Embedded Floating-Point Units in FPGAs

Michael J. Beauchamp Scott Hauck
University of Washington
Dept. of Electrical Engineering
{mjb7, hauck}@ee.washington.edu

Keith D. Underwood K. Scott Hemmert
*Sandia National Laboratories**
Scalable Computing Systems
{kdunder, kshemme}@sandia.gov

Abstract

Due to their generic and highly programmable nature, FPGAs provide the ability to implement a wide range of applications. However, it is this nonspecific nature that has limited the use of FPGAs in scientific applications that require floating-point arithmetic. Even simple floating-point operations consume a large amount of computational resources. In this paper, we introduce embedding floating-point multiply-add units in an island style FPGA. This has shown to have an average area savings of 55.0% and an average increase of 40.7% in clock rate over existing architectures.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design Studies
C.1.3 [**Other Architecture Styles**]: Adaptable Architectures – *Field-Programmable Gate Arrays*

General Terms

Design, Performance

Keywords

FPGA, FPGA Architecture, Floating-Point, FPU

1. Introduction

In recent years there has been a significant increase in the size of FPGAs. Architectures contain tens of thousands to hundreds of thousands of logic elements, providing a logic capacity into the millions of gates. With larger FPGAs comes the versatility and opportunity for larger and more complex circuits. This increase in size has allowed FPGAs to be considered for several scientific applications that require floating-point arithmetic [1, 2, 3, 4, 5, 6].

In the past, FPGAs have excelled in fixed-point computations, but were unable to effectively perform floating-point computations due to their limited size. Even though it is currently possible, it is not always practical to implement these floating-point computations on FPGAs as they can consume large amounts of chip resources. ASICs, which can be highly efficient at floating-point computations, can be prohibitively expensive and 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 [7].

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. Currently, floating-point operations are becoming more common. There are only a few floating-point operations of interest, and because of the acceptance of the IEEE 754 Standard for Binary Floating-Point Arithmetic [8], these operations can be included as embedded units in FPGAs. The idea of embedding coarse gained units in FPGAs is not a unique concept. Currently there are FPGAs that have embedded multipliers, block RAMs, and even full microprocessors. By also embedding floating-point units, the huge timing and area costs of implementing these computations in flexible resources are eliminated.

To test this concept, we have augmented VPR to support embedded functional units, as well as high-performance carry-chains. VPR 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 test the practicality of using embedded double-precision floating-point multiply-add units (FPUs) in FPGAs, each benchmark was tested using three versions. The first version uses only CLBs to implement the floating-point calculations, the second version uses a combination of CLBs and the embedded 18-bit x 18-bit embedded multipliers to perform the

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

FPGA'06, February 22-24, 2006, Monterey, California, USA.

Table 1. IEEE single and double precision floating-point computation logic resources for Xilinx Virtex II Pro 2 and 100

Computation	Single Precision			Double Precision		
	Slices	Logic Resources Used		Slices	Logic Resources Used	
		XC2VP100	XC2VP2		XC2VP100	XC2VP2
Add	328	0.7%	23.3%	799	1.8%	56.7%
Multiply	345	0.8%	24.5%	1177	2.7%	83.6%
Multiply Accumulate	693	1.6%	49.2%	2012	4.6%	142.9%

floating-point calculations, and the third version uses the embedded FPUs to perform the floating-point calculations. The embedded FPU version had an average area reduction of 55.0% compared to the version with the embedded 18-bit x 18-bit multipliers and an average area reduction of 63.6% compared to the version that used only CLBs. The embedded FPU version had an average speed increase of 40.7% over using the embedded 18-bit x 18-bit multipliers and a 85.1% speed increase over using only CLBs. These improvements were obtained while achieving an average reduction in the number of routing tracks by 8.6%.

The remainder of this paper is organized as follows. Section 2 presents background information on the floating-point numbering system and the island-style FPGA architecture. The details of how VPR was modified and used to test the feasibility of placing embedded floating-point units in FPGAs are presented in Section 3. Section 4 presents the testing methodology. The testing results and analysis are presented in Section 5. Finally, Section 6 presents conclusions and Section 7 presents future work.

2. Background

Floating Point Numbering System

Floating-point numbers consist of a mantissa, exponent, and sign bit. They are combined to form single-precision (equation 1) and double-precision (equation 2) floating-point numbers [8, 9].

