International Journal of Scientific Research and Management (IJSRM)
||Volume||5||Issue||05||Pages||5331-5335||2017||
Website: www.ijsrm.in ISSN (e): 2321-3418
Index Copernicus value (2015): 57.47 DOI: 10.18535/ijsrm/v5i5.08

Design and Analysis of Multimode Single Precision Floating Point Arithmetic Unit Using Verilog

Sachin saraswat[1], and Sunita Malik[2]
Deenbandhu chhotu ram university of science & technology Murthal (sonepat)

Abstract—This Paper Presents a Design and Analysis of Multimode Single Precision Floating Point Arithmetic Unit Using VERILOG Hardware Description Language on FPGA. The multimode floating point arithmetic unit have addition, subtraction, multiplication and division operations. The device used is Zed Board Zynq Evaluation and Developed Kit (xc7z020clg484-1) on which the proposed design will be physically verified. We design and analyse the efficient multimode floating point arithmetic unit for IEEE 754 floating point number system, which gives a better implementation in terms of area of hardware. We have four separate units for four different arithmetic operations, by combining addition and subtraction unit into one and multiplication and division unit into one and by efficient optimization. The result of this combination is to reduce the number of LUTs used in FPGA. Thus the total area of hardware required will be reduced. The LUTs reduction is 14% and area reduction is 19%.

Keywords— Floating Point, Look Up Tables (LUTs), VERILOG, HDL, Adder, Multiplier.

I Introduction

Floating point numbers represents real numbers in binary format. The floating point arithmetic operations is generally used in business, financial and web based applications. The scientific applications are basically depends on the multimode computation. Multimode based computation are used to avoid underflow by removing multiplication by addition. Recent FPGA have a large number of look up tables (LUTs), registers, hardware multipliers and microprocessors [1]. Using these features the designs of multimode based arithmetic and floating point based circuits to be applicable to FPGAs. FPGA designers design a floating point arithmetic units on FPGA in 90's decade. Area is the main factor in all design. Even though scientific computations prefer floating point representation compared to fixed point representation, floating point arithmetic designs has increased complexity. Hence logarithmic number systems gains advantages over floating point systems [2]. So it is essential to seek out an option to feed binary numbers directly as input for these applications. By using this method the time is save and the method is easier, in current situation, this is unattainable, because within this adder/subtraction, input ought to lean in IEEE 754 format [3]. In floating point data format single precision consists of 32 bits and double precision consists of 64 bits. There are lots of efforts that are made over the past few decades to improve performance of floating point computation [4]. Floating point units are not only complex but also require more area and hence there are more power consumption as compared to fixed point multiplier and the complexity of the floating point unit increases as accuracy becomes the major issue. Even a small error in accuracy can cause large consequences. There are some scientific applications such as geometry computational, climate modelling require good computational requirements, for this it is required to have extreme precision in floating

DOI: 10.18535/ijsrm/v5i5.08

point calculations. But some applications do not require good precision. In that type of applications, such as even and approximate value will be sufficient for the correct operation [5]. It would be a luxury for applications which require lower precision to use double precision of quadruple precision floating point units. But it waste area and also increases latency. These all numerous modules are written in Verilog Hardware Description Language [6] and is simulated in Xilinx. After that they are synthesized in Xilinx integrated software environment (ISE) design suite.

This paper is presented as follows: section II discus the binary to floating point conversion, section III discuss the single precision floating point multiplier, section IV define the detection of underflow/overflow and section V discuss simulation and result.

II CONVERSION BINARY TO FLOATING POINT

Convert a decimal quantity into an associate degree IEEE 754 binary 32 format [7], the subsequent outline is

- Consider an associate degree number with a true range and a fraction like twelve half. 375
- Normalize and convert the number half into binary.
- The subsequent methodology shown below to convert the half fraction.
- For correct final conversion, add 2 results and modify.

Conversion of half fraction is shown below, we take a fraction zero.375. To convert this into binary fraction, multiply the fraction by two, and take the full number and remaining half is then re-multiply by two till a fraction of zero is found or till the preciseness limit is reached that is a twelve fraction digits for IEEE 754 binary 32 format.

0.375 * 2 = 0.750 = 0 + 0.750 = &gt; b-2 = 0

Then half number represents the binary fraction digit. Next step is to re-multiply by two and proceed.

0.750 * 2 = 1.500 = 1 + 0.500 =&gt; b-2 = 1

0.500 * 2 = 1.000 = 1 + 0.000 =&gt; b-2 = 1

Fraction = 0.000 i.e. terminated

We can say that the $(0.375)_{10}$ will be accurately converted into binary as $(0.011)_2$. Not all decimal fraction will be described in a very finite digit binary fraction as an example we take 0.1 cannot be describe in precisely binary form [8].

