Architectural Modifications to Enhance the Floating-Point Performance of FPGAs

Michael J. Beauchamp, Scott Hauck, Keith D. Underwood, and K. Scott Hemmert

***Abstract*—With the density of FPGAs steadily increasing, FPGAs have reached the point where they are capable of implementing complex floating-point applications. However, their general-purpose nature has limited the use of FPGAs in scientific applications that require floating-point arithmetic due to the large amount of FPGA resources that floating-point operations still require. This paper considers three architectural modifications that make floating-point operations more efficient on FPGAs. The first modification embeds floating-point multiply-add units in an island style FPGA. While offering a dramatic reduction in area and improvement in clock rate, these embedded units have the potential to waste significant silicon for non-floating-point applications. The next two modifications target a major component of IEEE compliant floating-point computations: variable length shifters. The first alternative to LUTs for implementing the variable length shifters is a coarse-grained approach: embedded variable length shifters in the FPGA fabric. These shifters offer a significant reduction in area with a modest increase in clock rate and a relatively small potential for wasted silicon. The next alternative is a fine-grained approach: adding a 4:1 multiplexer unit inside the slices, in parallel to the 4-LUTs. While this offers the smallest overall area improvement, it does offer a significant improvement in clock rate with only a trivial increase in the size of the CLB.**

***Index Terms*—Reconfigurable architecture, Floating-Point arithmetic, FPGA**

I. Introduction

A variety of research efforts are searching for alternative processor architectures to accelerate scientific applications. While modern supercomputers depend almost exclusively on a collection of traditional microprocessors, these microprocessors have poor sustained performance on many modern scientific applications. ASICs, which can be highly efficient at floating-point computations, do not have the programmability needed in a general purpose supercomputer. Even though microprocessors are versatile and have fast clock rates, their performance is limited by their lack of customizability [1]. One alternative that is being widely considered is the use of FPGAs. However, scientific applications depend on complex floating-point computations that could not be implemented on FPGAs until recently, due to size constraints. Increases in FPGA density, and optimizations of floating-point elements for FPGAs, have made it possible to implement a variety of scientific algorithms with FPGAs [2]-[7]. In spite of this, the floating-point performance of FPGAs must increase dramatically to offer a compelling advantage for the scientific computing application domain. Fortunately, there are still significant opportunities to improve the performance of FPGAs on scientific applications by optimizing the device architecture.

Because fixed-point operations have long since become common on FPGAs, FPGA architectures have introduced targeted optimizations like fast carry-chains, cascade chains, and embedded multipliers. In fact, Xilinx has created an entire family of FPGAs optimized for the signal processing domain, which uses this type of operation intensively [12]. Even though floating-point operations are becoming more common, there have not been the same targeted architectures for floating-point as there are for fixed-point – there is not a scientific-computing family of FPGAs.

Potential architectural modifications span a spectrum from the extremely coarse-grained to the extremely fine-grained. This paper explores ideas at three points in that spectrum. At the coarse-grained end, we evaluate the addition of fully compliant IEEE 754 standard [8] floating-point multiply-add units as an embedded block in the reconfigurable fabric. These are feasible because many scientific applications require compliance with the IEEE standard from any platform they use. These coarse-grained units provide a dramatic gain in area and clock rate at the cost of dedicating significant silicon resources to hardware that not all applications will use.

IEEE floating-point also has other features that lend themselves to finer grained approaches. The primary example is that floating-point arithmetic requires variable length and direction shifters. In floating-point addition, the mantissas of the operands must be aligned before calculating the result. In floating-point multiplication and division, the mantissa must be shifted before the calculation (if denormals are supported) and after the calculation to renormalize the mantissa [9]. The datapath for shifters involves a series of multiplexers, which are currently implemented using LUTs. In our highly

This work was supported in part by Sandia National Laboratories and the National Science Foundation.

M. J. Beauchamp is with the Electrical Engineering Department, University of Washington, Seattle, WA 98195 USA (phone: 206-221-6943; fax: 206-543-3842; e-mail: mjb7@ee.washington.edu).

S. Hauck is with the Electrical Engineering Department, University of Washington, Seattle, WA 98195 USA (hauck@ee.washington.edu).

K. D. Underwood is with the Scalable Computing Systems, Sandia National Laboratories, Albuquerque, NM 87185 USA (kdunder@sandia.gov).

K. S. Hemmert is with the Scalable Computing Systems, Sandia National Laboratories, Albuquerque, NM 87185 USA (kshemme@sandia.gov).

optimized double-precision floating-point cores for FPGAs [9], the shifter accounts for almost a third of the logic for the adder and a quarter of the logic for the multiplier. Thus, better support for variable length shifters can noticeably improve floating-point performance.

This led us to consider two approaches to optimizing the FPGA hardware for variable length shifters. At the fine grained end, we consider a minor tweak to the traditional CLB: the addition of a 4:1 multiplexer in parallel with the 4-LUT. This provides a surprisingly large clock rate improvement with a more modest area improvement and virtually no wasted silicon area. In the middle of the spectrum, we consider the addition of an embedded block to provide variable length shifting. This uses slightly more area than the CLB modification and provides a corresponding increase in area savings. Surprisingly, however, it provides only a modest improvement in clock rate.

To test these concepts, VPR was augmented to support embedded functional units and high-performance carry-chains. It was then used to place and route benchmarks that use double-precision floating-point multiplication and addition. The five benchmarks that were chosen were matrix multiply, matrix vector multiply, vector dot product, FFT, and LU decomposition. To determine the feasibility of these proposed architectural modifications, five versions of each benchmark were used: CLB only, embedded multiplier, embedded shifter, multiplexer, and embedded floating-point units.

The remainder of this paper is organized as follows. Section 2 presents background information on the floating-point numbering system and current FPGA architecture. A description of how VPR was modified and used to test the feasibility of the proposed FPGA architectural changes is presented in Section 3. Section 4 presents the testing methodology. The parameters for the embedded units are presented in Section 5. Results and analysis are presented in Section 6. Finally, Section 7 presents related work and Section 8 the conclusion.