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

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

The IEEE standard for single precision floating-point numbers is 1 sign bit, 8-bit exponent, and 23-bit mantissa as shown in Figure 1. Double precision is similar with 1 sign bit, 11-bit exponent, and 52-bit mantissa as shown in Figure 2.

The advantage of floating-point numbers over fixed-point is the range of numbers that can be represented with the same number of bits. In addition, the IEEE standard for floating-point numbers has defined representations for zero, negative and positive infinity, and not a number, NaN. This increased range and special cases results in more complex computations and requires additional hardware compared to fixed-point calculations.

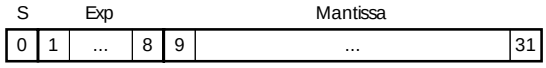

Figure 1. Single precision IEEE floating-point number

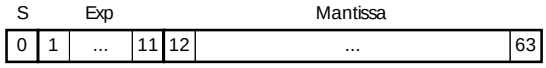

Figure 2. Double precision IEEE floating-point number

Island-Style FPGA

While the typical island-style FPGA consists only of IO and logic blocks, it is becoming more common to have various types of embedded units. Current embedded units include block RAMs, multipliers, DSP blocks, and even complete microprocessors [10, 11]. These advances have allowed for greater usability of circuits that use fixed-point numbers. However, circuits that use floating-point numbers still require a large number of CLBs (Configurable Logic Blocks) to perform basic operations. Thus, this paper examines the impact of adding floating-point units to the list of embedded blocks.

While floating-point calculations encompass a wide variety of operations including addition, subtraction, multiplication, division, powers, and roots, not all of these operations are heavily used in typical applications. Including operations that are not widely used would exacerbate the primary drawback of this approach: wasted hardware. The most popular of the operations are multiplication and addition, and they are often used in combination. Therefore, a floating-point unit that consists of a multiply-add will be used to test the feasibility of embedding floating-point units in FPGAs. Other, less frequent operations can be built by using multiple floating-point multiply-add operations to implement algorithms such as Taylor series expansion or Newton-Raphson iterations. Alternatively, other operations can be implemented in the general reconfigurable fabric.

3. VPR

VPR [12, 13] 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 embedded floating-point units, FPUs, in island-style FPGAs.

Previous versions of VPR supported three types of circuit elements, input pads, output pads, and CLBs. To test the feasibility of embedded floating-point units and to incorporate necessary architectural elements, VPR was modified to allow the use of embedded block units of parameterizable size. Additionally, a fast carry-chain was incorporated into the existing CLBs elements.

An FPGA architecture was used that approximately models the Xilinx® Virtex-II Pro FPGA family. Our model architecture consisted of IO blocks, CLBs (which included 4-input function generators, storage elements, arithmetic logic gates, and a fast carry-chain), 18Kb block RAMs, and embedded 18-bit x 18-bit multiplier blocks [10]. In addition to the Xilinx® Virtex-II Pro features, the architecture incorporated embedded floating-point units. The various blocks were arranged in a column based architecture similar to Xilinx's®, ASMBL™ (Advanced Silicon Modular Block) architecture [14], as seen in Figure 3, which is the architectural foundation of the Virtex-4 FPGA family [15].

To directly incorporate embedded blocks with the existing circuit elements of VPR, the new embedded block size must be quantized with the size of the existing circuit elements. Each embedded unit type has two size parameters, height and width, each in terms of CLB size. In keeping with the column based architecture, horizontal routing was allowed across the embedded units, but vertical routing was kept only at the periphery of larger units. The regular routing structure that existed in the original VPR was maintained as shown in Figure 4.

The outputs of the embedded units can be registered. Each embedded unit type has three timing parameters: sequential setup time, sequential clock-to-q, and maximum combinational delay if no clock is used.

The fast carry-chain is a dedicated route that is separate from the rest of the routing structure. It connects the carry-out at the top of one CLB to the carry-in on the bottom of the CLB directly above. Since the carry-chain connection between CLBs is independent from the normal routing structure and does not go through connection boxes, switch boxes, or isolation buffers, it has its own timing parameter.

