INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 6, Issue 1 APR 2017

Design of A Novel Pre Encoded Multiplier Architecture Based on NR4SD Encoding Technique

AMIT KUMAR [1]
amitjaiswalec162icfai@gmail.com[1]
[1]PG Scholar, VLSI , Bhagwant University
Ajmer, Sikar Road ,Ajmer,Rajasthan,India.

NIDHI VERMA [2]
verma.nidhi17@gmail.com [2]
[2]Associate Professor, Bhagwant University
Ajmer, Sikar Road ,Ajmer,Rajasthan,India.

***Abstract*:** The most effective way to increase the speed of a multiplier is to reduce the number of the partial products because multiplication precedes a series of additions for the partial products. To reduce the number of calculation steps for the partial products, MBA algorithm has been applied mostly where CSA has taken the role of increasing the speed to add the partial products. To increase the speed of the MBA algorithm, many parallel multiplication architectures have been researched. A modified booth multiplier has been designed which provides a flexible arithmetic capacity and a tradeoff between output precision and power consumption. Moreover, the ineffective circuitry can be efficiently deactivated, thereby reducing Power consumption and increasing speed of operation. The experimental results have shown that the proposed multiplier outperforms the conventional multiplier in terms of power and speed of operation. In this paper we used Xilinx-ISE tool for logical verification, and further synthesizing it on Xilinx -ISE tool using target technology and performing placing & routing operation for system verification.

Keywords: Computer arithmetic, multiplication by constants, common sub expressions sharing, Add-Multiply operation, arithmetic circuits, Modified Booth recoding, VLSI design.

I. INTRODUCTION

A Digital multiplier is the fundamental component in general purpose microprocessor and in DSP. Most of the DSP methods use discrete cosine transformations in discrete wavelet transformations. Compared with many other arithmetic operations multiplication is time consuming and power hungry. Thus enhancing the performance and reducing the power dissipation are the most important design challenges for all applications in which multiplier unit dominate the system performance and power dissipation. The one most effective way to increase the speed of a multiplier is to reduce the number of the partial products. Although the number of partial products can be reduced with a higher radix booth encoder, but the number of hard multiples that are expensive to generate also increases simultaneously. To increase the speed and performance, many parallel MAC architectures have been proposed. Parallelism in obtaining partial products is the most common technique used in this architecture. There are two common approaches that make use of parallelism to enhance the multiplication performance. The first one is reducing the number of partial product rows and second one is the carry-save-tree technique to reduce multiple partial product rows as two "carry-save" redundant forms.

Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in the general purpose processors today, especially since the media processing took off. In the past multiplication was generally implemented via a sequence of addition, Subtraction, and shift operations. Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product. In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two

parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them.

II. DIFFERENT MULTIPLIERS

Binary Multiplication

In the binary number system the digits, called bits, are limited to the set. The result of multiplying any binary number by a single binary bit is either 0, or the original number. This makes forming the intermediate partial-products simple and efficient. Summing these partial-products is the time consuming task for binary multipliers. One logical approach is to form the partial-products one at a time and sum them as they are generated. Often implemented by software on processors that do not have a hardware multiplier, this technique works fine, but is slow because at least one machine cycle is required to sum each additional partial-product. For applications where this approach does not provide enough performance, multipliers can be implemented directly in hardware.

Hardware Multipliers

Direct hardware implementations of shift and add multipliers can increase performance over software synthesis, but are still quite slow. The reason is that as each additional partial -product is summed a carry must be propagated from the least significant bit (LSB) to the most significant bit (MSB). This carry propagation is time consuming, and must be repeated for each partial product to be summed. One method to increase multiplier performance is by using encoding techniques to reduce the the number of partial products to be summed. Just such a technique was first proposed by Booth [BOO 511. The original Booth‟s algorithm ships over contiguous strings of l‟s by using the property that: 2” + 2(n-1) + 2(n-2) + . . . + 2hm) = 2(n+l) - 2(n-m). Although Booth's algorithm produces at most N/2 encoded partial products from an N bit operand, the number of partial products produced varies. This has caused designers to use modified versions of Booth‟s algorithm for hardware multipliers. Modified 2-bit Booth encoding halves the number of partial products to be summed.Since the resulting encoded partial-products can then be summed using any suitable method, modified 2 bit Booth encoding is used on most modern floating-point chips LU 881, MCA 861.

