Implementation of an Effective Self-Timed Multiplier for Single Precision Floating Values with Carry-Look Ahead Adder

Shradda Awate[1], Veerabhadrappa S.T[2]

[1]M.Tech Student, Department of Electronics and Communication, VLSI Design and Embedded Systems, JSS Academy for Technical Education, Bengaluru, India.
[2]Associate Professor, Department of Electronics and Communication, JSS Academy for Technical Education, Bengaluru, India.

Abstract - *The digital signals are represented also in the form of floating point values. Processing of data or signal involves the mathematical operations to get the desired output of the system. This paper presents a self-timed multiplier for 32-bit floating point values with carry look ahead adder based on IEEE754 standards using VHDL, implemented on Spartan-3E FPGA Board.*

Key Words*: *Floating point multiplier, IEEE754 floating-point representation, carry look ahead adder, VHDL, FPGA.

1. INTRODUCTION

In a wide range of the DSP application, FIR filter, and so forth need floating point number arithmetic [1]. Floating point representation is the possible way to represent real numbers on computer. Multiplications of the floating point values are useful in applications where a substantial dynamic range is required. The floating point multiplier helps to multiply two single precision floating point values on FPGA Spartan-3E board based on IEEE 754 standards.

The real numbers on UNIX, Linux, Mac and windows are represented using IEEE 754 standards floating point number representation [2]. The self-timed multiplier for 32-bit floating point consists of multiplexer, adder, shifter, normalize and subtract units. Salty Beohara et al say's that designing the floating multiplier block with synchronous logic has many disadvantages such as latency is more, throughput is less, higher power consumptions and complex clock distribution network [2]. These disadvantages are overcome by using the asynchronous approach. In asynchronous method, there is no necessity for clock synchronization. The self-timed multiplier works in all the environmental and operating conditions. Fu-Chiung et all demonstrated a self-timed multiplier is much more faster than the multiplier built using synchronous logic [3]. Since this asynchronous approach render solutions to all these problems, hence will do the asynchronous logic to make self-timed multiplier in this paper.

2. IEEE FLOATING POINT REPRESENTATION

The floating point representation in computer is first introduced by IEEE in 1985. According to the Michael L. Overton floating Point Representation method is used to store real number on computer [4]. The floating point numbers are represented by using single precision format (32-bit floating number) and double precision format (64-bit floating number).

Single Precision floating point format is of 32-bitsand it is composed of three fields, such as sign field, exponent field and mantissa field [4]. 1 bit is assigned for sign field, 8 binary bits are assigned for exponent field and 23 binary bits are assigned for mantissa field.

Sign Bit: - sign of the number depends on the sign bit.. If the sign bit is 1, the floating point number is negative and else is positive[4].

Exponent Field: - This field is used to represent absolute value of integer exponents. To find out the stored exponent a bias value is summed with the exponent. Bias value for exponent of single precision is 127 and the bias value for exponent of double precision is 1023.

Mantissa Field: - Mantissa is also called as significant which represents the precision bits of the real number. Binary "1" is appended to mantissa in this representation.

Figure 1 shows the single precision floating point format.

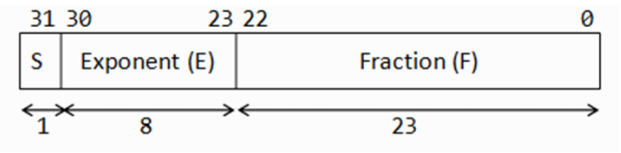

Figure 1: IEEE single precision floating point format

IEEE single precision floating point number is represented in this format as follows,

$$X = (-1)S * 2(E - Bias) * (1.M) \quad (1)$$

Where

$$M = b222\text{-}1 + b212\text{-}2 + \ldots\ldots\ldots\ldots + b12\text{-}22 + b02\text{-}23 \quad (2)$$

Bias for 32-bit floating point number = 2(8-1) - 1 = 127.

And S can be 1 or 0 depending on number.

As similar to single precision floating point numbers, double precision floating point numbers also consists of the three fields, they are sign field, exponent field and mantissa field [4]. 1 bit is assigned for sign field, 11binary bits are assigned for exponent field and 52 binary bits are assigned for mantissa field.

