ISSN:2348-2079

MAY-2016

International Journal of Intellectual Advancements and Research in Engineering Computations

Fused Multiply and Add Unit with Simulation and Verification Results

Jyoti Singh Chouhan, Nitin Jain

ABSTRACT

Floating point unit (FPU) is one of the most important custom applications needed in most hardware designs as it adds accuracy and ease of use. Recently, the floating point units of several commercials processors like IBM power PC; Intel/HP titanium, MIPS-compatible loongson-2F and HP PA-8000 have included a floating point fused multiply add (FMA) unit to execute the double –precision fused multiply add operation A+(B*C) as an indivisible operation, with no intermediate rounding.

INTRODUCTION

The FMA operation is very important in many scientific and engineering applications like digital signal processing (DSP), finite impulse response (FIR) filters, graphics processing, fast Fourier transform (FFTS), division and argument reduction. The first FMA is introduced in 1990 by IBM RS/6000. After that FMA is implement by several companies like HP, MIPS, ARM and Intel. It is a key feature of the floating _point unit because it greatly increases the floating point performance and accuracy since roundly is performed only once for the result A+ (B*C) rather than twice for the multiplier and then for the adder. It also realizes reduction in the latency and hardware cost. FMA can be used instead of floating-point addition and floating –point multiplication by using constants e.g. 0.0+ (B*C) for multiplication and A+ (B*1.0) for addition.

A field programmable gate array, FPGA, provides a versatile and inexpensive way to implement and test VLSI designs. It is mostly used in low volume application that cannot afford silicon fabrication or deigns which require frequent changes or upgrades.

In FPGAS, the bottleneck for designing efficient floating-point units has mostly been area and accuracy. With advancement in FPGA architecture, there is a significant incrcase in FPGA densities so latency has been the main focus of attention in order to improve the performance and error correction.

Proposed Fused Multiply Add

The proposed fused multiply add is actually the first implementation of the Lang/ Bruguera fused multiply add algorithm. The implementation is done with slight change in Lang/ Bruguera architecture. In Lang/Bruguera design the LZA and the sign detection modules have three inputs coming from the output of the multiplier and the aligned addend, but in the proposed design the inputs to these modules come from the output from multiplier after combining the outputs and the aligned addend. The sign detection module is used to detect negative sign of output. If the sign is negative the output of the 3:2 CSA is complemented. Comparison is needed between the output of the multiplier B×C (i.e. two output vectors from CSA reduction tree) and the aligned A. To compare between these three vectors there are two solutions, adding the two vectors output from the multiplier first by a carry propagate adder then comparing its output with aligned A or reducing these three vectors using 3:2 CSA then comparing the output of this CSA. The second solution is preferred to eliminate using of carry propagate adder in order to decrease its delay because the output of this block, comp signal, is used after that to select inputs to normalization shifter which are needed to be available as soon as possible before shift amount of normalization shifter is calculated.

Handling the Error of Inexact LZA

Earlier we mentioned that LZA is not always exact. In some cases, the actual leading 1 is located one position to the right. Although solutions have been proposed for exact LZA, there is always a tradeoff between area/energy efficiency and guaranteed correctness. Exact solutions eliminate the need for correction and thus improve latency, however, the latency reduction often

comes at the cost of a considerable area increase. In many cases where area is of importance, postponed correction is much more attractive because it can be implemented with considerable less hardware. The price one pays for misprediction is not very high. A two-to-one multiplexer in terms of latency, and an adder or subtracter for exponent correction; insignificant compared to the cost of exact LZA. Correction after misprediction is easily achieved. After shifting the significand to the left by the amount resulting from counting the leading zeros in the indicator vector, a simple check on the MSB of the `normalized' significand reveals the error. If the MSB is 1, then the prediction was correct and no further action is required. If the MSB is 0, then the prediction was incorrect and the significand needs to be shifted one more position to the left, and the exponent decremented. In terms of hardware this can be realized by using two exponent subtracters in parallel (one that subtracts the amount predicted by the LZA and one that subtract the same amount plus one) and selecting the result based on the MSB of the shifted significand. Another solution would be to conditionally increment the amount to subtract.

