ARCHITECTURAL MODIFICATIONS TO IMPROVE FLOATING-POINT UNIT EFFICIENCY IN FPGAS

Michael J. Beauchamp, Scott Hauck

University of Washington
Electrical Engineering
Box 352500
Seattle, Washington 98195
email: {mjb7, hauck}@ee.washington.edu

Keith D. Underwood, K. Scott Hemmert

Sandia National Laboratories*
Scalable Computing Systems
PO Box 5800, MS-1110
Albuquerque, New Mexico 87185
email: {kdunder, kshemme}@sandia.gov

ABSTRACT

FPGAs have reached densities that can implement floating-point applications, but floating-point operations still require a large amount of FPGA resources. One major component of IEEE compliant floating-point computations is variable length shifters. They account for over 30% of a double-precision floating-point adder and 25% of a double-precision multiplier. This paper introduces two alternatives for implementing these shifters. One alternative is a coarse-grained approach: embedding variable length shifters in the FPGA fabric. These units provide significant area savings with a modest clock rate improvement over existing architectures. Another alternative is a fine-grained approach: adding a 4:1 multiplexer inside the slices, in parallel to the LUTs. While providing a more modest area savings, these multiplexers provide a significant boost in clock rate with a small impact on the FPGA fabric.

1. INTRODUCTION

While modern supercomputers depend almost exclusively on a collection of traditional microprocessors, these microprocessors have poor sustained performance on many modern scientific applications [1]. FPGAs may provide an alternative, but scientific applications depend on IEEE compliant floating-point computations for numerical stability and reproducibility of results. Increases in FPGA density, and optimized floating-point unit designs, have made it possible to implement a variety of scientific algorithms with FPGAs [2], [3], [4]. In spite of this, there are still significant opportunities to improve the performance of FPGAs on scientific applications by optimizing the device architecture.

Floating-point units (FPUs) can be embedded in the FPGA fabric to provide a large area savings and increased clock rates for floating-point based kernels, but they also consume 17.6% of the chip [5]. An alternative is to focus on less area intensive enhancements to the FPGA fabric that improve floating-point units.

A fundamental component in floating-point arithmetic is a variable length and direction shifter. In floating-point addition, the mantissas of the operands must be aligned before the computation. For full IEEE compliance, floating-point multiplication and division require normalization of the mantissa before and after the calculation [6]. Shifters require a series of multiplexers, which are currently implemented using LUTs. In our double-precision floating-point cores, the shifter accounts for almost a third of the adder and a quarter of the multiplier. Thus, better support for variable length shifters can noticeably improve floating-point performance.

We consider two approaches that optimize the FPGA hardware for variable length shifters. The first approach is to embed shifters in the FPGA logic. Recent FPGAs have included embedded units including embedded multipliers, block RAMs, and even full microprocessors. Embedding variable length shifters allows configurable logic in the FPGA to be used for other purposes and yields a large area savings. The trade-off is an increase in the silicon area that is used for dedicated functionality that may not be used by all applications. The second approach is to modify the general purpose logic of the FPGA by adding a 4:1 multiplexer in parallel with the traditional LUT. This approach decreases the area required to implement shifters in the general purpose logic of the FPGA without wasting a significant amount of silicon area.

To test these concepts, we modified VPR to support embedded functional units and high-performance carry-chains. VPR was used to place and route benchmarks that use double-precision floating-point operations. The benchmarks included matrix multiply, matrix vector multiply, vector dot product, FFT, and LU decomposition. Each benchmark was tested using three versions of the FPGA. The first is similar to the Xilinx Virtex II Pro [7], and is representative of current commercial devices. The second adds embedded shifters, while the third uses modified CLBs that have the additional 4:1 multiplexer.

