Intermediate Fabrics: Virtual Architectures for Near-Instant FPGA Compilation

Greg Stitt and James Coole, *Member, IEEE*

***Abstract*—Field-programmable gate arrays (FPGAs) suffer from lower application design productivity than other devices, which is largely due to compilation taking hours or even days. Making FPGA compilation comparable to software compilation is critical for continued FPGA usage due to competitive technologies, such as graphics-processing units, that use languages with runtime compilation models. In this letter, we evaluate virtual reconfigurable architectures, referred to as *intermediate fabrics*, which enable near-instant placement and routing of applications for commercial FPGAs.**

***Index Terms*—Field-programmable gate arrays (FPGAs), placement and routing, productivity, virtual architectures.**

I. Introduction

DESPITE performance, power, and energy advantages compared to other devices [4], field-programmable gate array (FPGA) usage has been limited by poor application design productivity [12]. Such productivity has resulted largely from a requirement for low-level device expertise and prohibitive compilation times caused by placement and routing (PAR), which commonly requires many hours or even days. Although numerous high-level synthesis studies have raised abstraction levels, PAR times have largely been ignored and are now a major productivity bottleneck that prevents designers from using mainstream design and debugging methodologies based on rapid compilation. Furthermore, PAR times prevent high-level synthesis from increasingly used languages (e.g., OpenCL) that require runtime compilation.

Intermediate fabrics (IFs) [3] were recently introduced as a potential solution for fast FPGA compilation, providing several orders of magnitude faster PAR compared to vendor tools. IFs are virtual reconfigurable architectures implemented atop commercial-off-the-shelf (COTS) FPGAs, which designers either select from a library or custom generate to provide an application-specialized virtual architecture that contains appropriate resources (e.g., floating-point cores, FFTs, filters, etc.), as opposed to mapping the application onto possibly hundreds of thousands of fine-grained look-up-tables (LUTs) in the physical device. Such specialization reduces PAR input size by orders of magnitude, thus enabling fast compilation. Although numerous coarse-grained reconfigurable devices have similar advantages, IFs enable designers to use COTS devices, which have significant cost advantages. Other advantages include application portability across devices and rapid reconfiguration that is orders of magnitude faster than partial reconfiguration on FPGAs.

The main limitation of IFs is area and performance overhead. Although previous work [3] showed reasonable overhead, that study focused on specializing IFs for a single netlist. In this letter, we evaluate a more realistic usage scenario of a single, larger IF specialized for common image-processing kernels. In addition, we evaluate reconfiguration of IFs on an FPGA that does not support partial reconfiguration.

Although not appropriate for all usage cases, IFs achieved an average PAR speedup of $700\times$ over vendor tools. Due to a lack of proprietary routing details for commercial devices, we evaluated area overhead using highly pessimistic mux-based routing resources that required 34% to 44% of an Altera Stratix III E260. Performance overhead was only 7%. Fortunately, FPGA vendors could directly map an IF onto physical routing resources to eliminate much of this overhead, which is decreasing in significance as FPGA sizes approach one million LUTs.

II. Previous Work

Previous work has focused on fast PAR algorithms, but those studies either provided insufficient speedup for runtime compilation [11], required custom devices [14], or were intended for specific usage scenarios (e.g., runtime reconfiguration [1]). Coarse-grained reconfigurable architectures (e.g., [6]) also enable fast PAR, but require specialized devices. IFs complement these approaches by enabling rapid PAR on COTS FPGAs. A recent approach [9] introduced preplaced and routed hard macros that provided $10\times$–$50\times$ PAR speedup, with a $2\times$–$4\times$ performance overhead. IFs provide faster PAR and less performance overhead [3], but are generally less flexible due to the requirement for an appropriate fabric. Hard macros could potentially be combined with IFs for rapidly generating specialized fabrics.

IFs are conceptually similar to other overlay networks [8] and virtual architectures [13]. Such approaches implement custom architectures atop FPGAs, but do so for purposes other than rapid compilation, portability, or partial reconfiguration.

Manuscript received May 31, 2011; accepted August 17, 2011. Date of publication September 12, 2011; date of current version October 28, 2011. This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant EEC-0642422. This manuscript was recommended for publication by P. Lysaght.

