ON THE DIFFICULTY OF PIN-TO-WIRE ROUTING IN FPGAs

Niyati Shah and Jonathan Rose

The Edwards S. Rogers Sr. Department of Electrical and Computer Engineering,
University of Toronto,
shahniya|jayar@eecg.utoronto.ca

ABSTRACT

While FPGA programmable routing networks are designed to connect logic block output pins to input pins, FPGA users and architects sometimes become motivated to create connections between pins and specific wires in an FPGA. We call these *pin-to-wire* connections, and they are motivated by several reasons: first, a desire to employ routing-by-abutment, as commonly done in custom VLSI, to build modular, pre-laid out systems. Second, partial reconfiguration of FPGAs often requires that circuits in the FPGA connect by abutment. Third, pin-to-wire routing is required to make use of resources that reside within the routing network itself, such as the plentiful multiplexers in the network, or even the configuration bits themselves. In this paper we attempt to understand and measure how difficult it is to form such pin-to-wire connections. We show, for example, under an experimental scenario close to routing-by-abutment, that the total routed wirelength compared to a flat placement of the complete system increases by about 6%, that the critical path delay increases by 15% and the router effort goes up by a factor of 3.5. To achieve this result, it is important to be careful in selecting the specific target wires. Overall we demonstrate that while pin-to-wire connections definitely impose increased stress on the routing architecture and router, it is possible to route some reasonable number of them, and so they can be used under some circumstances.

1. INTRODUCTION

FPGA routing architectures are large-scale interconnection networks that are designed to efficiently connect the output pins of logic blocks to the input pins of other logic blocks. They must do so for a broad class of application circuits with varying demand for interconnect quantity and flexibility [1].

Every so often, FPGA architects and CAD tool developers become motivated to use the interconnection network in a way that it was not intended: creating connections from output pins of logic blocks to specific wire segments in the network, or from specific wire segments to input pins. We will refer to this as *pin-to-wire* routing. One such motivation is to enable routing by *abutment*, in the manner that standard cell-based ASICs connect power rails in rows of cells [2], or that a datapath layout generator might abut stages in a layout [3]. In the FPGA context, this idea translates to having pre-placed and pre-routed modules that could connect together by abutment, presuming efficient routing to and from *specific wire segments* in an FPGA routing channel is possible. In this way, each of the abutting modules can be created independently, as long as the specific wire segments that form the abutting connections are known in advance. With this capability, several different modules could be connected using the same wire abutment pattern, but again only as long as pin-to-wire routing is efficient. Our underlying motivation for routing-by-abutment is to attack the compile-time issue for FPGAs - if pre-placed and routed modules can be placed coarsely and rapidly, and then routed by abutment, a major portion of the compile time can be eliminated.

A second motivation for pin-to-wire routing arises in the use of partial reconfiguration of FPGA modules [4]. Here specific areas of the FPGA are designated to contain different functional modules at different times, yet these must connect to neighbouring modules using the same wire segments. This connectivity problem has been solved by using overlapping 'proxy' LUTs [5] at the cost of some significant amount of intervening logic. If the wire segments themselves can be used to achieve abutment routing, it may reduce this cost.

A third motivation for pin-to-wire routing arises whenever an architect considers making use of the circuits elements within the routing architecture itself: a classic observation in this realm is that although it is inefficient to implement multiplexers in an FPGA using LUTs, it is ironic that the FPGA itself is full of multiplexers as the key building blocks of the routing fabric. In order to make use of those multiplexers (after architecturally modifying access to their selection inputs) there has to be an ability to connect to specific data inputs on the routing multiplexers from output pins of logic blocks. For example, [6] makes use of intra-cluster multiplexers to implement the programmable shift in floating point operations, and shows that with careful layout, the pin-to-wire routing to multiplexers within a cluster is reasonably effective. Oldridge [7] needs a similar capa-

978-1-4673-2256-0/12/$31.00

bility to gain access to the switch block configuration bits of an FPGA, and employ them as a user memory.

