Design and Implementation of FAM based Optimized Modified Booth Recoder

Y.Ramesh[1], E.Adinarayana[2]
M.Tech Scholar, Dept. of ECE, Sri Mittapalli College Of Engineering, Guntur[1]
Associate Professor Dept. of ECE, Sri Mittapalli College Of Engineering, Guntur[2]
Email: Email: yrameshle4@gmail.com[1], adinarayanamtech@gmail.com[2]

Abstract- 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. This paper aims at additional reduction of latency and area of the Wallace tree multiplier. This is accomplished by the use of Booth algorithm and compressor adders. The coding is done in Verilog HDL and synthesized for Xilinx Virtex 6 FPGA device.

Keywords: Add-Multiply operation, arithmetic circuits, Modified Booth recoding, VLSI design, Arithmetic, Booth Encoder, Compressors, Radix-8, and Wallace-Tree.

1. INTRODUCTION

A multitude of various multiplier architectures have been published in the literature, during the past few decades. The multiplier is one of the key hardware blocks in most of the digital and high performance systems such as digital signal processors and microprocessors. With the recent advances in technology, many researchers have worked on the design of increasingly more efficient multipliers. They aim at offering higher speed and lower power consumption even while occupying reduced silicon area. This makes them compatible for various complex and portable VLSI circuit implementations [2].However, the fact remains that the area and speed are two conflicting performance constraints. Hence, innovating increased speed always results in larger area. The proposed architecture enhances the speed performance of the widely acknowledged Wallace tree multiplier when implemented on a FPGA. The structural optimization is performed on the conventional Wallace multiplier, in such a way that the latency of the total circuit reduces considerably. The conventional Wallace tree multiplier architecture comprises of an AND array for computing the partial products, a carry save adder for adding the partial products so obtained and a carry propagate adder in the final stage of addition. In the proposed architecture, partial product generation and reduction is accomplished by the use of Booth recoding algorithm, 3:2, 4:2,and 5:2 compressor structures. Modern consumer electronics make extensive use of Digital Signal Processing (DSP) providing custom accelerators for the domains of multimedia, communications etc. Typical DSP applications carry out a large number of arithmetic operations as their implementation is based on computationally intensive kernels, such as Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), Finite Impulse Response (FIR) filters and signals’ convolution. As expected, the performance1 of DSP systems is inherently affected by decisions on their design regarding the allocation and the architecture of arithmetic units. Recent research activities in the field of arithmetic optimization [1], [2] have shown that the design of arithmetic components combining operations which share data, can lead to significant performance improvements. Based on the observation that an addition can often be subsequent to a multiplication (e.g., in symmetric FIR filters), the Multiply-Accumulator (MAC) and Multiply-Add (MAD) units were introduced [3] leading to more efficient implementations of DSP algorithms compared to the conventional ones, which use only primitive resources [4]. Several architectures have been proposed to optimize the performance of the MAC operation in terms of area occupation, critical path delay or power consumption [5]–[7]. As noted in [8], MAC components increase the flexibility of DSP data path synthesis as a large set of arithmetic operations can be efficiently mapped onto them. Except the MAC/MAD operations, many DSP applications are based on Add-Multiply (AM) operations (e.g., FFT algorithm [9]). The straightforward design of the AM unit, by first allocating an adder and then driving its output to the input of a multiplier, increases significantly both area and critical path delay of the circuit. Targeting an optimized design of AM operators, fusion techniques [10]–[13], [23] are employed based on the direct recoding of the sum of two numbers (equivalently a number in carry-save representation [14]) in its Modified Booth (MB) form [15]. Thus, the carry-propagate (or carry-look-ahead) adder [16] of the conventional AM design is eliminated resulting in considerable gains of performance. Lyu and Matula [10] presented a signed bit MB recoder which transforms redundant binary inputs to their MB recoding form. A special expansion of the preprocessing step of the recoder is needed in order to handle operands in carry-save representation. In [12], the author proposes a two-stage recoder which converts a number in carry-save form to its MB representation. The first stage transforms the carry-save form of the input number into signed-digit form which is then recoded in the second stage so that it matches the form that the MB digits request. Recently, the technique of [12] has been used for the design of high performance flexible coprocessor architectures targeting the computationally intensive DSP applications [17].

Zimmermann and Tran [13] present an optimized design of [10] which results in improvements in both area and critical path. In [23], the authors propose the recoding of a redundant input from its carry-save form to the corresponding borrow-save form keeping the critical path of the multiplication operation fixed. Although the direct recoding of the sum of two numbers in its MB form leads to a more efficient implementation of the fused Add-Multiply (FAM) unit compared to the conventional one, existing recoding schemes are based on complex manipulations in bit-level, which are implemented by dedicated circuits in gate-level. This work focuses on the efficient design of FAM operators, targeting the optimization of the recoding scheme for direct shaping of the MB form of the sum of two numbers (Sum to MB – *S-MB*). More specifically, we propose a new recoding technique which decreases the critical path delay and reduces area and power consumption. The proposed *S-MB* algorithm is structured, simple and can be easily modified in order to be applied either in signed (in 2's complement representation) or unsigned numbers, which comprise of odd or even number of bits. We explore three alternative schemes of the proposed *S-MB* approach using conventional and signed-bit Full Adders (FAs) and Half Adders (HAs) as building blocks.

