FPGAs vs. CPUs: Trends in Peak Floating-Point Performance

Keith Underwood*
Sandia National Laboratories
PO Box 5800, MS-1110
Albuquerque, NM 87185-1110
kdunder@sandia.gov

ABSTRACT

Moore's Law states that the number of transistors on a device doubles every two years; however, it is often (mis)quoted based on its impact on CPU performance. This important corollary of Moore's Law states that improved clock frequency plus improved architecture yields a doubling of CPU performance every 18 months. This paper examines the impact of Moore's Law on the peak floating-point performance of FPGAs. Performance trends for individual operations are analyzed as well as the performance trend of a common instruction mix (multiply accumulate). The important result is that peak FPGA floating-point performance is growing significantly faster than peak floating-point performance for a CPU.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance Attributes; C.1.3 [**Other Architecture Styles**]: Adaptable Architectures

General Terms

Performance

Keywords

FPGA, floating point, supercomputing, trends

1. INTRODUCTION

The consistency of Moore's law has had a dramatic impact on the semiconductor industry. Advances are expected to continue at the current pace for at least another decade yielding feature sizes of 45 nm by 2009[1]. Every two years the feature size for CMOS technology drops by over 40%. This translates into a doubling of transistors per unit area and a doubling of clock frequency every two years. The microprocessor industry has translated this into a doubling of CPU performance every 18 months. Over the last six years, clock rate has yielded a 12× improvement in CPU performance while architectural changes have only yielded a 1× to 4× improvement (depending on the operation considered). The additional transistors are typically used to compensate for the "memory wall"[2]. Design constraints and legacy instruction sets prevent CPU architects from moving the necessary state closer to the computation; thus, additional functional units would go underutilized.

Unlike CPUs, FPGAs have a high degree of hardware configurability. Thus, while CPU designers must select a resource allocation and a memory hierarchy that performs well across a range of applications, FPGA designers can leave many of those choices to the application designer. Simultaneously, the dataflow nature of computation implemented in field programmable gate arrays (FPGAs) overcomes some of the issues with the memory wall. There is no instruction fetch and much more local state can be maintained (i.e. there is a larger "register set"). Thus, data retrieved from memory is much more likely to stay in the FPGA until the application is "done" with it. As such, applications implemented in FPGAs are free to utilize the improvements in area that accompany Moore's law.

Beginning with the Xilinx XC4085XL, it became possible to implement IEEE compliant, double-precision, floating-point addition and multiplication; however, in that era, FPGAs were much slower than commodity CPUs. Since then, FPGA floating-point performance has been increasing dramatically. Indeed, the floating-point performance of FPGAs has been increasing more rapidly than that of commodity CPUs. Using the Moore's law factors of 2× the area and 2× the clock rate every two years, one would expect a 4× increase in FPGA floating-point performance every two years. This is significantly faster than the 4× increase in CPU performance every three years. But area and clock rate are not the entire story. Architectural changes to FPGAs have the potential to accelerate (or decelerate) the improvement in FPGA floating-point performance. For example, the introduction of 18 × 18 multipliers into the Virtex-II architecture dramatically reduce the area needed to build a floating-point multiplier. Conversely, the introduction of extremely high speed I/O and embedded processors consumed

*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.

FPGA'04, February 22-24, 2004, Monterey, California, USA.

area that could have been used to implement additional programmable logic[1].

These trends, coupled with the potential of FPGAs to sustain a higher percentage of peak performance, prompted this analysis of floating-point performance trends. Modern science and engineering is becoming increasingly dependent on supercomputer simulation to reduce experimentation requirements and to offer insight into microscopic phenomena. Such scientific applications at Sandia National Labs depend on IEEE standard, double precision operations. In fact, many of these applications depend on full IEEE compliance (including denormals) to maintain numerical stability. Thus, this paper presents the design of IEEE compliant single and double precision floating-point addition, multiplication, and division. The performance and area requirements of these operators, along with the multiply accumulate composite operator, is given for five FPGAs over the course of 6 years. From this data, long term trend lines are plotted and compared against known CPU data for the same time period. Each line is extrapolated to determine a long term "winner".

The remainder of this paper is organized as follows. Section 2 presents related work on floating-point in FPGAs. The implementation of the floating-point operations is then presented in Section 3. Section 4 presents the comparison of FPGA and CPU performance trends. Finally, Section 5 presents conclusions and Section 6 presents future work.

2. BACKGROUND

This work is motivated by an extensive body of previous work in floating-point for FPGAs. A long series of work[3, 4, 5, 6, 7] has investigated the use of custom floating-point formats in FPGAs. There has also been some work in the translation of floating-point to fixed point[8] and the automatic optimization of the bit widths of floating-point formats[9]. In most cases, these formats are shown to be adequate for some applications, to require significantly less area to implement than IEEE formats[10], and to run significantly faster than IEEE formats. Most of these efforts demonstrate that such customized formats enable significant speedups for certain chosen applications. Unfortunately, many scientific applications depend on both the dynamic range and high precision of IEEE double-precision floating-point to maintain numerical stability. Thus, this work focuses on the IEEE standard. Indeed, some application developers within the DOE labs are beginning to discuss the need for greater precision than the standard IEEE formats, and such formats may be the topic of future work.