II. Background

A. Floating-Point Numbering System

The IEEE-754 standard specifies a representation for single and double precision floating-point numbers and is currently the standard that is used for real numbers on most computing platforms. Floating point numbers consist of three parts: sign bit, mantissa, and exponent. They are arranged such that the mantissa, f, is multiplied by the base number (two) to an exponent, e, as shown in Equations 1 and 2, single and double precision respectively [8], [10].

$$X = (-1)^S \cdot 1.f \cdot 2^{E-127} \tag{1}$$

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

The IEEE standard specifies a sign bit, an 8-bit exponent, and a 23-bit mantissa for a single-precision floating-point number, as seen in Fig 1. Double-precision floating-point has a sign bit, an 11-bit exponent and 52-bit mantissa, as seen in Fig 2. Since the mantissa is normalized to the range [1,2) there will always be a leading one on the mantissa. By implying the leading one instead of explicitly specifying it, a single bit of precision is gained, but it does raise the complexity of floating-point implementations.

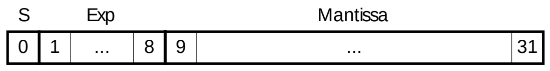


Fig 1. Single precision IEEE floating-point number

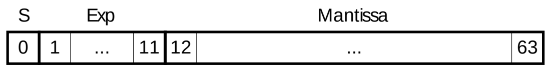


Fig 2. Double precision IEEE floating-point number

The exponent is represented in the somewhat unusual biased notation. It represents positive and negative numbers by having a bias equal to half of the exponent range. This representation somewhat simplifies floating-point comparators.

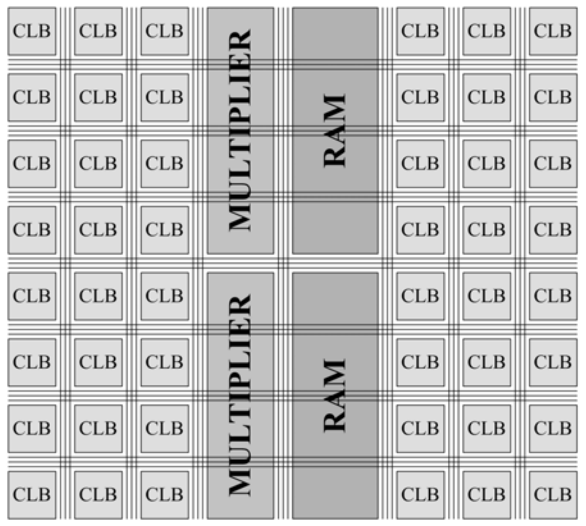
B. Island-Style FPGA

The current 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 additional block units have been added that have increased FPGAs versatility and efficiency, especially for circuits that use fixed-point numbers. Embedded RAMs, DSP blocks, and even microprocessors have been added to island-style FPGAs [11], [12]. However, circuits that use floating-point numbers still require a large number of CLBs to perform basic operations. Thus, this paper examines hardware changes and modifications that will improve the efficiency of floating-point unit implementations.

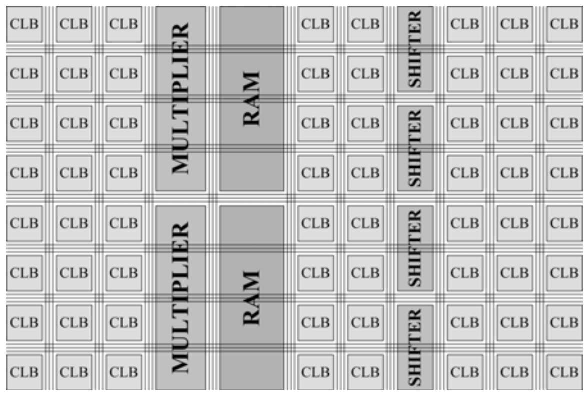
III. VPR

VPR [13], [14] is the leading public-domain FPGA place and route tool. It uses simulated annealing and a timing based semi-perimeter routing estimate for placement and a timing driven detailed router. In this paper, VPR was used to determine the feasibility of three changes to the traditional island-style FPGAs: embedding floating-point units (FPUs), embedded shifters, and modified CLBs to facilitate shifting for floating-point calculations.

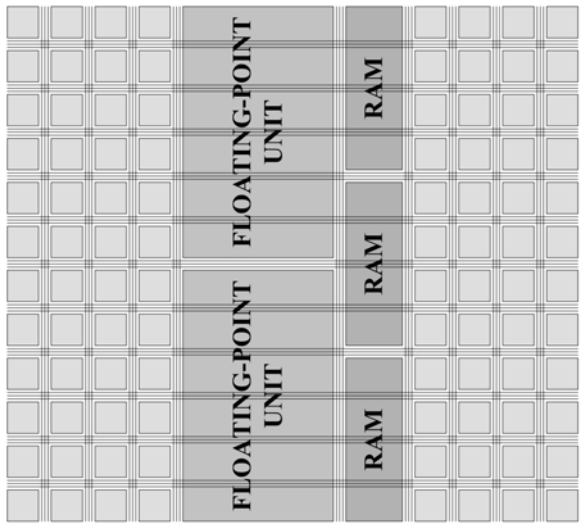
In previous versions, VPR supported only three types of circuit elements: input pads, output pads, and CLBs. To test the proposed architectural modifications and to incorporate the necessary architectural elements, VPR was modified to



(a)



(b)



(c)

Fig 3. (a) Column based architecture with CLBs, embedded multipliers, and block RAMs (b) Embedded shifters added to 'a' (c) Embedded floating-point units replacing multipliers in 'a'

allow the use of embedded block units of parameterizable size. 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 periphery of the embedded blocks. The regular routing structure that existed in the original VPR was maintained as shown in Fig 3. Additionally, a fast carry-chain was incorporated into the existing CLBs to insure a reasonable comparison with state-of-the-art devices. The benchmarks were synthesized using Synplify and technology mapped using Xilinx ISE (rather than the VPR technology mapping path). See methodology, Section 4, for more details.