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

2. BACKGROUND

The IEEE-754 standard [8] specifies the floating-point numbers used on most computing platforms. Floating-point numbers consist of sign, mantissa, and exponent. The mantissa, f, is multiplied by the base number (two) to an exponent, e, as shown in Equation 1 (double-precision).

$$X = (-1)^S \cdot 1.f \cdot 2^{E-1023} \qquad (1)$$

Compliance with the IEEE double-precision format is important for cross-platform portability and verifiability; double-precision also improves numerical stability.

Double-precision floating-point has a sign bit, an 11-bit exponent and a 52-bit mantissa. Since the mantissa is normalized to the range [1,2), there will always be a leading one on the mantissa. The implicit leading one gains a single bit of precision, but raises the complexity of floating-point implementations. The exponent is represented in a biased notation. All stored numbers are "positive", but have been "biased" by half the exponent range. This representation simplifies floating-point comparators.

The dominant style of FPGAs is the island-style FPGA consisting of a two dimensional lattice of CLBs (Configurable Logic Blocks). Connecting the CLBs are regular horizontal and vertical routing structures that allow configurable connections at the intersections. In recent years, embedded RAMs, DSP blocks, and even microprocessors have been added to the island-style FPGAs. However, floating-point applications still require a large number of CLBs to perform basic operations.

3. VPR

VPR [9] is the leading public-domain FPGA place and route tool. It uses simulated annealing, a timing based semi-perimeter routing estimate for placement, and a timing driven router. VPR was used to determine the feasibility of the proposed techniques.

A similar version of VPR was used to test the feasibility of coarse grained embedded floating-point units [5]. In addition to previous VPR supported types of input pads, output pads, and CLBs, the modified version of VPR supports multiple embedded blocks. These embedded blocks have parameterizable heights and widths that are quantized by the size of the CLB. Horizontal routing is allowed to cross the embedded units, but vertical routing only exists at the border of the embedded blocks. As in previous work [5], fast carry-chains were added to insure a reasonable comparison. The synthesis and technology mapping approach are covered in the methodology section.

The baseline FPGA architecture was modeled after the Xilinx Virtex-II Pro family of FPGAs and includes most of the major elements of current FPGAs (CLBs, 18Kb block RAMs, and embedded 18-bit x 18-bit multiplier blocks). The CLBs include 4-input function generators, storage elements, arithmetic logic gates, and a fast carry-chain. In addition to the standard Xilinx Virtex-II Pro features, the architecture incorporates embedded shifters and a modified CLB with a 4:1 multiplexer in parallel with the LUT. The various blocks were arranged in a column based architecture (Fig. 1.) similar to modern FPGAs (e.g. [10]).

Table 1. Ratio of resources for different configurations

Configuration	CLB	RAM	MULT	SHIFT
CLB & MULT	36	1	1	0
Embedded Shifters	60	1	2	1
CLB w/ 4:1 mux	28	1	1	0

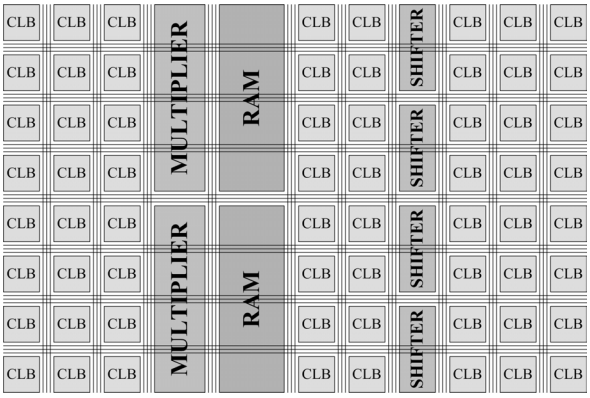

Fig. 1. Column based architecture

The ratio of the number of columns of each component type was based on the average requirement for the benchmarks and is shown in Table 1. These ratios are based on the average resource requirements, so each benchmark be constrained by a different limiting resource. The percent of each resource used for each benchmark is given in Table 2. through Table 4. where the limiting components for each benchmark are shaded.

