Architecture and Design of Generic IEEE-754 Based Floating Point Adder, Subtractor and Multiplier

Sahdev D. Kanjariya
VLSI & Embedded Systems Design
Gujarat Technological University PG School
Ahmedabad, India
sahdev.kanjariya@gmail.com

Rutarth Patel
VLSI & Embedded Systems Design
Gujarat Technological University PG School
Ahmedabad, India
rutarth91@gmail.com

***Abstract*—** The Floating point numbers are being widely used in various fields because of their great dynamic range, high precision and easy operation rules. In this paper, architecture of generic floating point unit is proposed and discussed. This generic unit is compatible with all three IEEE-754 binary formats. Further based on this architecture, floating point adder, subtractor and multiplier modules are designed and functionally verified for Virtex-4 FPGA. The design is working properly and giving accurate result up to the last point.

Keywords- *Floating point; IEEE-754; FPGA; Generic; Verilog HDL*

*****

I. INTRODUCTION

In terms of computing, Floating point is the method of representing an approximation to real number with the help of fixed number of bits. With the increase in digital devices, the need of having floating point capable hardware on the same device is also increasing. Applications working with DSP (Digital Signal Processing), 3D graphics, Image processing etc can get benefited by including a floating point unit along with the main processor on the device. Now in these days, Field Programmable Gate Arrays (FPGAs) are becoming one of the major platforms to implement and check any complex digital design because of their high integration density, comparatively low price, high performance, flexibility and many more others.

This paper mainly focuses on obtaining a basic architecture which can be used to design a FPU (Floating Point Unit), which is IEEE-754 standard compliant. A generic architecture for such FPU is discussed. This architecture can be used to design any of the three binary formats of floating point representations, which are: 32-bit (Single Precision), 64-bit (Double Precision) and 128-bit (Quadruple precision). Based on this design, generic code for the Adder, Subtractor and Multiplier is written in Verilog HDL (Hardware Description Language). The code is written such that just by changing parameters in the code, designs for any of the available format are obtained. Furthermore these sub modules are obtained and each module is functionally verified separately. This FPU is designed for Virtex-4 FPGA platform. Implementation part of the whole design is still remaining and will be done in future.

The rest of the paper is organized as follows: Section II describes all the literature surveyed for this paper. These include some research papers as well as standards and manuals. Section III gives understanding about the floating point numbers and operation on them. Section IV provides overview of the designed Generic units: Adder, Subtractor and Multiplier. Section V is for discussion on simulations and results of the all designed units. Furthermore acknowledgements and conclusion, followed by the references ends up the paper.

II. LITERATURE

The IEEE has standardized the binary and decimal floating point computer representation and rules about them in their standard IEEE-754. Current version of the standard is IEEE 754-2008[1]. For understanding floating point numbers properly, this standard is studied in detail.

Many researchers have worked with the fraction or real numbers. Many of them have also implemented their designs on different platforms. Some of them have used single precision format, some of them have used other options to represent real numbers like fixed point numbers. The key factor driving many of the researchers to use floating point numbers are because of their many advantages over fixed point.

In [2] P. Karlstrom, A. Ehliar and D. Liu designed floating point adder/subtractor and multiplier units. They have implemented their design on Virtex-4. However their design is capable of doing single precision operations only. Also their design is not fully IEEE-754 standard compatible. It does not handle infinities, Not-a-Numbers (NaNs) and denormalized numbers. Getao Liang, JunKyu Lee and Gregory D. Peterson proposed a dynamic precision supporting architecture for real number operations in [3]. Their proposed architecture is target for both, fixed point as well as floating point devices. Their architecture can work on different precision without need of changing hardware. They have implemented this architecture for some fixed point adders and multipliers only. Addanki Puma Ramesh, A. V. N. Tilak and A. M. Prasad designed and implemented double precision floating point multiplier in [4]. They have used DSP slices of Virtex-6 FPGA for the multiplication operation to make best use of the available resources. Their design handled overflow, underflow and round to zero (truncation) rounding mode. In [5], Manisha Sangwan and A Anita Angeline designed and compared different arithmetic modules and after that they implemented these modules together. They used single precision format for floating point numbers. They have not specified anything about rounding mode as well as different exceptions.

