A Low-Overhead Interconnect Architecture for Virtual Reconfigurable Fabrics

Aaron Landy, Dr. Greg Stitt
University of Florida
Department of Electrical & Computer Engineering
Gainesville, FL, USA

landy@hcs.ufl.edu, gstitt@ece.ufl.edu

ABSTRACT

Field-programmable gate arrays (FPGAs) have been widely shown to have significant performance and power advantages compared to microprocessors and graphics-processing units (GPUs), but remain a niche technology due in part to productivity challenges. Although such challenges have numerous causes, previous work has shown two significant contributing factors: 1) prohibitive place-and-route times preventing mainstream design methodologies, and 2) limited application portability preventing design reuse. Virtual reconfigurable architectures, referred to as intermediate fabrics (IFs), were recently introduced as a potential solution to these problems, providing 100x-1000x place-and-route speedup, while also enabling application portability across potentially any physical FPGA. However, one significant limitation of existing intermediate fabrics is area overhead incurred from virtualized interconnect resources. In this paper, we perform design-space exploration of virtual interconnect architectures and introduce an optimized virtual interconnect that reduces area overhead by 48% to 54% compared to previous work, while also improving clock frequencies by 24% with a modest routability overhead of 16%.

Categories and Subject Descriptors

J.6 [**Computer-Aided Enginering**]: Computer-aided Design

General Terms

Performance, Design

Keywords

FPGA, intermediate fabrics, overlay networks, placement and routing, virtualization

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) are reconfigurable devices capable of implementing application-specific circuits that can provide orders of magnitude improvements in performance, power, and energy compared to mainstream microprocessors and graphics-processing units (GPUs) [2][9][12][27]. Although these advantages potentially advance the state-of-the-art for many applications, application designers often only use FPGAs when mainstream technologies cannot meet power and size constraints.

This mainstream resistance to FPGAs has resulted in part from low designer productivity, which previous work has shown to be an order of magnitude worse than other devices [24]. Although the main contributor to low FPGA productivity is an ASIC-prototyping-focused design methodology [24], advances in high-level synthesis from mainstream languages such as CUDA [26] and OpenCL [25] have enabled design flows similar to other devices. However, even with perfect compilers and synthesis tools (hereafter referred to collectively as *compilation*), FPGA productivity still suffers from prohibitive compilation times, often requiring many hours or even days for place-and-route [8], which prevents mainstream design methodologies. Furthermore, the lack of FPGA application portability prevents design reuse that is a common source of improved productivity on other devices.

To address these problems, previous work introduced application-specialized virtual devices, referred to as *intermediate fabrics (IFs)* [8][31]. Through abstraction of fine-grained resources, intermediate fabrics speed up place-and-route by several orders of magnitude while also enabling application portability across any physical FPGA that can implement the virtual fabric. Figure 1 illustrates a simple example of an intermediate fabric specialized

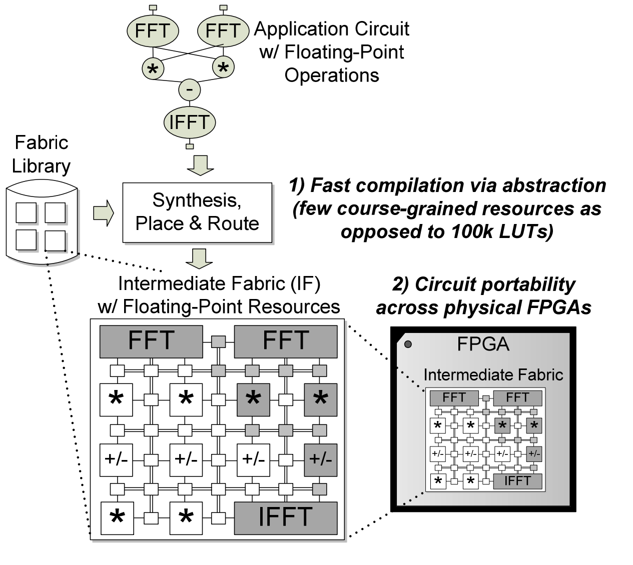
FFT
FFT
Application Circuit w/ Floating-Point Operations
*
*
-
IFFT
Fabric Library
Synthesis, Place & Route
1) Fast compilation via abstraction (few course-grained resources as opposed to 100k LUTs)
Intermediate Fabric (IF) w/ Floating-Point Resources
2) Circuit portability across physical FPGAs
FPGA
Intermediate Fabric
+/-
IFFT

Figure 1: Intermediate fabrics (IFs) are virtual application-specialized fabrics implemented atop FPGAs that hide physical device complexity to achieve fast place-and-route and application portability.

for frequency-domain signal processing by providing coarse-grained, floating-point Fast-Fourier Transforms (FFTs) and arithmetic resources. By compiling a circuit to this intermediate fabric, the compiler avoids decomposing the circuit into tens-of-thousands of lookup tables (LUTs), enabling fast compilation on commercial FPGAs.

Although intermediate fabrics provide significant productivity improvements, previous fabric implementations have limited applicability due to area overhead incurred by the virtual interconnect, which prohibits many usage cases. Although this overhead can be reduced via specialization [8], previous intermediate fabrics can still use 2.5x the area of a circuit directly implemented on a physical FPGA [31].

To address the limitations of previous intermediate fabrics, in this paper we perform design-space exploration of virtual interconnect architectures to determine tradeoffs between area overhead, clock overhead, place-and-route time, bit file size, and reconfiguration time, among others. Such issues have been widely studied for FPGAs over the past two decades [4][28], but conclusions drawn for physical FPGAs are not necessarily applicable to virtual, application-specialized fabrics. Therefore, we revisit fundamental exploration in the context of virtual fabrics to identify key tradeoffs. Based on this exploration, we present an optimized virtual fabric that reduces LUT requirements by 48%-54% and flip-flop requirements by 46%-59%, while improving clock frequencies by an average of 24%. To achieve these improvements, the new interconnect has a modest routability overhead of 16%, which could be addressed by sacrificing a small amount of area savings to include more virtual routing resources.

The paper is organized as follows. Section 2 discusses related work. Section 3 provides an overview of previous intermediate fabrics and their interconnect. Section 4 describes the optimized virtual interconnect. Section 5 presents experimental results.

