Indian Journal of Science and Technology, Vol 9(5), DOI: 10.17485/ijst/2016/v9i5/87159, February 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

An Efficient Single Precision Floating Point Multiplier Architecture based on Classical Recoding Algorithm

J. Jean Jenifer Nesam* and Sivanantham Sathasivam

School of Electronics Engineering, VIT University, Vellore - 632014, Tamil Nadu, India; jean.jenifernesam@vit.ac.in, ssivanantham@vit.ac.in.

Abstract

Background: Floating Point (FP) multiplication has found its importance in many microprocessors but it is very difficult to implement on FPGA because of its complicated internal computation. **Methods:** We investigate partial product (PP) reduced FP multiplication based on Radix-4 Booth Encoded Algorithm (BEA). Radix-4 BEA reduces the number of PP generation by half. PP reduction performed in three steps such as Grouping bits (3-bit for each group), Encode the group and PP calculation for each group. **Findings:** The investigation results show that Radix-4 BEA works perfectly on signed multiplication and unsigned (FP mantissa) multiplication needs some extra consideration. Radix-4 BEA grouping multiplier bits need overlapping one bit from both adjacent group that limits block and parallel processing. 2's complement calculation and sign extension essential for PP generation that increases the resource utilization. In this paper, 32 bit improved FP multiplication based on classical recoding and parallel processing method is proposed. Classical recoding reduces PP generation by half without overlapping, sign extension and 2's complement. 24 bit mantissa split into blocks (8 bit each) and each block is recoded using classical recoding algorithm and all blocks are performed in parallel. **Applications:** The experimental results show that our proposed design runs with high frequency with less resource utilization and suitable for signal processing applications.

Keywords: Block Multiplication, Classical Recoding, Floating Point Multiplier, Parallel Processing, Single-Precision

1. Introduction

Floating point arithmetic has been used in most of the DSP processor and scientific calculation because of its wide range and accuracy[1]. The implementation of Floating-Point (FP) on FPGA (Field Programmable Gate Array) has some limitation in terms of speed and area. Related works on FP multiplier based on Booth[2-4] and other architectures [5-8]shows that lot of changes need algorithm itself for unsigned FP mantissa multiplication. Booth originally developed for fixed point signed multiplication. Booth performs recoding on multiplier based on 2's complement form and also need sign extension for getting perfect ` result. FP does not need this consideration because mantissa of any FP format is an unsigned number format.

Investigation works provide knowledge about Booth on FP multiplication[2] which needs some consideration like unsigned to signed number format conversion. These changes need extra calculation makes FP multiplication more complex. Above all there is no guarantee for perfect result.

The floating point multiplication actually has three different calculations

1. Sign calculation
2. Exponent calculation
3. Mantissa calculation

The difficult part in above three is mantissa calculation. We consider single precision floating point number for both algorithms whereas mantissa has 24 bit (including hidden bit). Since the FP number has separate bit

*Author for correspondence

for sign, we do not worry about the signed mantissa multiplication, because the mantissa of any FP number always an unsigned number. The modified Radix-4 Booth multiplication is suitable for signed number multiplication. It takes MSB as sign bit. We all know that MSB of '1' always represent negative number. Since the MSB of any FP number (hidden bit) is always '1', that not mean the mantissa is negative number. The mantissa of FP number is always a positive. Hence here itself the modified Radix-4 Booth encoding fails to provide perfect answer with their original behavior.

Booth encoding used for unsigned multiplication needs some extra added bit and calculation[2]. However that not matches for all FP number. The main focused thing on using booth encoding is to reduce the number of partial product to half of its actual strength. The new proposed scheme also provide the same advantage whereas the booth encoding needs overlapping bit for proper encoding the proposed one does not need that.

In this paper, we have illustrated the novel recoding suitable for unsigned FP multiplication without any need of modification. Instead of pipelined architecture we design architecture with divide the operands and parallel processing method[9]. Our design performance shows better performance in terms of area, power and speed on FPGA family devices.