While these motivations are interesting, the requirement for effective pin-to-wire routing conflicts with the basic purpose of the FPGA interconnection network, which is pin-to-pin routing. Good architects of those networks [8], [9], [10] choose a quantity of routing and the total routing flexibility to satisfy the interconnection demand of broad classes of application circuits, but only for pin-to-pin connections. This suggests that there may be insufficient quantity and flexibility of routing if connections are made between pins and wire segments because there will be fewer stages of the interconnection network available for such connections. For example, for pin-to-wire routing, the final connection block and intra-cluster routing multiplexers will be missing, giving far fewer potential paths from start to finish. Similarly, for wire-to-pin routing, the output connection block's flexibility is missing. Thus, it is clear that pin-to-wire routing will be more difficult, for each such connection, than pin-to-pin routing. However, most FPGAs are somewhat over-architected in the routing to ensure that circuits with widely varying routing demand will achieve routability. We hypothesize that some of this extra routability could make up for the loss of routability in pin-to-wire routing, and hence, we seek to understand and measure if this is true. The goal of this paper is to experimentally measure the impact of pin-to-wire routing on routing wire length, critical path speed and router effort. In doing so we seek to understand how much pin-to-wire routing could be used in an FPGA, and perhaps enable some of the applications discussed above.

The remainder of the paper is organized as follows. Section 2 presents background on related work. In Section 3 we describe our experimental methodology for measuring various instances of pin-to-wire routing, and also give the corresponding results and understanding generated. In Section 4 we conclude.

2. BACKGROUND

In this section we review prior related work, and provide the basic terminology that is used in this paper.

2.1. Related work

When using the partial reconfiguration feature of an FPGA, each reconfigurable region needs to be placed and routed independently, yet have connections between those regions. Researchers have previously looked at several methods of making those connections.

Athanas et al. [4] deal with the issue of connecting between reconfigurable 'sandbox' regions/modules by encasing them in wrapper structures before placement and routing. The wrapper structure provides anchor points for the module's ports, which exist at pre-fixed locations so that compilation of different modules will overlap and therefore connect. At run-time, module-level placement is performed with emphasis on reducing the interconnect between neighbouring modules, and attempts to make use of routing-by-abutment. If the placement is unable to achieve abutment of all the modules that need to connect, then routing is performed using a greedy approach and is restricted to the channel segment between adjacent modules. The routing architecture is abstracted to make the routing solution fast. Both the use of anchor points and abstracted routing implicitly sacrifice area and routed wire length to achieve the routing by abutment, but the paper does not quantify how much. In this work, we're interested in measuring that cost.

The partial reconfiguration design flow provided by Xilinx [5] solves the boundary crossing connection problem in a similar way, by using a fixed placement for a one-input LUT that must be the same for every dynamic logic module that is placed in a reconfigurable region. The routing to that LUT from the non-reconfigurable or static region is determined during the configuration of the static region and is kept untouched during the configuration of the partially reconfigurable region. However, routing to that LUT within the reconfigurable region can vary based on the logic and the routing of the dynamic module being implemented in that reconfigurable region. This LUT, one of which is required for every signal that crosses the boundary, is called 'proxy' logic. A clear disadvantage of this method is the logic overhead involved in implementing such LUTs and also, the delay resulting from passing through the LUT (0.4ns on a Virtex II FPGA [11]).

Koch et al. [11] solve the problem of connection between regions by pre-assigning the routes that flow between regions, eliminating the need for proxy logic. As a result, both the area occupied by the proxy logic LUTs and the delay incurred for passing through them get eliminated. The authors do not, however, indicate if there is any difficulty (in terms of area, wirelength, delay and router effort) in the routing required around these pre-assigned wires, the purpose of the present paper.

Moctar et al. [6] seek to make use of the intra-cluster routing multiplexers to implement shifters for floating point mantissa alignment. This requires signals to be routed to the multiplexer's inputs in a pre-specified order which means that there is pin-to-wire routing. The authors measure the impact of this difficulty by measuring the increase in minimum routing channel width. They show that the minimum routing channel width increases proportionally with the number of shifters in the netlist. They do not observe an increase in critical path delay, but it is also not clear how much stress their example architectures are under. Also, the benchmarks used in this work are I/O limited with relatively sparse logic block utilization, which would result in a relatively easier routing problem.

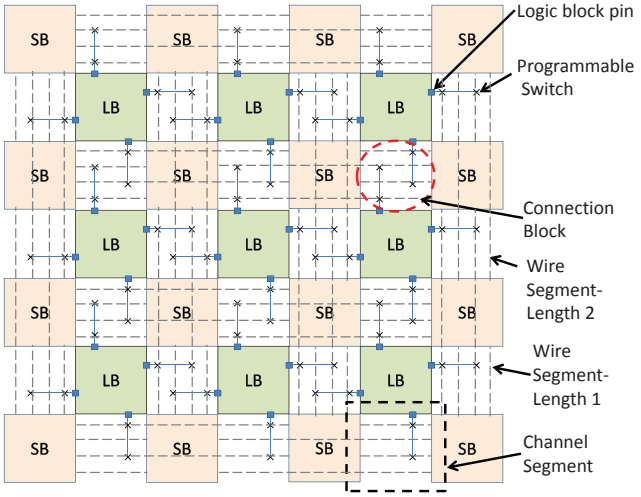

Fig. 1. FPGA Architecture and Related Terminology.

