGA. The speed of an interposer-based FPGA is strongly correlated to *delay increase*: a $0.5ns$ interposer delay increases the critical path delay by 4%, while a $1ns$ interposer delay increases critical path delay by approximately 12% vs. a monolithic FPGA. But once again, the critical path delay shows little correlation to the *% wires cut*.

5.4 Routing-Interposer Interface

As described in 3.4, we have added three boolean options to VPR that allow us to experiment with different routing-interposer interfaces. These interfaces provide more flexibility to route signals between different dice and hence help us achieve lower *minimum channel width* and *critical path delay*.

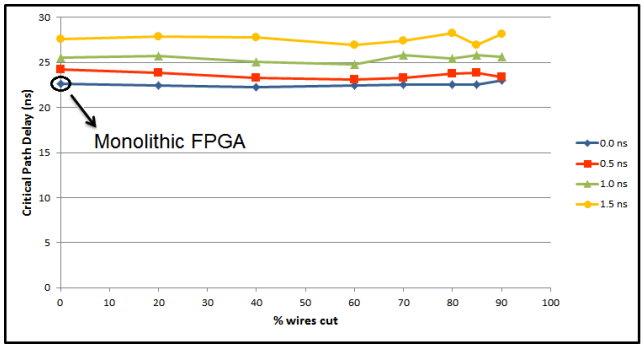


Figure 15: Critical path delay vs. *% wires cut* for 2 dice and 0.0, 0.5, 1.0 and $1.5ns$ of *delay increase*.

To measure the impact of routing-interposer interface parameters, we set *delay increase*=$1ns$, *number of cuts=1*, and *% wires cut=80%*; then, we measure the impact of the *bidirectional interposer wires*, *fanin transfer*, and *fanout transfer* parameters as well as different combinations of them. Figure 16 summarizes the results of these experiments.

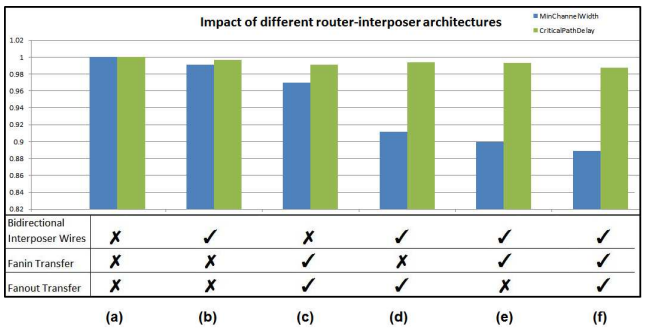


Figure 16: Exploring more flexible interposer-based architectures by modifying routing-interposer interfaces.

Experiments show that using bidirectional interposer wires instead of unidirectional wires (column *b* in Figure 16) improves minimum channel width by 1% and critical path delay by 0.5%. Therefore, while it is beneficial to use bidirectional interposer wires, it does not dramatically improve performance. We believe that is because a well placed and routed design that fills the chip completely with little spare room has approximately equal number of signals crossing the interposer in each direction (i.e. number of signals transferred from die 1 to die 2 is close to the number of signals transferred from die 2 to die 1).

Adding multiplexers in front of interposer wires allows dangling routing wires to connect to interposer wires (*"fanin transfer"*) and adding extra load on the output of the interposer wires (*"fanout transfer"*) allows the interposer wire to drive dangling wires on the other side of the cut. Therefore these modifications provide higher flexibility to better utilize interposer wires. Experiments show that enabling these two options alone without using bidirectional interposer wires

(column c in Figure 16) reduces minimum channel width by 3% and reduces critical path delay by 1%.