3.1. Component Area

The areas of the CLB, 18x18 bit multiplier, and 18 Kb RAM were approximated using a die photo of a Xilinx Virtex-II 1000 [16] courtesy of Chipworks Inc. The area estimate of each component includes the associated connection blocks, which dominate the routing area. The baseline size of the floating-point unit

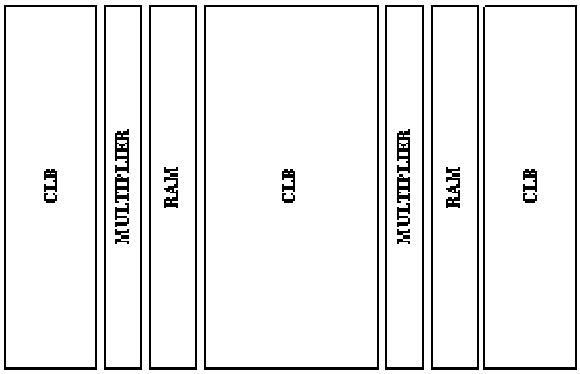

Figure 3. ASMBL

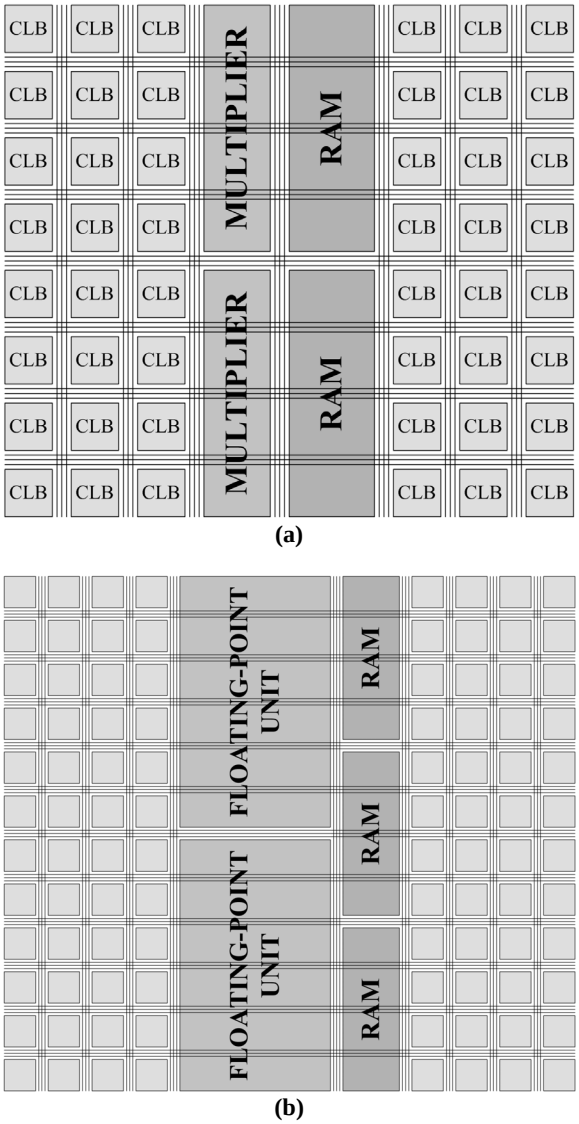

Figure 4. (a) Our column based architecture with CLBs, embedded multipliers, and block RAMs. (b) Our column based architecture with CLBs, embedded floating-point units, and block RAMs

was conservatively estimated from commodity microprocessors and embedded cores in previous work [17, 18] 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 areas were normalized by the process gate length. All areas are referenced to the smallest component, which is the CLB. These values are shown in Table 2.

Table 2. Embedded Component Timing & Area

	T_{SETUP} [ns]	$T_{CLK \rightarrow Q}$ [ns]	Area [10^6 L^2]	Area [CLBs]
CLB	0.32	0.38	0.662	1
Multiplier 18x18	2.06	2.92	11.8	18
RAM 18Kb	0.23	1.50	18.5	28
FPU 64-bit	0.50	0.50	107	161

3.2. 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. To accurately represent the timing of this in VPR's CLB architecture, nineteen VPR subblocks were used. For each subblock the sequential setup time, sequential clock-to-q, and maximum combinational delay if no clock is used were found in Xilinx data sheets or experimentally using Xilinx design tools.

The latency of the embedded multipliers and RAMs are based on the Xilinx® Virtex-II Pro -6. The latency of the FPUs was more difficult to estimate appropriately. Given a processor on a similar process (the Pentium® 4) can achieve 3 GHz operation with a 4 cycle add and a 6 cycle multiply, we assume that an FPU in 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.

3.3. Track Length & 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 3 [10].

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.

