Design of Low-Cost High-performance Floating-point Fused Multiply-Add with Reduced Power

Zichu Qi[1,2,3], Qi Guo[1,2], Ge Zhang[1,3], Xiangku Li[1,3,] Weiwu Hu[1,3]

1. Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

2. Graduate University of Chinese Academy of Sciences (GUCAS), Beijing, China

3. Loongson Technologies Corporation Limited

{qizichu, guoqi, gzhang, lixiangku, hww}@ict.ac.cn

Abstract

This paper presents a floating-point fused multiply-add (FMA) unit with low-cost and low power techniques. To improve the performance, two single-precision operations can be performed concurrently with one double-precision datapath, 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. A modified dual-path algorithm is proposed by classifying the exponent difference into three cases and implementing them with CLOSE and FAR paths, which can reduce latency and facilitate lowering power consumption by enabling only one of the two paths. In addition, in case of FADD instructions, the multiplier in the first stage is bypassed and kept in stable mode, which can significantly improve FADD instruction performance and lower power consumption. The overall FMA unit has a latency of 4 cycles while the FADD operation has 3 cycles. Each cycle has a time delay of about 0.66ns in the ST 65nm CMOS technology. Compared with the conventional double-precision FMA, about 13% delay is reduced and about 22% area is increased, which is acceptable since two single-precision results can be generated simultaneously.

1. Introduction

The floating-point fused multiply-add (FMA) unit has become one of the key features of recent commercial general-purposed processors [1, 2. 3, 4, 5, 6]. It executes the floating-point multiply-add, A×B+C, as a single instruction with only one rounding error. Besides the increased accuracy, this unit is designed to reduce the latency and hardware cost by sharing several components. Moreover, this unit can support standard floating-point addition (FADD) and floating-point multiply (FMUL) by making B＝1 or C=0.

Since the first FMA was introduced in the IBM RS/6000 [1], several FMA architectures have been proposed to reduce the delay. One of such cases has been proposed in [7]. It anticipates the normalization before the final addition and combines the final addition with rounding to reduce the number of serial stages required by a conventional FMA. To reduce the latency of FADD instructions, the design in [7] is improved by bypassing the multiplier and by introducing a dual-path scheme [8], which is similar to the dual-path FADD architecture [12]. The FMA architecture presented in [9] identifies five multi-paths and implemented them with two parallel hardware paths. To further reduce the dual-path complexity and avoid the large alignment shifter in the FAR path, a three-path double-precision FMA is presented in [10], which can reduce delay of about 11.7%, but with about 39% additional area compared with the conventional FMA.

Though many recent works focus on decreasing the delay of the FMA, several works have been presented to improve the throughput and decrease the power consumption. Based on the conventional FMA, a FMA architecture supporting single-precision, double-precision and parallel single-precision is introduced with about 18% extra hardware and 9% increment in delay compared with the conventional FMA [11]. This FMA exhibits higher performance for single-precision than for double-precision. The three-stage Concordia FMA in [13] is presented to reduce the power and latency at the cost of reduced precision and increased area. Power consumption is also analyzed in [10], but no methods are presented.

The implementation of low cost high performance FMA is a difficult task. Most of the proposed designs reduce the latency at a high cost of area, which further aggravates the power consumption. The performance reduction of FADD instructions is also a notable problem in some floating-point addition intensive applications. In this paper, a four-stage fully pipelined FMA is proposed by carefully trading-off the area, latency, throughput and power. To provide details on the design costs of the proposed FMA, comparison results with the conventional FMA is presented. All circuits have been designed and implemented with ST 65nm CMOS technology to provide a realistic and fair comparison on the performance and area.

2. Conventional FMA unit

Though many FMA architectures have been proposed and several improvements have been attained, the basic architecture remains invariant. Since most of FMA as in [7, 8, 9, 10, 11] have been compared with the conventional FMA, we first give a brief introduction of it. Figure 1 illustrates the architecture of the conventional FMA. It consists of several steps and can be divided into three stages:

1. Multiply the two input mantissas to produce a product 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

Supported by the National Basic Research 973 Program of China（No.2005CB321600),the National High Technology Development 863 Program of China(No.2008AA110901， No.2009AA01Z125) and the National Natural Science Foundation of China(No.60803029).

