FPGA Implementation of Pipelined Booth Encoded Wallace Tree Multiplier

Twinkle Pajiyar, Swedshree Patil, Yogini shirude
#1 Dept. Of Electronics And Telecommunication.
PCCOE, Savitribai Phule Pune University,
Pune, India, 9028048460
#2 Dept. Of Electronics And Telecommunication.
PCCOE, Savitribai Phule Pune University,
Pune, India, 8983504656
#3 Dept. Of Electronics And Telecommunication.
PCCOE, Savitribai Phule Pune University,
Pune, India, 9420699167

ABSTRACT

Multiplier is one of most important parts in signal processing, computer organization and other computationally intensive application. Here we have designed a multiplier by Booth Encoding the values and using Wallace tree multiplier method that are high speed, low power and reduce latency. Pipelining stages are used in Wallace tree Booth multiplier. They have been implemented in FPGA Spartan II kit using Verilog coding. Simulation has been done on Modelsim Simulator.

Key words: Booth Encoding, Wallace tree Structure, Pipelining stages, Spartan II kit, Verilog, FPGA, Modelsim.

Corresponding Author: Twinkle Pajiyar

INTRODUCTION

A standard approach that might be taken by a novice to perform multiplication is to "shift and add", or normal multiplication. That is, for each column in the multiplier,shift the multiplicand by the appropriate number of columns and multiply it by the value of the digit in that column to obtain a partial product. The partial products are then added to obtain the final result. It is shown in the example below.

0 0 1 0 1 1
0 1 0 0 1 1
0 0 1 0 1 1
0 0 1 0 1 1
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 1 1
0 0 1 1 0 1 0 0 0 1

0 0 1 1 0 1 0 0 0 1

0 0 1 1 0 1 0 0 0 1
With this system, the number of partial products is exactly the number of columns in the multiplier. So, the method to reduce the number of partial products and also increase the speed of computation used is Booth multiplier with Wallace tree structure. The one of the two operands are booth encoded by applying booth encoding scheme. The partial products formed out of those are then added using Wallace tree algorithm with ripple carry adder. Pipelining has been introduced at intermediate stages so that one or more tasks are performed simultaneously and delay is minimized.

II. BLOCK DIAGRAM.

A. The complete operation has been done stage wise. There are four basic stages. The two operands A and X involved in multiplication are termed as multiplicand and multiplier respectively.

B. One’s Complement.
The multiplicand A is given to the one’s complement block where its one’s complement is taken. Both the multiplicand and its complement are required for further computations. Thus these values are stored separately.

C. Booth Encoder
The multiplier X on the other hand is given to the Booth Encoder Block where it has been booth encoded by using the technique of radix 4 Booth recoding.
To Booth recode the multiplier term, we consider three bits at a time, such that their is one bit overlapping the previous block of bits. We start grouping from the LSB, and the first block only uses two bits of the multiplier (since there is no previous block to overlap).

Fig. 2: Grouping of bits from the multiplier term, for use in Booth recoding.

The least significant block uses only two bits of the multiplier, and a zero is assumed for the third bit. The overlapping is required to know what happened in the last block, as the MSB of the block acts like a sign bit. We then consult the table below to decide what the encoding will be.

<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>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>

Table 1 Booth Recoding Table

The basic idea is that, instead of shifting and adding for every column of the multiplier term we only take every second column and multiply by ±1, ±2, or 0 to obtain the same results. The advantage of this method is the halving of the number of partial products. This is important in circuit design as it relates to the propagation delay in the running of the circuit, and the complexity and power consumption of its implementation.

It is also important to note that there is comparatively little complexity penalty in multiplying by 0, 1 or 2.

**D. Partial Product Generation Logic**

All that is needed is a multiplexer or equivalent, which has a delay time that is independent of the size of the inputs. As shown in the figure below a two stage multiplexer logic has been used for generating the partial products. In the first stage the ‘neg’ signal acts as the select line, it decides whether the multiplicand (A) or its complement (Ac) is to be passed to the next stage. The neg signal depends on the operation to be performed in booth encoding table.

In the next multiplexer stage ‘two, one, zero’ serve as the select lines. If two is high then it will pass the last state of the input, if one is high it will pass the output from previous multiplexer to the final output stage else if zero is set high then it sets the output to '0'. Negating 2's complement numbers has the added complication of needing to add a "1" to the LSB, but this can be overcome by adding a single correction term with the necessary "1"s in the correct positions. This is known as the correction bit. For every combination of 3 bits a partial product
for eg. PP_0(0) is formed and when the entire multiplier is scanned then the complete PP_0 is obtained and similarly PP_1,PP_2 and PP_3.

![Multiplexer Logic for generating partial products](image)

**E. Wallace Multiplier Algorithm**

Once the partial products are generated they are added using full and half adders. The method used for addition is known as Wallace Multiplier Algorithm. It makes use of a full adder for adding three bits and a half adder for adding two bits. The carry generated in each stage is propagated to the next stage. This is called a carry ripple adder. Wallace when combined with Booth is said to reduce the complexity of circuit and also save time. The representation of the Wallace structure has been shown in the figure below.

**III. LINEAR PIPELINING**

Pipelining has been introduced in intermediate steps so that more tasks are performed at a time. Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline has been divided in stages. At each stage a part of an instruction is executed in parallel. The stages are interconnected to form a pipe - instructions progress through the stages, and give the final output. Pipelining increases instruction throughput.

**IV. SIMULATION AND RESULT**

The simulation of the program is done using ModelSim tool and Xilinx ISE Design Suite. The results for the multiplication of 8x8 multiplication using Modified Booth Multiplier is shown in this section.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices:</td>
<td>128 out of 2352</td>
</tr>
<tr>
<td>Number of 4 input LUTs:</td>
<td>202 out of 4704</td>
</tr>
<tr>
<td>Number of bonded IOBs:</td>
<td>64 out of 180</td>
</tr>
<tr>
<td>Maximum combinational path delay:</td>
<td>34.622ns</td>
</tr>
</tbody>
</table>

Table 2. Synthesis Result
CONCLUSION
The booth encoding scheme has been used here which reduces the no. of partial products to 4 for 8*8 multiplication. With the reduced partial products the computation time is reduced and speed of the circuit has increased. We have used Wallace tree structure for adding all the partial products. The Carry generated in all the stages is forwarded to the next stage. This type of adder is called a Carry ripple adder. Linear pipelining also has been introduced in intermediate nodes so as to reduce the propagation delay of the circuit.

REFERENCE


