POWER, AREA AND DELAY COMPARISON OF DIFFERENT MULTIPLIERS

Abstract—Multipliers play an important role in today's digital signal processing and various other applications. High speed and low power multiplier unit is the requirement of today's VLSI systems and Digital Signal Processing applications. Multiplication operation involves generation of partial products and their accumulation. The speed of multiplication can be increased by reducing the number of partial products. So, minimization of partial products is the main requirement. Here the fast multipliers like Booth multiplier, Vedic multiplier and Modified Booth recoded multiplier are designed, analyzed and compared on the basis of Power, Speed and Area. The analysis i.e., simulation and synthesis of above multipliers were done using XILINX 13.1.

Index terms—Low power consumption, LUTs, Modified Booth recoded multiplier, partial products, Vedic mathematics.

I. INTRODUCTION:

Low power consumption and smaller area are some of the most important criteria for the fabrication of DSP systems and high performance systems. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation [1-3].

The common multiplication method is “add and shift” algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms [2,3]. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing [10-12]. On the other hand “serial-parallel” multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this paper we discussed the multiplication algorithms of Booth multiplier, Vedic multiplier and Modified Booth recoded multiplier. Later, simulation results of these multipliers are shown and compared in terms of speed, area, power.

II. BOOTH MULTIPLIER

It is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly [1-3]. For the standard add-shift operation, each multiplier bit generates one multiple of the multiplicand to be added to the partial product. If the multiplier is very large, then a large number of multiplicands have to be added. In this case the delay of multiplier is determined mainly by the number of additions to be performed. If there is a way to reduce the number of the additions, the performance will get better.

Booth algorithm is a method that will reduce the number of multiplicand multiples. For a given range of numbers to be represented, a higher representation radix leads to fewer digits. Since a k-bit binary number can be interpreted as K/2-digit radix-4 number, a K/3-digit radix-8 number, and so on, it can deal with more than one bit of the multiplier in each cycle by using high radix multiplication.

ALGORITHM:

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

1. Xi Xi-1
- 0 0 Shift only
- 1 1 Shift only
- 0 1 Add Y to U and shift
- 1 0 Minus Y from U and shift

Table 1. Cases in Booth Algorithm

<table>
<thead>
<tr>
<th>Cases</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>+1</td>
</tr>
<tr>
<td>10</td>
<td>-1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

2. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2’s complement number.
So, positive numbers and negative numbers remain positive and negative respectively.
3. Circularly right shift X because this will prevent us from using two registers for the X value. Repeat the same steps until n no. of cycles are completed.
In the end we get the product of X and Y.

Example

Multiply 14 times -5 using 5-bit numbers (10-bit result).
14 in binary: 01110
-14 in binary: 10010 (so we can add when we need to subtract the multiplicand)
-5 in binary: 11011
Expected result: -70 in binary: 11101 11010

Table 2. Example of Booth multiplier operation

<table>
<thead>
<tr>
<th>Step</th>
<th>Multiplier</th>
<th>Action</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01110</td>
<td>Initialization</td>
<td>00000 11011 0</td>
</tr>
<tr>
<td>1</td>
<td>01110</td>
<td>10: Subtract Multiplier</td>
<td>00000+10010=10010 10010 11011 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right Arithmetic</td>
<td>11001 11101 1</td>
</tr>
<tr>
<td>2</td>
<td>01110</td>
<td>11: No-op</td>
<td>11001 11101 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right Arithmetic</td>
<td>11100 10110 1</td>
</tr>
<tr>
<td>3</td>
<td>01110</td>
<td>01: Add Multiplier</td>
<td>11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right Arithmetic</td>
<td>00101 01101 0</td>
</tr>
<tr>
<td>4</td>
<td>01110</td>
<td>10: Subtract Multiplier</td>
<td>00101+10010=10111 10111 01111 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right Arithmetic</td>
<td>11011 10101 1</td>
</tr>
<tr>
<td>5</td>
<td>01110</td>
<td>11: No-op</td>
<td>11011 10101 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift Right Arithmetic</td>
<td>11101 11101 1</td>
</tr>
</tbody>
</table>

III. VEDIC MULTIPLIER:

Vedic mathematics [13] was reconstructed from the ancient Indian scriptures (Vedas) by Swami Bharati Krishna Tirthaji Maharaja (1884-1960) after his eight years of research on Vedas. Vedic mathematics is mainly based on sixteen principles or word-formulae which are termed as sutras. This is a very interesting field and presents some effective algorithms which
can be applied to various branches of engineering such as computing and digital signal processing. Integrating multiplication with Vedic Mathematics techniques would result in the saving of computational time [14,15].

Thus, integrating Vedic mathematics for the multiplier design will enhance the speed of multiplication operation.

The multiplier architecture is based on UrdhvaTiryagbhyam [4] (vertical and cross-wise algorithm) sutra. An illustration of UrdhvaTiryagbhyam sutra is shown in Figure 1.

The 4x4 multiplication has been done in a single line in UrdhvaTiryagbhyam sutra, whereas in shift and add (conventional) method, four partial products have to be added to get the result. Thus, by using UrdhvaTiryagbhyam Sutra in binary multiplication, the number of steps required calculating the final product will be reduced and hence there is a reduction in computational time and increase in speed of the multiplier [6-9].

