An optimized implementation of Pre-Encoded Multipliers Based on NR4SD Encoding technique for DSP/Multimedia applications

B. Mounika[1] boddupallymounika93@gmail.com[1]

DR.D.Nageshwarrao[2] deshmukhnag@gmail.com[2]

[1]PG Scholar, Dept of ECE, TKR College of Engineering and Technology, Medbowli, Saroornagar Rangareddy, Telangana, India

[3]Associate Professor, HOD,Dept of ECE, TKR College of Engineering and Technology, Medbowli, Saroornagar Rangareddy, Telangana, 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 due to using of SPST architecture. 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

In this paper, we study the various parallel MAC architectures and then implement a design of parallel MAC based on some booth encodings such as radix-2 booth encoder and some final adders such as CLA, Kogge stone adder and then compare their performance characteristics. In general, a multiplier uses Booth algorithm and an array of full adders, this multiplier mainly consists of three parts Wallace tree, to add partial products, booth encoder and final adder. A Digital multiplier is the fundamental component in general purpose microprocessor and in DSP [1]. 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. An architecture was proposed in [2] to provide the tact to merge the final adder block to the accumulator register in the MAC operator to provide the possibility of using two separate N/2-bit adders instead of one N-bit adder to accumulate the N–bit MAC results. The most advanced types of MAC has been proposed by Elguibaly in which accumulation has been combined with the carry save adder (CSA) tree that compresses partial products and thus reduces the critical path. While it has better performance as compared to the previous MAC architectures. Later on a new architecture for a high-speed MAC is proposed by Seo and Kim. The difference between the two is that the latest one carries out the accumulation by feeding back the final CSA output rather than the final adder results.

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 in terpreted 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. 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.

High-Speed Booth Encoded Parallel Multiplier Design:

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.

III. Proposed System

A. Motivation

In this paper, we focus on AM units which implement the operation Z=X.(A+B). The conventional design of the AM operator (Fig. 1(a)) 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 data path block (Fig. 1(b)) by direct recoding of the sum Y=A+B to its MB representation. The fused Add-Multiply (FAM) component contains only one adder at the end (final adder of the parallel multiplier). As a result, significant 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. Let us consider the multiplication of 2's complement numbers X and Y with each number consisting of $n = 2k$ bits. .The multiplicand Y can be represented in MB form as:

$$Y \quad \langle y_{n-1}y_{n-2}\dots y_1y_0\rangle_{2's} \quad y_{2k-1}\cdot 2^{2k-1}+\sum_{i\ 0}^{2k\ 2} y_i\cdot 2^i$$

$$-\langle \mathbf{y}_{k\ 1}^{MB}\mathbf{y}_{k\ 2}^{MB}\dots \mathbf{y}_1^{MB}\mathbf{y}_0^{MB}\rangle_{MB}-\sum_{j\ 0}^{k\ 1}\mathbf{y}_j^{MB}\cdot 2^{2j} \qquad (1)$$

$$\mathbf{y}_j^{MB}=-2y_{2j\ 1}+y_{2j}-y_{2j\ 1}. \qquad (2)$$

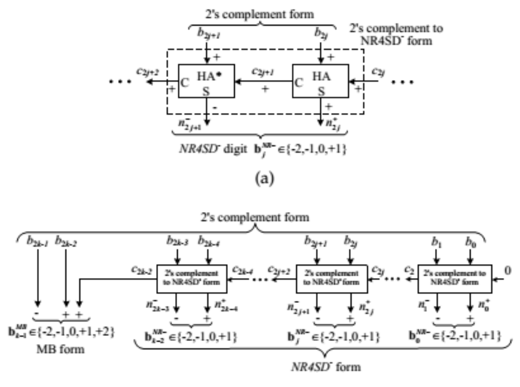


Fig. 1. Block Diagram of the NR4SD$^-$ Encoding Scheme at the (a) Digit and (b) Word Level.

