Journal of Theoretical and Applied Information Technology
30th June 2014. Vol. 64 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT FLOATING-POINT MULTIPLIER DESIGN USING COMBINED BOOTH AND DADDA ALGORITHMS

[1]DHANABAL R, [2]BHARATHI V, [3]NAAMATHEERTHAM R SAMHITHA, [4]PAVITHRA S, [5]PRATHIBA S, [6]JISHIA EUGINE

[1]Asst Prof. (Senior Grade), School of Electronics Engineering,
[2]Asst. Prof., GGR College of Engineering, Anna University, Vellore,
[3]M.Tech VLSI Student, School of Electronics Engineering,
[4, 5, 6] B.Tech ECE Students, School of Electronics Engineering,
VIT University, Vellore, Tamil Nadu, India
E-mail: [1]rdhanabal@vit.ac.in , [2]bharathiveerappan@yahoo.co.in, [3]samhitha.nr@gmail.com, [4]pavithra.girija@gmail.com , [5]prathiba.sivakumar@yahoo.com , [6]jishhhia@gmail.com

ABSTRACT

This paper includes the design of efficient double precision floating-point multiplier using radix-4 Modified Booth Algorithm (MBE) and Dadda Algorithm. This hybrid multiplier is designed by using the advantages in both the multiplier algorithms. MBE has the advantage of reducing partial products to be added. Dadda scheme has the advantage of adding the partial products in a faster manner. Our main objective is to combine these two schemes to make the multiplier design power efficient and area efficient. The floating-point multiplier is designed using Verilog HDL. The design is simulated using Altera ModelSim and synthesized using Cadence RTL compiler in TSMC 45 nanometre technology. It is found that multiplier has reduced power and area and it consumes 4619.23 μW and 34880 μm^2 .

Keywords: *Floating Point Multiplication, Dadda Reduction, Modified Booth, Computer Arithmetic*

1. INTRODUCTION

Multipliers are among the fundamental components in modern electronic systems that run complex calculations in both digital signal processors and general purpose processors. As the technology is improving, many researchers are trying to develop efficient multiplier designs which can offer high speed or low power or low area or the combination of all these three in single multiplier. But for the portable devices power and area is mainly considered and it should be minimized as much as possible. Such type of multiplier is implemented here.

The Booth Algorithm is relatively straight forward way of doing signed multiplications [1]. Modified Booth algorithm reduces the partial products than any other method. Then comes addition of these partial products. In the recent days Dadda or Wallace are been used for addition of partial products. But when we reconsider Wallace and Dadda multipliers it is proved that the hardware requirement for Dadda is less than the Wallace multiplier [2, 3]. Dadda reduction method performs faster addition of partial products [2]. In this work a Booth encoded Dadda multiplier is implemented and the improved performance is compared with some regular multipliers.

The floating-point multiplier (FPM) is the major logical block in the floating-point unit (FPU) [11] [12] and ALU [13]. The IEEE-754 standard sets down specific rules and formats for any system that uses floating-point arithmetic's [4]. The main reason to consider double precision floating-point multiplier is that many standard FPUs support this format [5, 6].

This paper is divided into five sections. Basic floating-point multiplication algorithm is explained in section II. Section III provides the actual design of FPM. In section IV results and comparison is provided. Conclusion is given in section V.

2. FLOATING POINT MULTIPLICATION

2.1 Standard IEEE Floating-Point Representation

Our multiplier unit performs multiplication on double precision floating-point data. i.e., the inputs and outputs to the unit are in standard IEEE-754 double precision format [4]. The standard IEEE-754 double precision floating-point format consists

Journal of Theoretical and Applied Information Technology
30th June 2014. Vol. 64 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of a 64-bit vector split into three sections as shown in Fig. 1. The double precision floating-point number is calculated as shown in equation (1). To represent any floating-point number, the three fields are combined and interpreted as follows:

$$A = (-1)^{sign_A} \times 1.fraction_A \times 2^{\exp_A - bias} \quad (1)$$

Sign(1-bit)	Exponent(11-bit)	Mantissa(52-bit)

Fig. 1. IEEE-754 Double Precision (64-bit) Floating-Point Format

2.2 Floating-Point Multiplication

The floating-point multiplication involves following steps: Assume that the operands A and B are in IEEE-754 double precision format, performing floating-point multiplication $A \times B = (-1)^{A_{\exp}}(A_{man} \times 2^{A_{\exp}}) \times (-1)^{B_{\exp}}(B_{man} \times 2^{B_{\exp}})$ where "man" represents mantissa, "exp" represents exponent.

