Design & Analysis of High-Performance Floating-point Fused Multiply-add with Reduced Latency

Ms. Aruna Jyothi C B
Assistant Professor
Ekalavya Institute of Technology, Ummathur, Chamarajanagar

Mrs. Bhagyavathy V K
Assistant Professor
K V G College of Engineering, Sullia, DK

Abstract: Floating-point unit is an integral part of any modern microprocessor. The fused multiply add (FMA) operation is important in many scientific and engineering applications. With the advancement in FPGA architecture latency has been the important focus of attention in order to improve performance. To improve the performance, two single-precision operations can be performed concurrently with one double-precision data path, which is very useful in multimedia and even scientific applications. Moreover, to reduce the additional area costs for supporting two single-precision operations in parallel, multiple double-precision units, i.e., the multiplier, shifter and adder, are reused as much as possible. In addition, in case of FADD instructions, the multiplier in the first stage is bypassed and kept in stable mode, which can significantly improve fused addition instruction performance and lower power consumption. The paper claims an estimated 15-20% reduction in latency as compared to a conventional fused multiply add (FMA).

Keyword: floating-point unit (FPU), fused multiply-add (FMA)

I. INTRODUCTION

Floating-point unit (FPU) is one of the most important scientific applications needed in most hardware designs because it adds accuracy and ease of use. Recently, the floating-point units of several commercial processors like IBM PowerPC, Intel/HP Itanium [5] have included a floating-point fused multiply add (FMA) unit to execute the double-precision fused multiply add operation $A+(B\times C)$ as an indivisible operation, with single step rounding.

The FMA operation is very important in many scientific and engineering applications like digital signal processing (DSP), fast Fourier transform (FFTs), Finite impulse response (FIR) filters, graphics processing. The first Fused multiply add is introduced in 1990 by IBM RS/6000. After that FMA is implemented by several companies like Intel, MIPS, ARM and HP. Key feature of the FPU is, it greatly increases the floating-point performance and accuracy since rounding is performed only once for the FMA rather than twice for the multiplier and then for the adder. Fused multiply add can be used for only floating-point addition or floating-point multiplication by using constants e.g., $0.0+(B\times C)$ for multiplication and $A+(B\times1.0)$ for addition.

A Field Programmable Gate Array (FPGA), provides an inexpensive way to implement and test VLSI designs. It is most used in low volume applications where silicon fabrication cannot be afford or designs which require frequent changes or upgrades. In FPGAs, the bottleneck for designing efficient floating-point units has mostly been area. With advancement in FPGA architecture, there is a significant increase in FPGA densities so latency has been the main focus in order to improve performance.

A. Conventional FMA unit

*Corresponding Author: Aruna Jyothi C B
E-mail: aruna.raju89@gmail.com
Design & Analysis of High-Performance Floating-point Fused Multiply-add with Reduced Latency

Though many FMA architectures have been proposed and several improvements have been made, the basic architecture remains invariant. Since most of FMA have been compared with the conventional FMA, let us go through a brief introduction of it. Figure 1 illustrates the architecture of the conventional FMA. It is divided into three main stages:

- Multiply the two input mantissas to produce a product result of A×B in carry-save format. Align the inverted addend C as a right shift by placing C at the position of two bits left of the most significant bit (MSB) of A×B. The two extra bits are used to allow correct rounding when A is not shifted. For the double-precision operation, the alignment shifter is 161-bit
- Combine the addend and the product using a 3:2 CSA (carry save adder). Then a 161-bit carry-save adder is used to produce the intermediate sum and carry. In parallel with the addition, normalization is done using leading-zero anticipator (LZA) to determine the shift amount. If the addition result is negative, complement should be done.
- At the last stage, normalization and rounding are performed to get the final result. The conventional FMA has the benefit of less area, but the delay is comparatively long as discussed in [6], which is difficult to be designed at high-frequency.

Fig. 1. Conventional FMA unit

B. Problem Statement

The conventional FMA has the benefit of less area, but the delay is comparatively long as discussed in [7], which is difficult to be designed at high-frequency.

