Architectural Modifications to Enhance the Floating-Point Performance of FPGAs

Michael J. Beauchamp

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in Electrical Engineering

University of Washington

2006

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Architectural Modifications to Enhance the Floating-Point Performance of FPGAs

Chair of the Supervisory Committee:
Professor W. H. Carl Ebeling
Computer Science and Engineering

With the density of Field programmable Gate Arrays (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 thesis 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 (Look Up Tables) 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 reduction in overall area, it does offer a significant increase in clock rate with only a minimum increase in the size of the CLB (Configurable Logic Block).

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Table Number Page

ACKNOWLEDGEMENTS

The author wishes to express appreciation to National Science Foundation and Sandia National Laboratories† for their support and research funding, Keith D. Underwood and K. Scott Hemmert of Sandia National Laboratories for their guidance, support, editing, and creation of the benchmarks, Scott Hauck for his leadership, guidance, patience, and encouragement, and family and friends for their encouragement and devotion, without them, this thesis would never have been completed.

† 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.

DEDICATION

To my family and friends.

1. 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 become common on FPGAs, FPGA architectures have introduced targeted optimizations for fixed-point operations 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, addition of embedded units, to the extremely fine-grained, addition or modification of logic gates. This thesis explores ideas at three points in that spectrum. At the coarse-grained end, the addition of fully compliant IEEE 754 standard [8] floating-point

multiply-add units was evaluated as an embedded block in the reconfigurable fabric. These embedded floating-point units are feasible because many scientific applications require compliance with the IEEE standard from any platform they use. These coarse-grained units provide a dramatic reduction in area and increase in 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 Underwood and Hemmert's highly 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. Therefore, by developing a more efficient implementation of a variable length an direction shifter can noticeably improve floating-point performance.

This led to two approaches in optimizing the FPGA hardware for variable length shifters. At the fine-grained end, a minor change to the traditional CLB (Configurable Logic Block): the addition of a 4:1 multiplexer in parallel with the 4-LUT was considered. This provides a large increase in clock rate with a more modest area reduction and virtually no wasted silicon area. In the middle of the spectrum, the addition of an embedded block to provide variable length shifting was considered. This uses slightly more area than the CLB modification and provides a corresponding increase in area savings. Unlike the embedded floating-point units, the embedded shifters provide only a modest improvement in clock rate.

To test these three proposed architectural modifications to the FPGA architecture, the leading public-domain academic FPGA place and route tool, VPR (Versatile Place and

Route), was augmented to support embedded functional units and high-performance carry-chains. It was then used to place and route five scientific 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** – All floating-point operations are performed using the CLBs. The only other units in this version are embedded RAMs and IO blocks.
- **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 blocks of the EMBEDDED MULTIPLIER version are used, the CLBs have been slightly modified to include a 4:1 muliplexer 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 blocks 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 EMBEDDED SHIFTER, MULTIPLEXER, and EMBEDDED FPU benchmark versions implement the proposed architectural modifications to enhance floating-point performance on FPGAs. In order to measure the benefit of these modifications, these benchmark versions will be compared to the EMBEDDED MULTIPLIER version, which is representative of what is currently available in commercial FPGAs.

The maximum clock rate, or frequency, and area will be compared to quantify the benefit of these three proposed architectural modifications. While track count will also be measured, it is not considered for potential improvement but to ensure that the place and routes generated by VPR are reasonable.

The remainder of this thesis is broken down as follows: Section 2 presents a background of the floating-point numbering system and island-style FPGAs. Section 3 details how VPR was modified and used to place and route the benchmarks. Section 4 gives the specifics of the three proposed FPGA architectural modifications. Section 5 presents the results, Section 6 gives some related work, and Section 7 is the conclusion.

2. Background

2.1. Floating-Point Numbering System

Often, for scientific calculations, the range of fixed-point numbers is insufficient. Therefore, floating-point numbers are used. The larger dynamic range of floating-point numbers comes at a cost of more complex computations over fixed-point numbers. Floating-point numbers are similar to scientific notation. Floating-point numbers consists of two parts, the mantissa M and exponent E, as seen in equation 1, where β is the base or radix of the number.

$$X = M \cdot \beta^{E} \tag{1}$$

The base is consistent for all floating-point number of a given system. For binary floating-point numbers the base is 2. Both the mantissa and the exponent are signed numbers.

IEEE standard 754 specifies that the mantissa is stored in sign-magnitude notation with a sign bit S and an unsigned fraction M, which is normalized to the range [1,2). Since a normalized floating-point number will always have a leading one for the unsigned fraction M, it is not necessary to store this bit. The leading one becomes "hidden", allowing for an extra bit in the mantissa and thus, increasing the precision. The resulting mantissa is given in equation 2, where f is the fractional part of the mantissa after the leading one has been removed.

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

The exponent is a sum of the true two's complement value of the exponent and a constant bias, as given in equation 3.

$$E = E_{true} + bias \tag{3}$$

The range of E_{true}, where *e* is the bits of the exponent, is

$$-2^{e-1} \le E_{true} \le 2^{e-1} - 1$$

The bias, given in equation 4, is the magnitude of the most negative exponent, resulting in a non-negative exponent, which has a range

$$0 \le E \le 2^{e} - 1$$

$$bias = 2^{e-1} - 1 \tag{4}$$

By using a bias instead of another format, sign-magnitude or 2's complement, the exponent is always a non-negative value. This makes comparison of two exponents easier, as they can be considered unsigned numbers. Additionally, floating-point numbers in the order S, E, and M can be compared as if they were single sign-magnitude binary numbers.

Table 1. IEEE floating-point component lengths and exponent bias

Precision	Word Length [bits]	Mantissa Length† [bits]	Exponent Length [bits]	Exponent Bias
Single	32	23	8	127
Double	64	52	11	1023

† Does not include "hidden" bit.

IEEE standard 754 specifies that single precision binary floating-point numbers have 32-bits, which consists of a sign bit, a 23-bit mantissa (not including the hidden bit), and an 8-bit exponent, as shown in Table 1. The resulting representation is shown in equation 5.

$$X = (-1)^{S} \cdot 1.f \cdot 2^{E-127} \tag{5}$$

Double precision binary floating-point numbers have 64-bits, which consists of a sign bit, a 52-bit mantissa (not including the hidden bit), and an 11-bit exponent, as shown in Table 1. The resulting representation is shown in equation 6.

$$X = (-1)^S \cdot 1.f \cdot 2^{E-1023} \tag{6}$$

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 Figure 1. Double-precision floating-point has a sign bit, an 11-bit exponent and 52-bit mantissa, as seen in Figure 2.

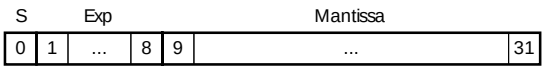


Figure 1. Single precision IEEE floating-point number

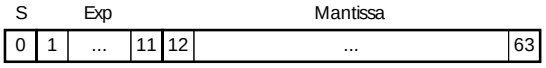


Figure 2. Double precision IEEE floating-point number

Floating-point numbers have a larger dynamic range that fixed-point numbers, but due to their complex formulation, they are more difficult to implement in hardware. It is this complexity that results in floating-point operations requiring a large percentage of resources in FPGAs.

2.2. Island-Style FPGA

FPGAs are an array of digital logic that can be programmed for specific tasks. Because of the large amount of logic, for applications, whose operations can be performed in parallel, FPGAs can be faster than general-purpose processors, while generally being less expensive than an Application-Specific Integrated Circuits (ASICs).