2. General Floating Point Multiplier Architecture

2.1 Floating point (FP) Number format (IEEE 754 Standard)

The IEEE 754 standard FP number format is composed of a sign bit (1 bit length), exponent and mantissa[3,10]. They usually using 32 bit for single precision, 64 bit for double precision. Single precision FP word format is given in Figure 1.

Each word represented as,

$$X = S \times 2^{efp-bias} \times F.m_{fp} \quad (1)$$

F in the above equation is the hidden bit always 1 for normalized number. The term biasing ensures the exponent always positive and the bias value is equal to (2^{efp-1}-1). Different types of representation used to indicate FP number as zero, infinites, NaN (Not an Number) and normalized and de normalized number is tabulated in Table 1. This differentiation is based on exponent and mantissa values. FP operation includes the number type identification, Exception handling, Normalize the number if required. All above defined criteria is known as pre normalization[11]. After pre normalization, calculation unit performs FP multiplication that includes sign calculation, Exponent adder, and Mantissa multiplication. Obtaining all above the results is sent to post normalization unit.

The basic steps involved in floating point multiplication are shown in Figure 2. The post normalization handles the normalization the number if required, Rounding, Result exception handling, Format the result to fit the specified format. In this paper all the calculation except the mantissa multiplication is performed in regular manner.

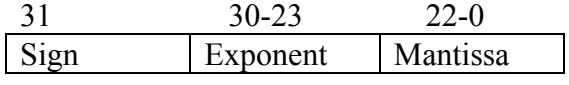

Figure 1. FP number format.

3. Modified Radix-4 Booth Recoding

The technique used to reduce the partial product by half is radix-4 booth recoding. Instead of multiplying the multiplier by 1 or 0, booth recoding multiply it by ±1,±2,±0 to obtain the same result. The booth recoding considers three blocks of bits for recoding and each block is overlap with previous block with one bit. We add '0' as LSB that include the block which is considered as first block. Booth recoding is mainly used for signed

Table 1. Number type based on exponent and mantissa value

Number type	Exponent value	Mantissa value	F(hidden bit	Number value
Zero	0	0	-	±0
Infinites	2^{efp-1}	0	-	±∞
Denormalized	0	≠0	0	As in Equation (1)
Normalized	1 to 2^{efp}-2	-	1	As in Equation (1)
NaN	2^{efp-1}	≠0	-	Not number

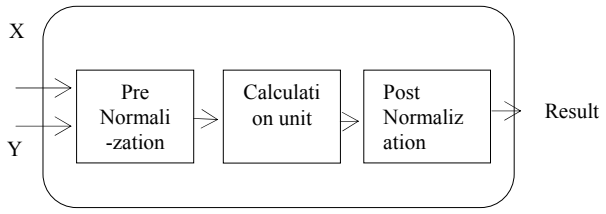

Figure 2. Basic steps involved in FP multiplication.

number multiplication and the MSB bit of each block acts like a sign bit. We found little complexity in booth recoding because -1,-2 recoding needs 2's complement of multiplier and 2's complement with '0' adding as a LSB respectively. Extra calculation needed and also booth is suitable for sign multiplication not for unsigned FP number multiplication. Recoding needs 8 different combination (000.001,010,011,100,101,110,111) since each group compose of 3 bits and five different way of recoding needed with +1,-1,+2,-2,0. The term sign extension usually used in signed magnitude multiplication. Table 2 gives the Booth encoding of radix-4 algorithm. Each block is overlapped with previous block has a limitation that we cannot split the operands and processing it in parallel. All these recoding, 2 bit left shifting and sign consideration must need in booth recoding. Our proposed recoding only uses four different types of combination (00,01,10,11) and no overlapping and no need for sign consideration. Radix-4 booth recoding is best suitable for fixed point multipliers with sign consideration. Since the FP has separate sign bit and the mantissa is an unsigned number, our proposed recoding method is best suitable for FP multiplier.