G. Stitt is with the NSF Center for High-Performance Reconfigurable Computing and the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611-6200 USA (e-mail: gstitt@ece.ufl.edu).

J. Coole is with Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611-6200 USA (e-mail: jcoole@ufl.edu).

Digital Object Identifier 10.1109/LES.2011.2167713

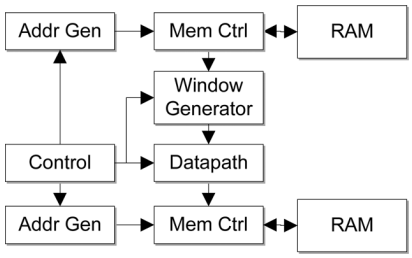

Fig. 1. Common circuit structure for pipelined image-processing kernels.

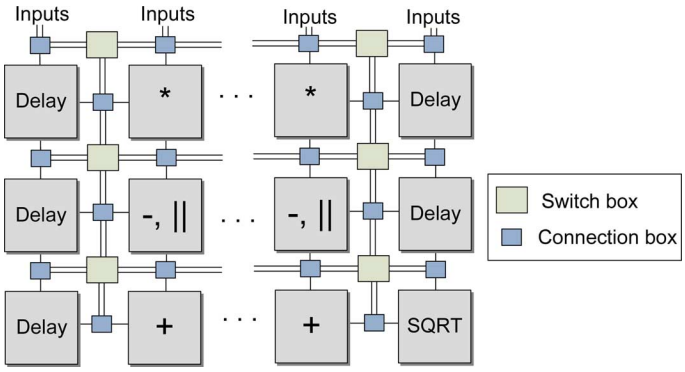

Fig. 2. Overview of the island-style datapath fabric used for the evaluated image-processing kernels. Each row has 66 columns.

III. Intermediate Fabrics

A. Architecture

Although IFs can implement architectures specialized for any domain, in this letter, we use an architecture based on common characteristics of image-processing circuits, as shown in Fig. 1. This architecture consists of a controller and a pipelined datapath that accepts a window—a subset of the image—as input every cycle, and generates an output every cycle after some initial latency. Because pipelines require high memory bandwidth, circuits commonly use a window generator [5] to exploit data reuse between consecutive windows. The architecture also uses address generators and memory controllers to stream data to and from memories.

Many image-processing circuits use this architecture with different pipelined datapaths. Therefore, for the IF in this letter, we implement a reconfigurable version of the architecture in Fig. 1, focusing mainly on creating a reconfigurable fabric for pipelined datapaths, as shown in Fig. 2. Although the fabric could include any resource, we chose resources common to image-processing kernels: multipliers, subtractors with an optional absolute-value output, adders, a square root, and delay components that act as configurable shift registers. For the total size, we chose 128 inputs, five delays capable of delaying up to 9000 cycles, 64 multipliers, 64 subtractors, 63 adders, one square root, and one single output. We chose this size to be large enough to implement numerous kernels while using a midsized Stratix III E260 FPGA. All operations use 16-bit fixed-point, signed arithmetic. To implement the virtual fabric on a physical FPGA, we specified the computational resources in VHDL. We implemented the delays using block RAMs and the square root using an Altera-provided IP core.

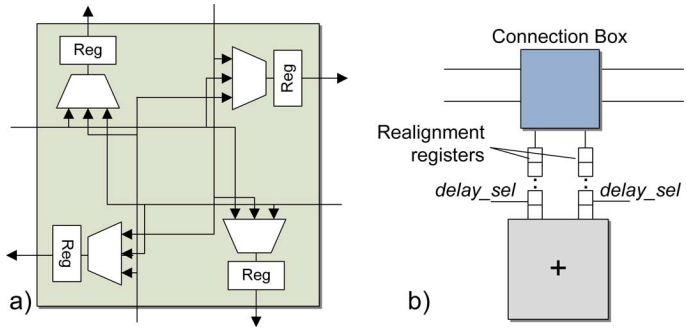

Fig. 3. (a) RTL implementation of the virtual switchbox for a single track per channel. (b) Realignment registers for adjusting routing latencies.

The IF routing resources are similar to island-style FPGAs, with connections boxes to connect resources to virtual routing tracks, and planar switch boxes to connect tracks. The IF uses two 16-bit bidirectional tracks in each row and column.

