A Design and Implementation of Radix-4 Based Fast Multiplier on FPGA

Gadi. Sanjeev¹, Dr.S.Tamil², Dr. Bharti Chourasia³

¹Research Scholar, Dept. of ECE, SRK University, Bhopal, India.
²,³Professor, Dept. of ECE, SRK University, Bhopal, India.

ABSTRACT:

Many advanced signal processing and image processing systems are being constructed on VLSI chips in which multiplication is the main operation, thanks to the recent rapid increase in the scale of integration. The compute capacity and power consumption of these systems determine their performance. In this paper, a simplified method for generating n/2 partial product rows with regular partial product arrays and negligible overhead is proposed, reducing partial product complexity. This paper uses a 32-bit fast multiplier to generate partial products using the Modified Booth-radix 4 algorithm, a CSA-based Wallace-Tree-like tree compress partial products, and a Carry look-ahead adder is used to perform final compression. This strategy was introduced in Verilog and simulated with Xilinx ISE 12.1 software before being implemented on the FPGA xc3s50-5pq208. When compared to a multiplier that employs a conventional carry select adder, simulation results show that the delay of both multipliers is reduced, as is the number of logic levels, with a slight increase in the number of slices and LUTS.

Keywords: Multiplication. Modified Booth Multiplier, Wallace tree Multiplier, CLA, CSA

I. INTRODUCTION

Many advanced signal processing and video processing systems are now being constructed on VLSI chips in which multiplication is the main operation, thanks to the recent rapid increase in the scale of integration. Multiplication is an essential arithmetic operation used in a plethora of processors and digital signal processing systems. Multiplication-based operations such as multiply and accumulate (MAC) and inner product are widely utilized across many digital signal processing (DSP) operations such as convolution, fast Fourier transform (FFT), filtering, and in microprocessors' arithmetic and logic unit [1]. Over the last few years, portable multimedia and digital signal processing (DSP) systems, which often demand low power consumption, a prerequisites cycle, and flexible processing ability, have grown in popularity. Because many multimedia and DSP applications require a lot of multiplication, multipliers dominate these systems' performance and power consumption [2, 3]. Regrettably, most portable devices run on stand-alone batteries, yet multipliers need a lot of energy. In most cases, the performance of signal processing applications, multimedia, and 3D graphics is heavily reliant on the efficacy of the hardware used for computing multiplications because multiplication, in addition to addition, is widely used in these contexts. As a result, developing power-efficient multipliers is critical for assembling a high-performance, low-power portable multi-media and DSP system. Two key reasons drive the demand for low-power VLSI systems. First, when operating frequencies and processing capacity per chip increase, large currents must be delivered, and heat generated by high power consumption must be evacuated using proper cooling techniques. Second, battery life is restricted in portable electronic gadgets. In these portable devices, the low power design immediately correlates to a longer operating duration. This has sparked the development of new circuit techniques intending to lower the power consumption of multiplication computations by including high-speed architectures with enough performance. A computing system's multiplier is a rather substantial component. The square of the multiplier's resolution, i.e. the multiplier's size, is directly proportional to the multiplier's size. The digital Finite Impulse Response (FIR) is widely utilized in a wide range of Signal Processing tasks, including wireless communication, video, and image processing. The essential components of a digital filter are the adder, multiplier, and delay elements. In an FIR filter, the multiplier is a critical and challenging building block.

The study outlined several approaches for using a Field Programmable Gate Array (FPGA) in a digital FIR filter [4, 5]. Any multiplier can be broken down into three stages: partial product creation, partial product addition, and ultimate product addition. By lowering the number of partial products, the speed of multiplication can be boosted. To speed up multiplication, many high-performance algorithms and designs have been suggested. Several multiplication algorithms have been proposed, including Booth, Modified Booth, Braun, and Baugh-Wooley [6-9]. The improved Booth method, also known as the fastest multiplication algorithm, lowers the number of partial products that must be formed. Save the Wallace Tree to sum the partial products in less time; adder structures were used.

www.turkjphysiotherrehabil.org
The multiplier is chosen in this paper depending on the type of technique where it will be used. The design is established for the multiplication of 32 bits by 32 bits. Our goal is to use Booth's approach for multiplication to reduce computation time and to use Carry Save Adders arranged in a Wallace tree structure to reduce chip area. In the last stage, any high-speed adder is used to combine the tree's two-row outputs. We employ the Booth-radix 4 technique to build partial products to accomplish them. A CSA-based Wallace-Tree-like tree is used to compress partial product, followed by a Carry look-ahead adder (CLA).

