International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

VLSI Design of a Novel Pre Encoding Multiplier Using DADDA Multiplier

G.V.S.M.Kumar, E. Adinarayana, Dr .V.S.R.Kumari,
[1]M.Tech Scholar, Dept. of ECE, Email Id: kumargvsm@gmail.com
[2]Associate Professor, Dept. of ECE, Email Id: adinarayanamtech@gmail.com
[3]Professor, Dept. of ECE, Email Id: vsrk46@gmail.com
[1,2,3]Sri Mittapalli College Of Engineering, Tummalapalem(V), Prathipadu(M), Guntur(Dt),A.P.

***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 NR4SD encoding is used mostly where CSA has taken the role of increasing the speed to add the partial products. In this NR4SD⁻ and NR4SD⁺ are used to reduce no of partial products. To further implement the*
Performance of the multiplier we are using the DADDA multiplier. 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 Simulation.

Keywords: Computer arithmetic, multiplication by constants, arithmetic circuits, , VLSI design.

I. INTRODUCTION

Multiplier plays an important role for performing the arithmetic operations in both Digital Signal Processors and Microprocessors. In order to enhance the performance characteristics of either the D.S.P. or the Microprocessors, the efficient and effective Multiplication Algorithm has to be adopted[2],[3],[6]. In digital systems, the multiplier are the basic blocks. The Multipliers also contribute to the Computational speed and Power consumption of the digital system. So, the need for designing High speed Multiplier with minimal power dissipation is very crucial for a digital system[8]. Thus, it can boost the efficiency of the Digital systems.

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[10]. 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[9]. 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[5],[1]. 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 one most effective way to increase the speed of a multiplier is to reduce the number of the partial products[6]. 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[7]. 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.

II. Existing system

A)NR4SD⁻ Encoding Scheme

The following Boolean equations summarize the HA* operation:

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

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

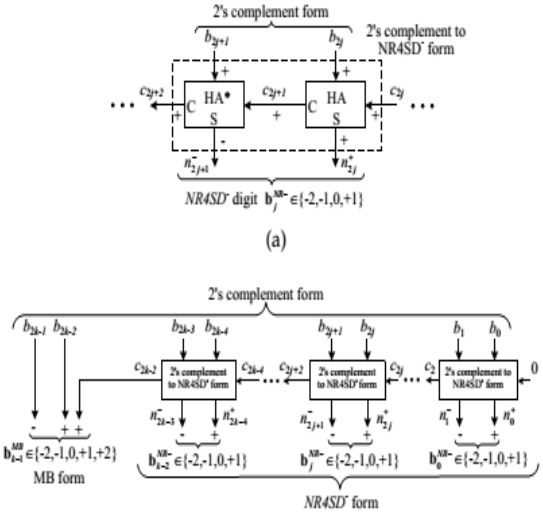


Fig. 1. Block Diagram of the NR4SD⁻ Encoding Scheme at the (a) Digit and (b) Word Level.

Table 1
HA* Operation

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

$$c = p \vee q,\ s = p \oplus q \qquad (2)$$

Calculate the value of the b_j^{NR-}digit

$$b_j^{NR-} = -2n_{2j+1}^{-} + n_{2j}^{+} \qquad (3)$$

Table2
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}^{+}	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

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.

B)NR4SD⁺ Encoding Scheme

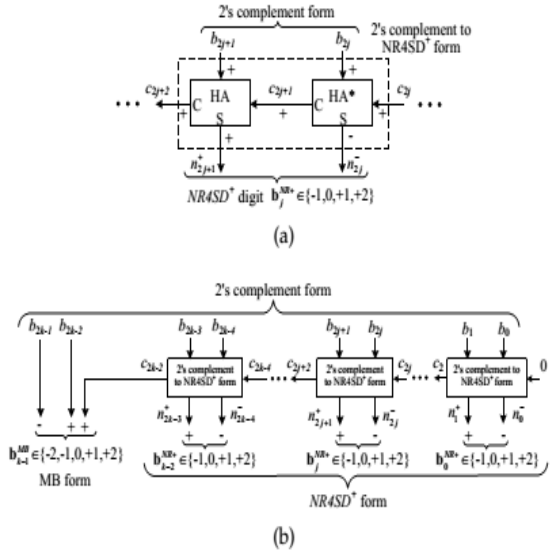


Fig. 2. Block Diagram of the NR4SD⁺ Encoding Scheme at the (a) Digit and (b) Word Level.

Calculate the value of the b_j^{NR+}digit

$$b_j^{NR+} = 2n_{2j+1}^{+} - n_{2j}^{-} \qquad (4)$$

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}^{-}	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[7]. Note in that case , 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[8] .

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

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 .

In the pre-encoded MB multiplier scheme, the coefficient B is encoded off-line according to the conventional MB form. The resulting encoding signals of B are stored in a ROM[1],[2]. The circled part, 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 are omitted. The new ROM is used to store the encoding signals of B and feed them into the partial product generators (PPj 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: $s_j = b_{2j+1} \oplus (b_{2j+1} \wedge b_{2j} \wedge b_{2j-1})$

However, the ROM width is increased. Each digit requests three encoding bits (i.e., s, two and one) to be stored in the ROM[7]. Since the n-bit coefficient B needs three bits per digit when encoded in MB form, 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. Two bits are now stored in ROM: n^-_{2j+1}, n^+_{2j} (Table 2) for the NR4SD or n^+_{2j+1}, n^-_{2j} (Table 3) for the NR4SD$^+$form. In this way, we reduce the memory requirement to n +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 for the NR4SD$^-$ and NR4SD$^+$ form, respectively.

