Abstract— Floating point multiplication is one of the most frequently used arithmetic operation in a wide variety of applications, but the high memory and speed requirement of the IEEE-754 standard floating point multiplier prohibits its implementation in many systems which requires fast computing, such as in wireless sensors and in real time applications. This paper presents an improved algorithm for the floating point multiplier which provides a better performance in terms of time requirement for the implementation. This floating point multiplier is implemented and synthesized on Xilinx Spartan-3E FPGA.

Index Terms—Booth algorithm, Floating-point arithmetic, Multiplication, Toom Cook, Xilinx.

I. INTRODUCTION

Many Basic microprocessors / microcontrollers are unable to handle real number arithmetic. They can only manipulate integers. For real number arithmetic operations it is required to connect a floating point arithmetic unit. Floating point unit allows processors to handle rational, as well as irrational numbers to some extent. Floating point numbers are used to represent very small to very large numbers. The floating point operations are slightly more complex than those for standard integer numbers. A tremendous variety of algorithms have been proposed for floating point systems. Implementation is usually based on refinements of the few basic algorithms as presented here. In addition to choosing algorithms for arithmetic operations, the computer architect must make other choices also. Which application the floating point unit is going to work for? What precision is required? How exceptions should be handled? Which rounding technique should be used?

Our discussion of floating point will focus on the IEEE floating point standard only, because of its rapidly increasing acceptance. [1]The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). A finite number in IEEE is described by three integers:

\[ s = \text{a sign (zero or one)}, \]
\[ c = \text{a significand}, \]
\[ q = \text{an exponent}. \]

The value of the number is

\[ (-1)^s \times c \times b^q \]  

Where b is the base (2 or 10), also called radix.

In this format half (16 bits), single (32 bits), double (64 bits) and quad (128 bits) precisions are considered. The table given below summaries them.
Table 1. Summary of different floating point formats

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Half Precision</th>
<th>Single Precision</th>
<th>Double Precision</th>
<th>Quad Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias, E-e</td>
<td>15</td>
<td>127</td>
<td>1023</td>
<td>16383</td>
</tr>
<tr>
<td>sign bit</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>exponent bits</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>significand bits</td>
<td>10</td>
<td>23</td>
<td>52</td>
<td>112</td>
</tr>
</tbody>
</table>

In this paper single precision format is considered.

Fig. 1. Single precision floating point format

Single precision gives from 6 to 9 significant decimal digits precision. Sign bit gives the sign of the number, which is the sign of significand also. Exponent is unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 binary32 definition. The significand has 23 fraction bits to the right of binary point and an implicit leading bit having value 1 unless exponent is having all zeros. Thus only 23 bits are used for significand in the memory but the precision is 24 bits.

II. FLOATING POINT MULTIPLICATION

A basic multiplier circuit deduced from proposed algorithm is shown in Figure 2. It consists of five major blocks 24 x 24 multiplier, 8 bit adder, guard digit and sticky bit generation, normalize and rounding. Multiplication process starts by checking special cases i.e NaN, Infinity, zero. If a special case occurs then the multiplication output is directly generated without doing any further processing. If not then significands are multiplied by multiplier unit, exponents are added together and sign bits are XORed.[2] Guard Digit and Sticky bit generation block computes the sticky bit and concatenates it with 26 MSB bits of product of significands coming out of multiplier unit. Normalization block updates the significand and the exponent if the value of product is greater than or equal to 2. Rounding block rounds the 27 bits significand to a 24 bits significand using round to the nearest, tie to even method [3].

For better understanding let’s take an example of two 2-digit numbers in radix-10 say m, n. So,

\[ m = 10a + b \]
\[ n = 10c + d \]

where a, b, c and d are decimal digits. The product is

\[ m \times n = (10a + b) \times (10c + d) \]
It requires four single digit multiplication and three additions. Karatsuba's idea was to reduce the number of multiplication to three [6].

He computed \((a \cdot d + b \cdot c)\) in (5) as

\[a \cdot c + b \cdot d - (d - a) \cdot (d - c)\]

He computed the following products

\[
\begin{align*}
    p &= a \times c \\
    q &= c \times d \\
    r &= (b - a) \times (d - c)
\end{align*}
\]

\[
m \times n = 100p + 10(q + r) + q
\]

The total number of operations is three multiplications, three subtractions and three additions.