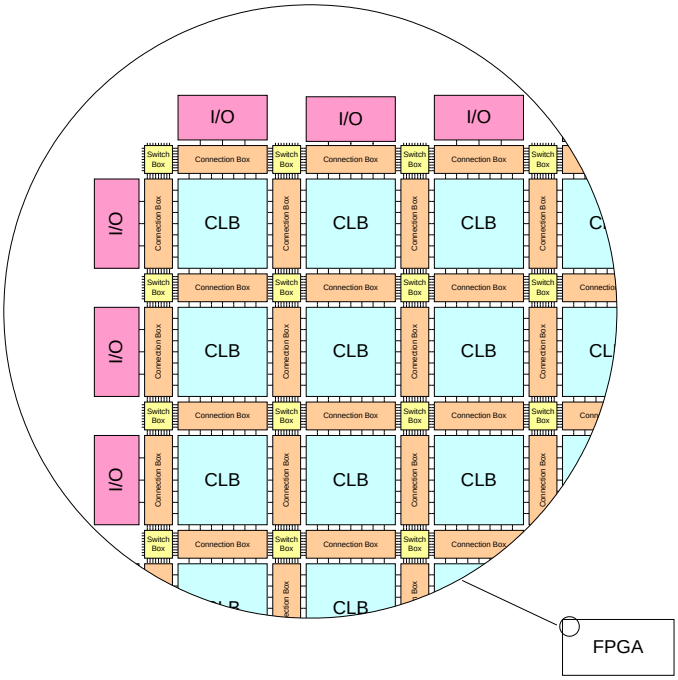


Figure 3. Basic island-style FPGA

The current dominant style of FPGAs is the island-style FPGA, consisting of a two dimensional lattice of CLBs (Configurable Logic Blocks), as shown in Figure 3. Around the outside of the FPGA are IO blocks. Connecting the CLBs are regular horizontal and vertical routing structures that allow configurable connections at the intersections. The horizontal and vertical routing is in the form of connection and switch boxes. The connection boxes allow programmable connection of the CLB's signals to enter and exit the general purpose routing. Switch boxes allow programmable connections between the connection boxes. The connection and switch boxes consist of programmable transistors that can connect wire segments. The CLBs consist of combinational and sequential logic components in the form of 4-input function generators (or 4-LUTs), storage elements (or flip-flops), arithmetic logic gates, and a fast carry-chain.

In recent years, additional block units have been added that have increased the versatility and efficiency of FPGAs, 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 thesis examines hardware changes and modifications to FPGAs that will improve the efficiency of floating-point unit implementations.

3. VPR

VPR [13], [14] is the leading public-domain academic 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 based on PathFinder [15]. In this thesis, 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.

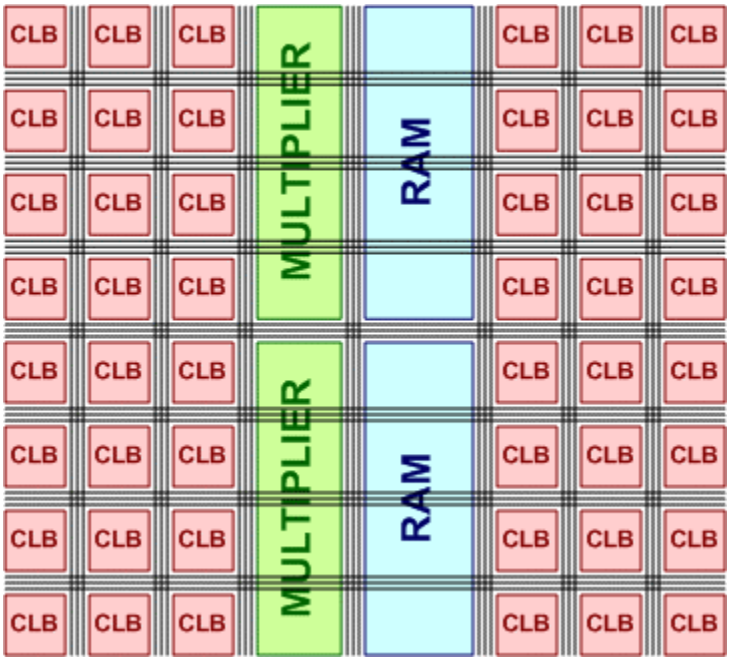


Figure 4. Column based architecture with CLBs, embedded multipliers, and block RAMs

Figure 5. Column based architecture with addition of embedded shifters

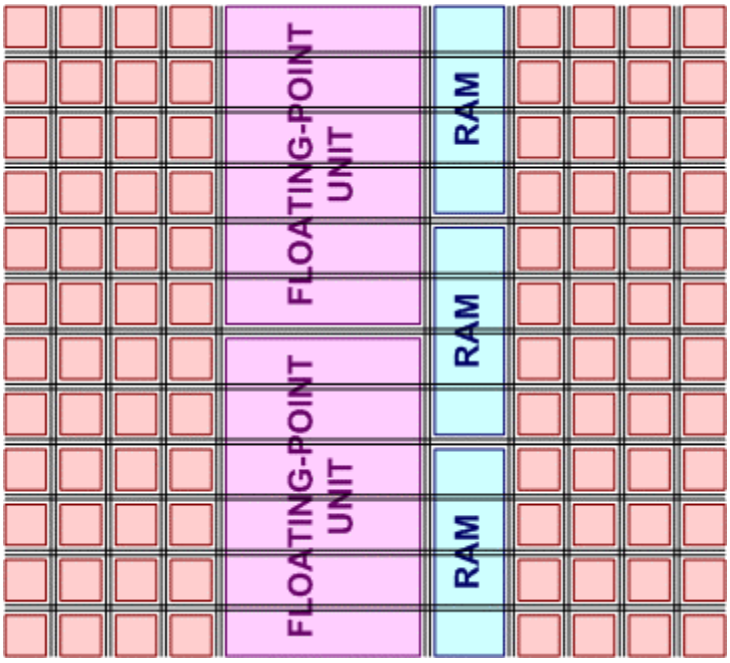


Figure 6. Embedded floating-point units replacing multipliers in Figure 4

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 allow the use of embedded block units with user defined size. The heights and widths of these embedded

blocks are quantized by the size of the CLB. For example the embedded RAM size can be specified as one CLB in width and four CLBs in height. In order for continuity of signals, horizontal routing is allowed to cross the embedded units, but because the embedded blocks are arranged in columns, 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 Figure 4 through Figure 6. Additionally, a fast carry-chain was incorporated into the existing CLBs to ensure a reasonable comparison with state-of-the-art devices. The fast carry-chain is explained in greater detail in Section 3.4. The benchmarks were synthesized using Synplify and technology mapped using Xilinx ISE (rather than the VPR technology mapping path). See benchmarks, Section 4.4, for more details.

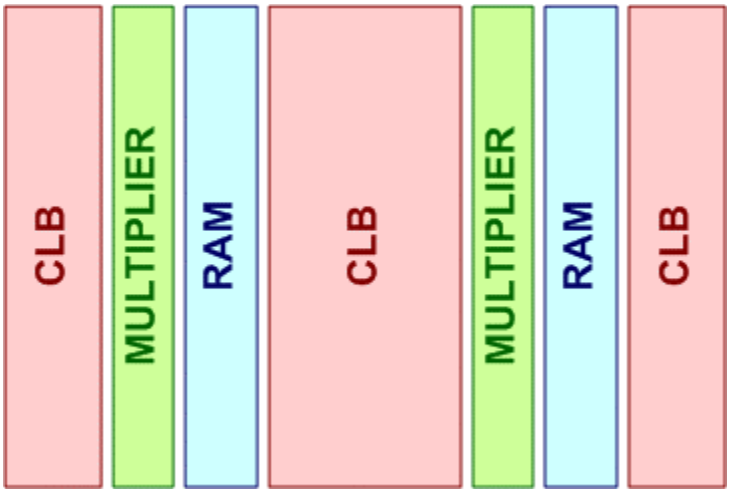


Figure 7. ASMBL

The baseline FPGA architecture, which proposed architectural modifications will be compared, 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, the proposed architectural modifications to the FPGA include embedded FPUs, embedded shifters, and the addition of a 4:1 mux in parallel to the 4-LUT. Independent of whatever embedded block is bein

used, they are arranged in a column-based architecture similar to Xilinx's ASMBL (Advanced Silicon Modular Block) architecture [16], as seen in Figure 7, which is the architectural foundation of the Virtex-4 FPGA family [17].

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.

3.1. Component Area

The areas of the CLB, embedded multiplier, and block RAM were approximated using a die photo of a Xilinx Virtex-II 1000 [18] 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 2.

Table 2. 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

3.2. Component Latency

The CLBs that were used were modeled after 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. This was done to ease the modeling of the embedded multipliers and block RAMs in VPR. 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 2.

3.3. Track Length and Delay

Island-style FPGAs use track lengths of various sizes. However, in the process of placing and routing VPR minimized the number of routing tracks used, but maintains a track length percentage as specified by the user. Four different lengths of routing tracks were used: 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 3 [11].

