International Journal on Emerging Technologies **2**(1): 14-16(2011) **ISSN : 0975-8364**

Implementation of Radix-2 Booth Multiplier and Comparison with Radix-4 Encoder Booth Multiplier

Sandeep Shrivastava*, Jaikaran Singh* and Mukesh Tiwari*

Department of Electronics and Communications,
**SSSITS, Sehore, (M.P.)*

(Received 5 Jan., 2011, Accepted 10 Feb., 2011)

ABSTRACT : This paper Describes implementation of radix-2 Booth Multiplier and this implementation is compared with Radix-4 Encoder Booth Multiplier. This Implementation describes in the Form of RTL Schematic and Comparison is also done by using RTL Schematic. A Conventional Booth Multiplier consists of the Booth Encoder, the partial-product tree and carry propagate adder [2, 3]. Different schemes are addressed to improve the area and circuit speed effectively.

I. INTRODUCTION

Multipliers are key components of many high performance systems such as FIR filters, Microprocessor, digital signal processors, etc. A systems performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue however; area and speed are usually conflicting constraints so that improving speed results mostly in larger areas [5, 6].

Booth Multiplier reduces number of iteration step to perform multiplication as compare to Conventional steps. Booth algorithm 'scans' the multiplier operand and skips chains of This algorithm can reduce the number of additions required to produce the result compared to Conventional Multiplication algorithm, where each bit of the multiplier is multiplied with the multiplicand and the partial products are aligned and added together [4]. More interestingly the number of additions is data dependent, which makes this algo

II. COMMON FEATURES OF MULTIPLIERS

(*i*) Counterflow Organization: A novel multiplier organization is introduced, in which the data bits flow in one direction, and the Booth commands are piggybacked on the acknowledgments flowing in the opposite direction.

(*ii*) Merged Arithmetic/Shifter Unit: An architectural optimization is introduced that merges the arithmetic operations and the shift operation into the same function unit, thereby obtaining significant improvement in area, energy and speed.

(*iii*) Overlapped Execution: The entire design is pipelined at the bit-level, which allows overlapped execution of Proceedings of multiple iterations of the Booth algorithm, including across successive multiplications. As a result, both the cycle time per Booth iteration, as well as the overall cycle time per multiplication are significantly improved.

(*iv*) Modular Design: The design is quite modular, which allows the implementation to be scaled to arbitrary operand widths without the need for gate resizing, and without incurring any overhead on iteration time.

(*v*) Precision-Energy Trade-Off: Finally, the architecture can be easily modified to allow dynamic specification of operand widths, i.e., successive operations of a given multiplier implementation could operate upon different word length

III. RADIX-2 BOOTH MULTIPLIER

Our multiplier is of the iterative Radix-2 Booth Multiplier type, implemented using asynchronous circuits [6, 7]. An iterative implementation was chosen, as opposed to a combinational array type, for higher area efficiency. A Booth implementation was chosen so as to uniformly handle signed as well as unsigned operands. However, a minor modification to the controller can easily transform our design into a simple (*i.e.* non-Booth) iterative multiplier.

A. Radix-2 Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation. The booth algorithm with the following example:

Example: 2 ten × (–4) ten

0010 two × 1100 two

Step 1: Making the Booth table

I. From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.

i.e., 0010 — From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, and so there are two changes on this one,

1100 — From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.

Therefore, multiplication of 2 × (–4), where 2 ten (0010 two) is the multiplicand and (–4) ten (1100two) is the multiplier.

II. Let X = 1100 (multiplier)

Let Y = 0010 (multiplicand)

Take the 2's complement of Y and call it –Y

–Y = 1110

III. Load the X value in the table.

IV. Load 0 for X-1 value it should be the previous first least significant bit of X

V. Load 0 in U and V rows which will have the product of X and Y at the end of operation.

VI. Make four rows for each cycle; this is because we are multiplying four bits numbers.

Step 2: Booth Algorithm

Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules:

Look at the first least significant bits of the multiplier "X", and the previous least significant bits of the multiplier "X - 1".

I 0 0 Shift only

1 1 Shift only.

0 1 Add Y to U, and shift

1 0 Subtract Y from U, and shift or add (-Y) to U and shift

II Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

III Shift X circular rights shift because this will prevent us from using two registers for the X value.

