Abstract: Fast multipliers are essential parts of digital signal processing systems. The system performance is based on the performance of multiplier used in the system, because it is the slowest component among all components used. One of the major design issue is optimizing speed and area of the multiplier which are two conflicting constraints. Hence realization of high speed multipliers is done to enhance parallelism and to decrease the number of calculation stages. For high speed arithmetic logics, a new architecture called Radix -4 Booth Multiplication algorithm which is based on MAC logic have been designed and implemented on Xilinx FPGA device. It is a combination of multiplication with accumulation and a hybrid adder (CSA and CLA) is designed to improve system performance. Here, multiplication is done as the series of repeated addition by generating partial products and finally adding them. The modified Booth Algorithm is used to reduce the number of partial products generated by a factor of 2. The multiplicand is considered as the number to be added and the multiplier is the number of times that it is added, and the result is the product.

Key Words: - VLSI, FPGA, Carry Select Adder (CSA), Carry Look Ahead Adder (CLA), ASM

I. INTRODUCTION

With the recent advancements in multimedia and communication systems, real-time signal processing like audio signal processing, video/image processing, or large-Capacity data processing are of major challenges. The multiplier and multiplier-and-accumulator (MAC) [1] are the essential elements of the digital signal processing such as filtering, convolution, and inner products. Most digital signal processing methods use nonlinear functions such as discrete cosine transform (DCT) [2] or discrete wavelet transform (DWT) [3]. Because they are basically accomplished by repeated application of multiplication and addition, the speed of the multiplication and addition arithmetic’s determines the execution speed and performance of the entire calculation.

II. BASIC BINARY MULTIPLIER

Multipliers are circuits found in every computer, cellular telephone, and digital audio/video equipment. In fact, essentially any digital device used to handle speech, stereo, image, graphics, and multimedia content contains one or more multiplier circuits. The multiplier circuits are usually integrated within microprocessor, media co-processor, and digital signal processor chips. These multipliers are used to perform a wide range of functions such as address generation, Discrete Cosine Transformations (DCT), Fast Fourier Transforms (FFT), multiply-accumulate, etc. As such, multipliers play a critical role in processing audio, graphics, video, and multimedia data.
A multiplying circuit is able to perform a multiplication of n-bits X n-bits at a high speed by increasing the speed of the forming process of the partial products so that the delay time may be inhibited from increasing for a large n, and which can inhibit the chip size becoming large. Multiplication is more complicated than addition, being implemented by shifting as well as addition. If the number of partial products generated during multiplication are more in number, the system requires more time and more circuit area to compute, allocate, and sum the partial products to obtain the multiplication result. Fig.1 shows the flow chart for basic binary multiplier.

Fast carry propagate adders are important to high performance multiplier design in two ways. First, an efficient and fast adder is needed to make any "hard" multiples that are needed in partial product generation. Second, after the partial products have been summed in a redundant form, a carry propagate adder is needed to produce the final non redundant product.

A. Ripple Adder

N bit numbers are added by designing a circuit using multiple Full adders. Each full adder inputs a C_{in} which is the C_{out} of the previous adder. This kind of adder is a ripple carry adder, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder in some cases.

The layout of a ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Each full adder requires three levels of logic.

B. Carry Look-Ahead Adder (CLA)

The concept behind the CLA is to avoid the rippling carry present in a conventional adder design. The rippling of carry produces unnecessary delay in the circuit. Carry look-ahead logic uses the concepts of generating and propagating carries. Although in the context of a carry look ahead adder, it is most natural to think of generating and propagating in the context of binary addition, the concepts can be used more generally than this. In the descriptions below, the word digit can be replaced by bit when referring to binary addition.

C. Carry Select Adder (CSA)

The carry select adder generally consists of two ripple carry adders and a multiplexer. Adding two k-bit numbers with a carry select adder is done with two k/2 adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The number of bits in each carry select block can be uniform, or variable.
In the uniform case, the optimal delay occurs for a block size of square root of K. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added.