2.2. BASIC TERMINOLOGY

Figure 1 introduces the architecture terminology employed in this paper. Logic block (LB) input and output pins connect to all or some of the wire segments in their neighbouring channels via a connection block. The length of the routing channel spanning a single LB is called a channel segment. The fraction of wire segments in an adjacent channel segment to which an logic block input pin can connect is called input connection block flexibility (Fc_{in}), and the fraction of wire segments in an adjacent channel segment that an logic block output pin can drive is called output connection block flexibility (Fc_{out}). Wire segments connect at channel intersections and connect to F_s (the switch block flexibility) other segments via switch blocks (SB).

3. EXPERIMENT DESIGN AND RESULTS

Our purpose is to measure the impact of pin-to-wire routing on area, speed and router effort. We will do this in the context of a hypothetical (but fairly standard) FPGA architectural and open-source tools. The set of architecture parameters for the FPGA used in these experiments are as follows: The logic architecture is a homogeneous array of logic clusters that contain four 4-input LUTs with 10 input pins per cluster. The routing architecture employs the single-driver approach [12] with staggered length 4 segments, a Wilton switch block [13] with $F_s = 3$ and a connection block with $Fc_{in} = 0.15$ and $Fc_{out} = 0.25$. Note that all length 4 wire segments are internally switch block populated [14]. The resulting switch block multiplexers range in size from 9-10 data inputs. The delays are based on the iFar repository delays for a 45nm CMOS process [15] [16]. In the following, we will also make use of the classical 20 largest MCNC benchmarks [17]. Although these benchmarks are relatively small, we expect that, because there are many wires and switches, that we will gain useful insight into the pin-to-wire question. Nevertheless, the effect of fixed routing structures like carry chains would remain unknown, and be a part of the future work.

Table 1. Net and BLE Statistics

Circuit	# BLE's	Total Nets	Single-Sided Nets		Double Sided
			Left	Right	
alu4	1522	1019	369	370	280
apex2	1878	1408	588	517	303
apex4	1262	959	331	332	296
bigkey	1707	1036	375	454	207
clma	8383	6133	2725	2663	745
des	1591	1499	416	939	144
diffeq	1497	1179	476	489	214
dsip	1370	919	379	371	169
elliptic	3604	2449	1066	887	496
ex1010	4598	3410	1534	1420	456
ex5p	1064	859	262	273	324
frisc	3556	2279	907	873	499
misex3	1397	996	406	339	251
pdc	4575	3178	1266	1174	738
s298	1931	1011	413	397	201
s38417	6406	5045	2304	2342	399
s38584.1	6447	4704	2187	2001	516
seq	1750	1286	492	452	342
spla	3690	2539	1046	902	591
tseng	1047	827	348	318	161

Table 1 lists the set of circuits used in the experiments, and their characteristics - the number of logic elements, the number of nets and several other attributes described in the discussion below. We will now present measurements of pin-to-wire routing in a number of different experimental contexts. Most of these contexts are meant to be proxies for the wiring-by-abutment motivations presented in the introduction, with varying degrees of difficulty.

3.1. Experiment 1: Easy Abutment-Oriented Routing

The first experiment is as follows: we begin by taking a benchmark circuit, and run packing using T-VPack [18] followed by placement and routing using VPR 5.0.2 [19], which employs the timing-driven router described in [14]. This flow is used to determine the minimum number of tracks per channel required to route; we then set the track count to be 30% higher than this minimum (as is common [19]) for all experiments with this circuit. The placement and routing step is then re-run at this track count, and we measure the critical path delay, wire length and router effort. These measurements form the *base* measurement for that circuit.

Next, to serve as a proxy for routing by abutment, we divide the circuit in half, by drawing a vertical line down the