We evaluated the performance of the proposed *S-MB* technique by comparing its three different schemes with the state-of the art recoding techniques [12], [13], [23]. Industrial tools for RTL synthesis [18] and power estimation [19] have been used to provide accurate measurements of area utilization, critical path delay and power dissipation regarding various bit-widths of the input numbers. We show that the adoption of the proposed recoding technique delivers optimized solutions for the FAM design enabling the targeted operator to be timing functional (no timing violations) for a larger range of frequencies. Also, under the same timing constraints, the proposed designs deliver improvements in both area occupation and power consumption, thus outperforming the existing *S-MB* recoding solutions.

2. EXISTING STRUCTURE

To generate and reduce the number of partial products of multiplier, modified Booth Algorithm has been used, In the modified Booth Algorithm, multiplier has been divided in groups of 4 bits and each groups of 4 bits have been operational according to modified Booth Algorithm for generation of partial products 0, ±1A, ±1A, ±2A, ±2A, ±3A, ±3A, ±4A. [2] These partial products are summed using compressors in structure of Wallace Tree. [1] In radix-8 Booth Algorithm, multiplier operand B is Partitioned into 11 groups having each group of 4 bits. In first group, first bit is taken zero and other bits are least significant three bit of multiplier operand. In second group, first bit is most significant bit of first group and other bits are next three bit of multiplier operand. In third group, first bit is most significant bit of second group and other bits are next three bits of multiplier operand. This process is carried on. For each group, Partial product is generated using multiplicand operand A. For n bit multiplier there is n/3 or [n/3 + 1] groups and partial products in proposed modified Booth Algorithm radix-8. Table I shows the algorithm of radix-8 proposed architecture. Here we need to increment the shift value by 3 after each step. Y is the multiplicand. We can also reduce the number of partial products using a higher radix (radix-16) in the multiplier recoding; its operation table is shown below, thereby obtaining a simpler Wallace tree. This implies a less delay through the compressors and a smaller active area size. In the other hand, we will need some multiples of the multiplicand which are not immediately available, but are generated by a previous adder, making worse the overall multiplication time.

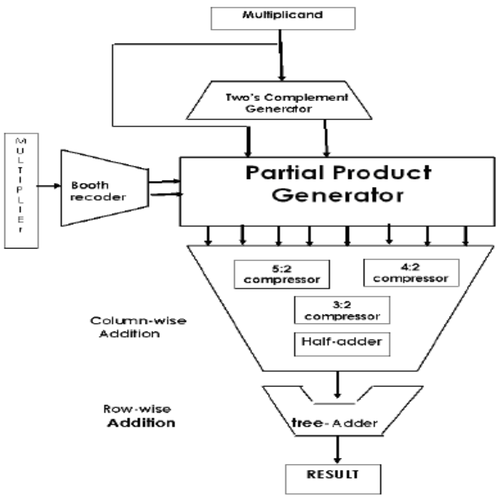


Fig. 1 Existing architecture

TABLE I. RADIX-8 BOOTH ENCODER

Multiplier bits	Operation for group
0000	0
0001	(Y << shft)
0010	(Y << shft)
0011	(Y << (shft +1))
0100	(Y << (shft+1))
0101	(Y<<(shft+1)) + (Y << (shft))
0110	(Y<<(shft+1)) + (Y << (shft))
0111	(Y<<(shft+2))
1000	(((~Y)+1)<<(shft+2))
1001	(((~Y)+1)<<(shft+1)) + (((~Y)+1) << (shft))
1010	(((~Y)+1)<<(shft+1)) + (((~Y)+1) << (shft))
1011	(((~Y)+1)<<(shft+1))
1100	(((~Y)+1)<<(shft+1))
1101	(((~Y)+1)<< shft)
1110	(((~Y)+1)<< shft)
1111	0

3. FAM IMPLEMENTATION

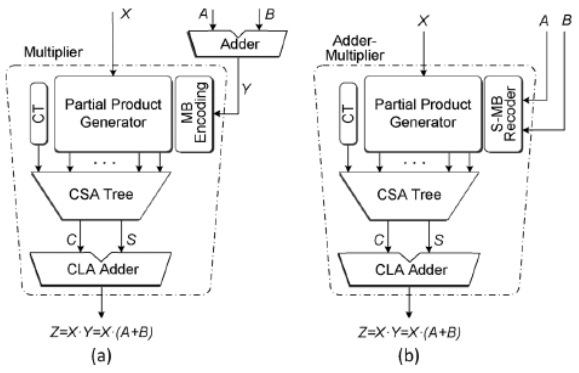


Fig.2 AM operator based on the (a) conventional design and (b) fused design with direct recoding of the sum of A and B in its MB representation. The multiplier is a basic parallel multiplier based on the MB algorithm. The terms CT, CSA Tree and CLA Adder are referred to the Correction Term, the Carry-Save Adder Tree and the final Carry-Look-Ahead Adder of the multiplier.