Ideally, we would implement the IF routing resources directly on an FPGA's physical routing resources. However, because such resources are proprietary, we instead perform a highly pessimistic evaluation using mux-based virtual routing resources. We implement switch boxes as shown in Fig. 3(a), where each bidirectional, virtual track becomes two unidirectional wires with a mux defining which tracks define each output. Each mux has a configuration register (not shown) that controls the mux select, as determined by PAR. In addition, each output has a register to pipeline the interconnect, which was necessary to achieve clock frequencies comparable to FPGAs with the pessimistic evaluation. Considering the minimal performance overhead, it is likely that a direct mapping onto physical routing resources would not need pipelined interconnect. Note that pipelining is possible because we specialize the IF for pipelined datapaths. We implement each track as a mux that defines a sink by selecting from each source. Due to the numerous routing resources, these muxes are the main source of the evaluated IF overhead. To load a bitfile, the IF uses a configuration shift register that requires only 9234 bits, compared to several megabytes for commercial FPGAs.

Pipelined interconnect normally requires complicated PAR algorithms [7] to ensure related nets have the same routing delays. To simplify the problem, we add realignment registers to the inputs of each computational resource, as shown in Fig. 3(b). The realignment registers are variable shift registers that delay a signal up to eight cycles. As long as related nets are not misaligned by more than eight cycles, PAR can use traditional algorithms. For delays greater than eight cycles, the netlist can be altered to explicitly use the IF delay components.

The IF uses the window generator from [5], which buffers image rows in block RAMs. The window generator continually shifts buffered data into shift registers to create a new window each cycle. To add reconfigurabilty to the window generator, we added padding to support any image size up to 1080×1920. The generator also supports any window size up to 8×8, but requires manual padding in software due to application-specific padding requirements.

IF address generators and memory controllers are the same as a normal circuit. The IF also includes a PCIe controller (not

shown) to communicate with a host processor, although we could use any interface (e.g., those used by soft processors).

The controller for the IF is simplified compared to more general-purpose circuits due to the specialization for pipelined image-processing kernels. Pipelined circuits often have simple control that first enables the address generators, stalls the pipeline whenever inputs are not available or when the output memory buffer is full, and signifies when a circuit has completed execution. To implement this control, the IF uses a state machine with registers that control input sizes.

B. Tools

To implement the IF on the FPGA, we created an IF generation tool that reads an XML description of the fabric and outputs synthesizable VHDL.

To compile an application onto the IF, we created IF PAR tools. The placer uses VPR [2] with modified simulated annealing parameters that we empirically determined to be effective for this level of granularity. The router uses Pathfinder [10], which does not support pipelined routing, but as mentioned earlier, the realignment registers allow the router to ignore nets that are misaligned by eight or less cycles.

Currently, we have no synthesis tools for IFs and therefore manually created technology-mapped IF netlists. We could potentially synthesize from any HDL, and also plan to port OpenCL synthesis tools onto IFs for runtime compilation.

IV. Limitations

IFs are not intended for all usage scenarios due to overhead that may be too significant for small embedded systems or large high-performance systems. Fortunately, rapidly increasing FPGA sizes continually lessen the impact of this overhead. Furthermore, by mapping IFs directly to physical routing resources, device vendors can potentially eliminate all muxes used for routing, which represents almost all overhead.

Flexibility is also a potential limitation. A custom circuit can use any mixture of operators and precisions. An IF circuit can also include any mixture, but it is unlikely that highly specialized IFs would exist in a predetermined library. In this situation, a designer could create a custom IF as we did by simply defining an appropriate XML fabric description. While this approach does require a single FPGA PAR iteration to implement the IF, that PAR time is amortized over the lifetime of the IF. To reduce the time for custom IF generation, we plan to investigate rapid IF generation via bitfile relocation.

Certain application domains will likely not benefit from IFs. For example, control-intensive applications clearly cannot be implemented on the IF used in this letter. We could create an IF for control applications, but fast IF PAR comes from coarse granularities. Fortunately, control circuits tend to have small compilation times even without IFs.