The earliest work on IEEE floating-point[11] focused on single precision and found that, although feasible, it was extremely slow. Later work[12] found that the performance was improving, but still relatively poor. Eventually, it was demonstrated[13] that while FPGAs were uncompetitive with CPUs in terms of peak FLOPs, they could provide competitive sustained floating-point performance. Since then, a variety of work[14, 4, 7, 15] has demonstrated the growing feasibility of IEEE compliant, single precision floating-point arithmetic and other floating-point formats of approximately that complexity. Indeed, some work[16] suggests

[1]This is not to say that these innovations are not good or not important to the future of FPGAs in computing.

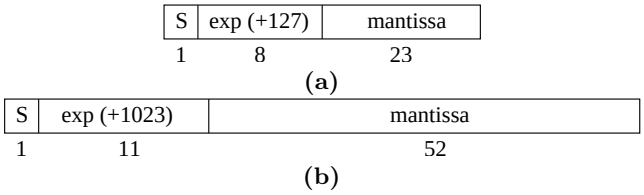


Figure 1: IEEE floating-point formats: (a) single precision and (b) double precision

that a collection of FPGAs can provide dramatically higher performance than a commodity processor.

A number of these works have focused on optimizing the format and the operators to maximize FPGA performance. In [14] a delayed addition technique is used to achieve impressive clock rates. In other work, [4, 7], the details of the floating-point format are varied to optimize performance. The specific issues of implementing floating-point division in FPGAs has been studied[15]. Similarly, [17] studied how to leverage new FPGA features to improve general floating-point performance, but neither covered double precision arithmetic. Indeed, little work has considered the performance of IEEE double precision floating-point and no work to date has considered the trends in FPGA performance.

3. IMPLEMENTATION

Three IEEE 754 compliant arithmetic operations (addition, multiplication, and division) were implemented for both single precision and double precision formats. Figure 1 illustrates the IEEE format. The exponents are maintained in biased notation (bias-127 or bias-1023 as noted). Exponents of zero and the maximum value of the field (255 and 2047, respectively) are reserved for special values. The mantissa is maintained with an implied one. That is, in all but special cases, the mantissa is formed by adding a "1" before the value stored. The decimal place is always immediately to the left of the stored value. This requires numerous normalization elements in compliant implementations.

The IEEE 754 standard has a number of interesting features that must be taken into account for a compliant implementation. First, there are exception conditions that require the generation of NaN, infinity, and zero. Second, these special values must be handled as inputs to the operations. Third, gradual underflow (or denormal processing) must be provided. Finally, IEEE specifies four rounding modes.

These floating-point units provide correct output in all exception conditions, but do not provide special exception signals. It is unclear how such signals would be used in an FPGA design, but addition of these signals would be trivial if needed. They also provide proper handling of all special values and provide full denormal processing. Round-to-nearest-even is the IEEE rounding mode provided in the initial implementation, but other rounding modes could be added as options.

Each of these experiments used Synplify 7.3 from Synplicity running on a Redhat 9 platform for synthesis. IOB flip-flops were prohibited and aggressive clock rate targets (relative to realized clock rates) were used. Each design was synthesized for 5 parts from Xilinx families spanning the last 6 years (the choice of parts will be discussed in Section 4).

Table 1: IEEE double precision floating-point adder characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	1334	1226	960	33
Virtex 1000-5	1433	1408	1096	78
Virtex-E 3200-7	1458	1349	1075	94
Virtex-II 6000-5	1407	1408	1090	125
Virtex-IIP 100-6	1406	1408	1090	143

Table 2: IEEE single precision floating-point adder characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	587	600	441	38
Virtex 1000-5	622	651	487	85
Virtex-E 3200-7	622	693	501	110
Virtex-II 6000-5	611	696	496	165
Virtex-IIP 100-6	611	696	495	195

Xilinx 4.2i tools on a Solaris platform were used to place and route the XC4000XLA series part and Xilinx 5.2i tools on a Solaris platform were used for all other parts. Timing constraints were adjusted to optimize final timing results with an "extra effort level" of 2. No floorplanning or placement constraints were used at this time.

3.1 Addition Implementation

IEEE floating-point addition consists of three major functions: pre-normalization, operation (including rounding), and post-normalization. Pre-normalization and post-normalization are shift operations requiring shifts (in increments of one) from 0 to N, where N is the width of the mantissa including the implied 1. The nature of these shifts provides proper handling of denormal cases with very little modification. The operation at the core of floating-point addition is either an addition or subtraction (depending on the signs of the inputs) that is $N + 4$ bits wide. The increase in width is necessary to handle the possibility of carry out and to handle all rounding cases.

The floating-point addition units accept one parameter — a maximum latency. This parameter controls the number of pipeline registers that are inserted. This is handled by inserting a total of 14 register components (for double precision, 13 for single precision) that accept a boolean to indicate if a register should be inserted. If more than half of the registers are requested, additional registers are added at the start of the pipeline (where the stages are slightly more complicated). Register insertion points were chosen with a number of metrics. Heavy emphasis was placed on regularity of the pipeline with registers preferentially placed in places that would yield an approximately constant width. Pre-normalization and post-normalization shifts were broken into smaller shifts to reduce routing and logic complexity with an optional register at each shift stage.