Table 3. Track Length

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

4. Methodology

4.1. Benchmarks

Five benchmarks were used to test the feasibility of embedding floating-point units in an island style FPGA. They were matrix multiply, matrix vector multiply, vector dot product, FFT, and LU decomposition. The LU decomposition benchmark is still preliminary, with a complete datapath, but a control path that is still under development. All of the benchmarks use double-precision floating-point addition and multiplication. Additionally, LU decomposition includes floating-point division, which must be implemented in the reconfigurable fabric for all architectures. Each of the benchmarks were placed and routed in three FPGA versions. In the first version, the floating-point calculations were performed using only the CLBs. The CLB only version is used as a point of reference to compare the other versions. In the second version, the floating-point calculations were performed using CLBs and embedded 18-bit x 18-bit multiplier blocks. This version is representative of existing FPGAs, specifically the Xilinx® Virtex-II Pro. The third version adds the embedded double precision floating-point multiply/add units in various aspect ratios.

Table 4. Number of components in CLB only benchmark

Benchmark	CLBs	I/O	RAM
Matrix Mult.	32,478	196	128
Vector Mult.	35,630	1,274	14
Dot Product	30,653	782	0
FFT	46,590	555	152
LU	33,534	194	64
Average	35,777	600	72

Table 5. Number of components in embedded multiplier and embedded FPU benchmarks

Benchmark	Embedded Mult		Embedded FPU	
	CLBs	MULT	CLBs	FPU
Matrix Mult.	25,290	72	9,373	8
Vector Mult.	27,836	90	8,012	10
Dot Product	21,301	72	4,926	8
FFT	37,130	72	15,432	28
LU	24,257	72	8,108	8
Average	27,163	76	9,170	12

The floating point benchmarks were written in a hardware description language, either VHDL or JHDL

[19], and synthesized into an EDIF (Electronic Data Interchange Format) file. 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; thus, the traditional VPR T-Vpack path for mapping is completely bypassed.

The benchmarks vary in size and complexity. Table 4 gives the number of components for the benchmarks that perform the floating-point operations using the CLBs only. The number of IO and 18Kb RAM blocks will remain constant for all three versions of the benchmarks. Table 5 gives the number of components for the benchmark versions that perform the floating-point calculations using 18 x 18 embedded multipliers and the number of components for the benchmark versions that perform the floating-point calculations using the embedded FPUs.

There are a few interesting trends and distinguishing aspects to notice. There is an average reduction by 24% in the number CLBs from the CLBs only benchmark versions to the embedded multiplier version, and a 66% reduction from the embedded multiplier version to the embedded FPU version. This is to be expected because the embedded multipliers replace some of the CLBs allocated for floating-point calculations in the CLB only version and the FPUs replace the embedded multipliers that were used for floating-point calculations in the embedded multiplier version. The reduction in the number of CLBs between the embedded multiplier benchmarks and the FPU benchmarks varies from 58% for the FFT to 77% for the dot product. This variation is due to the ratio of control to datapath logic and the number and type of floating-point calculations performed.

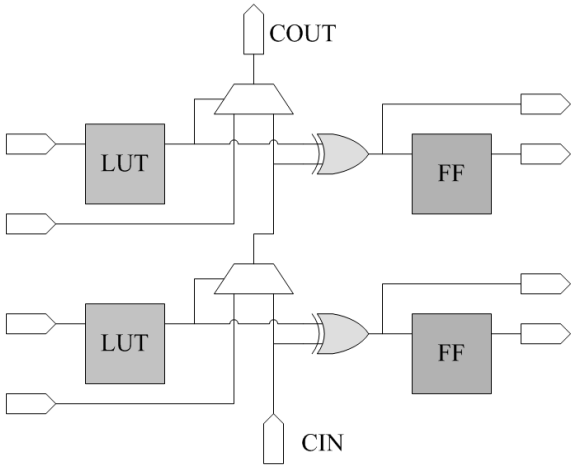

Figure 5. Simplified CLB with fast vertical carry-chain

The number of IOs differs by almost an order of magnitude between the different benchmarks from 194 for the LU decomposition to 1,274 for the matrix vector multiply. This is largely an artifact of how the benchmarks are extracted and has no significant impact on the results. The number of block RAMs also has a significant variation from the dot product which does not use any block RAMs to the FFT which uses 152.

4.2. 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 Figure 5. 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 as seen in the simplified CLB shown in Figure 5.

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. 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.