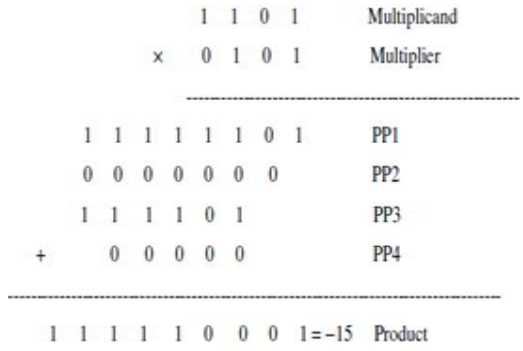


Fig 1.General multiplier

A few designers have even turned to modified 3 bit Booth encoding, which reduces the number of par tial products to be summed by a factor of three IBEN 891. The problem with 3 bit encoding is that the carry-propagate addition required to form the 3X multiples often overshadows the potential gains of 3 bit Booth encoding. To achieve even higher performance advanced hardware multiplier architectures search for faster and more efficient methods for summing the partial-products.

Most increase performance by eliminating the time consuming carry propagate additions. To accomplish this, they sum the partial -products in a redundant number representation. The advantage of a redundant representation is that two numbers, or partial -products, can be added together without propagating a carry across the entire width of the number. Many redundant number representations are possible.

One commonly used representation is known as carry-save form. In this redundant representation two bits, known as the carry and sum, are used to represent each bit position. When two numbers in carry -save form are added together any carries that result are never propagated more than one bit position. This makes adding two numbers in carry-save form much faster than adding two normal binary numbers where a carry may propagate. One common method that has been developed for summing rows of partial products using a carry-save representation is the array multiplier.

III. Existing system

Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in the general purpose processors today, especially since the media processing took off. In the past multiplication was generally implemented via a sequence of addition, subtraction, and shift operations.

Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product. In most computers , the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them. The basic multiplication principle is two fold i.e. evaluation of partial products and accumulation of the shifted partial products. It is performed by the successive additions of the columns of the shifted partial product matrix. The „multiplier" is successfully shifted and gates the appropriate bit of the „multiplicand". The delayed, gated instance of the multiplicand must all be in the same column of the shifted partial product matrix. They are then added to form the product bit for the particular form. Multiplication is therefore a multi operand operation. To extend the multiplication to both signed and unsigned.

Modified Booth is a redundant radix-4 encoding technique]. Considering the multiplication of the 2's complement numbers A, B, each one consisting of n = 2k bits, B can be represented in MB form as:

$$B = \langle b_{n-1} \dots b_0 \rangle_{2's} = -b_{2k-1}2^{2k-1} + \sum_{i=0}^{2k-2} b_i 2^i = \langle \mathrm{b}_{k-1}^{MB} \dots \mathrm{b}_0^{MB} \rangle_{MB} = \sum_{j=0}^{k-1} \mathrm{b}_j^{MB} 2^{2j}. \quad (1)$$

Digits $\mathrm{b}_j^{MB} \in \{-2,-1,0,+1,+2\}, 0 \le j \le k-1$, are formed as follows:

$$\mathrm{b}_j^{MB} = -2b_{2j+1} + b_{2j} + b_{2j-1}, \quad (2)$$

where $b_{-1} = 0.$ Each MB digit is represented by the bits s, one and two (Table 1).

TABLE 1
Modified Booth Encoding

b_{2j+1}	b_{2j}	b_{2j-1}	b_j^{MB}	s_j	one_j	two_j
0	0	0	**0**	0	0	0
0	0	1	**+1**	0	1	0
0	1	0	**+1**	0	1	0
0	1	1	**+2**	0	0	1
1	0	0	**−2**	1	0	1
1	0	1	**−1**	1	1	0
1	1	0	**−1**	1	1	0
1	1	1	**0**	1	0	0

The bit s shows if the digit is negative (s = 1) or positive (s = 0). One shows if the absolute value of a digit equals 1 (one = 1) or not (one = 0). Two shows if the absolute value of a digit equals 2 (two = 1) or not (two = 0). Using these bits, we calculate the MB digits bMBj as follows:

$$\mathrm{b}_j^{MB} = (-1)^{s_j} \cdot (one_j + 2two_j). \quad (3)$$

Equations (4) form the MB encoding signals.

$$s_j = b_{2j+1}, \quad one_j = b_{2j-1} \oplus b_{2j}, \quad two_j = (b_{2j+1} \oplus b_{2j}) \wedge \overline{one}_j. \quad (4)$$

IV. Proposed System

NR4SD⁻ Encoding Scheme

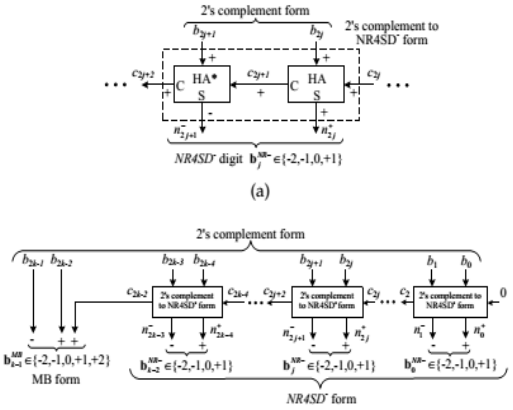


Fig. 2. Block Diagram of the NR4SD⁻ Encoding Scheme at the (a) Digit and (b) Word Level.