Algorithm For Leading zero detection

1: Form a pair of bits B_i, B_{i+1} for $0 \leq i \leq (n-1)$.
2: Determine the P and V bit for each pair.
3: **while** depth $< log_2\{n\}$ **do**
4: Determine the P_{next} and V_{next} bits from the P and V bits as follows:
5: $V_{next} = V_{left} \vee V_{right}$
6: **if** $V_{left} = 1$ **then**
7: $P_{next} = 0$ & P_{left} (concatenation)
8: **else if** $V_{right} = 1$ **then**
9: $P_{next} = 1$ & P_{right} (concatenation)
10: **else**
11: $V_{next} = 0$
12: **end if**
13: **end while**

Design details of blocks used in proposed FMA: To implement this architecture there is a need to fully design the sign detection module with two inputs and redesign add/round module to include the correction of LZA error. In the remaining part of this chapter we will include the design of these blocks and the design of blocks that were modified from the basic architecture to be suitable in this implementation like LZA block and the anticipated part of the adder.

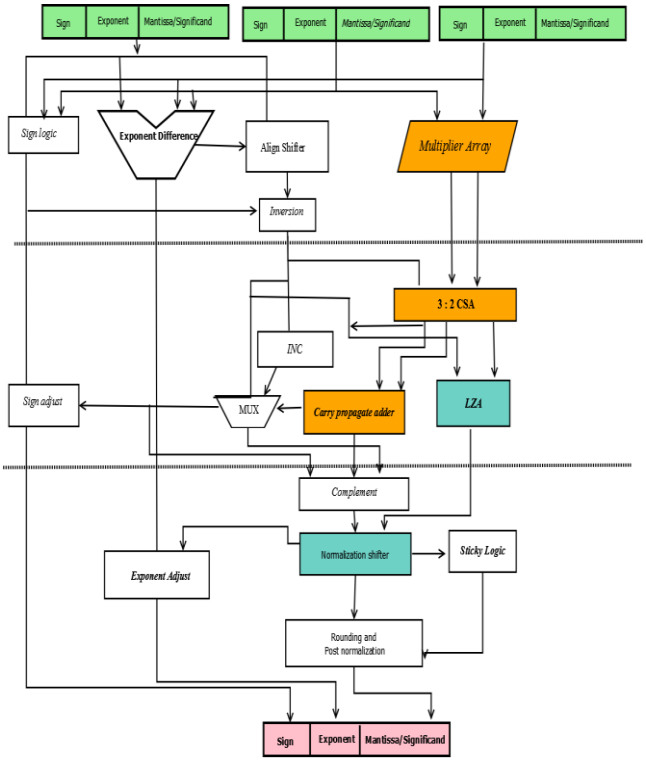


Figure 1 The proposed Fused Multiply Add with modified LZA for Error correction

Leading zero anticipator (LZA) and Normalization shifter

As cited before in chapter 3 the LZA block is used to determine the normalization amount. It is composed of two main modules the detection and the correction modules. The detection module has two main parts, the LZA logic which determines the position of the leading one by getting a string of bits having the same number of leading zeros as the sum and LZD which encodes the number of leading zeros in a binary representation. In the reduced latency Fused Multiply Add the LZA block is in the critical path. Lang/Bruguera made this modification by replacing the LZD by a set of gates and Multiplexers. The basic idea of the encoding is to check the existence of the leading one in different groups of bits in LZA logic output where the number and the position of the checked bits depend on the weight of the output bit .

Figure shows which groups are explored to get each bit for a 16-bit normalization shift, for example, to get the most significant bit s1 the 8 most-significant bits of LZA logic output are checked and if all of them are zeros (s1=1) a normalization shift is needed by 8 bits. To get s2 two different groups of four bits are checked and then selected by s1 and so on. Figure shows the implementation of this algorithm. NOR gates are used to determine if there is some 1 in different groups of the string. Each bit si+1 is obtained after a delay of 2:1 Multiplexer from the previous bit *ssss*.

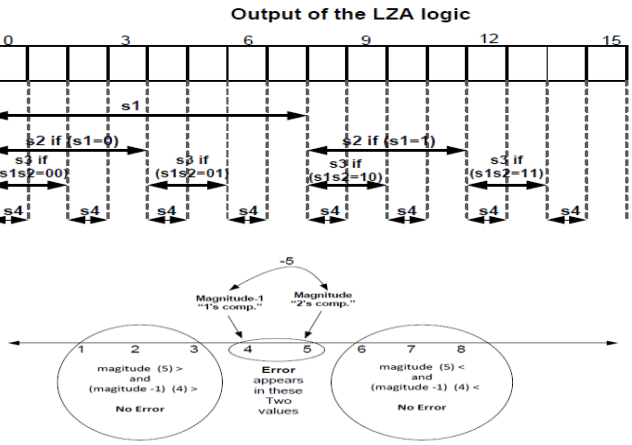


Figure 2 Comparison (magnitude) and (magnitude-1) of -5 with any positive number.