Now Toom Cook algorithm is another algorithm which is a modification and generalization of Karatsuba's method [7]. To understand it let us consider two n-digit numbers \(a\) and \(b\) in base B number system. We take \(x = B^{n/2}\) and express

\[
\begin{align*}
    a &= a_1x + a_0 \\
    b &= b_1x + b_0
\end{align*}
\]

We treat \(x\) in (10) and (11) as a formal variable, so we have the polynomials

\[
\begin{align*}
    a(x) &= a_1x + a_0 \\
    b(x) &= b_1x + b_0
\end{align*}
\]

The product,

\[
c(x) = a(x)b(x) = c_2x^2 + c_1x + c_0
\]

Where

\[
\begin{align*}
    c_2 &= a_1 \cdot b_1 \\
    c_1 &= a_1 \cdot b_0 + a_0 \cdot b_1 \\
    c_0 &= a_0 \cdot b_0
\end{align*}
\]

The coefficients \(c(x)\) can be uniquely determined by evaluating it at three different points. This gives us the coefficient of products in Karatsuba-Ofman algorithm. Toom-Cook algorithm uses higher order polynomial to represent the numbers. We are using Toom-3 algorithm in which we represent a number in quadratic form. Let \(a\) and \(b\) be two n-digit base B numbers. We take \(x = B^{n/3}\).

Corresponding polynomial representation is

\[
\begin{align*}
    a(x) &= a_2x^2 + a_1x + a_0 \\
    b(x) &= b_2x^2 + b_1x + b_0 \\
    c(x) &= a(x)b(x) = c_4x^4 + c_2x^2 + c_2x + c_0
\end{align*}
\]

The coefficients of product \(c(x)\) can be determined by evaluating following five multiplication.

\[
\begin{align*}
    p_4 &= a_2 \cdot b_2 \\
    p_3 &= (a_2 + a_1 + a_0) \cdot (b_2 + b_1 + b_0) \\
    p_2 &= (a_2 - a_1 + a_0) \cdot (b_2 - b_1 + b_0) \\
    p_1 &= (4a_2 - 2a_1 + a_0) \cdot (4b_2 - 2b_1 + b_0) \\
    p_0 &= a_0 \cdot b_0
\end{align*}
\]

These products are further used to evaluate coefficients of \(c(x)\) as

\[
\begin{align*}
    c_4 &= p_4 \\
    c_3 &= \frac{1}{6}(12p_4 + p_3 + 3p_2 - p_1 - 3p_0) \\
    c_2 &= \frac{1}{2}(-2p_4 + p_3 + 3p_2 - 2p_0) \\
    c_1 &= \frac{1}{6}(-12p_4 + 2p_3 - 6p_2 + p_1 + 3p_0) \\
    c_0 &= p_0
\end{align*}
\]

The time complexity of multiplying \(n/3\) digit number by 2, 3, and 12 is \(O(n)\). The main contribution to time complexity comes from five multiplications of \(n/3\) digit integers. These multiplications can be done recursively in a similar manner until we come to a stage where \(n\) is so small that further processing will not be efficient if we will use Toom-Cook algorithm. So from there we will use Booth algorithm.

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation. We will demonstrate the Booth algorithm with the following example:

Example: \(-85_{10} \times 45_{10}\)

Above two numbers can be represented as \(M = (84)_{10} = (01010100)_{2}\) and \(R = (45)_{10} = (00101101)_{2}\) in radix 2 number system. Here we need three register let us say A, S and P of 2n+1 bits i.e. \(n = 8\) here. Initially for A fill the most significant bits with the value of \(M\), for S fill the most significant bits with the value of -M and fill their remaining bits with zero. So,

\[
A = (01010100 00000000 0)_2
\]
\[ S = (10101100 \ 00000000 \ 0)_2 \]  

(30)

For \( P \) fill the most \( n \) significant bits with zero. To the right of this add the value of \( R \). Fill the least significant bit with zero. So,

\[ P = (00000000 \ 0101101 \ 0)_2 \]  

(31)

Determine the two least significant bits of \( P \). If they are “01” find the value of \( P + A \) otherwise find \( P + S \) if they are “10”. For “00” or “11” do nothing. Now arithmetically shift the value of \( P \) obtained above. After repeating these process \( n \) times, product can be obtained from \( P \) by dropping its least significant bit.