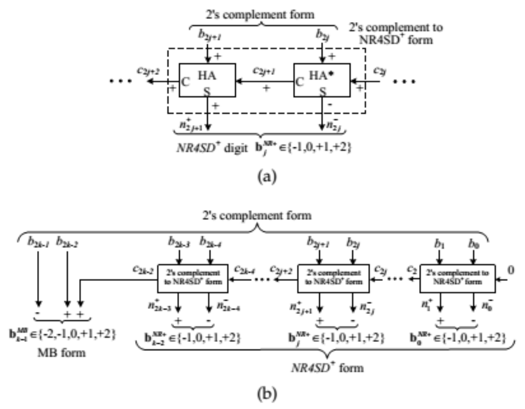


Fig. 2. Block Diagram of the NR4SD+ Encoding Scheme at the (a) Digit and (b) Word Level.

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

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

TABLE I
MODIFIED BOOTH ENCODING TABLE.

Binary			$\mathbf{y}_j^{MB}$	*MB Encoding*			*Input Carry*
y_{2j+1}	y_{2j}	y_{2j-1}		sign=s_j	×1=one_j	×2=two_j	$c_{in,j}$
0	0	0	**0**	0	0	0	0
0	0	1	**+1**	0	1	0	0
0	1	0	**+1**	0	1	0	0
0	1	1	**+2**	0	0	1	0
1	0	0	**-2**	1	0	1	1
1	0	1	**-1**	1	1	0	1
1	1	0	**-1**	1	1	0	1
1	1	1	**0**	1	0	0	0

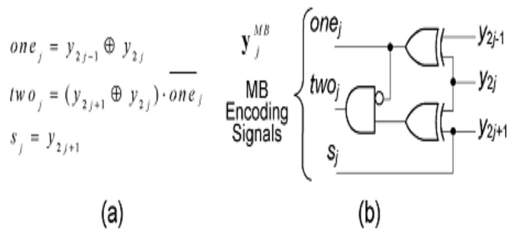


Fig. 2. (a) Boolean equations and (b) gate-level schematic for the implementation of the MB encoding signals.

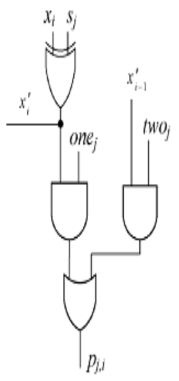


Fig. 3. Generation of the i-th bit $p_{j,i}$ of the partial product PP_j for the conventional MB multiplier.

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 Wallace Carry-Save Adder (CSA) tree along with the Correction Term (CT) which is given by the following equations:

$$Z = X \cdot Y = CT + \sum_{j=0}^{k-1} PP_j \cdot 2^{2j}$$

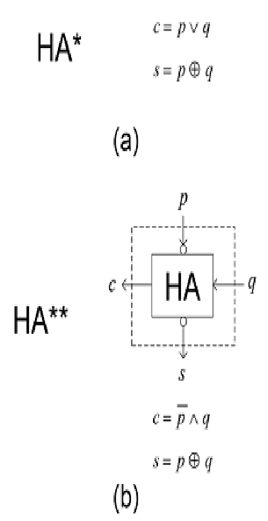


Fig. 4. Boolean equations and schematics for signed (a) HA* and (b) HA**

$$CT = CT(low) + CT(high) =$$
$$= \sum_{j=0}^{k-1} c_{in,j} \cdot 2^{2j} + 2^{n} \left(1 + \sum_{j=0}^{k-1} 2^{2j+1} \right) \qquad ($$

where $c_{in,j} = (one_j \vee two_j) \wedge s_j$ (see Table I).

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 as shown in Fig.1 (b).

IV. PRE-ENCODED MULTIPLIERS DESIGN

A. Conventional MB Multiplier

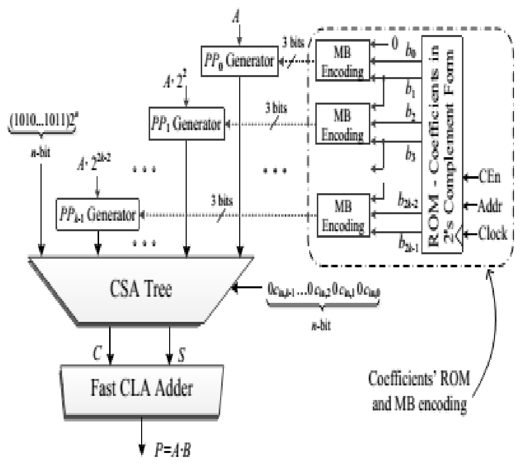


Fig. 3. System Architecture of the Conventional MB Multiplier.