1. Compute the sign of the result (A_{exp} ^ B_{exp}).
2. Multiply the mantissa.
3. Normalize the product if needed.
4. Compute the exponent of the result: exponent of the result = biased exponent (A_{exp}) + biased exponent (B_{exp}) – *bias*
5. Round the result to the no. of mantissa bits.

The double precision floating-point multiplier architecture is as shown in Fig. 2. The architecture contains a multiplier tree which multiplies two 53-bit numbers (1 hidden bit+52 bit mantissa). Output from the multiplier tree is in carry save format and it is passed to a combined add/round stage, where the carry save product is added and rounded. Both the sign and exponent calculation logic runs in parallel with the mantissa multiplication. To get the final exponent value, the exponent needs to be adjusted based on the rounding. Sometimes there is a chance of getting a wrong value for the exponent as the actual exponent value exceeds the bit range. So to avoid this in our design exception handling is included. Whenever the bit range is high, it generates an exception while performing exponent logic or exponent adjust.

2.3 IEEE Rounding Modes

As per IEEE-754 standards there are four rounding modes as follows:

1. Round to nearest (RN)
2. Round to zero (RZ)
3. Round to positive infinity (RP)
4. Round to negative infinity (RN)

Implementation wise these rounding are further reduced to three. Round up (RU) with fix up, round to infinity, round to zero [7]. Round to nearest can be implemented as Round up with a fix up [7]. We are using RU in our floating-point arithmetics. Mathematically RU is given as follows

$$x = \begin{cases} \lceil x \rceil \; if \;\; x - \lfloor x \rfloor \geq 0.5 \\ \lfloor x \rfloor \quad otherwise \end{cases}$$

Where $\lceil x \rceil$ and $\lfloor x \rfloor$ are the ceiling and floor functions.

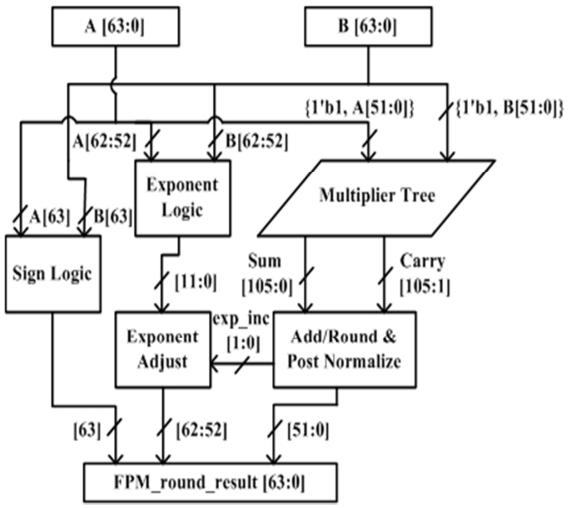

Fig. 2. Floating-Point Multiplier Architecture

3. FLOATING-POINT MULTIPLIER USING BOOTH AND DADDA ALGORITHMS

Simple multiplication operation involves three major steps:

1. Generation of partial products.
2. Reduce the generated partial products into two rows i.e., one row of final sums and one row of carries.
3. These final sums and carries need to be added to generate final result.

In our design for generation of partial products we have used radix-4 Modified Booth Encoding (MBE) approach. Dadda algorithm is used to reduce the generated partial products into one row of final sums and one row of carries [8] [9]. Finally these two rows are added using an efficient parallel prefix adders [10].

3.1 Modified Booth Encoding(MBE)

In our floating-point multiplier design a modified booth encoding scheme is used as it reduces the

Journal of Theoretical and Applied Information Technology
30th June 2014. Vol. 64 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

number of partial products generated. MBE generates at most $\left\lfloor \frac{N}{2} \right\rfloor + 1$ partial products, where N is the number of bits. MBE algorithm involves following steps:

1. Pad the LSB with one zero.
2. Pad the MSB with 2 zeros if n is even and 1 zero if n is odd.
3. Divide the multiplier into overlapping groups of 3-bits.
4. Determine the partial products scale factor from the recoding table.
5. Compute the multiplicand multiples which is nothing but partial products.