The following Boolean equations summarize the HA* operation:

$$c_{2j+2} = b_{2j+1} \vee c_{2j+1}, \quad n^-_{2j+1} = b_{2j+1} \oplus c_{2j+1}.$$

HA* DUAL OPERATION.

Inputs		Output Value[2]	Outputs	
p (-)	q (-)		c (-)	s (+)
0	0	0	0	0
0	1	-1	1	1
1	0	-1	1	1
1	1	-2	1	0

[2] $Output Value = -2 \cdot c + s = -p - q$

HA*

$c = p \vee q$

$s = p \oplus q$

Calculate the value of the $\mathbf{b}_j^{NR-}$ digit.

$$\mathbf{b}_j^{NR-} = -2n^-_{2j+1} + n^+_{2j}.$$

Table 2 shows how the NR4SD digits are formed. The NR4SD encoding signals one_j^+, one_j^- and two_j^- of Table 2 are generated.

TABLE 2
NR4SD⁻ Encoding

2's complement			NR4SD⁻ form			Digit	NR4SD⁻ Encoding		
b_{2j+1}	b_{2j}	c_{2j}	c_{2j+2}	n^-_{2j+1}	n^+_{2j}	$\mathbf{b}_j^{NR-}$	one_j^+	one_j^-	two_j^-
0	0	0	0	0	0	**0**	0	0	0
0	0	1	0	0	1	**+1**	1	0	0
0	1	0	0	0	1	**+1**	1	0	0
0	1	1	1	1	0	**-2**	0	0	1
1	0	0	1	1	0	**-2**	0	0	1
1	0	1	1	1	1	**-1**	0	1	0
1	1	0	1	1	1	**-1**	0	1	0
1	1	1	1	0	0	**0**	0	0	0

NR4SD⁺ Encoding Scheme

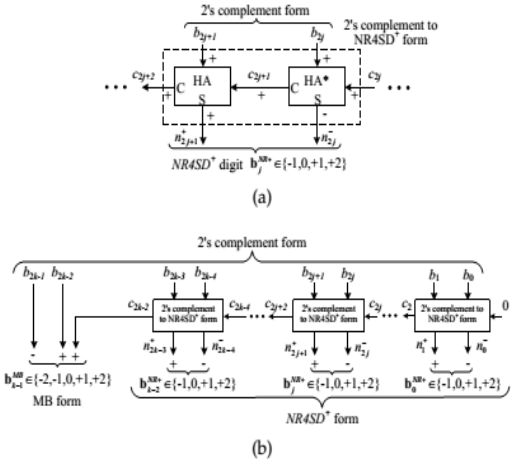


Fig. 3. Block Diagram of the NR4SD+ Encoding Scheme at the (a) Digit and (b) Word Level.

Calculate the value of the $\mathbf{b}_j^{NR+}$ digit.

Table 3 shows how the NR4SD digits are formed. The NR4SD encoding signals one_j^+, one_j^- and two_j^- of Table 3 are generated

TABLE 3
NR4SD⁺ Encoding

2's complement			NR4SD⁺ form			Digit	NR4SD⁺ Encoding		
b_{2j+1}	b_{2j}	c_{2j}	c_{2j+2}	n^+_{2j+1}	n^-_{2j}	$\mathbf{b}_j^{NR+}$	one_j^+	one_j^-	two_j^+
0	0	0	0	0	0	**0**	0	0	0
0	0	1	0	1	1	**+1**	1	0	0
0	1	0	0	1	1	**+1**	1	0	0
0	1	1	0	1	0	**+2**	0	0	1
1	0	0	0	1	0	**+2**	0	0	1
1	0	1	1	0	1	**-1**	0	1	0
1	1	0	1	0	1	**-1**	0	1	0
1	1	1	1	0	0	**0**	0	0	0

For the computation of the least and the most significant bits of the partial product we consider and respectively. Note that in case that , the number of the resulting partial products is and the most significant MB digit is formed based on sign extension of the initial 2's complement number.

After the partial products are generated, they are added, properly weighted, through a Carry-Save Adder (CSA) tree .