2. PREVIOUS WORK

Numerous previous studies have focused on overlay networks, which are conceptually similar to intermediate fabrics and implement a virtual network atop a physical FPGA. For example, Kapre et al. [15] compared tradeoffs between packet-switched and time-multiplexed overlay networks implemented on an FPGA. Intermediate fabrics differ from these overlay networks by providing a virtual interconnect capable of implementing register-transfer-level (RTL) circuits at different levels of granularity as opposed to arbitrary communication between abstract processing elements. By this definition, an intermediate fabric is an overlay network, but an overlay network is not necessarily an intermediate fabric.

Previous work has also investigated fine-grained overlay networks for virtual FPGAs [5][18]. Virtual FPGAs are conceptually similar to intermediate fabrics, which also provide virtual reconfigurable fabrics for implementing digital circuits. However, overlays for virtual FPGAs closely imitate fine-grained FPGA architectures [5][18] (e.g. LUTs as resources). Intermediate fabrics can also implement LUT-based architectures, but instead are usually specialized for specific domains and even individual applications using a resource granularity uncommon to FPGAs, which provides fast place-and-route. Previous virtual FPGAs can be viewed as specific, low-level instances of an intermediate fabric. One key difference is that because intermediate fabrics can be specialized, interconnect requirements differ from fine-grained virtual FPGAs, and also vary between specializations.

Numerous previous studies have introduced reconfigurable, coarse-grained physical devices for different application domains [3][7][10][13][14][21][29][30][32]. Although those devices provide good performance for their targeted applications, the disadvantage of such an approach is that specialized physical devices generally have high costs due to limited economy of scale. Intermediate fabrics can provide the same architectures implemented virtually atop common commercial-off-the-shelf FPGAs, which has significant cost advantages and an acceptable overhead for some use cases.

Several studies have also considered virtual coarse-grained architectures for specific domains [30][34]. These approaches are complementary and represent individual instances of intermediate fabrics.

Much previous work has also focused on fast place-and-route using both coarse-grained architectures [6][16][30][35] and specialized algorithms [1][17][23], in some cases combined with a place-and-route-amenable fabric [19][20][33]. Intermediate fabrics are complementary to these approaches and could potentially use these algorithms for place-and-route.

3. INTERMEDIATE FABRICS

This section overviews intermediate fabrics in Section 3.1 and then discusses the virtual interconnect architecture used by previous intermediate fabrics in Section 3.2.

3.1 Overview

As shown in Figure 1, an intermediate fabric is a virtual reconfigurable device, implemented atop a physical FPGA, which implements circuits from HDL or high-level code via synthesis, placement, and routing. Intermediate fabrics, like overlay networks [15] and virtual FPGAs [5][18], provide a fabric capable of implementing numerous circuits. However, unlike those techniques, intermediate fabrics tend to be specialized for the requirements of a specific set of applications, while providing enough routability to support similar applications or different functions in the same domain.

The example in Figure 1 illustrates an intermediate fabric specialized for a frequency-domain signal-processing circuit, and provides corresponding floating-point resources for FFTs and arithmetic computation. When directly compiling this circuit to an FPGA, place-and-route is likely to require hours due to the compiler decomposing the circuit into tens-of-thousands of LUTs. However, when targeting the intermediate fabric, the compiler decomposes the circuit into several coarse-grained resources, which reduces the place-and-route input size by orders of magnitude and provides 100x to 1000x place-and-route speedup [8][31].

A complete discussion of intermediate fabric usage models and their implementations is outside the scope of this paper; we instead summarize two basic models. The library model provides a large, pre-implemented set of intermediate fabrics that a designer or synthesis tool can choose from based on the requirements of the application. For the example in Figure 1, a designer or tool could choose the selected fabric from one of many fabrics that provide different fabric sizes, different combinations of resources, different precisions, etc. An alternative is the synthesis model, during which the synthesis tool creates a specialized fabric based on the application requirements. The advantage to the synthesis model is reduced area overhead. However, the disadvantage is that the application designer must

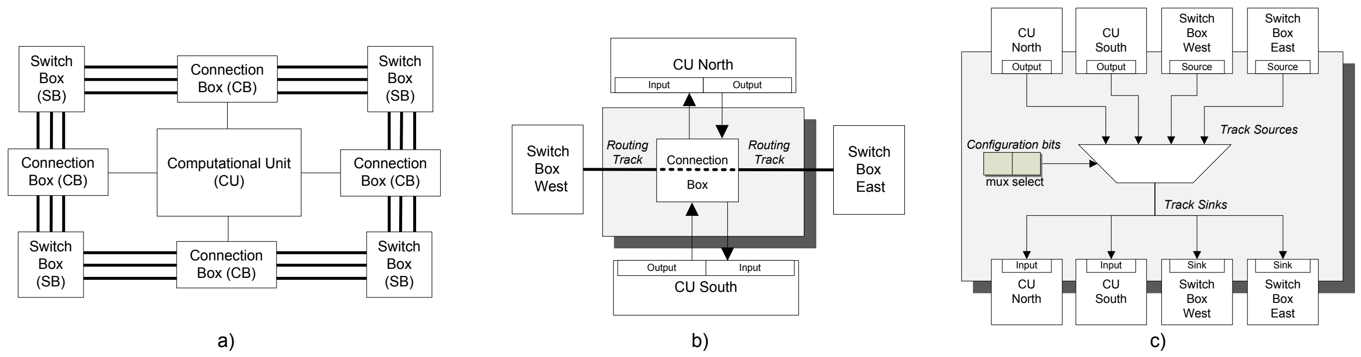

Figure 2: (a) Previous intermediate fabric interconnect architecture, where (b) routing tracks between resources were implemented as (c) multiplexors based on the number of track sources.

wait for place-and-route to implement the intermediate fabric on the physical FPGA. Although such place-and-route may require hours, the compilation time is amortized over the lifetime of the fabric because the physical place-and-route is only needed once.

3.2 Previous Interconnect Architecture

Figure 2(a) illustrates the basic island-style fabric used in previous intermediate fabrics [8][31]. Such a fabric closely imitates the widely studied structure of physical FPGAs consisting of switch boxes, connection boxes, and bidirectional routing tracks, but replaces LUTs with application-specific resources (e.g., floating-point units, FFTs) referred to as *computational units* (CUs). Note that because intermediate fabrics can be specialized, the CUs and virtual routing tracks can potentially be any width. For example, a fabric with floating-point CUs might provide 32-bit routing tracks. Intermediate fabrics also contain specialized regions for control and memory operations. However, in this paper, we focus on the areas of a circuit that contribute the most to long place-and-route, which for many applications are coarse-grained, pipelined datapath operations (e.g., FFTs).