The embedded units and fast carry chain are dedicated resources that use their own timing parameters. Embedded units can have optional registered inputs and/or registered outputs, and are characterized by two timing parameters: sequential setup time and sequential clock-to-q. The dedicated route of the carry chain also has its own timing.

3.1. Component Latency and Area

The CLBs that were used are comparable to the Xilinx slice. Each CLB has two 4-input function generators, two D flip-flops, arithmetic logic gates, and a fast carry-chain. VPR uses subblocks to specify the internal contents of the CLB. Representing the timing of an unmodified CLB required twenty VPR subblocks. The 4:1 multiplexer modification added two VPR subblocks. Each subblock can specify a combinational and sequential logic element and has three timing parameters, similar to the embedded

Table 2. FPGA resource usage (in percent) for baseline

Benchmark	CLB	RAM	MULT
Matrix Mult.	75	100	75
Vector Mult.	89	3	100
Dot Product	89	0	100
FFT	84	100	50
LU	91	67	100

Table 3. FPGA resource usage (in percent) with shifter

Benchmark	CLB	RAM	MULT	SHIFT
Matrix Mult.	40	100	38	33
Vector Mult.	87	6	100	89
Dot Product	87	0	100	89
FFT	40	100	25	39
LU	66	100	75	70

Table 4. FPGA resource usage (in percent) with 4:1 mux

Benchmark	CLB	RAM	MULT
Matrix Mult.	93	100	75
Vector Mult.	100	3	96
Dot Product	100	0	97
FFT	95	100	50
LU	100	63	94

Table 5. Embedded component timing and area

	T_{SETUP} [ns]	$T_{CLK \rightarrow Q}$ [ns]	Area [10^6 L^2]	Area [CLBs]
CLB	0.32	0.38	0.662	1
Embedded Multiplier	2.06	2.92	11.8	18
RAM	0.23	1.50	18.5	28
Shifter	0.3	0.7	0.843	1.27

units: sequential setup time, sequential clock-to-q, and maximum combinational delay. These timing parameters were found in Xilinx data sheets or using Xilinx design tools and are shown in Table 5. We also used two embedded units that were modeled after the Xilinx Virtex-II Pro: an 18x18 bit multiplier and an 18 Kb RAM. However, like the Xilinx Virtex-4 [11], these units are independent of each other. The timing parameters of the embedded multipliers and RAMs are based on the Xilinx Virtex-II Pro -6 and are shown in Table 5.

The area (including routing) of the CLB, embedded multiplier, and embedded RAM were approximated using a die photo of a Xilinx Virtex-II 1000 courtesy of Chipworks, Inc. The areas were normalized by the process gate length. All areas are referenced to the smallest component (the CLB) and are shown in Table 5.

3.2. Track Length & Delay

Four different routing track lengths were used: single, double, quad, and long, where long tracks spanned the entire chip. The ratio of routing tracks (11:14:21:4) was modeled after the Xilinx Virtex-II Pro. The delay of routing in VPR is calculated based on a resistive and capacitive model. Appropriate values for the routing track segments were found experimentally by laying out and extracting them using Cadence IC design tools.

3.3. Embedded Shifter

For floating-point operations, the mantissa can be shifted by any distance up to the full length of the mantissa. Thus, up to 53 bits of shift can be required for IEEE double-precision, but shifters tend to be implemented in powers of two. Therefore, shifters of length 32 (for single precision) and 64 bits were implemented as shown in Fig. 6.

The embedded shifter was designed with five modes (shift left, rotate left, shift right logical, shift right arithmetic, and rotate right) to increase versatility. In addition, the normalization shifting in floating-point units requires calculating a sticky bit. The sticky bit is the logical OR of all of the bits that are lost during a logical right shift. The logic to calculate the sticky bit is included in each shifter as it adds less than 1% to the shifter size.