Table 3. 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.

3.4. Fast Carry-Chains

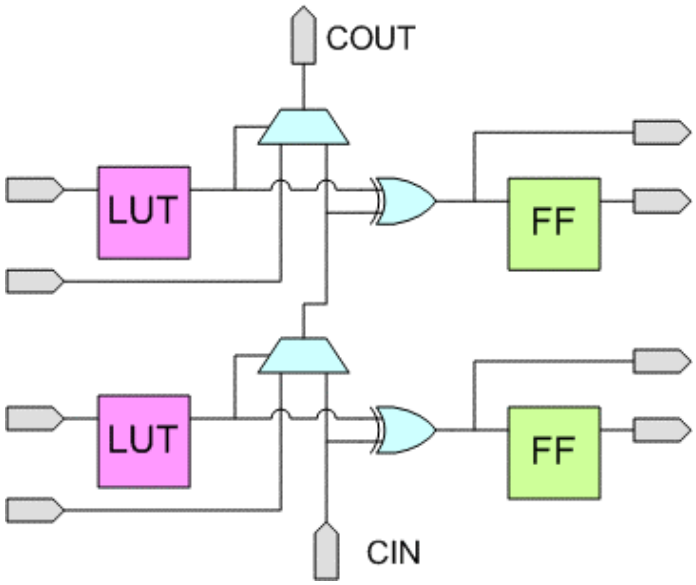


Figure 8. Simplified CLB with fast vertical carry-chain

VPR was also modified to allow the use of fast carry-chains. Along with the two 4-input function generators (4-LUT), two storage elements (flip-flops), 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 Figure 8. 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.

The addition of the carry-chain was necessary to make a reasonable comparison between the baseline FPGA architecture and the proposed FPGA architecture modifications, which include embedded FPUs, embedded shifters, and additional 4:1 mux in parallel to the 4-LUT in the CLB. Floating-point addition makes extensive use of the fast carry-chains. For example, the double-precision addition requires a 57-bit adder. The fast

carry-chain is specifically designed to implement a ripple carry adder and is much faster than if the carry signal was required to go out on the general routing structure. This would dramatically skew the results in favor of the embedded FPUs, which implement the floating-point addition in an embedded block and not in the CLBs.

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

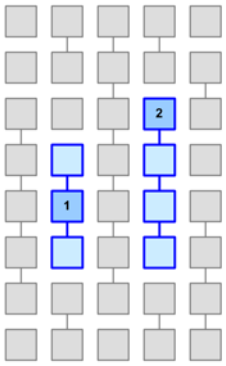
Table 4. Maximum clock rate with and without the use of the fast carry-chain

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, all of the CLBs of a given carry-chain are initially placed 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:

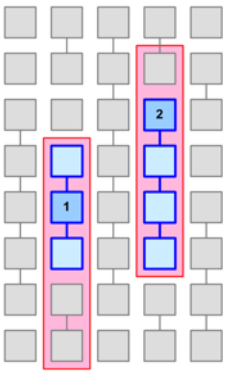
1. The CLBs that are to be moved or swapped are randomly determined based on the constraints of the placement algorithm.
2. If the CLBs are part of a carry-chain the beginning and end of the carry-chain are determined by traversing the carry-chain.
3. The number of CLBs traversed to find the beginning and end of each carry-chain are compared.

EXAMPLE: The first CLB chosen to be swapped has one CLB until the beginning of the carry-chain and one until the end of the carry-chain for a total of three CLBs (including the original CLB). The second CLB chosen to be swapped has two CLBs until the beginning of the carry-chain and none until the end of the carry-chain for a total of twelve (including the original CLB).

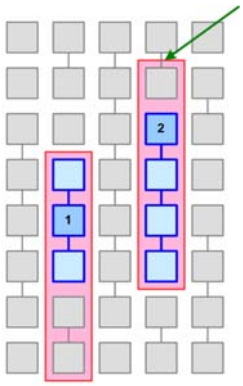


4. The larger of these values is used to determine how many CLBs will be considered for the move.

EXAMPLE: The first CLB chosen to be swapped has one CLB until the beginning of its carry-chain and the second CLB chosen to be swapped has three CLBs until the beginning of its carry-chain. The maximum distance to the beginning of the two carry-chains is three CLBs and this determines that three CLBs before each of the two CLBs originally chosen to be swapped will be included in the attempted swap. The first CLB chosen to be swapped has one CLBs until the end of its carry-chain and the second CLB chosen to be swapped does not have any CLBs until the end of its carry-chain. The maximum distance to the end of the two carry-chains is one CLB and this determines that one CLBs after each of the two CLBs originally chosen to be swapped will be included in the attempted swap. Therefore, five CLBs surrounding the first CLB chosen for the swap and five CLBs surrounding the second CLB chosen for the swap will be considered for the attempted swap.



5. 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.



EXAMPLE: In the ongoing example the second column of CLBs to be swapped would break a carry-chain (shown with green arrow).

6. 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.
7. 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.
8. The move is accepted or rejected based on the current simulated annealing temperature.
9. 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.
10. 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.

4. Methodology

4.1. Embedded Floating-Point Units (FPUs)

The embedded FPU implements a double-precision floating-point multiply-add operation, as described by Equation 7. The FPU can be configured to implement a double-precision multiply, add, or multiply-add operation.

$$X_{64-bit} = \left(A_{64-bit} * B_{64-bit}\right) + C_{64-bit} \tag{7}$$

The baseline size of the FPU was conservatively estimated from commodity microprocessors and embedded cores in previous work [19], [20]. In addition to the size of the FPU the area necessary to connect to the general purpose routing structure was taken into account. This area for this connection to the general purpose routing structure is the addition of one vertical side of the FPU being filled with connection blocks. Since the exact size of a connection block is unknown, the conservative estimate of a connection block being the same size as a "CLB" was used. This made the true area of the FPU dependent on the shape chosen, resulting in a conservative area estimate.

The latency of the FPUs was more difficult to estimate experimentally. Given that a processor on a similar process (the Pentium 4) can achieve 3 GHz operation with a 4 cycle add and a 6 cycle multiply, it is assumed 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.

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 as shown in Figure 6. With an increase in the height of the FPU (decrease in the aspect ratio), there will be fewer FPUs on a single column. To 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 of the five benchmarks, matrix multiply, matrix vector multiply, vector dot product, FFT, and a LU Decomposition, were 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 clock rate, and number of routing tracks as given in Figure 9 through Figure 11.

There is a significant difference in the maximum clock rate between the benchmarks with different FPU aspect ratios. The benchmarks with FPUs of height 32 had the highest average clock rate as shown in Figure 9. The lower frequencies were found at the extremes, those with very large and very small aspect ratios. The benchmarks with large aspect ratios, small FPU heights, were very wide and consequently had large horizontal routes that increased the overall circuit latency. The benchmarks with small aspect ratios, large FPU heights, used a larger percentage of the routing tracks near the FPUs. This would dominate the minimum number of routing tracks needed and slower clock frequencies.