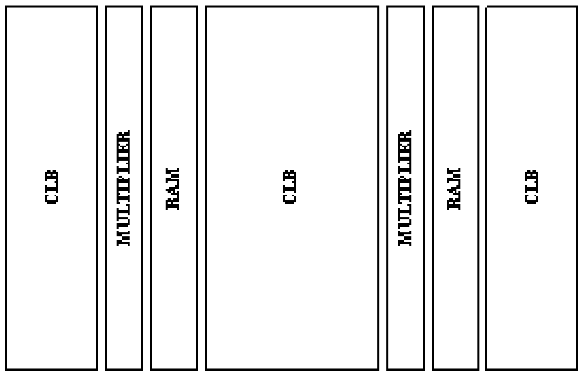


Fig 4. ASMBL

The baseline FPGA architecture was modeled after the Xilinx Virtex-II Pro FPGA family and includes most of the major elements of current FPGAs (IO blocks, CLBs, 18Kb block RAMs, and embedded 18-bit x 18-bit multiplier blocks) [11]. 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, alternative architectures incorporate embedded FPUs, embedded shifters, or a modified CLB with 4:1 multiplexer in parallel with the LUT. The various blocks were arranged in a column based architecture similar to Xilinx's ASMBL (Advanced Silicon Modular Block) architecture [15], as seen in Fig 4, which is the architectural foundation of the Virtex-4 FPGA family [16].

The embedded units have optional registered inputs and/or registered outputs. Each unit is characterized by three timing parameters: sequential setup time, sequential clock-to-q, and maximum combinational delay if unregistered. The fast carry-chain is a dedicated route that does not use the normal routing structure of switch boxes and connection boxes. The carry-chain has a dedicated route that goes from the carry-out at the top of the CLB to the carry-in at the bottom of the CLB above it. Because it does not make use of the normal routing graph, it has its own timing parameters.

A. Component Area

The areas of the CLB, embedded multiplier, and block RAM were approximated using a die photo of a Xilinx Virtex-II 1000 [17] courtesy of Chipworks Inc. The area estimate of each component includes the associated connection blocks, which dominate the routing area. The areas were normalized by the process gate length, L. All areas are referenced to the smallest component, which is the CLB, and are shown in Table 1.

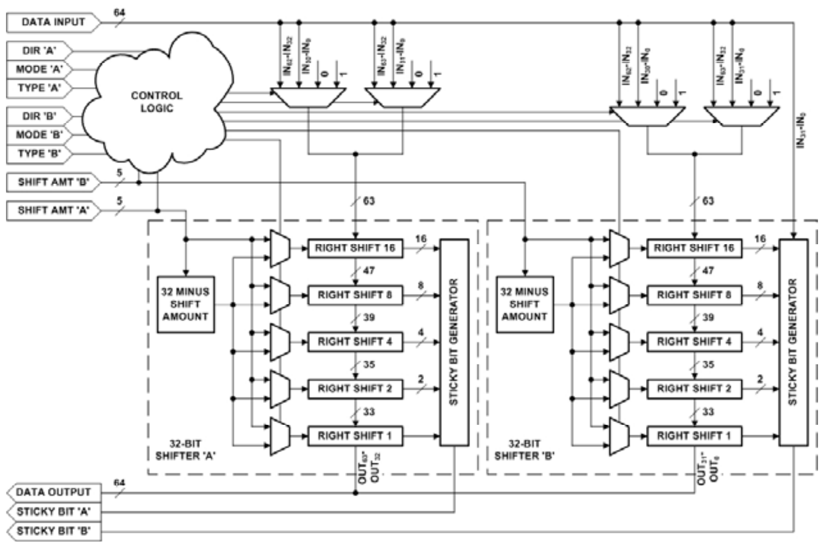


Fig 5. Embedded shifter block diagram

TABLE 1
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
Block RAM	0.23	1.50	18.5	28
Shifter	0.30	0.70	0.843	1.27
FPU	0.50	0.50	107	161

B. Component Latency

The CLBs that were used are comparable to the Xilinx slice. Each CLB is composed of two 4-input function generators, two storage elements (D flip-flops), arithmetic logic gates, and a fast carry-chain. VPR uses subblocks to specify the internal contents of the CLB. Each subblock can specify a combinational and sequential logic element and has three timing parameters, similar to the embedded units: sequential setup time, sequential clock-to-q, and maximum combinational delay if the output subblock is unregistered. More subblocks results in a more accurate timing representation of the CLB. To adequately represent the timing of an unmodified CLB, twenty VPR subblocks were used. With the 4:1 multiplexer modification to the CLB, twenty-two VPR subblocks were used. The embedded multiplier and block RAM were modeled after the Xilinx Virtex-II Pro. However, unlike the Xilinx Virtex-II Pro (and more similar to the Xilinx Virtex-4), these units are independent of each other. These timing parameters are based on the Xilinx Virtex-II Pro -6 and were found in Xilinx data sheets or experimentally using Xilinx design tools and are shown in Table 1.

C. Track Length and Delay

We use four different lengths of routing tracks: single, double, quad, and long, where long tracks spanned the entire length of the architecture. The percentages of different routing track lengths were based on Xilinx Virtex-II Pro family and can be seen in Table 2 [11].

TABLE 2
TRACK LENGTH

Size	Length	Fraction
Single	1	22%
Double	2	28%
Quad	4	42%
Long	All	8%

VPR uses a resistive and capacitive model to calculate the delay for various length routing tracks. Based on previously determined component area, the resistive and capacitive values were estimated by laying out and extracting routing tracks using Cadence IC design tools. Timing results for the overall design were found to be reasonable based on previous experience with Xilinx parts.