B. Synthesis of Radix – 2 Booth Multiplier

This is RTL Schematic of Radix-2 Booth Multiplier shown in figure:1. Here first block is for finding the Multiplier between two given numbers and Second block is for the Multiplication process.

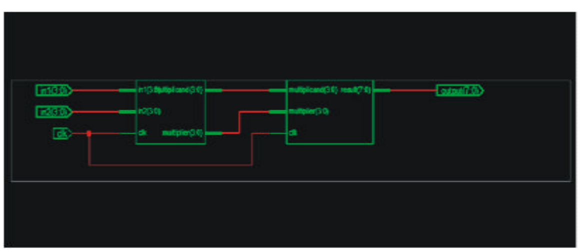

Fig. 1. RTL Schematic of Radix–2 Booth Multiplier.

To represent detail of Second step perform in Algorithm is given in Fig. 2.

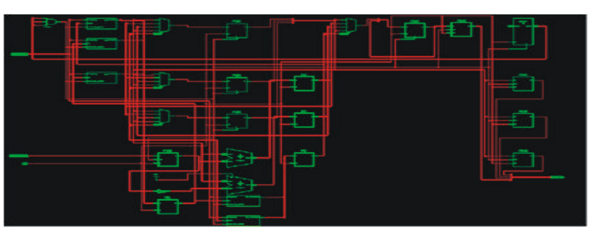

Fig. 2. Detail RTL Schematic of Second step

To represent detail of first Step perform in the Algorithm is given Fig. 3.

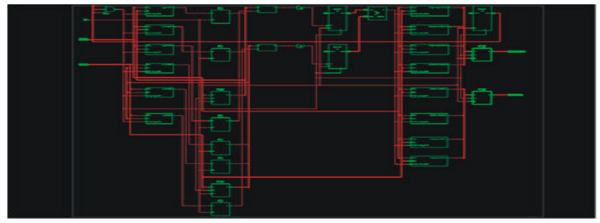

Fig. 3. Detail RTL Schematic of first step.

IV. RADIX-4 ENCODER BOOTH MULTIPLIER

Radix-4 Booth algorithm which scan strings of three bits with the algorithm given below:

(1) Extend the sign bit 1 position if necessary to ensure that n is even.

(2) Append a 0 to the right of the LSB of the multiplier.

(3) According to the value of each vector, each Partial Product will be 0, $+y$, $-y$, $+2y$ or $-2y$. The negative values of y are made by taking the 2's complement.

The negative values of y are made by taking the 2's complement and Carry-look-ahead (CLA) fast adders are used. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing n-bit parallel multipliers, only $n/2$ partial products are generated [10,11,12].

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.

Let us consider an example:

Multiplicand - $(001011)_2$

Multiplier - $(010011)_2$

First of all we will make group of three bits for Multiplier

Encoding for Radix-4 Booth Multiplier will be done according to the table: 3.1 given below:

Table1: Encoding of Radix-4 Booth Multiplier.

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

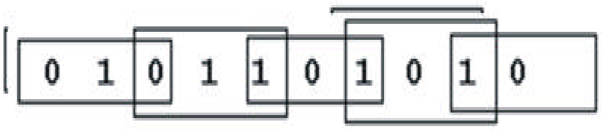

Fig. 4. Grouping of bits for Multiplier.

From the table 1.

(010)2 - 1

(001)2 - 1

(110)2 - (-1)

Therefore Multiplicand is multiplied with the three encoded digit which is 1, 1 and (–1).

(i) –1 * $(001011)_2 = (001011)_2$

And 1111 added with the result because of negative sign. Thus final answer of multiplication of (-1) is (1111001011)2 {Negative term Sign Extended}

(*ii*) 1 * $(001011)_2 = (001011)_2$

(*iii*) 1 * $(001011)_2 = (001011)_2$

(*iv*) $(00001)_2$ are added with these three resultants as an error correction for negation.