III. FLOATING POINT NUMBERS AND ARITHMETIC

A. *Floating point numbers*

In floating point format, exponent is used to scale up or down a number, represented by mantissa. Floating point format is shown below.

sign(s)	exponent(e)	mantissa(f)
MSB		LSB

Sign field represent sign of the number. Exponent field represent exponent by which the fraction part will be scaled up or scaled down. Mantissa field represent fraction part of the real number. Most of the time, this mantissa or fraction part is used along with the implicit hidden bit '1' before the mantissa. This bit is used along with the mantissa to provide precision of the format. These combinations also represent some special values[6][7].

IEEE has defined three binary formats for floating point numbers which are:

i. 32-bit
ii. 64-bit
iii. 128-bit

They have also defined four rounding modes:

i. Round to nearest (even)
ii. Round up
iii. Round down
iv. Round towards zero

Five types of exceptions:

i. Invalid
ii. Division by zero
iii. Underflow
iv. Overflow
v. Inexact

B. *Arithmetic*

This section gives overview about the arithmetic operation on the floating point numbers. William Stallings has discussed about floating point formats and arithmetic operations on these numbers very well in his book [8].

Addition and subtraction of the two floating point numbers are done as follows:

- In order to add/subtract two mantissas, the exponents of both the numbers should be made equal.
- This is done by calculating difference between two mantissas and then right shifting the mantissa of the number with smaller exponent by the same amount as of difference.
- Then both significands are added/subtracted accordingly to obtain result significand, the result exponent is the same as the bigger one from both exponents.
- If there is an overflow/underflow during addition/subtraction of the significands, then it should be handled by adjusting result exponent accordingly.

Multiplication of two floating point numbers requires different approach than the previous one.

Multiplication is comparatively easy task for floating point numbers than the addition and subtraction. The basic steps of the multiplications are:

- Add exponents of two operands to obtain resultant exponent
- Multiply mantissas of the two operands to obtain resultant significand
- Signs of these operands are XORed to obtain resultant sign

For the final result, still some approximations and round up need to be done. The resultant significand after multiplying two mantissas are of double length then the original operands. So some adjustments needs to be done on this intermediate result so that significand can be fitted into desired width. Also one should need to take of overflow or underflow and also make sure that normalization on resulting significand is done properly.

IV. ARCHITECTURE

An architecture for Floating point unit which can perform arithmetic operations like addition, subtraction or multiplication at a time is discussed in this paper. This architecture is shown below.

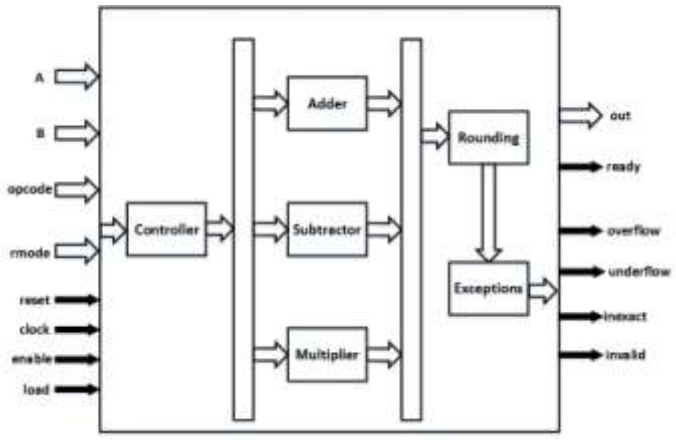

Figure 1. Basic Architecture of the FPU

This floating point unit shown here is designed to operate IEEE standard numbers. This architecture is having all rounding modes and can handle exceptions whichever occurs during these operations. Input operands and different controlling signals as well as result and some other signals as output are shown in architecture.

Descriptions of different blocks of the architecture are given in the following section.

A. *Controller*

- It will generate and propagate controlling signals for other modules
- Store incoming data (operands) into two registers
- Make sure that if operation is addition and either of the operand is negative, data will be routed towards subtractor module
- Likewise, if the operation is subtraction and either of the operand in negative, data will be routed towards adder module, also sign will be adjusted accordingly

The block diagram of the controller is shown in the figure 2.