Table 6. Maximum frequency with a and without the use of the fast carry-chain for the embedded multiplier version

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

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 6. By using the fast carry-chain the benchmarks had an average speed increase of 49.7%.

Because the carry-chains only exist in columns of CLBs and only in the upward direction, it is necessary to initially place all of the CLBs of a given carry-chain in proper relative position to each other and to move/swap all of the CLBs that comprise a carry-chain as one unit. To accomplish this, the move/swap function of VPR was modified. 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. The success of simulated annealing depends on trying a large enough number of moves. Therefore, before a move can be considered as accepted or rejected, a legal move must be found.
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

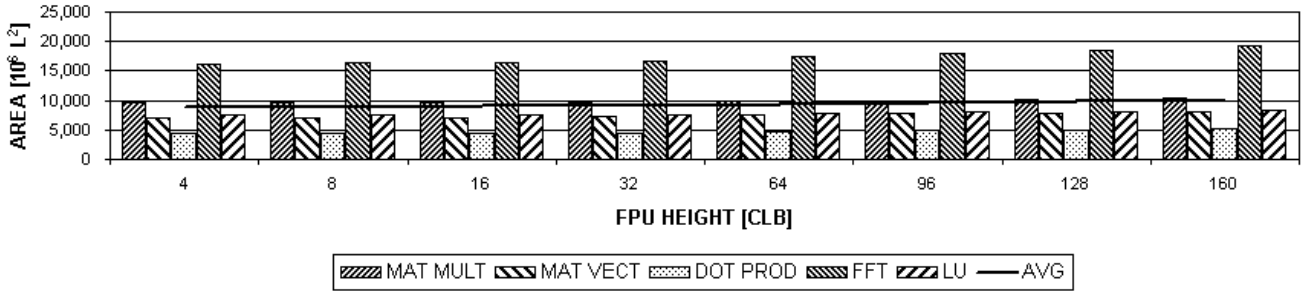

Figure 6. Floating-point unit benchmark area

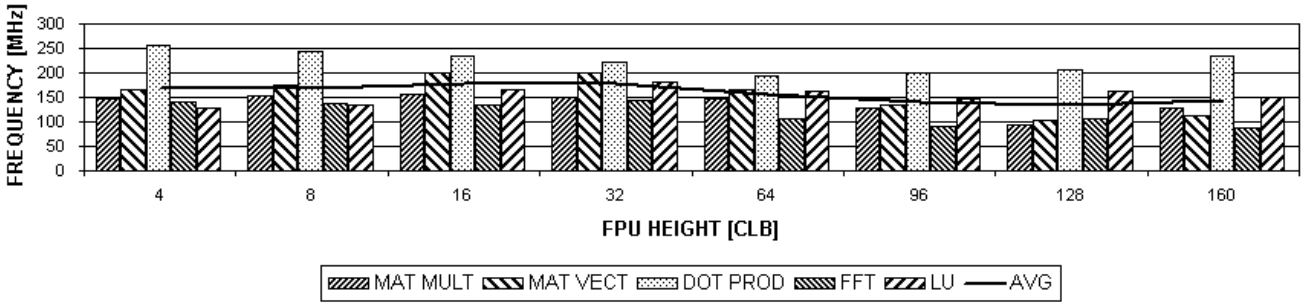

Figure 7. Floating-point unit benchmark maximum frequency

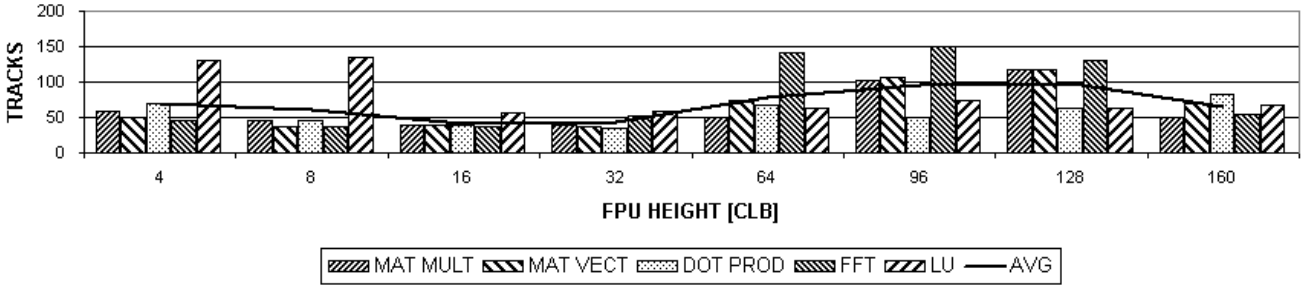

Figure 8. Floating-point unit benchmark channel width

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 was necessary to maintain the integrity and results of simulated annealing.