III FLOATING POINT SINGLE PRECISION MULTIPLIER

Floating point number is the way to represent the real number into binary form, the IEEE 754 is the standard way to represents two different floating point font like binary interchange format and decimal interchange format [9]. For DSP application multimode floating point representation is required because the DSP applications involves large dynamic range. Fig 1. Shows the single precision floating point format IEEE 754; which consists of one bit for sign (S), for exponent (E) eight bit required, and for mantissa (M) twenty three bit required. An extra bit is added to the fraction to form what is called the significant. If the value of exponent is greater than 0 and less than 255, and MSB is 1, then the number is said to be normalized number, and the real number is represented by 1.

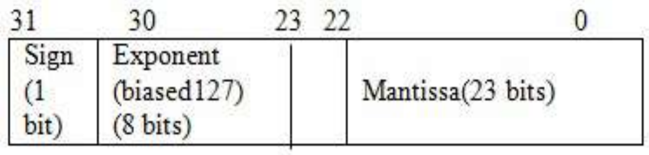


Figure 1.IEEE single precision floating point format.

When we multiply two floating point format, 1- added to the exponent of the number then subtracting the bias from their result, 2- multiplying the significant of the numbers, and 3- calculating the sign by XORing the sign of the two numbers [10].

DOI: 10.18535/ijsrm/v5i5.08

Multiplication algorithm of Floating Points

We discussed in introduction, the normalized floating point numbers are written in the form of [11]

$Z = (-1^{S}) * 2^{(E - Bias)} * (1.M)$ (1)

For multiplying two floating numbers, the following steps are done

- Multiply the significant; i.e. (1.M * 1.M)
- In the result placing the decimal point.
- Exponents are adding; i.e. $(E_1 + E_2 - Bias)$
- The sign bits are obtained; i.e. $s_1$xor s_2.
- The results are normalized i.e. obtaining 1 at the MSB of the result significant.
- Rounding the result to fit in the available bits.
- Underflow/ overflow occurrence are checked.

A floating point representation is similar to the IEEE 754 single precision floating point format, but the mantissa bits are reduced, while still retaining the hidden. The single precision floating number range is $\pm (2-2^{-23}) * 2^{127}$ in binary and $\pm 10^{38.53}$ in decimal format [12].

Normalized numbers for '1' bit is

A = 0 10000100 0100 = 40, B = 1 10000001 1110 = -7.5.

Now multiply A and B

1. Multiply significant : 1.0100
```
              *1.1110
       ____________

          00000
          10100
         10100
        10100
       10100
     ______________
       1001011000
```
2. Place the decimal point: 10.01011000
3. Exponent adding: 10000100
```
   + 10000001
   __________
   100000101
```

The two number exponent is already shifted/biased by the bias value (127) and is not the true exponent; i.e. $E_A = E_{A\text{-true}} + \text{bias}$ and $E_B = E_{B\text{-true}} + \text{bias}$

And

$E_A + E_B = E_{A\text{- true}} + E_{B\text{- true}} + 2\ \text{bias}$

After that the bias is subtracted from the resultant exponent, otherwise the twice bias is added.

```
     100000101
  -  01111111
  ______________

     100001101
```

4. Sign bit is obtained and put this value with result

   1 10000110 10.01011000

5. After that we will normalizes the result so that there is a 1 just before the radix point (decimal point). The radix point is moved by one place left to increment the exponent by 1; and moving one place by right to decrements the exponent by 1.

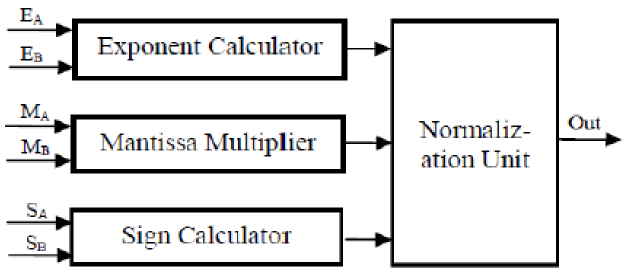
1 10000110 10.01011000 (before normalized)

1 10000111 1.001011000 (normalized)

This result is without the hidden bit result:

1 10000111 00101100

6. After that the mantissa are more than 4 bits (mantissa available bits); there are much needed rounding. If the rounding truncation



mode is applied then the stored value is:

1 10000111 0010

Figure 2. Floating point multiplier block diagram [4].

The sign-magnitude format explain the multiplication operation in floating point multiplication because it was similar to an integer format [13]

IV DETECTION OF OVERFLOW/ UNDERFLOW

From the result exponent we decide overflow/underflow. If the exponent is large or small decided by the exponent field. The exponent size is must be 11 bit. And the value must be in between 1 and 2048 otherwise the value is not normalized one. When two exponent is added then overflow occurs during normalization. These overflow is compensated by subtraction the bias; after that the resulting is normal output value. Underflow occurs when subtracting bias from the intermediate exponent. The underflow is compensated only and only if the intermediate exponent is > 0. If intermediate exponent is < 0 then it can't be compensated and if the intermediate exponent is =0 then it is underflow and it may be compensated by adding 1 during normalization. When if the overflow occurs and the overflow flag signal goes to high and the result may be ± infinity. Similarly when underflow occurs, then the underflow flag signal goes high and the underflow result may be ± zero. The renormalized numbers are signed to zero with the appropriate sign calculation. Assume the E1 and E2 are the exponent of two numbers A and B respectively. The overall exponent is calculated by this below formula.