1063-9667/10 $26.00 © 2010 IEEE
DOI 10.1109/VLSI.Design.2010.41

is 161-bit.

2. Combine the addend and the product with a 3:2 CSA (carry save adder). Then a 161-bit carry-save adder is used to produce the intermediate sum. In parallel with the addition, leading-zero anticipator (LZA) is used to determine the normalize shift amount. If the addition result is negative, complement is performed.
3. At the last stage, normalization and rounding are performed to get the final result.

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

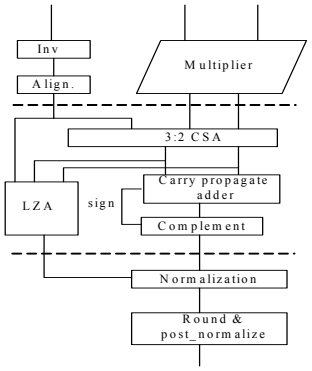

Figure 1. Conventional FMA unit

3. Overview of proposed FMA

The objective of our proposed FMA is to reduce the delay and improve the performance while maintaining relative low area cost and low power consumption. To achieve such complicated goals, several novel schemes are implemented.

- To reduce the timing delay, a modified dual-path algorithm is implemented in which the paths are categorized into CLOSE and FAR paths. The definition of datapaths is similar to the algorithm in [8], but the proposed algorithm has less delay by using smaller shifters and adders. In the CLOSE path, instead of using a 161-bit normalization shifter as in [8], a 57-bit normalization shifter is used. To shorten the FAR path delay which is most critical in [8], one 55-bit and one 31-bit alignment shifters are implemented and divided into two stages. The shifting operator is either A×B or C and two sub-paths are further divided according to the sign of exponent differences in the FAR path. The algorithm proposes use small adders and small shifters, which is then simpler than in [8] and has less area than in [10].
- To reduce the FADD cycle time, multiplier in the first pipeline-stage is bypassed. If a FMA instruction has just completed its first stage and a FADD 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. In this paper, we propose a conflict resolution method, with which the FMA has no pipeline stopping and operations can be started at every cycle. Therefore a throughput of one-instruction one-cycle can be achieved.
- Since the FMA usually occupies a large percentage of area in the FPU (floating-point unit), power consumption has become a key issue. Based on the idea of activating only one of the dual paths, switching power can be reduced. Clock gating cells are then inserted to control the clocks in CLOSE and FAR path separately. Furthermore, clocking gating cells are also used to control the input of the multiplier. When operating FADD instructions, the input of the multiplier can be kept in stable mode and thus the whole multiplier has no switching activity. Since the multiplier in the first stage contributes to the major part of total FMA area, which is about 40% from our synthesis results, the power can be reduced significantly.
- To improve the performance, the proposed FMA supports parallel single-precision instructions, in which two single-precision operations are executed concurrently within one instruction. To lower the area cost we maximally reuse the double-precision unit. 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 FMA, the shifter, adder, and multiplier are usually the main parts of the total area. We then modify these units and divided them into two parts. Each part is for one single-precision, and the two parts together are for one double-precision.

The proposed FMA is illustrated in Figure 2. The major steps are described in the following while detail implementations will be discussed in the next section.

The multiplier is divided into two stages to support both the double-precision and the two single-precision operations. In the first stage, Booth-encoding and two CSA trees are used to produce four intermediate multiply products. In the second stage the 4:2 CSA is used to compress the four products from the two CSA trees. When performing FADD instructions, multiplication in the first stage is bypassed. Therefore, operand C and operand A are directly transferred to the second stage.

In parallel with the multiplication, the exponent difference is calculated, which is d=exp(C) – (exp(A)+exp(B)-1023). Taking into account that the multiplication can produce an overflow, the CLOSE path is classified with the exponent difference d=0,1,2,-1 for effective multiply-subtraction. The FAR path is used for the remaining cases. In CLOSE path, only the full length normalization can occur, and in the FAR path only the full length alignment can occur, which can avoid having two shifters in the critical paths and thus reduce the latency.