5. Testing & Analysis

5.1. Floating-Point Units

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 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 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. Table 7 shows the relative number of columns of CLBs, RAMs, and FPUs for each of the different FPU heights.

Table 7. Component column architecture ratios for various FPU sizes

FPU Height	CLB Col.	RAM Col.	FPU Col.
4	46	2	1
8	46	2	1
16	45	2	1
32	40	2	2
64	40	2	4
96	40	2	6
128	40	2	8
160	40	2	10

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.

Each benchmark was tested with eight different FPU heights. These benchmarks with different FPU sizes were compared on three criteria: area, maximum frequency, and number of routing tracks. These results are given in Figure 6 through Figure 8.

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 as seen in Figure 6. However, the average difference in area is only 3.3%.

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 (see Figure 8). The Virtex II Pro, for example, has enough routing tracks to accommodate most of the benchmarks in most of the configurations. Some configurations (16, 32, 160) are completely routing neutral. These are likely to be better choices if all else is equal.

The last consideration, maximum frequency, is the most significant aspect. There is a significant difference in the maximum frequency between the benchmarks with different aspect ratio FPUs. As seen in Figure 7, the benchmarks with FPUs of height 32 had the highest average frequency. 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. Also, as seen in Table 7, with the larger FPU height there was a greater number of FPU columns to keep the overall number of FPUs constant, this causes congestion due to the large number of nets associated with each FPU.

5.2. Results

Eight different FPU aspect ratios were compared to determine which size gave the best results. The different sizes ranged from a FPU with an equivalent CLB height of 4, up to a FPU with an equivalent CLB height of 160. On average, the benchmarks that used the FPU of height 32 had the highest frequency, fewest number of routing tracks, and did not have a significant area increase over those with other aspect ratios. Therefore, it is the architectures with FPUs of height 32 that are being compared to the architectures with the embedded multipliers and with CLBs only. They were compared by area, maximum frequency, and track count.

Area

As seen in Figure 9, the FPU had an average reduction in area of 55.0% compared to the embedded multiplier version and an average reduction in area of 63.6% compared to the CLB only version. While all of the benchmarks with the embedded FPUs had an area reduction, there was some variation in the amount. The dot product had the largest area reduction of 70.2% compared to the embedded multiplier and 77.9% compared to the CLB only version. While still significant, the FFT had the smallest area reduction of 41.0% compared to the embedded multiplier version and

50.4% compared to the CLB only version. This variation in area reduction is due to a few different factors. First, benchmarks like the FFT had a larger percentage of control logic compared to the other benchmarks. In addition, the FPU version of the matrix multiply had 24.5% of its area occupied by block RAMs. The area due to control logic and the block RAMs are constant between the different versions and

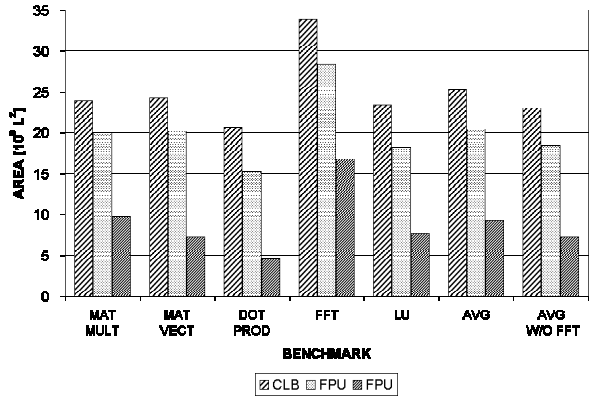

Figure 9. Area of circuits with CLBs only, embedded multipliers, and FPUs

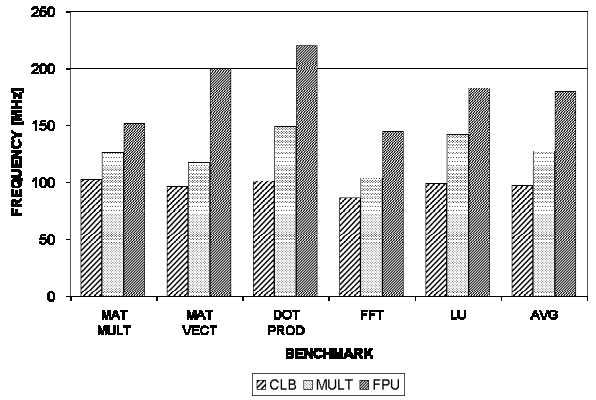

Figure 10. Maximum frequency of circuits with CLBs only, embedded multipliers, and FPUs

