INTERNATIONAL JOURNAL OF SCIENTIFIC PROGRESS AND RESEARCH (IJSPR) ISSN: 2349-4689
Volume-08, Number-03, 2015

A Review on IEEE-754 Floating Point Multiplier Using Verilog

Prakhi Agrawal[1], Prof. Shravan Sable[2], Dr. Rita Jain[3]
[1]M-Tech Research Scholar, [2]Research Guide, [3]HOD, Department of Electronics & Communication Engineering
Lakshmi Narain College of Technology, Bhopal,(M. P.)

Abstract- In this review paper we have presented a brief literature review for FPGA based Floating Point Multiplier. FPGAs provide good speedup outcomes while the retaining much of the flexibility of a software solution at a fraction of the startup cost of an ASIC. In the recent years, there has been a lot of work with the logarithmic number system as a possible alternative to floating point. Because of the complexity of floating point computations, floating point operations are often part of these serious portions and so these are good for implementation to obtain speedup of a system. Current technology provides two major options for implementations. These are the application specific integrated circuit and the field programmable gate array.

Keywords- ASIC, FPGA,IEEE-754. Double precision, Floating point, Multiplier.

I. INTRODUCTION

Floating-point units (FPU) are a math coprocessor which is designed specially to carry out operations on floating point numbers [1]. Typically FPUs can handle operations like addition, subtraction, multiplication and division. FPUs can also perform various transcendental functions such as exponential or trigonometric calculations, though these are done with software library routines in most modern processors. Our FPU is basically a single precision IEEE754 compliant integrated unit.

Floating Point Unit

When a CPU executes a program that is calling for a floating-point (FP) operation, there are three ways by which it can carry out the operation. Firstly, it may call a floating-point unit emulator, which is a floating-point library, using a series of simple fixed-point arithmetic operations which can run on the integer ALU. These emulators can save the added hardware cost of a FPU but are significantly slow. Secondly, it may use an add-on FPUs that are entirely separate from the CPU, and are typically sold as an optional add-ons which are purchased only when they are needed to speed up math-intensive operations. Else it may use integrated FPU present in the system [2].

The FPU designed by us is a single precision IEEE754 compliant integrated unit. It can handle not only basic floating point operations like addition, subtraction, multiplication and division but can also handle operations like shifting, square root determination and other transcendental functions like sine, cosine and tangential function.

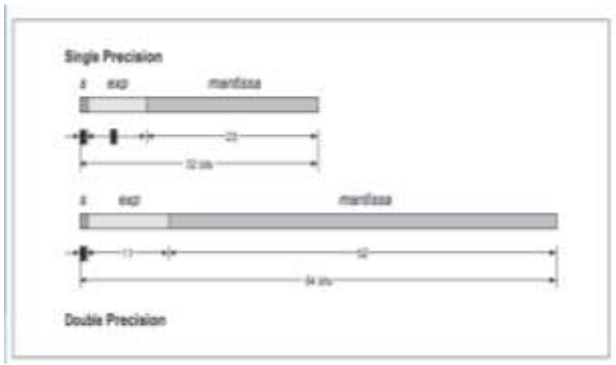

Fig. 1 Single & Double Precision Floating point representation

IEEE 754 Standards

IEEE754 standard is a technical standard established by IEEE and the most widely used standard for floating-point computation, followed by many hardware (CPU and FPU) and software implementations [3]. Single-precision floating-point format is a computer number format that occupies 32 bits in a computer memory and represents a wide dynamic range of values by using a floating point. In IEEE 754-2008, the 32-bit with base 2 format is officially referred to as single precision or binary32. It was called single in IEEE 754-1985. The IEEE 754 standard specifies a single precision number as having sign bit which is of 1 bit length, an exponent of width 8 bits and a significant precision of 24 bits out of which 23 bits are explicitly stored and 1 bit is implicit 1.

Sign bit determines the sign of the number where 0 denotes a positive number and 1 denotes a negative number. It is the sign of the mantissa as well. Exponent is an 8 bit signed integer from −128 to 127 (2's Complement) or can be an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 single precision definition. In this case an exponent with value 127 represents actual zero. The true mantissa includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the mantissa appear in the memory format but the total precision is 24 bits.

For example:

```
    S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
  31 30      23 22                    0
```

IEEE754 also defines certain formats which are a set of representation of numerical values and symbols. It may also include how the sets are encoded.

The standard defines [4]:

- Arithmetic formats which are sets of binary and decimal floating-point numbers, which consists of finite numbers including subnormal number and signed zero, a special value called "not a number" (NaN) and infinity.
- Interchange formats which are bit strings (encodings) that are used to exchange a floating-point data in a compact and efficient form.
- Rounding rules which are the properties that should be satisfied while doing arithmetic operations and conversions of any numbers on arithmetic formats.
- Exception handling which indicates any exceptional conditions similarly division by zero, underflow and overflow. occurred during the operations.