 ISSN: 2321-8169
2690 - 2694

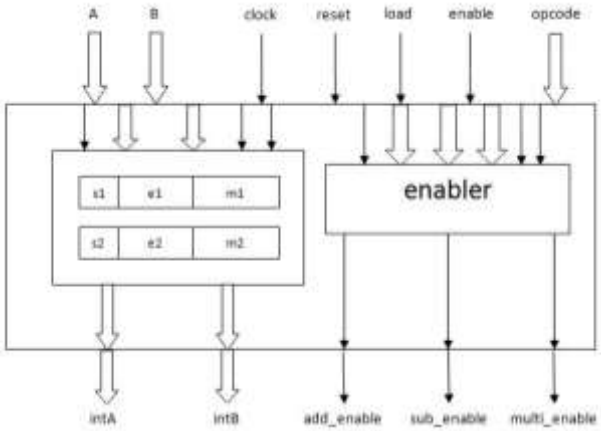

Figure 2. Block diagram of Controller

B. Adder

- Adds two operands without considering the signs of the numbers
- The input operands are first separated into their mantissa and exponent parts
- The larger operand goes into larger field, smaller operand into smaller field because of the controlling from controller module
- Then exponents of both operands are made equal by shifting the significand of the operand with the smallest exponent (intB) to the right, such that both exponents become equal
- Then significands are added to obtain result significand

The block diagram of the generic adder module is shown below.

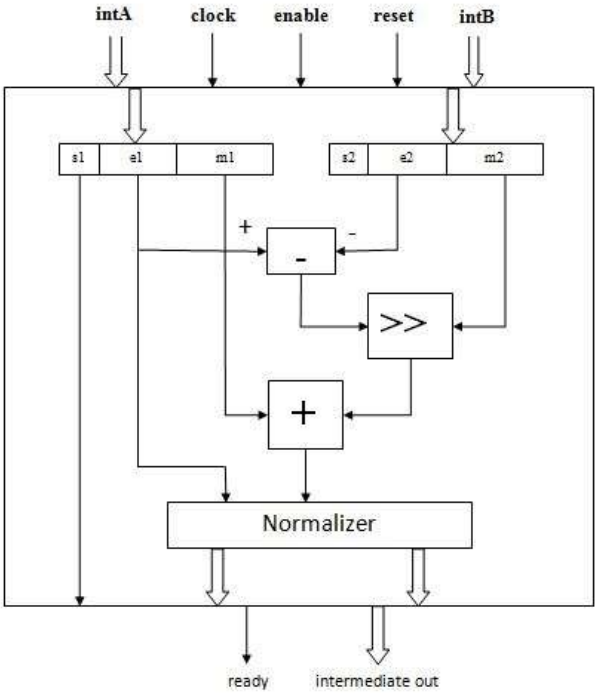

Figure 3. Block diagram of Adder

C. Subtractor

- Performs subtraction of the two input operands
- Input operands are first separated into their exponent and mantissa parts
- Difference between the two exponents is calculated
- Then according to this value, the mantissa of the smaller operand is shifted right by that value.
- After that normal subtraction between two mantissas is performed to obtain resulting mantissa

The working of the subtractor is almost similar to that of adder. The only difference is that we have to subtract shifted mantissa from the other one and also during normalizing, we have to take care of underflow, if there occurs any opposite to the overflow in case of addition.

The block diagram of the generic subtractor is shown in the figure below.

