Design And Implementation Of An Efficient Single Precision Floating Multiplier Using Vedic Multiplication

Bhavesh Sharma, Amit Bakshi

Abstract—This paper contains design of a single precision floating point multiplier by modifying the proposed architecture[6] and then comparing the different floating point multiplier architecture for the various performance parameters. The designs are modeled in Verilog HDL and synthesized based on the TSMC 180nm standard cell library. Comparisons are based on the synthesis result obtained by synthesizing all the multiplier using Cadence Encounter RTL Compiler.

Index Terms—Floating Point, Floating Point Multiplier, Vedic Multiplication.

1 INTRODUCTION

Multipliers have an important part of the modern electronic years. Floating point multiplier’s can be found in electronic systems that run complex calculations especially in DSP processor. In parallel multipliers the number partial product addition required determines the performance for the multipliers.

A study in [1] showed that array multiplier has the largest area and delay when compared to other multiplier design, while the Wallace tree multiplier has less area and delay compared to the array multiplier. In [2] the study stated that depending on the application either booth multiplier and wallace multiplier can be used since Booth Multiplier uses less area compared to wallace tree multiplier but wallace tree multipliers dissipates less power. In study [3] stated that in Vedic Multiplier when ripple carry adder is used the area required is less compared to the use of carry save adder.

2 FLOATING POINT MULTIPLIER

The floating point multiplier represented in IEEE 754 format is divided in four unit.

i. Multiplier Unit
ii. Exponent Calculation Unit
iii. Sign Calculation Unit
iv. Control Unit

The Mantissa of the resultant is calculated using a 24X24 bit multiplier. The control unit is used to raise overflow and under flow conditions. The exponent calculation is done by four 8 bit ripple carry adder in the architecture[6].

In the exponent bits are biased to 127 that

\[ EA = Ea + 127 \] (1)
\[ EB = Eb + 127 \] (2)

That final resultant is obtained by performing by

\[ Er = Ea + Eb \] (3)

Now Er is again biased to 127.

\[ ER = Er + 1 \]

Fig. 1. Architecture For Floating Point Multiplier[6].
3 MODIFIED FLOATING POINT MULTIPLIER

The basic working of the modified architecture is the same as the proposed architecture[6] with the only difference in the exponent unit. The Mantissa of the resultant is calculated using a 24X24 bit multiplier. The control unit is used to raise overflow and underflow conditions. The Exponent calculation is done by using single 8 bit carry save adder. The two 127 biased exponents are added along with the 2's complement of 127 which gives the resultant exponent. The 2's complement of 127 is added because we want the resultant output to be biased to 127. That is if ER is the final result then

$$ER = EA + EB - 127$$  \hfill (5)

4 FLOATING POINT MULTIPLICATION

The Fig 1 shows the architecture for the floating point multiplier. Consider the multiplication of two floating point numbers A and B, where A = -131.25 and B = -7.75. The IEEE representation of the two numbers are

<table>
<thead>
<tr>
<th>SIGN</th>
<th>EXPONENT</th>
<th>MANTISSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>10000110</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>10000001</td>
</tr>
</tbody>
</table>

The mantissa is expressed in 23 bits, so they are normalized by placing a 1 at the MSB. The two normalized mantissa are:

- 100000110100000000000000
- 111100000000000000000000

The resultant of the multiplication is a 48 bit number which is then again normalized to 23 bits by eliminating the most significant 1.

We obtain the result as:

0 10001000 1111111001001100000000000000

This resultant is converted to 1017.1875

5 VEDIC MULTIPLIER

Vedic multiplier is based on the vedic multiplication sutra. These sutras are used for the multiplication of two numbers in decimal system. The multiplier is based on Urdhava Triyakbyam Sutra. In this concept the generation of partial product can be done and then parallel addition of these partial product is done.

For an example let us take a 3X3 multiplier which is shown in Fig. 3. Consider the numbers X and Y where

A = X1X2X3 and Y = Y1Y2Y3.

The operations are:

$$Su_0 = X_3Y_3$$  \hfill (6)

$$Ca_1Su_1 = X_3Y_2 + X_2Y_3$$  \hfill (7)

$$Ca_2Su_2 = Ca_1 + X_1Y_3 + X_2Y_2 + X_3Y_1$$  \hfill (8)

$$Ca_3Su_3 = Ca_2 + X_1Y_2 + X_2Y_1$$  \hfill (9)

$$Ca_4Su_4 = Ca_3 + X_1Y_1$$  \hfill (10)

Now the final result of multiplication of X and Y is

$$Ca_4Su_4Su_3Su_2Su_1Su_0.$$  \hfill (11)

The 24X24 bit multiplier is designed by using four 12X12 bit multipliers. The 12X12 Multiplier is again designed using four 6X6 multipliers. The 6X6 Multiplier is again designed using four 3X3 multipliers. So the multiplier uses hierarchical structure to reduce the number of partial product generation.
6 RESULT AND DISCUSSION

In [7] the proposed architecture was used design a floating point multiplier using different multiplier algorithms for the 24X24 multiplier. Upon comparison it was concluded that floating point multiplier with vedic multiplier has the least area and power delay product . The same vedic multiplier is therefore used for comparison in the modified architecture.

<table>
<thead>
<tr>
<th>TABLE 1</th>
<th>PERFORMANCE COMPARISON OF THE TWO ARCHITECTURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Floating point multiplier [7]</td>
</tr>
<tr>
<td>Delay(ns)</td>
<td>8.0</td>
</tr>
<tr>
<td>Area</td>
<td>71946</td>
</tr>
<tr>
<td>Power Dissipation (mw)</td>
<td>20.305</td>
</tr>
<tr>
<td>A.D(10^-3)</td>
<td>0.575</td>
</tr>
<tr>
<td>P.D(10^-12)</td>
<td>162.44</td>
</tr>
<tr>
<td>Power density(10^-7)</td>
<td>2.822</td>
</tr>
</tbody>
</table>

6.1 Delay Comparison
Floating point multiplier with modified architecture has a larger delay compared to the architecture used in[7]. The increase in delay is 15%.

6.2 Area Comparison
Floating point multiplier with old architecture[6] has a larger area compared to the modified architecture proposed because of the additional adders used. Since in the modified architecture a single 8 Bit carry save adder is used due to which area is reduced by 9.5%.

6.3 Power Comparison
Due to the reduction in area the power of the modified architecture is reduced by 26%.

7 CONCLUSION
Based on the experimental data gathered as seen in table 1, the modified architecture is better in aspect of area , power , power delay product and power density . Although with a slight increase in delay we achieve a low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture. This is a known trade off between low power and low area architecture.
REFERENCES