II. BACKGROUNDWORKS

The authors suggested an energy-efficient approximate multiplier using radix-4 Booth encoding and logarithmic product approximation [10]. Furthermore, a data path pruning approach is presented and investigated to reduce the multiplier's hardware complexity. The multiplier's error performance and effectiveness in terms of energy and area consumption were evaluated through several tests. The provided study relied on TSMC-180nm simulations. The proposed multiplier's usefulness in image smoothing and convolutional neural networks is further investigated. According to the application evaluation, the proposed multiplier can substitute an exact multiplier, resulting in a 75% reduction in energy consumption and a 50% reduction in the area use. A comparison of the proposed approach to state-of-the-art multipliers reveals that it has the potential to be a groundbreaking approach for approximation multipliers. Compared to existing approximate non-logarithmic multipliers, the suggested multiplier uses less energy while maintaining the same level of applicability in image processing and classification. On the other hand, several state-of-the-art approximation logarithmic multipliers consume less power than the recommended multiplier but degrade performance significantly in the application instances studied.

Functional Units that are designed to receive inputs and generate outputs in a non-redundant style often perform poorly. Carry-save and partial carry-save formats have been developed to overcome this constraint. When it comes to implementing addition trees, both ways are excellent. To avoid invoking the distributive property, the inputs to the multiplier must be reduced to a non-redundant form if there are multiplications in the datapath. The authors of [11] presented a multiplier that can take two values in partial carry-save format and return a partial carry-save result. This is accomplished by tweaking the Booth encoder and taking advantage of the partial carry-save format's produce and propagate group signals. As a result, it may be possible to fully implement datapaths without incurring additional penalty cycles due to non-redundant form reductions. Experiments have demonstrated that the suggested multiplier has a 15% shorter delay than a traditional Booth radix-4 multiplier. Furthermore, when combined with partial carry-save adders, it is possible to lower execution time by 36% on average for many benchmarks while reducing the energy-delay product (EDP) by 32.7%.

The traditional FIR filter architecture wastes more power due to typical multiplier topologies. Partial product production consumes more electricity in the multiplier section. The authors of [12] proposed using the Improved Booth Recoding Algorithm to create a 16-Tap FIR Filter design employing a Radix-4 Booth Multiplier. The proposed multiplier architecture reduces the number of steps in multiplication while simultaneously reducing the propagation delay in digital circuits. Finally, these architectures improve the FIR filter's performance in terms of area, latency, and power consumption. Based on the performance evaluation, FPGA synthesis tools generate implementation results. The results clearly show that employing Radix-4 Booth Multiplier with Improved Booth Recoding Algorithm of FIR filter achieves greater performance in power, resource utilization, and area than FIR filter with a traditional multiplier. The Radix-4 Booth Multiplier with Improved Booth Recoding Algorithm condenses power and capacity by 52.27% and 22.20%, respectively, compared to a standard FIR filter. According to the results, the Improved Booth multiplier-based FIR (radix-4) filter produces the most negligible power and area. FIR filters can also be used for communication purposes.

The authors of [13] developed a multiplier for high-speed, low-energy operations. This paper developed a high-speed multiplier by combining the Modified Radix-4 booth technique with the Redundant Binary Adder. This work also compares different booth algorithms regarding power consumption, delay, area, energy, and energy-delay product. The Cadence simulation tool is used to simulate all of the circuits utilizing 180nm technology. According to the test findings, the proposed booth multiplier has a high speed, low energy, and low energy-delay product compared to existing booth multipliers. Radix-2 and Radix-4 algorithms were also implemented and compared to a modified Radix-4 algorithm. The Modified Radix-4 algorithm provides the best system throughput, energy, power, and energy-delay product according to the performance evaluation results. For the addition, the proposed architecture included a redundant binary adder, which enhanced the speed. As per the analysis, the proposed circuit is susceptible to high performance and saves energy compared to classical architectures.

Many Digital Signal Processing (DSP) applications, such as Multiple-Accumulate Units and Fast Fourier
Transforms, employ signed multiplications, which are exceedingly expensive (FFT). The speed with which a multiplication operation can be performed typically determines the performance of DSP processing blocks. As a result, a high-speed signed multiplier is essential for high-speed DSPs. The authors recommended in [14] that a new high-speed signed multiplier based on Vedic mathematics be designed and implemented. Compared to a standard Booth radix-2 multiplier, the proposed architecture has a shorter delay and takes up less area. It employs a 2s complement circuit and an unsigned multiplier based on Urdhva Tiryakbyham. The recommended signed multiplier and traditional booth multiplier are written in Verilog, synthesized, and tested using the ISE simulator. The iwave-systems Unified Learning Kit Spartan6 family xc6slx25t-2fgg484 FPGA is used to implement it. The suggested Signed Multiplier and the traditional Booth Multiplier are evaluated in terms of area and maximum combinational path delay.