D. Embedded FPU

The embedded FPU implements a double-precision floating-point multiply-add operation as described by equation 3. The FPU can be configured to implement a double-precision multiply, add, or multiply-add operation.

$$X_{64-bit} = \left(A_{64-bit} * B_{64-bit}\right) + C_{64-bit} \tag{3}$$

The baseline size of the floating-point unit was conservatively estimated from commodity microprocessors and embedded cores in previous work [18], [19] and the actual area used was increased to accommodate one vertical side of the FPU being filled with connection blocks (assumed to be as large as a "CLB"). This made the true area of the FPU dependent on the shape chosen. We believe this to be an extremely conservative estimate.

The latency of the FPUs was more difficult to estimate appropriately. Given that a processor on a similar process (the

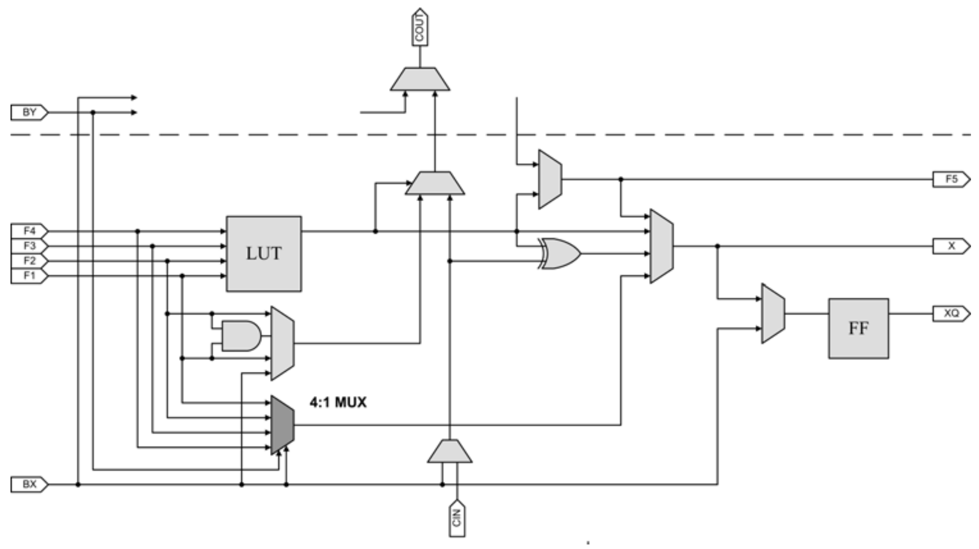


Fig 6. Simplified representation of bottom half of modified CLB showing addition of 4:1 multiplexer

Pentium 4) can achieve 3 GHz operation with a 4 cycle add and a 6 cycle multiply, we assume that an FPU implemented for an FPGA could achieve 500 MHz at the same latency. Setup and clock-to-q were set conservatively assuming the latency included registers front and back.

E. Embedded Shifter

For floating-point operations, the mantissa can be shifted either left or right, by any distance up to the full length of the mantissa. This means that up to a 24 bit shift can be required for IEEE single precision and up to 53 bits of shift can be required for IEEE double precision. However, in hardware, shifters tend to be implemented in powers of two. Therefore, shifters of length 32 and 64 bits were implemented for single and double precision floating-point operations, respectively, as shown in Fig 5.

Even though floating-point operations only require a logical shift, the embedded shifter should be versatile enough to be used for other circuits that do not necessarily have floating-point operations. Otherwise, the embedded shifters will have a higher probability of being wasted in other types of computations. Therefore, the embedded shifter that was used has five modes: shift left logical/arithmetic, rotate left, shift right logical, shift right arithmetic, and rotate right.

During the shifting that accompanies the normalization of floating-point numbers it is necessary to calculate a sticky bit. The sticky bit is the result of the logical OR of all of the bits that are lost during a right shift, and it is an integral part of the shift operation. Adding the necessary logic to the shifter to compute the sticky bit increases the size of the shifter by less than 1%. Thus, the logic for the sticky bit calculation is included in each shifter. The sticky bit outputs are undefined when a shift other than a logical right shift is performed.

The embedded shifter also has optional registers on the inputs and outputs of the datapath. There are 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 needed when the shifter is used as two independent 32-bit shifters).

The benchmark circuits use the embedded shifters in the fully registered mode, so only two timing parameters were needed: sequential setup time of 300 ps and sequential clock-to-q of 700 ps. Internally, the combinational delay of the shifter was 1.52 ns, which is far from the limiting timing path. The sequential times were derived from similar registered embedded components of the Xilinx Virtex-II Pro -6, while the combinational time and 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 the additional number of connection of the embedded shifter compared to the CLB, and 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 examined. There were only trivial differences in the area and clock rate results, so this analysis uses size four, since 4 CLBs have more connections than the shifters in question.

F. Multiplexer

In modifying the CLB to better implement variable length shifters, two general principles were observed: minimize the impact on the architecture, and have no impact on general purpose routing. To accomplish these goals, the only change that was made to the CLB's architecture was to add a single 4:1 multiplexer in parallel with each 4-LUT, as shown in Fig 6. 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 CLB on the Xilinx Virtex II Pro has two LUTs, each CLB would have two 4:1 multiplexers. Since there are only two select lines, both of the 4:1 multiplexers would have to share their select lines. However, 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, but are blocked in the new mux mode. However, the D flip-flops can still be driven by the LUTs in the CLB, and can be used as normal when not using the mux mode. This is a trade-off that is made to not increase 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. It was determined that adding the 4:1 multiplexer increased the delay of the 4-LUT by only 1.83%. A 4:1 multiplexer was also laid out and simulated. The delay of the 4:1 multiplexer was 253 ps, which is less than the 270 ps that was determined for the 4-LUT from the Xilinx Virtex-II Pro -6 datasheet. The area of the 4:1 multiplexer was 1.58 10^3 L^2, and adding two 4:1 multiplexers to each CLB increases the size of the CLB by less than 0.5% (the original area of the CLB takes into account all logic, routing, and control bits associated with the CLB as given in Table 1).