Tables 1 and 2 give the area requirements and performance of the double and single precision floating-point adders when pipelined to their maximum depth. This table does not show 3-LUT usage for the XC4000 series because 3-LUTs seldom dominate area usage. There is a slight variation in the area usage across device families. These relatively minor variations are an artifact of the optimizations performed by the tool chains (including such optimizations as logic replication to reduce fan-out). One clear trend is that the mapper uses far more slices than would appear to be necessary. Some experimentation was done to determine if a "more full" device would prompt the mapper to pack the logic more densely, but there was no significant impact observed. Careful hand optimization would likely cut the number of slices used by almost 30%.

Table 3: IEEE double precision floating-point multiplier characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	4381	4940	2954	25
Virtex 1000-5	4535	5047	3810	50
Virtex-E 3200-7	4535	5171	3844	68
Virtex-II 6000-5	2016	2354	1607	105
Virtex-IIP 100-6	2016	2354	1608	142

Another striking trend observed in Tables 1 and 2 is the relatively small performance gain when moving from double to single precision. This is because the longest pipeline stage is an early pipeline stage which is handling exponent calculations and manipulating the mantissa. These stages change little between the single and double precision operations — the exponent only changes in length by three bits and mux width has relatively little impact on performance.

3.2 Multiplication Implementation

The fundamental multiplication operation is conceptually simple: multiply the mantissas, round, and add the exponents. The multiply (of the mantissas) is not required to maintain the full precision of an $N \times N$ multiply. All of the partial products must be generated, but the summation can accumulate most of the lower bits into a single "sticky bit". The result only needs to be maintained to two bits wider than the mantissa (plus the sticky bit) to preserve enough information for proper rounding. Unfortunately, compliance with the IEEE standard complicates this significantly. First, the inputs could be denormal. If so, maintaining proper precision (without a full $N \times N$ multiplier) requires the smaller number to be normalized. If the larger input is denormal, the result will underflow. Second, one input could be the special value, NaN. If so, the exact value of the NaN input must be preserved and propagated. Finally, two non-denormal inputs could produce a denormal result. This requires that the final output be normalized, or rather denormalized, appropriately.

In this implementation, the multiply of the mantissas was separated into a separate component. This hides the multiply implementation from the rest of the design (save for the pipeline depth of the multiply which must be specified in the upper level design). This was important because there is a wide variety of multiply implementations offering optimizations for such things as datapath regularity, latency, or use of the embedded multipliers. Two multiplier implementations were developed in this design — one that uses the

Table 4: IEEE single precision floating-point multiplier characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	1661	1579	1103	38
Virtex 1000-5	1268	1487	1080	89
Virtex-E 3200-7	1279	1487	1087	113
Virtex-II 6000-5	772	821	598	124
Virtex-IIP 100-6	762	821	592	176

embedded multipliers (and requires nine of them for double precision) in Virtex-II and Virtex-IIPro and a second (pipelined bit serial) that is very regular in structure for use in XC4000 and Virtex series parts.

The floating-point multiplier design has three parameters. The maximum latency specifies the pipeline stages which are allocated as they are in the adder. The worst case latency is 20 cycles for the double precision multiplier (or 39 with no embedded multipliers) and 16 cycles for the single precision multiplier (or 24 with no embedded multipliers). The second parameter provides the option to support denormals. Eliminating denormals implies that denormal numbers are treated as zeroes for both inputs and outputs and saves significant area. The final parameter provides optional use of the embedded multipliers. This parameter must be false for XC4000, Virtex, and VirtexE designs and may be false for newer parts if desired.

Tables 3 and 4 highlight the performance characteristics of the double precision and single precision multipliers, respectively. Area requirements drop substantially when the embedded multipliers are used. Performance of the Virtex-II implementation would be significantly higher, but Stepping-0 parts were used instead of the Stepping-1 parts[2], which would provide significantly higher performance embedded multipliers. As with the adder, hand placement of the multiplier could probably reduce the slice utilization by as much as 30%.

3.3 Division Implementation

The implementation of floating-point division is very similar to that of floating-point multiplication. It must divide the mantissas, round, and subtract the exponents. A full precision divide is required, but the first N steps can be skipped because the inputs are normalized. The remainder is used to calculate the "sticky" bit. As with the multiplier, compliance with the IEEE standard is complicated. Denormal inputs must be normalized, but in this case if both inputs are denormal, both must be normalized. Similarly, NaN inputs must be preserved and non-denormal inputs can create a denormal output that requires normalization.

As with the floating-point multiplier, the divide of the mantissas is separated from the rest of the design to simplify the substitution of better divide implementations. Only one divide implementation (pipelined bit serial) was provided. This is probably not the best divide option (in terms of area); however, it is adequate to be competitive with a commodity CPU. Two parameters are available for the divider.

[2]Stepping-1 is a silicon revision of the Virtex-II parts that provides a significant boost in the performance of the embedded multipliers.

Table 5: IEEE double precision floating-point divider characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	4910	9199	4821	23 (est.)
Virtex 1000-5	8061	9231	6856	50
Virtex-E 3200-7	8076	9323	6909	58
Virtex-II 6000-5	7981	9391	6858	83
Virtex-IIP 100-6	7976	9391	6880	98

Table 6: IEEE single precision floating-point divider characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	1583	2380	1498	28
Virtex 1000-5	2207	2475	1925	60
Virtex-E 3200-7	2214	2473	1928	78
Virtex-II 6000-5	2200	2476	1929	100
Virtex-IIP 100-6	2199	2476	1927	120

The first is a maximum latency that is used to allocate registers in the design in the same way it is used in the multiplier and adder. The worst case latency is 67 cycles for the double precision divider and 37 cycles for the single precision divider. The second parameter enables (or disables) the support of denormals, which can provide a substantial area savings at the cost of IEEE compliance.