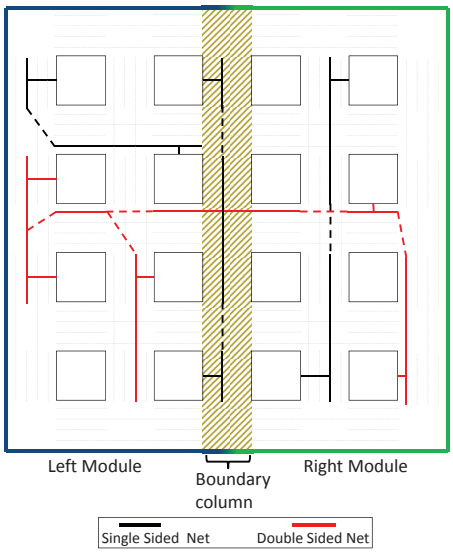

Fig. 2. Partitioned Layout

middle of the circuit (creating a 'middle' or 'boundary' column of the layout) as illustrated in Figure 2. This partition gives rise to two modules which we will call *Left* and *Right* which we will route, in a sense, by abutment. The placement of the Left and Right modules will remain intact from this original placement. In our experiments, we will deal with two kinds of nets from this placement: those whose sources and sinks reside wholly within the Left or Right modules (*single-sided* nets) and those that cross from one side to the other (*double-sided* nets). We will use the double-sided nets to model the inter-module nets in the routing-by-abutment approach. For each circuit, Table 1 gives the number of nets that are on the left side, right side, and the number that are double-sided.

The first experiment was to re-run the routing of this same placement, but with the following restrictions:

1. All single-sided nets are restricted to be routed on their respective sides, as would be true in modules routed by abutment. (In the base routing, these nets could have travelled across the boundary and back).

2. Every double-sided net will be split into two nets, called *faux* nets, one on the Left and one on the Right. In addition, one of the wire segments in the base routing of the double-sided net that crosses the middle column will be chosen as the *boundary* wire segment for that net, as illustrated in Figure 3. If the source of the net is on the left side of the boundary, then the left side faux net consists of the source, all the left side sinks, and the boundary segment acting as a net sink. The right side faux net consists of the boundary wire segment (acting as a *source*) and the remaining right side sinks. (If the source is on the right side, then these constructions reverse appropriately). The constructed faux nets for Figure 3 are illustrated in Figure 4. Note that even though the base routing of the original net may have crossed the middle more than once, in the new routing scenario it will only cross once, as illustrated between Figures 3 and 4. This experiment design decision, made to be similar to what would happen in routing-by-abutment, has specific side-effects discussed in the results below.

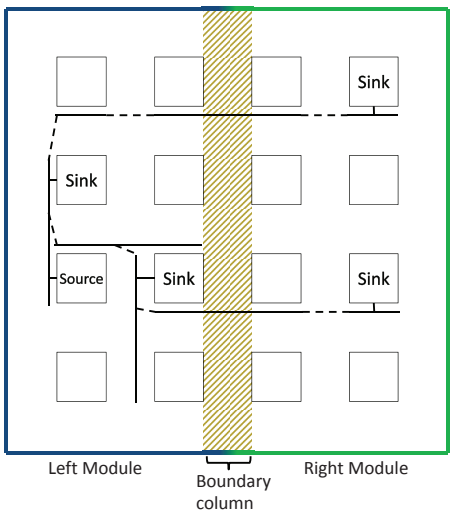

Fig. 3. A Multiple Crossing Double-Sided Net in the Original Netlist

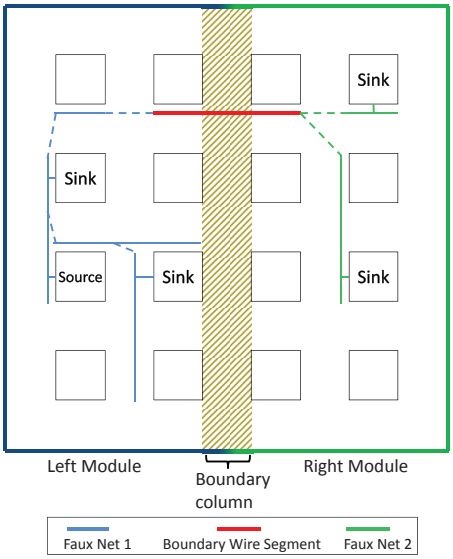

Fig. 4. Faux-Nets built from Net shown in Figure 3 now with just one crossing point

We will also consider two different orderings of the routing process: the first is called *Contiguous Net Ordering* (CO), in which nets are routed in the same order as the base routing, and where the two faux nets that make up each double-sided net are routed one after the other. We are interested

in this base case, as we would generally expect the results of this approach to be the most similar to the original base routing. The second net ordering more directly reflects what would happen in routing-by-abutment, and is called *Partition-Wise Ordering* (PWO): here all the nets in the netlist are rearranged such that the Left nets are routed before the Right nets. This includes the faux nets, and so all Left faux nets are routed together with left-sided single nets.