Figure 2 shows the double precision floating point format.

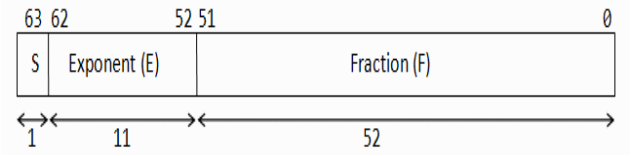

Figure 2: IEEE double precision floating point format

IEEE double precision floating point number is represented in this format as follows,

Bias for 32-bit floating point number = $2^{(10)} - 1 = 1023$.

3. MULTIPLICATION OF TWO FLOATING VALUES

The following steps are applied to multiply single precision floating point numbers.

1) Convert decimal value to its appropriate binary.
2) Converting binary representation of two operands to standard floating format.
3) Identifying the sign bit of the result by XOR-ing the sign bits of operands.
4) Multiplying the mantissa fields of both the operands along with the hidden '1'.
5) Sum of the exponents of both the operands is computed and then subtracted with the bias value.

Figure 3 shows the basic block diagram of the floating point multiplier.

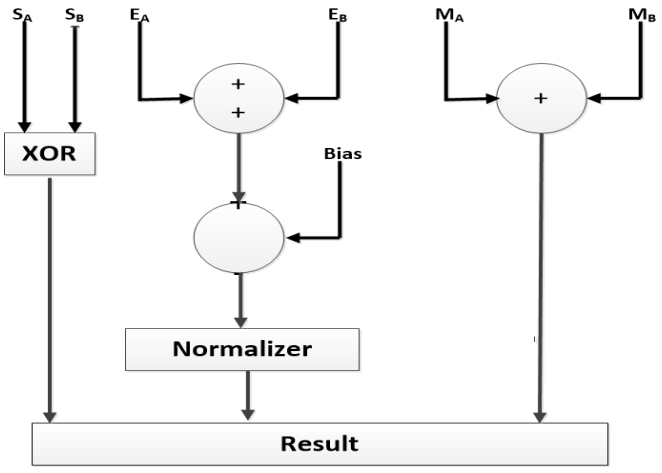

Figure 3: floating point multiplier.

Demonstration of multiplication of two 32-bit floating point numbers based on IEEE 754 standard using the above steps.

A = 6.25 and B = 585.25

1) Binary Representation of above two operands are,

A = 110.01

B = 1001001001.01

2) IEEE 754 single precision representation of the operands

A = 01000000110010000000000000000000

B = 01000100000100100101000000000000

3) Two sign bits of multiplier and multiplicand are XOR'd to get the resultant sign bit. Hence in this case sign bit of result is 0.

4) Mantissa multiplication yields resultant mantissa bits after normalizing it. In this case final output mantissa is, 11001001001110100000000.

5) Now exponent bits are found out by addition and subtraction operations. Therefore, final result of multiplication in IEEE 754 standard is as shown in Figure 4.

0	10001010	11001001001110100000000

Figure 4: Result of multiplication.