The main limitation of previous intermediate fabrics is area overhead incurred by implementing the virtual fabric atop a physical FPGA (i.e., synthesized VHDL for the virtual fabric). Such overhead results from several sources. The largest source of overhead comes from mux logic in the virtual interconnect. Previous intermediate fabrics use virtual bidirectional routing tracks [8][31], whose register-transfer-level (RTL) implementation is shown in Figure 2(b) and (c). For an *m*-bit track with *n* possible sources, the RTL implementation uses an *m*-bit, *n*:1 mux, in some cases with a register or latch on the mux output. For example, Figure 2(b) shows a common configuration of a bidirectional track with four sources: two switch boxes and two CUs, with the corresponding RTL implementation shown in Figure 2(c) as a 4:1 mux, with a select value stored in a 2-bit virtual configuration register. Considering the large number of tracks found in most fabrics, this mux-based implementation of virtual tracks uses numerous LUT resources in the physical FPGA, and is responsible for over 50% of the total LUT usage in many intermediate fabrics.

Similarly, virtual switch boxes and connection boxes implement various topologies using additional muxes between virtual tracks. The exact percentage of LUT usage for switch/connection boxes varies depending on the box topology and flexibility, but is also a significant contributor to area overhead. When combining all interconnect resources (tracks, switch boxes, and connection boxes), we determined that the virtual interconnect is commonly responsible for over 90% of LUT requirements.

In addition to the mux overhead, intermediate fabrics also require physical flip-flop resources for any storage. Virtual registers are technically not overhead because synthesis tools can directly implement virtual registers on physical flip-flops in the FPGA. However, virtual configuration flip-flops and any pipelined interconnect is overhead because the resulting physical flip-flops would not be used by a circuit directly targeting the FPGA.

4. OPTIMIZED INTERCONNECT

Based on the significant overhead caused by the virtual interconnect described in the previous section, in this paper we

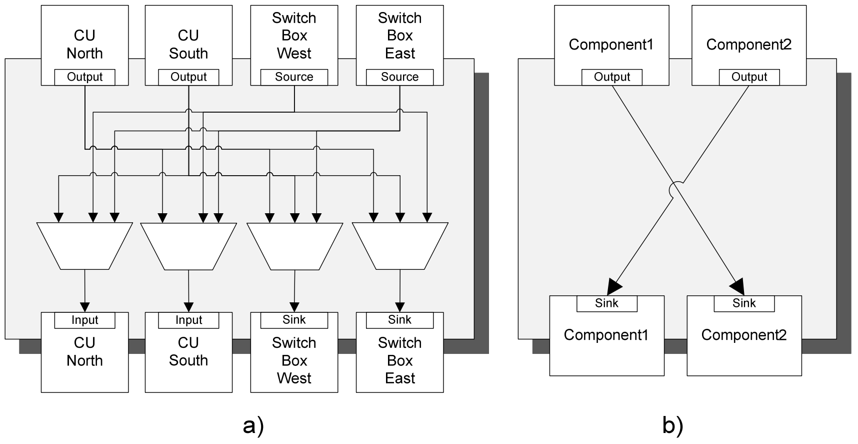

Figure 3: (a) A virtual-track implementation to reduce routing redundancy, which eliminates muxes when (b) tracks have two sources.

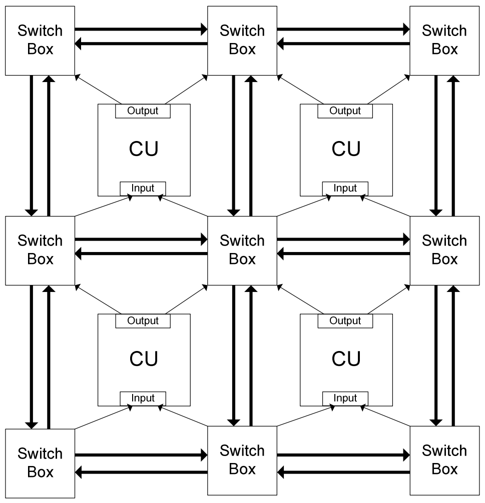

Figure 4: Layout of intermediate fabric using optimized interconnect with CU I/O connected directly to adjacent switch boxes.

focus on virtual interconnect optimizations to reduce muxes, with the goal of retaining high routability.

During an initial attempt at optimizing virtual tracks, we observed that the RTL implementation shown in Figure 2(c) contains some redundancy that could potentially be removed. Specifically, a physical track would never have a common source and sink, which results in an unnecessary input to the mux. For example, a physical FPGA would never route a signal out of a switch box and back into the same switch box using the same track. Therefore, we can eliminate the redundant routes and replace the *n*:1 mux with *n* different, *n*-1:1 muxes, where each mux defines one of the possible track destinations. Figure 3(a) shows an example for the previous track in Figure 2(c), where *n*=4. Despite eliminating routing redundancy, such an approach does *not* save area because in most cases, *n* separate *n*-1:1 muxes require more LUTs than a single *n*:1 mux.

However, we have observed there is a special case where the track implementation in Figure 3(a) can achieve reduced area. For any virtual track with exactly two possible sources, this implementation simplifies into two directional wires as shown in Figure 3(b). In other words, a 2-source virtual track requires two separate 1:1 muxes, but a 1:1 mux is just a wire.

Therefore, by using only 2-source virtual tracks throughout the entire intermediate fabric, we can potentially replace all mux logic and wires in Figure 3(a) with two wires for each track. Such an optimization has significant potential due to virtual tracks contributing to over 50% of area overhead. Furthermore, this optimization saves a significant amount of wires per track, while simultaneously improving routability by enabling routing in two directions. An additional advantage is that by reducing muxes, the fabric requires less configuration registers to store the corresponding select values, which reduces flip-flop overhead while also improving reconfiguration times.

Although using 2-source virtual tracks reduces area, replacing the 3- and 4-source tracks used in previous fabrics is a significant challenge. In a traditional island-style architecture, a track typically has 3-4 possible sources: 2 switch boxes and 1-2 CUs. If we eliminate the switch box connections, the track can only route between adjacent resources, which significantly limits routability. Similarly, if we remove the CU connections, then there is no way for routing to reach CUs.