In all cases, the routing algorithm used is timing-driven. For the base case, the original timing-driven VPR algorithm is used, and the circuit is run through timing analysis as usual to determine the critical path. In the case where the double-sided nets are split into two faux nets, the slack assigned to the faux nets needs to be set carefully. The slack assigned to the source-to-boundary connection is set to be the worst case (smallest) slack for all of the downstream boundary-to-sink connections on the other faux net. This is to ensure that the most critical source-sink connections on the other side are fed by a net that is treated as equally critical, optimizing the speed in the most appropriate way.

3.1.1. Experiment 1 Results

Before discussing the results, it is important to note that the selection of the boundary wire segment turned out to be quite important - making poor choices for the set of boundary segments could result in a deterministically unroutable circuit. (We also take this as evidence that pin-to-wire routing can cause immediate problems.) This unroutable situation occurred primarily in three ways: The first set of issues occurred when there were several boundary segments in close proximity - if there are too many that are too close together, they simply block each other from routing success. Also, if an input or an output pin of a logic block is close to a set of unassociated boundary segments (boundary segments assigned to nets that neither terminate nor originate at this pin), then that pin might simply be blocked. So it is important to make sure that the boundary segments are not too close together. The second issue occurred if the boundary segment of an unidirectional wire simply pointed in the wrong direction, (left to right, say) when the net needed to go the other way. A third issue occurred when both the faux nets wanted to expand using the boundary segment and the boundary segment had insufficient exit points; resulting in resource contention between the nets and hence, routing failure.

For each of the cases (base, modified CO, modified PWO) we measure the total wirelength required after routing, the critical path delay, and the *router effort*. Router effort, at a fixed channel width, is measured as the number of heap push and pop operations performed by the VPR router. The heap in VPR contains the priority-ordered list of wiring segments to pursue while routing. The push count measures the number of nodes that are placed on the heap as potential candidates for expansion, whereas the pop count measures the number of nodes that are actually explored to arrive at the solution.

Table 2. Percentage Increase in Average Metrics for the Easy Abutment Routing Experiment

Net Order	Wire Length	Critical Delay	Heap Push Count	Heap Pop Count
CO	-4%	5%	13%	104%
PWO	-4%	5%	11%	99%

Table 3. Percentage Increase in Average Metrics for the Harder Abutment-Oriented Routing Experiment

Net Order	Wire Length	Critical Delay	Heap Push Count	Heap Pop Count
CO	6%	16%	67%	264%
PWO	6%	16%	66%	255%

Finally, we note that two of the circuits, *bigkey* and *diffeq*, failed to route in this experiment, likely due to the reduced flexibility of the wire sources and sinks.

Table 2 gives the experimental results: the first column indicates which net ordering was used, while the subsequent columns give the percentage increase in geometric mean (across all benchmarks, relative to the base case) for total routed wirelength, critical path delay, heap push count, and pop count, respectively. Interestingly, for the contiguous net order case (CO), the total wire length is actually lower than than the base case. While we were surprised by this, there are two explanations: first that the router is given a good starting point (one of the crossing points in the base case) and so can use its effort to find better routes. The second is that, for the double-sided nets, multiple crossings are forbidden in the approach described above, and so the total net length may end up shorter, but sacrificing a better connection for the more critical path. The latter hypothesis is borne out by the increase in critical path delay. The most telling result is that the router effort, as measured by heap operations, has significantly increased, by a factor of two. We conclude that the smaller number of choices available to the pin-to-wire connections in the faux nets has increased the effort needed to succeed in routing. It is interesting, also, to note that the routing order of the nets has little effect on the results, likely due to the inherently iterative nature of the pathfinder routing algorithm [20].

3.2. Experiment 2: Harder Abutment Routing

In experiment 1, the boundary nodes chosen were set to be the ones that the were chosen in the original routing, base case. In experiment 2, we make one simple change to the above. Here we explicitly choose a different segment than the one chosen in the original routing, but in roughly