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

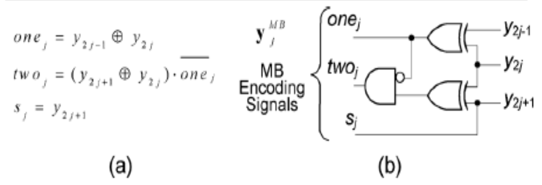


Fig. 3 (a) Boolean equations and (b) gate-level schematic for the implementation of the MB encoding signals

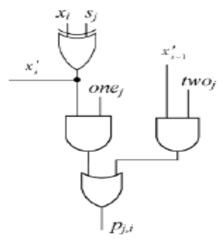


Fig. 4 Generation of the ith bit Pji of the partial product PPj for the conventional MB multiplier

In the FAM design presented in Fig. 2(b), the multiplier is a parallel one based on the MB algorithm. Let us consider the product X.Y. Both X and Y consist of n=2k bits and are in 2's complement form.

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

The generation of the i-th bit Pj,i of the partial product PPj is based on the next logical expression while Fig. 4 illustrates its implementation at gate level [20], [24]:

$$p_{j,i} = ((x_i \oplus s_j) \wedge one_j) \vee ((x_{i-1} \oplus s_j) \wedge two_j).$$

After the partial products are generated, they are added, properly weighted, through a Wallace Carry-Save Adder (CSA) tree [21] along with the Correction Term (CT) which is given by the following equations:

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

4. SUM TO MODIFIED BOOTH RECODING TECHNIQUE (*S-MB*)

4.1 Defining Signed-Bit Full Adders and Half Adders for Structured Signed Arithmetic

In *S-MB* recoding technique, we recode the sum of two consecutive bits of the input A with two consecutive bits of the input B into one MB digit yj^MB. As we observe from (2), three bits are included in forming a MB digit. The most significant of them is negatively weighted while he two least significant of them has positive weight. Consequently, in order to transform the two aforementioned pairs of bits in MB form we need to use signed-bit arithmetic. For this purpose, we develop a set of bit-level signed Half Adders (HA) and Full Adders (FA) considering their inputs and outputs to be signed.

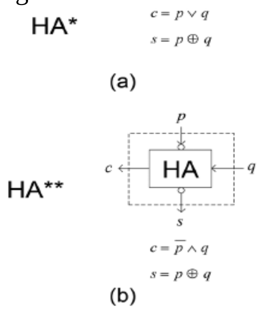


Fig. 5 Boolean equations and schematics for signed (a) HA* and (b) HA**

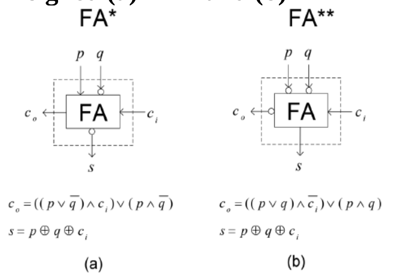

FA*
FA**
FA
$c_o = ((p \vee \overline{q}) \wedge c_i) \vee (p \wedge \overline{q})$
$s = p \oplus q \oplus c_i$
$c_o = ((p \vee q) \wedge \overline{c_i}) \vee (p \wedge q)$
$s = p \oplus q \oplus c_i$
(a)
(b)

Fig. 6 Boolean equations and schematics for signed (a) FA* and (b) FA**

4.2 Proposed S-MB Recoding Technique

We use both conventional and signed HAs and FAs in order to design and explore three new alternative schemes of the *S-MB* recoding technique. Each of the three schemes can be easily applied in either signed (2's complement representation) or unsigned numbers which consist of odd or even number of bits.

1) *S-MB1 Recoding Scheme:* The first scheme of the proposed recoding technique is referred as *S-MB*1 and is illustrated in detail in Fig. 6 for both even (Fig. 6(a)) and odd (Fig. 6(b)) bit-width of input numbers. As can be seen in Fig. 6, the sum of and is given by the next relation:

$$Y = A + B = \mathbf{y}_k \cdot 2^{2k} + \sum_{j=0}^{k-1} \mathbf{y}_j^{MB} \cdot 2^{2j}$$

where $\mathbf{y}_j^{MB} = -2s_{2j+1} + s_{2j} + c_{2j}.$

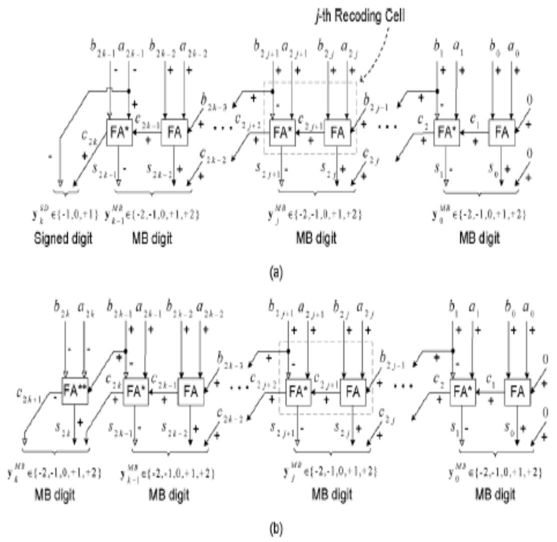


Fig. 7 *S-MB*1 recoding scheme for (a) even and (b) odd number of bits

TABLE II
HA* BASIC OPERATION.