The standard defines five rounding rules:

- Round to the nearest even which rounds to the nearest value with an even (zero) least significant bit.
- Round to the nearest odd which rounds to the nearest value above (for positive numbers) or below (for negative numbers)
- Round towards positive infinity which is a rounding directly towards a positive infinity and it is also called rounding up or ceiling.
- Round towards negative infinity which is rounding directly towards a negative infinity and it is also called rounding down or floor or truncation.

The standard also defines five exceptions, and all of them return a default value. They all have a corresponding status flag which are raised when any exception occurs, except in certain cases of underflow.

The five possible stages are:

- Invalid operation are like square root of a negative number, returning of not a number" qNaN by default, etc., output of which does not exist.
- Division by zero is an operation on a finite operand which gives an exact infinite result 1/0 or log(0) that returns positive or negative infinity by default.
- Overflow occurs when an operation results a very large number that can't be represented correctly i.e. which returns ± infinity by default (for round-to-nearest mode).
- Underflow occurs when an operation results very small i.e. outside the normal range and inexact (de-normalized value) by default.
- Inexact occurs when any operation returns correctly rounded result by default.

II. MULTIPLICATION ALGORITHM

Multiplication of negative number using 2‟s complement is more complicated than multiplication of a positive number. This is because performing a straightforward unsigned multiplication of the 2's complement representations of the inputs does not give the correct result. Multiplication can be designed in such that it first converts all their negative inputs to positive quantities and use the sign bit of the original inputs to determine the sign bit of the result. But this increases the time required to perform a multiplication, hence decreasing the efficiency of the whole FPU. Here initially the Bit Pair Recoding algorithm which increases the efficiency of multiplication by pairing. To further increase the efficiency of the algorithm and decrease the time complexity.

Multiplication Using Bit Pair Recoding

This technique divides the maximum number of summands into two halves. It is directly derived from the Booth‟s algorithm [9]. It basically works on the principle of finding the cumulative effect of two bits of the multiplier at positions i and i+1 when performed at position i. This is further clarified in the following table.

III. LITERATURE REVIEW

Kodali, R.K., Gundabathula, S.K. and Boppana, L investigated the floating point arithmetic, specifically multiplication, is a widely used computational operation in many scientific and signal processing applications. In general, the IEEE-754 single-precision multiplier requires a 23 × 23 mantissa multiplication and the double-precision multiplier requires a large 52 × 52 mantissa multiplier to obtain the final result. This computation exists as a limit on both area and performance bounds of this operation. A lot of multiplication algorithms have been developed during the past decades. In this research, the two of the popular algorithms, namely, Booth and Karatsuba (Normal and Recursive) multipliers have been implemented, and a performance comparison is also made. The algorithms have been implemented on an uniform reconfigurable FPGA platform providing a comparison of FPGA resources utilized and execution speeds. The recursive Karatsuba is the best performing algorithm among the algorithms.

Hang Zhang, Wei Zhang, Lach, J. proposed a low-power accuracy-configurable floating point multiplier based on Mitchell's Algorithm. Post-layout SPICE simulations in a 45nm process show same-delay power reductions up to 26X for single precision and 49X for double precision compared to their IEEE-754 counterparts. Functional simulations on six CPU and GPU benchmarks show significantly better power reduction vs. quality degradation trade-offs than existing bit truncation schemes.

Ramesh, A.P., Tilak, A.V.N. and Prasad, A.M. researched on the Floating Point multiplication is widely used in large set of scientific and signal processing computation. Multiplication is one of the common arithmetic operations in these computations. A high speed floating point double precision multiplier is implemented on a Virtex-6 FPGA. In addition, the proposed design is compliant with IEEE-754 format and handles over flow, under flow, rounding and various exception conditions. The design achieved the operating frequency of 414.714 MHz with an area of 648 slices.

Sheikh, B.R., Manohar and R. presented the details of our energy-efficient asynchronous floating-point multiplier (FPM). Authors discussed design trade-offs of various multiplier implementations. A higher radix array multiplier design with operand-dependent carry-propagation adder and low handshake overhead pipeline design is presented, which yields significant energy savings while preserving the average throughput. Our FPM also includes a hardware implementation of denormal and underflow cases. When compared against a custom synchronous FPM design, our asynchronous FPM consumes 3X less energy per operation while operating at 2.3X higher throughput. To our knowledge, this is the first detailed design of a high-performance asynchronous IEEE-754 compliant double-precision floating-point multiplier.

Su Bo, Wang Zhiying and Huang Libo expanded the large computing platforms and the increasing of on chip transistors, power consumption becomes a significant problem. Many methods are proposed in different design levels to solve the problem. In this paper, researchers introduce asynchronous technique to an IEEE-754 double-precision floating-point multiplier aiming to reduce its power consumption. The control path of the asynchronous multiplier employs redundant four-phase latch controllers and asymmetric delay elements. Experimental results show the power consumption of the asynchronous multiplier is at least 16% lower than its synchronous equivalent when running the PARSEC benchmarks. After synthesized in UMC 180nm technology, the area overhead of the asynchronous multiplier is 0.31%.