The embedded shifter has a total of 83 inputs and 66 outputs. The 83 inputs include 16 control bits, 64 data bits, and 3 register control bits (clock, reset, and enable). The 66 outputs include 64 data bits and 2 sticky bits (two independent sticky bit outputs are need when the shifter is used as two independent 32-bit shifters). The I/O connections are evenly distributed around the periphery of the shifter and connect to CLB-like connection blocks.

The benchmark circuits use the embedded shifters in the fully registered mode, so only sequential setup time (300 ps) and sequential clock-to-q (700 ps) were needed:. Internally, the combinational delay of the shifter was only 1.52 ns. The sequential times were derived from similar registered embedded components of the Xilinx Virtex-II Pro -6, while the combinational time and the area (0.843 10^6 L^2) were derived by doing a layout in a 130 nm process. The area is 1.27 times the size of the CLB and its associated routing, but it does not take into account the area needed for additional connections (relative to a CLB) or the area needed for connections to the routing structure. Because this area overhead is difficult to estimate, three different shifter sizes (two, four, and eight equivalent CLBs) were considered. There were only trivial differences in the area and clock rate results (data not shown), so this analysis uses size four, which have more connections than one shifter.

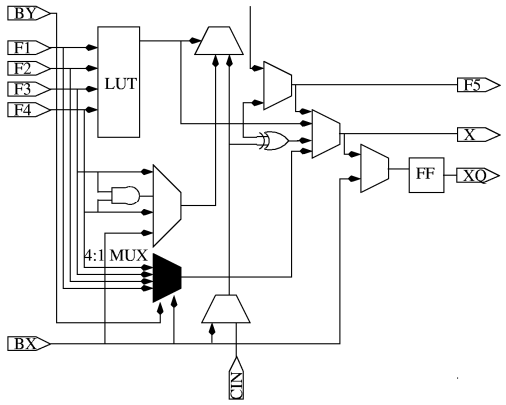

Fig. 2. Bottom half of the CLB with 4:1 mux

3.4. Multiplexer

The fine-grained optimization attempts to enhance shifting without impacting the general routing. To accomplish this, the only change that was made to the CLB was to add a single 4:1 multiplexer in parallel with each 4-LUT as shown in Fig. 2. The multiplexer and LUT share the same four data inputs. The select lines for the multiplexer are the BX and BY inputs to the CLB. Since each logic block using the unmodified Xilinx Virtex II Pro slice has two LUTs, each CLB would have two 4:1 multiplexers that share their select lines. For shifters and other large datapath elements it is easy to find muxes with shared select inputs. The BX and BY inputs are normally used as the independent inputs for the D flip-flops. This new usage prohibits that and requires that the input to the D flip-flops be from the logic within the CLB. This trade-off prevents increasing the number of inputs to the CLB.

To test the impact of adding the 4:1 multiplexer, a 4-LUT and associated logic was laid out and simulated with and without the capacitive load of the 4:1 multiplexer. Adding the 4:1 multiplexer increased the delay of the 4-LUT by only 1.83%. The delay of the 4:1 multiplexer was 253 ps, which is less than the 270 ps for the 4-LUT from the Xilinx Virtex-II Pro -6 datasheet. The area of the 4:1 multiplexer was 1.58 10^9 L^2, and adding two of them to each CLB increases the size of the CLB by less than 0.5%.

4. METHODOLOGY

Five benchmarks were used to test the feasibility of the proposed modifications. They were matrix multiply, matrix vector multiply, vector dot product, FFT, and an LU decomposition datapath. All of the benchmarks use double-precision floating-point addition and multiplication, and LU decomposition includes floating-point division. Each benchmark was tested in three FPGA versions. The first version is representative of a modern FPGA and includes a combination of CLBs and the embedded 18-bit x 18-bit embedded multipliers The second version adds an embedded variable length shifter to the baseline, and the third version augments the baseline with a 4:1 multiplexer in parallel with the LUT.