Tables 5 and 6 list the performance and area requirements of the double precision and single precision dividers. The double precision divider did not fit on the XC4085XLA so the performance was estimated using the ratio between double and single precision multiply (a similar operation) on the same device. Surprisingly, a single double precision floating-point unit in an FPGA has throughput to match that of a commodity CPU. The latency (in terms of clock cycles and absolute time) is substantially worse in the FPGA, but the advantage in throughput is substantial. As with the other components, the slice utilization is substantially higher than seems necessary.

3.4 Multiply Accumulate Implementation

Multiply accumulate is the fundamental composite operator for a number of operations including matrix multiply, matrix vector multiply, and a number of matrix solution methods including the popular LINPACK benchmark[18] that qualifies supercomputers for the Top500 list[19]. As such, multiply accumulate was chosen for comparison along with the other basic operations. The implementation of multiply accumulate simply combines a multiplier and an adder with some appropriate control logic. Because both the multiplier and adder are deeply pipelined, multiple concurrent multiply accumulates must occur to maximize performance (although correctness is maintained in either case). The multiply accumulate operator accepts all of the parameters available on the multiplier or adder as well as `CONCURRENCY`, which controls the number of concurrent multiply accumulates, and `VLENGTH`, which controls the number of results from the multiplier that are accumulated into one result.

Table 7: IEEE double precision floating-point multiply accumulate characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	5812	6512	3679	25 (est.)
Virtex 1000-5	6333	6512	5031	50
Virtex-E 3200-7	6333	6528	5034	63
Virtex-II 6000-5	3804	3818	2877	100
Virtex-IIP 100-6	3806	3818	2875	140

Table 8: IEEE single precision floating-point multiply accumulate characteristics.

Part	4-LUTs	FFs	Slices	MHz
XC4085XLA-09	2339	2363	1673	37
Virtex 1000-5	2125	2242	1699	78
Virtex-E 3200-7	2137	2242	1707	110
Virtex-II 6000-5	1617	1573	1206	127
Virtex-IIP 100-6	1612	1573	1203	175

Tables 7 and 8 list the characteristics of the double and single precision multiply accumulate operations, respectively. Note that performance of the double precision multiply accumulate on the XC4085XLA must be estimated as it did not fit in the device. The performance of the slower component from the composite operation (the double precision multiplier) was used for this estimate. Again, some variation in the 4-LUT, FF, and Slice counts is seen between families with minor variations in the optimization done by the synthesizer. In each case, the latency of the multiplier and the adder in the multiply accumulate is the same as was used in the stand alone tests.

4. PERFORMANCE PROJECTIONS

In this section, the peak floating-point performance of FPGAs is derived from the implementation data in Section 3. Peak FPGA performance is calculated as the number of functional units that can be instantiated times the clock rate. In turn, the number of functional units that can be instantiated is simply the number of slices (or CLBs in the case of the XC4000 series) available divided by the number of slices required for the functional unit. Although these are first order approximations, this number is approximately as realistic as peak performance numbers for a commodity microprocessor. Peak performance is presented for five FPGA devices over six years and is compared with three commodity microprocessors over the same time period.

Peak floating-point performance is extrapolated for both the microprocessors and FPGAs. For the microprocessor, performance is doubled every 18 months to represent the well known corollary of Moore's law. The trend line is forced through the 2003 data point for microprocessors. Since no such corollary has been established for FPGAs, the trend line is derived by starting with the 1997 data point, selecting a very conservative fit, and rounding the slope down.

Table 9 lists the parts chosen for the performance comparison. The commodity microprocessor with the highest peak performance was chosen for each of three years (regardless of the release date during the year). Since microprocessor performance trends are well known, only three data points were deemed necessary. Similarly, five FPGAs were chosen for data points over the course of the same 6 years. An effort was made to choose the largest, fastest speed grade part with reasonable availability during that year[3]. For 2001, a stepping 0 Virtex-II 6000-5 device was chosen. Device stepping 1 was released early the next year and significantly improved the embedded multiplier performance. The part chosen for 1997 was chosen to be as representative as possible of the devices that would have been available; however, there was a constraint on which devices could be placed and routed for these experiments. The oldest tools that were available were Xilinx 4.2i, which do not support parts older than the XC4000XLA series. The XC4085XL-2 might have been a better choice, but the tools available would not target that device.

