International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 9 (2015) pp. 22385-22392

http://www.ripublication.com

Modified Booth Recoding For Complex Arithmetic Operation In DSP Application

Ms. V. Lokeshwari [1] and Mr. Vishnu Vardhan [2]

[1]M.E Scholar, Dept of ECE,
Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College, Chennai.
lokece24@gmail.com, vishnuvardhan@veltechmultitech.org
[2]Assistant Professor, Dept of ECE,
Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College, Chennai

Abstract

A Modified booth (MB) recoding is used to recode the input values, so that it reduces the partial product, area, critical delay and power consumption. In this work we focus on Sum-Modified booth(S-MB) recoding for operating the Fused Add Multiply (FAM). Using S-MB recoding, the partial product will be reduced more than the modified booth. In FAM, both the addition and booth recoding is done in the same block, so that the area occupation is occur. A carry save adder (CSA) and carry look ahead adder (CLA) is used to operate the partial product, by using in this method we can reduced the run time.

Index Terms— Fused Add Multiply, ALU, Modified Booth recoding, CLA, CSA, Simulation.

I. INTRODUCTION

I used radix 4 8-bit operands and produces 32-bit multiplication result of the two of its output. The main advantage of modified booth recoder is reducing the number of partial product by half during multiplication while comparing other radix-2 multiplication. The other advantages are reducing critical path delay, power dissipation, usage of logic gates were reduced, and power and time consumption majorly than normal booth recoder. Complex arithmetic operations are widely used in Digital Signal Processing (DSP) applications. In this work, we focus on optimizing the design of the fused Add-Multiply (FAM) operator for increasing performance. We investigate techniques to implement the

direct recoding of the sum of two numbers in its Modified Booth (MB) form. We introduce a structured and efficient recoding technique and explore three different schemes by incorporating them in FAM designs. Comparing them with the FAM designs which use existing recoding schemes, the proposed technique yields considerable reductions in terms of critical delay, hardware complexity and power consumption of the FAM unit.

A. Radix-4 Booth Recoding

To Booth recode the multiplier term, we consider the bits in blocks of three, such that each block overlaps the previous block by one bit. Grouping starts from the LSB, and the first block only uses two bits of the multiplier (since there is no previous block to overlap):

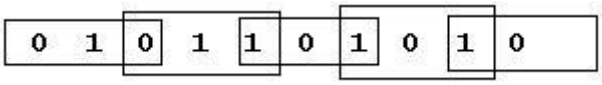


Fig.1.1 Booth recoding in multiplier term.

The least significant block uses only two bits of the multiplier, and assumes a zero for the third bit. The overlap is necessary so that we know what happened in the last block, as the MSB of the block acts like a sign bit. We then consult the table 2-3 to decide what the encoding will be.

Table.1.1 Booth recoding strategy for each of the possible block values.

OP code	Partial Product
000	0
001	1
010	1
011	2
100	-2
101	-1
110	-1
111	0

Since we use the LSB of each block to know what the sign bit was in the previous block, and there are never any negative products before the least significant block, the LSB of the first block is always assumed to be 0. Hence, we would recode our example of 7 (binary 0111) as :

0 1 1 1		
Block 0 :	1 1 0	Encoding : * (-1)
Block 1 :	0 1 1	Encoding : * (2)

In the case where there are not enough bits to obtain a MSB of the last block, as below, we sign extend the multiplier by one bit.

0 0 1 1 1		
Block 0 :	1 1 0	Encoding : * (-1)
Block 1 :	0 1 1	Encoding : * (2)
Block 2 :	0 0 0	Encoding : * (0)

One possible implementation is in the form of a Booth recoder entity, such as the one in figure 2-16, its outputs being used to form the partial product:

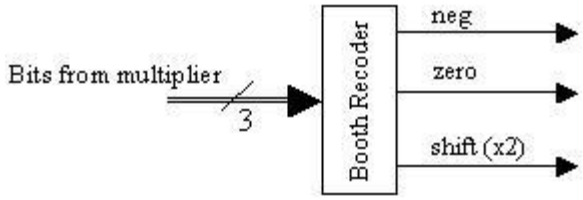


Fig.1.2 Booth Recoder and its associated inputs and outputs.