In the CLOSE path, only a 3-bit alignment shifter and a 58-bit normalization shifter are employed in the critical path. Two dual-adders, one is 29-bit and the other is 28-bit, are implemented to calculate the sum and sum+1. In case of two single-precision, each of them is for one single-precision operation, and together of them are for double-precision operation. We use two adders instead of two HAs (half adder) after 3:2 CSA to avoid the two normalization shifter for carry and sum words. Since shifters have much larger area than the two adders from our experimental results, area costs are reduced. Moreover, the results of two dual-adders are also used to detect the sign of the result. Then, the sign detection unit is removed. The latency of the dual-adders is almost the same with the sign detection unit in [7, 8]. Therefore, there is almost no time impact on the cycle time. In parallel with the two dual-adders, the 57-bit LZA is implemented to anticipate the normalization shift amount. If all bits of the higher 57 bits are ‘0’, the lower 51 bits for double-precision or lower 22 bits from the multiplier are then normalized and are selected. Then rounding and normalization are performed by right or left shifting of 1-bit or with no shifting.

In the FAR path, the alignment is performed fully depending on the input operators, which is further classified

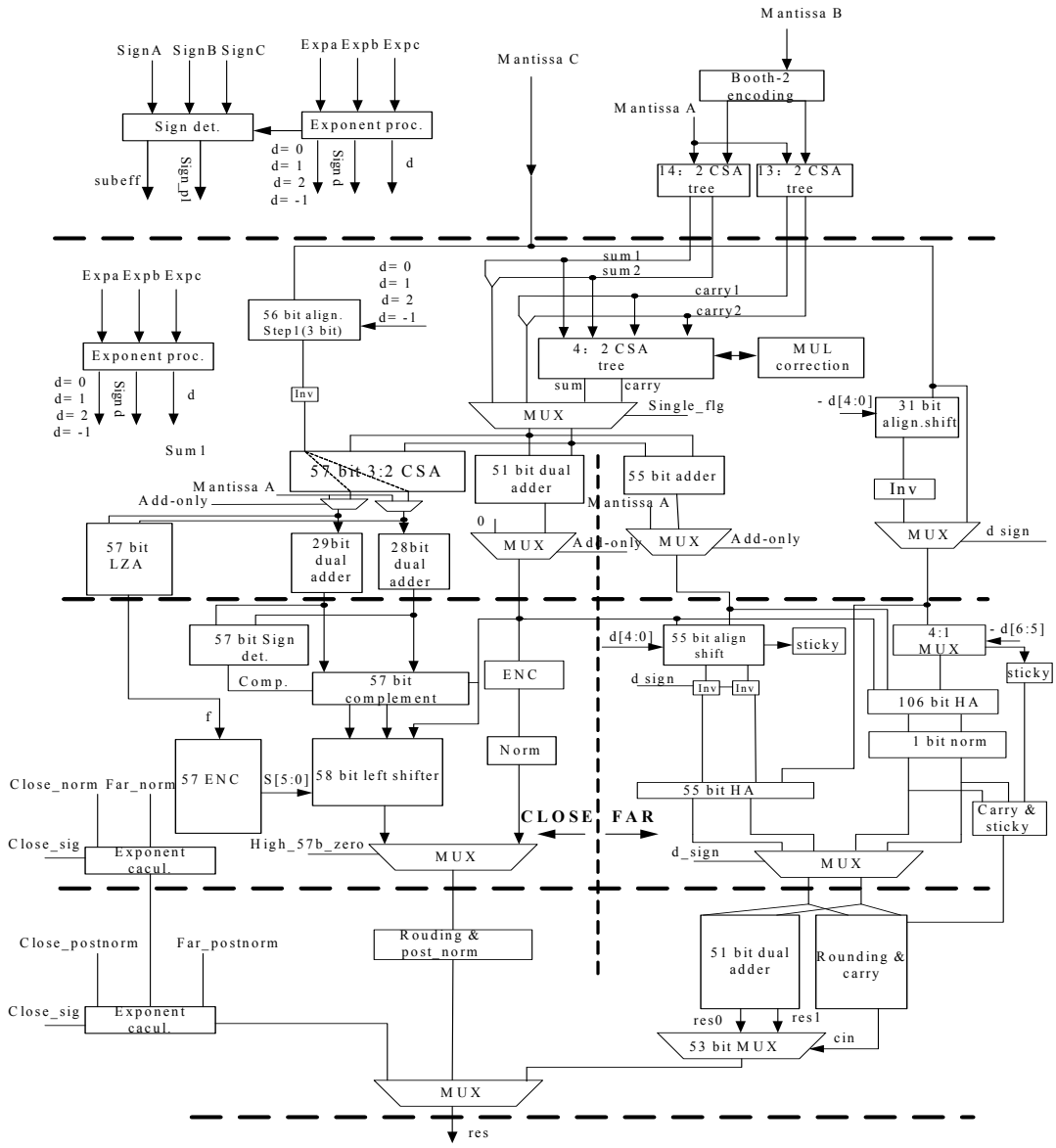

Figure 2. General scheme of the proposed FMA

into two sub-paths according to the sign of the exponent difference. In case of d ≤0, operand C is right aligned with respect of the A×B. If the exponent difference is larger than 106 in double-precision or 48 in single-precision, only 106-bit or 48-bit alignment shift is performed. Two-stage alignments are performed on operand C to balance the delay of second stage and third stage. The first alignment stage for operand C is a 31-bit alignment shifter controlled by the lowers 5 bits of the negative exponent difference, and the second alignment stage is a 4:1 MUX controlled by the higher 2 bits. The alignment result of C is then passed to a 106-bit HA. If the higher integer bit of multiplication is 0, 1-bit normalization is performed to guarantee that the MSB bit is 1. So, only one 1-bit right or left shifting are needed in the post normalization and the rounding stage. In case of d >0, the multiplier results are right shift with respect of the operand C, the maximum shift amount is 55-bit in double-precision or 26-bit in single-precision. If the exponent difference is larger than 55 or 26, all bits of the A×B are treated as sticky bits. After the alignment shifter, the result of A×B is inverted, and the 55-bit HA is performed. Because the