IV. PROPOSED METHODOLOGY

Floating-point calculation is analyzed to be an esoteric subject in the field of Computer Science [5]. This is obviously surprising, because floating-point is omnipresent in computer systems. Floating-point (FP) data type is almost present in every language. From PCs to supercomputers, all have FP accelerators in them. Most compilers are called from time to time to compile the floating-point algorithms and virtually every OS have to respond to all FP exceptions during operations such as overflow. Also FP operations have a direct effect on designs as well as designers of computer systems. So it is very important to design an efficient FPU such that the computer system becomes efficient. Further, FPU can be improvised by using efficient algorithm for the basic as well as transcendental functions, which can be handled by any FPU, with reduced complexity of the logic used. This FPU further can be worked upon to improvise further complex operations-viz. exponent. It can be designed so that it can handle different data types like character, strings etc, can serve as a backbone for designing a fault tolerant IEEE754 compliant FPU on higher grounds and such that pipeline can be implemented.

For efficient FPU for different kind of operations, the objective of proposed work is as follows:

- To develop an efficient algorithms for FP operations like addition, subtraction, division, multiplication and few transcendental functions.
- To implement the proposed scheme using Verilog.
- To synthesize the above proposed algorithm.

V. CONCLUSION & FUTURE WORK

The algorithm for the FPU is comparable in some case other are efficient algorithms like in the same case of block CLA and CLA in terms of delay, memory used, and device utilization. Since the FPU using possible efficient algorithms with several changes incorporated at our ends as far as the scope permitted, all the unit functions are unique in certain aspects and given the right environment these functions will tend to show comparable efficiency and speed and if pipelined then higher throughput may be obtained. We briefly studied the meaning of FPU and the IEEE 754 standard, different rounding modes, arithmetic formats, exceptions and interchange formats. This analysis gives an overview about the motivation and the objective of proposed methodology. It has been also studied the efficient algorithm to enhance the operation of the FPU. FPU have less delay, less memory requirement, low code complexity, comparable clock cycle and but still a vast amount of work that can be put on this FPU to additional improvise the efficiency of the FPU. Further implement operations like Exponential functions and Logarithmic functions can be used.

REFERENCES

[1] Kodali, R.K.; Gundabathula, S.K.; Boppana, L., "FPGA implementation of IEEE-754 floating point Karatsuba multiplier," *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on* , vol., no., pp.300,304, 10-11 July 2014.

[2] Hang Zhang; Wei Zhang; Lach, J., "A low-power accuracy-configurable floating point multiplier," *Computer Design (ICCD), 2014 32nd IEEE International Conference on* , vol., no., pp.48,54, 19-22 Oct. 2014.

[3] Ramesh, A.P.; Tilak, A.V.N.; Prasad, A.M., "An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog," *Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), 2013 International Conference on* , vol., no., pp.1,5, 7-9 Jan. 2013.

[4] Sheikh, B.R.; Manohar, R., "An Asynchronous Floating-Point Multiplier," *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on* , vol., no., pp.89,96, 7-9 May 2012.

[5] Su Bo; Wang Zhiying; Huang Libo; Shi Wei; Wang Yourui, "Reducing Power Consumption of Floating-Point Multiplier via Asynchronous Technique," *Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on* , vol., no., pp.1360,1363, 17-19 Aug. 2012.

[6] Manolopoulos, K.; Reisis, D.; Chouliaras, V.A., "An efficient multiple precision floating-point multiplier," *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on* , vol., no., pp.153,156, 11-14 Dec. 2011.

[7] Jinwoo Suh, Dong-In Kang, and Stephen P. Crago, “Efficient Algorithms for Fixed-Point Arithmetic Operations In An Embedded PIM”, 2005, University of Southern California/Information Sciences Institute.

[8] David Narh Amanor, “Efficient Hardware Architectures for Modular Multiplication”, Communication and Media Engineering, February, 2005, University of Applied Sciences Offenburg, Germany/

[9] Andre Weimerskirch and Christof Paar, “Generalizations of the Karatsuba Algorithm for Efficient Implementations”, Communication Security Group, Department of Electrical Engineering & Information Sciences, Ruhr-UniversitÄat Bochum, Germany

[10] Yamin Li and Wanming Chu, “A New Non-Restoring Square Root Algorithm and Its VLSI Implementations”, International Conference on Computer Design (ICCD‟96), October 7–9, 1996, Austin, Texas, USA

[11] Claude-Pierre Jeannerod, Herv´e nochel, Christophe Monat, Member, IEEE, and Guillaume Revy, “Faster floating-point square root for integer processors”, Laboratoire LIP (CNRS, ENSL, INRIA, UCBL)

[12] Prof. Kris Gaj, Gaurav, Doshi, Hiren Shah, “Sine/Cosine using CORDIC Algorithm”

[13] Samuel Ginsberg, “Compact and Efficient Generation of Trigonometric Functions using a CORDIC algorithm”, Cape Town, South Africa

[14] J. Duprat and J. M. Muller, “The CORDIC Algorithm: New Results for fast VLSI Implementation”, IEEE Transactions on Computers, vol. C-42, pp. 168 178, 1993