Inputs		*Output Value*[1]	*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

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

The critical path delay of *S-MB*1 recoding scheme (Fig. 7) is constant in respect to the input bit-width and is given by the equation:

$$T_{S-MB1} = T_{FA,carry} + T_{FA^*,sum}$$

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

TABLE VI
FA** OPERATION.

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

[2] *Output Value* $= -2 \cdot c_o + s = -p - q + c_i$

2) *S-MB2 Recoding Scheme:* The second approach of the proposed recoding technique, *S-MB*2, is described in Fig. 7 for even (Fig. 8(a)) and odd (Fig. 8(b)) bit-width of input numbers. We consider the initial values C0,1=0 and C0,2=0.

The critical path delay of *S-MB*2 recoding scheme is calculated as follows:

$$T_{S-MB2} = T_{HA,carry} + T_{FA,carry} + T_{HA^*,sum}$$

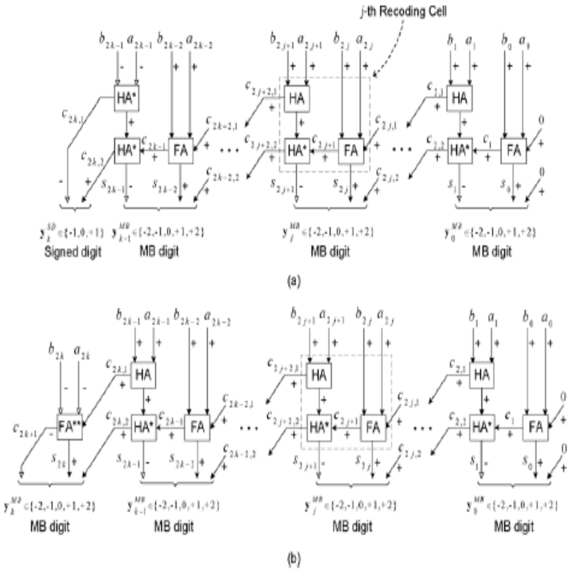


Fig. 8 *S-MB*2 recoding scheme for (a) even and (b) odd number of bits

3) *S-MB3 Recoding Scheme:* The third scheme implementing the proposed recoding technique is *S-MB*3. It is illustrated in detail in Fig. 9 for even (Fig. 9(a)) and odd (Fig. 9(b)) bit-width of input numbers.

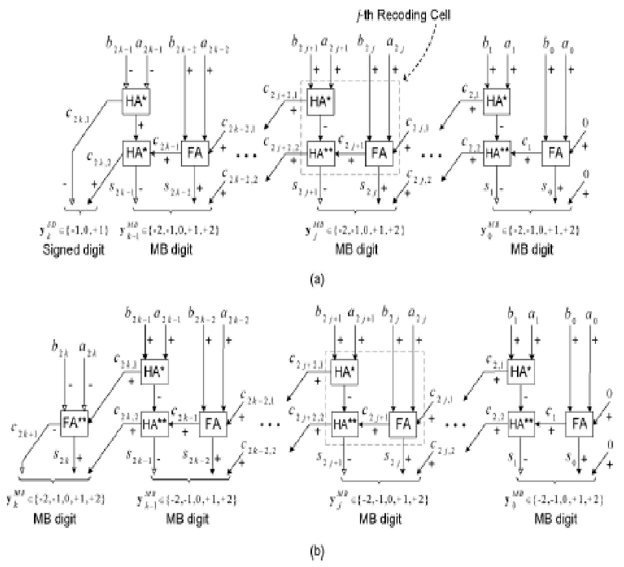


Fig. 9 *S-MB*3 recoding scheme for (a) even and (b) odd number of bits

The critical path delay of *S-MB*3 recoding scheme is calculated as follows:

$$T_{S-MB3} = T_{HA^*,carry} + T_{FA,carry} + T_{HA^{**},sum}$$

4) *Unsigned Input Numbers:* In case that the input numbers A and B are unsigned, their most significant bits are positively signed. Figs. 10–12 present the modifications that we have to make in all *S-MB* schemes for both cases of even (the two most significant digits change) and odd (only the most significant digit change) bit-width of and , regarding the signs of the most significant bits of A and B. The basic recoding block in all schemes remains unchanged.