AxB = 6.25 x 585.5 = 1110010010001.1101 = 657.8125

Flowchart for the floating point numbers multiplication is shown in figure 5,

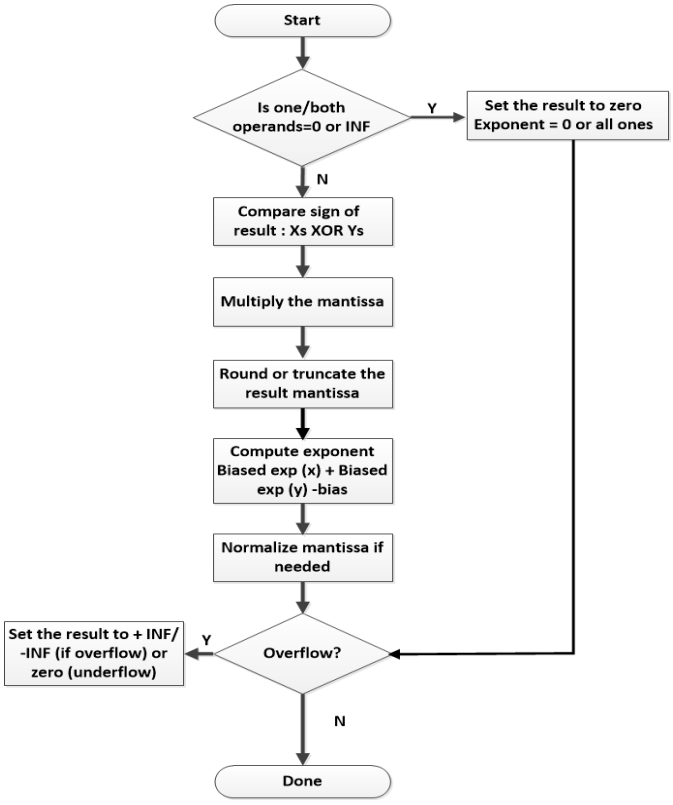

Figure 5: Flowchart

4. METHODOLOGY

The resultant exponent is obtained by simply adding operation which is done by using a single asynchronous carry look ahead adder. The carry-look-ahead adder also termed as fast adder is used in this multiplier. The entire operation of the system becomes self-timed as self-timed CLA is used to build the self-timed multiplier. The proposed multiplier is termed as self-timed multiplier because the self-timed carry-look-ahead adder is used in this paper.

In order to perform addition operation of two binary exponent bits of two floating values a single carry look ahead adder is used. The reason behind using this fast adder is that the addition process is faster than that in ripple carry adder. In carry-look-ahead adder carry signals of subsequent adder stages are computed faster. The carry bit in self-timed carry-look-ahead adder is computed by using the input bits.

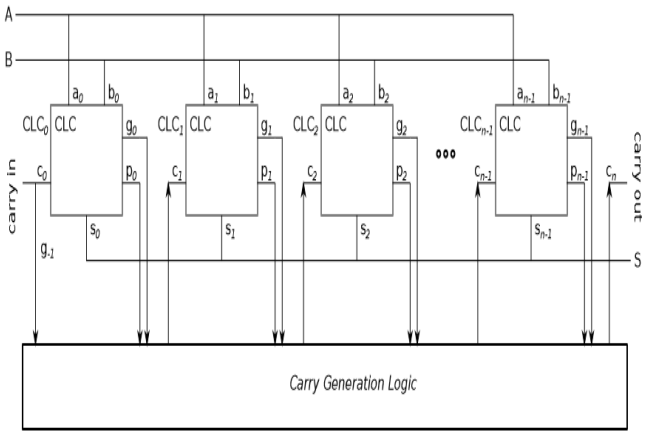

Figure 6: Structure of 4-bit carry look ahead adder

Carry look-ahead adders structure is as shown in the Figure 6. It consists of mainly three blocks namely, Propagate/generate generator, Sum generator and carry generator

The general expression of the carry look ahead adder is as follows,

$$P_I = A_I * B_I \quad \text{Carry propagate} \quad (3)$$

$$G_I = A_I * B_I \quad \text{Carry generate} \quad (4)$$

$$S_I = P_I * C_{I-1} \quad \text{output sum} \quad (5)$$

$$C_{I+1} = G_I + P_I * C_I \quad \text{carryout} \quad (6)$$

From above expressions, it is clear that present carry-in doesn't depend on its previous carry-out. Hence the general expression for carry is,

$$C_{I+1} = G_I + P_I G_{I-1} + P_I P_{I-1} G_{I-2} + \ldots\ldots + P_1 P_0 C_0 \quad (7)$$

5. RESULTS AND DISCUSSIONS

Figure 7 shows the block diagram of the self-timed multiplier for two 32-bit floating point numbers using VHDL. Subsequent operations are performed to get the output of the multiplication process which involves various steps such as normalizing, addition, subtracting, multiplying, XOR operation.

The VHDL code for the self-timed multiplier is synthesized and simulated by utilizing Xilinx ISE 14.4 software. The Xilinx tool is used to design circuit based on the users description and enables to simulate, place and route.

FPGA Spartan-3A board is used to execute the VHDL code for self-timed floating point multiplier. The code is downloaded from host to the Spartan-3A board via USB port. Field programmable gate array (FPGA) is used to debug the VHDL code.