Radix-4 recoding, the most common modified booth's recoding scheme and is used with the digit set {-2, -1, 0, 1, 2} is shown in Table I.

Table. I. Radix-4 Modified Booth's Recoding (for $A \times B$)

Bits of multiplier B			Encoding operation on multiplicand A
C_{i+1}	C_i	C_{i-1}	
0	0	0	0
0	0	1	+B
0	1	0	+B
0	1	1	+2B
1	0	0	-2B
1	0	1	-B
1	1	0	-B
1	1	1	0

Each three consecutive bits of the multiplier B represents the input to the booth recoding block. This block selects the right operation on multiplicand A which can be shift or invert (-2B) or invert (-B) or zero or no operation (B) or shift (2B). Fig. 3 shows the generation of partial product using MBE.

For double precision floating –point multiplication two 53-bit (1 hidden bit + mantissa 52 bits) numbers are to be multiplied. If we use normal method for generation of partial products 53 partial products will be obtained. But by using MBE the partial products can be reduced to 27. Each partial product can be obtained using block shown in Fig. 3.

3.2 Dadda Reduction Approach

After the generation of partial products, the partial products need to be added. Usually addition of these partial products consumes time. For this reason Dadda scheme is used to minimize the number of adder stages, by which delay can be minimized. Reduction of these partial products into two rows is done in stages using half adders and full adders. The reduction in size of each stage is calculated by working back from the final stage. Each preceding stage height must be not greater than $\lfloor 3 \bullet successorheight / 2 \rfloor$ [9]. Heights for the various stages can be 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, etc., The dot diagram for Dadda reduction for 9 bit by 9 bit multiplication is shown in Fig. 4, this is performed in 4 stages.

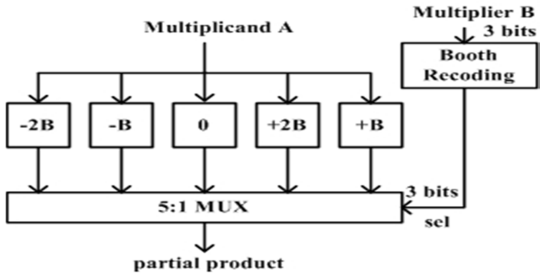

Fig. 3. One Partial Product Generator Using MBE

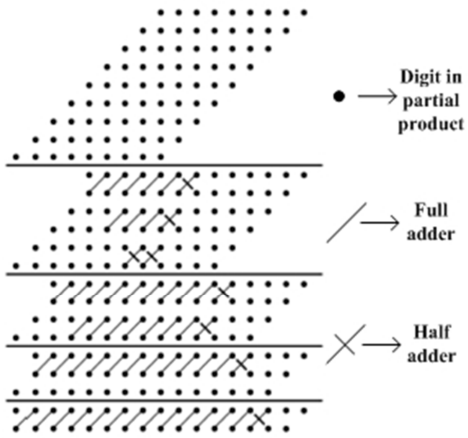

Fig. 4. 9bit by 9 bit Dadda Reduction

3.3 Parallel Prefix Adder

The final sums and carries are added using parallel prefix adders as it offer a highly efficient solution to the binary addition and suitable for VLSI implementations [10]. Among the parallel

Journal of Theoretical and Applied Information Technology
30th June 2014. Vol. 64 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

prefix adders, Kogge-Stone architecture is the widely used and the popular one.

Kogge-Stone adder is the parallel version of carry-look-ahead adder. The term prefix means the outcome of operation depends on the initial inputs. Parallel means the execution of an operation is in parallel. This is done by segmenting into smaller pieces, which are computed in parallel. Kogge-stone adder is considered as the fastest adder design possible [10]. 8-bit Kogge-Stone adder is as shown in Fig. 5. In our design to add the final sums and carries a 106-bit Kogge-Stone adder is used.