result of the FAR path is always positive, no complement stage is performed. Compared with architecture in [8], only a 55-bit alignment shifter is located in the critical path, then latency is reduced significantly. The two cases of the FAR path of our algorithm share the same rounding stage, and employ less bit-width units than the algorithm in [9]. Moreover, to reduce the shifting amount, the multiplier results are directly added with the 51-bit and 55-bit adders in the second stage, and only one shifter is performed to align the result of A×B when d>0.

In the proposed FMA, the shifter, LZA, adders and HA adders are all divided into two parts for supporting parallel single-precision, which will be described into details in the next section.

4. Detail implementation

4.1 Multiplier

The multiplier in the proposed FMA is able to perform either one 53-bit or two 24-bit multiplications. The multiplier uses the radix-4 Booth-encoding, with which the 53 bits can obtain 27 products and 24 bits can obtain 13 products. Only one Booth-encoding is used to get the products of double-precision and parallel single-precision. For parallel single-precision, 4-bit zero is inserted between the two 24-bit mantissa to form a 53-bit mantissa, which will make the partial product-13 zero, as shown in Figure 3a. We use two CSA trees. One is the 14:2 CSA tree to compress the partial product 0-13, and the other is the 13:2 CSA tree to compress the partial products 14-26. Each CSA tree can produce two products for one single-precision, respectively. For double-precision, four products from the two CSA need further compression with 4:2 CSA as shown in Figure 3b. Using these methods, two single-precision multiplications can be performed in parallel with one double-precision multiplication unit.

The multiplier results are directly added with two adders. One is the 51-bit adder, and the other is 55-bit adder, the results of them together are for one double-precision. In case of two single-precision, the 51-bit adder produces the first single-precision multiplication result, while the 55-bit adder generates the second single-precision multiplication results. These two adder are bypassed with 0 and mantissa A in the second stage in case of FADD instructions (add_only=1 in Figure 2). The complement of multiplication result of lower 51 bits is also needed in CLOSE path in case of the result of 3:2 CSA being negative. So the dual 51-bit adder is implemented in the design. One for directly addition, the other is for the complement addition. Using the addition to get the multiplication results directly helps reducing the area of the shifters, which is wire-dominated and has larger area than the adder.