III. PROPOSED MODEL

Multiplication is the process of multiplying an integer by itself a certain number of times. A number (multiplicand) is multiplied by another number (multiplier) a certain number of times to generate a result. There are three main steps in the multiplication process:

1. Generation of a partial product.
2. Partial product reduction.
3. The final addition.

When an n-bit multiplicand is multiplied by an m-bit multiplier, m partial products are generated, and the final result is \( n + m \) bits long.

![Block diagram of 32-bit Wallace Booth Multiplier](image-url)

Figure 1: Block diagram of 32-bit Wallace Booth Multiplier

Figure 1 depicts a 32-bit Wallace Booth Multiplier block diagram, including Booth radix-4-based partial product creation, CSA-based Wallace tree for partial product reduction, and CLA-based adder for final compression.

**Modified booth’s (radix 4) algorithm:**
The exact number of partial products formed in a multiplication operation is determined by the number of bits multiplier/multiplicand is composed of. As a result, performing partial product addition is the critical bottleneck in multiplication operations and is vital to speeding up multiplication. The Modified Booth (Radix 4) Algorithm employs the partial product reduction technique to speed up multiplication. When two even ‘\( n \)’ bit numbers are multiplied, the number of partial products formed is ‘\( n/2 \)’, whereas when ‘\( n \)’ is odd, the number of partial products generated is ‘\( n+1/2 \)’. As a result, the number of partial products in Radix 4 is cut in half. Modified Booth’s Algorithm is the best way to get high-speed multipliers. The Radix-4 booth encoder's functional functioning is depicted in Table.1. It has eight different types of states, and we can get the results by multiplying the multiplicand with 0, -1, and -2 consecutively during these states. This approach examines three-bit strings at a time. The following is a list of the steps in the Radix 4 multiplication algorithm:

1. If necessary, extend the sign bit 1 place to ensure that \( n \) is even.
Add a 0 to the right of the multiplier's LSB.
Each Partial Product will be 0, +y, −y, +2y, or −2y, depending on the value of each vector.

Table 1: Booth recoding table for radix-4

<table>
<thead>
<tr>
<th>Multiplier Bits Block</th>
<th>Recoded 1-bit pair</th>
<th>2 bit booth</th>
</tr>
</thead>
<tbody>
<tr>
<td>i+1</td>
<td>i</td>
<td>i-1</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>-1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>-2</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>-1</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>-2</td>
</tr>
</tbody>
</table>

The 2's complement is used to generate negative y values. The 2's complement is employed to create negative y values, and Carry-look-ahead (CLA) fast adders are used for addition in this study. The multiplication of y is achieved by moving y to the left by one bit. In any case, only n/2 partial products are formed when designing n-bit parallel multipliers.

**Carry Look Ahead Adders (CLA):**

Rather than computing the carry bit by bit, carry look-ahead adders predict the block carry output. The propagate and generate terms are defined by the CLA blocks. The generate G term indicates whether a carry out can be generated autonomously from the inputs from within the block. The propagate term P controls whether or not the block's input carry will propagate to the output. The architecture of a one-level CLA adder is shown in Figure 2 [15]. A carry look-ahead adder's delay is proportional to n/k, where k is the number of bits per block. The CLA adder expressions are as follows:

\[
G_i = A_i \cdot B_i \quad P_i = A_i + B_i \quad (1)
\]

\[
G^* = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 \quad (2)
\]

\[
P^* = P_3 \cdot P_2 \cdot P_1 \cdot P_0 \quad (3)
\]

\[
C^* = C_4 \cdot P^* + G^* \quad (4)
\]

**Carry Save Adder (CSA):**

A carry-save adder calculates the sum of three or more binary n-bit values. The CSA's output has the exact dimensions as the input. The partial sum bits sequence is one of the two outputs, while the carry bits sequence is the other. A carry propagates adder, such as RCA, CLA, or CSelA, calculates the final total. The carry-save unit consists of n full adders; each calculates a carry and the sum of the three inputs. Let A, B, and C be the three n-bit values, respectively; the result will be a partial sum (PS) and a shift-carry (SC):

\[
PS_i = A_i \oplus B_i \oplus C_i \quad (5)
\]

\[
SC_i = (A_i \land B_i) \lor (A_i \land C_i) \lor (B_i \land C_i) \quad (6)
\]

The total sum is calculated by shifting the carry sequence SC one position to the left. We are adding a zero to the partial sum of PS's MSB.