Table 6. Components for baseline architecture

Benchmark	CLB	I/O	MULT	RAM
Matrix Mult.	41,502	195	144	192
Vector Mult.	36,926	2,034	144	4
Dot Product	36,737	1,492	144	0
FFT	34,745	590	72	144
LU	37,634	193	144	96

Table 7. Components for enhanced architectures

	Embedded Shift		Modified CLBs	
	CLB	SHIFT	CLB	CLBs w/ 4:1 Mux
Matrix Mult.	36,483	64	39,894	9.9%
Vector Mult.	30,207	64	33,604	11.7%
Dot Product	30,018	64	33,403	11.8%
FFT	27,907	56	30,777	11.7%
LU	30,506	67	34,145	12.2%

The floating-point benchmarks were written in a hardware description language, either VHDL or JHDL [12]. The benchmarks were synthesized using Synplicity's Synplify 7.6 into an EDIF file. Technology mapping was performed with Xilinx ISE 6.3. While these are slightly older tools, the floating-point units were already hand mapped and so only small parts of the design were synthesized and/or mapped. The Xilinx *NGDBuild* and the Xilinx *map* tool were used to reduce the design from gates to slices (which map one-to-one with our CLBs). The Xilinx *NCDRead* was used to convert the design to a text format. A custom program converted the mapping of the NCD file to the NET format used by VPR.

The benchmarks vary in size and complexity. Table 6. gives the number of components for the benchmarks in the baseline architecture. The number of IO, block RAMs, and embedded multipliers remain constant for all three versions of the benchmarks. Table 7. gives the number of CLBs and embedded shifters for the benchmark versions that use the embedded shifters. Table 7. also shows the number of CLBs for each benchmark version that uses the modified CLBs and the percentage of the CLBs that make use of the 4:1 multiplexer modification. Using embedded shifter reduces the average number of CLBs by 17.3%. Similarly, the 4:1 multiplexor provides an 8.4% reduction.

5. TESTING & ANALYSIS

Even with an extremely conservative estimate of the embedded shifter size, adding embedded shifters to modern FPGAs significantly reduced circuit size. As seen in Fig. 3. and Fig. 4. , adding embedded shifters reduces average area

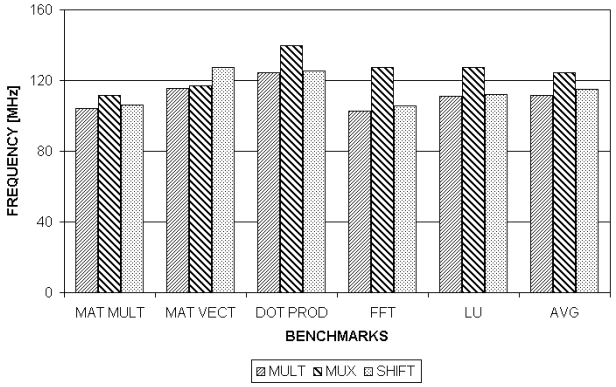

Fig. 3. Benchmark clock rates

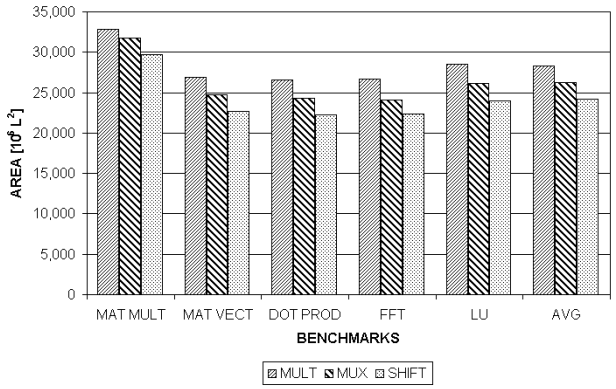

Fig. 4. Benchmark areas

by 14.6% and increases average clock rate by 3.3% compared to the baseline benchmarks that only used CLBs and embedded multipliers to perform floating-point computations. There was an average increase of 16.5% (not shown) in the number of routing tracks used, but this is still well within the limits of modern FPGAs (less than 70).