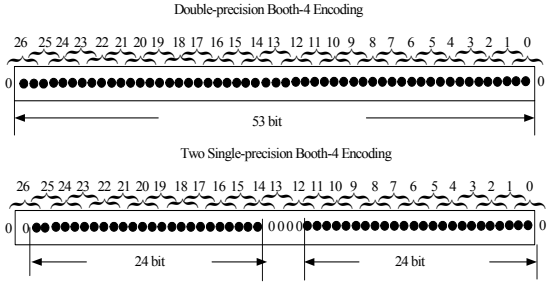

Figure 3a. Booth-4 encoding

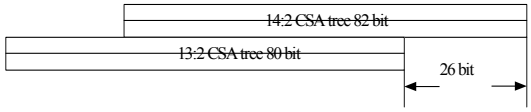

Figure 3b. 106-bit 4:2 CSA

4.2 Sign and exponent processing

Two bits are used to deal with the sign processing. The higher bit is used for the sign of one single-precision, the lower bit is used for the sign of the double-precision or another single-precision. In the CLOSE path, the mantissa of C is always inverted, then the sign is temporarily determined by Sign(A×B). If the final addition results are negative, the sign should be inverted. In the FAR path, A or B×C are inverted depending on the exponent difference, and the results are always positive. Therefore, the sign of the result is either Sign(A×B) in case of d≤0 or Sign(C) in case of d >0.

The exponents of the three operands are processed to obtain the control signals and the shift amounts for the alignment. In order to reduce the latency of the floating-point addition, the first stage is bypassed. So, for FADD instructions the exponent processing logic should be replicated in the second stage. In case of FMA instructions, the exponent processing is done in parallel with the multiplier. While in case of FADD instructions, the exponent processing is done in the beginning of the second-stage.

In order to support parallel-single precision, one single-precision exponent processing unit is added in the datapath, while the other single-precision shares the double-precision exponent unit. Since the exponent processing unit is much small, the added area is much small.

4.3 Alignment and 2's complement

We describe the alignment for the CLOSE path and FAR path separately as follows:

- The alignment shift of C in the CLOSE path is implemented as maximum 1-bit right shift or 2-bit left shift with respect of the result of A×B. In case of d=0, no shift is performed. If d=-1, 1-bit right shift is performed. If d=1,2, 1-bit or 2-bit left shift is performed. Figure 4a shows the word lengths of the shifter input, and shows how to support two single-precision shifting in CLOSE path. For the double-precision the shifted result is 56 bits, and for the single-precision the shifted result is 27 bits. For parallel-single precision, 5-bit '0's are inserted between the two single-precision before shifting, as shown in Figure 4a. Thus one alignment shifter is enough for the double-precision and two single-precision. The alignment of C is inverted and then sent to the 3:2 CSA. Since the result of CLOSE path could be negative, an additional sign bit is added to the shifted results, which leads to a 57 bit 3:2 CSA for double precision or two single-precision. In Figure 4a, We also add 2-bit 0 between the two higher 26 bits of multiplier products of two single-precision according to the shifted C. An additional 1 has to be added to the LSB (Least Significant Bit) of alignment C_{inv} to complete 2's complement. After 3:2 CSA, there will be an empty slot in the LSB of the carry word, the additional 1 is then added at LSB of the carry word for the double-precision and one single-precision. For another single-precision, two '1's are added at the first slot bit (FSB) between the two 26-bit single-precision multiplication products as shown in Figure 4a. In this way, it is the same as a '1' is added at the LSB of the 27-bit shifted result.
- In the FAR path, C or A×B is aligned depending on the sign of the exponent difference. The shift amount is given by d or –d. In case of d>0, A×B should be right shifted, and the maximum shift amount is 55 bits for double-precision and 26 bits for single-precision. Two extra bits are added to LSB of C for correct rounding when A×B shift amount is large than 55 or 26. If d≤0, C should be right shift with –d, and the maximum shift amount is 106-bit for double-precision and 48-bit for single-precision. Figure 4b shows the alignment shifter in FAR path for double-precision. Two shifters are used for the alignment. One is right shifter for C, and the other is right shifter for A×B. To lower latency of critical path in the second stage, the full 106-bit alignment of C is separated into two steps. The first one is a right shifter with maximum 31-bit shift amount, and the other is the 4 to 1 multiplexers. During shifting, the sticky bit is

