Reducing the Cost of Floating-Point Mantissa Alignment and Normalization in FPGAs

Yehdhih Ould Mohammed Moctar[1] Nithin George[2]
Hadi Parandeh-Afshar[2] Paolo Ienne[2] Guy G. F. Lemieux[3] Philip Brisk[1]

[1]Department of Computer Science and Engineering
University of California, Riverside
{moctar, philip}@cs.ucr.edu

[2]School of Computer and Communication Sciences
École Polytechnqiue Fédérale de Lausanne (EPFL)

[3]Department of Electrical and Computer Engineering
University of British Columbia
lemieux@ece.ubc.ca

{nithin.george, hadi.parandehafshar, paolo.ienne}@epfl.ch

ABSTRACT
In floating-point datapaths synthesized on FPGAs, the shifters that perform mantissa alignment and normalization consume a disproportionate number of LUTs. Shifters are implemented using several rows of small multiplexers; unfortunately, multiplexer-based logic structures map poorly onto LUTs. FPGAs, meanwhile, contain a large number of multiplexers in the programmable routing network; these multiplexer are placed under static control of the FPGA's configuration bitstream. In this work, we modify some of the routing multiplexers in the intra-cluster routing network of a CLB in an FPGA to implement shifters for floating-point mantissa alignment and normalization; the number of CLBs required for these operations is reduced by 67%. If shifting is not required, the routing multiplexers that have been modified can be configured to operate as normal routing multiplexers, so no functionality is sacrificed. The area overhead incurred by these modifications is small, and there is no need to modify every routing multiplexer in the FPGA. Experiments show that there is no negative impact in terms of clock frequency or routability for benchmarks that do not use the dynamic multiplexers.

Categories and Subject Descriptors
B.7.1 [**Integrated Circuits**]: Types and Design Styles – *gate arrays.*

General Terms
Design, Performance.

Keywords
Field Programmable Gate Array (FPGA), Floating-point, Mantissa Alignment, Normalization.

1. INTRODUCTION
There is considerable interest in using FPGAs to accelerate scientific applications that are dominated by floating-point computations. As FPGAs have abundant spatial parallelism, the best strategy to optimize a floating-point datapath for an FPGA is to minimize the size of each operator, as doing so maximizes the number of operators that can be synthesized onto a device of fixed size; this, in turn, maximizes throughput. Recent work on floating-point datapath compilation for FPGAs has identified the wide shifts required for mantissa alignment and normalization as significant sources of area overhead [16, 17]. For IEEE single-precision floating-point addition, mantissa alignment requires a right shift from 0-24 bits on a 24-bit mantissa (which includes the "hidden '1'" in the most significant position); and normalization requires a left shift from 0-27 bits on a 27-bit mantissa that has been extended with three additional bits ("guard," "round," and "sticky,") which are used for rounding.

Shifters are generally implemented using several layers of multiplexers. In this paper, we consider FPGAs with 6-input lookup tables (6-LUTs), which can implement a 4:1 multiplexer in a single layer of logic level; since a 4:1 multiplexer has two control bits, it maps perfectly onto a 6-LUT. The 24- and 27-bit shifters required for mantissa alignment and normalization can be implemented with three layers of 6-LUTs: two layers of 4:1 multiplexers and one layer of 2:1 multiplexers. The 24- and 27-bit shifters require 72 and 81 6-LUTs respectively.

A typical single-precision floating-point adder requires 350 to 550 6-LUTs, depending on various implementation choices (e.g., support for all IEEE rounding modes, vs. support for just one). Regardless, of the details, the two shifters consume 153 6-LUTs, which is a significant portion of the overall area of the operator.

1.1 Static vs. Dynamic Multiplexing in FPGAs
Multiplexer-based logic maps inefficiently onto LUTs [24], and shifters are no exception. With that in mind, it is interesting to note that FPGAs contain many multiplexers in their routing networks; however, these multiplexers are not accessible to the user, as they are placed under static control of the FPGA's configuration bitstream. In principle, we would like to leverage some of these multiplexers to implement the shifters required for mantissa alignment and normalization in floating-point addition; this paper describes architectural mechanisms to accomplish this goal, along with the supporting CAD tools.

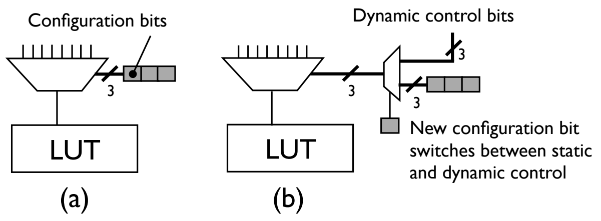

Configuration bits
Dynamic control bits
3
3
3
3
LUT
LUT
New configuration bit switches between static and dynamic control
(a)
(b)

Figure 1. A multiplexer in a traditional FPGA's routing network is placed under static control of the configuration bitstream (a); an extension that allows the user to configure the multiplexer to have static or dynamic control (b).

As a motivating example, consider the 8:1 multiplexer shown in Figure 1(a), which drives one input of a LUT; the other LUT inputs are driven by similar multiplexers, which are not shown. Three FPGA configuration bits drive the multiplexer's selection inputs. The purpose of this multiplexer is to provide some flexibility to the FPGA CAD tools—in particular, the router—when synthesizing a circuit onto the FPGA. In this case, there are 8 physical wires within the FPGA that can connect to this LUT input, via the multiplexer. One signal must route to that particular LUT input, and the router is given 8 possible wires to use. Once the route is complete, the configuration bits are set to select the chosen wire. This configuration is *static,* i.e., it does not change until the FPGA is reprogrammed. As there is no possibility to dynamically drive the selection inputs of this multiplexer, there is no possibility for the user to utilize it as an actual 8:1 multiplexer. As it is not architecturally visible, the typical user—who is not an FPGA architect—will be completely unaware of its existence.

Figure 1(b) illustrates a *Static-Dynamic Multiplexer (SD-MUX)*, which can be configured for either static or dynamic control. A 2:1 multiplexer now drives the configuration inputs of the 8:1 multiplexer. The 2:1 multiplexer can select either the control bits or a set of wires that are available to the user to provide dynamic control. An extra configuration bit drives the selection input of the 2:1 multiplexer, thus allowing the user to configure the 8:1 multiplexer to provide either static or dynamic control. This basic idea easily generalizes to a multiplexer with any number of inputs, as long as a sufficient number of control bits are provided.

When the SD-MUX is configured to provide static control, one signal can be routed to any of the 8 multiplexer inputs, and the configuration bits are set accordingly, as shown in Figure 2(a); as noted earlier, this provides flexibility to the router, as there is fierce competition for routing resources. When the SD-MUX is configured as a dynamic multiplexer, as shown in Figure 2(b), 8 signals are routed to the 8 multiplexer inputs in pre-specified order; e.g., if the user logic expects the multiplexer to select signal *x* when the selection bits are 010, then *x* must be routed to multiplexer input 010 in order to preserve this functionality; thus, the flexibility afforded to the router in the static case is sacrificed.

If we assume that the multiplexers in the routing network are 27:1 or larger, then 24 of them can implement mantissa alignment, and 27 can implement normalization. If we ignore the other LUT inputs, and configure the LUT to implement the identity function, then these two shifters can be implemented using 51 LUTs: a savings of 66.7% over the LUT-based implementation. This paper provides a solution that realizes this best-case savings.

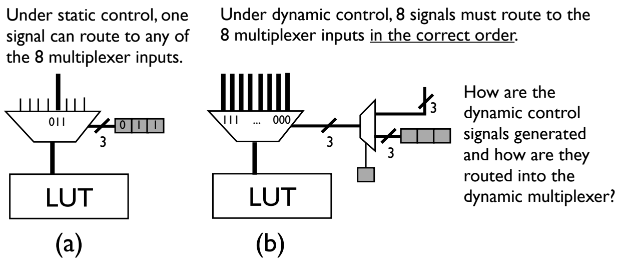

Under static control, one signal can route to any of the 8 multiplexer inputs.
Under dynamic control, 8 signals must route to the 8 multiplexer inputs in the correct order.
011
0 1 1
3
111 ... 000
3
3
3
How are the dynamic control signals generated and how are they routed into the dynamic multiplexer?
LUT
LUT
(a)
(b)

Figure 2. A static multiplexer provides flexibility to the router: in this case, 8 inputs are available to route one signal (a); a dynamic multiplexer offers no flexibility to the router, as all multiplexer inputs are used, and routes must be computed for the dynamic control signals as well.