Moreover, better performance can be achieved if either of these two options is used together with bidirectional interposer wires. Enabling *fanout transfer* with bidirectional interposer wires (column d in Figure 16) reduces the minimum channel width by 9% and critical path delay by 0.7%. Enabling *fanin transfer* with bidirectional interposer wires (column e in Figure 16) has an even stronger impact and reduces the minimum channel width by 10% and critical path delay by 0.7%. This signifies that the flexibility to drive an interposer wire is more important for routability compared to the flexibility of having an interposer wire drive multiple vertical routing wires on the other side of the cut.

Enabling all of these architecture modifications simultaneously (column f in Figure 16) provides the highest level of flexibility for the router and achieves 11.2% reduction in minimum channel width and 1.3% reduction in critical path delay. These models show that interposer-based FPGAs can extensively benefit from hardware that provides a flexible routing-interposer interface.

The *fanin transfer* and *fanout transfer* options ensure that no vertical routing wires are dangling at the cutline, hence, each vertical routing wire either has exactly *one* interposer wire driving it or it drives a mux of exactly *one* interposer wire. As a final experiment, we try to add additional fanins and fanouts to interposer wires, so that each vertical routing wire can *potentially* drive multiple interposer wires, or be driven by one of many interposer wires. We found that adding extra fanouts to interposer wires does not help performance or area, but adding additional fanins to interposer wire muxes helps reduce the minimum channel width while maintaining the same critical path delay. As shown in Figure 17, adding 4 additional inputs to interposer wire muxes reduces the minimum channel width by 1% and adding 8 additional inputs reduces minimum channel width by 1.5%. After that point the performance is no longer limited by the ability to get on an interposer wire and hence we do not see any gains for adding more than 8 additional inputs. Notice that column *0* of Figure 17 is the same as column f of Figure 16.

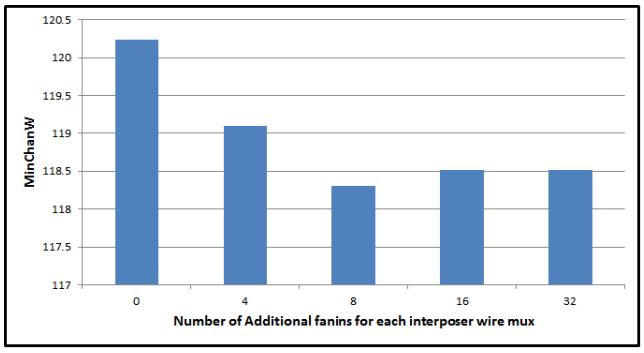


Figure 17: Adding up to 8 extra fanins to interposer wire muxes helps reduce minimum channel width.

6. CONCLUSIONS AND FUTURE WORK

We have extended VPR to model and optimize for interposer-based multi-FPGA systems. While such systems are now a commercial reality, we do not know of any prior study of their key architectural parameters. We found that by modifying VPR's placement cost function we could improve the routability (reduce minW) by 18%, while simultaneously improving speed by 2%. Interestingly, we found that a smooth cost function that less precisely models the interposing wiring demand outperformed direct monitoring of the number of required interposer connections during annealing.

We defined three key architecture parameters for interposer-based FPGAs, and used this extended VPR to analyze their impact on minimum channel width and critical path delay. We found that the *minimum channel width* increases steeply after more than 60% of the within-die wires are cut at the interposer, and when 80% of wires are cut the *minimum channel width* is more than double that required by a monolithic FPGA. The Xilinx *XC7V2000T* FPGA has 77% of the normal wiring cut at the interposer, indicating that commercial interposer-based FPGAs will require high-quality CAD optimization to maintain good routability. The *critical path delay* is not strongly influenced by the *% wires cut* but is strongly influenced by the interposer delay and the number of cuts. Our results show that the critical path usually crosses the interposer cutline more than once, as the *critical path delay* is increased by more than the interposer *delay increase*. Increasing the number of dice in an interposer-based FPGA does not significantly impact the *minimum channel width*, but does lead to a larger *critical path delay*.