calculated. To complete the 2's complement of C or A×B, an additional 1 is added in the empty slot of the carry word coming from the two HAs.

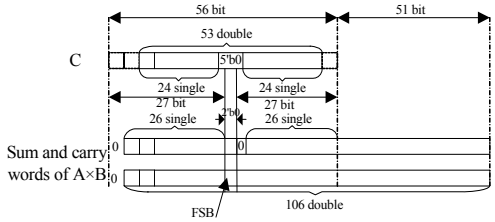

Figure 4a. Alignment of CLOSE path

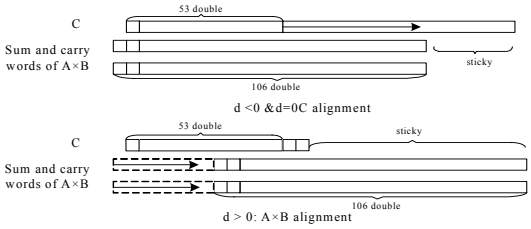

Figure 4b. Alignment of FAR path

To lower area costs, alignment shifters in the FAR path are also designed to support the two single-precision operations. To achieve the sharing purposes, shifters are divided into two parts, the higher part and the lower part. For example, the right alignment shift amount for C is 31 bits in the second stage, which results in an 84-bit aligned value for double-precision. For the parallel single-precision, two alignment shifters with maximum 15-bit shift amount are needed, in which the aligned values are 39 bits. The 84-bit alignment shifter is then divided into higher 42-bit and lower 42-bit alignment. In case of double-precision, the shifted in data of the lower part comes from the shifted out data of the higher part. While in single-precision the shifted in data of the lower part is zero. Figure 5 shows the two parts alignment shifter, with which the two single-precision is completed with the same unit as the double-precision. To achieve this, 18-bit 0s are inserted between the two single-precision to form a 84-bit shift in value. After alignment, the higher 39 bits from the two parts of the shifter are the results of two single-precision. To reduce the area costs, the two part shifter techniques are also employed for A×B alignment in case of d>0.

4.4 LZA and normalization

Though we can get the exact shift amount from the output of the adders, it may cause some additional delay in the CLOSE path in the third stage. So we choose to use LZA to anticipate the first leading bit in parallel with the two dual adders. The method mentioned in [14] is exploited to detect the position of the leading bit, and a 57 bit encoding is employed to calculate the shift amount. The LZA in [14] could cause 1 bit anticipator error, which makes the shift amount is 1-bit less than the correct value. Take into account that the carry bit from the lower 51 bit is not concerned in the LZA, another 1-bit error can occur. In this case, the shift amount is 1-bit larger than the correct one. The uncertainty of left shift and right shift will be corrected in the rounding and the post normalization stage.

A normalization shifter of 58 bits is implemented. As described with the 31-bit alignment shifter in section 4.3, the 58-bit normalization shifter is also divided into two parts to support the parallel single-precision operation. The maximum shift amount is 57 bit, and one extra bit is added to reserve the shifted out bit when the shift amount is 1-bit larger than the correct one. The shift in value of the normalization is the lower 51-bit in double-precision or the lower 22-bit in the parallel single-precision. The lower 22 bits of the two single-precision come from the 51-bit adder and 55-bit adder, respectively. If the shift amount is larger than 57 or 28, only the lower 51 bits for double-precision and lower 22 bits for each single-precision are normalized and used.