1.2 Fundamental Challenges

The benefit of the SD-MUX is evident for circuits that require a significant amount of multiplexing; however, the introduction of dynamic multiplexers into an FPGA fabric comes at a non-negligible cost, and creates new challenges for physical design tools. The following issues must be addressed in order to justify the inclusion of dynamic multiplexers in an FPGA fabric:

(1) Given that FPGA routing networks consume as much as 90% of on-chip area [7], is the area overhead of replacing static multiplexers with SD-MUXes justifiable?

(2) When the SD-MUX is configured for dynamic control, how can the router overcome the lack of flexibility arising from the fact that 8 input signals must be routed to 8 multiplexer inputs in a pre-specified order, as shown in Figure 2(b)? How is routability achieved in the general case?

(3) How are the dynamic control bits generated, and how are they routed into SD-MUX, as noted in Figure 2(b)?

The answer to question (1) is that SD-MUXes can be introduced sparingly. Realistic user circuits may contain a significant amount of multiplexer-based logic that benefits from the presence of dynamic multiplexers; however, they also contain other logic that maps better onto existing FPGA logic and arithmetic resources, such as LUTs, carry chains, and DSP blocks. As an example, floating-point operators require a large number of LUTs for shifters, but also include components that would not benefit from dynamic multiplexers, such as fixed-point adders and multipliers and leading zero counters. Thus, there is no need to replace more than a handful of static multiplexers with SD-muxes.

The answer to questions (2) and (3) is solved through CAD algorithms. As shown in Figure 2(b), SD-MUXes configured as dynamic multiplexers impose significantly more constraints on the router than static multiplexers. To handle these constraints, the CAD tools extract *macro-cells*, which are subcircuits comprised of the user logic that will use the dynamic multiplexers, plus the immediately preceding logic layer as well. The macro-cells are placed-and-routed separately from the remainder of the circuit. This ensures that the router can satisfy all of the constraints imposed by the dynamic multiplexers without having the macro-cells compete with the remainder of the circuit for limited routing resources in congested areas. Placement then proceeds as normal, with some additional provisions to handle the macro-cells: the placer can move the entire macro-cell around within the FPGA, but cannot change the placement within the macro-cell. Once placement completes, routing resources within each macro-cell are reserved; unused routing resources within the perimeter of the macro-cell are not reserved, as their usage does not affect the macro-cell's functionality. The remainder of the circuit is then routed as normal, with the restriction that the reserved routing resources within each macro-cell are not perturbed, thereby ensuring its correct functionality.

1.3 Paper Goals and Organization

The remainder of this paper focuses on the architectural design choices to enable the SD-MUXes, the supporting CAD toolflow, and the experimental evaluation thereof. In particular, it is of great importance to ensure than the introduction of SD-MUXes does not significantly impair the performance of industrial-scale circuits that do not use the dynamic multiplexing functionality; fortunately, no adverse affects were observed in our experiments.

Section 2 describes the architectural modification that are necessary to integrate SD-MUXes into an FPGA, including the key design choices and trade-offs involved. Section 3 describes the CAD flow in greater detail. Section 4 presents our experimental evaluation, including an analysis of the benefits that SD-MUXes offer floating-point datapath circuits. Section 5 summarizes related work, and Section 6 concludes the paper.

2. ARCHICTECTURAL MODIFICATIONS TO INTEGRATE SD-MUXES INTO FPGAS

This section describes the necessary architectural enhancements to FPGAs to integrate SD-MUXes into the routing fabric. Starting with an overview a typical FPGA architecture (Section 2.1), we consider two locations in the FPGA to introduce the SD-MUXes (Sections 2.2 and 2.3). Lastly, we introduce the macro-cell and describe how a standard FPGA CAD flow can be modified to achieve routability (Section 2.4).

2.1 FPGA Architecture Overview

This paper targets an FPGA architecture based on the *Versatile Place and Route (VPR)* tool, which is publicly available from the University of Toronto [20, 21]. The user specifies several architectural parameters in a configuration file. VPR generates an FPGA architecture based on these parameters. For each LUT, VPR generates a *Basic Logic Element (BLE)*, depicted in Figure 3(a), which includes a bypassable flip-flop. The BLE can be configured to implement either combinational or sequential logic.

BLEs are grouped into clusters called *Configurable Logic Blocks (CLBs)*, with fast *Intra-Cluster Routing*, as shown in Figure 3(b). A *Connection Block (C Block)* interfaces the CLB with the global routing segments on either side. Figure 3(c) shows a floorplan of an *island-style FPGA*, which includes *Switch Blocks (S Blocks)* at the intersection points between horizontal and vertical routing channels, as well as I/O pads.

The user specifies several parameters that VPR uses to generate the logic and routing architecture:

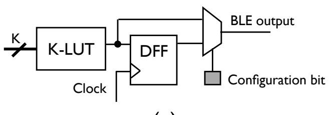
K: the LUT size (i.e., a K-LUT);

N: the number of LUTs per CLB;

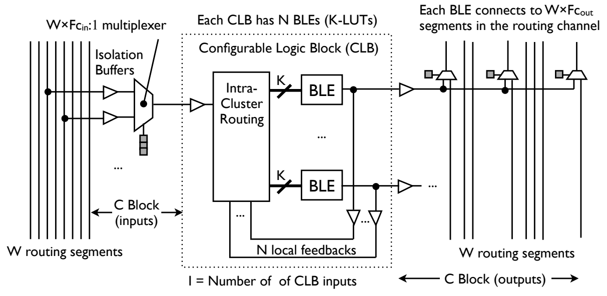
I: the number of CLB inputs;

W: the number of segments per routing channel; and

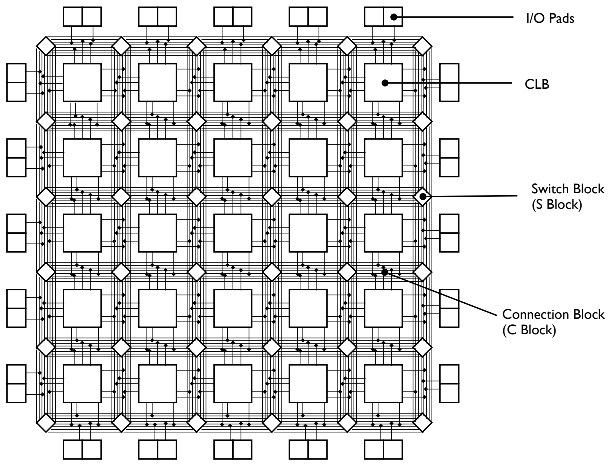
Fc_{in} and Fc_{out}: C Block connectivity parameters



(a)



(b)



(c)

Figure 3. A Basic Logic Element (BLE) (a); a Configurable Logic Block (CLB) contains several BLEs, with fast intra-cluster routing; the Connection Block (C Block) interfaces the intra-cluster routing inside the CLB with the global routing network (b); the floorplan of a generic island-style FPGA (c).

As shown in Figure 3(b), each input multiplexer in the C Block selects one of $W \times Fc_{in}$ wires in the adjacent routing channel on the left, and each BLE drives $W \times Fc_{out}$ segments in the adjacent routing on the right. Most FPGAs use single driver routing [18], so the C Block output is a conceptual description of the routing topology (i.e., which BLE outputs drive which segments in the channel). The multiplexers shown on the right-hand side of Figure 3(b) are actually implemented in the S Blocks, which are shown (without detail) in Figure 3(c). Figure 3(b) depicts inputs coming in from the left hand side of the CLB and outputs leaving to the right; in actuality, inputs and outputs may enter and exit from all four sides of the CLB, and the user may specify the percentage of inputs and outputs that enter and exit on each side.