Fig. 3 presents the architecture of the system which comprises the conventional MB multiplier and the ROM with coefficients in 2’s complement form. Let us consider the multiplication A.B. The coefficient B = (b n-1……b0)s consists of n=2k bits and is driven to the MB encoding blocks from a ROM where it is stored in 2’s complement form. It is encoded according to the MB algorithm (Section 2) and multiplied by A = (an …. a0) , which is in 2’s complement representation. We note that the ROM data bus width equals the width of coefficient B (n bits) and that it outputs one coefficient on each clock cycle.The k partial products are generated as follows:

$$PP_j = A\cdot \mathbf{b}_j^{MB} = \bar{p}_{j,n}2^n + \sum_{i=0}^{n-1} p_{j,i}2^i.$$

The generation of the ith bit pj;i of the partial product P Pj is illustrated at gate level in Fig. 4a For the computation of the least and most significant bits of P Pj , we consider a 1=0 and an = an 1, respectively. After shaping the partial products, they are added, properly weighted, through a Carry Save Adder (CSA) tree along with the correction term (COR):

$$P = A\cdot B = COR + \sum_{j=0}^{k-1} PP_j 2^{2j},$$

$$COR = \sum_{j=0}^{k-1} c_{in,j}2^{2j} + 2^n(1 + \sum_{j=0}^{k-1} 2^{2j+1}),$$

Where cin;j = (onej _twoj)^sj (Table 1). The CS outputof the tree is leaded to a fast Carry Look Ahead (CLA) adder to form the final result P = A B (Fig. 3).

Pre-Encoded MB Multiplier Design 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 of Fig. 5. The MB encoding blocks of Fig. 3 are omitted. The new ROM of Fig. 5 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:

$$s_j = b_{2j+1} \oplus (b_{2j+1} \wedge b_{2j} \wedge b_{2j-1}).$$

As a result, the PPG of Fig. 4a is replaced by the one of Fig. 4b. Compared to (4), (12) leads to a more complex design. However, due to the pre-encoding technique, there is no area / delay overhead at the circuit.

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 computed as cin;j = sj based on (12) and Table 1. The CS output of the tree is finally merged by a fast CLA adder.

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 (Fig. 3).

B. Pre-Encoded NR4SD Multipliers Design

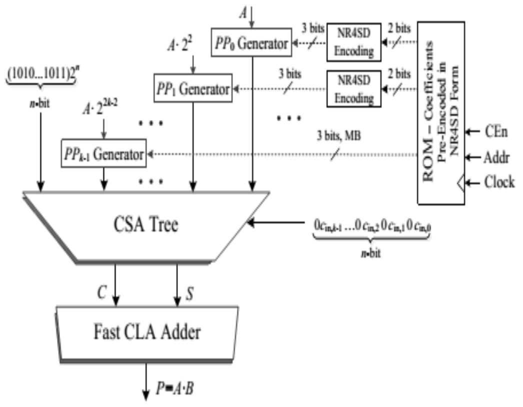


Fig.4 System Architecture of the NR4SD Multipliers

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. The NR4SD encoding blocks of Fig. 6 implement the circuitry of Fig. 7.

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.

TABLE III
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] $OutputValue = -2\cdot c + s = -p - q$

TABLE IV
HA** OPERATION.

Inputs		Output Value[3]	Outputs	
p (-)	q (+)		c (+)	s (-)
0	0	0	0	0
0	1	+1	1	1
1	0	-1	0	1
1	1	0	0	0

[3] $OutputValue = 2\cdot c - s = -p + q$

TABLE V
FA* OPERATION.