To address this problem, we considered several significant modifications to traditional fabrics. First, we started with 2-source tracks between adjacent switch boxes, with each switch box as a possible source. However, that interconnect configuration does not provide a mechanism for connecting CUs to the routing tracks. We could have added connection boxes, but that would violate the 2-source restriction. Therefore, we considered adding additional channels to each switch box with direct connections to the CU I/O. The overall fabric layout for this optimized virtual interconnect is shown in Figure 4. As illustrated, in this unconventional fabric, no virtual track has more than 2 sources, which eliminates all muxes previously needed to implement tracks.

One challenge in designing this optimized interconnect is that although we eliminated track muxes, we added additional muxes inside of the switch boxes to support the additional CU channels. Unless the switch boxes add fewer muxes than we removed from the tracks, this optimization does not reduce area.

To ensure that the optimized interconnect reduces LUT usage, we exploit the internal characteristics of the switch box to handle the additional routing requirements with minimal logic. Previous

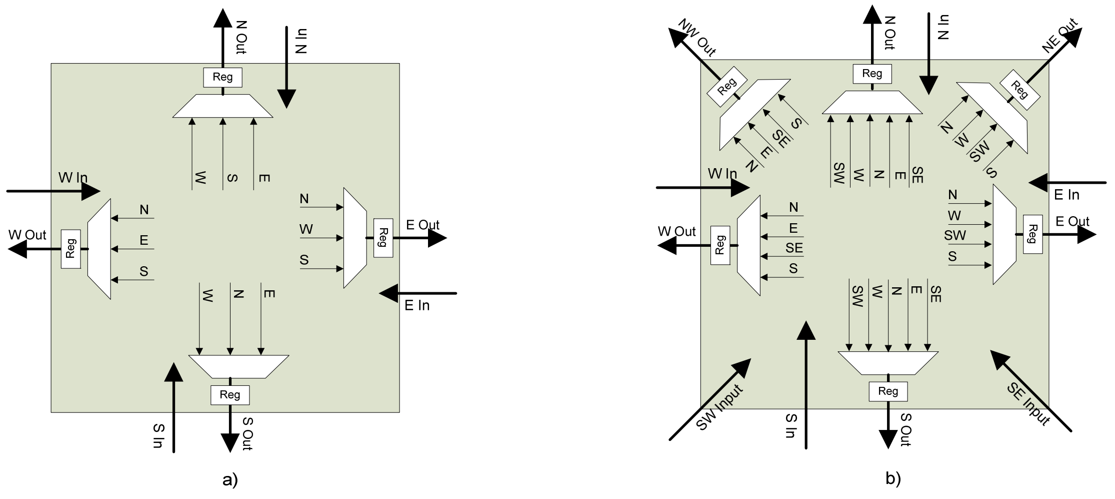

Figure 5: Switch box topology for (a) previous intermediate fabric interconnect and (b) the presented interconnect with diagonal CU channels.

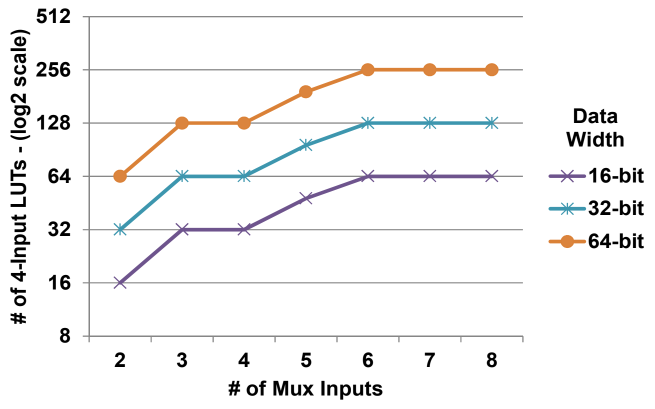

Figure 6: Virtex 4 LX100 LUT usage for different MUX input amounts. The plateaus provide opportunities for switch boxes to add more connections without an area penalty.

intermediate fabric switch boxes use a planar topology, where each output from the switch box uses a 3:1 mux that selects an input from one of the three other channels, as shown in Figure 5(a). For the new interconnect, these multiplexors could potentially require four more inputs to handle routing of the four adjacent CUs, which would significantly outweigh track savings. However, we can exploit the fact that increasing mux inputs does not always increase LUT requirements. As shown in Figure 6, FPGAs have different area "plateaus" where additional mux inputs have the same LUT requirements as lesser inputs (e.g., 3-4 inputs and 6-8 inputs). The optimized interconnect exploits this characteristic by adding CU I/O connections to the muxes until reaching the largest input size of a plateau, which maximizes routability without any increase in area. Interestingly, the presented interconnect can be specialized for different physical FPGAs, which have different mux plateaus due to varying LUT sizes.

Although the optimized interconnect switch boxes are not restricted to a specific topology, we choose a planar-like topology for evaluation and target the mux plateaus for 4-input muxes. Therefore, the switch boxes increase 3-input muxes to 4 inputs wherever possible. The switch boxes also use 5-input muxes, but do not increase the inputs to 6 or more, despite the plateau between 6 and 8 inputs. Increasing the mux inputs to 8 may improve routability with additional overhead, but we defer such analysis to future work. An example topology is shown in Figure 5(b), where the switch box provides a planar topology for the north, east, south, and west channels, which correspond to virtual tracks. In this example, the CU channels (southeast, southwest, northwest, northeast) connect to the other channels in customizable ways. Note that we are not proposing a specific switch box topology for the optimized interconnect. Instead, like any intermediate fabric, we expect the topology to change based on application and routability requirements. For the applications we evaluated, using a highly directional fabric was beneficial due to pipelined, feed-forward datapaths. However, the switch box can easily be customized for other topologies. In the experiments, we use a fabric generation tool that allows specification of the exact switch box topology in a fabric description file.

5. EXPERIMENTS

In this section, we compare intermediate fabrics using the presented virtual interconnect with previous work [8][31]. Section 5.1 describes the experimental setup. Section 5.2 compares area requirements, clock speedups, and routability of both approaches for unspecialized, uniform fabrics. Section 5.3 presents similar experiments for application-specialized fabrics.

5.1 Experimental Setup

This section describes the intermediate fabric tool flow used for the experiments (Section 5.1.1), along with the routability measurements (Section 5.1.2), and the tools used for evaluating the different interconnects (Section 5.1.3).