Given all of these architectural parameters, VPR generates the interconnect topology algorithmically. A number of architectural parameters, such as those that describe the switch box, have been omitted from this discussion for brevity.

The intra-cluster routing depicted in Figure 3(b) is a crossbar that connects I inputs and N local feedbacks to the $K \times N$ LUT inputs in the CLB. VPR 5.0 implements intra-cluster routing as a full crossbar, which provides a connection between every CLB input and LUT input. Full crossbars are costly in terms of area and power, but guarantee routability: i.e., any combination of signals routed to CLB inputs can be routed to any desired combination of LUT inputs. Highly routable sparse crossbar topologies for intra-cluster routing have also been investigated in recent years [7, 19].

Ahmed and Rose determined that the ideal number of CLB inputs is $I = K \times (N+1)/2$, which is less than the total number of LUT inputs, $K \times N$. This suffices because many signals fan-out to multiple LUT inputs within a CLB after the FPGA has been configured. As each CLB input (other than LUT feedbacks) is driven by a $W \times Fc_{in}:1$ multiplexer, reducing the number of CLB inputs reduces the overall cost of the C Block, at the expense of some flexibility. In other words, N independent K-input logic functions cannot be packed into a CLB due to I/O limitations, despite the fact that the CLB has sufficient LUT capacity.

Recall that the goal of this paper is to replace static multiplexers in the routing network with SD-MUXes. In Figure 3(b), there are two locations where this is possible: the C Block (input), and intra-cluster routing, as discussed in the next two subsections.

2.2 Integrating SD-MUXes into the C Block

Example 1. To illustrate the integration of an SD-MUX into a C Block, let us consider a conditional swap, which has three inputs, I_0, I_1 and c, and two outputs, J_0 and J_1. The operation is:

$$J_0 = c\ ?\ I_1 : I_0 \quad , \quad J_1 = c\ ?\ I_0 : I_1 \qquad (1)$$

Figure 4 depicts a portion of the C Block that has been modified with two SD-MUXes to implement the conditional swap. A static multiplexer provides the control bit, while two SD-MUXES compute J_0 and J_1. In this particular example, $W = 8$ segments per channel and $Fc_{in} = 0.5$, i.e., each C Block multiplexer connects to 4 wires in the channel. Each of the three 4:1 multiplexers in the C Block are implemented using three 2:1 multiplexers; two of the 2:1 multiplexers have been replaced with SD-MUXes in Figure 4.

The C Block in Figure 4 imposes routing constraints that must be satisfied in order to deliver input signals I_0 and I_1 in the correct order to the SD-MUX inputs. In particular, I_0 and I_1 must be routed on routing segments w_6 and w_7; the order is irrelevant, i.e., either I_0 can be routed on w_6 and I_1 on w_7, or vice-versa; similarly, I_0 and I_1 must be routed on w_4 and w_5 as well. The condition bit, c, has greater flexibility: it can be routed on w_0, w_1, w_2, or w_3.

The placer and router must satisfy these constraints. Let F_0 and F_1 be K-input logic functions that compute conditional swap inputs I_0 and I_1. F_0 and F_1 must be synthesized on LUTs whose outputs collectively drive a subset of the wires that satisfy the aforementioned constraints. Moreover, this assumes that such a combination of LUTs actually exists. Although it may be possible to satisfy this constraint for a 3-input conditional swap operation, it will be much more difficult to satisfy for a 24- or 27-bit shifter.

Example 2. Consider a 4-bit left shift with rotation. The inputs are $I_0 \ldots I_3$ and the outputs are $J_0 \ldots J_3$; two control bits c_0 and c_1 specify the shift amount (0-3 bit positions). Once again, we assume that $W = 8$ and $Fc_{in} = 0.5$, and the C Block contains 4:1 multiplexers. As the shifter has four data inputs rather than two, each of the four data inputs, I_0 ... I_3 must connect to *exactly one* input of each C Block multiplexer in a pre-determined pattern.

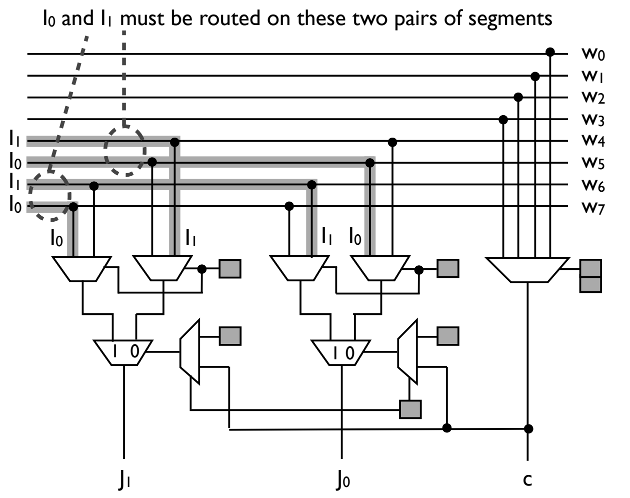
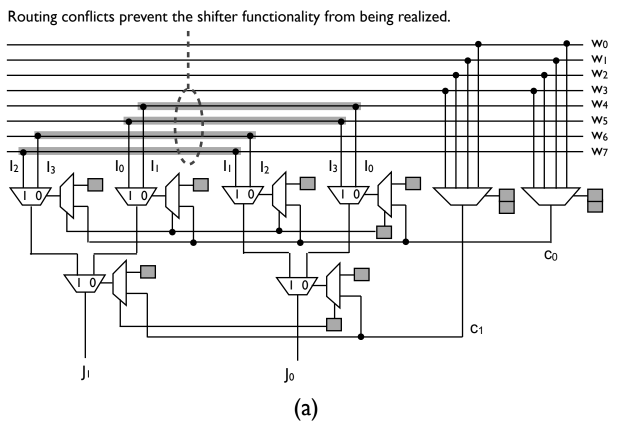


Figure 4. A C Block modified to implement a conditional swap by introducing two SD-MUXes. The requirements to deliver the correct signals to the SD-MUX inputs impose stringent routing constraints.

Figure 5(a) depicts a portion of the C Block that produces the lower-order data outputs, J_0 and J_1; however, the interconnection topology does not allow the design to be satisfied due to conflicts on the routing segments. For example, the multiplexer that produces J_0 requires I_0 to be routed on segment w_4, while the multiplexer that produces J_1 requires I_1 to be routed on the same segment concurrently. Similar conflicts occur on segments w_5, w_6, and w_7. In contrast, Figure 5(b) depicts an interconnect topology that eliminates the routing conflicts; of course, this topology *only* satisfies a 4-bit left shift with rotation, and would not necessarily be helpful for some other type of multiplexer-based circuit.

As mentioned earlier, dynamic multiplexers impose strict ordering constraints on the signals that are connected to their inputs. The examples shown in Figures 4 and 5 demonstrate that these constraints are propagated into the global routing network when SD-MUXes are integrated into the C Block. The ability to implement very simple multiplexing circuits, as shown in Figures 4 and 5, is dependent on the interconnect topology between the CLBs and the routing network; this interconnect topology depends on parameters W, Fc_{in} and Fc_{out}, and the algorithm [4] that generates the routing network from these parameters.

Although it may be possible to modify the routing network generation algorithm to favor certain interconnect topologies, it is difficult to determine whether the basic idea will generalize to larger structures. For example, in a 27-bit shifter, input bit I_0 will fan out to 27 outputs, $J_0 \ldots J_{26}$. This means that a single routing segment or a subset of segments driven by the same BLE must individually or collectively fan-out to 27 pre-specified C Block SD-MUX inputs, all in close quarters. Similarly I_1 will need to fan-out to 26 pre-specified C Block SD-MUX inputs, etc. Moreover, this must be done with parameters that are representative of commercial FPGAs, e.g., $N = 10$, $W = 300$, $Fc_{in} = 0.15$, and $Fc_{out} = 1/N = 0.1$. The likelihood of success in this case is too low to be considered realistic; consequently, we conclude that the C Block is not a particularly promising location to integrate SD-MUXes into the routing fabric.