**D. Hybrid Adder**

Hybrid Adder [11, 12] is a combination of any two adders. It is used in high speed applications. The proposed hybrid adder consists of two carry look ahead adders and amultiplexer. Adding two n-bit numbers with a hybrid adder is done with two adders (therefore two carry look ahead adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The propagation delay is less for hybrid adder and at the same time it occupies larger area compared to the other adders.

**IV. DESIGN APPROACH**

This section is about the design approach for Radix-2 and Radix-4 Booth multipliers by considering the necessary specifications and made in the form of state diagrams and ASM charts for develop the relevant source code in VHDL. The presented Figures elaborate the logics required for necessary operations.

**A. Booth Multiplication Algorithm for Radix-2**

It will encode the multiplicand based on multiplier bits. In Radix-2 we will compare 2 bits at a time with overlapping technique. Grouping starts from the LSB, and the first block only uses one bit of the multiplier and assumes a zero for the second bit.

<table>
<thead>
<tr>
<th>Block</th>
<th>Partial Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1*Multiplicand</td>
</tr>
<tr>
<td>10</td>
<td>-1*Multiplicand</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

The functional operation of booth encoder is as mentioned in Table 1. There are two inputs for booth encoder one is multiplicand and the other is 2 bits from multiplier, based on these two inputs it will encode the multiplicand.

**State diagram**

The state diagram of the Radix-2 Booth multiplier is shown in Fig.2. Here we have four different types of states. For 00, 11 states we can perform multiplication of multiplicand with zero. For 01 state, multiplicand is multiplied with one whereas for 10 state, multiplicand can be multiplied with -1.

**ASM chart**

The Fig.3 shows the ASM chart for Radix-2 booth multiplier. It represents conventional procedure for various operations required with respect to state of machine. Here we generate the partial products by Radix-2 booth encoder. By using this technique we can reduce the partial products generation and the computation time delay is less than ordinary multiplication.

**B. Booth Multiplication Algorithm for Radix-4**

To avoid the problems in Radix-2 algorithm, realization of high speed multipliers is needed. One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks. They are: (i) the number of add subtract operations and the number of shift operations are variable and become inconvenient in designing parallel multipliers. (ii) The algorithm becomes inefficient when there are isolated 1’s. These are overcome by using modified Radix-4 Booth multiplication algorithm. The design approach of Radix-4 algorithm is described with the pictorial views of state diagram and ASM chart.
Fig. 3. ASM chart for Radix-2 Booth Multiplier

Fig. 4. ASM chart for Radix-4 Booth Multiplier
Table 2: Radix-4 Booth Encoding Table

<table>
<thead>
<tr>
<th>Block</th>
<th>Partial Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1*multiplicand</td>
</tr>
<tr>
<td>010</td>
<td>1*multiplicand</td>
</tr>
<tr>
<td>011</td>
<td>2*multiplicand</td>
</tr>
<tr>
<td>011</td>
<td>2*multiplicand</td>
</tr>
<tr>
<td>100</td>
<td>-2*multiplicand</td>
</tr>
<tr>
<td>101</td>
<td>-1*multiplicand</td>
</tr>
<tr>
<td>110</td>
<td>-1*multiplicand</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
</tr>
</tbody>
</table>

This algorithm scans strings of three bits at a time as follows:
1) Extend the sign bit 1 position if necessary to ensure that n is even.
2) Append a 0 to the right of the LSB of the multiplier.
3) According to the value of each vector, each Partial Product will be 0, +y, -y, +2y or -2y.

The multiplicand encoding process using Radix-4 Booth encoder is based on the multiplier bits. It will compare 3 bits at a time with overlapping technique. Grouping starts from the LSB, and the first block uses only two bits of the multiplier and assumes a zero for the third bit. The functional operation of Radix-4 booth encoder is shown in Table 2.

The state diagram of the Radix-4 Booth multiplier is shown in Fig.4. It consists of eight different types of states as we are comparing 3 bits at a time and during these states we can obtain the outcomes, which are multiplication of multiplicand with 0, -1 and -2 consecutively. The state diagram presents various logics to perform the Radix-4 Booth multiplication in different states as per the adopted encoding technique.