In fig 2,

- The zero signal indicates whether the multiplicand is zeroed before being used as a partial product
- The shift signal is used as the control to a 2:1 multiplexer, to select whether or not the partial product bits are shifted left one position.
- Finally, the neg signal indicates whether or not to invert all of the bits to create a negative product (which must be corrected by adding "1" at some later stage) Procedure for Paper Submission

The described operations for booth recoding and partial product generation can be expressed in terms of logical operations if desired but, for synthesis, it was found to be better to implement

II. WORK DOWN

A. FUSED ADD MULTIPLY

In this paper, we focus on AM units which implement the operation, $Z=X(A+B)$. The conventional design of the AM operator requires that its inputs A and B are first driven to

an adder and then the input X and the sum $Y=A+B$ are driven to a multiplier in order to get Z. The drawback of using an adder is that it inserts a significant delay in the critical path of the AM. As there are carry signals to be propagated inside the adder, the critical path depends on the bit-width of the inputs. In order to decrease this delay, a Carry-Look-Ahead (CLA) adder can be used which, however, increases the area occupation and power dissipation. An optimized design of the AM operator is based on the fusion of the adder and the MB encoding unit into a single datapath block (Fig. 1(b)) by direct recoding of the sum $Y=A+B$ to its MB representation [10]–[13], [23]. The fused Add -Multiply (FAM) component contains only one adder at the end (final adder of the parallel multiplier). As a result, signifi-cant area savings are observed and the critical path delay of the recoding process is reduced and decoupled from the bit- width of its inputs. In this work, we present a new technique for direct recoding of two numbers in the MB representation of their sum.

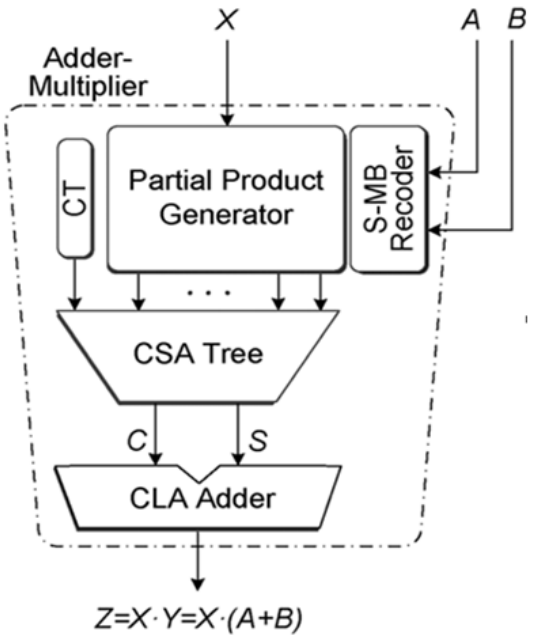


Fig 1.3 Block diagram for FAM

B. PROPOSED SYSTEM

SIGNED NUMBER REPRESENTATION

In computing, signed number representations are required to encode negative numbers in binary number systems. In mathematics, negative numbers in any base are represented by prefixing them with a minus ("−") sign. However, in computer hardware, numbers are represented only as sequences of bits, without extra symbols. The four best-known methods of extending the binary numeral system to represent signed numbers are: sign-and-magnitude, ones' complement, two's complement, and excess-K. Some of the alternative methods use the implicit instead of explicit signs, such as negative binary using base −2.

Corresponding methods can be devised for other bases, whether positive, negative, fractional, or other elaborations on such themes. There is no definitive criterion by which any of the representations is universally superior. The representation used in most current computing devices is two's complement, although the Unisys Clear Path Dorado series mainframes use ones' complement. An ALU must process numbers using the same formats as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement; two's complement, sign-magnitude format, and even true decimal systems, with various representation of the digits.

The ones' complement and two's complement number systems allow for subtraction to be accomplished by adding the negative of a number in a very simple way which negates the need for specialized circuits to do subtraction; however, calculating the negative in two's complement requires adding a one to the low order bit and propagating the carry. An alternative way to do two's complement subtraction of A−B is to present a one to the carry input of the adder and use ¬B rather than B as the second input. The arithmetic, logic and shift circuits introduced in previous sections can be combined into one ALU with common selection.

The more complex the operation, the more expensive the ALU is, the more space it uses in the processor, and the more power it dissipates. Therefore, engineers compromise. They make the ALU powerful enough to make the processor fast, yet not so complex as to become prohibitive.