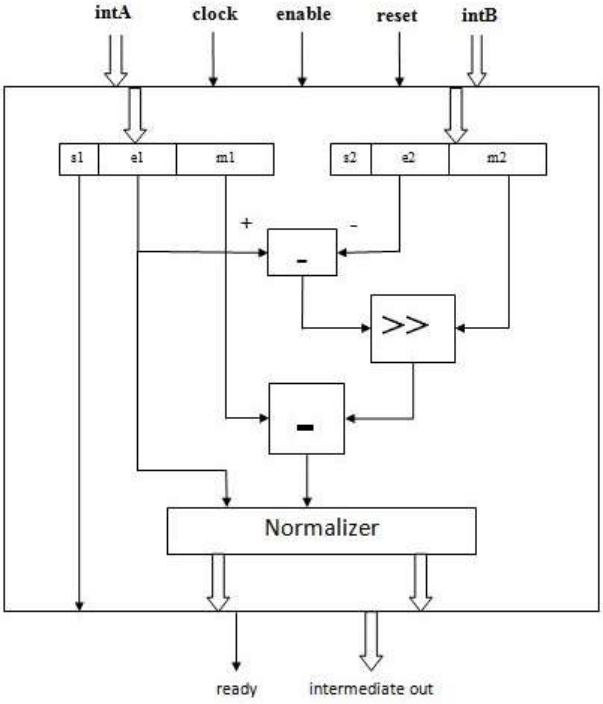

Figure 4. Block diagram of Subtractor

D. Multiplier

This unit multiplies two floating point numbers and provides result which is also a floating point number.

- At first, the input operands are separated into their mantissa and exponent parts.
- The mantissa of first operand and the leading '1' (for normalized numbers) are stored in to one register.
- The mantissa of the other operand and the leading '1' (for normalized numbers) are stored in the other register.
- The sign bit of the result is obtained by performing an XOR on the sign bit of both the operands.
- The exponent is obtained by adding the exponent of first operand with the exponent of the other operand and then by subtracting bias from this sum.
- Result mantissa is obtained by multiplying mantissas of both operands. Some extra bits are used along with these bits for further rounding and exception handlings.

 ISSN: 2321-8169
2690 - 2694

The block diagram of the Multiplier unit is shown in the figure below.

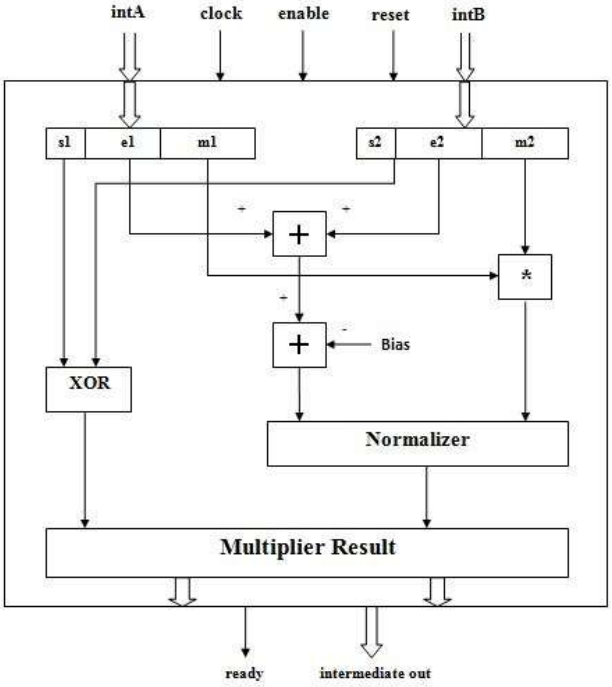

Figure 5. Block diagram of Multiplier

E. *Rounding*

This module is used to get approximate number from obtained intermediate results. The mode of rounding can be decided by the user by providing external output to the unit. This unit supports all the rounding modes described in the IEEE standard. Rounding is done to make sure that result after any arithmetic operation be fitted into the available bits for final output.

F. *Exceptions*

In the exceptions module, all of the special cases are checked for, and if they are found, the appropriate output is created, and the individual output signals of underflow, overflow, inexact and invalid will be asserted if the conditions for each case exist.

V. Implementation and Results

The above described modules are designed for Virtex-4 FPGA board and simulated to check if the results are functionally correct or not. Simulation results are shown in the figures below.