Table 2. Radix-4 booth recoding

Block	Partial product
000	0
001	1 * multiplicand
010	1 * multiplicand
011	2 * multiplicand
100	–2 * multiplicand
101	–1 * multiplicand
110	–1 * multiplicand
111	0

4. Integration of Topologies

CRSOPP stands for Classical Recoding Split Operand and Parallel Processing. The performance of CRSOPP algorithm is explained in this section. Since the mantissa of FP number is always positive, there is no need for sign consideration. New Unsigned floating point multiplicand recoding method is perfect suit for FP number multiplication compared to Booth recoding. The proposed method takes two consecutive numbers for recoding. No overlap needed. This method also reduces the partial product by half. Proposed recoding explained in Table 3. If the multiplier and multiplicand is n bit length the recoding as follows

It takes two consecutive numbers for recoding. No overlapping needed[12]. New recoding method reduces the partial product by the factor of two and also has some added advantage compare to booth recoding

- No sign extension needed
- Neglecting overlapping in grouping recode bit reduces the possible combination only by four (00, 01, 10, 11)
- No overlapping needed
- No need for 2's complement calculation

The 8x8 multiplier needs 8 partial products and addition of the PP's produce more carry propagation delay. The proposed architecture reduces the PP by 4 and split the 8 bit operands into two 4 bit blocks and performs multiplication and addition in parallel[13]. In this paper we take single precision multiplication that uses 24x24 bit multiplication. 24x24 multiplications generate 24 PP and adding PP produce more carry propagation delay. However usage of booth recoding reduces the partial product by half (13 PP one PP extra needed for unsigned conversion) the adder have to add all PP gives more delay. Proposed CRSOPP method is explained in Figure 4.

Table 3. Proposed Decoding

Possible combination	Partial product
00	String of zeros equal to n+1 bit length
01	'0' is concatenated as MSB with multiplicand
10	'0' is concatenated as LSB with multiplicand
11(01+10)	Sum of recoding 01 and recoding 10

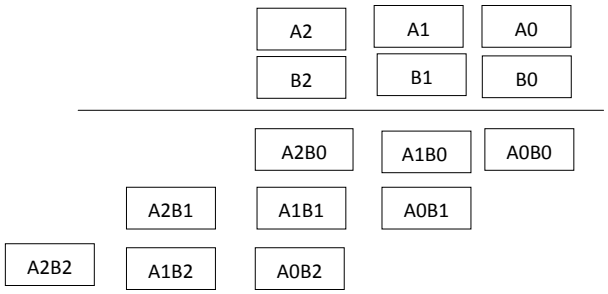

Figure 3. Split operands and parallel processing multiplication (SOPP).

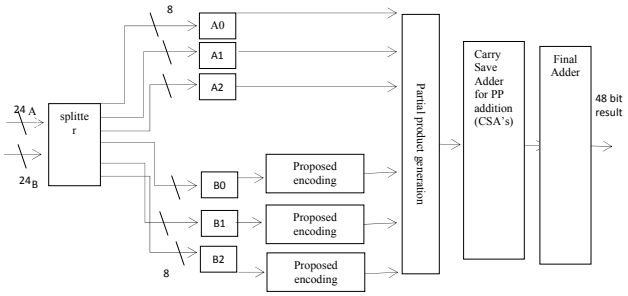

Figure 4. CRSOPP Architecture.

CRSOPP method includes the following steps

Step 1: split the multiplier and multiplicand with equal bit length (if necessary add '0' in MSB position)
Step 2: encode the multiplicand
Step 3: do splitter block multiplication in parallel
Step 4: partial product addition
Step 5: find overlapping bits
Step 6: final adder for getting result

Figure 3 shows that block multiplication. Each block consists of 8 bits. The mantissa 24 bit is splitted into three 8 bit block in the first step[11,14]. A0,A1,A2,B0,B1,B2 all are 8 bit length. The next step we recode B0,B1,B2. The next step we consider the partial product the block A0B0, A1B0,A2B0, A0B1, A1B1, A2B1, A0B2, A1B2, A2B2.