III. Proposed System

To further reduce the delay we are using the Dadda multiplier for the generation of partial products and final product in the existing system. The Dadda multiplier is a hardware multiplier design invented by computer scientist Luigi Dadda in 1965. It is similar to the Wallace multiplier, but it is slightly faster and requires fewer gates.

Dadda multipliers have the same 3 steps for two bit strings w1 and w2 of lengths l1 and l2 respectively:

1.Multiply (logical AND) each bit of , by each bit of , yielding results, grouped by weight in columns
2.Reduce the number of partial products by stages of full and half adders until we are left with at most two bits of each weight.
3.Add the final result with a conventional adder.

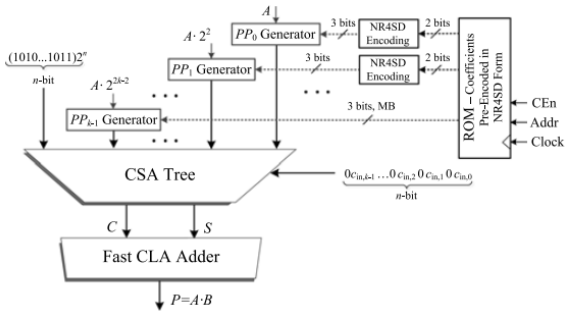


Fig : 3 System architecture of the pre encoded multipliers Using Dadda Multiplier.

The multiplication products of the first step carry different weights reflecting the magnitude of the original bit values in the multiplication. Dadda multipliers attempt to minimize the number of gates used, as well as input/output delay. Because of this, Dadda multipliers have a less expensive reduction phase, but the final numbers may be a few bits longer, thus requiring slightly bigger adders.

In this Dadda multipliers generated partial products are first applied to the partial product reduction tree. The progression of the reduction is controlled by a maximum-height sequence d_j.

$D_{j+1} = \text{floor}(1.5 * d_j)$

This yields a sequence like so: $d_1 = 2$, $d_2=3$, $d_3=4$, $d_4=6$, $d_5=9$, $d_6=13$,...

The initial value of j is chosen as the largest value such that $d_j < \min(n_1,n_2)$ where n_1 and n_2 are the

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

number of bits in the input multiplicand and multiplier. The lesser of the two bit lengths will be the maximum height of each column of weights after the first stage of multiplication.

IV. RESULTS

The simulation of the program is done using ISim Simulator tool and synthesis is done using Xilinx ISE Design Suite 14.5. The results for the multiplication 8x8 using Dadda multiplier is shown in this section.

Simulation results:

Design summary

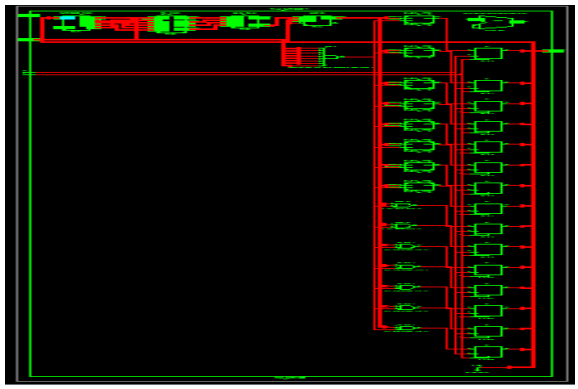
Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	120	4656	2%
Number of Slice Flip Flops	8	9312	0%
Number of 4 input LUTs	210	9312	2%
Number of bonded IOBs	34	232	14%
Number of GCLKs	1	24	4%

Timing report:

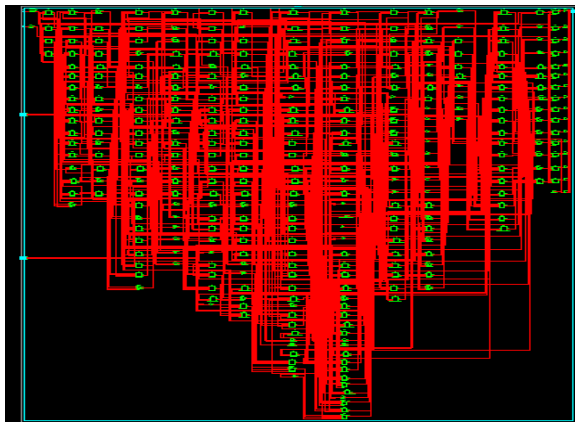
```
Offset:              4.040ns (Levels of Logic = 1)
  Source:            out_14 (FF)
  Destination:       out<14> (PAD)
  Source Clock:      clk rising

  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:



V. CONCLUSION

New designs of pre-encoded multipliers are explored by off-line encoding the standard coefficients and storing them in system memory. One of the input in the multiplier is encoded using Non-Redundant radix-4 Signed-Digit (NR4SD) form. That encoded input is used for the generation of partial products using Dadda multiplier. The proposed pre-encoded Dadda Multipliers are compared with conventional multiplier designs.

VI.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.