Area, and Power Performance Analysis of a Floating-point based Application on FPGAs

Gokul Govindu, Ling Zhuo, Seonil Choi, Padma Gundala, Viktor K Prasanna
Department of Electrical Engineering, Systems
University of Southern California
Los Angeles, CA 90089
{govindu, lzhuo, seonilch, pgundala, prasanna}@usc.edu

Introduction

Almost all signal processing algorithms are initially represented as double precision floating-point in languages such as Matlab. For hardware implementations, these algorithms have to be converted to large precision fixed-point to have a sufficiently large dynamic range. However the inevitable quantization effects and the complexity of converting the floating-point algorithm into a fixed point one, limit the use of fixed-point arithmetic for high precision embedded computing.

FPGAs have become an attractive option for implementing computationally intensive applications. However, the common conception has been that efficient FPGA implementations of floating-point arithmetic have a lot of performance, area and power overheads compared to fixed-point arithmetic. With recent technology advances, FPGA densities are increasing at a rate at which area considerations are becoming less significant. These advances have also reduced the performance and power overhead of floating-point arithmetic. With appropriate designs, floating-point applications can even be more efficient than fixed-point ones for large bitwidths. The overheads in the context of the overall application can be quite low. In this paper, we present a preliminary area, and power performance analysis of double precision matrix multiplication, an extensively used kernel in embedded computing and also show that FPGAs are good candidates for implementing high precision floating-point based applications when compared to a general-purpose processor.

Currently many FPGA based floating-point units, both open source [2] and commercial [1], are available. However, most of them consider only single precision floating-point operations, and do not make use of the recent advances in FPGAs. Moreover, an area, and power performance analysis of the floating-point units in the context of a common application is lacking.

Description of our Floating point units and the Matrix Multiply architecture

For matrix multiplication, we require add and multiply floating-point units. Our floating-point units follow the IEEE 754 single and double precision (64-bit) format. We developed both deeply pipelined and moderately pipelined units. The units essentially consist of three stages: denormalization, the add/multiply, and normalization/rounding/renormalization. Exception handling at all stages is done and enable/done signals are provided for easy integration into a pipelined architecture. The implementation of floating-point units involves extensive use of fast fixed point adder/subtractors, multiplier units, and large bus multiplexers (for shifting operations). Recent FPGAs, such as Virtex-II Pro [4], provide a large number of embedded multipliers as well as fast carry chains for addition. Similarly, large multiplexers used in shifting can make use of the MUXCY, MUXF attributes on the FPGAs. Recent FPGA fabrics also contain a lot of registers, which can be utilized for extensive pipelining between stages.

We used the block matrix multiplication architecture from [3] in which a linear array of n processing elements is used for an n x n matrix multiplication. Each processing elements essentially consists of an adder, a multiplier, storage elements, and related control logic. Since the matrix multiply architecture (see [3] for more details) is modular, multiple chips can be used in an array for large n. Here we use the GFLOPS per device for a given n as the performance metric.

Analysis of the Floating-point units

Table 1a and 1b show a comparison of the fixed and floating-point units for a bitwidth of 32 and 64. We see that the overhead for double precision is less than that for single precision. Note that, for the fixed-point designs, truncation to make the output bitwidth equal to the input bitwidth results in a lot of quantization error. Moreover the fixed-point multiplier unit takes up more embedded multipliers than the floating-point unit. We also show a comparison between an extensively pipelined and a moderately pipelined version of the floating-point units. We see that extensive pipelining to increase the clock frequency requires a lot of area for the registers in between the pipeline stages. The pipelining done to split the adder/multiplier, the large priority encoder and the shift registers for the normalizing unit shows an immediate improvement in frequency, without much increase in area. Further pipelining, shows diminishing returns in frequency and the area increases significantly. Hence a design trade-off will be the frequency required which influences the number of pipelining stages and area. Here, for the double precision matrix multiply, we decided to use the moderately pipelined units since we can achieve higher GFLOPs. From synthesis results, we saw that normalization takes up a lot of area (560 slices for the deeply pipelined and 200 slices for the moderately pipelined units, for double precision) and can also be the critical path for timing (because of a large priority encoder and shift registers). Hence a design trade-off would be the use of custom formats in the architecture, with conversion from and back to the IEEE754 standard at the interface to say, a processor. Considering power, the 64bit fixed-point multiplier unit with more embedded multipliers consumes a lot more power. Note that, for the power values of individual units, only clocks, logic and signal powers were included.