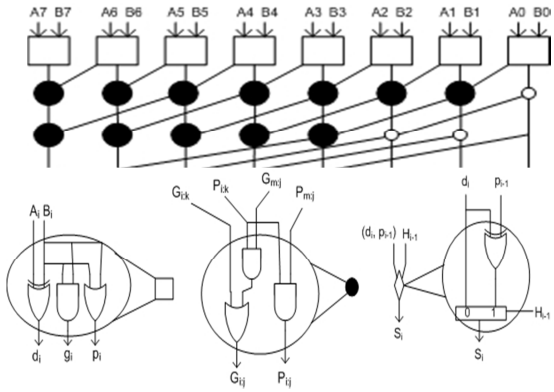

Fig. 5. (A) 8-Bit Kogge-Stone Adder, (B) Logic Implementation of Each Block of Kogge-Stone Adder [10]

3.4 Multiplier Using MBE and Dadda

As discussed in section II mantissas need to be multiplied. For mantissa multiplication, both the concepts of MBE and Dadda reduction is used. Initially the two 53 bit mantissas are considered to generate partial products using Radix-4 Modified Booth Encoding Algorithm as discussed in section III.A. From this 27 rows of partial products will be obtained. These 27 partial products are reduced using Dadda reduction scheme. For example consider Fig. 6, here multiplication of two 10-bit numbers is taken and 5 partial products are generated using MBE. Then these 5 rows of partial products are reduced to two rows in 3 reduction stages, where 4, 3, 2 is the height of each stage. The same context is extended for reduction of 27 partial products. These 27 rows of partial products are reduced to 2 rows in 7 reduction stages, where 19, 13, 9, 6, 4, 3, 2 is height of each stage as we go down in the reduction scheme.

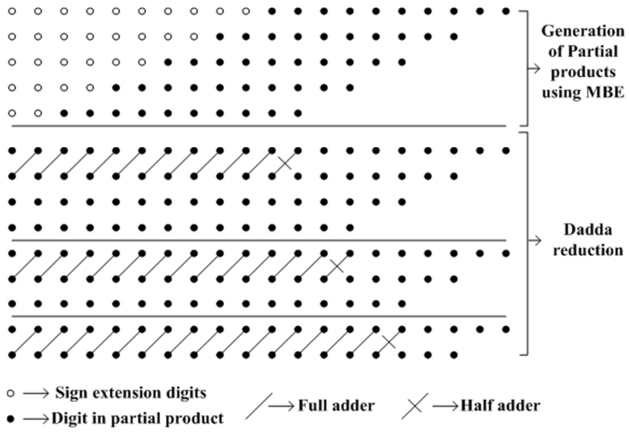

Fig. 6. 10 bit by 10 bit Booth Encoding with Dadda Reduction Scheme

4. RESULTS AND DISCUSSION

The simulation results of the floating-point Booth encoded Dadda multiplier is shown in Fig. 7. Table II provides the synthesis results for the multiplier designed. The Booth encoded Dadda floating-point multiplier has been compared with Booth encoded Wallace tree and simple Modified Booth Encoding Algorithm. Comparison chart for power, area, and delay is given in Fig. 8, Fig. 9, and Fig. 10 respectively. In contrast to the Wallace reduction, Dadda method does the least reduction necessary at each stage.

Dadda reduction scheme uses fewer half adders and full adders when compared to the Wallace reduction scheme. Thus Dadda multipliers area and power is less than Wallace. In conventional MBE, partial products are reduced using CSA trees but it consumes time. To overcome this Dadda reduction scheme is used.

Table II. Synthesis Results Using TSMC 45nm Technology

64-bit Floating –Point Multiplier	Total Power (μW)	Area (μm^2)	Delay (ns)
MBE with Dadda	4619.23	34880.00	6.19
MBE with Wallace	4765.80	35337.30	6.19
MBE	4883.52	35393.56	6.49

Journal of Theoretical and Applied Information Technology
30th June 2014. Vol. 64 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Fig. 7. Simulation results for 64-bit floating-point multiplier