So the objective of proposed FMA is to

- Reduce the latency
- Low power consumption.
- Improve the performance
- Maintain low area cost

If a Floating multiplication instruction has just completed its first stage and a Floating addition instruction is coming into the FMA, the hardware conflict of second stage occurs. Both instructions require the hardware of second stage at the same time. A method is being proposed to solve this conflict where FMA operations can be started at every cycle thereby achieving a throughput of one-instruction one-cycle.

To improve the performance, the proposed Fused multiply add supports parallel single-precision instructions, in which two single-precision operations are executed concurrently within one instruction.

To lower the area cost the double-precision unit is reused maximally. Since the processing units of double-precision mantissa are always two times wider than that of single-precision mantissa, it is possible to share most of the hardware by careful design. In the Fused multiply add, the shifter, adder, and multiplier are usually the main parts of the total area and hence these block needs modified design.
II. PROPOSED FMA ARCHITECTURE

The organization of Proposed FMA unit is shown in Figure 2. We can see that the addition of the multiplier product and the aligned addend is done firstly by carry save adder (CSA) then a carry propagate adder (CPA). The sticky bit calculation which is needed for rounding and the anticipation of the leading zeros which is needed for normalization are performed parallel with CPA. The standard architecture is the baseline algorithm for floating-point fused multiply-add in any kind of hardware and software design.

III. DETAIL IMPLEMENTATION

A. The Multiplier and Partial Product Generation

Multiplication is the process of generation and addition of the partial products using CSA. Multiplication algorithms differ in the method used to added together all partial products and how the partial products are added together to produce the final result. Floating point FMA unit includes a multiplier which uses a recoding Booth’s algorithm to generate partial products. In Booth’s algorithm the multiplier operand C is often in orders to reduce the number of partial products. The most common recoding is radix-4 recoding (modified booth’s recoding) is shown in table 1. For a series of consecutive 1’s, the recoding algorithm converts them into 0’s surrounded by a 1 and a (−1), which has the potential of reducing switching activity.

Each three consecutive bits of the multiplier C represent the input to booth recoding block and the output from this block selects the operation on the multiplicand B which may be “shift and invert” or “invert” or “equal zero” or “no operation” or “shift ” ( -2B, -B, 0, B, 2B) respectively due to the bits of multiplier. Also there is an output bit s to indicate the sign and complete the 2’s complement if the partial product is negative. For double
Design & Analysis of High-Performance Floating-point Fused Multiply-add with Reduced Latency

precision format the multiplier has 53 bits. By using modified booth recoding the number of partial products are reduced to 27 partial products. Figure 4 shows the generation of all 27 partial products.

Table 1. Modified Booth's Recording

<table>
<thead>
<tr>
<th>Bits of multiplier C</th>
<th>Encoding Operation on multiplicand B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{s1}, C_{s2}, C_{s3}</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>-B</td>
</tr>
<tr>
<td>0 1 0</td>
<td>-B</td>
</tr>
<tr>
<td>0 1 1</td>
<td>-2B</td>
</tr>
<tr>
<td>1 0 0</td>
<td>-3B</td>
</tr>
<tr>
<td>1 0 1</td>
<td>-5B</td>
</tr>
<tr>
<td>1 1 0</td>
<td>-3B</td>
</tr>
<tr>
<td>1 1 1</td>
<td>-5B</td>
</tr>
</tbody>
</table>

![Fig.4. All partial products generation](image)

B. The Alignment Shifter and 3:2 CSA

After generation of the partial products, next the partial Products begin compression using 3-2 CSA tree. The reduction occurs by rows where each three partial product in same level will be input to CSA adder and output 2 operands (i.e. partial products) to the next level, and so on. For 27 partial products, 8 stages are required to produce a product in carry-save, or a carry vector and sum vector that need only to be added for a complete multiply. To reduce the latency in a fused multiply add unit, the inversion and alignment of the significand of A is done in parallel with the multiplication. The inversion provides the one’s complement of A for an effective subtraction.