RTL schematic

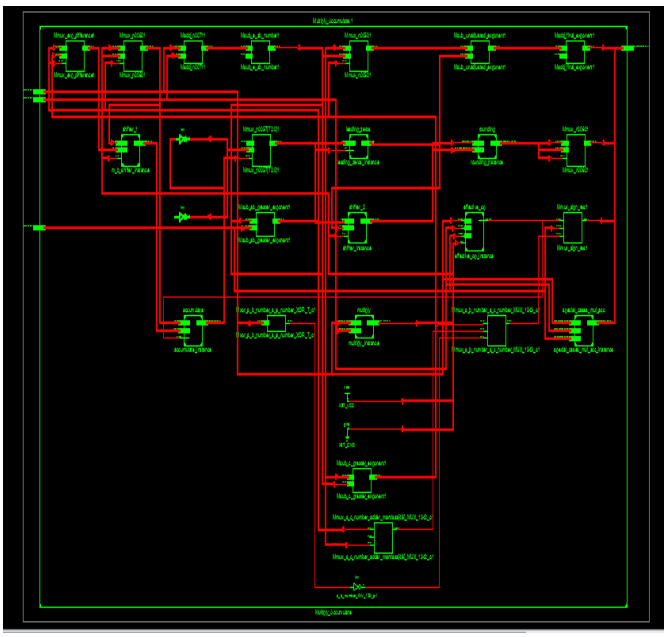
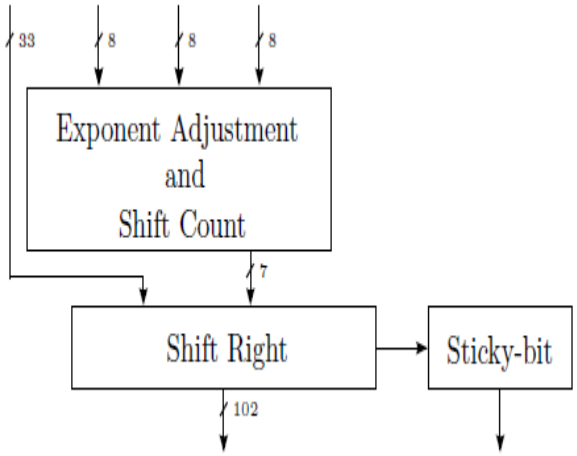


Figure 2 RTL Schematic of FMAC

Alignment Shift and Exponent Adjustment: After passing through the instruction decoder/operand formatter block, operand C is aligned with the product while operands A and B are multiplied to form the product itself .Alignment of two floating-point numbers consists of two actions: adjusting the exponent of one operand to match the exponent of the other, and shifting the significand of that same operand to match the new exponent. Adjustment of the exponents is often not discussed in publications related to floating-point arithmetic. This is mostly because it is assumed to be trivial. In a purely mathematically sense this is true, viz., multiplication is just addition of the exponents and for floating-point addition/subtraction the exponent simply equals the exponent of the operand that is not shifted. In binary floating-point arithmetic however, we have to deal with overflow, underflow and non-regular (e.g., zero and infinity) input. Especially in the case of FMA, exponent adjustment is anything but trivial.



Exponent Adjustment

The IEEE-754 oating-point interchange formats specify that the exponent is biased. This choice was made for historical reasons because comparing biased numbers by hand is a lot easier and quicker than comparing for example two's complement numbers. Unfortunately biased notation entails a few very unfortunate adverse properties for arithmetic implementation. Some of these properties include the accumulation of the bias during addition and the difficulty of underflow detection. A solution that is often implicitly assumed is to convert the exponent to a signed representation. In a signed representation (usually two's complement) exponents can be added and subtracted without having to worry about bias accumulation and underflow is easier to detect because the sign-bit inverts. However, we have chosen not to convert the exponent to a signed representation for two reasons: Conversion would add additional delay to the critical path of the first stage. All exponents would have to be converted before the actual computation starts, making it very difficult to overlap the conversion delay with other processing delays. If we assume two's complement representation, conversion would require a converter consisting of an inverter a subtracter and an incrementer. In terms of area and energy efficiency, this is costly. The solution we propose is to keep using the biased notation and extend the exponent by one bit. This means we still have to compensate for bias accumulation, but at least this can be done in parallel with the significand multiplication. Because we extended the exponent by one bit, overflow is easily detected. When after addition, either the MSB is 1 or all the other bits are 1, the intermediate exponent has exceeded the maximum representable number . Both cases are easily detectable. The MSB OR'ed with the AND-reduced LSBs provides a 1-bit overflow control signal (1 meaning overflow).