V. Experiments

In this section, we evaluate the discussed IF using three common image-processing kernels: 2-D convolution, Sobel edge detection, and sum-of-absolute differences (SAD). 2-D convolution multiplies each window of an image with a corresponding kernel (e.g., set of coefficients) and then accumulates the products. Sobel edge detection is a common filter that first performs two 3×3 convolutions, squares the result of each, adds the squares, and then takes the square root. SAD is used for determining image similarity by summing the absolute differences of each window and image pixel. For each kernel, we use a 1080×1920 image that software has converted to 16-bit grayscale. For 2-D convolution, we evaluate window sizes ranging from 3×3 to 8×8. Sobel uses 3×3 windows and SAD uses 8×8 windows (although other sizes are also applicable). For this IF, the window size is limited to 8×8 because of the 128 inputs.

In addition to the previously described IF, we also created a 32-bit floating-point IF that was limited to 5×5 windows due to the larger floating-point area requirements.

For each example, we compared an IF circuit with custom VHDL. For synthesis, we used Quartus II 9.1 SP2, running on a quad-core 2.66 MHz Intel Xeon W3520 with 12 GB of RAM. The targeted board was a GiDEL PROCStarIII PCIe card with an Altera Stratix III E260 FPGA. The board was attached to a 2.26 GHz quad-core Intel E5520 that we used for preprocessing and data transfer. To compare performances, we calculated speedup compared to software for each kernel, which we compiled using $\mathrm{g++}$ 4.4.3 with $-\mathrm{O3}$ optimizations.

Table I compares the IF and direct FPGA implementations. As shown in the left column, IF PAR times were $700\times$ faster on average. The longest IF PAR time was only 5.3 s. PAR speedup for the floating-point examples was larger due to a larger reduction in PAR input size.

Performance overhead is shown in the middle of Table I. IF clock frequencies are shown at the bottom (124 MHz for the fixed-point IF and 114 MHz for the floating-point IF). The average IF clock overhead was 17%, but the average performance overhead was only 7% due to slow PCIe data transfers that were independent of this clock. Actual speedup compared to software averaged $8.3\times$ for the IF and $8.8\times$ for the direct implementations. Sobel was slower on the FPGA in all cases due to PCIe transfer overhead.

The right column of Table I shows area utilization of the direct FPGA implementations. The applications ranged from using 13% to 25% of the LUTs. The fixed-point and floating-point IFs used 57% and 59%, respectively. Although the pessimistic mux-based IF required $2.4\times$ to $4.4\times$ more area, the extra area is not necessarily overhead because some of the resources could be used for other computation (e.g., larger IF circuits). The LUT percentages that could not be used for computation (i.e., muxes for routing resources, configuration registers) were 43% for the fixed-point IF and 34% for the floating-point IF, due to the smaller fabric size. As mentioned earlier, by mapping the IF directly onto physical routing resources, all of the muxes could be eliminated, which represents almost all of the overhead.

The fixed-point IF bitfile used 9 234 bits. The floating-point bitfile was 3600 bits because of the smaller fabric size. We stored the bitfiles in 128-bit wide block RAM, which enabled IF reconfiguration in 28 to 72 cycles. Partial reconfiguration on commercial devices typically takes on the order of tens of milliseconds. Furthermore, the IF reconfiguration occurred on an Altera device that does not support partial reconfiguration.

TABLE I
SUMMARY OF PLACE AND ROUTE TIMES, PERFORMANCE OVERHEAD, AND AREA UTILIZATION (% OF TOTAL RESOURCES)