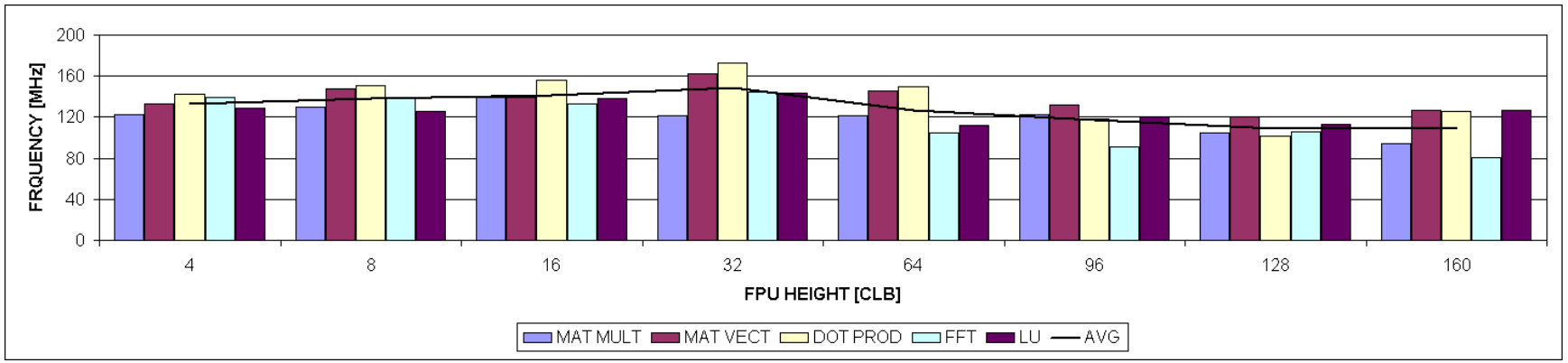


Figure 9. Embedded FPU benchmark clock rate

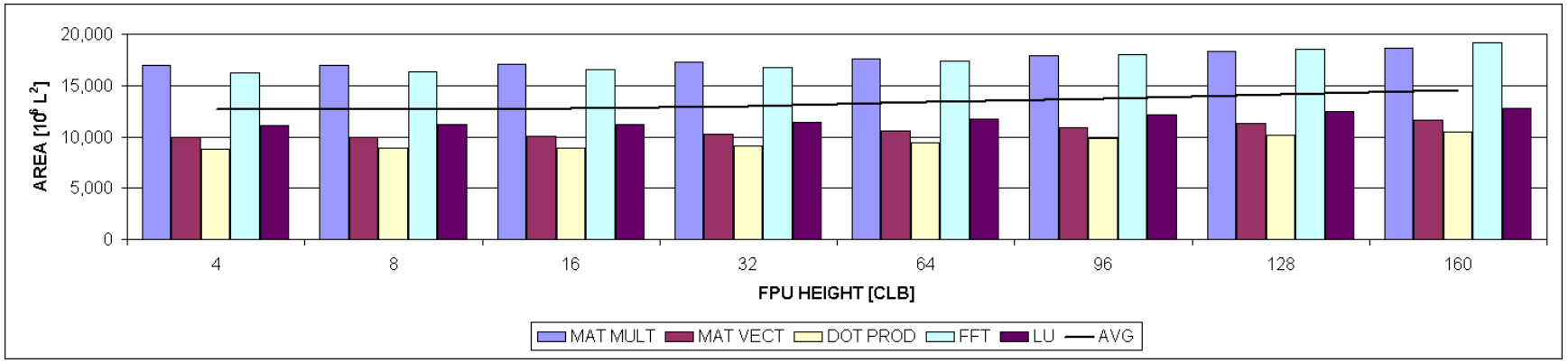


Figure 10. Embedded FPU benchmark area

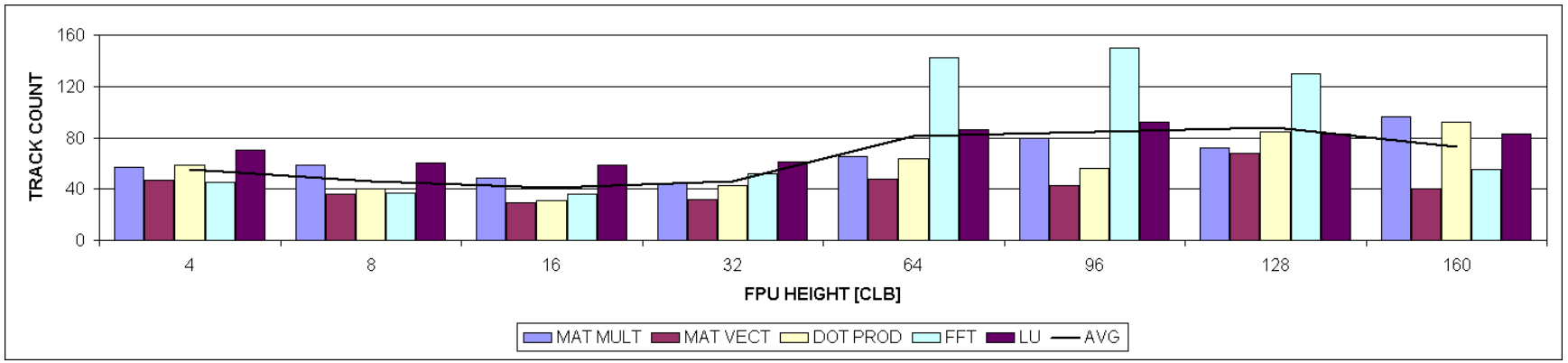


Figure 11. Embedded FPU benchmark track count

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 as shown in Figure 10.

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 as shown in Figure 11.

On average, the benchmarks that used the FPU of height 32 had the highest clock rate, 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.