Routing conflicts prevent the shifter functionality from being realized.
w0
w1
w2
w3
w4
w5
w6
w7
I2
I3
I0
I1
I1
I2
I3
I0
c0
c1
J1
J0
(a)

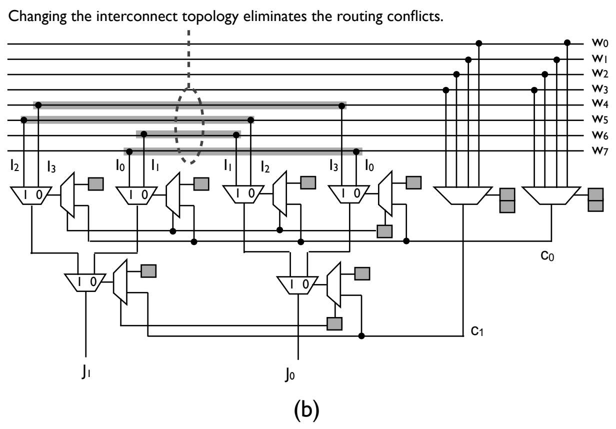

Changing the interconnect topology eliminates the routing conflicts.
w0
w1
w2
w3
w4
w5
w6
w7
I2
I3
I0
I1
I1
I2
I3
I0
c0
c1
J1
J0
(b)

Figure 5. A conflict in the interconnect topology makes it impossible to implement a 4-bit rotator using SD-MUXes in the C Block (a); changing the interconnect topology can eliminate the conflict (b).

2.3 Integrating SD-MUXes into Intra-Cluster Routing

Alternatively, we can introduce SD-MUXes into the intra-cluster routing instead of the C Block. The primary advantage of this approach is that it eliminates the routing constraints that arise due to the interconnection topology between the routing segments and the C Block. Instead, the interconnection topology constraints are internal within the intra-cluster routing. Signals that drive a specific SD-MUX input for dynamic multiplexing, as shown in Figure 6, are routed to pre-selected CLB inputs. Each pre-selected CLB input connects to one of the SD-MUX inputs: some to the data inputs, and others to the selection inputs.

Figure 6 shows an example of a 4:1 SD-MUX integrated into the intra-cluster routing; a significant portion of the intra-cluster routing is omitted from Figure 6 to conserve space. Two CLB inputs provide dynamic control (they may also drive other multiplexers, which are not depicted in the figure); control signals c_0 and c_1 must be routed to these two inputs. The other four CLB inputs drive the data inputs of the SD-MUX. The input signals are routed to these four CLB inputs in a specific order, e.g., the SD-MUX selects input I_0 if $c_1c_0 = 00$. Thus, the connection topology between CLB inputs and SD-MUX inputs determines which signals must be routed to each pre-selected CLB input.

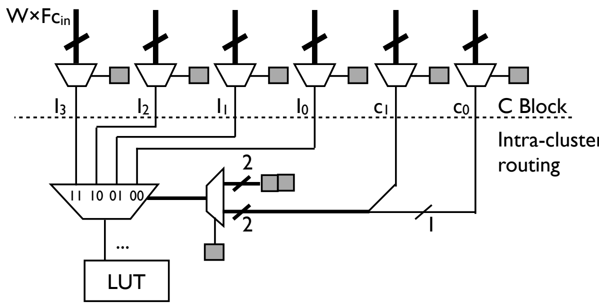
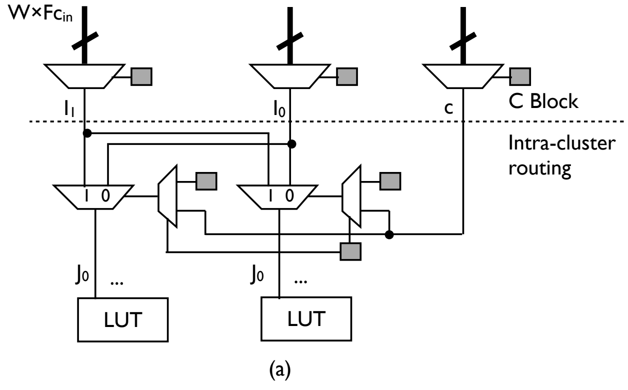


Figure 6. Integrating an SD-MUX into intra-cluster routing imposes a strict ordering on the signals that are routed to the CLB inputs.



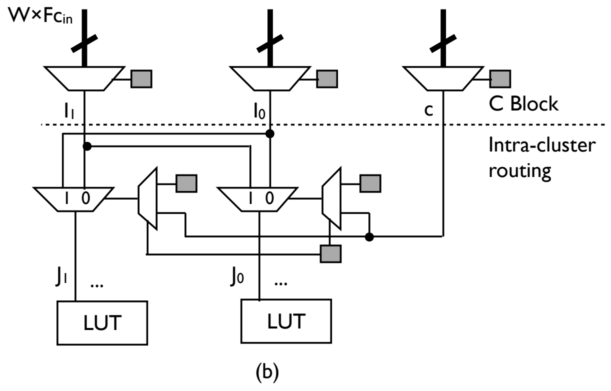


Figure 7. Integration of SD-MUXes into the intra-cluster routing. The interconnect topology may force both SD-MUXes to implement the same logic function when configured to implement dynamic control (a); rearranging the topology enables the SD-MUXes to implement different functions (b).

In Figure 6, any 4-input multiplexer can be realized by permuting either the control or the data bits; however, additional restrictions are imposed when we consider multiple-output functions because each CLB input may connect to multiple SD-MUX inputs.

Example 3. Let us reconsider the conditional swap operation from Example 1; this time, we want to implement it using SD-MUXes in the intra-cluster routing rather than the C Block. Figure 7(a) shows an initial attempt. Due to the interconnection topology within the intra-cluster routing, both SD-MUXes conditionally select the same input bit, i.e., they both compute logic function J_0.

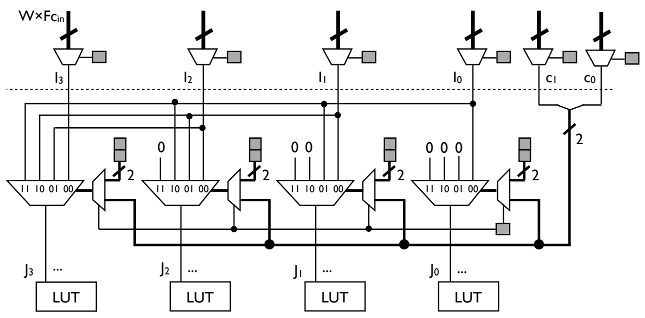


Figure 8. Intra-cluster routing with SD-MUXes modified to support a 4-bit left shift.

By swapping the order of I_0 and I_1 at the CLB inputs, then this intra-cluster routing topology would compute J_1, rather than J_0; however, by changing the topology, as shown in Figure 7(b), the two SD-MUXes compute logic functions J_0 and J_1, respectively. In this case, swapping the order of I_0 and I_1 at the inputs would likewise swap the order of J_0 and J_1 at the outputs.