**ALGORITHM:**

Consider two 4-bit binary numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$. The partial products [5] $(P_7P_6P_5P_4P_3P_2P_1P_0)$ generated are given by the following equations:

i. $P_0= a_0b_0$

ii. $P_1= a_0b_1 + a_1b_0$

iii. $P_2 = a_0b_2 + a_1b_1 + a_2b_0 + P_1$

iv. $P_3= a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + P_2$

v. $P_4 = a_1b_3 + a_2b_2 + a_3b_1 + P_3$

vi. $P_5 = a_1b_2 + a_2b_1 + P_4$

vii. $P_6 = a_3b_3 + P_5$

viii. $P_7 = carry$ of $P_6$

**EXAMPLE:**

The figure 2 shows the example of vedic multiplication for numbers 325 and 738.

**IV. MODIFIED BOOTH MULTIPLIER:**

A solution to improve the high-speed multiplier parallelism, which helps reduce the number of stages
of the calculation result, is to boost parallelism[1-3]. The original version of the Booth multiplier (Radix - 2) has two drawbacks:

1. The number of add / subtract operations became uneven and hence became inopportune while designing Parallel multipliers.
2. The Algorithm becomes incompetent when there are isolated 1s.

Booth multiplication algorithm consists of three major steps as shown in the structure of booth algorithm figure that includes generation of partial product called as recoding, reducing the partial product in two rows, and addition that gives final product.

This modified booth multiplier is used to perform high-speed multiplications using modified booth algorithm. This modified booth multiplier’s computation time and the logarithm of the word length of operands are proportional to each other. We can reduce half the number of partial product. Radix-4 booth algorithm used here increases the speed of multiplier and reduces the area of multiplier circuit.

In this algorithm, every second column is taken and multiplied by 0 or +1 or +2 or -1 or -2 instead of multiplying with 0 or 1 after shifting and adding of every column of the booth multiplier. Thus, half of the partial product can be reduced using this booth algorithm. Based on the multiplier bits, the process of encoding the multiplicand is performed by radix-4 booth encoder.

The overlapping is used for comparing three bits at a time. This grouping is started from least significant bit (LSB), in which only two bits of the booth multiplier are used by the first block and a zero is assumed as third bit as shown in the figure3.

**Figure 3: Bit Pairing as per Booth Recoding**
Modified booth will produce at most n/2+1 partial products.

**Algorithm:**

1. Pad the LSB with one zero.
2. Pad the MSB with two zeros if n is even and 1 zero if n is odd.
3. Divide the multiplier into overlapping groups of 3-bits.
4. Determine partial product scale factor from modified booth 2 encoding table.
5. Compute the Multiplicand Multiples

6. Sum Partial Products

   - Booth recoding table is given in table 3.

**Table 3. Booth Recoding Table for Radix-4**

<table>
<thead>
<tr>
<th>M(i+1)</th>
<th>M(i)</th>
<th>M(i-1)</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+2n</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2n</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-n</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+0</td>
</tr>
</tbody>
</table>

**Example:**

1. Pad LSB with 1 zero

   ![Example Image](image1)

2. n is even then pad the MSB with two zeros

   ![Example Image](image2)

3. Form 3-bit overlapping groups for n=8 we have 5 groups

   ![Example Image](image3)

4. Determine partial product scale factor from modified booth 2 encoding table.

   ![Example Image](image4)
5. Compute the Multiplicand Multiples as shown in table 4.

Table 4. Cases in example

<table>
<thead>
<tr>
<th>Groups</th>
<th>Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6. Sum Partial Products and the result of example is shown in figure 4.

V. SIMULATION RESULTS AND COMPARISON

BOOTH MULTIPLIER:

The simulation result of Booth Multiplier is shown in the figure 5:

Figure 4: Result of Example

Figure 5: Simulation Result of Booth Multiplier
VEDIC MULTIPLIER:

The simulation result of Vedic Multiplier is shown in the figure 6:

Figure 6: Simulation Result of Vedic Multiplier

6.3 MODIFIED BOOTH MULTIPLIER:

The simulation result of Modified Booth Multiplier is shown in the figure 7:

Figure 7: Simulation Result of Modified Booth Multiplier
COMPARISON:

Here we compared Booth, Vedic and Modified Booth multipliers based on three parameters TIME/DELAY, POWER and AREA. The power is calculated for 250 Hz clock cycle using Spartan 3E Power Estimator. The table 5 shows the comparison.

<table>
<thead>
<tr>
<th>PARAMETERS</th>
<th>BOOTH</th>
<th>VEDIC</th>
<th>MODIFIED BOOTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>52.627 ns</td>
<td>31.929 ns</td>
<td>19.959 ns</td>
</tr>
<tr>
<td>POWER (for 250 Hz clock cycle)</td>
<td>135 mw</td>
<td>133 mw</td>
<td>85 mw</td>
</tr>
<tr>
<td>no. of LUTs</td>
<td>31%</td>
<td>20%</td>
<td>8%</td>
</tr>
<tr>
<td>no. of slices</td>
<td>35%</td>
<td>27%</td>
<td>9%</td>
</tr>
</tbody>
</table>

VI. CONCLUSION & FUTURE SCOPE

In this paper, multipliers for low power applications were implemented. Three basic algorithms namely Booth, Vedic and Modified Booth multipliers were implemented on Spartan 3E XILINX 13.1 version. The Power, Delay and Area are calculated for these algorithms. Power measurements were performed using Xilinx power estimator. From the table5, it is evident that Modified Booth multiplier requires less time, low power and less area to implement compared with Booth and Vedic multipliers.

This low power, fast and area efficient multiplier can be used for FIR filter design, MAC design as an extension to this paper.

VII. REFERENCES