IV. Methodology

A. Benchmarks

Five benchmarks were used to test the feasibility of the proposed architectural modification. They were matrix multiply, matrix vector multiply, vector dot product, FFT, and a LU decomposition datapath. All of the benchmarks use double-precision floating-point addition and multiplication. The LU decomposition also includes floating-point division, which must be implemented in the reconfigurable fabric for all architectures. Five versions of the benchmarks were used.

- **CLB Only** – All floating-point operations are performed using the CLBs (Configurable logic Blocks). The only other units in this version are embedded RAMs and IO.
- **Embedded Multiplier** – This version adds 18-bit x 18-bit embedded multipliers to the CLB Only version. Floating-point multiplication uses the CLBs and the embedded multipliers. Floating-point addition and division are performed using only the CLBs. This version is similar to the Xilinx Virtex II Pro family of FPGAs, and thus is representative of what is currently available in commercial FPGAs.
- **Embedded Shifter** – This version further extends the Embedded Multiplier version with embedded variable length shifters that can be configured as a single 64-bit variable length shifter or two 32-bit variable length shifters. Floating-point multiplication uses the CLBs, embedded multipliers, and embedded shifters. Floating-point addition and division are performed using the CLBs and embedded shifters.
- **Multiplexer** – While the same embedded RAMs, embedded multipliers, and IO of the Embedded Multiplier version are used, the CLBs have been slightly modified to include a 4:1 multiplexer in parallel with the LUTs. Floating-point multiplication uses the modified CLBs and the embedded multipliers. Floating-point addition and division are performed using only the modified CLBs.
- **Embedded FPU** – Besides the CLBs, embedded RAMs, and IO of the CLB Only version, this version includes embedded floating-point units (FPUs). Each FPU performs a double-precision floating-point multiply-add. Other floating-point operations are implemented using the general reconfigurable resources.

The floating-point benchmarks were written in a hardware description language, either VHDL or JHDL [20]. Instead of using the traditional VPR path of synthesis using SIS [21] and technology mapping using Flow Map [22], the benchmarks were synthesized using Synplicity's Synplify 7.6 into an EDIF (Electronic Data Interchange Format) file. Technology mapping was accomplished using Xilinx ISE 6.3. While these are slightly older versions of the tools, only the Xilinx mapper was used and only small parts of the benchmarks were synthesized (the floating-point units were hand mapped in JHDL). Thus, we do not believe this had a negative impact on the results. The Xilinx *NGDBuild* (Native Generic Database) 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 NCD (Native Circuit Description) Read was used to convert the design to a text format. A custom conversion program was used to convert the mapping of the NCD file to the NET format used by VPR.

The benchmarks vary in size and complexity as shown in Table 3. The number of IO and block RAMs is the same for all benchmark versions. The number of embedded multipliers in the Embedded Shifter and Multiplexer versions is the same as the Embedded Multiplier version.

B. Fast Carry-Chains

VPR was also modified to allow the use of fast carry-chains. The CLBs were modeled after the Xilinx Virtex II Pro slice. Along with the two 4-input function generators, two storage elements, and arithmetic logic gates, each CLB has a fast carry-chain affecting two output bits. The carry-out of the CLB exits through the top of the CLB and enters the carry-in of the CLB above as shown in Fig 7. Each column of CLBs has one carry-chain that starts at the bottom of the column of CLBs and ends at the top of the column. Since each CLB has logic for two output bits, there are two opportunities in each CLB to get on or off of the carry-chain (Fig 7).

The addition of the carry-chain was necessary to make a reasonable comparison between the different benchmark versions. The versions of the benchmarks that implemented the floating-point multiply-add using the embedded multipliers or only CLBs make extensive use of the fast carry-chains. For example, the double-precision addition requires a 57 bit adder. If the carry signal was required to go out on general routing it would significantly decrease the adder frequency. This would dramatically skew the results in favor of the embedded FPUs.

To demonstrate the correct operation of the carry-chain modification, the benchmarks that used the embedded multipliers to implement the double-precision floating-point

TABLE 3
NUMBER OF COMPONENTS IN EACH BENCHMARK VERSIONS

Benchmark	CLB ONLY			EMBEDDED MULTIPLIER[†]		EMBEDDED SHIFTER[†‡]		MULTIPLEXER[†‡]		EMBEDDED FPU[†]	
	IO	RAM	CLB	CLB	MULT	CLB	SHIFT	CLB	CLBs Using 4:1 Mux	CLB	FPU
Matrix Mult.	195	192	56,973	41,502	144	36,483	64	39,894	9.9%	17,510	16
Vector Matrix Mult.	2,034	4	53,082	36,926	144	30,207	64	33,604	11.7%	11,250	16
Dot Product	1,492	0	51,924	36,737	144	30,018	64	33,403	11.8%	9,929	16
FFT	590	144	44,094	34,745	72	27,907	56	30,777	11.7%	15,432	28
LU Decomposition	193	96	56,093	37,634	144	30,506	67	34,145	12.2%	11,382	16
Maximum	*2,034*	*192*	*56,973*	*41,502*	*144*	*36,483*	*67*	*39,894*	*12.2%*	*17,510*	*28*
Average	*901*	*87*	*52,433*	*37,509*	*130*	*31,024*	*63*	*34,365*	*11.4%*	*13,101*	*19*

[†] Benchmarks include the same number of IOs and block RAMs as CLB ONLY version.
[‡] Benchmarks include the same number of embedded multipliers as the EMBEDDED MULTIPLIER version.

multiply-add were placed and routed using VPR with and without the carry-chain modification. The results are shown in Table 4. By using the fast carry-chain the benchmarks had an average speed increase of 49.7%.