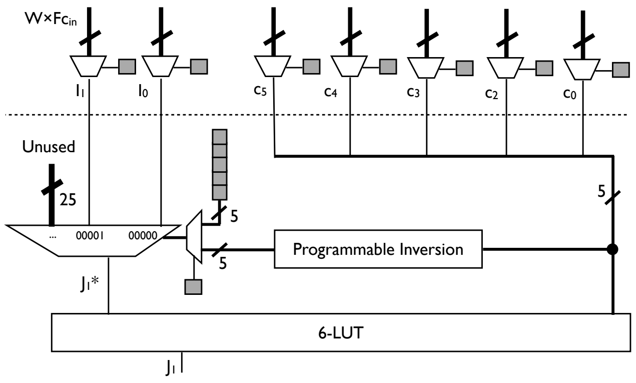
Example 4: Figure 8 illustrates intra-cluster routing with SD-MUXes that can implement a 4-bit left shifter (bits shifted in are set to zero). The basic interconnection pattern shown here easily generalizes to a larger shifter sizes. In this case, the SD-MUXes implement all of the shifting functionality; the LUTs are configured to pass the SD-MUX outputs through unmodified.

In Figure 8, many of the SD-MUX inputs in are '0' bits. It is not immediately clear how these bits should be handled. It would be unrealistic to extend some of the SD-MUXes to account for a large number of '0' bits, e.g., in a K-bit shifter, the SD-MUX that computes least significant output bit J_0 requires $K-1$ '0' bits, the SD-MUX that computes J_1 requires $K-2$ '0' bits, etc.; the area overhead required to support larger shifters would be prohibitive.

Another issue is that the routing network may invert signals en route. The LUT sink is usually reprogrammed to compensate if some of its inputs arrive with the wrong polarity. SD-MUXes, however, are not programmable in this respect. One possibility is to add programmable inversion at the shifter inputs; however, this incurs significant area overhead. Another option is to reprogram the previous layer of LUTs that generate the shifter inputs to compute the complement of its logic function; however, this logic layer may have a large fan-out, where some fan-out bits are inverted and others are not, rendering this approach ineffective.

We can solve both the '0' SD-MUX input bit problem and the inversion problem by using LUTs in conjunction with the SD-MUXes. Each SD-MUX output drives a LUT input; we can then route the control bits to the remaining LUT inputs. The LUT is then programmed to invert the SD-MUX output if the selected input arrives in inverted form. The LUT is also programmed to output a '0' for the appropriate control bit combinations (e.g., $c_1c_0 = \{01, 10, 11\}$ for J_0 in Figure 8), which eliminates the need to route '0' bits to the SD-MUX inputs.

The CLBs in modern high performance FPGAs contain 6-LUTs; this limits the number of control bits that can be supported using this approach to 5 or less, which, in turn, limits the maximize SD-MUX size to 32:1. This suffices for the 24- and 27-bit shifters used for single-precision floating-point mantissa alignment and normalization. Figure 9 illustrates the preceding discussion for the second least significant bit, J_1 of a 27-bit shifter.



I_0 Not Inverted, I_1 Not Inverted						
c_4	c_3	c_2	c_1	c_0	J_1^*	J_1
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	1	1	1
--	--	--	--	--	--	0

I_0 Inverted, I_1 Not Inverted						
c_4	c_3	c_2	c_1	c_0	J_1^*	J_1
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	1
--	--	--	--	--	--	0

I_0 Not Inverted, I_1 Inverted						
c_4	c_3	c_2	c_1	c_0	J_1^*	J_1
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	1
0	0	0	0	1	1	0
--	--	--	--	--	--	0

I_0 Inverted, I_1 Inverted						
c_4	c_3	c_2	c_1	c_0	J_1^*	J_1
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	1
0	0	0	0	1	1	0
--	--	--	--	--	--	0

Figure 9. A LUT in conjunction with an SD-MUX solves the problems of inverted input bits and generates '0' outputs when appropriate. This example is the second least significant output bit, J_1, of a 27-bit left shifter. Four truth tables are possible, depending on whether I_0 and I_1 are inverted. Programmable inversion is necessary for the five control bits.

CLB parameters also limit the size of the SD-MUXes that can be introduced. The intra cluster routing has a total of $I+N$ inputs and $N\times K$ outputs. The inputs are the I CLB inputs provided by the C Block plus N LUT feedbacks from within the CLB. The cluster contains N K-LUTs; each LUT input is an output of the intra-cluster routing. A typical modern high-performance FPGA has $N = 8$, $K = 6$, and $I = K\times(N+1)/2 = 27$, using the formula provided by Ahmed and Rose [1]. To support a 27-bit shifter, we need to increase I to 32, to account for the control signals.

In VPR 5.0, the intra-cluster routing is a full crossbar. Given these parameter values, the intra-cluster routing would be composed of 48 40:1 multiplexers. Modern FPGAs, however, use sparsely populated crossbars [7, 19]. Depending on the population density of the sparse crossbar, the multiplexers may be smaller than 27:1. In this case, we would either need to limit the shift amount in accordance with the multiplexer size, or introduce SD-MUXes that are larger than the pre-existing static multiplexers; this latter option is unfavorable, because it introduces asymmetry in terms of delays: i.e., the delay through a statically configured SD-MUX is greater than the delay through a standard static multiplexer, which could affect performance and complicate routing.

The interconnection topology (i.e., which CLB inputs connect to exactly which SD-MUX inputs) has a significant impact on our ability to implement shifters in the intra-cluster routing; this was illustrated quite clearly by Figure 7. Figure 8 illustrates the general interconnect topology pattern required for a left shifter (which easily generalizes to more than 4 inputs), and a left shifter can implement a right shifter by reversing the order of the inputs. Shifters that perform rotation (e.g., Figure 5) require a different topology as they do not shift-in zeroes. To summarize, the topology must account for ordering constraints on SD-MUX inputs in order to ensure correctness.

Lastly, we do not advocate the introduction of SD-MUXes into every CLB, as the vast majority of CLBs in a given FPGA will not be configured to implement dynamic multiplexing circuits in most realistic designs. CLBs containing SD-MUXes are a new form of heterogeneity, similar in principle to the introduction of DSP blocks and block RAMs in past FPGAs. As a rough estimate, we suggest at most 10% of the CLBs in an FPGA should be enhanced with SD-MUXes, and that those that are enhanced should be laid out in columns within the FPGA; the column-based layout echoes the way that DSP blocks and block RAMs are currently laid out in FPGAs, and therefore makes intuitive sense.

2.4 Ensuring Routability with Macro-Cells

Consider a 27-bit shifter implemented with SD-MUXes integrated into the intra-cluster routing. In accordance with prior notation, let $I_0 \ldots I_{26}$ and $J_0 \ldots J_{26}$ denote the shifter inputs and outputs, and let $c_0 \ldots c_4$ denote the control bits. This is a total of 32 inputs (including control bits) and 27 outputs. Eight CLBs, $CLB_0 \ldots CLB_7$ realize the shifter. LUT L_i computes shifter input I_i, LUT S_i computes shifter output J_i, and LUT C_i computes control bit c_i.

Figure 10 depicts the interconnection pattern for the 27-bit shifter. The structure depicted in Figure 10 is called a *macro-cell*, because the LUTs and CLBs are pre-placed and routed. Without loss of generality, if the placer (generally an iterative improvement algorithm) randomly moves L_5 to a new CLB, the likelihood is quite small that a legal route will be found that delivers shifter input I_5 to the pre-specified CLB inputs in CLB_5, CLB_6, and CLB_7. By fixing the locations of the LUTs relative to one another in the macro-cell, routability is achieved.

3. CAD SUPPORT FOR MACRO-CELLS

We used VPR 5.0 [20] for architectural simulation, placement, and routing, T-VPack for packing [22], and ABC for logic synthesis and technology mapping [3].