The alignment is implemented as a right shift by placing the addend A to the left of the most significant bit of the product B×C by 56 bits. The shift amount for the alignment depends on the value of d=Ea−(Eb+Ec), where Ea, Eb and Ec are the exponents of the A, B and C operands, respectively. This is shown in Figure 5(a). Two extra bits are placed between the addend A and the product B×C to allow correct rounding when A is not shifted. For d ≥ 0 with this implementation, A is right shifted (56−d) bits; then, the shift amount is shift amount=max {0, 56−d}, see Figure 5(b). For d<0, A is right shifted 56−d bits, see Figure 5(c), then shift amount=min {161, 56−d}. By combining both cases, the shift amount is in the range [0:161], requiring a 161-bit right shifter. Moreover, the shift amount is computed as shift amount=56−d.

![Fig.5. Alignment of A. (a) before alignment. (b) Alignment with d ≥ 0. (c) Alignment with d < 0.](image)

The multiplier produce 106-bit sum and carry vectors. If we multiply two positive number ,the output must be positive (sign and magnitude representation), but one of the two output vectors of the multiplier (sum and carry)
may be negative because of using booth algorithm which use negative sets \{-1,-2\} which convert a positive number with sign and magnitude representation to a negative number with two’s complement representation. The addition of sum and carry vectors must be a positive number but one of them may be negative.

Instead of using 161-bit CSA, Only the 106-bit of LSB of the aligned A are needed as input to the 3:2 CSA, because the product output (i.e. sum and carry vectors) has only 106 bits and The rest 55 most-significant bits will be sign extension bits which have two cases \{0, 0\} or \{0, 1\} if both sum and carry vectors are positive or if one of them is negative respectively. For the 55 most significant bits, we use two multiplexers, one to select between A and inverted A as a sum vector and the second one to select between zeros and A as a carry vector by Xor-ing sign extension bits then the outputs of the two multiplexers are concatenated at the output of the CSA to obtain the 161-bit sum and carry words, as shown in Figure 6.

![Fig.6. Selection of MSB of sum and carry words of CSA by sign bits](image)

C. Carry Propagate Adder (CPA) and Leading Zero Anticipator (LZA)

The output vectors of 3:2 CSA are now input to a 161-bit carry propagate adder (CPA) and a leading zero anticipator (LZA) in the same stage. We use the prefix adder to implement the carry propagate adder in this thesis because it is very efficient in binary addition due to its regular structure and fast performance.

After the addition operation is done the adder output is left shifted by the number of leading zeros to obtain a normalized result. It is called the normalization step. Instead of waiting for the adder output to determine the number of leading zeros it can be anticipated in parallel with the adder using a LZA to eliminate this operation from the critical path. LZA unit anticipates the number of leading zeros from the operands of the adder. Since the shift amount is already determined by LZA the normalization operation is performed once the result of addition is obtained.

The leading zero anticipator (LZA) has two main parts: the encoding of the leading-one position (detection module) and the correction of this position (correction module). The detection module are divided into two parts the first one is called LZA logic and it determines the number of leading zeros (i.e. the position of the leading one) by producing a string of zeros and ones where the position of the most significant 1 is the position of the leading one. The second part, called leading zero detector (LZD), counts the number of zero digits from the left-most position until the first nonzero digit is reached (i.e. leading one position), since the detection is done from most significant bit to least significant bit (from left to right) regardless of the carry that may come from the least significant bit, the detection of leading one position may be off by one bit.

![Fig.7. Block diagram of LZA module](image)
Design & Analysis of High-Performance Floating-point Fused Multiply-add with Reduced Latency

D. Sign Detection Module

The function of the sign detection module is to detect the sign of the adder output and complement the outputs of the CSA when the result is negative. The two’s complement of the CSA output is performed by inverting the sum and carry words and adding two in the least-significant position.

The result can be negative only for effective subtraction. Moreover, since, in effective subtraction, we always complement the significand of A, the result can be negative only for \( d \geq 0 \) (i.e. \( Ea > (Eb+Ec) \)). When \( d \geq 2 \), the result is always negative. While for \( d=0 \) or \( d=1 \) with overflow in multiplication (i.e. case of equal exponents) the result may be positive or negative. A magnitude (unsigned) comparison between the two vectors output from CSA has to be performed. In this case the shift amount is equal 56 or 55 so at least the 55 most significant bits of aligned A are sign extension. Also the sign bit in the multiplier output appears in bit 108, so we need only 109-bit magnitude comparator to include at least one sign bit in the two vectors.