IV. RESULTS AND DISCUSSION

The proposed architecture of multiplier is compared with array multiplier as well as booth multiplier on Xilinx Spartan3E FPGA.

i. Methodology

We used Xilinx version 8.1i. The FPGA device XC3S400 was used with package PQ208 and speed grade -4. The design was synthesized using synthesis tool XST (VHDL/Verilog) and was simulated using ISE simulator (VHDL/Verilog).

ii. Simulation

In simulation Figure we are evaluating

\[ 2.5648 \times 5.68742 \]

Single precision floating point representation of 2.5648 and 5.68742 is

\[ 2.5648 = 0100000001.00100010010110101111 \]

\[ 5.68742 = 01000001011010110111111101011000 \]

On multiplying these two numbers we should get

\[ 2.5648 \times 5.68742 = 14.587094816 \]

For single precision we take only 6 significand digits after decimal. So our answer in single precision after rounding using an appropriate method should be 14.587094.

Single precision floating point representation of 14.587094 is

\[ 14.587094 = 0100001011010010101010111110 \]

This is what we are getting in the simulation. So the answer is correct.

iii. Result

The table given below shows result of all the three multipliers, the total number of LUTs used and the delay in nanoseconds of each multiplier are reported. We can see that the minimum period has been reduced to 18.39 Nano seconds.

Table 2. Comparison of different algorithms

<table>
<thead>
<tr>
<th></th>
<th>Array Multiplier</th>
<th>Radix -8 Booth Multiplier</th>
<th>Proposed multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>773/4656</td>
<td>1996/4656</td>
<td>2271/3584</td>
</tr>
<tr>
<td>Number of 4 input LUT’s</td>
<td>1508/9312</td>
<td>4059/9312</td>
<td>4020/7168</td>
</tr>
<tr>
<td>Minimum period</td>
<td>31.855ns</td>
<td>22.282ns</td>
<td>18.390ns</td>
</tr>
<tr>
<td>Maximum frequency</td>
<td>31.392MHz</td>
<td>44.879MHz</td>
<td>54.377MHz</td>
</tr>
</tbody>
</table>

V. CONCLUSION

A 24*24 significand multiplier for IEEE single-precision multiplication in Xilinx FPGA is presented. The proposed multiplier first applies Karatsuba to the given inputs and then the Booth algorithm. We observed that this algorithm uses less LUT’s as compared to array multiplier as well as Booth multiplier. The proposed multiplier is faster than array multiplier and is comparable with Booth multiplier in terms of time constraint.
REFERENCES


[7] Richard P. Brent and Paul Zimmermann, “Toom-cook multiplication” and “floating point arithmetic” in Modern Computer Arithmetic, version 0.1 of October 25, 2006

AUTHORS

1. Sushma Wadar completed her BE in 1999 from Walchand College of Engineering. She worked as lecturer at DKTE, Ichalkaranji (2001-2003). In 2010 completed her Master’s degree from Walchand College, Sangli. She is working as an Assistant professor at Army Institute of Technology since 2003. Currently, she is pursuing her PhD from Savitribai Phule Pune University under the guidance of Dr Y V Chavan. Her area of interest is Embedded system design and its applications. She has published research papers in 9 International and about 11 National conferences.

2. Dr. Y.V. Chavan has completed his BE in 1989. He worked as lecturer at Pravara Rural Engg., College, Loni (1990-1997) as senior lecturer at Amrutvahini College of Engg., Sangamner (1997-2005). On completion of M. Tech. in 1999 (VNT) promoted to Assistant Professor and In charge Head E&TC dept. and then as Assistant Professor at MAE, Alandi, Pune. Presently He is Professor & Vice Principal at JSPM’s Rajarshi Shahu College Of Engineering after PhD from SGSITS, Indore under RGPV, Bhopal. His area of interest is Computer Vision, Machine Vision system modeling and simulation. He has published 13 research papers in
international Journals and about 22-papers in the National and International Conferences. Out of wide teaching experience he has authored a book on Computer Network. He also worked counselor of IEEE student branch for 14-years. He is Life member for ISTE, fellow of IETE, New Delhi and Senior IEEE Member.

3. Avinash Patil have completed his BE in 2003 from Shivaji University. He worked as lecturer at Bharati Vidyapeeth COE, Pune (2004-2005). In 2007 he completed his Master’s degree from Bharati Vidyapeeth and is working as an Assistant professor at Army Institute of Technology since 2005 and Robotics club and R&D cell In charge of AIT. His area of interest is VLSI design and robotics. He has published research papers in 4 international and about 15 national conferences.