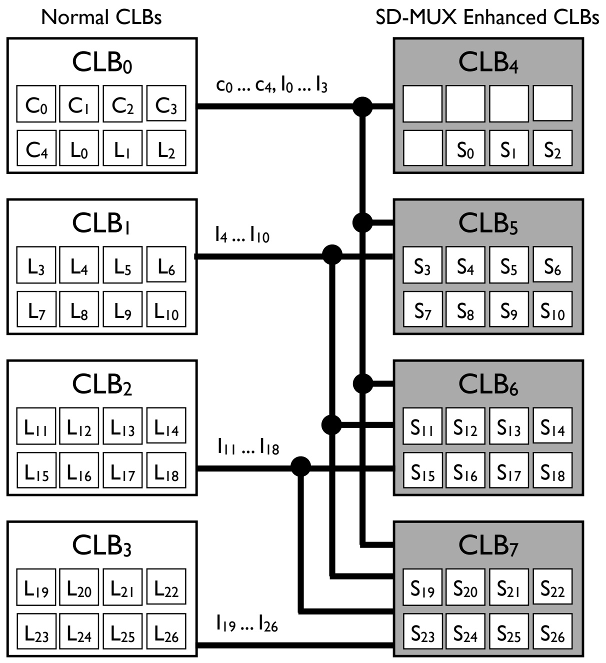


Figure 10. A macro-cell for a 27-bit shifter.

3.1 Programming Model, Assumptions, and Technology Mapping

We assume that the programmer will add annotations to the HDL code to specify when to configure the programmable macro-cell as a shifter, similar to how DSP blocks and carry chains are used. The technology mapper explicitly binds the annotated shifters to macro-cells rather than mapping them to LUTs. Large shifters are decomposed into smaller ones if macro-cell capacity is exceeded. Next, we extract the layer of LUTs that precedes each shifter, e.g., LUTs $L_0 \ldots L_{23}$ in Figure 10. The structure of the macro-cell effectively pre-packs, pre-places, and pre-routes these subcircuits.

3.2 Macro-cell Placement and Routing

VPR's router, which is based on PathFinder [23], assumes that CLB intra-cluster routing is a full crossbar. Any path from the source to a CLB input can route a net: the crossbar connects all CLB inputs to all LUT inputs. We modified VPR to allow the user to specify specific CLB inputs pins as targets for certain sinks. VPR can find a legal route for a macro-cell, establishing a path from the LUT source that computes each net to all of its pre-specified inputs in the second macro-cell layer. VPR successfully routed 24- and 27-bit shifters in macro-cells using this approach.

Macro-cells are placed-and-routed offline, prior to the rest of the circuit. Placement of the shifter onto SD-MUXes within the macro-cell is deterministic. Placement of the LUT layer preceding the shifter is more flexible: any placement that successfully routes all nets within the macro-cell suffices. We try to pack the LUTs tightly into a small number of CLBs in the vicinity of the shifter.

3.3 Global Placement and Routing

Extensive modifications were made to VPR's placer [21] in order to handle macro-cells. The input is a netlist, which may or may not contain macro-cells, and an architectural description of the FPGA in which certain columns have been annotated to indicate CLBs that have been enhanced with SD-MUXes.

Each shifter in the netlist, along with the layer of preceding LUTs, is placed onto a macro-cell. Each macro-cell is routed up-front. SD-MUXes in the remaining (unused) macro-cells are configured as normal CLBs, similar to how shadow clusters are used [11]. The placer considers all other CLBs to be functionally equivalent.

VPR's placer uses simulated annealing. We implemented two placement strategies. In the first, we place shifters onto macro-cells and fix their placement; the placer moves normal soft logic clusters around, but does not perturb the placement of the shifters onto macro-cells. The second option relaxes this constraint, and moves both soft logic clusters around the FPGA and may also move any shifter onto an unused macro-cell.

Macro-cells configured as shifters are similar to DSP blocks from the perspective of the CAD tools. The difference is that unused logic and routing resources within each macro-cell, after it has been placed-and-routed, remain available to the global placer and router and can be used by the rest of the circuit.

4. EXPERIMENTAL RESULTS

Our experimental goals are twofold. Firstly, we wish to quantify reduction in LUT count that can be achieved by synthesizing shifters onto SD-MUX enabled macro-cells. Secondly, we wish to ensure that the inclusion of macro-cells does not adversely affect routability for industrial-scale benchmarks.

4.1 Floating-Point Operators

We consider a set of single-precision multi-operand floating-point adders that have already been optimized for area. These operators are similar to those produced by Altera's floating-point datapath compiler [16, 17], which removes redundant normalizations. We used designs published by Verma et al. [25], which were slightly smaller than those produced by Altera's compiler. We used the smallest design approach, which implemented the internal fixed-point multi-operand addition using a tree of 3-input adders.

For a *K*-input adder, we denormalize *K-1* mantissas using shifters; the mantissa corresponding the largest exponent is not shifted. Normalization is only applied once, at the output of the operator.

For 2, 4, 8, and 16-input adders, Figure 11 reports that the area savings (in terms of Altera's ALMs) obtained by the macro-cell range from 25% to 32%. Assuming that the number of LUTs and CLBs in an FPGA are fixed, this means that 33-40% more operators can be packed into an area of fixed size when macro-cells are used to implement shifters.

We did not measure the effect of the macro-cells on critical path delay or pipeline depth of the adders. The throughput of floating-point data paths is driven mostly by spatial parallelism; reducing the area of an operator increases the number of operators that can be synthesized on a fixed area device. The area savings reported in Figure 11, thus, translate indirectly into increased throughput for parallel floating-point data paths that use these operators.

4.2 Experimental Setup: VPR

We modeled an FPGA enhanced with macro-cells using VPR 5.0 [20]. We did not use VPR 6.0, which is now part of the Verilog-To-Routing (VTR) flow, for these studies because it did not have timing models in-place at the time this work was performed. As our baseline, we took one of the VPR architecture files from the iFAR repository [14, 15]. Table 1 lists the baseline parameters for our architecture. CLB inputs and outputs are evenly distributed around all four sides of the CLB.

VPR explicitly models a C Block, but does not model the intra-cluster routing; as it is a full crossbar, only its delay is modeled. We do not model SD-MUXes explicitly. Our experiments strive to show that macro-cells, which reserve a non-trivial quantity of routing resources in localized areas, do not adversely affect the ability to route large-scale circuits that contain shifters.

Macro-cells are organized as vertical columns when they are introduced into the FPGA. The motivation is to mimic the layout of modern FPGAs. For example, logic clusters are generally laid out as columns; so are DSP blocks, block RAMs, etc. Only a small proportion of CLBs in the FPGA contain SD-MUXes.

For each experiment, we placed each benchmark once and routed it three times using different random number seeds. The delay for each benchmark is the average delay of the three runs. This reduces the noise in our delay results as different random number seeds can yield significantly different routing results.

4.3 Benchmarks

We selected the ten largest IWLS 2005 benchmarks [10], which are described in Table 2, to evaluate the impact of the macro-cell on large-scale applications. Using VPR, we synthetically added macro-cells (shifters) to these benchmarks; our goal is to ascertain whether these shifters, when pre-placed and routed onto macr-cells, adversely affect area, delay, and routability.

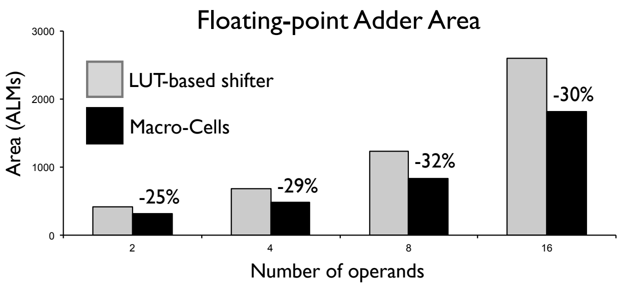


Figure 11. The area savings obtained by macro-cell based implementations of 24- and 27-bit shifters used for mantissa alignment and normalization on four optimized multi-operand single-precision floating-point adders.