E. The Normalization and The Rounding

Using the results from the LZD, the result from the adder is shifted left to normalize the result. That means now the first bit is 1. The normalizer is mostly a large shifter. The shifter has more than one stage. The last stage performs a shift by one or two due to correction signal. This should have a negligible effect on the delay of the last stage.

The rounding block rounds the result to nearest floating point number due to the round mode and performs post-normalization in case of an overflow. The round decision is taken by knowing also sticky and round bits. The sticky bit is calculated from the result by OR-ing all least significant bits after the round bit.

IV. RESULTS

The proposed architecture of the fused multiply add were implemented in the Verilog hardware description language, ModelSim ES 6.3f is a used to compile Verilog codes and to simulate them. The implemented architectures of fused multiply add are synthesized, placed and routed for FPGA device using XA3S400 Family Spartan3. The worst case delay of the proposed architectures of fused multiply add unit, it is shown in Table 2 it is clear that the delay improvement is 25.5%.

<table>
<thead>
<tr>
<th>Design</th>
<th>Worst-case delay ( ns )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional FMA</td>
<td>95.345</td>
</tr>
<tr>
<td>Proposed FMA</td>
<td>70.985</td>
</tr>
</tbody>
</table>

Aruna Jyothi C B and Bhagyavathy V K
Journal of Advanced Research and Science Volume 1 Issue 1 2015
Design & Analysis of High-Performance Floating-point Fused Multiply-add with Reduced Latency

The total number of logic elements that are used by FPGA is an important issue in comparing the performance in addition to the worst case delay. From compilation reports it is clear that the conventional architecture occupies 13,122 form 68,416 total logic elements which are corresponding to occupying 19% of total logic elements. The proposed architecture occupies 13,930 logic elements about 20% of total logic elements. The increment in the number of logic gates in the proposed architecture is about 6.2% only.

Both conventional and proposed architectures are similar in upper part (multiplier, bit invert and 3:2 CSA) and differ in the lower part, CPA, Complementer, normalizer and rounder in basic architecture, three parallel paths (i.e. sign detection ,LZA, normalization shifter and anticipated part of adder) and add round module for proposed architecture. Each block of lower part in two architectures is synthesized separately; Table 3 shows the worst case delay and the number of logic gates of each individual block.

Table 3: worst case delay and number of logic gates of lower part of (a) Conventional (b) proposed architectures.

<table>
<thead>
<tr>
<th>Worst Case Delay(ns)</th>
<th>Number of logic gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carry propagate adder</td>
<td>19.434</td>
</tr>
<tr>
<td>Complementer</td>
<td>52.308</td>
</tr>
<tr>
<td>Normalizer</td>
<td>17.468</td>
</tr>
<tr>
<td>Rounder</td>
<td>22.971</td>
</tr>
</tbody>
</table>

V. CONCLUSION

Many approaches are developed on floating-point fused multiply add unit to decrease its latency. The greatest deviation from the original IBM RS/6000 architecture comes from a paper by Lang and Bruguera on a reduced latency fused multiply add. Theirs proposal claims to achieve a significant increase in fused multiply add unit performance by the combination of the addition and rounding stage into one block. The paper claims an estimated 15-25% reduction in latency as compared to a standard fused multiply add. The main objective of our work is to with making some changes in the architecture to facilitate the implementation and on the other hand do not affect the performance. The change is to take the inputs to sign detection and LZA blocks from 3:2 CSA instead of the multiplier outputs and the aligned addend A. This is more efficient because using three inputs (multiplier output and aligned addend A) to sign detection module makes the need of a carry propagate adder is obligatory to add multiplier output before making comparison. This change will not increase the overall delay as the delay of CSA already exists in the critical path. Area and timing information for each design approach and algorithm is analyzed. It is clear that the proposed architecture achieves a delay improvement about 25.5% as compared to the basic architecture. The increase of area in the proposed architecture is about 6.2% which is not a big matter.

References