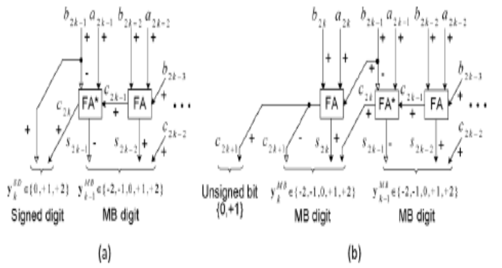


Fig. 10 Implementation of the MSD of the *S-MB*1 recoding scheme in case of unsigned input numbers for (a) even and (b) odd bit-width

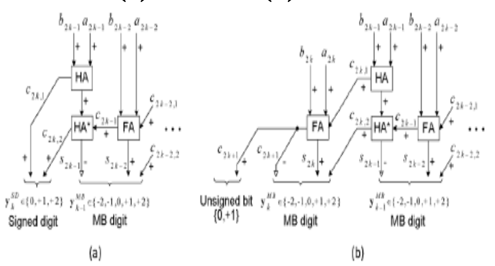


Fig. 11 Implementation of the MSD of the *S-MB*2 recoding scheme in case of unsigned input numbers for (a) even and (b) odd bit-width

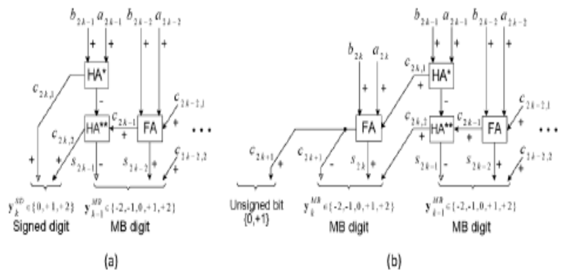


Fig. 12 Implementation of the MSD of the *S-MB*3 recoding scheme in case of unsigned input numbers for (a) even and (b) odd bit-width

The Multiplier-Accumulator (MAC) operation is the key operation not only in DSP applications but also in multimedia information processing and various other applications. As mentioned above, MAC unit consist of multiplier, adder and register/accumulator. In this paper, we used 8 bit modified booth S-MB multiplier. The MAC inputs are obtained from the memory location and given to the multiplier block. This will be useful in 8 bit digital signal processor. The input which is being fed from the memory location is 8 bit. When the input is given to the multiplier it starts computing value for the given 8 bit input and hence the output will be 8 bits. The multiplier output is given as the input to carry save adder which performs addition.

5. SIMULATION RESULTS

In Figs. 13 and 14 we present a comparison among all FAM designs, in terms of area and power consumption respectively, for even bit-width. For each case that we explored, we focus on the lowest clock period where all FAM designs are synthesized.

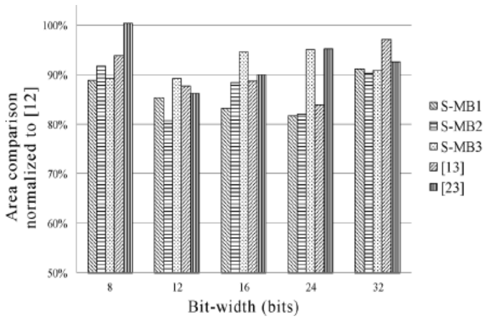


Fig. 13 Area comparison for even bit-width with all values normalized to the corresponding ones of [12]

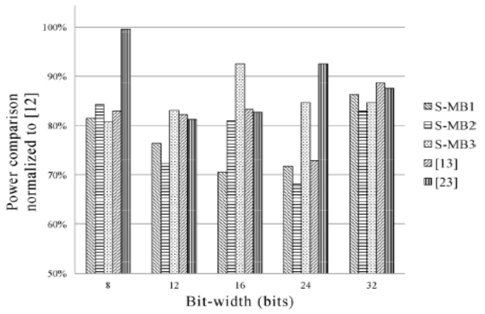
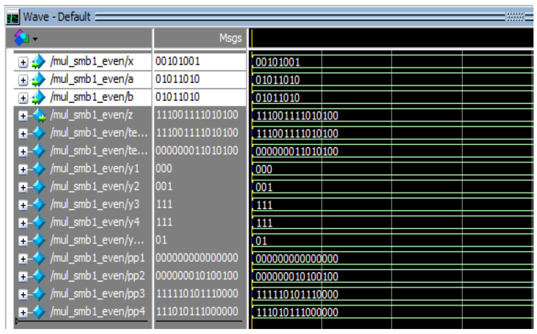
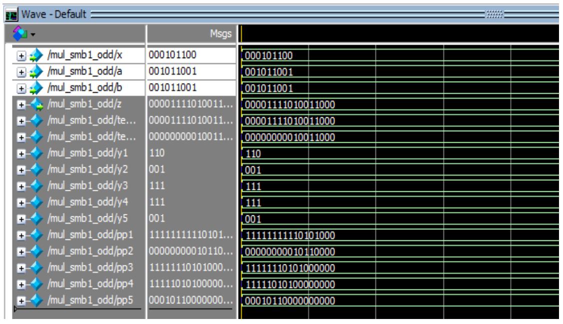


Fig. 14 Power comparison for even bit-width with all values normalized to the corresponding ones of [12]