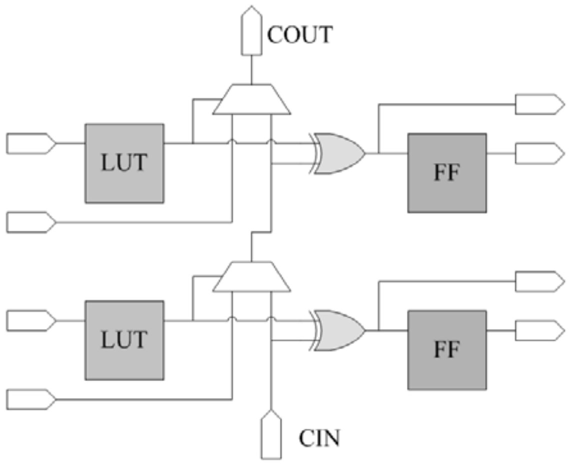


Fig 7. Simplified CLB with fast vertical carry-chain

TABLE 4
MAXIMUM CLOCK RATE WITH AND WITHOUT THE USE OF THE FAST CARRY-CHAIN

Benchmark	Max. Freq. w/o Fast Carry-Chain [MHz]	Max. Freq. with Fast Carry-Chain [MHz]
Matrix Multiply	87	126
Vector Multiply	89	117
Dot Product	87	149
FFT	79	104
LU Decomposition	84	142
Average	85	128

Because the carry-chains only exist in columns of CLBs and only in the upward direction, we initially place all of the CLBs of a given carry-chain in proper relative position to each other and move/swap all of the CLBs that comprise a carry-chain as one unit. To accomplish this, when a CLB that is part of a carry-chain is chosen to be moved or swapped the following algorithm is used:

1. The CLBs that are to be moved or swapped are randomly determined based on the constraints of the placement algorithm.
2. If the CLBs are part of a carry-chain the beginning and end of the carry-chain are determined.
3. Whatever carry-chain is the longer of the two CLBs to be swapped determines the number of CLBs to be swapped.
4. It is determined if the CLBs could be moved or swapped without violating the physical constraints of the chip and breaking any other carry-chain.
5. If the move swap is determined to be illegal, the move/swap is discarded and a new set of blocks are chosen for a potential move/swap. Even though this potential move is discarded, it is not considered a rejected move.
6. Once a legal move is found, all of the nets that connect to all of the CLBs that comprise the carry-chain are considered in the cost of moving the carry-chain.
7. The move is accepted or rejected based on the current simulated annealing temperature.
8. If a move/swap is accepted all of the CLBs on the carry-chain are moved together to maintain the physical constraints of the carry-chain architecture.
9. The accepted or rejected move of a carry-chain consisting of N CLBs is considered N accepted or rejected moves.

The rest of the details of the simulated annealing algorithm remain unchanged. This resulted in making VPR significantly slower, but is important for producing realistic comparisons.

V. PARAMETERS

A. Embedded FPU Aspect Ratio

To determine the appropriate aspect ratio for the FPU, each benchmark was run using eight different heights and widths. These FPUs with different aspect ratios were combined in a column based architecture with CLBs and block RAMs. With an increase in the height of the FPU (decrease in the aspect ratio), there will be fewer FPUs on a single column. To

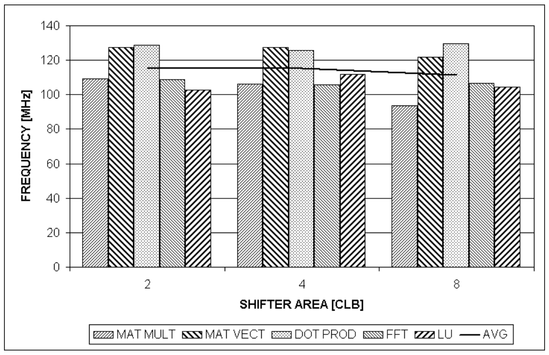


Fig 8. Embedded shifter benchmark clock rate

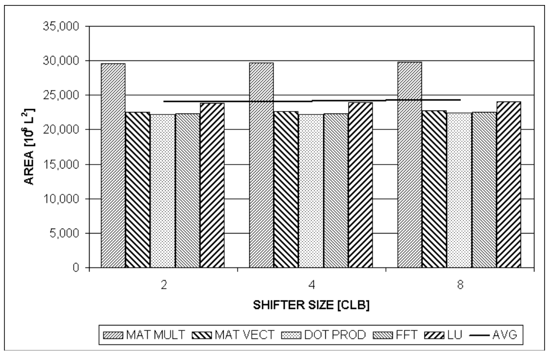


Fig 9. Embedded shifter benchmark area

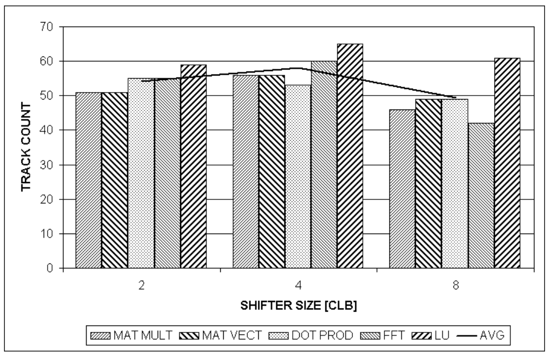


Fig 10. Embedded shifter benchmark track count

maintain the same ratio of FPUs, CLBs, and RAMs for all the different FPU sizes, the number of columns of FPUs was increased as the FPU height increased.

The area of the FPUs varies with the aspect ratio due to the overhead of connecting the FPU with the surrounding routing resources - for each row of CLBs along an FPU's edge, a row's worth of routing resources must be provided. A conservative estimate was used that for each CLB of height added to the FPU, an additional full CLB tile's worth of area was required for the programmable routing.