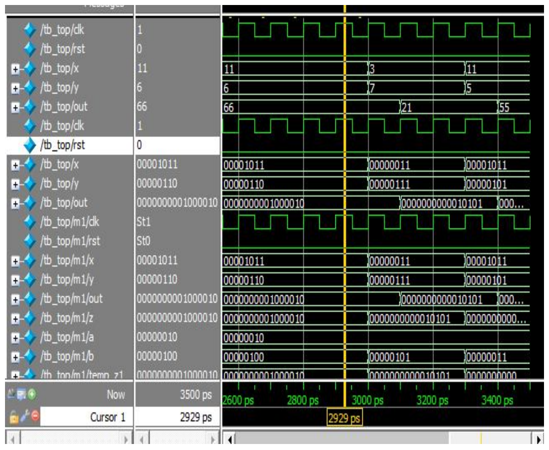
Inputs			Output Value[1]	Outputs	
p (+)	q (-)	c_i (+)		c_o (+)	s (-)
0	0	0	0	0	0
0	0	1	+1	1	1
0	1	0	-1	0	1
0	1	1	0	0	0
1	0	0	+1	1	1
1	0	1	+2	1	0
1	1	0	0	0	0
1	1	1	+1	1	1

[1] $OutputValue = 2\cdot c_o - s = p - q + c_i$

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.

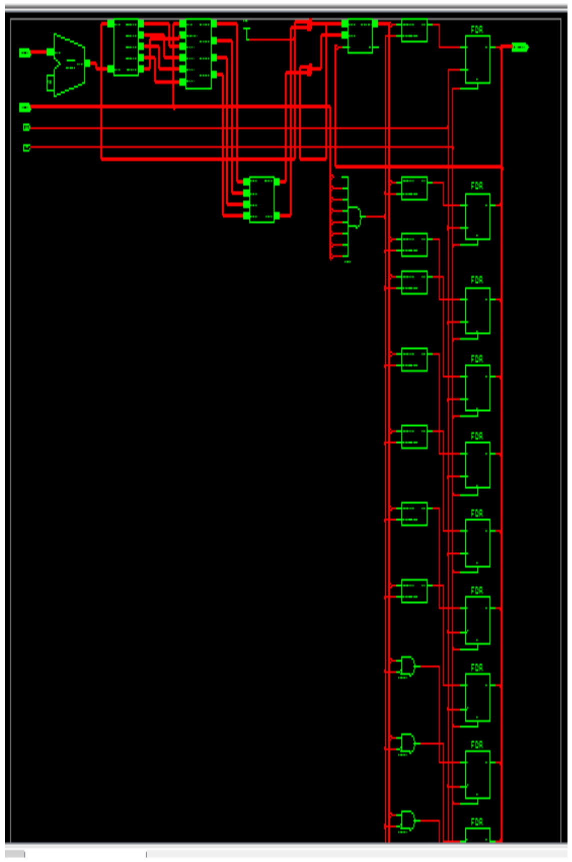
Simulation Result



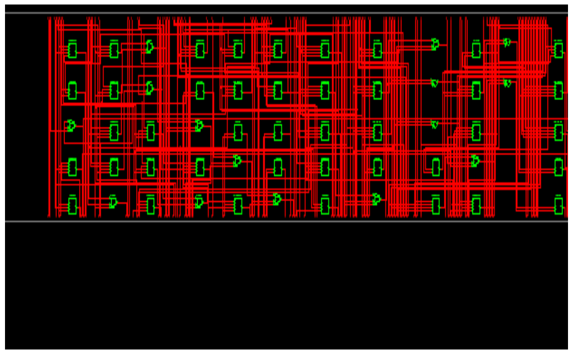
```
/tb_top/clk        1
/tb_top/rst        0
/tb_top/x          11                    11                 3                  11
/tb_top/y          6                     6                  7                  5
/tb_top/out        66                    66                 21                 55
/tb_top/clk        1
/tb_top/rst        0
/tb_top/x          00001011              00001011           00000011           00001011
/tb_top/y          00000110              00000110           00000111           00000101
/tb_top/out        0000000001000010      0000000001000010   0000000000010101   000...
/tb_top/m1/clk     St1
/tb_top/m1/rst     St0
/tb_top/m1/x       00001011              00001011           00000011           00001011
/tb_top/m1/y       00000110              00000110           00000111           00000101
/tb_top/m1/out     0000000001000010      0000000001000010   0000000000010101   000...
/tb_top/m1/z       0000000001000010      0000000001000010   0000000000010101   0000000000...
/tb_top/m1/a       00000010              00000010
/tb_top/m1/b       00000100              00000100           00000101           00000011
/tb_top/m1/temp_z1 0000000001000010      0000000001000010   0000000000010101   0000000000

Now       3500 ps    2600 ps   2800 ps   3000 ps   3200 ps   3400 ps
Cursor 1  2929 ps    2929 ps
```