As mentioned in Table 2 the recoded B0, B1, B2 are multiple the A0, A1, A2 in parallel. The next step each blocks multiplication generate 4 PP and 2 bit left shift needs between two PP. Generated PP are added in next step. Appropriate adder is used for PP addition explained in following section. Overlapping bit from the adjacent blocks is separated in this step. Final adder adds A1B0 + A0B1, A2B0 + A1B1 +A0B2, A2B1 + A1B2 and the addition of overlapping bit corresponding to each blocks as shown in Figure 5.

5. Partial Product Compression, Overlapping Bit Separation and Final Adder

The partial products are added by using the combination of 3:2 CSA and some Half Adder shown in Figure 6. From the diagram 24x24 multiplication needs nine 8x8 multiplication in PP generation step[15]. Each 8x8 multiplier performs multiplication in parallel. The results added and overlapping bits are separated in next step. Each block generates 4 PP and CSA's and HA's used to add PP's. PPs are added with multi-operand adder like CSA and final adder is used to add two operands. Normally final adder is known as two operands adder such as carry select adder.

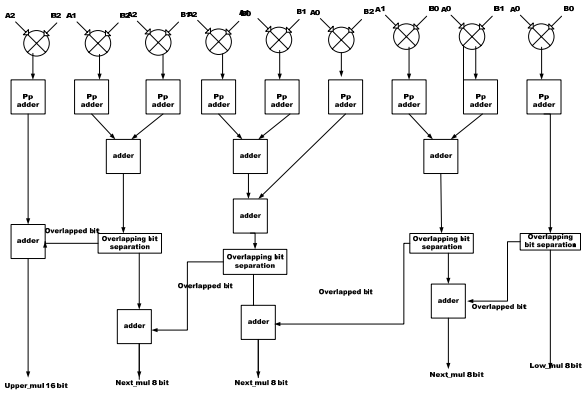

Figure 5. Proposed Multiplier Architecture.

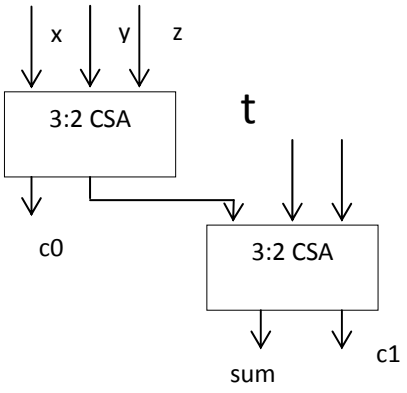

Figure 6. Basic block diagram of 4:2 carry save adder.

The compressor or carry save adder produce sum and their carry in parallel. The propagation of carry is prohibited and passed this sum and carry bit separately to the next level. Final adder have to add two operands such as sum and their carry. Final adder normally a carry select adder or carry skip adder. These adder guarantees for less delay.

The count of 4:2 CSA is only dependent on the inputs w,x,y,z not on cin. Hence we do not wait for carry propagation, it will increase the speed. The 4:2 CSA adders has five inputs and only two outputs (one sum bit, one carry bit). Sometimes in addition carry may be a two bit. For example adding five 1's the sum is 1 and the carry is '10'. In this paper we use two 3:2 CSAs instead of one 4:2 CSA. The five 1's input is given as Figure 7. One 3:2 CSA gives carry as '1' and another 3:2 CSA gives carry '1'. Actual carry '10' propagate to next bit level as two '1's (1 + 1= '10').

Figure 8 shows in stage 1, carry save adder add all PP and produce 16 bit output. In stage 2, splitted blocks PP (16 bit) are added using carry propagate adder. That is A0B1+A1B0, A2B0+A1B1+A0B2, A2B1+A1B2 blocks added together and again produce 16 bit output. Next step includes overlapping bit separation. Each block's upper 8 bits are overlapped with next adjacent blocks. The final adder adds that overlapping bit with corresponding blocks in stage 3. Proper concatenation of output gives 48 bit multiplier output.