5.1.1 Tool flow

To implement applications on the intermediate fabrics, we manually synthesize circuits by creating technology-mapped netlists. We plan to convert open-source synthesis tools to target intermediate fabrics, including OpenCL high-level synthesis, but such a project is outside the scope of this paper. For place-and-route, we use the algorithm previously described in [8] to ensure that the comparison between the new and previous interconnect is not unfairly skewed by improved placement. In fact, the place-and-route results for the new interconnect are likely pessimistic because we did not modify the placer cost function for the new interconnect. The place-and-route algorithm is a variation of VPR [4], and uses simulated annealing for placement with a cost function that minimizes bounding box size. Routing uses the well-known PathFinder [22] negotiated-congestion algorithm. Both the new and previous interconnect have varying amounts of pipelining in switch boxes or on tracks. Instead of using pipelined routing algorithms (e.g., [11]), both approaches use realignment registers in front of each CU to balance the routing delays of all inputs. Because this pipelining strategy only works for pipelined datapaths that can be retimed without affecting correctness, we limit the evaluation to fabrics with coarse-grained resources commonly needed by datapaths in signal processing.

To configure the intermediate fabric for different applications, the place-and-route tool outputs a configuration bit file that we store in a block RAM on the targeted FPGA. Each intermediate fabric includes a programmer which loads the bitfile from the block RAM by shifting bits into virtual configuration registers that control the CUs and virtual switch boxes.

5.1.2 Routability Metric

To fairly compare tradeoffs between interconnects, it is necessary to measure routability. To perform these measurements for a given intermediate fabric, we place-and-route a large number of randomly generated netlists of varying sizes, and determine the 'routability score' of the interconnect based on the percentage of netlists that route successfully. Due to the fast place-and-route time for intermediate fabrics we were able to test 1,000 netlists for each fabric to obtain a high-precision metric.

The random netlist generator creates directed acyclic graph structures representative of pipelined datapaths. Based on the CU composition of each individual fabric tested, the generator creates a random number of datapath stages, each consisting of a random number of technology-mapped cells, and creates random connections between each stage. Each stage contains at minimum enough cells, and enough connections are made between stages, such that each cell has at least one path to the next stage. This method results in netlists containing one or more disjoint pipelines of one or more stages each.

Table 1: A comparison between the presented virtual interconnect (*New*) and previous uniform virtual interconnect (*Prev*). The presented interconnect had significant area savings and a clock speedup of 24%, with a modest 16% decrease in routability.

		LUT Usage			Flip-Flop Usage			Routability			Clock		
FPGA	Fabric Size (# of CUs)	Prev	New	Savings	Prev	New	Savings	Prev	New	Overhead	Prev	New	Speedup
Xilinx V4LX200	3x3	2%	1%	**71%**	1%	0.2%	**72%**	100%	78%	**22%**	173 MHz	175 MHz	**1%**
	4x4	5%	2%	**64%**	1%	0.4%	**65%**	100%	95%	**5%**	163 MHz	172 MHz	**6%**
	5x5	8%	3%	**60%**	2%	1%	**62%**	100%	87%	**13%**	152 MHz	172 MHz	**13%**
	6x6	12%	5%	**55%**	3%	1%	**59%**	100%	85%	**15%**	144 MHz	171 MHz	**19%**
	7x7	17%	8%	**53%**	5%	2%	**57%**	100%	84%	**16%**	123 MHz	170 MHz	**38%**
	8x8	23%	11%	**52%**	6%	3%	**56%**	100%	85%	**16%**	125 MHz	170 MHz	**36%**
	9x9	30%	15%	**51%**	8%	4%	**55%**	99.7%	84%	**16%**	115 MHz	168 MHz	**46%**
	12x8	36%	20%	**46%**	10%	5%	**55%**	99.2%	79%	**20%**	113 MHz	160 MHz	**42%**
Xilinx V5LX330	13x13	37%	20%	**46%**	18%	9%	**53%**	98%	80%	**18%**	125 MHz	162 MHz	**30%**
	14x14	44%	24%	**46%**	21%	10%	**52%**	94%	83%	**12%**	131 MHz	148 MHz	**13%**
Altera S4E530	15x15	*n/a**	14%	***n/a****	*n/a**	18%	***n/a****	90%	71%	**21%**	*n/a**	175 MHz	***n/a****
	16x16	*n/a**	16%	***n/a****	*n/a**	21%	***n/a****	90%	70%	**22%**	*n/a**	177 MHz	***n/a****
	Average	21%	11%	**54%**	8%	3%	**59%**	98%	82%	**16%**	136 MHz	167 MHz	**24%**

**n/a corresponds to examples that failed to synthesize for the corresponding device, which are excluded from the averages*

5.1.3 Interconnect Evaluation

To evaluate different interconnects, we developed a tool capable of generating VHDL for intermediate fabrics using the new interconnect. The tool takes as inputs a fabric-description file that defines the parameters of the fabric, such as size, aspect ratio, bit-width and the makeup of the fabric, including CU composition, and row and column channel descriptions. Channel descriptions include number of tracks, direction of each track, and switchbox topology.

To obtain physical FPGA utilization and timing results, we synthesized the intermediate fabric VHDL using Xilinx ISE 10.1, Synopsys Synplify Pro 2012, and Altera Quartus II 10.1, depending on the targeted FPGA. To evaluate the effects of FPGA variation on each virtual interconnect, we implemented intermediate fabrics on Xilinx Virtex 4 LX100 and LX200, Xilinx Virtex 5 LX330, and Altera Stratix IV E530 FPGAs. The intermediate fabric HDL synthesized for each test case uses the fixed-logic multipliers available on each physical device for all CUs (Xilinx DSP48s and Altera 18x18 Multipliers); therefore all device utilization represents the LUT and flip-flop overhead of implementing the target application via an intermediate fabric rather than a direct HDL implementation.

5.2 Interconnect Comparison for Uniform Intermediate Fabrics

In this section we compare area, routability, and maximum clock speed of intermediate fabrics using the presented interconnect to intermediate fabrics using interconnect previously presented in [8] and [31]. We evaluate each interconnect using different fabric sizes, implemented on several different physical FPGAs. Although intermediate fabrics can be specialized to an application, in this section we evaluate fabrics independently of targeted applications by using a uniform fabric consisting of 16-bit DSP CUs with various dimensions (e.g., 5x5 = 5 rows and 5 columns of I/O and CUs).