Each benchmark was tested with eight different FPU heights; from 4 CLBs to 160 CLBs in height. These benchmarks with different FPU sizes were compared on three criteria: area, maximum frequency, and number of routing tracks.

There is a significant difference in the maximum frequency between the benchmarks with different aspect ratio FPUs. The benchmarks with FPUs of height 32 had the highest average clock rate. The lower frequencies were found at the extremes, those with very large and very small aspect ratios. The benchmarks with large aspect ratios and small FPU heights were very wide and consequently had large horizontal routes that increased the overall circuit latency. The benchmarks with small aspect ratios and large FPU heights had congestion on the vertical routing tracks that led to high track counts and slower clock frequencies.

Because there was an area penalty for greater FPU heights to account for connectivity and routing, the architecture with the shortest FPUs had the smallest area. However, there was only a 2.7% difference in the areas of the FPU benchmark with the highest frequency and the benchmark with the smallest area.

Modern FPGAs have a large number of routing tracks. Therefore, apart from its impact on maximum clock frequency, the required number of routing tracks is unlikely to be the driving consideration when choosing the best aspect ratio for the FPU. Even though there was a 12.8% difference in track count of the FPU benchmark with the lowest track count (FPU height 16) and the benchmark with the highest clock rate (FPU height 32), the benchmark with the highest clock rate only had an average routing track count of 46.

On average, the benchmarks that used the FPU of height 32 had the highest frequency, did not have a significant area increase over those with other aspect ratios, and had a reasonable track count. Therefore, it is the architectures with FPUs of height 32 that are being compared to the other architectures. In general, this is approximately the "shape" we would expect to be "good". "Tall and narrow" or "short and wide" configurations seem unnatural for the implementation of floating-point units or for integration with a reconfigurable fabric.

B. Embedded Shifter Size

The embedded shifter has an equivalent area of 1.27 CLBs. However, this size does not take into account the area needed for the additional number of connections of the embedded shifter compared to the CLB, and the area needed for connections to the routing structure. Because an exact area comparison with existing FPGA architecture is not easy to make, three different shifter sizes were examined. Shifters of size two, four, and eight equivalent CLBs were tested and their results are given in Fig 8 through Fig 10.

There is only an average difference of 3.7% in clock rate and 1.0% in area between the different shifter sizes. Since four CLBs have more combined I/O connections than a shifter, we can use this for an extremely conservative estimate of the shifter size. Therefore, an embedded shifter size equivalent to four CLBs is used to compare with other benchmark versions.

VI. RESULTS

A. Embedded FPUs

The embedded FPU had the highest clock rate, smallest area, and lowest track count of all the architectures, as seen in

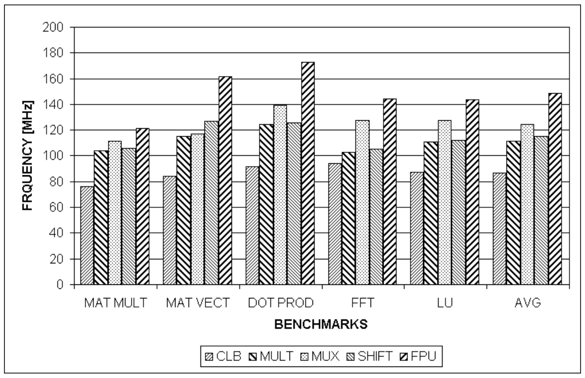


Fig 11. Benchmark clock rate

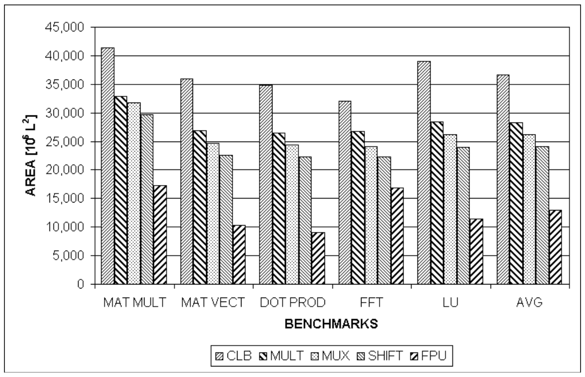


Fig 12. Benchmark area

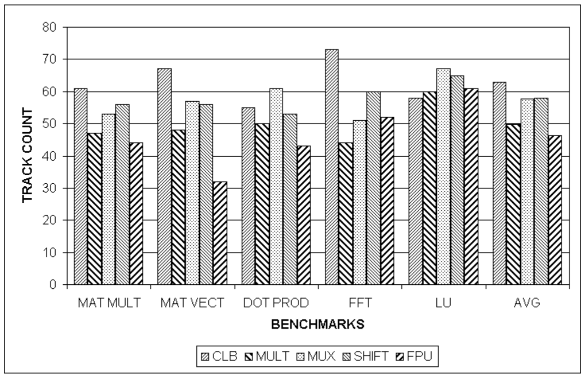


Fig 13. Benchmark track count

Fig 11 through Fig 13. By adding embedded FPUs there was an average clock rate increase of 33.4%, average area reduction of 54.2%, and average track count reduction of 6.83% from the EMBEDDED MULTIPLIER to the EMBEDDED FPU versions. To determine the penalty of using an FPGA with embedded FPUs for non floating-point computations, the percent of the chip that was used for each component was calculated. For the chosen FPU configuration, the FPUs consumed 17.6% of the chip. This is an enormous amount of "wasted" area for non-floating-point calculations and would clearly be received poorly by that community; however, this generally mirrors the introduction of the PowerPC to the Xilinx architecture. Ultimately, embedded floating-point units would only likely be added if a "scientific application" series of FPGAs were added (much like the DSP series currently in the Xilinx Virtex4 family).