Finally, the carry-save output of the Wallace CSA tree is leaded to a fast Carry Look Ahead (CLA) adder to form the final result Z = X . Y .

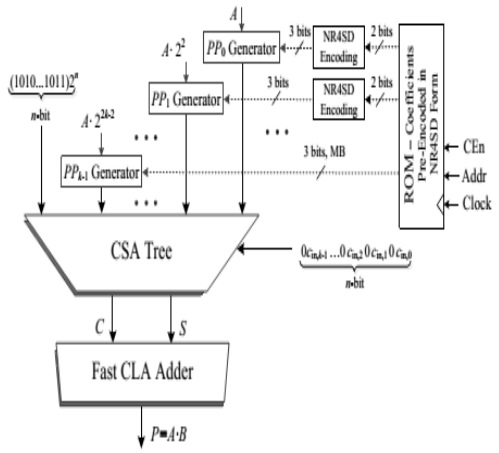


Fig.4 .System Architecture of the NR4SD Multipliers

In the pre-encoded MB multiplier scheme, the coefficient B is encoded off-line according to the conventional MB form (Table 1). The resulting encoding signals of B are stored in a ROM. The circled part of Fig. 3, which contains the ROM with coefficients in 2's complement form and the MB encoding circuit, is now totally replaced by the ROM .The MB encoding blocks of Fig. 3 are omitted. The new ROM is used to store the encoding signals of B and feed them into the partial product generators (P Pj Generators - PPG) on each clock cycle. Targeting to decrease switching activity, the value '1' of s j in the last entry of Table 1 is replaced by '0'. The sign s j is now given by the relation:

However, the ROM width is increased. Each digit requests three encoding bits (i.e., s, two and one (Table 1)) to be stored in the ROM. Since the n-bit coefficient B needs three bits per digit when encoded in MBform, the ROM width requirement is 3n/2 bits per coefficient. Thus, the width and the overall size of the ROM are increased by 50% compared to the ROM of the conventional scheme .

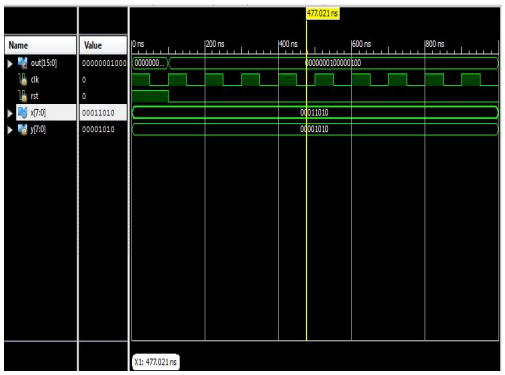
The system architecture for the pre-encoded NR4SD multipliers is presented in Fig. 6. Two bits are now stored in ROM: n2j+1, n+2j(Table 2) for the NR4SDor n+2j+1, n2j(Table 3) for the NR4SD+form. In this way, we reduce the memory requirement to +1 bits per coefficient while the corresponding memory required for the pre-encoded MB scheme is 3n/2 bits per coefficient. Thus, the amount of stored bits is equal to that of the conventional MB design, except for the most significant digit that needs an extra bit as it is MB encoded. Compared to the pre-encoded MB multiplier, where the MB encoding blocks are omitted, the pre-encoded NR4SD multipliers need extra hardware to generate the signals of (6) and (8) for the NR4SD and NR4SD+ form, respectively.

Each partial product of the pre-encoded NR4SD and NR4SD+ multipliers is implemented based on Fig. 4c and 4d, respectively, except for the P Pk 1 that corresponds to the most significant digit. As this digit is in MB form, we use the PPG of Fig. 4b applying the change mentioned in Section 4.2 for the s j bit. The partial products, properly weighted, and the correction term (COR) of (11) are fed into a CSA tree. The input carry cin;j of (11) is calculated as cin;j = twoj_ onej and cin;j = onej for the NR4SDand NR4SD+pre-encoded multipliers, respectively ,based on Tables 2 and 3. The carry-save output of the CSA tree is finally summed using a fast CLA adder.

V. RESULTS

The simulation of the program is done using ModelSim tool and Xilinx ISE Design Suite 13.2. The results for the multiplication of 4x4 and 8x8 using Modified Booth Multiplier is shown in this section.