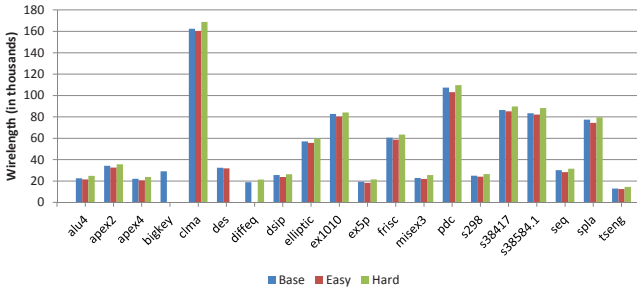

Fig. 5. Wirelength for Expts 1 and 2

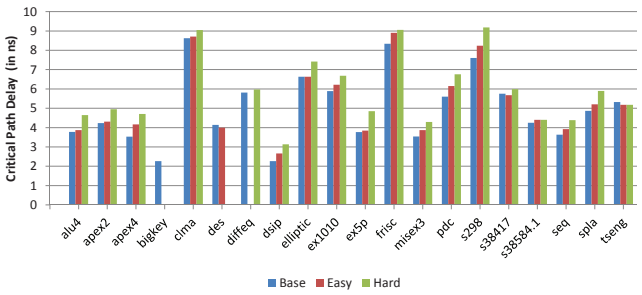

Fig. 6. Critical Path Delay for Expts 1 and 2

the same row/column location of the original boundary segment. Here we are then explicitly selecting a *different* segment than was known to have already worked. We expect that this is more like the typical pin-to-wire routing situation faced in abutment-style routing, where knowledge of a previous full route is not known. Table 3 gives the same summary results of this case, again compared to the base case, as in Table 2. It is interesting to see that the results are dramatically different: the average wirelength *increases* by about 6% (rather than decreasing) and the critical path delay increases by about 16%, a significantly worse change. Furthermore, the router is now working almost three and a half times harder in terms of heap pop counts, and 66% harder on heap pushes. Clearly, there is some pain and suffering dealing with these boundary routes (which represent on average 19% of all of the nets in Table 1). However, it is also interesting to note that these nets did mostly successfully route (with two exceptions, of the circuits *bigkey* and *des*, both of which had a faux net for which no connecting path existed) and as long as the boundary wire nodes are not too congested among themselves, pin-to-wire routing can succeed at some cost.

Figures 5, 6, 7 and 8, give the circuit-by-circuit results for experiments 1 and 2 in wire length, critical path delay heap push count and heap pop count (we give results for partition-wise ordering of nets only, since they were similar to the contiguous ordering case).

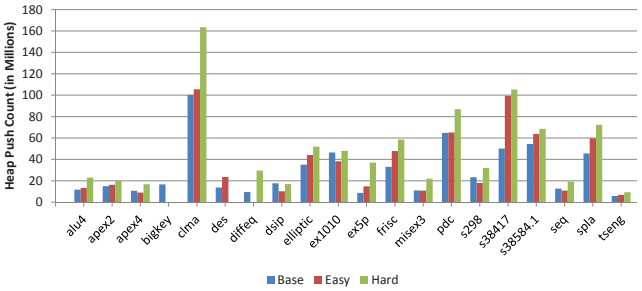

Fig. 7. Heap Push Count for Expts 1 and 2

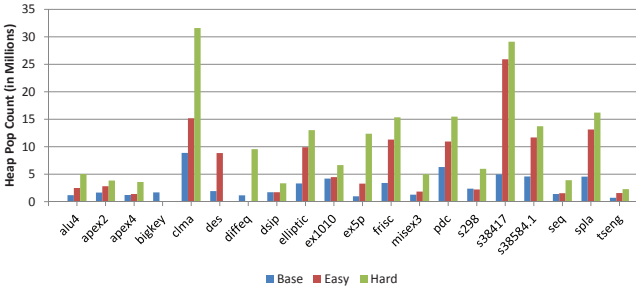

Fig. 8. Heap Pop Count for Expts 1 and 2

3.3. Experiment 3: Dispersed Pin-to-Wire Routing

The previous experiments focused on modelling the wiring-by-abutment motivation presented first in the introduction. Nevertheless, pin-to-wire routing is also motivated by other applications attempting to use the plentiful multiplexers present in the FPGA's routing fabric [6] or to implement wide shallow memories using switch block configuration bits [7]. Both these applications will benefit from knowing how much pin-to-wire routing can be introduced in a netlist before the area, delay and routability of the routed netlist are significantly impacted. We suspect that introducing a small amount of pin-to-wire routing in a netlist may not be significantly detrimental to the final routing, but that it will become more difficult as the amount is increased.

The next experiment allows the amount of pin-to-wire routing being introduced in a netlist to vary. Once the base measurements have been acquired as described in subsection 3.1, the routing step is re-run on the same placement but with these modifications:

1. A fraction of nets from the netlist are selected, which sets the amount of pin-to-wire routing in a netlist.

2. For each selected net, a wire segment lying near the centre of the net's bounding box in the base routing of this net is chosen as the *target wire segment* for that net. (This is similar to the *boundary* wire in experiments 1 and 2, except that it is not constrained to be in the middle channel.)