There are many interesting CAD and architecture questions for interposer-based FPGAs which we plan to investigate. Currently the wires that cross the interposer are accessed with the same switch structure as other wires in the FPGA. Since interposer wires are more scarce, possibly these wires should have larger multiplexers feeding them, which would make them easier to use; this may help routability of the system. Such changes will have to be carefully considered however, to make sure the FPGA can still be laid out as an array of regular tiles. Alternative CAD flows to improve the interposer routability are also possible. For example, instead of following the synthesize, place and route CAD flow of a conventional FPGA, we could add a partitioning step before placement which would divide the circuit into one partition per die. Such a flow may improve routability, as a partitioner's main focus is minimizing the number of cut signals. Enhanced partitioners will be required, as current partitioners such as hMetis [9] cannot model the heterogeneous balance constraint (i.e. use no more than the available device logic, RAM, DSP and I/O blocks in any die) present in FPGAs. Additionally, by dividing the placement problem into two pieces – partitioning and within-die placement – we may increase the critical path delay as we can no longer globally optimize the placement of timing critical paths in one unified placement step. Nonetheless, this is a viable alternative approach and a comparison to the approach taken in this paper would be very interesting.

7. ACKNOWLEDGMENTS

This work was supported by a Ciência sem Fronteiras scholarship from CNPq - Brazil and the NSERC/Altera Industrial Research Chair in Programmable Silicon.

8. REFERENCES

[1] M. Alexander, J. Cohoon, J. Colflesh, J. Karro, and G. Robins. Three-Dimensional Field-Programmable

Gate Arrays. In *IEEE Int. ASIC Conference and Exhibit*, pages 253–256, 1995.

[2] Altera. Stratix V Device Overview. *www.altera.com*, 2013.

[3] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *Int. Workshp on Field-Programmable Logic and Applications*, pages 213–222. Springer, 1997.

[4] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[5] R. Chaware, K. Nagarajan, and S. Ramalingam. Assembly and Reliability Challenges in 3D Integration of 28nm FPGA Die on a Large High Density 65nm Passive Interposer. In *IEEE Electronic Components and Technology Conference (ECTC)*, pages 279–283, 2012.

[6] C. Cheng. RISA: Accurate and Efficient Placement Routability Modeling. In *IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 690–695, 1994.

[7] J. Cunningham. The Use and Evaluation of Yield Models in Integrated Circuit Manufacturing. *IEEE Trans. on Semiconductor Manufacturing*, 3(2):60–71, 1990.

[8] A. Hahn Pereira and V. Betz. Cad and routing architecture for interposer-based multi-fpga systems. In *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, pages 75–84. ACM, 2014.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Trans. on VLSI*, 7(1):69–79, 1999.

[10] M. L., A. El Gamal, Y. Lu, and S. Wong. Performance Benefits of Monolithically Stacked 3-D FPGA. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):216–229, 2007.

[11] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and Single-Driver Wires in FPGA Interconnect. In *IEEE Int. Conf. on Field Programmable Technology*, pages 41–48, 2004.

[12] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, et al. The Stratix II logic and routing architecture. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 14–20. ACM, 2005.

[13] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, et al. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(2):6, 2014.

[14] A. Marquardt, V. Betz, and J. Rose. Timing-Driven Placement for FPGAs. In *ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, pages 203–213, 2000.

[15] K. Namhoon, D. Wu, D. Kim, A. Rahman, and P. Wu. Interposer Design Optimization for High Frequency Signal Transmission in Passive and Active Interposer using Through Silicon Via (TSV). In *IEEE Electronic Components and Technology Conference (ECTC)*, pages 1160–1167, 2011.

[16] J. Rose, J. Luu, C. Yu, O. Densmore, J. Goeders, A. Somerville, K. Kent, P. Jamieson, and J. Anderson. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays*, pages 77–86, 2012.

[17] Xilinx. Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency. *www.xilinx.com*, 2012.

[18] Xilinx. 7 Series FPGA Overview. *www.xilinx.com*, 2013.