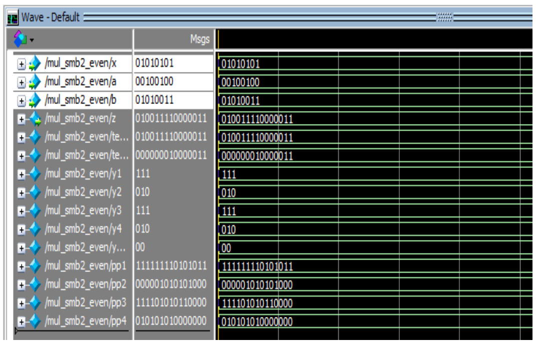
```
Wave - Default
                        Msgs
/mul_smb1_even/x        00101001            00101001
/mul_smb1_even/a        01011010            01011010
/mul_smb1_even/b        01011010            01011010
/mul_smb1_even/z        1110011110101 00    111001111010100
/mul_smb1_even/te...    111001111010100     111001111010100
/mul_smb1_even/te...    000000011010100     000000011010100
/mul_smb1_even/y1       000                 000
/mul_smb1_even/y2       001                 001
/mul_smb1_even/y3       111                 111
/mul_smb1_even/y4       111                 111
/mul_smb1_even/y...     01                  01
/mul_smb1_even/pp1      000000000000000     000000000000000
/mul_smb1_even/pp2      000000010100100     000000010100100
/mul_smb1_even/pp3      111110101110000     111110101110000
/mul_smb1_even/pp4      111010111000000     111010111000000
```

Fig. 15 S-MB1 even multiplication waveform



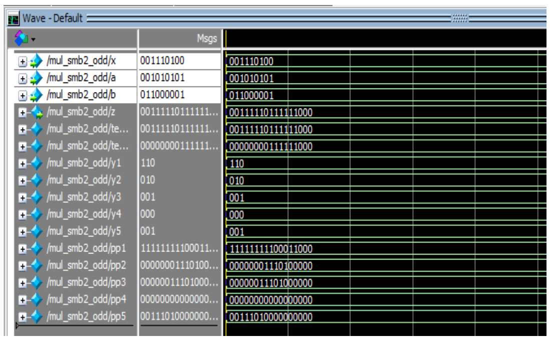
```
Wave - Default
                        Msgs
/mul_smb1_odd/x         000101100           000101100
/mul_smb1_odd/a         001011001           001011001
/mul_smb1_odd/b         001011001           001011001
/mul_smb1_odd/z         00001111010011...   00001111010011000
/mul_smb1_odd/te...     00001111010011...   00001111010011000
/mul_smb1_odd/te...     00000000010011...   00000000010011000
/mul_smb1_odd/y1        110                 110
/mul_smb1_odd/y2        001                 001
/mul_smb1_odd/y3        111                 111
/mul_smb1_odd/y4        111                 111
/mul_smb1_odd/y5        001                 001
/mul_smb1_odd/pp1       11111111110101...   11111111110101000
/mul_smb1_odd/pp2       00000000010110...   00000000010110000
/mul_smb1_odd/pp3       11111110101000...   11111110101000000
/mul_smb1_odd/pp4       11111010100000...   11111010100000000
/mul_smb1_odd/pp5       00010110000000...   00010110000000000
```

Fig. 16 S-MB1 odd multiplication waveform



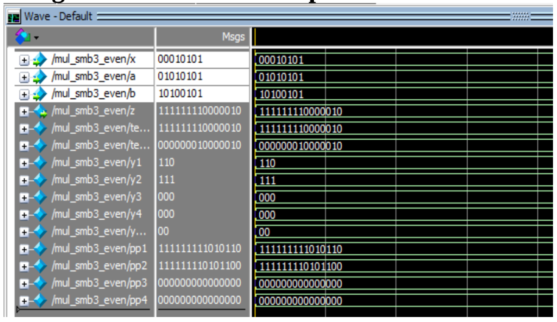
```
Wave - Default
                        Msgs
/mul_smb2_even/x        01010101            01010101
/mul_smb2_even/a        00100100            00100100
/mul_smb2_even/b        01010011            01010011
/mul_smb2_even/z        010011110000011     010011110000011
/mul_smb2_even/te...    010011110000011     010011110000011
/mul_smb2_even/te...    000000010000011     000000010000011
/mul_smb2_even/y1       111                 111
/mul_smb2_even/y2       010                 010
/mul_smb2_even/y3       111                 111
/mul_smb2_even/y4       010                 010
/mul_smb2_even/y...     00                  00
/mul_smb2_even/pp1      111111110101011     111111110101011
/mul_smb2_even/pp2      000001010101000     000001010101000
/mul_smb2_even/pp3      111101010110000     111101010110000
/mul_smb2_even/pp4      010101010000000     010101010000000
```