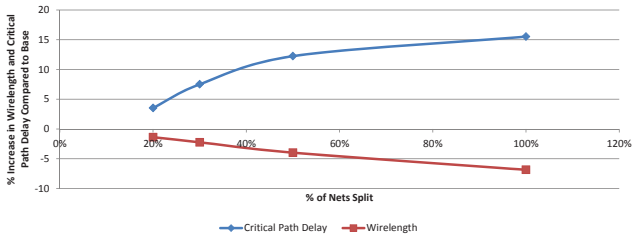

Fig. 9. Percentage variation in total wirelength and critical path delay as a function of amount of pin-to-wire routing

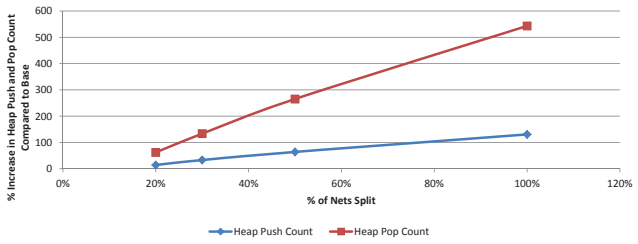

Fig. 10. Increase in Heap Push and Pop Count as a function of amount of pin-to-wire routing

3. The selected net will then be split and replaced by two faux nets. The sinks of the net that lie on the same side of the target wire segment as the source of the net form the sinks of the first faux net, with the target wire acting as a sink. The remaining sinks form the part of the second faux net, for which the target wire segment acts as a source.

4. The nets are routed in the same order as in the base routing.

The experiment was run with four cases, where the number of nets selected ranged through 20%, 30%, 50% and 100%.

Through this experiment we measured how an increasing fraction of selected nets, each of which contains the pin-to-wire and wire-to-pin connections described above, would affect the same metrics used in experiments 1 and 2.

Figure 9 plots the percentage increase in geometric mean (across all benchmarks, relative to the base case) for wirelength and critical path delay against the fraction of all nets that are split (as described above). Here we can see behaviour similar to Experiment 1, which shows the wirelength decreasing (from 1% to 7%), and the critical path delay increasing from 4% to 16%. The forced split again serves to prevent extra wire from yielding a reduced critical path. Figure 10 plots the percentage increase in heap push and pop counts as the percentage of split nets (and hence, pin-to-wire routing) increases. Here we also see a significant increase in the router effort, in the worst case, the heap push count is doubled while the heap pop count is sextupled. Clearly, the

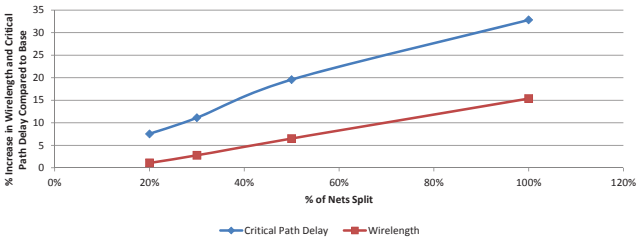

Fig. 11. Percentage variation in total wirelength and critical path delay as a function of amount of pin-to-wire routing - Harder Dispersed Routing

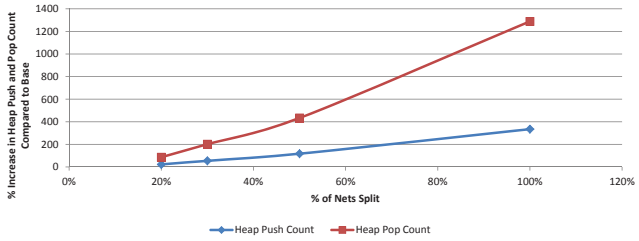

Fig. 12. Increase in Heap Push and Pop Count as a function of amount of pin-to-wire routing - Harder Dispersed Routing

router is having to work much harder to make these pin-to-wire connections.

It is interesting to note that only one circuit suffered an unroute across all of experiment 3 - the circuit *des* when 100% of the nets were selected (the reason was that there were too many targets in close proximity, which is a problem as described above). Also, we calculated the average fraction of all of the used segments that were being 'fixed' as target segments in each experiment. When the percentage of nets split ranges from 20%, 30%, 50% to 100%, the fraction of target wire segments that make up all of the routed segments ranges from 2.2%, 4.3%, 9.0% to 18%.

3.4. Experiment 4: Harder Dispersed Routing

In experiment 3, the target wire segment was set to be one of the wire segments in the net's original routing choen in the base case, lying near the centre of the net's bounding box. In this experiment, we explicitly select a target wire other than the original target wire but in roughly the same row/column location. This is similar to the increase in difficulty faced in experiment 2, and more like the expected routing-by-abutment situation as described there.