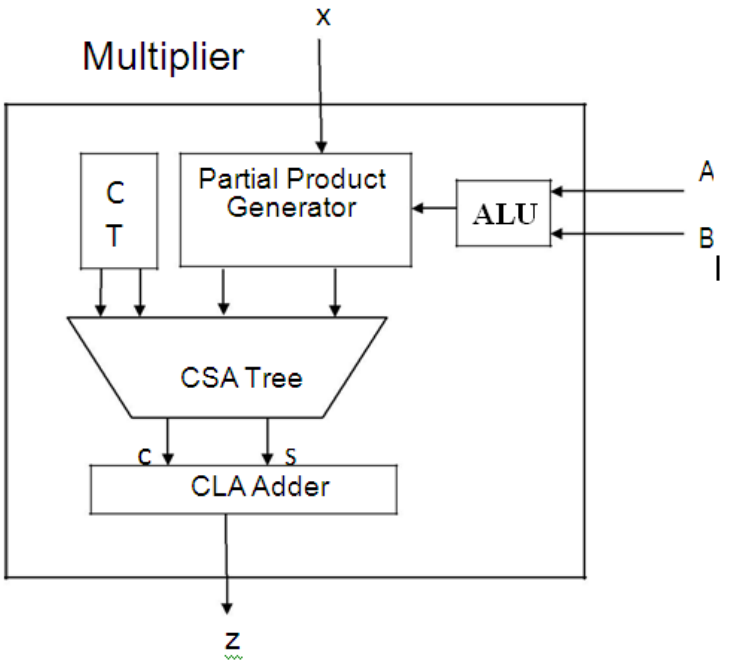


Fig 1.4 Block diagram for fused ALU multiply

C. Fused ALU-multiply

For example, computing the square root of a number might use:

a. Calculation in a single clock Design an extraordinarily complex ALU that calculates the square root of any number in a single step.
b. Calculation pipeline Design a very complex ALU that calculates the square root of any number in several steps. The intermediate results go through a series of circuits arranged like a factory production line. The ALU can accept new numbers to calculate even before having finished the previous ones. The ALU can now produce numbers as fast as a single-clock ALU, although the results start to flow out of the ALU only after an initial delay.
c. Iterative calculation Design a complex ALU that calculates the square root through several steps. This usually relies on control from a complex control unit with built-in microcode.
d. Co-processor Design a simple ALU in the processor, and sell a separate specialized and costly processor that the customer can install just beside this one, and implements one of the options above.
e. Software libraries tell the programmers that there is no co-processor and there is no emulation, so they will have to write their own algorithms to calculate square roots by software.
f. Software emulation Emulate the existence of the co-processor. Whenever a program attempts to perform the square root calculation, make the processor check if there is a co-processor present and use it if there is one; if there is not one, interrupt the processing of the program and invoke the operating system to perform the square root calculation through some software algorithm.

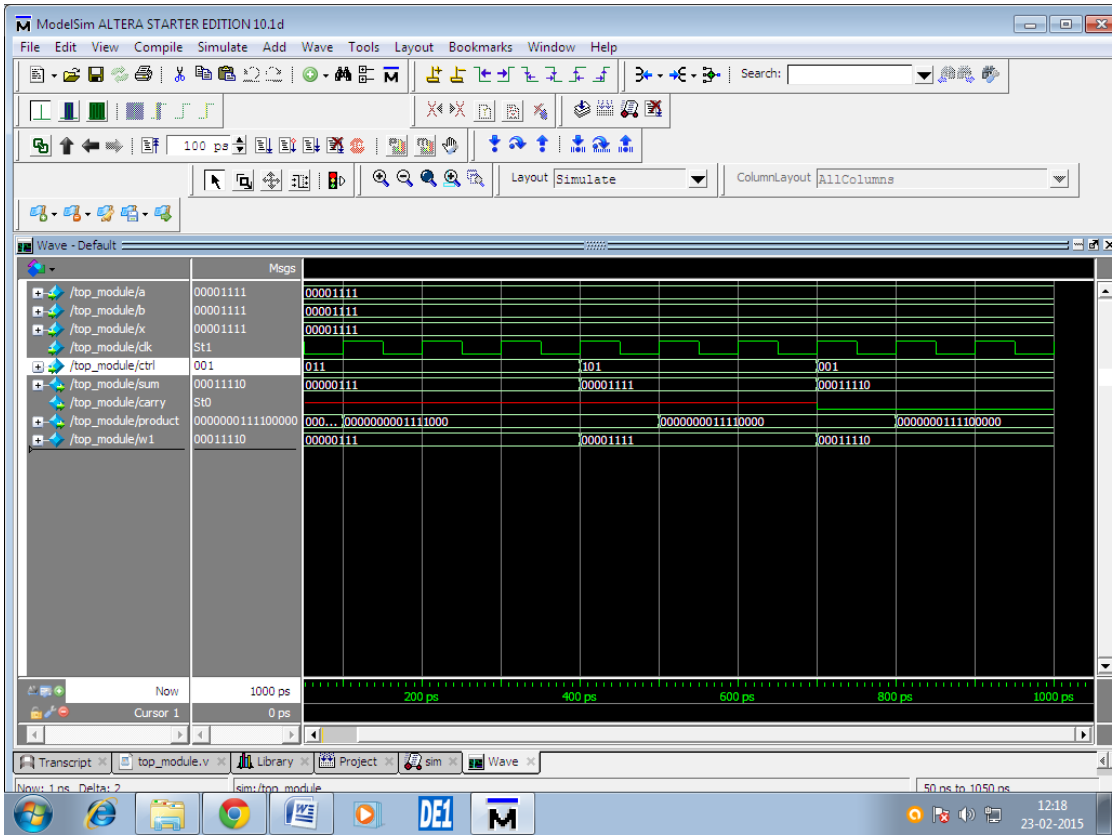
D. INPUTS AND OUTPUTS

The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation. One thing designers must keep in mind is whether the ALU will operate on big-endian or little-endian numbers.A floating-point unit also performs arithmetic operations between two values, but they do so for numbers in floating-point representation, which is much more complicated than the two's complement representation used in a typical ALU. In order to do these calculations, a FPU has several complex circuits built-in, including some internal ALUs. In modern practice, engineers typically refer to the ALU as the circuit that performs integer arithmetic operations (like two's complement and BCD). Circuits that calculate more complex formats like floating point, complex numbers, etc.