Table 9: Parts used for performance comparison

Year	FPGA	CPU
1997	XC4085XLA-09	Pentium 266 MHz
1999	Virtex 1000-5	
2000	Virtex-E 3200-7	Athlon 1.2 GHz
2001	Virtex-II 6000-5	
2003	Virtex-II Pro 100-6	Pentium-4 3.2 GHz

Admittedly, there are limitations to this type of analysis; however, the conservative assumptions that were made and the dramatic performance improvements projected should compensate for such limitations so that the "answer" is unchanged. For example, the order of magnitude performance advantage in 2009 may carry the same cost premium as current large devices. However, cheaper members of the same FPGA family will likely achieve a cost and performance advantage since FPGA performance is typically linear within a family, but cost is almost exponential. A second limitation is the lack of accounting for structures such as address generation and integer computation units. Such units are typically very simple in an FPGA. Furthermore, a hand placed version of the floating-point functional units should yield at least a 10% clock rate increase and a 25% reduction in area. The space that is freed (including the removal of 10% of the functional units while maintaining performance) should be more than adequate to support address generation structures. Finally, I/O between the CPU and FPGA is not considered. The underlying assumption is that FPGAs (with embedded CPUs) will either replace the primary CPU in a supercomputer or will be important enough to be coupled into a supercomputing system in such a way as to mitigate the I/O issues.

4.1 Addition

Figures 2 (a) and (b) indicate that FPGAs have significantly higher floating-point addition performance than commodity CPUs. This is a surprising revelation since FPGAs are generally considered to provide poor floating-point performance at best. More surprising still is the fact that FP-

[3]These selections were made based on input from Chuck Cremer at Quatra Associates, Jason Moore at Xilinx, and personal memory.

GAs have been ahead for almost four years. The trend line for floating-point addition on FPGAs projects a growth rate of 4× every two years. This trend line is diverging from the performance trend line for CPUs, which is only 2× every 18 months. Notably, double precision addition performance on CPUs has been growing slower than the trend line for the last 6 years, but the single precision addition performance has been growing slightly faster than the trend line. This is primarily attributable to a dramatic one time architectural improvement that was achieved with the addition of multimedia instructions.

FPGAs achieve this significant advantage over CPUs because their resources are configurable. If an application requires a very addition rich mixture of instructions, the FPGA can provide that. In contrast, commodity CPUs have a fixed allocation of resources that can be successful for a well mixed instruction stream, but has significant limitations in code that is dominated by one instruction.

4.2 Multiplication

FPGAs have not dominated CPUs in floating-point multiplication performance for nearly as long they have in floating-point addition performance. Figures 2(c) and (d) indicate that FPGAs have only exceeded CPUs in multiplication performance for three years in single precision arithmetic and two years in double precision arithmetic. However, the trend lines are diverging more rapidly with multiplication performance growing at an average rate of 5× per year over the last 6 years. This is primarily because of the addition of 18×18 fixed multipliers in recent parts. The use of these components to implement the multiply of the mantissas (9 for double precision, 4 for single precision) dramatically reduced the area required. This trend is likely to continue since architectural improvements (notably faster, wider, fixed multipliers) are likely to continue.

The CPU performance in Figures 2 (c) and (d) has grown slightly faster than the Moore's law trend line over the last 6 years. For double precision, this is primarily because of a change between 1997 and 2000 to allow multiplies to issue every cycle. For single precision, it is primarily from the addition of multimedia instructions that utilize SIMD parallelism. In both cases, similar improvements are not on the processor roadmaps over the next few years.

4.3 Division

As seen in Figures 2 (e) and (f), FPGAs have long exceeded CPUs in floating-point division performance — with one minor caveat. An XC4085XLA is not big enough (by a significant margin) to implement a double precision divider. Thus, the "performance" of a double precision divider on that part is the fraction of the divider it can implement times the estimated clock rate. This explains why the 4× trend line overestimates the performance of the 2003 part: components that can implement the operation are constrained to integer multiples of divide units. The XC4085XLA had a second artificial performance inflation because the mapper packed the CLBs much tighter to try (in vain) to make it fit. Thus, the area estimate is significantly smaller than it would otherwise be. For the single precision divider, performance of FPGAs has tracked a 4× trend line fairly well.

Commodity microprocessors are not well optimized for divide operations. This is a significant issue for some scientific applications[20]. Slow divides have first and second order effects: the division instructions are slow (and unpipelined) and these slow instructions clog the issue queue for modern microprocessors. This makes divide rich applications a good target for FPGA implementations.

4.4 Multiply Accumulate

Multiply accumulate is somewhat different from the other operations considered in that it is a composite operation. More importantly, it is a composite operation that is fundamental to a number of matrix operations (including LINPACK). Unfortunately, Figure 3 indicates that FPGAs are still somewhat slower than CPUs at performing this operation. FPGAs are, however, improving at a rate of 4.5× every two years. This improvement rate (effectively a composite of the performance improvements in addition and multiplication) yields a significant win for FPGAs by the end of the decade.

It would be easy to suggest that the comparison between FPGA and CPU in this case is not "fair" because the FPGA requires many concurrent multiply accumulates (in one multiply accumulate functional unit) to overcome the latency of the adder and achieve the projected performance; however, it should be noted that the Pentium-4 must alternate issuing two adds and two multiplies[4] with 4 cycle and 6 cycle latency, respectively, to achieve its peak performance [21]. With the small register set in the IA-32 ISA, this is not necessarily easier to exploit than the concurrency and parallelism available on the FPGA.