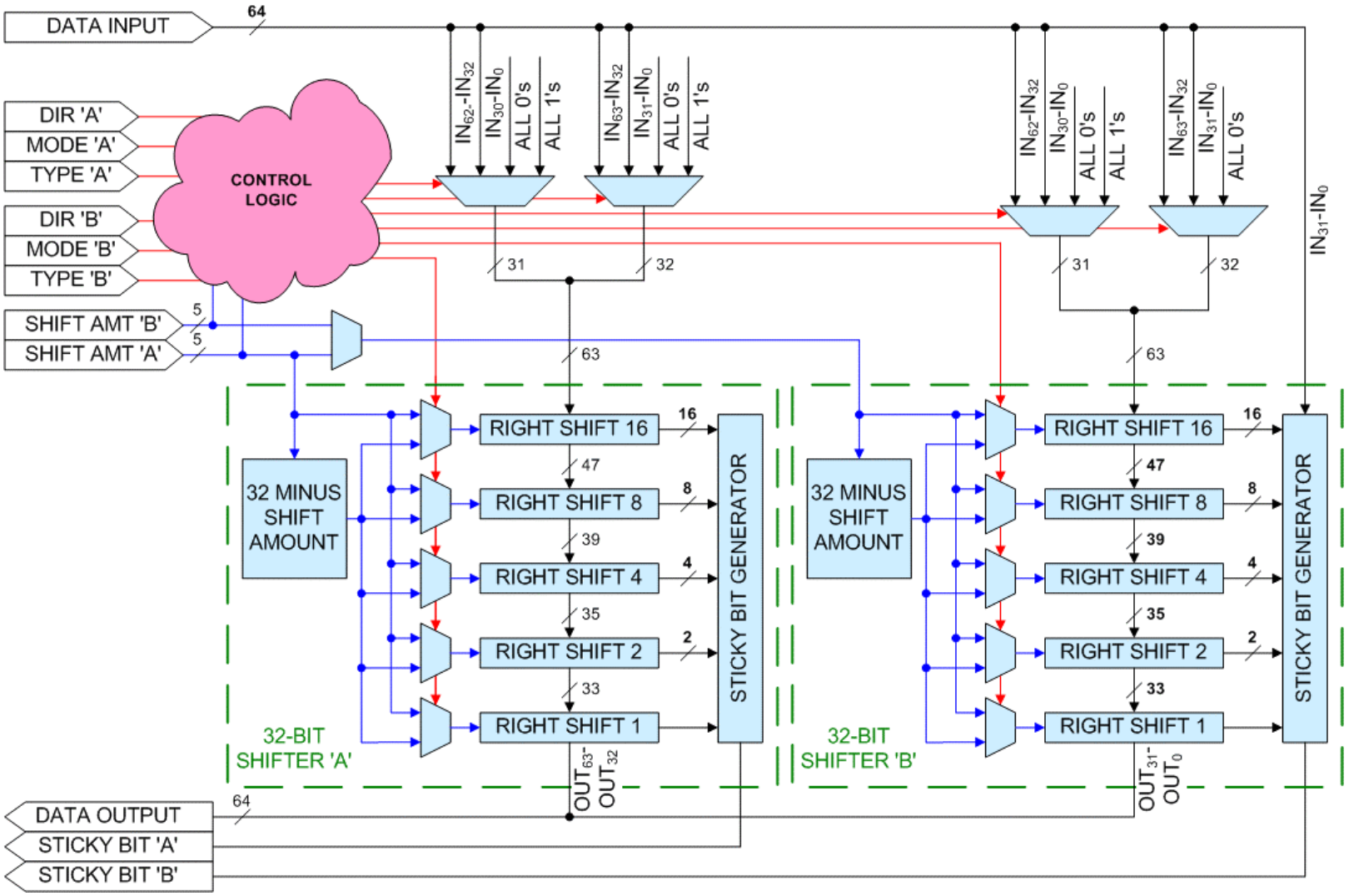


Figure 12. Embedded shifter block diagram

Table 5. Embedded shifter modes and control signals

Shift Configuration		Input				Control				
		Shift Amount (SA)	Direction	Mode	Type	Shifter 'A'		Shifter 'B'		Shift Muxs
						Top Mux	Bottom Mux	Top Mux	Bottom Mux	
SRL	32x2	–	RIGHT	SHIFT	LOGICAL	0	IN_{63}-IN_{32}	0	IN_{31}-IN_{0}	SHIFT AMT
SRA	32x2	–	RIGHT	SHIFT	ARITHMETIC	SIGN	IN_{63}-IN_{32}	SIGN	IN_{31}-IN_{0}	SHIFT AMT
SLL/SLA	32x2	–	LEFT	SHIFT	–	IN_{62}-IN_{32}	0	IN_{30}-IN_{0}	0	32-SA
ROR	32x2	–	RIGHT	ROTATE	–	IN_{62}-IN_{32}	IN_{63}-IN_{32}	IN_{30}-IN_{0}	IN_{31}-IN_{0}	SHIFT AMT
ROL	32x2	–	LEFT	ROTATE	–	IN_{62}-IN_{32}	IN_{63}-IN_{32}	IN_{30}-IN_{0}	IN_{31}-IN_{0}	32-SA
SRL	64x1	>32	RIGHT	SHIFT	LOGICAL	0	0	0	IN_{63}-IN_{32}	SHIFT AMT
SRL	64x1	<32	RIGHT	SHIFT	LOGICAL	0	IN_{63}-IN_{32}	IN_{62}-IN_{32}	IN_{31}-IN_{0}	SHIFT AMT
SRL	64x1	=32	RIGHT	SHIFT	LOGICAL	–	0	–	IN_{63}-IN_{32}	SHIFT AMT
SRA	64x1	>32	RIGHT	SHIFT	ARITHMETIC	SIGN	SIGN	SIGN	IN_{63}-IN_{32}	SHIFT AMT
SRA	64x1	<32	RIGHT	SHIFT	ARITHMETIC	SIGN	IN_{63}-IN_{32}	IN_{62}-IN_{32}	IN_{31}-IN_{0}	SHIFT AMT
SRA	64x1	=32	RIGHT	SHIFT	ARITHMETIC	–	SIGN	–	IN_{63}-IN_{32}	SHIFT AMT
SLL/SLA	64x1	>32	LEFT	SHIFT	–	IN_{30}-IN_{0}	0	0	0	32-SA
SLL/SLA	64x1	<32	LEFT	SHIFT	–	IN_{62}-IN_{32}	IN_{31}-IN_{0}	IN_{30}-IN_{0}	0	32-SA
SLL/SLA	64x1	=32	LEFT	SHIFT	–	–	IN_{31}-IN_{0}	–	0	32-SA
SLL/SLA	64x1	=0	LEFT	SHIFT	–	–	IN_{63}-IN_{32}	–	IN_{31}-IN_{0}	32-SA
ROR	64x1	>32	RIGHT	ROTATE	–	IN_{62}-IN_{32}	IN_{31}-IN_{0}	IN_{30}-IN_{0}	IN_{63}-IN_{32}	SHIFT AMT
ROR	64x1	<32	RIGHT	ROTATE	–	IN_{30}-IN_{0}	IN_{63}-IN_{32}	IN_{62}-IN_{32}	IN_{31}-IN_{0}	SHIFT AMT
ROR	64x1	=32	RIGHT	ROTATE	–	–	IN_{31}-IN_{0}	–	IN_{63}-IN_{32}	SHIFT AMT
ROL	64x1	>32	LEFT	ROTATE	–	IN_{30}-IN_{0}	IN_{63}-IN_{32}	IN_{62}-IN_{32}	IN_{31}-IN_{0}	32-SA
ROL	64x1	<32	LEFT	ROTATE	–	IN_{62}-IN_{32}	IN_{31}-IN_{0}	IN_{30}-IN_{0}	IN_{63}-IN_{32}	32-SA
ROL	64x1	=32	LEFT	ROTATE	–	–	IN_{31}-IN_{0}	–	IN_{63}-IN_{32}	32-SA
ROL	64x1	=0	LEFT	ROTATE	–	–	IN_{63}-IN_{32}	–	IN_{31}-IN_{0}	32-SA

4.2. Embedded Shifter

In floating-point addition, the mantissas must be aligned by shifting one of the mantissas to match the other. This requires a variable length and direction shifter. Because maximum range of the exponent is larger than the length of the mantissa, 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 754 single precision and up to 53 bits of shift can be required for IEEE 754 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 Figure 12. This unit can perform either one 64-bit shift (using the 'A' shift controls shown in Figure 12) or two independent 32-bit shifts (using both the 'A' and 'B' shift controls shown in Figure 12).

Even though floating-point operations only require a logical shift, the embedded shifter should be versatile enough to be used for other circuits that might make use of a different type of shifter. If the shifter is not versatile, implementing all shift modes, the embedded shifters will have a higher probability of being wasted in other types of circuits that require the modes that were not implemented. Therefore, the embedded shifter that was used has five modes: rotate left (ROL), rotate right (ROR), shift right logical (SRL), shift right arithmetic (SRA), and shift left logical/arithmetic (SLL/SLA).

During the shifting that accompanies the normalization of floating-point numbers it is important to calculate the sticky bit as it is an integral part of the shift operation. The sticky bit is used when implementing round-to-nearest-even and is the result of the logical OR of all the bits that are lost during a right shift. 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.

To help improve the maximum clock speed, 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 modes of operation and control of the embedded shifter is given in Table 5. A 64-bit left and right shifter can be created out of a 127-bit right only shifter. This shifter would have six levels of muxes (shift by 1, shift by 2, shift by 4, … , shift by 32). In addition to the muxes needed for shifting, a mux is needed on the input to select the correctly orientate the input, force a zero for logical shifts, or force a zero or one for sign extension. However, the top-level mux can be combined with the shift by 32 mux, minimizing the complexity of the shifter.

In order to use the 64-bit shifter as two independent 32-bit shifters, a few modifications were needed. The bottom five levels of muxes need to be duplicated and the control for the top level was modified to account for both 32-bit and 64-bit shift modes. The control for the embedded shifter is shown in Table 5. The control was determined such that in both the 32-bit and 64-bit shift modes the output is in correct sequence and another level muxes are not needed.