Fig. 17 S-MB2 even multiplication waveform



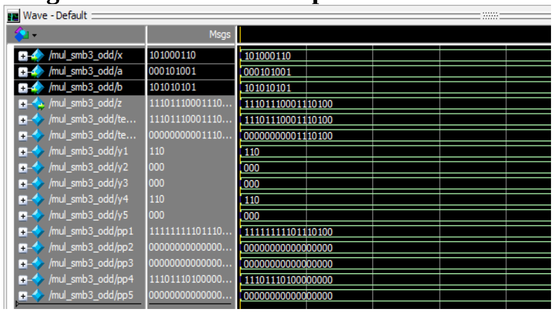
```
Wave - Default
                          Msgs
/mul_smb2_odd/x       001110100            001110100
/mul_smb2_odd/a       001010101            001010101
/mul_smb2_odd/b       011000001            011000001
/mul_smb2_odd/z       00111110111111...    00111110111111000
/mul_smb2_odd/te...   00111110111111...    00111110111111000
/mul_smb2_odd/te...   00000000111111...    00000000111111000
/mul_smb2_odd/y1      110                  110
/mul_smb2_odd/y2      010                  010
/mul_smb2_odd/y3      001                  001
/mul_smb2_odd/y4      000                  000
/mul_smb2_odd/y5      001                  001
/mul_smb2_odd/pp1     11111111100011...    11111111100011000
/mul_smb2_odd/pp2     00000001110100...    00000001110100000
/mul_smb2_odd/pp3     00000011101000...    00000011101000000
/mul_smb2_odd/pp4     00000000000000...    00000000000000000
/mul_smb2_odd/pp5     00111010000000...    00111010000000000
```

Fig. 18 S-MB2 odd multiplication waveform



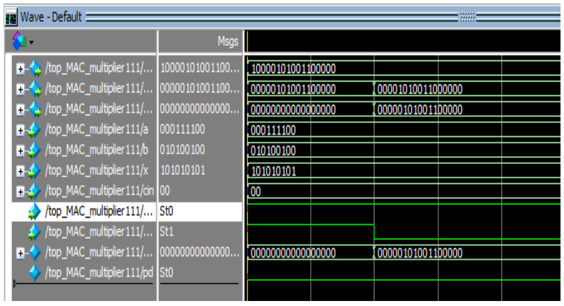
```
Wave - Default
                          Msgs
/mul_smb3_even/x      00010101             00010101
/mul_smb3_even/a      01010101             01010101
/mul_smb3_even/b      10100101             10100101
/mul_smb3_even/z      111111110000010      111111110000010
/mul_smb3_even/te...  111111110000010      111111110000010
/mul_smb3_even/te...  000000010000010      000000010000010
/mul_smb3_even/y1     110                  110
/mul_smb3_even/y2     111                  111
/mul_smb3_even/y3     000                  000
/mul_smb3_even/y4     000                  000
/mul_smb3_even/y...   00                   00
/mul_smb3_even/pp1    111111111010110      111111111010110
/mul_smb3_even/pp2    111111110101100      111111110101100
/mul_smb3_even/pp3    000000000000000      000000000000000
/mul_smb3_even/pp4    000000000000000      000000000000000
```

Fig. 19 S-MB3 even multiplication waveform



```
Wave - Default
                          Msgs
/mul_smb3_odd/x       101000110            101000110
/mul_smb3_odd/a       000101001            000101001
/mul_smb3_odd/b       101010101            101010101
/mul_smb3_odd/z       11101110001110...    11101110001110100
/mul_smb3_odd/te...   11101110001110...    11101110001110100
/mul_smb3_odd/te...   00000000001110...    00000000001110100
/mul_smb3_odd/y1      110                  110
/mul_smb3_odd/y2      000                  000
/mul_smb3_odd/y3      000                  000
/mul_smb3_odd/y4      110                  110
/mul_smb3_odd/y5      000                  000
/mul_smb3_odd/pp1     11111111101110...    11111111101110100
/mul_smb3_odd/pp2     00000000000000...    00000000000000000
/mul_smb3_odd/pp3     00000000000000...    00000000000000000
/mul_smb3_odd/pp4     11101110100000...    11101110100000000
/mul_smb3_odd/pp5     00000000000000...    00000000000000000
```

Fig. 20 S-MB3 odd multiplication waveform



```
Wave - Default
                               Msgs
/top_MAC_multiplier111/...   10000101001100...   10000101001100000
/top_MAC_multiplier111/...   00000101001100...   00000101001100000   00001010011000000
/top_MAC_multiplier111/...   00000000000000...   00000000000000000   00000101001100000
/top_MAC_multiplier111/a     000111100           000111100
/top_MAC_multiplier111/b     010100100           010100100
/top_MAC_multiplier111/x     101010101           101010101
/top_MAC_multiplier111/cin   00                  00
/top_MAC_multiplier111/...   St0
/top_MAC_multiplier111/...   St1
/top_MAC_multiplier111/...   00000000000000...   00000000000000000   00000101001100000
/top_MAC_multiplier111/pd    St0
```

Fig. 21 Mac output waveform

5. CONCLUSION

This paper focuses on optimizing the design of the Fused-Add Multiply (FAM) operator. We propose a structured technique for the direct recoding of the sum of two numbers to its MB form. We explore three alternative designs of the proposed *S-MB* recoder and compare them to the existing ones [12], [13] and [23]. The proposed recoding schemes, when they are incorporated in FAM designs, yield considerable performance improvements in comparison with the most efficient recoding schemes found in literature. Hence a design of high performance 16 bit Multiplier-and-Accumulator (MAC) is implemented in this paper.

REFERENCES

[1] 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.

[2] 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.