4.5 Analysis

A common theme among the performance graphs is the flattening of the performance trend line from 2000 to 2003. This is supported by the data in Figure 4, which clearly indicates a flattening in the growth of area, increase in clock rate, and feature size reduction. This appears to bode ill for the projected performance trends; however, a closer look at Figure 4 indicates differently. In Figure 4(a), the trend in FPGA feature size broke sharply between 2001 and 2003, but it is still on the same overall trend line for the 6 year period as CPUs. Indeed, low cost FPGAs have already been introduced on the 90 nm process — well ahead of CPUs. High performance parts are expected to be introduced next year concurrently with CPUs based on 90 nm technology. Similarly, Figure 4(b) shows that the pace of FPGA density improvements has dropped sharply from 2000 to 2003, but overall density increases are still above the Moore's law projection. Even if a much larger device (the XC40125EX) was used as the 1997 baseline, the overall density improvement would remain slightly above the Moore's law projection.

Figure 4(c) seems to tell a slightly different story with regards to clock rate. The "Moore's law" trend line for clock rate provides a reference that clearly indicates that clock rate has not scaled as expected. However, this seeming discrepancy is relatively easy to explain. This device used as representative of 1997 technology for these experiments was the XC4085XLA-09. A more accurate part would have been the XC4085XL-2, but the Xilinx 4.2i tools that were available for these experiments would not process such a device. A XC4085XL-2 part is approximately 40% slower than the XC4085XLA-09 part used. Combining this with the significant performance increase that Virtex-II Pro parts

[4]The throughput of the SSE2 multiplier is one instruction per 2 cycles. Each instruction can do two multiplies.

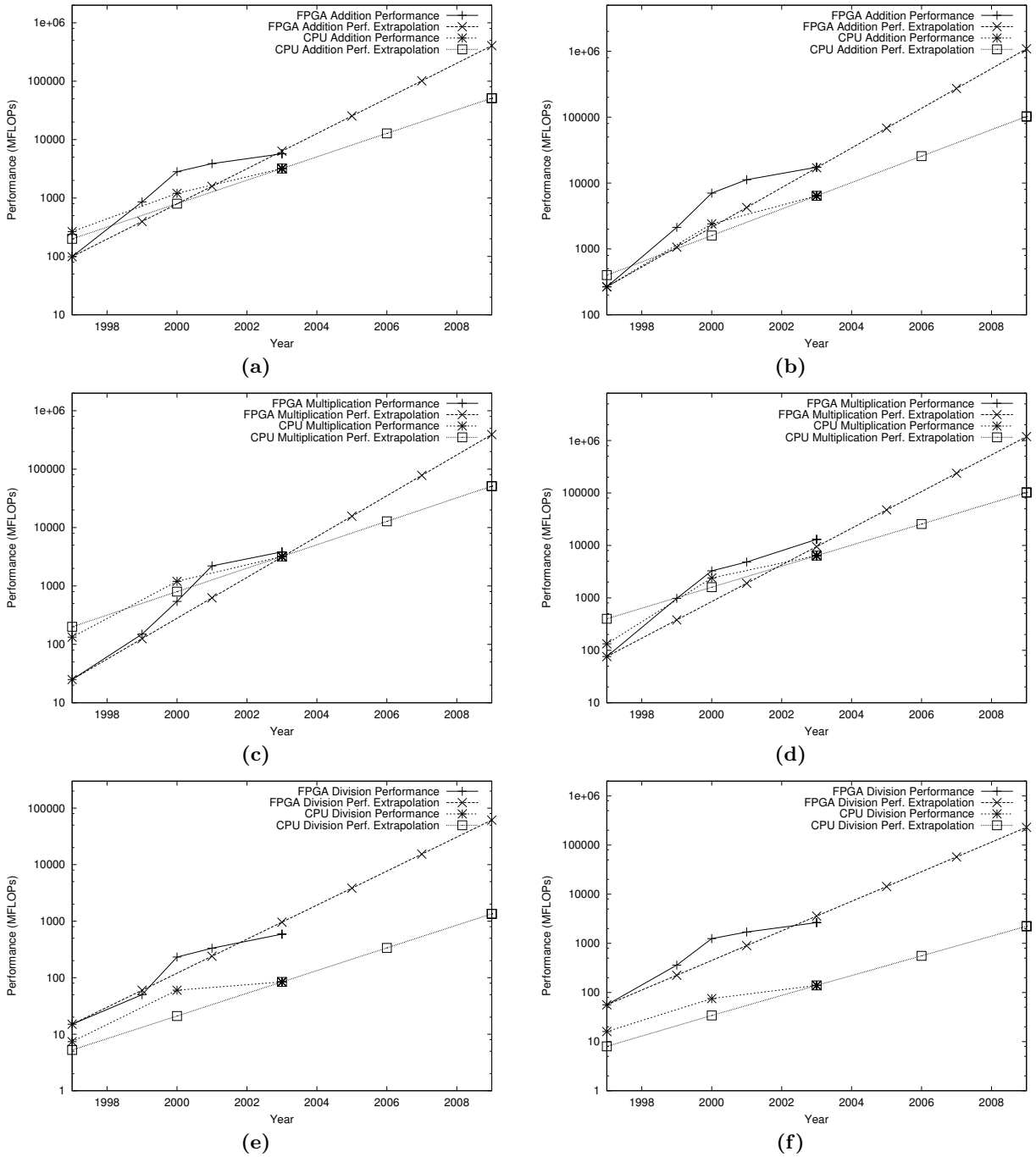


Figure 2: Floating-point addition performance for: (a) double precision, and (b) single precision. Floating-point multiplication performance trend for: (c) double precision, and (d) single precision. Floating-point division performance trend for: (e) double precision, and (f) single precision.