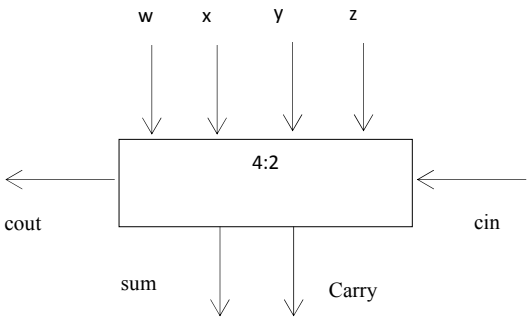

Figure 7. Two bit carry generating method.

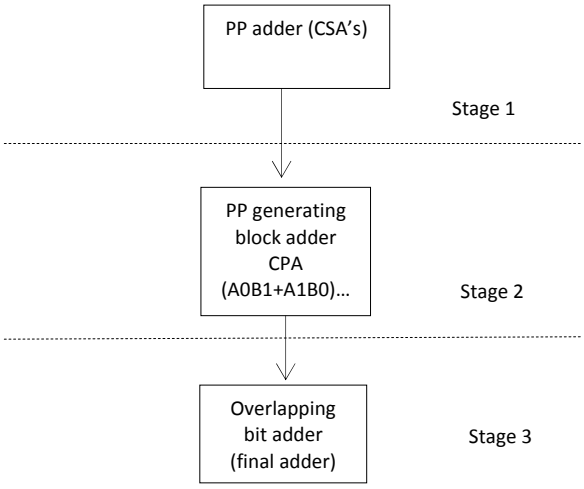

Figure 8. Adder Stages.

6. Sign and Exponent Calculation

In this paper, we put more concentration on mantissa multiplication. Other calculation such as sign, exponent and normalization and exception handling are done in traditional manner[1,2]. The resultant sign is the logical 'XOR' of two sign bit of two operands

$$Result\_sign = sign\_bit1 \text{ xor } sign\_bit2$$

The output of exponent is given by adding two exponent values and adjusts it to its BASE value. The single precision FP BASE value is 127.

$$Result\_exponent = (expn1 + expn2) - 127$$

7. Normalization and Rounding

The next step composes of result normalization and rounding. The result of mantissa multiplication has MSB as 1 means we just left shift the result in one bit position and add 1 to exponent to get the correct answer. If the MSB is not '1' we uses leading zero detector that detects how many zeros present in front of first '1' in the result and again adjust the exponent equal to number of zero detection[2,16]. This process is known as normalization. 48 bit normalized mantissa multiplied result gets back to IEEE 754-2008 format of 24 bit mantissa by the help of rounding. Five different rounding mode is available. Rounding with rounding to nearest even method is followed in this paper.

8. Design Verification

8.1 Implementation Platform

Our design has been implemented on Altera Cyclone II EP2C35F672C6 with speed grade 6 and 33216 logic elements FPGA family. Implementation on Altera is developed by Quartus II v9.1 version and power is

calculated by power play power analyzer. Results are compared against with Altera IP core and some other experiments. Comparison results show that our design can performed with maximum running frequency, optimized area and less power consumption. Our design is synthesized, placed and routed by Quartus EDA tool successfully. The power analysis synthesis result report from Quartus II EDA tool. The total number of logic elements used for this design is synthesized and the report is given in Table 4.

8.2 Performance Comparison

The power comparison made on Altera core and X. Jiang et al. The results are tabulated below in Table 5. The compared result shows that the power consumption reduced slightly.

The synthesized logic elements and maximum running frequency is compared with X. Jiang's design and Gong's design. The comparison tabulated in Table 6.

Table 4. Analysis and synthesis report for logic elements

Power Play Analyser Status	Successful
Quartus II Version	9.1 Build 350
Revision Name	Top1
Top-Level	Top1
Family	Cyclone II
Device	EP2C35F672C6
Power Models	Final
Total Thermal Power Dissipation	121.49mW
Core Dynamic Thermal Power Dissipation	0.00mW
Core Static Thermal Power Dissipation	79.96mW
I/O Thermal Power Dissipation	41.53mW
Total Logic Elements	1.571
Total Combinational Functions	1.571
Dedicated Logic Registers	0