Eresult =E1 + E2 – 1023

V SIMULATION RESULTS

In this paper all the simulation are done using Xilinx simulator and are shown below.

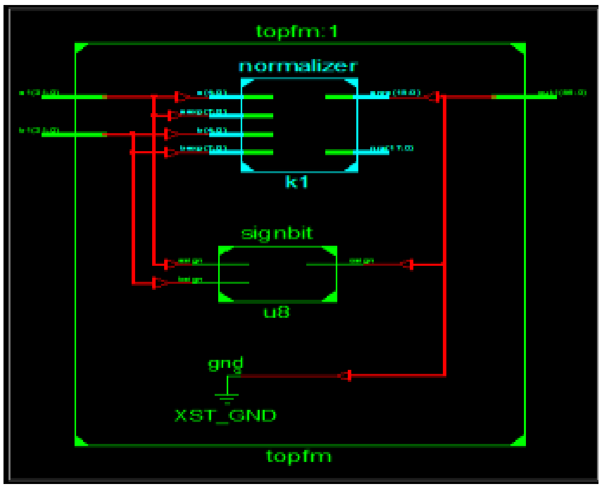


Figure 3. RTL schematic block-log LUT multiplication.

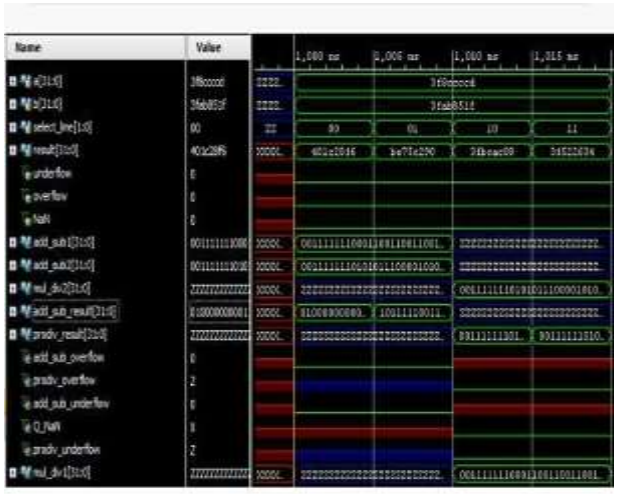


Figure 4. Simulation result for single precision Floating point multiplier.

VI CONCLUSION

This paper presented a design and analysis of Multimode Single Precision Floating Point Arithmetic Unit Using Verilog. To improve speed of arithmetic operation, we use Dadda multiplier replacing Carry save multiplier. This design the total area of hardware required will be reduced. The LUTs reduction is 14% and area reduction is 19%.

REFERENCES

1. Smaranika Rout, Dr. S.K.Mandal, "Implementation of Low Power and High Speed Single Precession Floating" in International Journal of VLSI System Design and Communication System, vol.04, Issue.09, pp.0688-0674, sep-2016.
2. N.Ramya Rani, V.Subbiah and L.Sivakumar, "Design of Logarithm Based Floating Point Multiplication and Division on FPGA" in ARPN journal of engineering and applied sciences, vol. 11, no. 2, January 2016.
3. Mohamed Al-Ashrfy, Ashraf Salem and WagdyAnis "An Efficient implementation of Floating Point Multiplier" IEEE Transaction on VLSI 978-1-4577-0069-9/11, 2011.
4. G.Sruthi, M.Rajendra Prasad, "An Efficient Implementation of Floating Point Multiplier

using Verilog", in international journal of innovation technologies, vol.03, issue.11, pp.2107-2112, dec-2015.

5. Elby C Varghese, Merlin Thomas, "Implementation of Single Precision Floating Point Processor Using Residue Number
6. System" in internal journal of advanced research in electrical Electronics and Instrumentation Engineering, vol.04, Issue 11, pp.227-231, nov-2015.
7. Whytney J. Townsend, Earl E. Swartz, "A Comparison of Dadda and Wallace multiplier delays". Computer Engineering Research Center, the University of Texas.
8. L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"96), pp. 107-116, 1996.
9. B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365-367, 1994.
10. B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365-367, 1994.
11. Zichu Qi; Qi Guo; Ge Zhang; Xiangku Li; Weiwu Hu, "Design of Low-Cost High-Performance Floating-Point Fused Multiply-Add with Reduced Power," VLSI Design, 2010. VLSID '10. 23rd International Conference on, vol., no., pp.206,211, 3-7 Jan. 2010.