**Partial Product Reduction:**

Most multipliers are constructed at this stage with different types of multi operand adders, which can add more than two input operands and produce two outputs, sum and carry. In high-speed designs, the Wallace tree approach is

www.turkjphysiotherrehabil.org
utilized to add incomplete products. Wallace Tree speeds up the process by minimizing the phases of sequential addition of partial products. The Wallace tree comprises many compressors that take three or more inputs and produce two outputs that are the same size as the inputs. The multipliers’ speed, size, and power consumption will be directly proportional to the compressors’ efficiency. Figure 3 shows a Wallace tree with 4:2 compressors.

Figure 3: Wallace tree structure with 4:2 compressors

A partial product generator creates partial products by multiplying each bit of the multiplier with the multiplicand. Both the multiplier and the multiplicand are 32-bit, resulting in 32 partial products. The Wallace tree structure, built up of many 4:2 Compressors, is used to add these partial products, speeding up the accumulation process. The total and carry bits obtained at the end are fed into the Carry look-ahead adder; the final carry propagates adder. Figure 4 shows the block diagram of a 32-bit Wallace tree multiplier.

Figure 4: Block diagram of 32×32-bit Wallace tree Multiplier

Modified Booth-Wallace Tree Multiplier:
The Radix-4 Booth Algorithm is used to generate partial products. This reduces the number of partial products generated to half, or $m/2$, where $m$ is the number of multiplier bits available.
These partial products are added using a Wallace tree structure comprising many 4:2 Compressors, which results in a sequence of sum and carry bits that are added using the final Carry look-ahead adder. The number of partial products to be added is reduced using the Modified Booth method. The number of sequential adding stages is lowered using Wallace Tree, resulting in increased speed. Figure 5 shows the block diagram of a 32-bit Modified Booth-Wallace tree multiplier.

**Final Addition:**
This stage is particularly critical for any multiplier because it performs the addition of operands with many bits, making a fast carry propagate adder possible. CLA (Carry-look Ahead Adder) should be used according to the specifications.

**IV. RESULTS AND DISCUSSION**

FPGA stands for field-programmable gate array, and it is a logic device with a two-dimensional array of generic logic cells and programmable switching. A digital system is described and modelled using hardware description languages like Verilog. We’re modelling multiplier design in Verilog and performing synthesis to convert HDL to modular gate-level components. The multipliers were designed with Xilinx ISE 12.1 and simulated with I-Sim. Below are the simulation results, device utilization table, and RTL schematics for all multipliers. Their area and maximum combined path delay are contrasted and studied as well.

**Figure 6:** RTL Schematic and Technology view of Fast Multiplier

Figure 6 shows the RTL Schematic and Technology view of Fast Multiplier in which it contains two 32-bit inputs namely multiplicand and multiplier, in addition to this a sign bit is used.
Figure 7 shows the RTL Schematic and Technology view of carry look-ahead adder in which it contains two 32-bit inputs namely a and b, in addition to this a carry in c\textsubscript{in} bit is used.

The multiplication based on the Radix-4 Booth method was simulated using the Xilinx 12.1 software’s I-Sim simulator and implemented on an FPGA platform, yielding the above results. Figures 8 and 9 show that the 32-bit multiplier with CLA delivers minimal delay compared to traditional tree structures. This is because while using the multiplier with CLA, the number of reduction stages was decreased. Furthermore, as the size of the input word grows larger, CLA delivers faster multiplier architectures. Another intriguing finding, though, is that while the latency drop, but the estimated logic utilization of the FPGA decreased as well, which will undoubtedly help with power reduction goals.
When the instances of Radix-2 and Radix-4 multiplication are considered, it can be inferred that Radix-4 Booth multiplication reduces the number of partial products and speeds up the multiplication process. The coding for the radix-4 modified booth multiplier is prepared in Verilog, followed by the Xilinx design tool 12.1. The first two partial products are generated directly with the help of the booth encoding table, then accumulated with the help of the carry-save adder, and finally added with the use of the CLA. Finally, different adders, such as carry select adder, carry look-ahead adder, and ripple carry adder, were utilized in the third stage of multiplication and compared. In terms of delay, the carry select adder shows a significant improvement. On the other hand, Wallace trees in accumulation stages can be employed to improve the circuit's delay.

REFERENCES