Only the floating-point units were optimized with the embedded shifters – the control and the reminder of the data path remained unchanged. If we consider only the units, the embedded shifters reduced the number of CLBs for each double-precision floating-point addition by 31% and required two embedded shifters. For the double-precision floating-point multiplication, the number of CLBs decreased by 22% and two embedded shifters were used as shown in Fig. 5.

Use of the 4:1 multiplexer modification to the CLB also showed significant improvements. Even though only the floating-point cores were optimized, there was an area savings of 7.3% over the reference benchmarks. In addition to the area savings, there was a speed increase of 11.6%, as seen in Fig. 3. and Fig. 4. Numerous multiplexers are known to exist in VHDL datapaths outside of the floating-point units, and so the size of this advantage should grow if this modification was exposed to the synthesis flow. If we consider only the floating point units, the addition of the

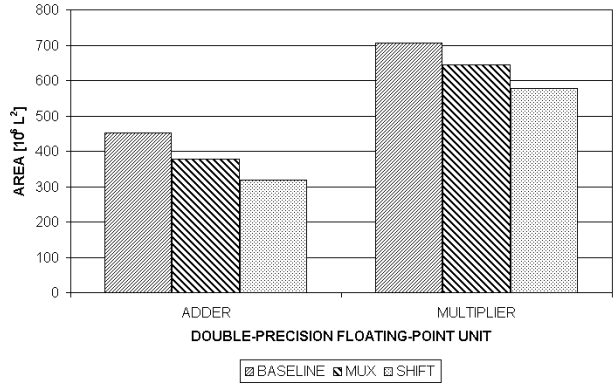

Fig. 5. Double-precision floating-point unit area

multiplexer reduced the size of the double-precision floating-point adder by 17% and reduced the size of the double-precision multiplier by 10% as shown in Fig. 5.

6. RELATED WORK

While there has not been a great deal of work dedicated to increasing the efficiency of floating-point operations on FPGAs, there has been some work that might be beneficial to floating-point operations on FPGAs. Ye showed the benefits for bus-based routing for datapath circuits [13]. Because IEEE floating-point numbers have 32 or 64 bits (single or double-precision) and these signals will generally follow the same routing path. This naturally lends itself to bus-based routing.

Xilinx recently announced their next generation of FPGAs; the Virtex-5 replaces the 4-LUTs with 6-LUTs [15]. These 6-LUTs would clearly offer the same advantage as using a dedicated 4:1 mux, but would also consume somewhat more area. It is likely that the 6-LUTs would be more flexible than the dedicated 4:1 mux.

The embedded multipliers in the Xilinx architectures can also implement shifters, but this approach is infeasible in modern designs where the multipliers are consumed by the floating-point units to do multiplication. Xilinx AppNote 195 also implies that a 56 bit shift would be an inefficient technique with regards to silicon area.

7. CONCLUSION

The results indicated that adding shifters to the fabric or 4:1 multiplexers to the CLBs will significantly reduce circuit size for floating-point applications with an increase in circuit frequency. The embedded shifter provided an average area savings of 14.6% and a clock rate increase of 3.3%. The 4:1 multiplexer provided an average area savings of 7.3% while achieving an average speed increase of 11.6%. Neither modification significantly increased track count. The embedded shifters are only 1.5% of the