Simulation Verification and Results: The proposed architecture of the fused multiply add is implemented in the Verilog hardware description language. ModelSim and Xilinx 14.1 are used to compile Verilog codes and to simulate them. The simple way to test the design is to write a test bench that exercises various features of the design.

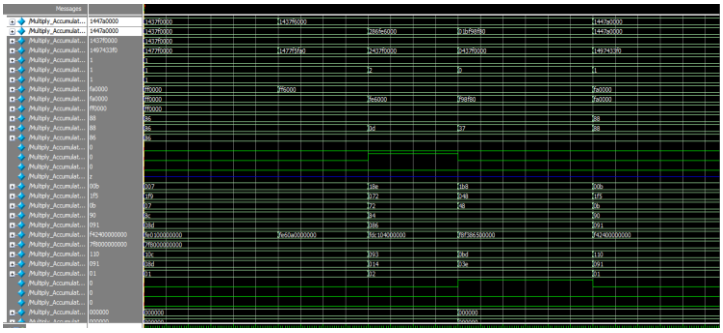


Figure 3 Example ModelSim run for floating-point fused multiply add unit

In the FPGA's technology logic functions are implemented by truth tables defined by programming the SRAM that defines the FPGA's functionality. Each of the truth tables has a delay which contributes to the delay from inputs to outputs of the function being implemented. In addition, the connections in the FPGA between the inputs, truth tables, and outputs pass through buffers, multiplexers and pass transistors as determined by the circuit specification and the routing paths determined by the implementation tools. The decomposition into truth tables combined with the routing of the interconnections between them yields considerable uncertainty in the propagation delay from input to output of an implemented circuit. The worst case delay which occurs in the circuit from any combinational logic input to any combinational logic output is determined by adding up the maximum expected delays through the combinational circuit including both logic and interconnections. To decrease uncertainty we use constraints to specify the maximum delay allowable, forcing the tools to attempt to meet or better this delay.

CONCLUSION

We discuss in this paper design of blocks used to implement the proposed fused multiply add unit like sign detection and add round modules. Sign detection module produces the *cccccccc* signal which is used after that to select inputs to the normalization shifter. These inputs are needed to be available as soon as possible before shift amount of normalization shifter is calculated. Because of this reason we use a new scheme to implement sign detection module to reduce its latency. We use magnitude minus one (1's complement) instead of magnitude (2's complement) in magnitude comparator to eliminate the need of carry propagate adder.

REFERENCES

[1]. Chen, C., Chen, L., and Cheng, J., "Architectural Design of a Fast Floating-Point Multiplication-Add Fused Unit Using Signed-Digit Addition," Proceedings of the IEE Computer Digital Technology, vol. 149, no. 4, July 2002, Pages: 113 – 120.

[2]. Dimitrakopoulos, G., Galanopoulos, K., Mavrokefalidis, C., and Nikolos, D., "Low-Power Leading-Zero Counting and Anticipation Logic for High-Speed Floating Point Units," IEEE Transactions on VLSI Systems, vol. 16, no. 7, July 2008, Pages: 837 – 850.

[3]. Hinds, C., "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications," Proceedings of the 33th Asilomar Conference on Signals, Systems, and Computers, vol. 1, October 1999, Pages: 147 – 151.

[4]. Hinds, C., and Lutz, D., "A Small and Fast Leading One Predictor Corrector Circuit," Proceedings of 39th Asilomar Conference on Signals, Systems and Computers, October 2005, pages: 1181– 1185.

[5]. Hokenek, E., and Montoye, R., "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating Point Execution Unit," IBM Journal of Research and Development, vol. 34, no.1, January1990, pages: 71-77.

[6]. Lang, T.; Bruguera, J.D., "Floating-point multiply-add-fused with reduced latency," Computers, IEEE Transactions on , vol.53, no.8, pp.988,1003, Aug. 2004

[7]. Zichu Qi; Qi Guo; Ge Zhang; Xiangku Li; Weiwu Hu, "Design of Low-Cost High-PerformanceFloating-Point Fused Multiply-Add with Reduced Power," VLSI Design, 2010. VLSID '10. 23rd International Conference on , vol., no., pp.206,211, 3-7 Jan. 2010

[8]. Saleh, H.; Swartzlander, E.E., "A floating-point fused add-subtract unit," *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on* , vol., no., pp.519,522, 10-13 Aug. 2008