B. Embedded Shifters

Even with a conservative size estimate, adding embedded shifters to modern FPGAs significantly reduced circuit size. As seen in Fig 11 through Fig 13, adding embedded shifters increased the average clock rate by 3.3% and reduced the average area by 14.6% from the EMBEDDED MULTIPLIER to the EMBEDDED SHIFTER versions. Even though there was an average increase in the track count of 16.5%, a track count of 58 is well within the number of routing tracks on current FPGAs.

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

C. Modified CLBs with additional 4:1 Multiplexers

Using the small modification to the CLB architecture showed surprising improvements. Even though only the floating-point cores were optimized with the 4:1 multiplexers, there was an average clock rate increase of 11.6% and average area reduction of 7.3% from the EMBEDDED MULTIPLIER to the MULTIPLEXER versions. The addition of the multiplexer reduced the size of the double-precision floating-point adder by 17% and reduced the size of the double-precision multiplier by 10%. Even though there was an average increase in the track count of 16.1%, a track count of 58 is well within the number of routing tracks on current FPGAs.

D. Single vs. Double Precision

The computing usage at Sandia National Laboratories is oriented toward scientific computing which requires double-precision. It is because of this that the benchmarks were written using double-precision floating-point numbers. With some modification, a double-precision FPU could be configured into two single precision units, and should show similar benefits.

VII. 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 [23] showed the benefits for bus-based routing for datapath circuits. 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.

Altera Corporation's Stratix II has a logic architecture that consists of smaller LUTs that can be combined into two 6-LUTs if the two 6-LUTs share four inputs [24]. While 6-LUTs that share no more than two inputs would have been ideal for implementing a 4:1 multiplexer and possibly produced similar results to adding a 4:1 multiplexer, the fact that the Stratix II 6-LUT requires sharing of four inputs reduces the efficiency of the Stratix II for implementing shifters and thus performing floating-point operations. Xilinx just announced their next generation of FPGAs; the Virtex-5 will have true 6-LUTs [25]. While the details of the Virtex-5 are not know, it is expected that it could be used to implement a 4:1 multiplexer.

Another alternative to implement shifting is to use the embedded multipliers that are common in Xilinx architectures. Unfortunately, this approach is infeasible in modern designs where the multipliers are completely consumed by the floating-point units to do multiplication. Extending the techniques of Xilinx AppNote 195 to 56 bits would be an inefficient technique with regards to silicon area as 7 multipliers would be needed along with over 100 4-LUTs. That would not include the sticky bit which needs to be generated as well.

VIII. Conclusion

This paper has demonstrated three architectural modifications that make floating-point operations more efficient. Adding complete double-precision floating-point multiply-add units, adding embedded shifters, and adding a 4:1 multiplexer in parallel to the LUT, each provide an area and clock rate benefit over traditional approaches with different trade-offs.

At the most coarse-grained end of the spectrum is a major architectural change that consumes significant chip area, but provides a dramatic advantage. Despite a "worst case" area estimate, the embedded FPUs provided an average reduction in area of 54.2% compared to an FPGA enhanced with embedded 18-bit x 18-bit multipliers. This area achievement is in addition to an average speed improvement of 33.4% over using the embedded 18-bit x 18-bit multipliers. There is even an average reduction in the number of routing tracks required by an average of 6.8%.

The embedded shifter provided an average area savings of 14.3% and an average clock rate increase of 3.3%. At the finest-grain end of the spectrum, adding a 4:1 multiplexer in the CLBs provided an average area savings of 7.3% while achieving an average speed increase of 11.6%. The former comes at the cost of a slightly larger increase (1.5%) in the silicon area of the FPGA versus only a 0.35% increase in FPGA area for the latter change; however, neither of these changes is a significant amount of wasted spaces. It is somewhat surprising that the smaller change to the FPGA architecture amounts to the bigger net "win".

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. D. Underwood and K. S. Hemmert. Closing the gap: CPU and FPGA Trends in sustainable floating-point BLAS performance. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, 2004.

[3] 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.

[4] 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.

[5] G. Govindu, S. Choi, V. K. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar. A high-performance and energy efficient architecture for floating-point based lu decomposition on fpgas. In *Proceedings of the 11th Reconfigurable Architectures Workshop (RAW)*, Santa Fe, NM, April 2004.

[6] L. Zhuo and V. K. Prasanna. Scalable and modular algorithms for floating-point matrix multiplication on fpgs. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, NM, April 2004.

[7] 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.

[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] 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.

[10] I. Koren, Computer Arithmetic Algorithms, 2nd Edition, A.K. Peters, Ltd. Natick, MA 2002.

[11] 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.

[12] Virtex-4 Family Overview. June 2005 (Rev 1.4), [cited Sept 2005], http:// direct.xilinx.com/ bvdocs/ publications/ ds112.pdf.

[13] 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.

[14] V. Betz and J. Rose. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, Boston, MA 1999.

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

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

[17] Virtex-II Platform FPGAs: Complete Data Sheet. Mar 2005 (Rev 3.4), [cited Aug 2005], http://direct.xilinx.com/bvdocs/publications/ds031. pdf

[18] MIPS Technologies, Inc. 64-Bit Cores, MIPS64 Family Features. 2005, [cited Jan 2005], http://www.mips.com /content/Products/Cores/64-BitCores.

[19] J. B. Brockman, S. Thoziyoor, S. Kuntz, and P. Kogge. A Low Cost, Multithreaded Processing-in-Memory System. In *Proceedings of the 3rd workshop on Memory performance issues*, Munich, Germany, 2004.

[20] 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.

[21] E. Sentovich et al, "SIS: A System for Sequential Circuit Analysis," *Tech Report No. UCB/ERL M92/41*, University of California, Berkley, 1992.

[22] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. CAD*, Jan 1994, pp 1-12.

[23] 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.

[24] D. Lewis, et al, "The Stratix II Logic and Routing Architecture," In *Proceeding of the ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2005.

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