Synthesis Results:

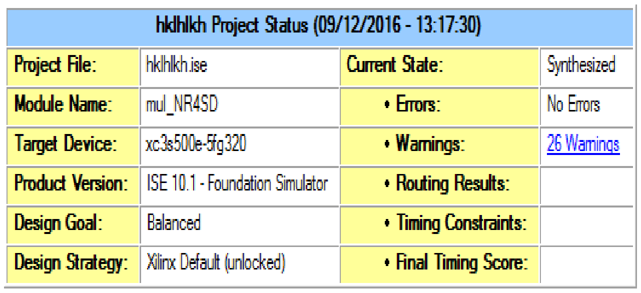
RTL schematic:



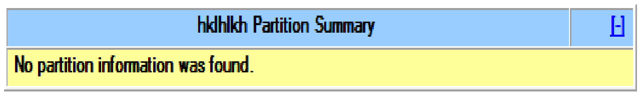
Technology Schematic:



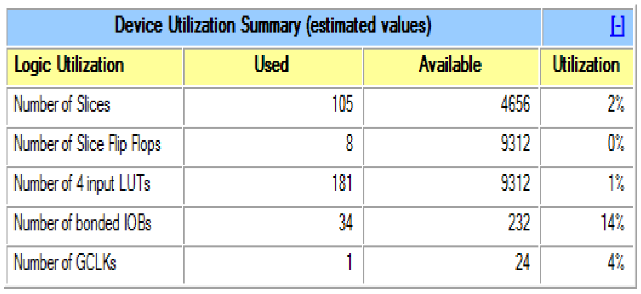
Design Summary:



hklhlkh Project Status (09/12/2016 - 13:17:30)			
Project File:	hklhlkh.ise	**Current State:**	Synthesized
Module Name:	mul_NR4SD	**• Errors:**	No Errors
Target Device:	xc3s500e-5fg320	**• Warnings:**	26 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	



hklhlkh Partition Summary
No partition information was found.



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	105	4656	2%
Number of Slice Flip Flops	8	9312	0%
Number of 4 input LUTs	181	9312	1%
Number of bonded IOBs	34	232	14%
Number of GCLKs	1	24	4%

VI. CONCLUSION

In this study, new signed-digit two operand and multi operand decimal adders are proposed. Performance of recent decimal adder architectures have been investigated and compared. The proposed SD adder excels in speed and area usage among previously proposed SD adders. The use of constant addition for speculation and the merging of adjacent modules with sharable operations enable efficient implementation of two operand and multi-operand decimal addition.

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.

[11] P. Boonyanant and S. Tantaratana, "FIR Filters with Punctured Radix-8 Symmetric Coefficients: Design and Multiplier-Free Realizations,"Circuits Systems Signal Processing,vol. 21, no. 4, pp. 345-367, 2002.

[12] C.K.S. Pun, S.C. Chan, K.S. Yeung, and K.L. Ho, "On the Design and Implementation of FIR and IIR Digital Filters with Variable Frequency Characteristics,"IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing,vol. 49, no. 11, pp. 689-703,
Nov. 2002.

[13] S.C. Chan and W.L.K.L. Ho, "Multiplierless Perfect Reconstruction Modulated Filter Banks with Sum-of-Powers-of-Two Coefficients," Signal Processing Letters, IEE,vol. 8, no. 6, pp. 163-166, 2001.

[14] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, "Theory and Applications of the Double-Base Number System,"IEEE Trans. Computers,vol. 48, no. 10, pp. 1098-1106, Oct. 1999.

[15] J. Li and S. Tantaratana, "Multiplier-Free Realizations for FIR Multirate Converters Based on Mixed-Radix Number Representation,"IEEE Trans. Signal Processing,vol. 45, no. 4, pp. 880-890, Apr. 1997.

[16] N. Homma, T. Aoki, and T. Higuchi, "Evolutionary Graph Generation System with Transmigration Capability and Its Application to Arithmetic Circuit Synthesis,"IEE Proc., vol. 149, no. 2, pp. 97-104, Apr. 2002.