BCD Approach Based High Performance Floating Point Multiplier for DSP Applications

Pagadala Gowri Lahari and *Divya Kiran
Faculty of Engineering and Technology, M.S. Ramaiah University of Applied Sciences, Bangalore 560 054
*Contact Author E-mail: divyakiran.ec.et@msruas.ac.in

Abstract
For Applications such as Digital Signal Processing (DSP), floating point computations provide higher accuracy and improved precision. IEEE-754 standard is widely used to attain the precision, but with a trade-off of accuracy due to truncation error. For an efficient hardware implementation, complexity of the architecture is the most important element. This paper proposes a combined approach of Binary Coded Decimal (BCD) and binary computation for floating point multiplications. The developed architecture is based on formulated conditions, which speeds up the operations as well as reduces the power requirement. The developed architecture has been implemented on FPGA Virtex-6 and verified. The proposed architecture has utilized Braun’s multiplier with Kogge Stone adder for performing binary multiplication. The proposed architecture utilizes 4689 slice registers, operating at speed of 40.175 MHz for 64-bit. Proposed architecture has been compared with existing design achieving 54.6% and 11.2% better delay and resource utilization respectively.

Keyword: Binary Coded Decimal, Braun’s Multiplier, Kogge Stone Adder, Floating point, FPGA

1. INTRODUCTION
Floating-point computation is an important approach to obtain high accuracy and precision output for the applications related to high performance computing, multidimensional graphics, digital signal processing etc. Desired performance can be attained by achieving higher throughput from the combinational circuit i.e. multiplier in here. Due to the complexity involved and available possibility multiplier has been a topic of interest for decades.

IEEE-754 is widely used floating point standard with two types of representation formats [1]. Single precision of size 32-bit and double precision of 64-bit format are used for the floating point computation. For the applications such as 2D drawing, 3D printing, robot motion planning, matrix multiplication and FFT, decimal points accuracy is expected which deviates due to truncation error.

In DSP applications floating point multiplier is an imperative part. Reducing delay of a multiplier is required to achieve higher speed. The existing approaches lacks in precision due to truncation errors and also has the complex hardware architectures due to the IEEE standard formats. This paper proposes architecture for floating point multiplier combining BCD and binary approaches of computation. The proposed work has utilized Braun’s multiplier with Kogge Stone adder as binary multiplier to achieve higher speed.

The paper is organized as follows. Section 2 discusses previous works in the field of floating point multiplier. Sections 3 presents the proposed hardware architecture. Section 4 evaluates and discusses the performance of the proposed method with respect to FPGA resources utilization, minimum delay, power consumption, frequency and area. Section 5 draws the conclusion and also recommends suggestions for future extension.

2. RELATED WORKS
IEEE-754 floating point standard is one of the approaches to perform floating point operation with a major drawback of truncation error. Thus they are used to limited applications like graphics and animations [1,2]. In a sequence of floating point operation, error at each operation stage can accumulate and greatly deflect from expected output at the end of computation. In a program that draws a 2D convex polygon the error in floating point operation output could cause control path deviations that eventually result in a non-convex polygon. Other applications like 3D printing and robot motion planning divergence from expected outcome due truncation error becomes critical issue [3]. Different floating point multiplier architectures have been proposed to reduce the power consumption, area and delay like IEEE-754 floating point multiplier with booth algorithm [4], Karatsuba algorithm [6], Vedic algorithm [5] but they could not reduce the hardware complexity.

BCD has been used to eliminate the truncation error [6]. BCD approach uses element-wise computation hence the need of approximation is not there; which eliminates the truncation error. But it raises another issue of complexity as BCD multiplication is an expensive operation. In the work proposed by Kaivani [7] a reversible logic approach has been applied for developing a BCD converter using IEEE754R approach. The paper has instigated the development of the architecture using two different approaches to achieve better performance. The other work [8] shows the importance of floating point multiplication for FFT computations. It also highlights the parameters to be considered for improving the performance of the design. The work proposed by [9,10] provides an insight about the multipliers especially Braun’s multipliers. It also shows the advantages of utilizing Kogge Stone adders.
The proposed work is an attempt to resolve the issue of truncation error by eliminating the need of approximation. It also focuses on improving the speed and attaining reasonable power consumption.

3. PROPOSED FLOATING POINT ARCHITECTURE

To overcome the issues associated with truncation error, BCD approach for floating point multiplier has been hosen. The proposed architecture utilizes BCD number system and has adopted binary multiplication to minimize the computation complexity hence increasing the performance of the design.

3.1 Modes of Operation

In the proposed 2-input floating point multiplier, each input is divided in significand and mantissa and they are fed separately, thus in 4-input architecture for floating point multiplier. Based on the input feed, set of conditions has been derived and shown in Table 1.

Table 1. Conditions for operation of proposed floating point multiplier

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>M1</td>
<td>S2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>M1</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>Others</td>
<td>S1.M1XM2</td>
</tr>
</tbody>
</table>

Specific conditions formulated for designing the 64-Bit Floating point multiplier architecture are as follow:

- **Condition 1:** If “S1” and “S2” is equal to “0” and “M1” and M2 is equal to “0” then, output is Significand = “0” and Mantissa = “0”
- **Condition 2:** If “S1” and “S2” is equal to “0” and “M1” and M2 is equal to “0” then, output is Significand = “0” and Mantissa = “M1 x M2”
- **Condition 3:** If “S1” and “S2” is not equal to “0” and “M1” and M2 is equal to “0” then, output is Significand = “0” and Mantissa = “0”
- **Condition 4:** If “S1” and “S2” is not equal to “0” and “M1” and M2 is not equal to “0” then, output is Significand and Mantissa, i.e. Normal floating point operation

3.2 Proposed Architecture

Based on the conditions shown in Table 1, architecture for performing floating point operations has been developed. The proposed architecture of Floating point multiplier is shown in Fig. 1.