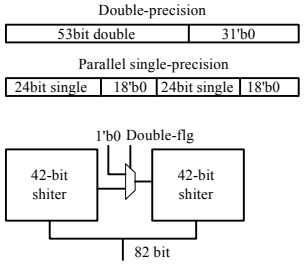

Figure 5. Two part alignment shifter

4.5 Final addition and rounding

Addition and rounding are separated for the CLOSE path and FAR path from the implementation perspective, which is suitable for the purpose of reducing latency. They are described separately in the following.

- For the CLOSE path, there are three cases for the rounding. As explained in section 4.4, LZA can cause two bit uncertainty. Thus, the MSB of the 58-bit result from the third stage may have 0-bit zero, 1-bit zero or 2-bit zero. So the rounding and post normalization are performed based on such three possibilities, and the final result is select from the three cases. An increment adder is implemented to get the result after rounding, and the carry from the lower bit is used to select the final result.
- For the FAR path, one additional bit is extended from the 55-bit result of the third stage to reserve the overflow bit of addition. Since the final result is attained from 1-bit left shift or 1-bit right shift or no shift, the higher 51 bits (bit55-bit5) are selected to put into the final dual-adder, which calculates the sum and sum+1. The carry from the lower 5-bit addition decides whether sum0 or sum1 should be selected. The lower 5 bits are added together with the carry from the rounding part, the sticky bit (only in the RI mode) and the rounding bit (in RI and RN mode). The R (rounding) bit is in position bit1 when no shift is needed. When left shift is needed, the R bit is in the bit2. If right shift is needed, the R bit is in the bit0. The lower 5-bit together with the HA adder in the third stage guarantee that only 1 bit carry is given out to the higher 51-bit.
- To support two single-precision operations, one rounding and normalization unit of single-precision is added. The other single-precision shares the same unit with the double-precision.

4.6 Resolve the hardware conflict of second-stage

As discussed, the FADD instructions will bypass the first stage and directly come into the second stage to reduce operating cycles. This could cause a hardware conflict in the second stage if a FMA instruction has just finished its first

stage and a FADD instruction is directly coming in. To resolve the hardware conflict of the second stage, FADD instruction must have the precedence of using the second stage. An enable signal which is the inversion of the signal of the 'add_only' (shown in Figure 6 and Figure 2) is then used to control whether to update the data in the first stage or not. If a FADD instruction comes in, the enable signal keeps low. Then in the next cycle, the FMA instruction is reserved in the first stage registers, and the FADD instruction can continue. If the following instructions are still FADD, the FMA will be kept in the first stage all the time. Until the next instruction is FMA or FMUL, current FMA instruction can continue. Therefore, the FMA unit has no pipeline bubble, and can output one operating result one cycle. Thus, the throughput of one instruction one cycle is achieved. If the FMA has the precedence of using the second stage, outside logics should stop providing the instructions for one cycle to let FMA finish it's second stage and no instruction will be allowed in, which will cause performance degrading. Figure 6 illustrates the algorithm.

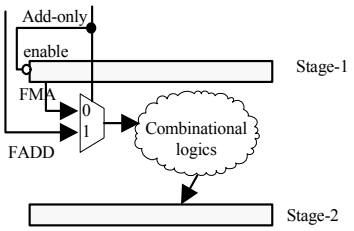

Figure 6. Hardware conflict resolution

5. Results

The FMA unit is described with verilog RTL and synthesized with the ST 65nm CMOS standard-cell library. Table 1 gives the latency of each cycle. From the experimental result, we can see each stage has almost the same latency.

Table 1. Synthesized Results

	stage1	stage2	stage3	stage4
Delay(ns)	0.69	0.66	0.65	0.67
Area(um^2)	66227	43723	27687	13738
% of total area	43.6%	28.8%	18.2%	9.1%

Table 2 compares the conventional FMA and the proposed FMA in the categories of latency and area. To get a reasonable comparing result, the internal pipeline registers are removed to obtain a total latency and area. The proposed FMA shows an approximate 13% decrease in latency with an approximated 22 % increment in area. The area increment is acceptable, because the performance of single-precision is doubled. Moreover, it has lower area costs than the three path algorithm in [9], which is about 39% and only supports the double-precision operation.