**ASM chart**

The ASM chart for Radix-4 booth multiplier is as shown in Fig.5. This represents the conventional flow of operations that are required for Radix-4 booth multiplier in various states. Here we can generate the partial products by Radix-4 booth encoder. By using this technique we can further reduce the partial products generation and the computation time delay, which is less than that of Radix-2 multiplication.

**V. SIMULATION RESULTS**

![Fig.5. State diagram of Radix-4 Booth Multiplier](image)

![Fig.6. Simulation Results of Radix-2 Booth multiplier](image)
VI. FPGA REALIZATION

The designed system is targeted onto Xilinx xc2vp70-7-ff1704 FPGA device belonging to virtex2p family with a speed grade of –7. The logical routing can be observed from the obtained Place and route result from the FPGA Editor option in Xilinx synthesizer. It is observed that about 40% area for the targeted FPGA is covered for the implementation of this System. The CLB’s are connected in cascade manner to obtain the functionality for the designed system.

A. Synthesis Report

The synthesis result for the proposed algorithm is presented:

Macro Statistics

- # Registers: 49
- # Multiplexers: 25
- # Tristates: 74
- # Adders/Subtractors: 618
- # Multipliers: 29
- # Comparators: 128

Design Statistics

- # IOs: 26
- Cell Usage:
  - # BELS: 181

Minimum period: 5.220ns (Maximum Frequency: 191.571MHz)

From the result it is observed that logical counts of 181 Basic Element Logic (BEL) is required for the realization of DST processor. The real time maximum operating frequency obtained is 191.571 MHz and this operation frequency is considerably
higher than the current sample frequency and makes it more suitable for real time current analysis.

B. RTL Views

![Fig.8. RTL Schematic of Radix-2 Booth multiplier](image)

![Fig.9. RTL Schematic of Radix-4 Booth multiplier](image)

C. Routing and Placement

![Fig.10. Routing of logical placement in targeted FPGA](image)

D. Implementation Observations

The implementation of proposed Radix-4 Booth Multiplication algorithm is illustrated in various pictorial views obtained during the process of realization i.e., from Fig.8 to Fig.12. Fig.8 & Fig.9 shows the RTL views of existing and proposed algorithms. Routing of logical placement in targeted FPGA is shown in Fig.10 and Fig.11 represent the placement of the targeted logic onto FPGA device.

VII. PERFORMANCE OF MULTIPLIERS

<table>
<thead>
<tr>
<th></th>
<th>Radix-4 booth multiplier with hybrid adder</th>
<th>Radix-2 booth multiplier with hybrid adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of slices</td>
<td>119</td>
<td>166</td>
</tr>
<tr>
<td>No. of LUTs</td>
<td>213</td>
<td>300</td>
</tr>
<tr>
<td>Path delay</td>
<td>29.198ns</td>
<td>37.881ns</td>
</tr>
</tbody>
</table>

The above Table 3 is valid for 8 bit x 8 bit multiplier. The table distinguishes the performance of proposed Radix-4 Booth Multiplier with the existing Radix-2 Booth Multiplier. The main advantage of using Radix-4 is it has less propagation delay, i.e. speed and at the same time it occupies lesser area than Radix-2.
VIII. FUTURE SCOPE

The algorithm has been implemented using hybrid adder to add the partial products in parallel for the final output. Hybrid adder is a combination of carry look ahead adder and carry select adder. It can be further extended by taking combination of any two adding techniques so that propagation delay is further reduced. For higher inputs Radix 2^n multipliers will give better performance.

IX. CONCLUSION

It is to be concluded that this presentation deals with the design approach of modified (Radix-4) Booth’s algorithm. Further, we have observed the simulation results of the booth multiplier and booth encoder for radix-2 and radix-4 algorithms. The proposed Booth multiplier is realized on Xilinx FPGA device using relevant synthesizer. The design flow was discussed with the aid of necessary ASM charts and state diagrams.

REFERENCES