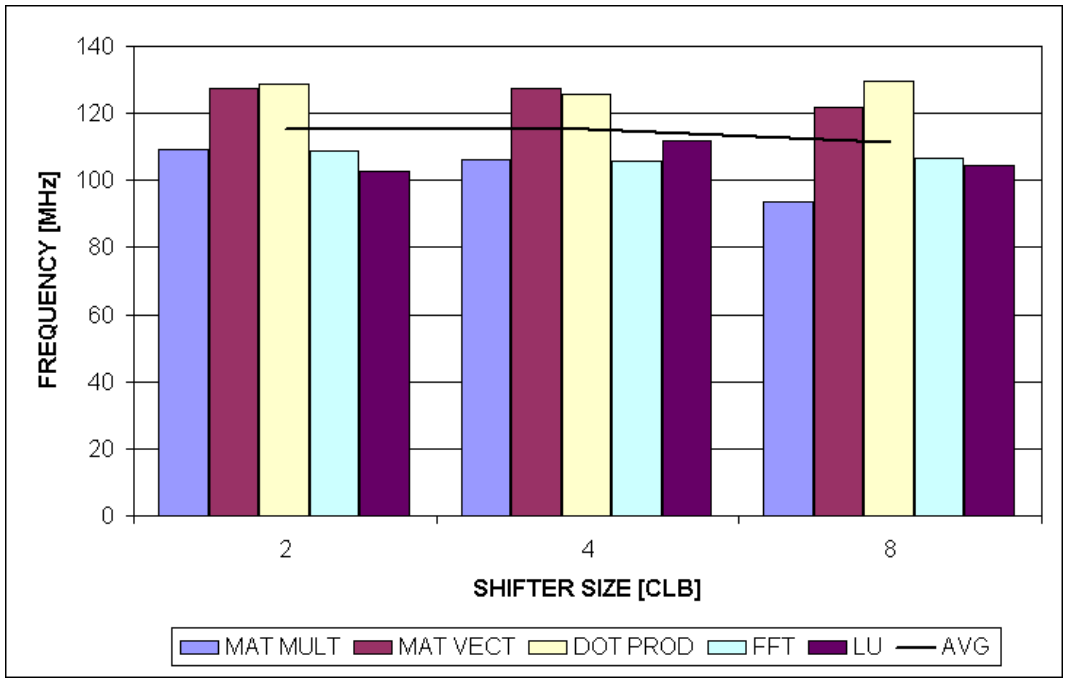


Figure 13. Embedded shifter benchmark clock rate

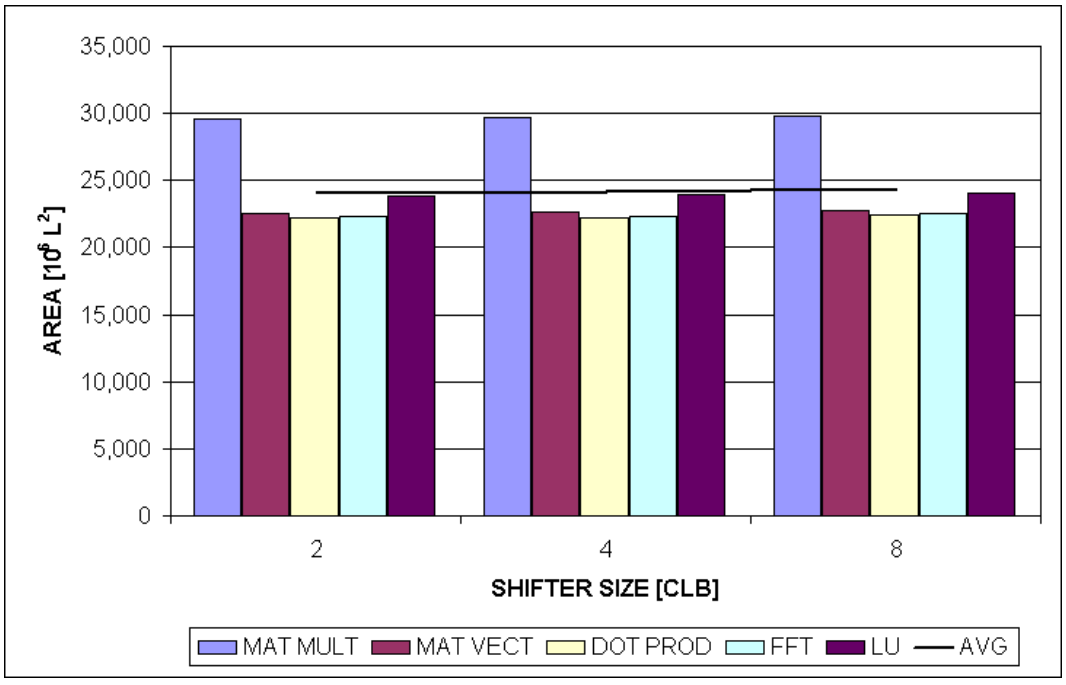


Figure 14. Embedded shifter benchmark area

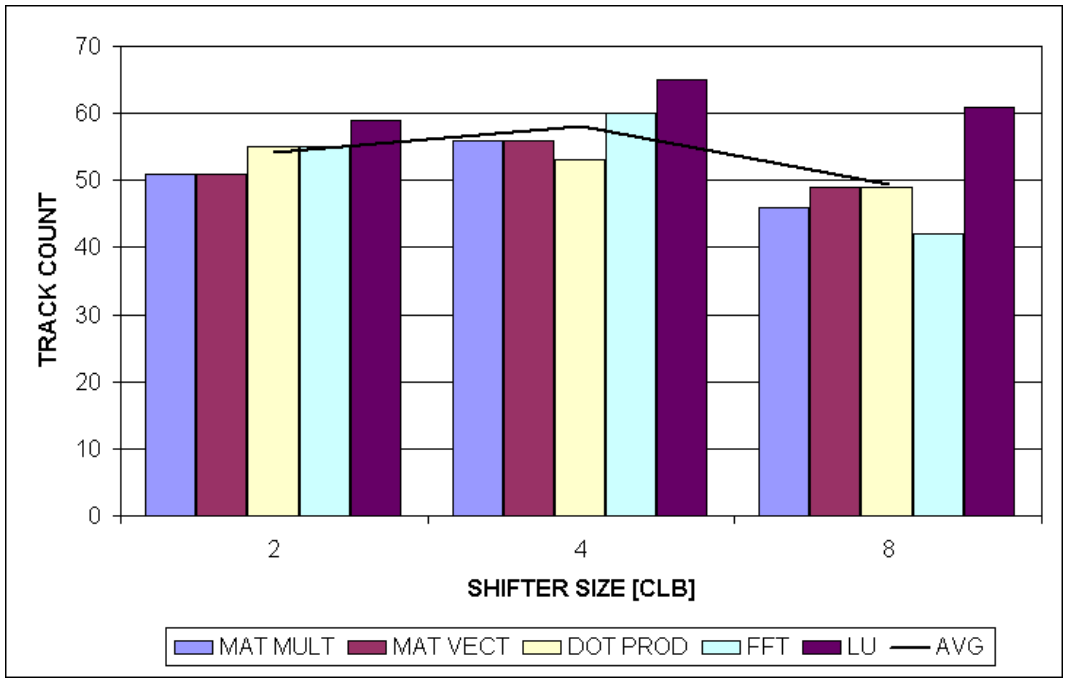


Figure 15. Embedded shifter benchmark track count

The benchmark circuits used to test the feasibility of embedded shifter in FPGAs, matrix multiply, matrix vector multiply, vector dot product, FFT, and LU decomposition, 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. While this is not needed for simulation, it is necessary to ensure that it is not 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 of the embedded shifter is 1.27 times the area of the CLB and its associated routing. The area of the shifter does not take into account the area needed for the additional number of connections to the general purpose routing structure of the embedded shifter compared to the CLB. Because this area overhead is difficult to estimate, three different embedded shifter sizes (two, four, and eight equivalent CLBs) were examined and their results are given in Figure 13 through Figure 15.

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, this can be used for an extremely conservative estimate of the shifter size. Therefore, an embedded shifter size equivalent to four CLBs is used to compare with the baseline architecture.