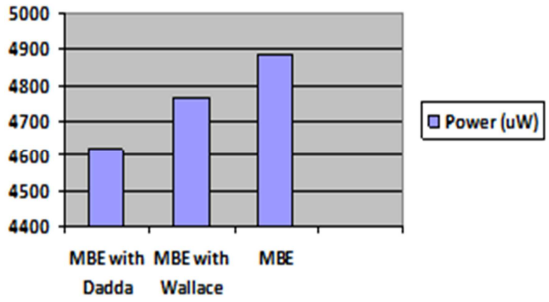

Fig. 8. Power comparison chart for the 64-bit floating-point multiplier

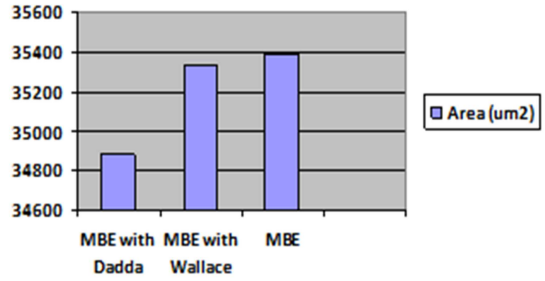

Fig. 9. Area Comparison Chart for the 64-Bit Floating-Point Multiplier

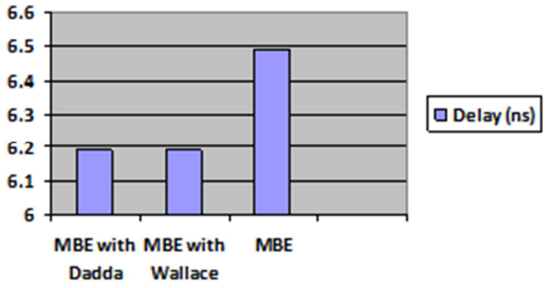

Fig. 10. Delay Comparison Chart for the 64-Bit Floating-Point Multiplier

5. CONCLUSION

This paper presents a low power and area efficient double precision floating point multiplier using Modified Booth Encoding and Dadda reduction. The design has been compared with MBE and MBE with Wallace. It is found that multiplier has reduced power and area and it consumes 4619.23 μW and 34880 μm^2. The comparison results in the previous sections prove that the design is efficient. This multiplier design is suitable for high performance floating point units or floating point multiply-add units of the co-processors.

As the future work this work can be extended to achieve better speeds by using Residue Number System [14].

REFRENCES:

[1] D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanical and Applied Math*., vol. 4, pp.236-240, 1951.

[2] K.C. Bickerstaff, E. E. Swartzlander, and M. J. Schulte, "Analysis of column compression multipliers," in *proceedings of the 15th IEEE symposium on Computer Arithmetic*, pp. 33-39, June 2001.

[3] W. J. Townsend, E. E. Swartzlander, and J. A. Abraham, "A comparison of Dadda and Wallace multiplier delays," in *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, vol. 5205 of *Proceedings of the SPIE*, pp. 552–560, August 2003.

[4] *IEEE Standard for Binary Floating-Point Arithmetic,* ANSI/IEEE Standard 754-1985, Reaffirmed Dec. 6, 1990, Inc., 1985.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[5] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," *IBM J. Res. Development*, vol. 34, pp. 59–70, 1990.

[6] E. Hokenek, R. Montoye, and P. W. Cook, "Second-generation RISC floating point with multiply-add fused," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1207–1213, Oct. 1990.

[7] N. Quach, N. Takagi, and M. Flynn, "On fast IEEE rounding," *Stanford Univ., Stanford, CA, Tech. Rep.* CSL-TR-91-459, Jan. 1991.

[8] L. Dadda, "Some schemes for parallel multipliers," *IEEE Transactions on Computers*, vol. 13, pp.14-17, 1964.

[9] Waters. R. S, Swartzlander. E. E, "A Reduced Complexity Wallace Multiplier Reduction," *Computers, IEEE Transactions on*, vol.59, no.8, pp.1134-1137, Aug. 2010.

[10] Giorgos Dimitrakopoulos and Dimitris Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 225-231, Feb 2005.

[11] Dhanabal R, Bharathi V, Shilpa K, Sujana D.V and Sahoo S.K, **"**Design and Implementation of Low Power Floating Point Arithmetic Unit", *International Journal of Applied Engineering Research,* ISSN 0973-4562, vol. 9, no. 3, pp. 339-346, 2014.

[12] Ushasree G**,** Dhanabal R, Sarat kumar sahoo, "VLSI Implementation of a High Speed Single Precision Floating Point Unit Using Verilog", *Proceedings of IEEE Conference on Information and Communication Technologies (ICT 2013)*, pp. 803-808, 2013.

[13] Dhanabal R, Bharathi V, Salim S, Thomas B, Soman H, and Sahoo.S.K, "Design of 16-bit low power ALU-DBGPU", International Journal of Engineering and Technology, vol. 5 no. 3, pp. 2172 – 2180, Jun 2013.

[14] Dhanabal R, Sarat Kumar Sahoo, Barathi V, N.R.Samhitha, Neethu Acha Cherian, Pretty Mariam Jacob, "Implementation of Floating Point MAC using Residue Number System", *Journal of Theoretical and Applied Information Technology,* vol. 62, no. 2, April 2014.