[3] J. J. F. Cavanagh,Digital Computer Arithmetic. NewYork:McGraw- Hill, 1984.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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. [10] C. N. Lyu and D. W. Matula, "Redundant binary Booth recoding," in Proc. 12th Symp. Comput. Arithmetic, 1995, pp. 50–57.

[11] J. D. Bruguera and T. Lang, "Implementation of the FFT butterfly with redundant arithmetic," IEEE Trans. Circuits Syst. Il, Analog Digit. Signal Process., vol. 43, no. 10, pp. 717–723, Oct. 1996.

[12] W.-C. Yeh, "Arithmetic Module Design and its Application to FFT," Ph.D. dissertation, Dept. Electron. Eng., National Chiao-Tung University,, Chiao-Tung, 2001.

[13] R. Zimmermann and D. Q. Tran, "Optimized synthesis of sum-of-products," in Proc. Asilomar Conf. Signals, Syst. Comput., Pacific Grove, Washington, DC, 2003, pp. 867–872.

[14] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs. Oxford: Oxford Univ. Press, 2000.

[15] O. L. Macsorley, “High-speed arithmetic in binary computers,” Proc. IRE, vol. 49, no. 1, pp. 67–91, Jan. 1961.

[16] N. H. E. Weste and D. M. Harris, “Datapath subsystems,” in CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed. Readington: Addison-Wesley, 2010, ch. 11.

[17] S. Xydis, I. Triantafyllou, G. Economakos, and K. Pekmestzi, “Flexible datapath synthesis through arithmetically optimized operation chaining,” in Proc. NASA/ESA Conf. Adaptive Hardware Syst., 2009, pp. 407–414.

[18][Online].Available: http://www.synopsys.com/Tools/Implementation/ RTLSynthesis/DCUltra/Pages/default.aspx

[19][Online].Available: http://www.synopsys.com/Tools/Implementation/ SignOff/PrimeTime/Pages/default.aspx

[20] Z. Huang, “High-Level Optimization Techniques for Low-Power Multiplier Design,” Ph.D., University of California, Department of Computer Science, Los Angeles, CA, 2003.

[21] C. S. Wallace, “A suggestion for a fast multiplier,” IEEE Trans. Electron. Comput., vol. EC-13, no. 1, pp. 14–17, 1964.

[22] M. Daumas and D. W. Matula, “A Booth multiplier accepting both a redundant or a non-redundant input with no additional delay,” in Proc. IEEE Int. Conf. on Application-Specific Syst., Architectures, and Processors, 2000, pp. 205–214.

[23] Z. Huang andM. D. Ercegovac, “High-performance low-power left-toright array multiplier design,” IEEE Trans. Comput., vol. 54, no. 3, pp. 272–283, Mar. 2005.

[24].Young-Ho Seo and Dong-Wook Kim, "New VLSI Architecture of Parallel Multiplier-Accumulator Based on Radix-2 Modified Booth Algorithm," IEEE Transactions on very large cale integration (vlsi) systems, vol. 18, no. 2,february 20 10

[25]. Ron S. Waters and Earl E. Swartzlander, Jr., "A Reduced Complexity Wall ace Multiplier Reduction, " IEEE Transactions On Computers, vol. 59, no. 8, Aug 20 10

[26]. C. S. Wallace, "A suggestion for a fast multiplier," iEEE Trans. ElectronComput., vol. EC-13, no. I, pp. 14-17, Feb. 1964

[27]. Shanthala S, Cyril Prasanna Raj, Dr.S.Y.Kulkarni, "Design and VLST Implementation of Pipelined Multiply Accumulate Unit," IEEE International Conference on Emerging Trends in Engineering and Technology, ICETET-09

[28]. B.Ramkumar, Harish M Kittur and P.Mahesh Kannan, "ASIC Implementation of Modified Faster Carry Save Adder ", European Journal of Scientific Research, Vol. 42, Issue 1, 2010.

[29]. R.UMA, Vidya Vijayan, M. Mohanapriya and Sharon Paul, "Area, Delay and Power Comparison of Adder Topologies", International Journal of VLSI design & Communication Systems (VLSICSj Vo1.3, No.1, February 2012

[30]. V. G. Oklobdzija, "High-Speed VLSI Arithmetic Units: Adders and Multipliers", in "Design of High-Performance Microprocessor Circuits", Book edited by A.Chandrakasan,IEEE Press,2000

[31]. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349-356, 1965

[32]. C.S. Wall ace "A Suggestion for a fast multipliers," IEEE Trans. Electronic Computers, vol. 13, no.l,pp 14-17, Feb. 1967

[33]. L.Dadda, "On Parallel Digital Multiplier", Alta Frequenza, vol. 45, pp. 574-580, 1976

[34]. WJ. Townsend, E.E. Swartzlander Jr., and J.A. Abraham, "A Comparison of Dadda and Wall ace Multiplier Delays," Proc. SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, pp. 552-560, 2003

[35]. Fabrizio Lamberti and Nikos Andrikos, " Reducing the Computation Time in (Short Bit-Width) Two's Complement Multipliers", IEEE transactions on computers, Vol. 60, NO. 2, FEBRUARY 20 1 1