	Place and Route Times			Performance					Area Utilization		
	IF	*Quartus 9.1*	*Speedup*	*Clk FPGA*	*Clk Overhead*	*Speedup IF*	*Speedup FPGA*	*Perf. Overhead*	*LUT*	*REG*	*DSP*
Conv 3x3	0.9s	14min 48s	**943**	150 MHz	17%	3.2	3.5	**9%**	13%	14%	1%
Conv 4x4	1.5s	15min 06s	**613**	148 MHz	16%	5.9	6.3	**6%**	13%	14%	2%
Conv 5x5	2.1s	15min 33s	**447**	146 MHz	15%	8.0	8.5	**6%**	13%	14%	4%
Conv 6x6	3.0s	15min 41s	**312**	151 MHz	18%	11.1	11.9	**7%**	13%	15%	5%
Conv 7x7	4.0s	16min 19s	**243**	139 MHz	11%	14.7	15.5	**5%**	13%	15%	8%
Conv 8x8	5.3s	16min 08s	**184**	146 MHz	15%	18.8	20.0	**6%**	13%	15%	8%
Sobel	4.2s	14min 56s	**214**	154 MHz	19%	0.53	0.58	**9%**	13%	14%	1%
SAD 8x8	5.3s	16min 51s	**190**	143 MHz	13%	18.6	19.5	**5%**	15%	16%	0%
Conv 5x5 (float)	1.7s	25min 28s	**919**	148 MHz	23%	5.2	5.5	**5%**	21%	29%	13%
Sobel (float)	1.5s	18min 58s	**759**	144 MHz	21%	0.32	0.36	**11%**	16%	19%	3%
SAD 5x5 (float)	0.6s	30min 43s	**2880**	140 MHz	19%	5.3	5.5	**4%**	25%	38%	0%
Average	2.7s	18min 14s	**700**	146 MHz	17%	8.3	8.8	**7%**	15%	18%	4%
IF				124 MHz					57%	58%	17%
IF (float)				114 MHz					59%	61%	13%

Fast reconfiguration also counteracts area overhead by enabling multiple circuits to be swapped into the IF, as opposed to implementing the circuits in a single FPGA bitfile. Assuming an IF provides all necessary resources, the number of circuits is limited only by the amount of memory. The evaluated Stratix III E260 has more than 15 million bits of memory, which supports more than 1 600 IF circuits.

VI. CONCLUSION

In this letter, we evaluated an intermediate fabric for common image-processing kernels, which enabled placement and routing speedups averaging $700\times$ compared to vendor tools. The main limitation is overhead, which was a modest 7% for performance, and a more significant 34% to 44% for area. However, FPGA vendors could eliminate such area overhead by directly mapping virtual routing resources onto physical FPGA resources. Finally, we showed that intermediate fabrics enable fast partial reconfiguration, even on devices lacking physical support.

ACKNOWLEDGMENT

The authors would like to acknowledge vendor equipment and/or tools provided by Altera, Nallatech, and Xilinx.

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 *Proc. Int. Workshop Field-Program. Logic Appl. (FPL'97)*, London, U.K., 2007, pp. 513–516.

[2] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. Int. Workshop Field-Program. Logic Appl. (FPL'97)*, London, U.K., 1997, pp. 213–222.

[3] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign Syst. Synth. (CODES/ISSS'10)*, Scottsdale, AZ, 2010, pp. 13–22.

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

[5] Y. Dong, Y. Dou, and J. Zhou, "Optimized generation of memory structure in compiling window operations onto reconfigurable hardware," in *Proc. Int. Symp. Appl. Reconfigurable Comput. (ARC'07)*, Rio De Janeiro, Brazil, 2007, pp. 110–121.

[6] C. Ebeling, D. C. Cronquist, and P. Franklin, "Rapid—Reconfigurable pipelined datapath," in *Proc. Int. Workshop Field-Program. Logic Appl. (FPL'96)*, Darmstadt, Germany, 1996, pp. 126–135.

[7] K. Eguro and S. Hauck, "Armada: Timing-driven pipeline-aware routing for FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Array. (FPGA'06)*, Monterey, CA, 2006, pp. 169–178.

[8] 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 *Proc. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM'06)*, Napa Valley, CA, 2006, pp. 205–216.

[9] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "HMFlow: Accelerating FPGA compilation with hard macros for rapid prototyping," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM'11)*, Salt Lake City, UT, 2011.

[10] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM Int. Symp. Field Program. Gate Array. (FPGA'95)*, Monterey, CA, 1995, pp. 111–117.

[11] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Array. (FPGA'01)*, Monterey, CA, 2001, pp. 29–36.

[12] B. E. Nelson, M. J. Wirthlin, B. L. Hutchings, P. M. Athanas, and S. Bohner, "Design productivity for configurable computing," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA'08)*, Las Vegas, NV, 2008, pp. 57–66.

[13] S. Shukla, N. W. Bergmann, and J. Becker, "Quku: A two-level reconfigurable architecture," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerging VLSI Technol. Arch. (ISVLSI'06)*, Karlsruhe, Germany, 2006.

[14] 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.