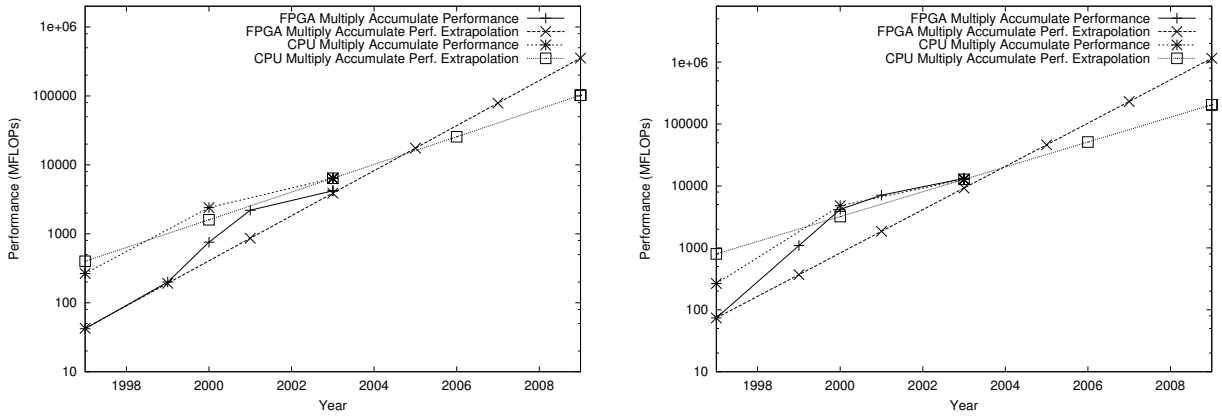


Figure 3: Floating-point multiply accumulate performance trend for: (a) double precision, and (b) single precision.

should receive as the tool chain develops trends in clock rates that meet the expectations of Moore's law.

Improvement in addition and division performance are derived strictly from technology improvement; however, the 5× every two years performance growth of multipliers will be difficult to maintain indefinitely. Fortunately, only minor improvements are needed each generation to realize this gain. This should be readily achievable through 2009 (wider embedded multipliers, enhanced support for shifting, etc.).

5. CONCLUSIONS

This paper has presented designs for single and double precision IEEE compliant floating-point addition, multiplication, and division. For each of these operations, an FPGA has a higher peak floating-point performance than a CPU. This occurs because FPGAs are able to customize the allocation of resources to meet the needs of the application while CPUs have fixed functional units. CPUs currently have higher peak performance than FPGAs for applications that fully utilize the CPU's fixed resources. An excellent example of this is the multiply accumulate primitive which is the core of matrix multiplication and matrix vector multiplication. Preliminary work[22], however, indicates that real applications can have instruction mixes that are highly biased toward a single type of operation.

The more important result is the illustration of the trends in floating-point performance for both FPGAs and CPUs. While CPUs are following a well known corollary of Moore's law (doubling in performance every 18 months), FPGA performance is increasing by 4× every two years. For operations that use the architectural improvements in FPGAs, performance is increasing at a rate of 5× every two years. Projecting these trends forward yields an order of magnitude advantage in peak performance for FPGAs by 2009. This has important implications for the design of supercomputers in that era since FPGAs should sustain as high, and potentially higher, percentage of peak than CPUs.

6. FUTURE WORK

Peak performance is the metric most commonly used when announcing the latest supercomputer acquisition. If for no other reason, this makes it an important metric to study; however, in the end, sustained performance is a better metric for how quickly a system can perform work. As an example, the Top500 list[19] uses LINPACK[18] performance as a metric. Unfortunately, for many applications, LINPACK performance is closer to peak performance than sustained performance on real applications. Future work will focus on floating-point computational cores that are representative of the workload at Sandia. These cores will be implemented to compare the sustained performance of FPGAs and CPUs. Furthermore, a portion of these studies will focus on how an FPGA might be integrated in future supercomputers to leverage these performance advantages.

7. REFERENCES

[1] *International Technology Roadmap for Semiconductors*. December 2003.

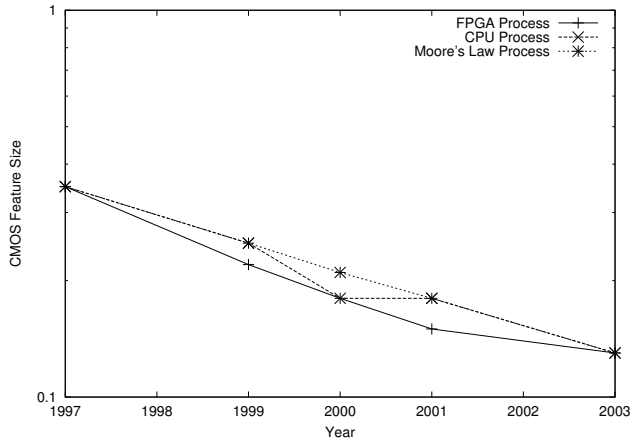
[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *Computer Architecture News*, vol. 23, pp. 20–24, March 1995.

[3] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on fpga based custom computing machines," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 155–162, 1995.