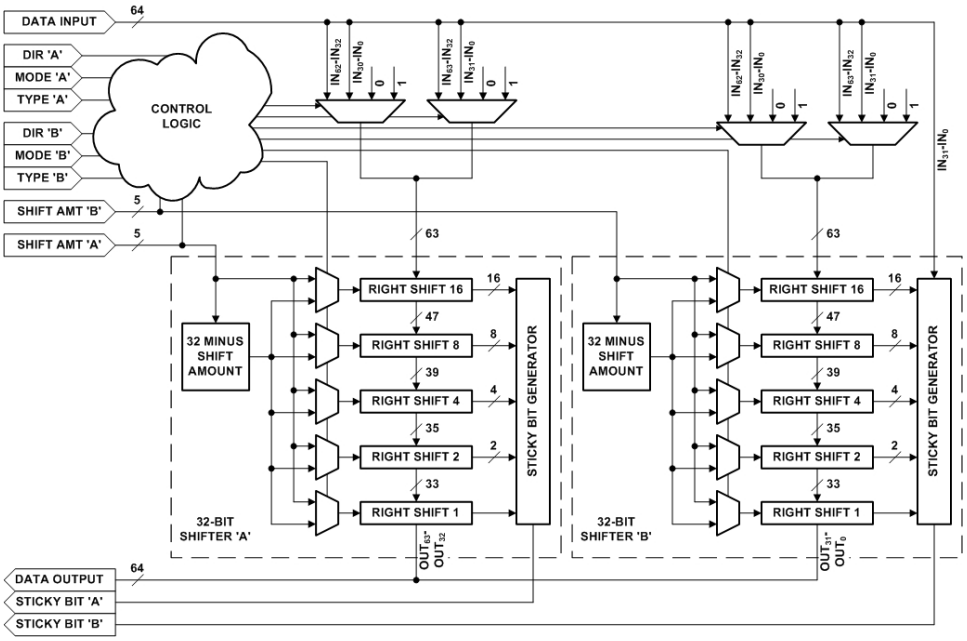

Fig. 6. Embedded shifter block diagram

total chip area and the 4:1 multiplexer composed 0.48% of the CLB and 0.35% of the total chip area.

8. REFERENCES

[1] K. D. Underwood. FPGAs vs. CPUs: Trends in Peak Floating-Point Performance. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 2004.

[2] K. S. Hemmert and K. D. Underwood. An Analysis of the Double-Precision Floating-Point FFT on FPGAs. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA 2005.

[3] M. de Lorimier and A. DeHon. Floating point sparse matrix-vector multiply for FPGAs. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 2005.

[4] L. Zhuo and V. K. Prasanna. Sparse matrix-vector multiplication on FPGAs. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 2005.

[5] M. J. Beauchamp, S. Hauck, K. D. Underwood, K. S. Hemmert. Embedded Floating-Point Units in FPGAs. In Proceedings of the ACM International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 2006.

[6] K. S. Hemmert and K. D. Underwood. Open Source High Performance Floating-Point Modules. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, 2006.

[7] Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. June 2005 (Rev 4.3), [cited Aug 2005], http://direct.xilinx.com/ bvdocs/ publications/ ds083.pdf.

[8] IEEE Standards Board. IEEE standard for binary floating-point arithmetic. Technical Report ANSI/IEEE Std. 754-1985, The Institute of Electrical and Electronic Engineers, New York, 1985.

[9] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp 213-222, 1997.

[10] Xilinx: ASMBL Architecture. 2005 [cited Sept 2005], http://www.xilinx.com/products/silicon_solutions/fpgas/ virtex/virtex4/overview/

[11] Virtex-4 Data Sheet: DC and Switching Characteristics. Aug 2005 (Rev 1.9), [cited Sept 2005], http://direct.xilinx. com/bvdocs/publications/ds302.pdf

[12] B. Hutchings, P. Bellows, J. Hawkins, K. S. Hemmert, B. Nelson, and M. Rytting. A CAD Suite for High-Performance FPGA Design. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1999.

[13] A. Ye, J. Rose, "Using Bus-Based Connections to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits," In *Proceeding of the ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2005.

[14] Virtex-5 LX Platform Overview. May 12, 2006 (Rev 1.1), [cited May 2006], http://direct.xilinx.com/bvdocs/ publications/ ds100..pdf