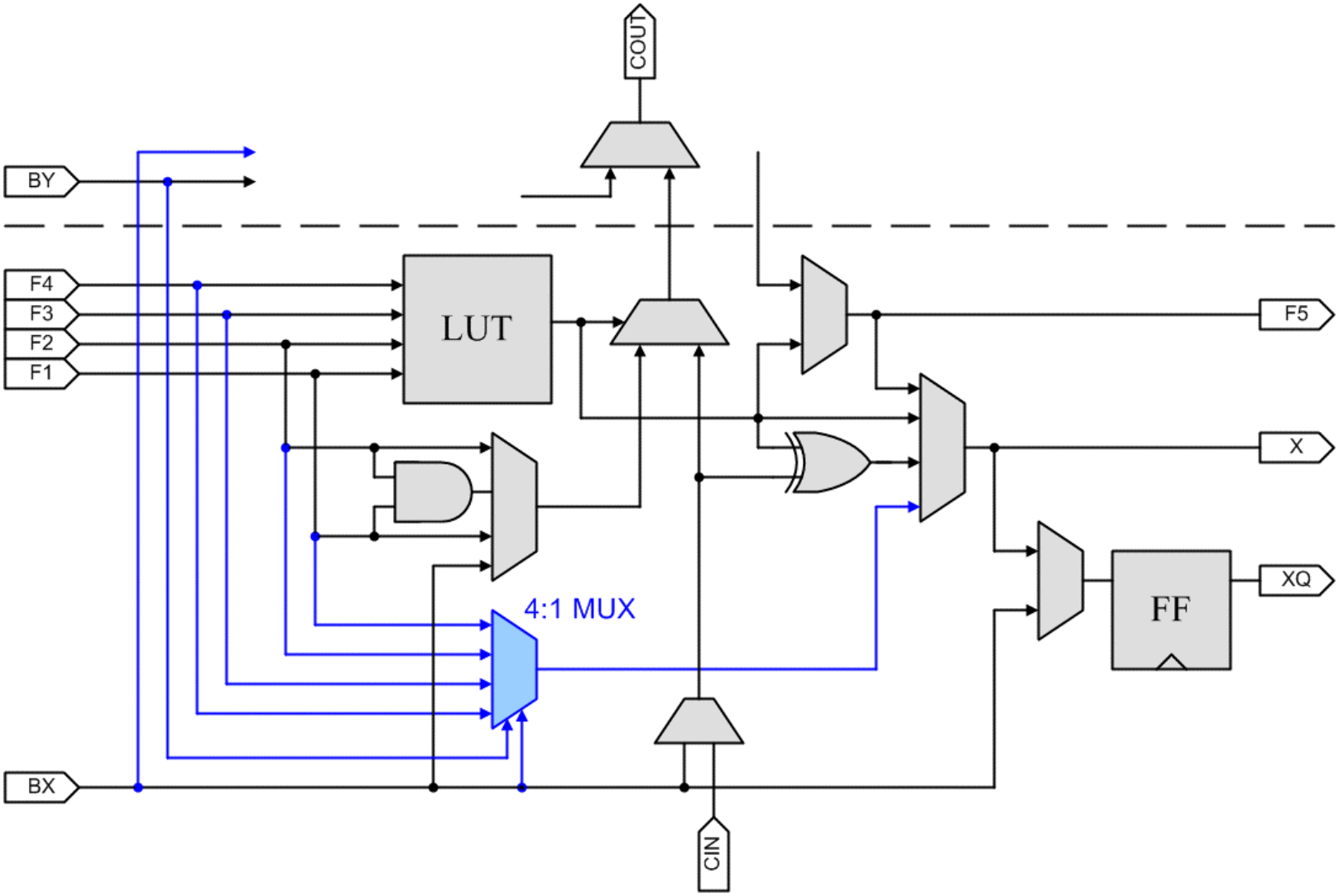


Figure 16. Simplified representation of bottom half of modified CLB showing addition of 4:1 multiplexer

4.3. Modified CLB with additional 4:1 Multiplexer

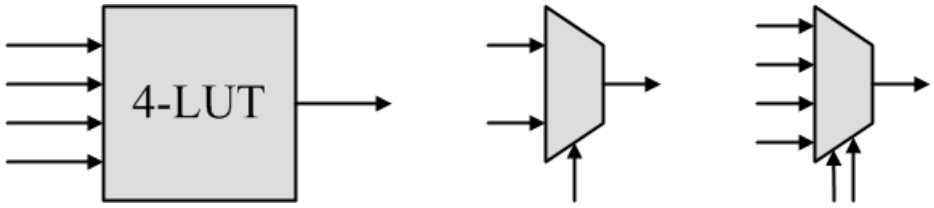


Figure 17. (a) 4-input LUT (b) 2:1 mux (c) 4:1 mux

The Xilinx CLB consists of a combination of 4-LUTs. A 4-LUT can implement any 4-input logic function. Since binary shifters consist of a series of muxes in powers of two, a 2:1 mux can be used. A 2:1 mux has three inputs as seen in Figure 17. Implementing a 2:1 in a 4-LUT results in an unused input. Two 4-LUTs can be combined to form a 5-LUT, which is still not large enough to implement a 4:1 mux, which can also be used in binary shifters. The 4:1 mux has six inputs as seen in Figure 17 and would require a 6-LUT.

In modifying the CLB to better implement variable length shifters, two general ideas 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 Figure 16. 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 2).

4.4. 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 which were created by K. Scott Hemmert and Keith D. Underwood at Sandia National Laboratories. 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

* LU Decompostion is the decomposition of an N x N matrix, A, into a product of a lower triangular matrix L and an upper triangular matrix U, such that $LU=A$.

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.

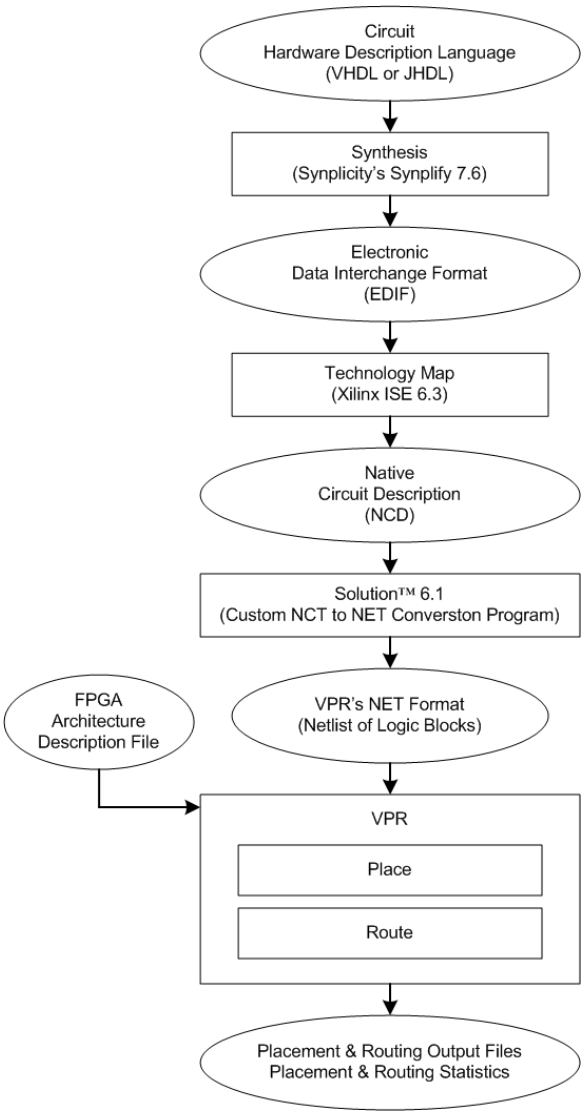


Figure 18. CAD flow

The floating-point benchmarks were written in a hardware description language, VHDL or JHDL [21]. LU Decomposition was written in JHDL. Matrix multiply, matrix vector multiply, vector dot product, and FFT were written in VHDL. Instead of using the traditional VPR path of synthesis using SIS [22] and technology mapping using Flow Map [23], 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). 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. This flow is shown in Figure 18

The benchmarks vary in size and complexity as shown in Table 6. 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.

Table 6. 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 embedded multipliers as the EMBEDDED MULTIPLIER version.

† Benchmarks include the same number of IOs and block RAMs as CLB ONLY version.

5. Results

Three FPGA architectural modifications were proposed: adding embedded FPUs, embedded shifters, and adding a 4:1 mux in parallel with the 4-LUT in the CLB. Each of these architectural modifications and the baseline architecture, which represents what is currently available in FPGAs, were implemented in five benchmarks: matrix multiply, matrix vector multiply, vector dot product, FFT, and LU Decomposition. Each of the architectures was placed and routed using VPR. The maximum clock rate (or circuit frequency), circuit area, and minimum track count was noted.