Figure 11 plots the percentage increase in geometric mean (across all benchmarks, relative to the base case) for wirelength and critical path delay against the fraction of all nets that are split (as described above). The results are quite different from experiment 3 – the wirelength increases from

1% to 15%, and the critical path delay increasing from 8% to 33%. Figure 12 plots the percentage increase in heap push and pop counts as the percentage of split nets (and hence, pin-to-wire routing) increases. There is also a significant increase in the router effort, in the worst case, the heap push count is quadrupled while the heap pop count increases by a factor of 14. Compared to experiment 3, eight circuits suffered unroutes when 100% of the nets were selected, while three circuits suffered unroutes when 50% nets were selected. No unroutes were observed when 20% or 30% nets were selected. This data indicated that the router is having to work extremely hard to make these pin-to-wire connections.

4. CONCLUSION

In this paper we have attempted to measure the difficulty faced by a routing algorithm and a specific routing architecture in the face of pin-to-wire routing. We have shown that there is a significant increase in wire length and critical path delay, and an occasional loss of routability. This suggests that pin-to-wire routing can be used under some careful circumstances with this price to be paid. We would also note that, due to the lower flexibility of wire segments as router targets, it is important to select the set of wire segment targets carefully, as described in Section 3.1.1. We have also shown the rate at which the difficulty increases as the quantity of pin-to-wire routing increases. In the future we will to explore the effect of routing architecture flexibility on these results, and also attempt to use routing-by-abutment in practical ways.

5. ACKNOWLEDGMENTS

We would like to thank Huimin (Hannah) Bian from Altera for some good suggestions in this work and Jason Luu for great support and advice.

6. REFERENCES

[1] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization)," in *ACM/SIGDA FPGA*, Feb. 1999, pp. 69–78.

[2] M. Guruswamy, *et al.*, "Cellerity: A fully automatic layout synthesis system for standard cell libraries," in *DAC*, June 1997, pp. 327–332.

[3] C.-T. Lin, *et al.*, "Modem floorplanning with abutment and fixed-outline constraints," in *IEEE ISCAS*, vol. 6, no. 6214–6217, May 2005.

[4] P. Athanas, *et al.*, "Wires on demand: Run-time communication synthesis for reconfigurable computing," in *FPL*, Aug. 2007, pp. 513–516.

[5] "Xilinx Partial Reconfiguration User Guide," Xilinx, July 2011. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/ug702.pdf

[6] Y. O. M. Moctar, *et al.*, "Reducing the cost of floating-point mantissa alignment and normalization in FPGAs," in *ACM/SIGDA FPGA*, Feb. 2012, pp. 255–264.

[7] S. Oldridge and S. Wilton, "Placement and routing for FPGA architectures supporting wide shallow memories," in *IEEE FPT*, Dec. 2003, pp. 154–161.

[8] "Stratix II Device Handbook," Altera, Apr. 2011. [Online]. Available: http://www.altera.com/literature/hb/stx2/stx2_sii5v1.pdf

[9] "Stratix IV Device Handbook," Altera, Dec. 2011. [Online]. Available: http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf

[10] "7 Series FPGAs Overview," Xilinx, Mar. 2012. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[11] D. Koch, C. Beckhoff, and J. Torresen, "Zero logic overhead integration of partially reconfigurable modules," in *SBCCI*, 2010, pp. 103–108.

[12] G. Lemieux, *et al.*, "Directional and single-driver wires in FPGA interconnect," in *IEEE FPT*, Dec. 2004, pp. 41–48.

[13] D. Paladino, "Academic clustering and placement tools for modern field-programmable gate array architectures," M.S. thesis, University of Toronto, 2008. [Online]. Available: https://tspace.library.utoronto.ca/handle/1807/11159

[14] J. Rose, V. Betz, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, Massachusetts: Kluwer Academic Publishers, 1999.

[15] "iFAR intelligent FPGA Architecture Repository," Feb. 2008. [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures/

[16] I. Kuon and J. Rose, "Area and delay trade-offs in the circuit and architecture design of FPGAs," in *ACM/SIGDA FPGA*, 2008, pp. 149–158.

[17] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0*, MCNC, Jan. 1991.

[18] A. R. Marquardt, "Cluster-based architecture, timing-driven packing and timing-driven placement for fpgas," M.S. thesis, University of Toronto, 1999.

[19] J. Luu, *et al.*, "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *ACM/SIGDA FPGA*, 2009, pp. 133–142.

[20] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for fpgas," in *ACM FPGA*, 1995, pp. 111–117.