1 1 1 1 1 1 0 1 0 0 Negative term sign extended

0 0 1 0 1 1

0 0 1 0 1 1

0 0 0 0 1 Error Correction for Negation

0 0 1 1 0 1 0 0 0 1 Discarding the carried high bit

A. Synthesis of Radix-4 Booth Multiplier

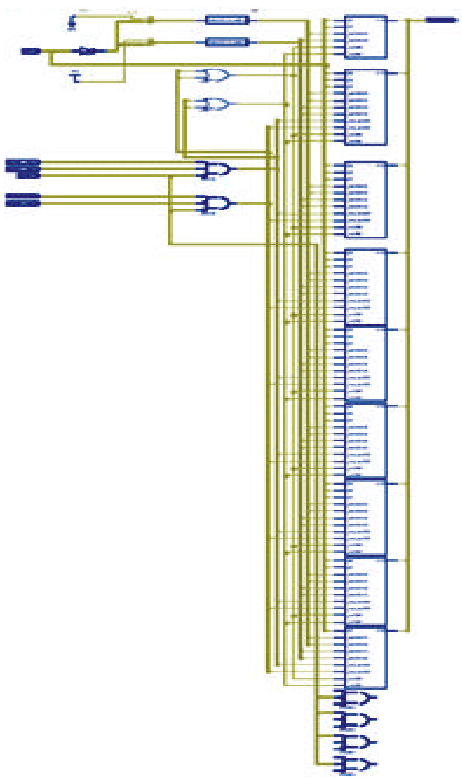

Fig. 5. RTL schematic of Radix-4 Encoder Multiplier.

This is detail diagram of RTL Schematic of Radix-4 Booth Multiplier which provide Output according to the Algorithm.

V. CONCLUSION

Radix-2 Booth Multiplier is implemented here; the complete process of the implementation is giving higher speed of operation. The four cycle of shifting process including addition and subtraction is available. Now at the same time RTL Schematic generated here is giving the comfortable execution of it. This RTL Schematic can be implemented in FPGA CPLD kit that will give the proper Output.

Now this RTL Schematic of Radix-2 Booth Multiplier is compared with implemented RTL Radix-4 Encoder Booth Multiplier. The Speed and Circuit Complexity is compared, Radix-4 Booth Multiplier is giving higher speed as compared to Radix-2 Booth Multiplier and Circuit Complexity is also less as compared to it. It is completely depend on the Algorithm used in both Multipliers.

REFRENCE

[1] A High-Speed Multiplication Algorithm Using Modified Partial Product Reduction Tree P. Asadee, *International Journal of Electrical and Electronics Engineering*, **4**: 4, (2010).

[2] N. Jiang and D. Harris, "Parallelized Radix-2 Scalable Montgomery Multiplier," submitted to IFIP Intl. Conf. on VLSI, (2007).

[3] D. Kudeeth., "Implementation of low-power multipliers", Journal of low-power electronics, vol. **2**, 5-11, (2006).

[4] Y.N. Ching, "Low-power high-speed multipliers", *IEEE Transactions on Computers*, vol. **54**, no. 3, pp. 355-361, 2005.

[5] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 Montgomery multiplier," Proc. 17th IEEE Symp. Computer Arithmetic, pp. 172-178, 2005.

[6] K. Kelley and D. Harris, "Parallelized very high radix scalable Montgomery multipliers," Proc. Asilomar Conf. Signals, Systems, and Computers, pp. 1196-1200, Nov. (2005).

[7] J. Hensley, A. Lastra, and M. Singh. An area- and energy-efficient asynchronous booth multiplier for mobile devices. In Proc. Int. Conf. Computer Design (ICCD), (2004).

[8] A. Efthymiou, W. Suntiamorntut, J. Garside, and L. Brackenbury. An asynchronous, iterative implementation of the original Booth multiplication algorithm. In Proc. Int. Symp. On Advanced Research in Asynchronous Circuits and Systems, pages 207–215. *IEEE Computer Society Press*, Apr. (2004).

[9] P. D. Chidgupkar and M. T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engg. Edu, vol. **8**, no. 2, pp. 153–158, (2004).

[10] M. Sheplie, "High performance array multiplier", IEEE transactions on very large scale integration systems, vol. 12, no. 3, pp. 320-325, (2004).

[11] Design of a Novel Radix-4 Booth Multiplier, Hsin-Lei Lin, Robert C. Chang, Ming-Tsai ChanDepartment of Electrical Engineering, National Chung Hsing University, Taichung, TaiwanThe 2004 IEEE Asia-Pacific Conference on Circuits and Systems, December 6-9, (2004).

[12] H. Lee, "A High-Speed Booth Multiplier", ICCS, (2002).