Table 1 compares LUT and flip-flop utilization (as a % of total device resources), routability of 1000 randomly generated netlists, and maximum clock speed for identical intermediate fabrics using the new and previous interconnects. We implemented fabric sizes between 3x3 and 12x8 on a Virtex 4 LX200, where an NxM fabric is composed of one row of *M* inputs, N-2 rows of *M* CUs, and one row of *M* outputs. We evaluated larger fabric sizes of 13x13 and 14x14 on a Virtex 5 LX330, and sizes 15x15 and 16x16 on a large Stratix IV E530. For fabrics using the previous interconnect, we used 3 16-bit tracks per channel with specialized connection boxes from [8], as previous work indicated this configuration to be an effective tradeoff between routability and overhead. For fabrics using the new interconnect, we used 2 16-bit tracks per row and 4 tracks per column with the switch box topology described in Section 4 optimized for 4-input muxes.

These results show the LUT and flip-flop utilizations of the new interconnect are significantly less than the previous interconnect, with an average LUT savings of 54% and flip-flop savings of 59% for the fabrics evaluated. Note that we were unable to synthesize the old interconnect on the Stratix IV device. We tried three different version of Quartus, but the old interconnect would cause a crash during the retiming stage of synthesis. For this reason, we exclude the Stratix IV results from the averages.

Additionally, the new interconnect showed significant maximum clock frequency speedup for larger fabrics. When implemented on the Virtex 4, new interconnect clock speeds decreased only 6.3% between fabrics of size 3x3 to 12x8, whereas the previous interconnect suffered from a 34.7% decrease in clock speed over the same range. Overall, the new interconnect averaged 167 MHz compared to 136 MHz.

The new interconnect did incur a routability penalty, with a average decrease of 16% compared to the previous interconnect. While this overhead is a potential limitation of the new interconnect, especially when applied to a general-purpose fabric, we believe this overhead to be an acceptable tradeoff when compared to the significant area savings provided by the new interconnect. Routability overhead can also be easily compensated for when designing the CU composition of a fabric. Because the placer algorithm used in these experiments is unchanged from that used for the old fabric, it is likely that an appropriately customized placer cost function would significantly improve the routability of the new interconnect. Similarly, fabrics using the new interconnect could account for decreased routability by including many more routing resources while still saving area. Routability decreased monotonically with increased fabric size due to the increased difficulty of routing larger netlists. The one exception was the 3x3 fabric with the new interconnect, which had lower routability than the larger fabrics. We identified the source of this problem as limited connections between I/O and

Table 2: A comparison between intermediate fabrics (IFs) with the presented virtual interconnect (*IF New*) and previous application-specialized interconnect (*IF Prev*). The results show 1350x place-and-route speedup compared to an FPGA, and 2.4x speedup compared to previous IFs. Average area savings were 48%, with little effect on average clock speed or routability. Two examples had significantly higher routability on the new interconnect.

	Place-and-Route Time					Area and Routability			Clock Speed		
	IF Prev	IF New	FPGA	Speedup Prev	Speedup New	LUT Savings	Flip-Flop Savings	Routability Overhead	IF Prev	IF New	Clock Overhead
Matrix multiply FXD	0.6s	0.6s	1min 08s	**112x**	**112x**	**56%**	**60%**	1%	170 MHz	186 MHz	**-9%**
Matrix multiply FLT	0.6s	0.6s	6min 06s	**602x**	**602x**	**59%**	**59%**	1%	184 MHz	222 MHz	**-21%**
FIR FXD	0.6s	0.6s	0min 33s	**54x**	**58x**	**45%**	**41%**	5%	174 MHz	158 MHz	**9%**
FIR FLT	0.6s	0.6s	4min 36s	**454x**	**484x**	**35%**	**35%**	5%	203 MHz	215 MHz	**-6%**
N-body FXD	0.5s	0.2s	0min 57s	**126x**	**300x**	**40%**	**32%**	1%	185 MHz	165 MHz	**11%**
N-body FLT	0.5s	0.2s	3min 42s	**491x**	**1168x**	**37%**	**26%**	1%	218 MHz	200 MHz	**8%**
Accum FXD	0.1s	0.02s	0min 26s	**280x**	**1733x**	**52%**	**53%**	0%	186 MHz	187 MHz	**-1%**
Accum FLT	0.1s	0.02s	0min 30s	**323x**	**2000x**	**52%**	**50%**	0%	225 MHz	241MHz	**-7%**
Normalize FXD	0.2s	0.3s	1min 10s	**299x**	**241x**	**66%**	**71%**	-63%	178 MHz	162 MHz	**9%**
Normalize FLT	0.2s	0.3s	6min 44s	**1726x**	**1393x**	**43%**	**54%**	-63%	197 MHz	222 MHz	**-13%**
Bilinear FXD	0.3s	0.3s	1min 08s	**230x**	**213x**	**51%**	**47%**	0%	184 MHz	165 MHz	**10%**
Bilinear FLT	0.3s	0.3s	8min 48s	**1784x**	**1650x**	**41%**	**42%**	0%	206 MHz	200 MHz	**3%**
Floyd-Steinberg FXD	0.1s	0.1s	1min 27s	**621x**	**926x**	**53%**	**50%**	2%	182 MHz	169 MHz	**7%**
Floyd-Steinberg FLT	0.1s	0.1s	5min 37s	**2407x**	**3585x**	**48%**	**44%**	2%	196 MHz	179 MHz	**9%**
Thresholding	1.4s	1.3s	0min 33s	**24x**	**26x**	**44%**	**36%**	5%	167 MHz	181MHz	**-8%**
Sobel	0.3s	0.4s	2min 28s	**500x**	**344x**	**44%**	**31%**	2%	181MHz	162 MHz	**10%**
Gaussian Blur	3.3s	2.2s	3min 19s	**60x**	**90x**	**39%**	**41%**	-42%	170 MHz	181MHz	**-6%**
Max Filter	0.2s	0.03s	1min 16s	**444x**	**2533x**	**48%**	**41%**	0%	186 MHz	176 MHz	**5%**
Mean Filter 3x3	0.2s	0.01s	2min 30s	**962x**	**10714x**	**52%**	**52%**	10%	185 MHz	187 MHz	**-1%**
Mean Filter 5x5	1.9s	1.9s	3min 25s	**110x**	**108x**	**64%**	**65%**	-1%	169 MHz	161MHz	**5%**
Mean Filter 7x7	8.9s	4.7s	5min 03s	**34x**	**64x**	**39%**	**40%**	-38%	157 MHz	183 MHz	**-17%**
Average	1.0s	0.7s	2min 56s	**554x**	**1350x**	48%	**46%**	-8%	186 MHz	186 MHz	**0%**

CUs for very small fabrics using the new interconnect. Because we expect 3x3 to be an unusually small size for actual usage, this overhead is not a significant limitation.