Table 2. Comparison result of FMA

Design	Latency(ns)	Area(um^2)
Convention FMA	2.38	107697
Proposed FMA	2.06	131524
Δ	13.44% decrease	22.12% increase

6. Conclusion

This paper proposes a four stage low-cost FMA design, which is capable of performing one double-precision or two single-precision operations in parallel. In order to support two single-precision in parallel with lower area cost, the traditional double-precision multiplier, shifter and adder, which contribute the major area of the FMA, are divided into two parts. Furthermore, a modified dual-path algorithm is implemented to reduce the cycle time by classifying the exponent difference into three cases. Moreover, to reduce the cycle time of performing FADD instructions, the multiplier in the first stage is bypassed, and the resolution of hardware conflict is proposed to achieve a throughput of one instruction one cycle. The proposed FMA is also a lower power design, in which multiple clock gating cells are inserted to enable only one of the two paths and keep the multiplier stable in case of FADD instructions Compared with the conventional double-precision FMA, 13% delay is reduced with about 22% area increment. The increased area is mainly used to support parallel single-precision and reduce latency of FADD instructions.

References

[1] R.K. Montoye, E. Hokenek and S.L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit", IBM journal of Research & Development, Vol. 34, pp. 59-70, 1990.

[2] E. Hokenek, R. Montoye and P.W. Cook, "Second-Generation RISC Floating point with Multiply-Add Fused", IEEE Journal of Solid-State Circuits, Vol. 25, pp. 1207-1213, 1990.

[3] R. Jessani and C. Olson, "The floating-point Unit of the PowerPC 603e", IBM Journal of Research and Development, Vol. 40, pp. 559-566, 1996.

[4] A. Kumar, "The HP PA-8000 RISC CPU", IEEE Micro Magazine, Vol. 17, Issue 2, pp. 27-32, April, 1997

[5] K.C. Yeager, " The MIPS R10000 superscalar microprocessor", IEEE Micro Magazine, Vol. 16, No.2, pp. 28-40, March, 1996.

[6] B.Grer, J. Harrison, G.. Henry, W. Li and P. Tang, "Scientific Computing on the Itannium Processor", Proceedings of the ACM/IEEE SC2001 conference, pp. 1-8, 2001.

[7] T. Lang and J. D. Bruguera, "Floating-point Fused Multiply-Add with Reduced Latency", IEEE Transactions on Computers, Vol. 53, pp. 988-1003, 2004.

[8] T. Lang and J. D. Bruguera, "Floating-point Fused Multiply-Add: Reduced Latency for Floating-point Addition", Proceedings of the 17th IEEE Symposium on Computer Arithmetic, pp. 42-51, 2005.

[9] P.-M. Seidel, "Multiple Path IEEE Floating-Point Fused Multiply-Add", Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, pp. 1359-1362, 2003.

[10] E. Quinnell, E. E. Swartzlander, Jr, C. Lemonds, "Floating-Point Fused Multiply-Add Architectures", Proceeding of the Fortieth Asilomar Conference on Signals, Systems, and Computers, 2007.

[11] L. Huang, L. Shen, K. Dai, Z. Wang, "A New Architecture for Multiple-Precision Floating-Point Multiply-Add Fused Unit Design", Proceeding of the 18th IEEE Symposium on Computer Arithmetic (ARITH18), 2007.

[12] P.-M.Seidel, G. Even, "On the Design of Fast IEEE Floating-Point Adders", Proceeding of the 15th IEEE Symposium on Computer Arithmetic (ARITH15), pp. 184-194. 2001.

[13] R.V.K. Pillai, S.Y.A. Shah, A.J. Al-Khalili, and D.Al-Khalili, "Low Power Floating Point MAFs − A Comparative Study", sixth International Symposium on Signal Processing and its Applications, Vol. 1, pp. 284-287, August, 2001.

[14] M.S. Schmookler and K.J. Nowka, "Leading Zero Anticipation and Detection − A comparison of Methods", Proceeding of 15th IEEE Symposium on Computer Arithmetic (ARITH15), pp. 7-12, 2001.