Table 1. FPGA architectural parameters

Parameter	Value	Parameter	Value
LUT Size	6	Fc input	0.15
Cluster Size	8	Fc output	0.1
Channel Width	96	Technology*	65nm CMOS
Cluster Inputs	36	Tile Area**	18940

* Berkeley predictive models ** Min-width transistors

Table 2. Ten largest IWLS 2005 benchmarks

Benchmark	Description
ac97_ctrl	Interface to external AC 97 audio codec
aes_core	Advanced Encryption Standard (AES)
des_perf	16-cycle pipelined DES/3-DES Core
ethernet	10/100 Mbps IEEE 802.3/802.3u MAC
mem_ctrl	Embedded memory controller
pci_bridge32	Bridge interface to PCI local bus
systemcaes	Area-optimized AES implementation
usb_func	USB 2.0 compliant core
vga_lcd	Embedded VGA/LCD controller
wb_conmax	Wishbone Interconnect Matrix IP Core

We modified each benchmark's netlist to include 20, 40, 60, 80, and 100 shifters, which are connected at arbitrarily chosen points to ensure that they are not completely disjoint from the remaining logic. In VPR, we pre-allocated macro-cell columns and pre-placed the shifters and preceding layers of logic onto them. We pre-routed the macro-cells, and marked the routing resources used as unavailable. Our primary concern was that locking down these resources up-front would adversely affect the quality of the routes obtained for the remainder of the circuit; fortunately, practically no degradation was observed.

VPR generates a custom FPGA for each benchmark, based on its demand for logic and routing resources. Each benchmark is repeatedly placed and routed, varying the channel width each time; VPR converges onto the minimum channel width (W_{min}) for which a legal route can be found. The average W_{min} obtained by VPR across all benchmarks (with no macro-cells) here is 84.4.

Figure 12 reports the area of each benchmark with a varying number of shifters. The area is reported in terms of minimum-width transistors; this accounts for the fact that CLBs that have been augmented with SD-MUXes are larger than regular CLBs.

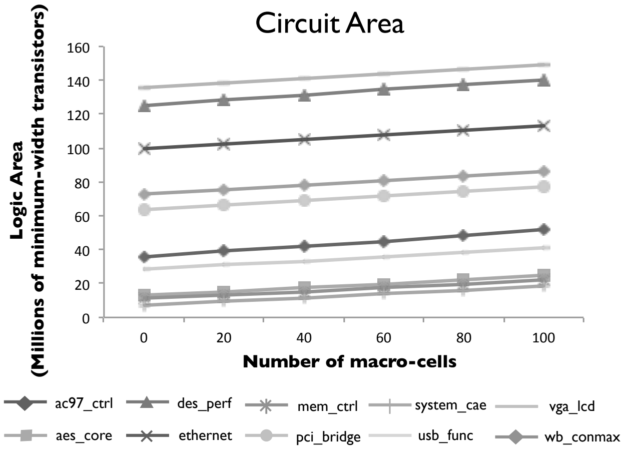


Figure 12. Area of the 10 circuits synthesized using VPR 5.0.

4.4 Routability

Figure. 13 reports the critical path delay of the IWLS benchmarks with a varying number of macro-cells; we observe practically no impact on critical path delay from the inclusion of as many as 100 shifters per benchmark. As noted in Section 3.3, we considered two different placement strategies: a *constrained* strategy in which the logic placed onto macro-cells is fixed a-priori, and an *unconstrained* strategy in which the placer can move the macro-cell logic (the shifter, and logic layer preceding it) onto any macro-cell. The results reported in Figure. 13 are for the constrained strategy; we observed that the unconstrained strategy produced essentially identical results, where the differences in delays for each data point are in the range of tens of pico-seconds.

Figure. 14 shows that introducing macro-cells may adversely affect W_{min}, as each macro-cell requires some routing resources. For many benchmarks, W_{min} steadily increases when the number of macro-cells ranges from 20 to 80, but decreases rapidly from 80 to 100. The reason for this observation is that VPR automatically generates an FPGA that is sized to a specific application; based on the number of CLBs used and I/O pads required, VPR generates the smallest square FPGA that can provide sufficient resources. VPR then repeatedly places and routes the circuit to determine W_{min}.

Many of the IWLS benchmarks are I/O bound, so CLB utilization is relatively sparse, and there is relatively little congestion in the routing network. Each macro-cell that is added increases CLB utilization, and introduces congestion, which increases W_{min}. If we assume a fixed-size FPGA, eventually, the inclusion of more macro-cells will cause utilization to exceed 100%. VPR then generates a larger FPGA, with much lower utilization; consequently, the benchmark circuit routes much easier, and W_{min} is reduced. This is precisely what occurred, for example, for benchmarks aes_core and des_perf (and a few others) between 80 and 100 macro-cells in Figure 14. It is important to recall that these benchmarks are synthetic. A floating-point operator, in contrast, would contain shifters and use the available macro-cells. Moreover, W_{min} as reported in Figure 14 is much smaller than the routing channel width of commercially available FPGAs.

These experiments demonstrate that macro-cells are quite useful for benchmarks that contain shifters, while their presence will not adversely affect other benchmarks that do not contain shifters.

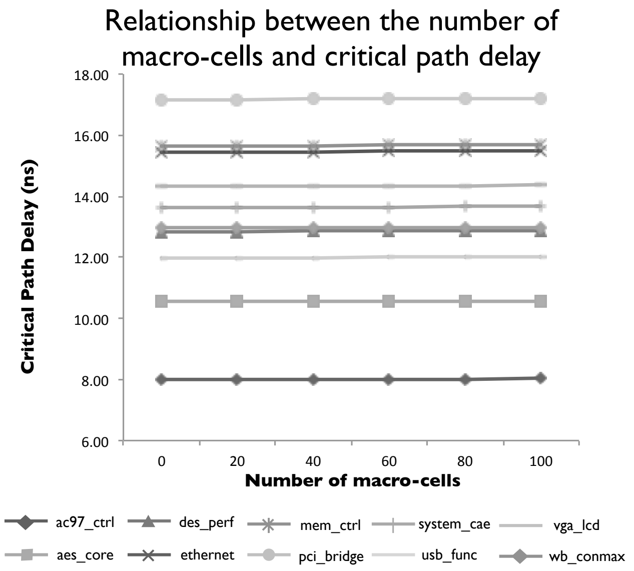


Figure 13. Introducing as many as 100 macro-cells into the benchmarks does not increase the critical path delay.

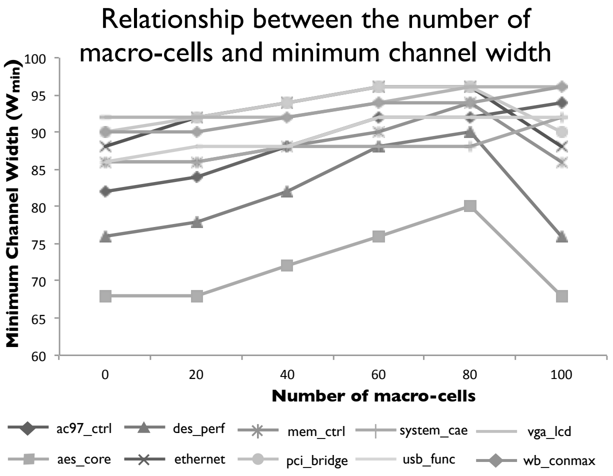


Figure 14. Introducing macro-cells into the benchmarks does have some affect on channel width; however, the dimensions of the FPGA change as well, as the number of macro-cells changes each benchmark's demand for logic resources.

5. RELATED WORK

The goal of this work is to reduce the cost of mantissa alignment and normalization in floating-point operations. One alternative is to integrate floating-point units as hard blocks [2, 5, 9]; however, applications that are not floating-point intensive will be unable to use these blocks. To date, FPGA vendors do not sell device families with dedicated blocks for floating-point applications. Beauchamp et al. [2] advocate integrating hard shifters or 4:1 multiplexors in parallel with FPGA logic; however, when the shifters are not used, the nearby routing resource are wasted; and when the 4:1 multiplexors are used, significant routing resources are still required to form large shifters.

Shifters and multiplexers can be synthesized onto multipliers in the DSP blocks [8, 12], and Xilinx has added 17-bit barrel shifters to their DSP48E1 blocks [26]; however, a DSP block used for shifting, cannot perform other operations. Benchmarks that require multiplication and shifting can still benefit from FPGAs containing DSP blocks and macro-cells.