These results also show decreased LUT overhead savings of only 46% in fabrics implemented on the Virtex 5 device. This smaller improvement is likely due to different CLB configuration used by that device, with slightly altered mux-area plateau characteristics, whereas the optimizations used by the evaluated interconnect were optimized for 4-input muxes. Despite being optimized for a different LUT configuration, the new interconnect still had significant savings.

Flip-flop usage on the Altera device was significantly higher than both Xilinx devices, which resulted from the Xilinx FPGAs implementing the realignment registers as SRL16 primitives, in contrast to the Altera FPGA which used flip-flops. As future work, we will investigate optimizations for Altera FPGAs.

One additional advantage of reducing muxes throughout the interconnect is the corresponding elimination of configuration registers to store the select values. The fewer registers reduce flip-flops, which was shown in Table 1, but also reduces configuration bitfile size, which correspondingly reduces configuration times and block RAM overhead of the fabric. For the examples in this section, the new interconnect improved configuration times by an average of 55% compared to the previous interconnect.

5.3 Interconnect Comparison for Specialized Intermediate Fabrics

One advantage of intermediate fabrics is that a designer or tool can specialize the architecture and interconnect for a given domain or even an individual application. In this section, we compare intermediate fabrics using application-specialized interconnect presented in [8] with the new interconnect. To enable a fair comparison, we evaluate the same application circuits from [8] using the same specialized fabrics as previous experiments. Specialization used in the previous experiments included varying fabric sizes and non-uniform interconnects. For the new interconnect, we limit specialization to fabric sizes, making the results pessimistic. For all specialized fabrics, we used the smallest fabric and interconnect that could successfully route the target application netlist. For these experiments, the physical FPGA is a Virtex 4 LX100, which we chose to match the previous experiments.

To perform the comparison, we used the twelve applications from [8], seven of which were implemented using both 16-bit fixed point arithmetic and 32-bit floating point arithmetic, indicated with a FXD or FLT suffix respectively. All track widths matched the CU widths. All circuits without a suffix used 16-bit fixed-point CUs. We briefly summarize the previous applications as follows. *Matrix multiply* performs the kernel of a matrix multiplication, calculating the inner product of two 8-element vectors using 7 adders and 8 multipliers. *FIR* implements a 12-tap finite impulse response filter in transpose form with symmetric coefficients using 11 adders and 12 multipliers. *N-body*, representing the kernel of an N-body simulation, calculates the gravitational force exerted on a particle due to other particles in two-dimensional space using 13 adders, multipliers, and a divider. *Accum* monitors a stream, counting the number of times the value is less than a threshold. It is the smallest netlist, consisting of 4 comparators and 3 adders. *Normalize* normalizes an input stream using 8 multipliers and 8 adders. *Bilinear* performs bilinear interpolation on an image, requiring 8 multipliers and 3 adders. *Floyd-Steinberg* performs image dithering using 6 adders and 4 multipliers. *Thresholding* performs automatic image thresholding using 8 comparators and 14 adders. *Sobel* uses a 3x3 convolution to perform Sobel edge detection with 2 multipliers and 11 adders. *Gaussian blur* uses a 5x5 convolution to perform noise reduction using 25 multipliers and 24 adders. *Max filter* performs a 3x3 sliding-window image filter with 8 comparators. *Mean filter* similarly calculates the average of a sliding window, which we vary from 3x3 to 7x7, requiring a maximum of 48 adders and 1 multiplier.

Table 2 compares the interconnects for each case study. The first major column, *Place-and-Route Time*, compares place-and-route execution times for an intermediate fabric with the previous interconnect (*IF Prev*), an intermediate fabric with the new interconnect *(IF New)*, and when synthesizing VHDL for each example directly to the FPGA. The table also shows the resulting place-and-route speedup for the new and previous interconnects. The results show comparable place-and-route times for both the old and new interconnect. However, because the previous interconnect already achieves a place-and-route speedup of 554x compared to an FPGA, the further improvement by the new interconnect provided a 1350x place-and-route speedup. The place-and-route speedup was larger for the floating-point examples due to longer place-and-route times for the physical FPGA. Furthermore, these place-and-route speedups are highly pessimistic because the specialized examples from [8] do not include common board logic such as PCIe and memory controllers. Other studies have shown that including these controllers with tight timing constraints can add up to 20 minutes to FPGA place-and-route time, but have no effect on intermediate fabric place-and-route time [31].

The second major column in Table 2 reports area savings of the new interconnect in terms of FPGA LUTs and flip-flops, along with the routability overhead incurred to achieve these savings. On average, the new interconnect significantly reduced LUT usage by 48% and flip-flop usage by 46%, despite the significant specialization by the previous fabrics. On average, routability slightly improved by 8% with the new interconnect. However, this average is skewed by three outliers, *normalize*, *Gaussian*, and *mean7x7*, which had very low routability due to significant specialization in the previous fabrics. Excluding these outliers, the new interconnect had a 2% routability overhead. The smaller routability overhead compared to the previous section is due to the specialized versions of the previous interconnect, which used just enough routing resources to route the targeted application, and therefore lowered general routability.

The final column of table 2 compares the maximum clock speed of the specialized fabrics using both the new and old interconnect. For specialized fabrics, these experiments show a negligible average impact on clock speed, with both interconnects showing an average clock frequency of 186 MHz. However, there was significant variation as high as 21% between specialized fabrics. It should be noted that these results are contrary to the results for larger fabrics presented in the previous section, which showed a clear trend of faster clock speeds for larger fabrics using the new interconnect. The reason for the smaller clock improvement compared to the previous section is due to the higher specialization of the previous interconnect, as opposed to using a uniform interconnect.

6. LIMITATIONS AND FUTURE WORK

Even with a 50% reduction in LUT utilization, intermediate fabrics will still have prohibitive overhead for use cases where an FPGA is close to being fully utilized. Fortunately, the trends towards multi-million-LUT FPGAs will lessen this problem over time. In addition, we plan to investigate virtual interconnect that directly targets the physical FPGA interconnect without using muxes. Such an approach could map virtual switch boxes directly onto physical switch boxes, potentially eliminating much of the remaining overhead. However, such an approach requires knowledge of proprietary routing architectures, and is therefore deferred to future work.