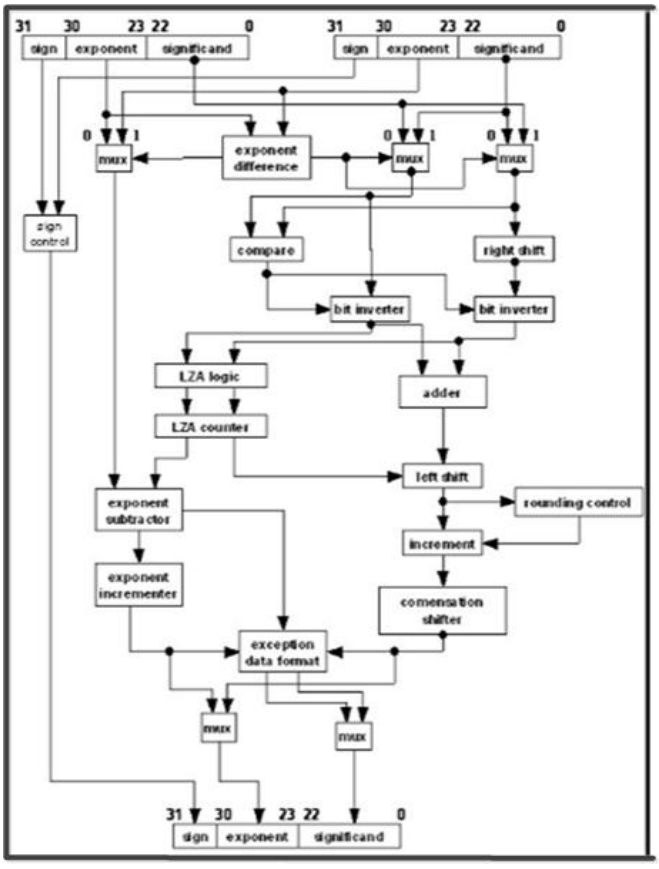

Figure 7: The block diagram of the overall project.

Table - I shows the results of the various two 32-bit floating point number multiplier. Figure 8 shows the sample input and output waveforms for the multiplication of numbers 445.65 and 745.78. The proposed self-timed multiplier achieves better precision. The precision of the self-timed multiplier is good as compared to multiplying 8-bits of floating point multiplier.

Table -1: Multiplier output

A	B	Output
5.25	286.75	1505.4375
6.25	585.25	3657.8125
23	12	276
44	5	220
9	5	45

Figure 8: Input and output waveforms

6. CONCLUSION AND FUTURE SCOPE

The Simulation result demonstrates that multiplier with self-timed Carry Look Ahead adder successfully multiplies two single precision floating values. The self-timed multiplier works correctly under normal, overflow and underflow conditions. In future this self-timed multiplier can be extended for multiplying two 64-bit floating point data format. The self-timed multiplier can also be built for multiplying more than two floating point number system.

ACKNOWLEDGEMENT

The authors would like to thank JSS Academy for Technical Education, Bengaluru for providing an opportunity to learn and work on a project at Cranes Varsity Pvt. Ltd.

REFERENCES

1. Karthik.S, Sunilkumar B.S "Implementation of Floating Point Multiplier Using Dadda Algorithm" International Journal of Electrical, Electronics and Computer Systems (IJEECS).

2. Salty Beohara and Sandip Nemade "VHDL Implementation of Self-Timed 32-Bit Floating Point Multiplier with Carry Look Ahead Adder". Electronics and Communication Engineering, Technocrats Institute of Technology (TIT) Bhopal, India.

3. Fu-Chiung Cheng Stephen H. Unger Michael Theobald Wen- Chung Cho "Delay-Insensitive Carry-Look ahead Adders", VLSI Design, 1997. Proceedings. Tenth International Conference on4-7 Jan 1997.

4. Michael L. Overton "Floating Point Representation", http://homepage.cs.uiowa.edu/~atkinson/m170.dir/overton.pdf

5. Omid Sarbishei and Katarzyna Radecka "On the Fixed-Point Accuracy Analysis and Optimization of FFT Units with CORDIC Multipliers", 2011 20th IEEE Symposium on Computer Arithmetic