Simulation results:



Design summary

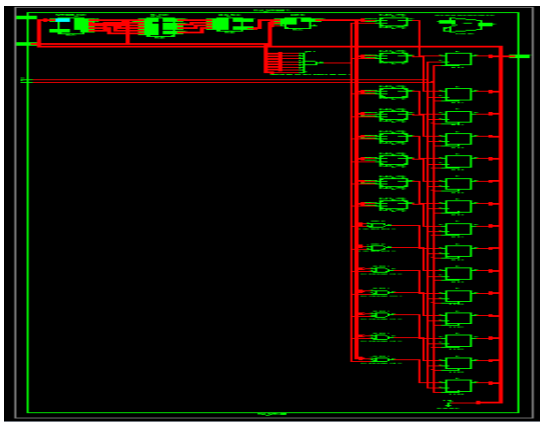
Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	120	4656	2%
Number of Slice Flip Flops	8	9312	0%
Number of 4 input LUTs	210	9312	2%
Number of bonded IOBs	34	232	14%
Number of GCLKs	1	24	4%

Timing report:

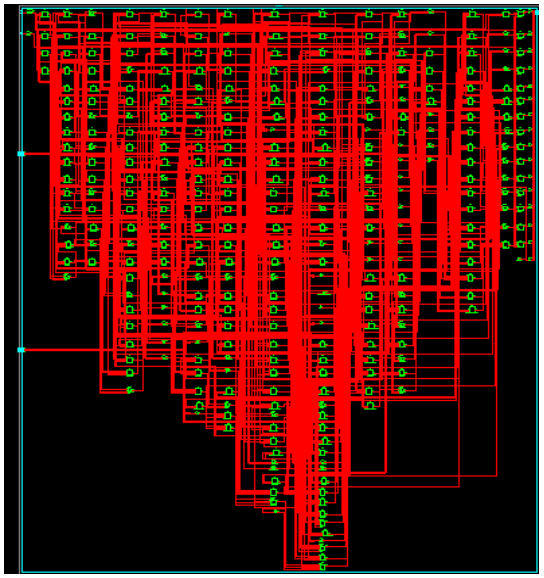
```
Offset:              4.040ns (Levels of Logic = 1)
  Source:            out_14 (FF)
  Destination:       out<14> (PAD)
  Source Clock:      clk rising

  Data Path: out_14 to out<14>
                               Gate     Net
    Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q             1    0.514    0.357  out_14 (out_14)
     OBUF:I->O                 3.169           out_14_OBUF (out<14>)
    ----------------------------------------
    Total                      4.040ns (3.683ns logic, 0.357ns route)
                                       (91.2% logic, 8.8% route)
```

RTL schematic:



Technology schematic:



VI. CONCLUSION

New designs of pre-encoded multipliers are explored by off-line encoding the standard coefficients and storing them in system memory. We propose encoding these coefficients in the Non-Redundant radix-4 Signed-Digit (NR4SD) form. The proposed pre-encoded NR4SD multiplier designs are more area and power efficient compared to the conventional and pre-encoded MB designs. Extensive experimental analysis verifies the gains of the proposed pre-encoded NR4SD multipliers in terms of area complexity and power consumption compared to the conventional MB multiplier.

REFERENCES

[1] D.J. Magenheimer, L. Peters, K.W. Pettis, and D. Zuras, "IntegerMultiplication and Division on the HP Precision Architecture,"IEEE Trans. Computers,vol. 37, no. 8, pp. 980-990, Aug. 1988.

[2] A.D. Booth, "A Signed Binary Multiplication Technique,"Quarterly J. Mechanical Applications of Math.,vol. IV, no. 2, pp. 236-240,1951.

[3] R. Bernstein, "Multiplication by Integer Constants,"Software—Practice and Experience,vol. 16, no. 7, pp. 641-652, July 1986.

[4] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices,"Proc. 16th IEEE Symp.Computer Arithmetic (ARITH 16),J. -C. Bajard and M. Schulte, eds.,pp. 20-27, June 2003.

[5] M. Potkonjak, M.B. Srivastava, and A.P. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,vol. 15, no. 2, pp. 151-165, Feb. 1996.

[6] M.D. Ercegovac and T. Lang,Digital Arithmetic.Morgan Kaufmann, 2003.

[7] M.J. Flynn and S.F. Oberman,Advanced Computer Arithmetic Design.Wiley-Interscience, 2001.

[8] R.I. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," IEEE Trans. Circuits and Systems II:Analog and Digital Signal Processing,vol. 43, no. 10, pp. 677-688,Oct. 1996.

[9] K.D. Chapman, "Fast Integer Multipliers Fit in FPGAs,"EDN Magazine,May 1994.

[10] S. Yu and E.E. Swartzlander, "DCT Implementation with Distributed Arithmetic,"IEEE Trans. Computers, vol. 50, no. 9, pp. 985-991, Sept. 2001.