7. CONCLUSIONS

Previous work introduced intermediate fabrics to address FPGA problems related to lengthy place-and-route times and a lack of application portability. Although previous intermediate fabric approaches achieve both application portability and significant place-and-route speedup, the area overhead of those approaches prohibits important use cases. To address this problem, we identified the virtual interconnect as the main source of the overhead, and performed design-space exploration to identify unconventional alternatives that could achieve effective Pareto-optimal tradeoffs between overhead and routability. Based on this analysis, we introduced an optimized virtual interconnect architecture that reduces area requirements by approximately 50% and improves clock frequencies by 24%, with a modest 16% reduction in routability.

8. ACKNOWLEDGMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422 and IIP-1161022. The authors gratefully acknowledge vendor equipment and/or tools provided by Altera and Xilinx.

9. REFERENCES

[1] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf, “Wires on demand: Run-time communication synthesis for reconfigurable computing,” in *FPL ’07: International Conference on Field Programmable Logic and Applications*, Aug. 2007, pp. 513–516.

[2] Z. Baker, M. Gokhale, and J. Tripp. “Matched filter computation on fpga, cell and gpu,” In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, 2007, pp. 207–218.

[3] J. Becker, T. Pionteck, C. Habermann, and M. Glesner, “Design and implementation of a coarse-grained dynamically reconfigurable hardware architecture,” in *VLSI ’01: Proceedings of IEEE Computer Society Workshop on VLSI*, May 2001, pp. 41–46.

[4] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *FPL ’97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1997, pp. 213–222.

[5] A. Brant and G. Lemieux. "XUMA: An open FPGA overlay architecture", In *FCCM '12: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2012.

[6] T. J. Callahan, P. Chong, A. DeHon, and J. Wawrzynek, "Fast module mapping and placement for datapaths in FPGAs," in *FPGA '98: Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 1998, pp. 123–132.

[7] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *FCCM'01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* 2001, pp. 111–119.

[8] J. Coole and G. Stitt. "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing." In *CODES/ISSS '10: Proceedings of the IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, 2010, pp. 13–22.

[9] A. DeHon, "The density advantage of configurable computing," *Computer*, vol. 33, no. 4, pp. 41–49, 2000.

[10] C. Ebeling, D. C. Cronquist, and P. Franklin, "Rapid - reconfigurable pipelined datapath," in *FPL '96: Proceedings of the 6th International Workshop on Field-Programmable Logic,Smart Applications, New Paradigms and Compilers*. London, UK: Springer-Verlag, 1996, pp. 126–135.

[11] K. Eguro and S. Hauck, "Armada: timing-driven pipeline-aware routing for FPGAs," in *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, 2006, pp. 169–178.

[12] J. Fowers, G. Brown, P. Cooke, and G. Stitt. "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications", In *FPGA '12: Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA'12, 2012, pp. 47–56.

[13] D. Grant, C. Wang, and G. G. Lemieux. "A CAD framework for Malibu: an FPGA with time-multiplexed coarse-grained elements", In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, 2011, pp. 123–132.

[14] M. Hammerquist and R. Lysecky, "Design space exploration for application specific FPGAs in system-on-a-chip designs," in *SOC '08: Proceedings of the IEEE International SOC Conference*, Sept. 2008, pp. 279–282.

[15] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, "Packet-switched vs. time-multiplexed FPGA overlay networks," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[16] A. Koch, "Structured design implementation: a strategy for implementing regular datapaths on FPGAs," in *FPGA '96: Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*, 1996, pp. 151–157.

[17] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. "HMFlow: Accelerating FPGA compilation with hard macros for rapid prototyping", In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, 2011, pp. 117 –124.

[18] R. Lysecky, K. Miller, F. Vahid, and K. Vissers. "Firm-core virtual fpga for just-in-time fpga compilation", In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, 2005, pp. 271–271.

[19] R. Lysecky, F. Vahid, and S. X.-D. Tan, "Dynamic fpga routing for just-in-time FPGA compilation," in *DAC '04: Proceedings of the 41st Annual Conference on Design Automation*, 2004, pp. 954–959.

[20] R. Lysecky, F. Vahid, and S. X. D. Tan, "A study of the scalability of on-chip routing for just-in-time FPGA compilation," in *FCCM '05: Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005, pp. 57–62.

[21] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *FPGA '99: Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, 1999, pp. 135–143.

[22] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *FPGA '95: Proceedings of the 1995 ACM Third International Symposium on Field Programmable Gate Arrays*, 1995, pp. 111–117.

[23] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *FPGA '01: Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, 2001, pp. 29–36.

[24] B. E. Nelson, M. J. Wirthlin, B. L. Hutchings, P. M. Athanas, and S. Bohner, "Design productivity for configurable computing," in *ERSA '08: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2008, pp. 57–66.

[25] M. Owaida, N. Bellas, K. Daloukas, and C. Antonopoulos. "Synthesis of platform architectures from opencl programs", In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, 2011, pp. 186–193.

[26] A. Papakonstantinou, K. Gururaj, J. Stratton, D. Chen, J. Cong, and W.-M. Hwu. "FCUDA: Enabling efficient compilation of cuda kernels onto fpgas", In *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, 2009, pp. 35–42.

[27] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. Van Hulle. "A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features", *Computers, IEEE Transactions on*, PP(99):1, 2011.

[28] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. "The VTR project: architecture and cad for fpgas from verilog to routing", In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA '12, 2012, pp. 77–86.

[29] L. Sekanina, *Evolvable Systems: From Biology to Hardware*. Springer Berlin / Heidelberg, 2003, ch. Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware, pp. 116–137.

[30] S. Shukla, N. W. Bergmann, and J. Becker, "Quku: A two-level reconfigurable architecture," in *ISVLSI '06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006, p. 109.

[31] G. Stitt and J. Coole. "Intermediate fabrics: Virtual architectures for near-instant FPGA compilation", *Embedded Systems Letters, IEEE*, 3(3):81 –84, sept. 2011.

[32] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon, "HSRA: high-speed, hierarchical synchronous reconfigurable array," in *FPGA '99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 125–134.

[33] F. Vahid, G. Stitt, and R. Lysecky, "Warp processing: Dynamic translation of binaries to FPGA circuits," *Computer*, vol. 41, no. 7, pp. 40–46, July 2008.

[34] J. Wang, Q. Chen, and C. Lee, "Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware," *Computers & Digital Techniques, IET*, vol. 2, no. 5, pp. 386–400, September 2008.

[35] P. Yiannacouras, J. G. Steffan, and J. Rose. Vespa: portable, scalable, and flexible fpga-based vector processors. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, CASES '08, 2008, pp. 61–70.