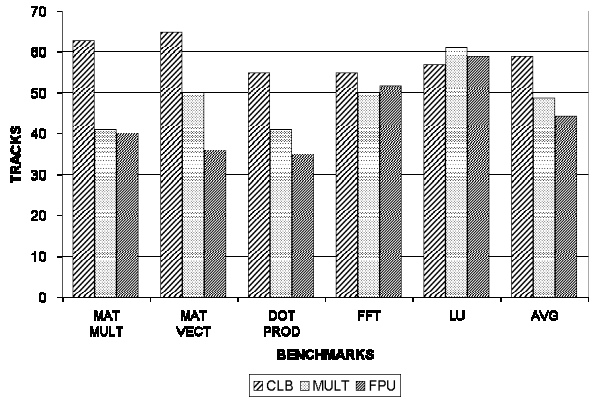

Figure 11. Track count of circuits with CLBs only, embedded multipliers, and FPUs

are not affected by using the embedded FPUs.

The third reason for the variation in area reduction pertains solely to the FFT. Even though the FFT uses floating-point multiplication and addition they are not used as a composite multiply-add operation as in the other benchmarks. Therefore, the multiply-add FPUs are being used as either a floating-point multiply or a floating-point add, but not both. This results in both a large number of FPUs being needed as shown in Table 5 and significant portions of the FPUs being unused. This is not an issue in the embedded multiplier and CLB only versions, as a floating-point multiplier or adder can be developed out of the embedded multipliers or CLBs as needed. With some optimization of the benchmark, the amount of single operation use could be reduced. Alternatively, by enabling the adders and multipliers to be independent, this effect could be greatly reduced. If you remove the FFT from the average, the FPU version had a reduction in area of 60.5% compared with the embedded multiplier version and an average reduction in area of 68.4% compared with the CLB only version.

Frequency

As seen in Figure 10, all five of the benchmarks ran faster on the FPGA architecture that used the embedded FPUs. The speed increase comes from the fact that the control logic has a shorter critical path than the floating-point units that are based on CLBs. Because the control logic was optimized for the floating-point units created out of the embedded multiplier and not for the faster embedded FPUs, the speed increases are underestimated. On average the benchmarks with embedded FPUs had a speed increase of 40.7% over the embedded multiplier version and a speed increase of 85.1% over the CLB only version. The matrix multiply had the smallest speed increase of only 19.5% and 47.3% for the FPU version over the embedded multiplier and CLB only versions, respectively. The reason for the smaller speed increase for the matrix multiply is a larger benchmark that has a larger number of block RAMs. This results in more congestion and consequently longer routes. The matrix vector multiply has the largest speed increase of 69.6% and 107.2% for the FPU version over the embedded multiplier and CLB only versions respectively. The matrix vector multiply has a smaller amount of control logic than the other benchmarks. The critical path is in the floating-point calculations; thus, having the embedded FPUs significantly reduces the critical path and results in a higher frequency.

Routing Tracks

As seen in Figure 11, on average the FPU had a reduction of 8.6% in the number of routing tracks compared to the embedded multiplier version. This is not considered a major benefit or detriment of using

Table 8. Comparison of area, maximum frequency, and channel width for benchmarks with CLBs only, embedded multipliers, and FPUs

	CLBs Only			Embedded Multipliers			Embedded FPU		
	Area [10^6 L^2]	Freq [MHz]	Routing Tracks	Area [10^6 L^2]	Freq [MHz]	Routing Tracks	Area [10^6 L^2]	Freq [MHz]	Routing Tracks
Matrix Mult	23,945	104	40	20,037	126	41	9,662	151	40
Vector Mult	24,283	78	48	20,187	117	50	7,265	199	36
Dot Product	20,565	104	38	15,223	149	41	4,544	220	35
FFT	33,856	92	55	28,443	104	50	16,792	145	52
LU	23,461	105	62	18,169	142	61	7,640	182	59
Average	25,222	97	47	20,412	132	49	9,181	179	44

embedded FPUs, but rather an indication that using the embedded FPUs does not cause a dramatic change in the number of routing tracks. In fact, the FPU version had a maximum reduction in the number of routing tracks of 28.0% compared with the embedded multiplier version for the matrix vector benchmarks. The FPU version of the FFT benchmark had the maximum increase in the number of routing tracks of 4.0% compared to the embedded multiplier version.