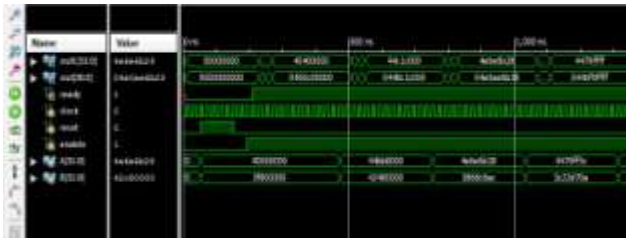

Figure 6. Simulated waveform of 32-bit Adder

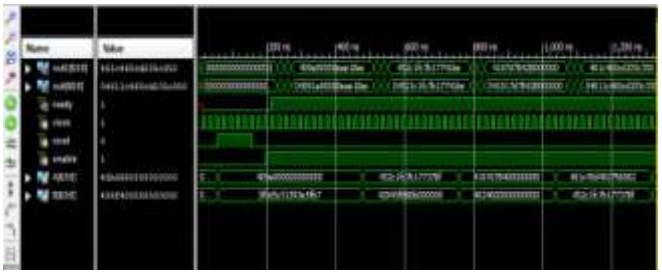

Figure 7. Simulated waveform of 64-bit Adder

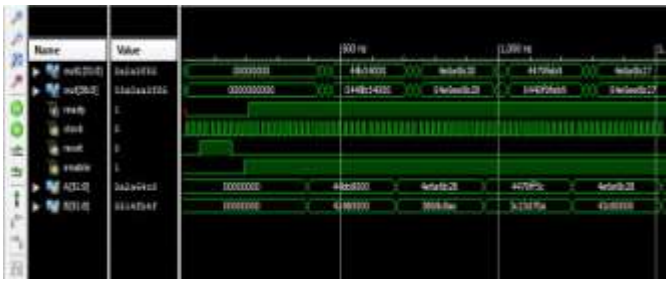

Figure 8. Simulated waveform of 32-bit Subtractor

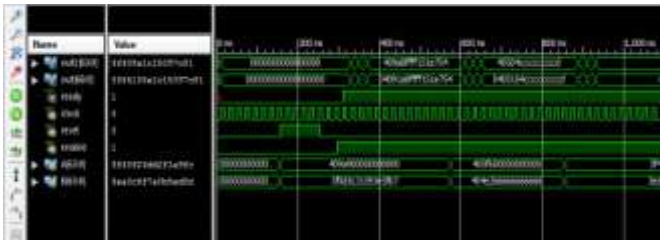

Figure 9. Simulated waveform of 64-bit Subtractor

Figure 10. Simulated waveform of 32-bit Multiplier

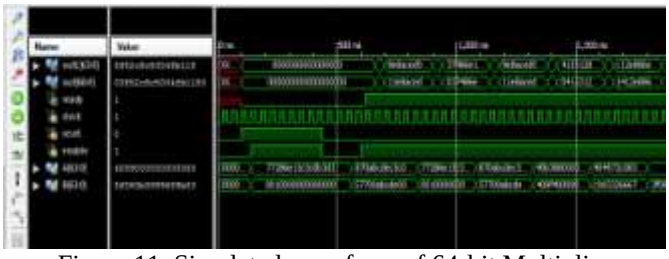

Figure 11. Simulated waveform of 64-bit Multiplier

VI. Conclusion

This paper discussed and designed architecture for floating point unit, which supports IEEE standard and the architecture itself can be used to implement any binary floating point format numbers. Furthermore some basic modules of this FPU is designed as well as functionally tested for their correctness, and they are giving very accurate results.

ACKNOWLEDGMENT

We would like to express our deepest appreciations to all those who helped us and gave us opportunity to complete this paper. Special thanks I give to my guide Mr. Rohit Khanna for helping me throughout my work. Their suggestions and encouragement helped me in writing this paper.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, 2008

[2] Karlstrom P. et al., "High-performance, low-latency field-programmable gate array-based floating-point adder and multiplier units in a Virtex 4", in Computers & Digital Techniques, IET, (Volume: 2, Issue: 4), July 2008, pp. 305-313

[3] Getao Liang et al., "ALU Architecture with Dynamic Precision Support", in Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on, Chicago IL, 2012, pp. 26-33

[4] Ramesh A. P. et al., "An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog", in Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), 2013 International Conference on, Tiruvannamalai, 2013, pp. 1-5

[5] Sangwan M. and Angeline A. A., "Design and implementation of single precision pipelined floating point co-processor", in Advanced Electronic Systems (ICAES), 2013 International Conference on, Pilani, 2013, pp. 79-82

[6] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture; 2014

[7] Numerical Computation Guide; Sun Microsystems, Inc.; Palo Alto, CA; 2001

[8] William Stallings; "Computer Arithmetic"; in *Computer Organization and Architecture Designing For Performance;* 8/E; Pearson Education, Upper Saddle River, New Jersey; 2009; pp. 334-344