The developed architecture, consist of a “Control Unit” block which controls, three operational pseudo blocks namely “Significand Block”, “Mantissa Block” and “Normal Operation Block”. “Control Unit” activates and deactivates the respective blocks based on the formulated conditions as shown in Table 2. Activating the selective blocks reduces power consumption and improves the computation speed.

Table 2. Blocks of proposed floating point multiplier

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significant</td>
<td>Mantissa Block</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
</tbody>
</table>

BCD approach provides an ease to handle the data with an ease than the binary approach. In the proposed approach a Braun’s multiplier has been used for the binary multiplication. The BCD inputs are first fed in the control unit block, where the formulated conditions are checked and then the respective blocks are activated. Prior to multiplier block the BCD inputs are concatenated and converted into binary and later the obtained product is converted back into the BCD. In the final stage the BCD output packet is separated into significand and mantissa output.

3.3 Modified Braun’s Architecture

For binary multiplication in the proposed architecture, Braun’s multiplier has been utilized and modified to improve the performance. The popular combination of Ripple Carry Adder (RCA) and the full adder has been modified with Kogge Stone Adder (KSA) and full adder using multiplexer as shown in Fig 2. It has been observed
that 1-bit full adder using 2:1 mux attains 0.197 ns lesser delay compared to 1-bit full adder using logic gates.

Fig. 2 Full adder using 2:1 multiplexers

To perform 20 digit multiplication 67x67 bits multiplier has been developed as shown in Fig 3. The implemented architecture utilized 4355 full adder cells and one 2-input koggestone adder, each input of size 66 bits.

Fig. 3 Modified Braun’s multiplier architecture

To evaluate the performance of the modified multiplier, the proposed floating point architecture has been developed for different bit sizes and analyzed. The performance comparison of architecture using RCA and KSA has been tabulated in Table 3.

The result obtained shows that KSA based floating point multiplier has 11.74 ns lesser delay than RCA based multiplier.

4. RESULTS DISCUSSION

The developed architecture has been implemented on virtex-6 and the simulation result is shown in Fig.4. The developed architecture has been targeted for 20 digits, which is 4 bits each. Hence, to overcome the computation issue of large number of bits, arithmetic operations has been carried out in binary format. To maintain the visibility and handling ease the input and output are in BCD form. The simulation results shows the 20 digit input significand and mantissa and 40 digit output.

Fig. 4 Simulation result

Input 1: 9999999999.9999999999
Input 2: 9899765678.7539615963

Expected output:

98997656787539615962.01002343212460384037

The obtained results in BCD form match the expected results.

Table 4 shows that delay obtained by proposed architecture is 54.6% better than existing architecture. Resource utilisation of the proposed architecture is 11.2% better than existing architecture.

Table 4. Comparison of proposed architecture with existing architecture implemented on Virtex-6

<table>
<thead>
<tr>
<th>Proposed Floating Point Multiplier</th>
<th>Delay(ns)</th>
<th>Power(mW)</th>
<th>Max Operating Frequency(MHz)</th>
<th>No. of Slice Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit (RCA)</td>
<td>5.603</td>
<td>145</td>
<td>178.489</td>
<td>120</td>
</tr>
<tr>
<td>16-bit (RCA)</td>
<td>8.394</td>
<td>178</td>
<td>119.130</td>
<td>288</td>
</tr>
<tr>
<td>32-bit (RCA)</td>
<td>19.114</td>
<td>220</td>
<td>52.317</td>
<td>1223</td>
</tr>
<tr>
<td>64-bit (RCA)</td>
<td>36.631</td>
<td>358</td>
<td>27.299</td>
<td>4622</td>
</tr>
<tr>
<td>64-bit (KSA)</td>
<td>24.891</td>
<td>527</td>
<td>40.175</td>
<td>4689</td>
</tr>
<tr>
<td>Proposed 64-bit Floating Point Multiplier (KSA)</td>
<td>24.891</td>
<td>196.198</td>
<td>54.899</td>
<td></td>
</tr>
<tr>
<td>64-bit Floating Point Multiplier using Booth Algorithm[6]</td>
<td>4689</td>
<td>6115</td>
<td>5285</td>
<td></td>
</tr>
</tbody>
</table>
5. CONCLUSION

Floating point multipliers play a critical role in any high precision requiring applications. The floating point architectural should occupy less area and should consume less power. Floating point multiplier follows IEEE-754 standard. Accuracy is an issue in existing architecture due truncation caused in the design. Floating point multiplier using BCD form is been proposed to solve the accuracy issue since it does not have any truncation error. Multiplier of 8-bit, 16-bit, 32-bit and 64-bit have been implemented to analyse the feasibility and efficiency of the proposed floating point multiplier technique. The proposed 64-Bit floating point multiplier is working at speed of 40.175 MHz and obtained minimum time delay is 24.891 ns. The obtained results are with the trade-off of larger bit size. Furthermore, this work will be extended to reduce the bit requirements to improve the speed and the power requirements.

REFERENCES