	32, 64bit Fixed-point (with 2, 4 pipeline stages)		32, 64bit Floating-point (with 6, 8 pipeline stages)		32, 64bit Floating-point (with 18, 23 pipeline stages)	
Area (slices)	36	139	293	693	504	1383
Max Freq. (MHz) achievable	250	250	140	130	250	200
Power (mW) at 100MHz	23.48	104	148.7	329	-	-

Table 1a: A comparison of addition units (Virtex2Pro-c2vp125-7)

	32, 64bit Fixed-point (with 5, 7 pipeline stages)		32, 64bit Floating-point (with 9, 11 pipeline stages)		64bit Floating-point (with 18, 23 pipeline stages)	
Area (slices)/Embedded multipliers	190 / 4	1024 / 16	249 / 3	775 / 10	492 / 3	1558 / 10
Max Freq. (MHz) achievable	200	130	140	130	200	200
Power (mW) at 100MHz	136.3	804	164.7	424	-	-

Table 1b: A comparison of multiplication units (Virtex2Pro-c2vp125-7)

Analysis of the Matrix Multiply

The area and power performance overhead of the floating-point units has to be seen in the context of an application. Table 2 shows the area and power performance of both fixed and floating-point implementations of a double precision, *n*-point matrix multiply on a FPGA. The double precision implementation shows us an interesting result of the floating-point unit having a better performance than the fixed-point implementation. The maximum number of fixed-point processing elements that the device can accommodate when block RAMs are used for storage, is smaller than the number of processing elements when slice based RAM is used. This is probably because of more routing resources used up due to the fixed locations of the block RAMs and the embedded multipliers. Moreover, the number of slices on a given device being constant, the device will accommodate fewer processing elements if deeply pipelined units occupy a large area. Hence, the performance of the device might be lower even if the frequency of the units is high. Also, the overall application's architecture's operating frequency should be considered. Performance was measured as one multiplication and one addition happening every clock cycle in each processing element. The total power for the matrix multiply takes into account output, input, quiescent, logic, signals and the clocks power. We see that floating-point unit overheads in terms of area, and power performance are not too drastic. Table 3 shows the performance comparison of a floating-point based *n*-point matrix multiplication both on an FPGA and a Pentium4 SSE2, 1.5GHz processor. The performance of the design on FPGAs shows a 3.48x improvement over that of the processor. Moreover the power per GFLOP of the FPGA is much lower than that of the processor.

	Fixed-point based		Floating point based (moderately pipelined) using block RAM	Floating point based (deeply pipelined) (estimated)
	using block RAM	using slice based RAM		
Area (slices) / BRAM / multipliers of each Processing element of matrix multiply	1344 / 4 / 16	1626/ 0 / 16	1872 / 4 / 10	3441 / 4 / 10
Maximum number of processing elements on the device	28	32	29	16
Frequency (MHz) of each element	130	130	130	200
Power of each PE (mW) at 100MHz	843	894	762	-
Frequency (MHz) achieved for the matrix multiply	110	110	120	200
Performance of matrix multiply, per device-Virtex2Pro xc2vp125-7	6.16 GOPS	7.04 GOPS	6.96 GFLOPS	6.4 GFLOPS
Total Power (W) per GOP or GFLOP for matrix multiply,	27.2 / 6.16 = 4.41	33.04/ 7.04 = 4.68	26.4 / 6.96 = 3.79	-

Table 2: A comparison of double precision, fixed and floating-point, *n*-point Matrix Multiply, requiring *n* processing elements on FPGAs (Virtex2Pro-c2vp125-7)

	FPGA (Virtex2Pro xc2vp125-7)	Pentium4 with SSE2 (1.5GHz)
GFLOPS	6.96	2
Power (W) per GFLOP	26.4 / 6.96 = 3.79	57.9/2 = 28.95

Table 3: A comparison of the performance of Matrix Multiply

All the above results were obtained after the VHDL code was synthesized and placed and routed using the Xilinx ISE5.2i, on a Virtex2Pro XC2VP125-7f1696 device. Power values were obtained from Xpower. The Pentium4 SSE2 results were from [5]. Better results can be obtained after the units have been optimized more, by manually placing them.

Conclusion and Future Work

We have presented a preliminary analysis of a floating-point implementation of a computationally intensive application on FPGAs. We show that when the floating-point units are considered in the context of an application, their overheads in terms of area, and power performance are not too drastic. We also show that a significant increase in performance can be obtained on FPGAs over general-purpose processors with much lower power expended. Future work will involve extensive analysis of the floating-point units to identify more design trade-offs. We will also provide a documented and extensively tested, open source library of the floating-point units, shortly.

References

1. Nallatech, www.nallatech.com
2. P. Belanović, M. Leeser, *Library of Parameterized Floating-point Modules and Their Use,* International Conference on Field Programmable Logic (ICFPL), September 2002.
3. J. Jang, S. Choi, and V. K. Prasanna, *Area and Time Efficient Implementation of Matrix Multiplication on FPGAs*, International Conference on Field Programmable Technology (ICFPT), December 2002.
4. Xilinx, www.xilinx.com
5. J. Dongarra. *An Update of a Couple of Tools: ATLAS and PAPI*. Technical report, DOE Salishan Meeting, April 2001.