Radix-2 Vs Radix-4 High Speed Multiplier

A B. Pawar
M.Tech Student, Department of Electronics and Telecommunication,
Government College of Engineering, Amravati,
Maharashtra, India

Abstract— This paper presents the implementation and comparison of high speed multipliers based on booth encoding. The paper particularly compares the performance of Radix-2 and Radix-4 based booth multipliers. The multipliers are designed for 8X8 bit multiplication operation. Each of the multiplier design incorporates 3:2 as well as 4:2 compressors. The design entry is done in Xilinx ISE 13.1 design suite and the simulation of the design is carried out using Modelsim SE 6.30 from Mentor Graphics. The design is targeted toward the Virtex 6 FPGA.

Keywords— FPGA, VHDL, Booth encoding, Compressors, Wallace Tree, CLA.

I. INTRODUCTION

The multiplier is an essential element of digital signal processing operations such as filtering and convolution. Most of the digital signal processing operations such as Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT) are accomplished by repetitive multiplication and addition. Hence the speed of these operations exclusively depends upon the speed of the multiplication operation being performed. It has been observed that the multiplier requires the longest delay among the basic operational blocks in a system, hence the critical path is predominantly determined by the multiplier [5]. Also, the multiplier has been observed to consume comparatively more area and power. Therefore a design which will reduce the consumed area or power or speed or any combination of the above three parameters is of research interest. Multiplication operation involves two major steps.

1. Generation of partial products
2. Accumulation of partial products

The speed of multiplication can be improved by two ways.

1. Reducing the number of partial products.
2. Accelerating the summation of partial products.

Various conventional methods for multiplication such as Shift and Add method, Array multiplier have high computational cost and delay. The Booth algorithm proposed by Andrew Donald Booth in 1951, is capable of multiplying two signed 2’s complement numbers [4]. The original algorithm devised by Booth performed radix-2 recoding of the multiplier in order to generate the partial products. Hence the original Booth algorithm gave N partial products for an N-bit multiplier. The speed of the multiplier can be increased appreciably if a higher radix Booth algorithm is used for the generation of the partial products. For high speed multipliers the summation of partial products is done using Wallace Tree architecture. Higher accumulation speeds can be obtained if the adders in the Wallace tree are replaced by compressors. A CLA can be used in the final addition stage to obtain the product.

II. ARCHITECTURE

Figure 1 shows the architecture of the designed multiplier. The below architecture makes use of the Booth algorithm to encode the multiplier bits. The Booth encoder can be designed to encode the multiplier bits as per the Radix-2 or Radix-4 encoding rules. The design incorporates the encoder and the partial product generator into a single design block. The generated partial products are then added in a parallel fashion using Wallace tree arrangement of different order compressors. For large multipliers of up to 32-bits, the performance of modified Booth algorithm deteriorates [5]. So Booth recoding along with Wallace tree architectures for partial product reduction have been used in the proposed design. So Booth recoding along with Wallace tree architectures for partial product reduction have been used in the proposed design. So Booth recoding along with Wallace tree architectures for partial product reduction have been used in the proposed design.
Andrew D. Booth, in 1950 devised an algorithm to multiply two numbers of either sign using a uniform process without any foreknowledge of the signs of the two numbers [1]. This basic Booth algorithm has been used in the design to generate the partial products by encoding the multiplier bits. Table I shown below gives the Radix-2 recoding of the multiplier. Here X and Y are multiplicand and multiplier respectively.

<table>
<thead>
<tr>
<th>Multiplier Bits</th>
<th>Recoded Multipland, Y_1 Y_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0*X</td>
</tr>
<tr>
<td>0 1</td>
<td>+1*X</td>
</tr>
<tr>
<td>1 0</td>
<td>-1*X</td>
</tr>
<tr>
<td>1 1</td>
<td>0*X</td>
</tr>
</tbody>
</table>

For a N-bit multiplier the Radix-2 Booth algorithm generates N number of partial products, which can then be added in a parallel fashion using either simple adders or more preferably compressors. The major disadvantages of the original radix-2 algorithm constitute the requirement of N shifts and an average of N/2 additions for a N-bit multiplier. Also the algorithm’s performance deteriorates in the case of isolated 1’s in the multiplier. These drawbacks of the Radix-2 algorithm are overcome by the Radix-4 Booth algorithm [5].