Table 5. Comparison of power with different designs

Design	Total Power Consumption(mW)
Altera core (without DSP)	121.65
Altera core (with DSP)	121.63
X. Jiang(5 pipeline stage)	121.71
Proposed CRSOPP	121.49

Table 6. Comparison of Proposed Design

Design	Maximum Running Frequency(MHz)	Logic elements utilization
Gong's design	130.01	1604
X.Jiang's design	131.25	1581
Proposed design	140.02	1571

9. Conclusion

Parallel processing 32 bit binary FP multiplier with novel recoding is investigated in this paper. The implementation results show that the performance can be improved by doing parallel processing in proper way. Divide the operands into blocks and each block can perform their function in parallel. Gather the results from each block and recombine them in proper way enhances the performance. Multiplication mainly composed of two steps is partial product generation and their accumulation. We concentrate on both steps. Classical recoding algorithm is used to reduce the partial product by half without any consideration like Booth. Mixed combination of CSA and carry select adder can increase the speed of addition. These performances affect the total thermal power to reduce.

In the future, implementing this multiplier on any multiplication oriented processor and also extend this design to double precision floating point multiplier.

10. References

1. Parhami B. Computer Arithmetic: Algorithms and Hardware Designs, Oxford Univ. Press: New York, 1999.
2. Jiang X, Xiao P, Qiu M, Wang G. Performance effects of pipeline architecture on an FPGA-based binary32 floating point multiplier. Microprocessors and Microsystems. 2013; 37(8):1183–191.
3. Booth AD. Signed binary multiplication technique. Quarterly Journal of Mechanical and Applied Mathematics. 1951; 4(2):236–40.

4. Manjunath, Harikiran V, Manikanta K, Sivanantham S, Sivasankaran K. Design and implementation of 16x16 modified booth multiplier. 2015 Online International Conference on Green Engineering and Technologies. 2015. p. 66–70.
5. Sivanantham S, Jagannadha Naidu K, Balamurugan S, Bhuvana Phaneendra D. Low power floating point computation sharing multiplier for signal processing applications. International Journal of Engineering and Technology. 2013; 5 (2):979–85.
6. Niharika S, SuhasSali A, Nithin V, Sivanantham S, Sivasankaran K. Implementation of radix-4 butterfly structure to prevent arithmetic overflow. 2015 Online International Conference on Green Engineering and Technologies. 2015. p. 16–20.
7. Rakesh Babu A, Saikiran R, Sivanantham S. Design of floating point multiplier for signal processing applications. International Journal of Applied Engineering Research. 2013; 8(6):715–22.
8. Sivanantham S. Design of low power floating point multiplier with reduced switching activity in deep submicron technology. International Journal of Applied Engineering Research. 2013; 8(7):851–59.
9. Jaiwal MK, Cheung RC. Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support. Microelectronics Journal. 2013; 44(5):421–30.
10. Even G, Mueller SM, Seidel PM. A dual precision IEEE floating-point multiplier. Integration, the VLSI Journal. 2000; 29(2):167–80.
11. Jaiswal MK, Cheung RCC. VLSI implementation of double precision floating-point multiplier using karatsuba technique. Circuits, systems, and signal processing. 2013; 32(1):15–27.
12. Wang JP, Kuang SR, Liang SC. High-Accuracy Fixed-Width Modified Booth Multipliers for Lossy Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2011; 19(1):52–60.
13. Akkas A, Schulte MJ. Dual-mode floating-point multiplier architectures with parallel operations. Journal of Systems Architecture. 2006; 52(10):549–62.
14. Park J, Kim S, Lee YS. A Low-Power Booth Multiplier Using Novel Data Partition Method. Proceeding of the EEE Asia-Pacific Conference on Advanced System Integrated Circuits (AP-AS1C2004). 2004. p. 54–57.
15. Elguibaly F. A fast parallel multiplier-accumulator using the modified Booth algorithm. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. 2000; 47 (9):902–908.
16. Jeong YS. Parallel Processing Scheme for Minimizing Computational and Communication Cost of Bioinformatics Data. Indian Journal of Science and Technology. 2015; 8(15):1–8.