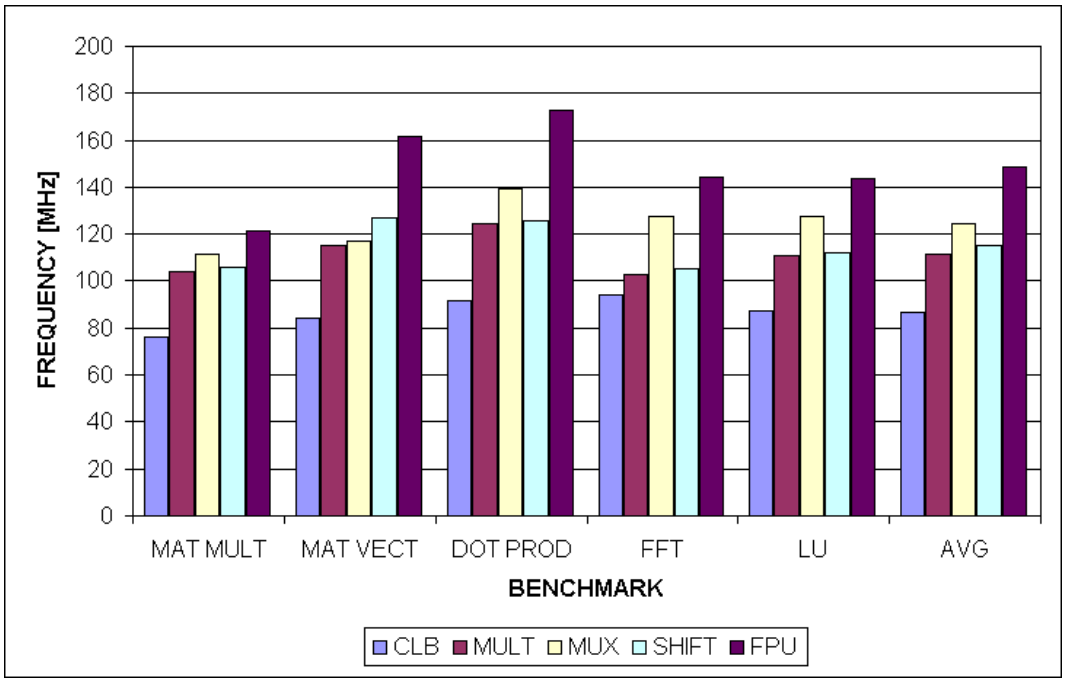


Figure 19. Benchmark clock rate

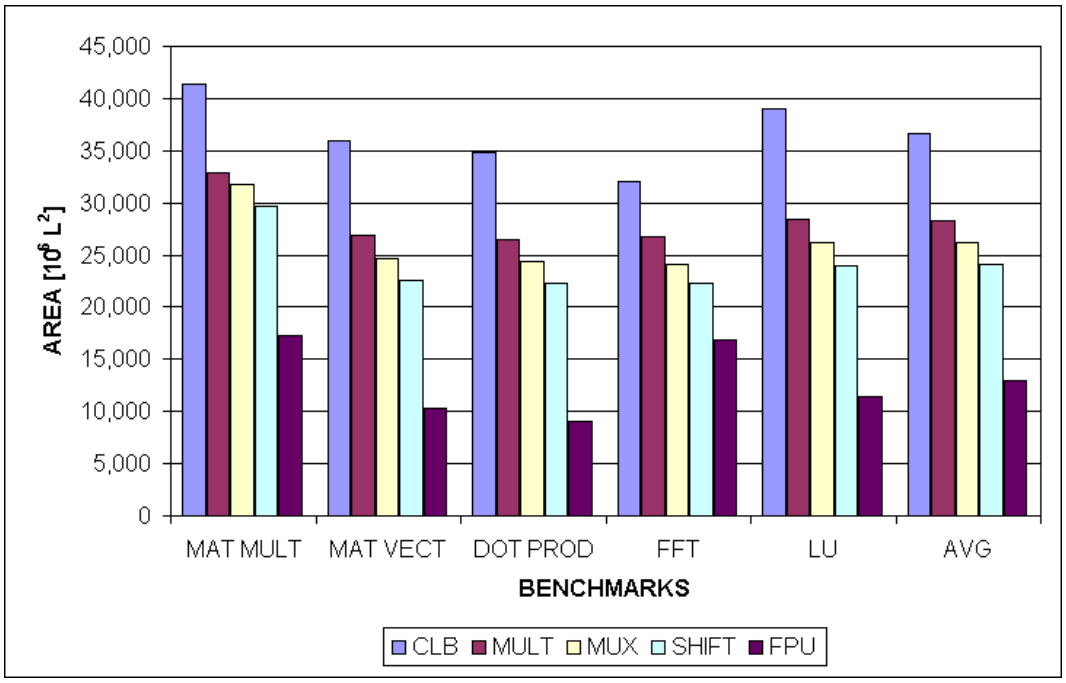


Figure 20. Benchmark area

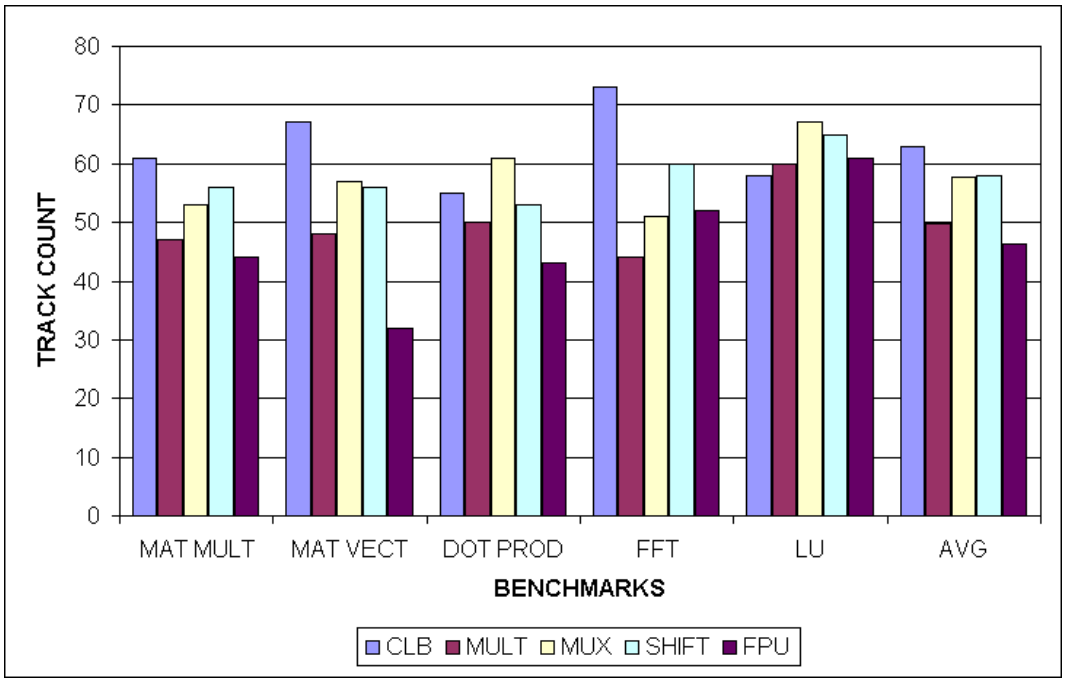


Figure 21. Benchmark track count

5.1. Embedded FPUs

The embedded FPU had the highest clock rate, smallest area, and lowest track count of all the architectures, as seen in Figure 19 through Figure 21. 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).

5.2. Embedded Shifters

Even with a conservative size estimate, adding embedded shifters to modern FPGAs significantly reduced circuit size. As seen in Figure 19 through Figure 21, 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. Only considering 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.

5.3. Modified CLBs with additional 4:1 Multiplexers

Using the small modification to the CLB architecture showed small circuit area and increased clock rates. 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.

5.4. 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.

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 [24] 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, circuits that use floating-point numbering system might naturally lend 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 [25]. 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 [26]. While the details of the Virtex-5 are not known, 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.

7. Conclusion

This thesis has demonstrated three architectural modifications that make floating-point operations more efficient on FPGAs. Adding complete double-precision floating-point multiply-add units, adding embedded shifters, and adding a 4:1 multiplexer in parallel to the 4-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% compared to the baseline architecture, which used embedded multipliers and CLBs to implement floating-point operations. 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% compared to the baseline architecture, which used embedded multipliers and CLBs to implement floating-point operations. 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 unexpected that the smaller change to the FPGA architecture amounts to the bigger net “win”.

There is a compromise that must be met between an FPGA that has fined grained generic logic and thus is extremely versatile, but inefficient and an application specific FPGA which has specialized coarse grained logic blocks to facilitate efficient operations. There might be a small continuous demand for generic FPGAs for research and prototyping, but

in order for FPGAs to obtain a greater use in industry they must become more efficient for scientific applications, and therefore floating-point operations. It is conceivable that this need will not make more generic FPGAs obsolete, but rather spawn other FPGA families.

END NOTES

[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] Larry McMurchie and Carl Ebeling, PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, February 1995.

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

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

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

BIBLIOGRAPHY

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

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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

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

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.

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.

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

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.

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

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

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

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

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.

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.

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.