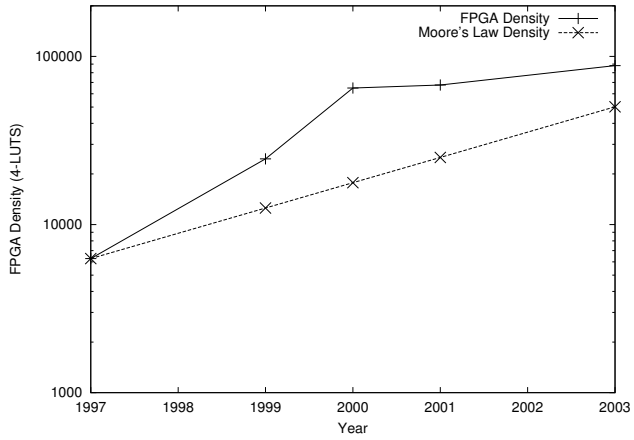
[4] P. Belanovic and M. Leeser, "A library of parameterized floating-point modules and their use," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[5] J. Dido, N. Geraudie, L. Loiseau, O. Payeur, Y. Savaria, and D. Poirier, "A flexible floating-point format for optimizing data-paths and operators in fpga based dsps," in *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, (Monterrey, CA), February 2002.

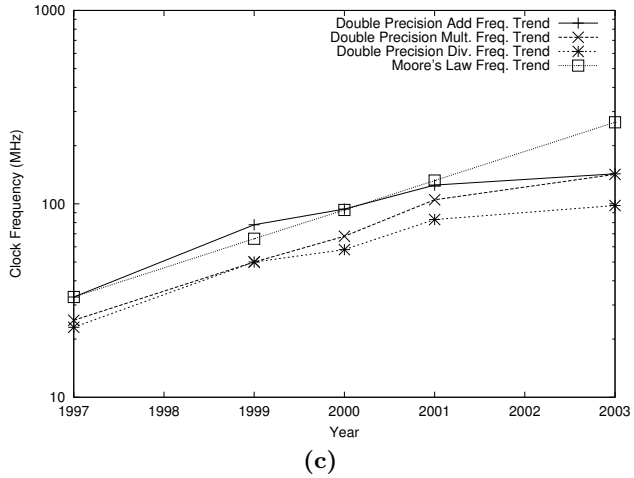
[6] A. A. Gaar, W. Luk, P. Y. Cheung, N. Shirazi, and J. Hwang, "Automating customisation of



(a)



(b)



(c)

Figure 4: Basic FPGA property trends including: (a) CMOS Feature size, (b) density in 4-LUTs, and (c) clock rate.

floating-point designs," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[7] J. Liang, R. Tessier, and O. Mencer, "Floating point unit generation and evaluation for fpgas," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, (Napa Valley, CA), pp. 185–194, April 2003.

[8] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong, "Automatic floating to fixed point translation and its application to post-rendering 3d warping," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, (Napa Valley, CA), pp. 240–248, April 1999.

[9] A. A. Gaar, O. Mencer, W. Luk, P. Y. Cheung, and N. Shirazi, "Floating point bitwidth analysis via automatic differentiation," in *Proceedings of the International Conference on Field Programmable Technology*, (Hong Kong), 2002.

[10] IEEE Standards Board, "IEEE standard for binary floating-point arithmetic," Tech. Rep. ANSI/IEEE Std. 754-1985, The Institute of Electrical and Electronics Engineers, New York, 1985.

[11] B.Fagin and C. Renard, "Field programmable gate arrays and floating point arithmetic," *IEEE Transactions on VLSI*, vol. 2, no. 3, pp. 365–367, 1994.

[12] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of ieee single precision floating point addition and multiplication on fpgas," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 107–116, 1996.

[13] W. B. Ligon, S. P. McMillan, G. Monn, F. Stivers, K. Schoonover, and K. D. Underwood, "A re-evaluation of the praticality of floating-point on FPGAs," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, (Napa Valley, CA), pp. 206–215, April 1998.

[14] Z. Luo and M. Martonosi, "Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques," *IEEE Transactions on Computers*, vol. 49, no. 3, pp. 208–218, 2000.

[15] X. Wang and B. E. Nelson, "Tradeoffs of designing floating-point division and square root on virtex fpgas," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, (Napa Valley, CA), pp. 195–203, April 2003.

[16] W. D. Smith and A. R. Schnore, "Towards and RCC-based accelerator for computational fluid dynamics applications," pp. 226–232, 2003.

[17] E. Roesler and B. Nelson, "Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture," in *Proceedings of the 12th International Workshop on Field Programmable Logic and Applications (FPL'2002)*, pp. 637–646, August 2002.

[18] J. J. Dongarra, "The linpack benchmark: An explanation," in *1st International Conference on Supercomputing*, pp. 456–474, June 1987.

[19] "Top 500 web site," September 2003. URL: http://www.top500.org.

[20] J. S. Vetter and A. Yoo, "An empirical performance evaluation of scalable scientific applications," in *Proceedings of the 2002 Conference on Supercomputing*, Nov. 2002.

[21] *IA-32 Intel Architecture Optimization: Reference Manual*. USA: Intel Corporation, 2003. Order Number:248966-009.

[22] A. Rodrigues, R. Murphy, P. Kogge, and K. Underwood, "Characterizing a new class of threads in scientific applications for high end supercomputers," in *in Preparation*.