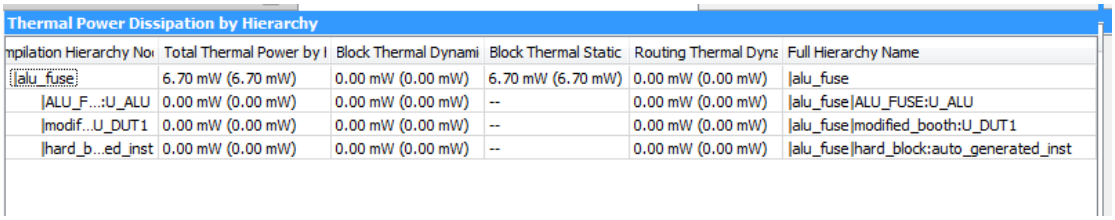
III. SIMULATION RESULTS AND FUTURE WORK

For using sum-modified booth we reduced the partial product and also we used ALU for control the usage of time. In final simulation we reduced the power, logic gates and usage of total pins is 53, total register 14 and I/O pins 53.

OUTPUT WAVEFORM



POWER DISIPATION



Thermal Power Dissipation by Hierarchy

mpilation Hierarchy No	Total Thermal Power by I	Block Thermal Dynami	Block Thermal Static	Routing Thermal Dyna	Full Hierarchy Name
alu_fuse	6.70 mW (6.70 mW)	0.00 mW (0.00 mW)	6.70 mW (6.70 mW)	0.00 mW (0.00 mW)	|alu_fuse
|ALU_F...:U_ALU	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	--	0.00 mW (0.00 mW)	|alu_fuse|ALU_FUSE:U_ALU
|modif...U_DUT1	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	--	0.00 mW (0.00 mW)	|alu_fuse|modified_booth:U_DUT1
|hard_b...ed_inst	0.00 mW (0.00 mW)	0.00 mW (0.00 mW)	--	0.00 mW (0.00 mW)	|alu_fuse|hard_block:auto_generated_inst

IV. CONCLUSION

In this project modified booth multiplier has been used for recoding input in fused add-multiplier and there by reducing the usage of logic gates. And also the critical delay has been reduced. In the future work it is planned to use ALU inside the multiplier so that we can perform any of the arithmetic operations in that specific multiplier ,which cannot be done in the previous one..

REFERENCES

[1] E. E. Swartzlander and H. H. M. Saleh, "FFT implementation with fused floating-point operations," IEEE Trans. Comput., vol. 61, no. 2, pp. 284–288, Feb. 2012.

[2] A. Amaricai, M. Vladutiu, and O. Boncalo, "Design issues and implementations for floating-point divide-add fused," IEEE Trans. Circuits Syst. II–Exp. Briefs, vol. 57, no. 4, pp. 295–299, Apr. 2010.

[3] Y.-H. Seo and D.-W. Kim, "A new VLSI architecture of parallel multiplier–accumulator based on Radix-2 modified Booth algorithm," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 18, no. 2, pp. 201–208, Feb. 2010.

[4] L.-H. Chen, O. T.-C. Chen, T.-Y.Wang, and Y.-C. Ma, "A multiplication-accumulation computation unit with optimized compressors and minimized switching activities," in Proc. IEEE Int, Symp. Circuits and Syst., Kobe, Japan, 2005, vol. 6, pp. 6118–6121.

[5] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," IEEE Trans. Signal Process., vol. 51, no. 3, pp. 864–874, Mar. 2003.

[6] [6] O. Kwon, K. Nowka, and E. E. Swartzlander, "A 16-bit by 16-bitMAC design using fast 5: 3 compressor cells," J. VLSI Signal Process. Syst., vol. 31, no. 2, pp. 77–89, Jun. 2002.

[7] [7] A. Peymandoust and G. de Micheli, "Using symbolic algebra in algorithmic level DSP synthesis," in Proc. Design Automation Conf., Las Vegas, NV, 2001, pp. 277–282.

[8] [8] S. Nikolaidis, E. Karaolis, and E. D. Kyriakis-Bitzaros, "Estimation of signal transition activity in FIR filters implemented by a MAC architecture," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 19, no. 1, pp. 164–169, Jan. 2000.