**B. Radix-4 Booth Algorithm**

Speeding up the multiplication using Booth algorithm can be achieved by recoding the multiplier in a higher radix than 2. Higher radix recoding means greater number of multiplier bits are inspected and eliminated per cycle resulting in less number of cycles required to obtain the product [4]. In case of Radix-r recoding, the number of bits inspected (n) per cycle is given by:

\[ n = 1 + \log_2 r \]  

and the number of bits eliminated (e) per cycle is:

\[ e = n - 1 = \log_2 r \]

In Radix-4 recoding, 3 multiplier bits are inspected simultaneously and 2 bits are eliminated per cycle. Hence, for a N-bit multiplier the number of partial products are reduced to N/2. The following table illustrates the Radix-4 recoding rules.
As the number of partial products is halved as compared to Radix-2 scheme, a definite speed improvement is exhibited by the Radix-4 scheme.

### C. Wallace Tree

The Wallace tree is an efficient hardware implementation of a digital circuit which adds the partial products [2]. The role of the Wallace tree is that of accelerating the summation of partial products. The Wallace tree is typically used in high speed designs to reduce the partial products to two rows, which can then be added in the last stage [2]. The conventional Wallace tree architecture contains an adder tree, while in the proposed design the Wallace tree is constructed using 3:2 and 4:2 compressors.

### D. Compressors

Compressors basically are logic circuits, capable of performing addition of 3 or more bits simultaneously. Compressors play an important role in multiplier architectures. They are extensively used in the partial product accumulation stage. The compressors reduce the number of conventional adders required for accumulation. The design particularly incorporates 3:2 and 4:2 compressors. A compressor based Wallace tree architecture helps in achieving reduced power consumption and critical path delay.

#### 3:2 Compressors

A 3:2 compressor can be thought of as a 1-bit full adder circuit. It takes 3 input bits and gives out two output bits viz. sum and carry. The 3:2 compressor can simultaneously add three bits and hence it is used in partial product accumulation where there are 3 bits in a single columns. A Wallace tree constructed with these compressors will reduce the entire partial product array into two bit streams of equal length which can then be added using any of the fast adder architectures.

#### 4:2 Compressors

4:2 compressors are similar to 3:2 compressors except that the number inputs are 4 and the numbers of outputs are 3. These take in 4 bits as input and produce 3 outputs viz. sum, carry and Cout, where Cout is the carry bit propagated to the next higher order compressor. These are used in partial product accumulation where there are four bits in a single column.
Although various different types of compressor architectures have been proposed, the simplest method of designing a 4:2 compressor is from two 3:2 compressors. Considerable speed improvement can be observed in designs implemented using compressors as compared to those implemented with just adders.

III. SIMULATION
The design entry of the multipliers is done using VHDL and simulation was carried out using ModelSim SE 6.30 design suite from Mentor Graphics. The design was then synthesized on a Xilinx XC6VCX75T FF484-2 FPGA using Xilinx ISE 13.1 design suite. Figures 4 and 5 exhibit the RTL schematic of the Radix-2 and Radix-4 multipliers respectively. Table III gives summary of the logic utilization of the two multiplier designs.

IV. CONCLUSION
The simulation results show that, the device utilization of Radix-2 multiplier is much greater as compared to Radix-4 multiplier. The Radix-2 multiplier consumes 43% more device resources as compared to Radix-4 multiplier. The Radix-4 multiplier is faster as compared to Radix-2 multiplier. A speed improvement of 19% is observed in the case of Radix-4 multiplier as compared to Radix-2 multiplier. The reason for the improvement in speed was the reduction in the number of partial products in case of Radix-4 recoding. Further improvement of speed can be achieved by use of CLA in the last addition stage.

Figure 4: RTL schematic of Radix-2 Booth multiplier

Figure 5: RTL schematic of Radix-4 Booth multiplier

Figure 6: Simulation Waveform of Multiplier
Table III: Device Utilization Summary

<table>
<thead>
<tr>
<th>Logic Utilization (XC6VCX75T)</th>
<th>Radix-2 Booth Multiplier</th>
<th>Radix-4 Booth Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slice LUT’s used</td>
<td>121</td>
<td>69</td>
</tr>
<tr>
<td>Number of occupied slices</td>
<td>41</td>
<td>21</td>
</tr>
<tr>
<td>Number of bonded IOB’s</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Max. i/p pad-o/p pad delay (ns)</td>
<td>14.097</td>
<td>14.060</td>
</tr>
<tr>
<td>Max. combinational path delay (ns)</td>
<td>8.322</td>
<td>6.746</td>
</tr>
<tr>
<td>Peak memory usage (MB)</td>
<td>396</td>
<td>394</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

I express my sincere gratitude to Mr. A. M. Shah, Assistant Professor, Department of Electronics and Telecommunication, Government College of Engineering, Amravati, for extending his valuable insight for completion this work.

REFERENCES