5.3. Flops

The performance of each benchmark was calculated in Gflops and normalized to a given architecture area. The results are shown in Figure 12. The FPU benchmarks had an average performance of 13.38 Gflops, which is an increase by a factor of 2.4 over the average performance of the embedded multiplier benchmarks of 3.99 Gflops and an increase by a factor of 4.4 over the average performance of the CLB only benchmarks of 2.49 Gflops. However, since the FFT benchmark uses the embedded FPUs to perform only one floating-point operation, either a multiply or an add but not both, the benefits of the embedded FPUs is greatly reduced. Removing the FFT benchmark from the average, the FPU benchmarks had an average performance increase by a factor of 2.6 over the embedded multiplier benchmarks and an average performance increase by a factor of 5.1 over the CLB only benchmarks. While removing the FFT from the average provides an upper bound for the benefit, it does provide insight into the potential advantages of making the multiplier and adder independently accessible. As noted earlier, the performance advantage of using embedded FPUs is grossly underestimated because new critical paths in the control logic are exposed. With sufficient effort, it is expected that the control logic performance could be increased significantly.

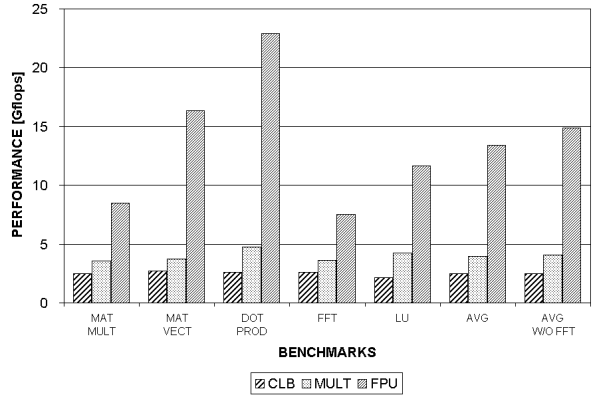

Figure 12. Normalized benchmark performance

5.4. Other Topics

Single-Precision 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.

Other Uses for FPUs

It has been shown that by adding embedded FPUs to an island-style FPGA the circuit size can be reduced and the maximum frequency can be increased for benchmarks that use floating-point without adversely affecting the number of routing tracks. However, if unused these embedded FPUs would be wasted space. The FPUs are comprised of large multipliers, adders, and barrel shifters. Small modifications to the design would expose these sub-components as alternative uses of the FPU block. This could help offset a potential disadvantage to using embedded FPUs.

FPU Area Overhead

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.

6. Conclusion

This paper has demonstrated that adding embedded floating-point multiply-add units into an island-style FPGA can significantly reduce circuit size and increase circuit frequency without a significant increase in channel width compared with floating-point multiply-add units that use traditional approaches. Despite a "worst case" area estimate, the embedded FPUs provided a significant advantage. The FPUs provided an average reduction in area of 55.0% compared to an FPGA enhanced with embedded 18-bit x 18-bit multipliers and 63.6% reduction in area compared to using only CLBs. This area achievement is in addition to an average speed improvement of 40.7% over using the embedded 18-bit x 18-bit multipliers and a 85.1% speed increase over the CLB only version. There is also an average reduction in the number of routing tracks by an average of 8.6% and a maximum increase in routing tracks of a mere 4.0%.

7. Future Research

While this paper has shown the potential of adding embedded FPUs to FPGAs, there is still room for improvement. The size estimation that was used for the FPU was very conservative. Obtaining a more accurate FPU size might improve results. Additional optimization of the way the FPUs were configured has potential for improvement. Also, since all 64 bits of a double-precision number generally tends to follow the same route, another possibility for improvement would be to implement bus based routing.

There are alternatives to FPUs that could also make FPGAs more conducive to floating-point computations without being as extensive. These could include smaller embedded units, changes to the logic blocks, or more efficient routing.

8. Acknowledgements

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

9. References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[10] 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.

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

[12] 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.

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

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

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

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

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

[18] 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.

[19] 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.