Floating-point datapath compilers use arithmetic transformations to synthesize floating-point operations efficiently on FPGAs [6, 16, 17]; reducing the cost of normalization is one of their goals. These compilers achieve better performance and logic density than using 2-input operators, but they sacrifice IEEE compliance. Our approach is amenable to IEEE-compliant operators.

A patent by Kaviani (*Xilinx*) [13] exposes the selection bits of C block multiplexers to the programmer; the idea is similar to Xilinx Virtex FPGAs, which do not have intra-cluster routing. No CAD tools are described, so the affect on routability is unknown.

6. CONCLUSION

The macro-cells introduced in this paper can implement 27-bit shifters for single-precision floating-point mantissa alignment and normalization. The macro-cells reduce the area of floating-point addition clusters by up to 32%, which increases the number of operators that can be synthesized into a fixed-area device. This aligns well with the strategy employed by Altera's floating-point datapath compiler [16, 17]. Our experiments show that macro-cells do not adversely affect routability for benchmarks that do not contain shifters. Future work will look to integrate macro-cells with FPGAs that contain sparse intra-cluster routing, and to see whether it is possible to extend them into the C Block.

REFERENCES

[1] Ahmed, E., and Rose, J. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. VLSI*, vol. 12, no. 3, March, 2003, pp. 288-298. DOI= http://dx.doi.org/10.1109/TVLSI.2004.824300

[2] Beauchamp, M. J., Hauck, S., Underwood, K. D., and Hemmert, K. S. Architectural modifications to enhance the floating-point performance of FPGAs. *IEEE Trans. VLSI*, vol. 16, no. 2, Feb. 2008, pp. 177-187. DOI= http://dx.doi.org/10.1109/TVLSI.2007.912041

[3] Berkeley Logic Synthesis and Verification Group. "ABC: A system for sequential synthesis and verification.: December 2005 release. URL= http://www.eecs.berkeley.edu/~alanmi/abc

[4] Betz, V., and Rose, J., "Automatic generation of FPGA routing architectures from high-level descriptions," ACM/SIGDA Int. Symp. FPGAs (FPGA '00), pp. 175-184, Feb. 10-11, 2000, DOI= http://doi.acm.org/10.1145/329166.329203

[5] Chong, Y. and Parameswaran, S., "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," ACM/SIGDA Int. Symp. FPGAs (FPGA '09), pp. 171-180, Feb. 22-24, 2009, DOI= http://doi.acm.org/10.1145/1508128.1508155

[6] de Dinechin, F., Klein, C., and Pasca, B., "Generating high-performance custom floating-point pipelines," Int. Conf. Field Programmable Logic and Applications (FPL '09), Aug. 31- Sept. 2, 2009. DOI=http://dx.doi.org/10.1109/FPL.2009.527255/

[7] Feng, W. and Kaptanoglu, S. Designing Efficient Input Interconnect Blocks for LUT Clusters Using Counting and Entropy. *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, Mar. 2008, pp. 1-28. DOI= http://doi.acm.org/10.1145/1331897.1331902

[8] Gigliotti, P., "Implementing barrel shifters using multipliers," XAPP – Application Note: Virtex II Family, pp. 1-4, Aug., 2004. URL= http://www.xilinx.com/support/documentation/application_notes/xapp195.pdf

[9] Ho, C. H., et al., Floating-point FPGA: architecture and modeling. *IEEE Trans. VLSI*, vol. 17, no. 12, Dec. 2009, pp. 1709-1718. DOI= http://dx.doi.org/10.1109/TVLSI.2008.2006616

[10] IWLS 2005 benchmarks. URL= http://iwls.org/iwls2005/benchmarks.html

[11] Jamieson, P., and Rose, J., "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," *IEEE Trans. CAD*, vol. 18, no. 12, Dec. 2010, pp. 1696-1709. DOI = http://dx.doi.org/10.1109/TVLSI.2009.2026651

[12] Jamieson, P., and Rose, J., "Mapping multiplexers onto hard multipliers in FPGAs," 3rd Int. IEEE Northeast Workshop on Circuits & Systems (IEEE-NEWCAS '05), pp. 323-326, June 19-22, 2005. DOI= http://dx.doi.org/10.1109/NEWCAS.2005.1496692

[13] Kaviani, A., FPGA with improved structure for implementing large multiplexors. U.S. patent, no. US 6,556,042 B1, Apr. 29, 2003.

[14] I. Kuon and J. Rose, "Area and delay trade-offs in the circuit and architecture design of FPGAs," ACM/SIGDA Int. Symp. FPGAs (FPGA '08), pp. 149-158, Feb. 24-26, 2008, DOI= http://doi.acm.org/10.1145/1344671.1344695

[15] I. Kuon and J. Rose, "Automated transistor sizing for FPGA architecture exploration," ACM/IEEE Design Automation Conference (DAC '08), pp. 792-795, June 8-13, 2008, DOI= http://doi.acm.org/10.1145/1391469.1391671

[16] Langhammer, M., "Floating point datapath synthesis for FPGAs," Int. Conf. Field Programmable Logic and Applications, (FPL '08), pp.355-360, Sept. 8-10, 2008. DOI= http://dx.doi.org/10.1109/FPL.2008.4629963

[17] Langhammer, M., and Vancourt, T., "FPGA floating point datapath compiler," IEEE Symp. 17th IEEE Symp. Field-programamble Custom Computing Machines (FCCM '09), April 5-7, 2009. DOI = http://dx.doi.org/10.1109/FCCM.2009.54

[18] Lemieux, G. Lee, E. Tom, M., and Yu, A. "Directional and single-driver wires in FPGA interconnect," IEEE International Conference on Field-Programmable Technology (FPT '04), pp. 41-48, Dec. 6-8, 2004, DOI: http://dx.doi.org/10.1109/FPT.2004.1393249

[19] Lemieux, G, and Lewis, D. "Using sparse crossbars within LUT clusters," ACM/SIGDA Int. Symp. FPGAs (FPGA '01), pp. 59-68, Feb. 11-13, 2001, DOI= http://doi.acm.org/10.1145/360276.360299

[20] Luu, J., Kuon, I., Jamieson, P., Campbell, T., Ye, A., Fang, W. M., and Rose, J. "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," ACM/SIGDA Int. Symp. FPGAs (FPGA '09), pp. 133-142, Feb. 22-24, 2009, DOI= http://doi.acm.org/10.1145/1508128.1508150

[21] Marquardt, A., Betz, V., and Rose, J. "Timing-driven placement for FPGAs," ACM/SIGDA Int. Symp. FPGAs (FPGA '00), pp. 203-213, Feb. 10-11, 2000, DOI= http://doi.acm.org/10.1145/329166.329208

[22] Marquardt, A., Betz, V., and Rose, J. "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," ACM/SIGDA Int. Symp. FPGAs (FPGA '99), pp. 37-46, Feb. 21-23, 1999, DOI= http://doi.acm.org/10.1145/296399.296426

[23] McMurchie, L., and Ebeling, C. "PathFinder: a negotiation-based performance-driven router for FPGAs," ACM/SIGDA Int. Symp. FPGAs (FPGA '95), pp. 111-117, Feb. 12-14, 1995, DOI= http://doi.acm.org/10.1145/201310.201328

[24] Metzgen, P., and Nancekievill, D. Multiplexer restructuring for FPGA implementation cost reduction. Design Automation Conf. (DAC '05) pp. 421-426, June 13-17, 2005, DOI= http://doi.acm.org/10.1145/1065579.1065692

[25] Verma, A., et al. "Synthesis of floating-point addition clusters on FPGAs using carry-save arithmetic," Int. Conf. Field Programmable Logic and Applications (FPL '10), pp. 19-24, Aug. 31- Sep. 2, 2010.

[26] Xilinx Corporation. Virtex-6 FPGA DSP48E1 Slice User Guide UG369 (v1.2), September 16, 2009. URL= http://www.xilinx.com/support/documentation/user_guides/ug369.pdf