

FPGA Based Quadruple Precision Floating Point Arithmetic for Scientific Computations

¹Mamidi Nagaraju, ²Geedimatla Shekar

¹Department of ECE, VLSI Lab, National Institute of Technology (NIT), Calicut, Kerala, India

²Asst.Professor, Department of ECE, Amrita Vishwa Vidyapeetham University Amritapuri, Kerala, India

Abstract

In this project we explore the capability and flexibility of FPGA solutions in a sense to accelerate scientific computing applications which require very high precision arithmetic, based on IEEE 754 standard 128-bit floating-point number representations. Field Programmable Gate Arrays (FPGA) is increasingly being used to design high end computationally intense microprocessors capable of handling floating point mathematical operations. Quadruple Precision Floating-Point Arithmetic is important in computational fluid dynamics and physical modelling, which require accurate numerical computations. However, modern computers perform binary arithmetic, which has flaws in representing and rounding the numbers. As the demand for quadruple precision floating point arithmetic is predicted to grow, the IEEE 754 Standard for Floating-Point Arithmetic includes specifications for quadruple precision floating point arithmetic. We implement quadruple precision floating point arithmetic unit for all the common operations, i.e. addition, subtraction, multiplication and division. While previous work has considered circuits for low precision floating-point formats, we consider the implementation of 128-bit quadruple precision circuits. The project will provide arithmetic operation, simulation result, hardware design, Input via PS/2 Keyboard interface and results displayed on LCD using Xilinx virtex5 (XC5VLX110TFF1136) FPGA device.

Keywords

FPGA, Floating-point, Quadruple precision, Arithmetic.

1. Introduction

Integer arithmetic is common throughout computation. Integers govern loop behavior, determine array size, measure pixel coordinates on the screen, determine the exact colors displayed on a computer display, and perform many other tasks. However, integers cannot easily represent fractional

amounts, and fractions are essential to many computations. Floating-point arithmetic lies at the heart of computer graphics cards, physics engines, simulations and many models of the natural world. Floating-point computations suffer from errors due to rounding and quantization. Fast computers let programmers write numerically intensive programs, but computed results can be far from the true results due to the accumulation of errors in arithmetic operations. Implementing floating-point arithmetic in hardware can solve two separate problems. First, it greatly speeds up floating-point arithmetic and calculations. Implementing a floating-point instruction will require at a generous estimate at least twenty integer instructions, many of them conditional operations, and even if the instructions are executed on an architecture which goes to great lengths to speed up execution, this will be slow. In contrast, even the simplest implementation of basic floating-point arithmetic in hardware will require perhaps ten clock cycles per instruction, a small fraction of the time a software implementation would require. Second, implementing the logic once in hardware allows the considerable cost of implementation to be amortized across all users, including users which may not be able to use another software floating-point implementation (say, because the relevant functions are not publicly available in shared libraries).

Quadruple precision arithmetic increases the accuracy and reliability of numerical computations by providing floating-point numbers that have more than twice the precision of double precision numbers. This is important in applications, such as computational fluid dynamics and physical modelling, which require accurate numerical computations. Most modern processors have hardware support for double precision (64-bit) or double-extended precision (typically 80-bit) floating-point multiplication, but not for quadruple precision (128-bit) floating-point arithmetic operations. It is also true, however, that double precision and double extended precision are not enough for many scientific applications including climate modelling, computational physics, and computational geometry. The use of quadruple precision arithmetic can greatly

improve the numerical stability and reproducibility of many of these applications. Due to these advantages quadruple precision arithmetic can provide in scientific computing applications, specifications for quadruple precision numbers are being added to the revised version of the IEEE 754 Standard for Floating-Point Arithmetic. Recently, the use of FPGA-based accelerators has become a promising approach for speeding up scientific applications. The computational capability of FPGAs is increasing rapidly. A top-level FPGA chip from Xilinx Virtex-5 series contains 51840 logic Slices, 10368 Kbits storage and 192 DSP processing blocks (25×18 MAC).

Initial plans for the floating-point unit envisioned were ambitious. Full support for quadruple precision-format IEEE 754 floating-point addition, subtraction, multiplication, and division, along with full support for exceptions. Floating-point unit functionality can be loosely divided into the following areas, the Adder module, Subtractor module the Multiplier module and Division module. Together these modules compose the internals of the FPU module, which encapsulates all behavior in one location and provides one central interface for calculation of floating point calculations. A keyboard is interfaced with Floating Point Unit to feed the 128 bit input operands making use of the ps2 interface available on the FPGA board. The 128 bit output is displayed on the 16*2 LCD present on the FPGA board. Field Programmable Gate Array (FPGA) is a silicon chip with unconnected logic blocks, these logic blocks can be defined and redefined by user at any time. FPGAs are increasingly being used for applications which require high numerical stability and accuracy. With less time to market and low cost, FPGAs are becoming a more attractive solution compared to Application Specific Integrated Circuits (ASIC). FPGAs are mostly used in low volume applications that cannot afford silicon fabrication or designs which require frequent changes or upgrades. In FPGAs, the bottleneck for designing efficient floating-point units has mostly been area. With advancement in FPGA architecture, however, there is a significant increase in FPGA densities. Devices with millions of gates and frequencies reaching up to 500 MHz are becoming more suitable for floating-point arithmetic reliant applications.

2. Floating Point Numerical Representation

The IEEE 754 standard specifies that a quadruple precision number consists of a 1-bit sign, a 15-bit biased exponent, and a 112-bit significant. The quadruple precision number format is shown in fig. 2.1.

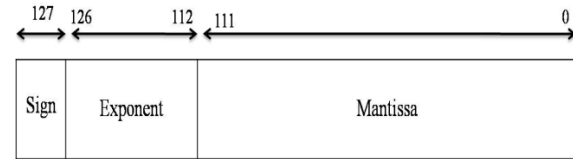


Fig. 2.1: Quadruple Precision Format

E is an unsigned biased number and the true exponent e is obtained as $e = E - E_{\text{bias}}$ with $E_{\text{bias}} = 16383$. For quadruple precision numbers value of E ranges from 0 to 32767. The number zero is represented with $E = 0$ and $f = 0$. An exponent $E = 2047$ and $f = 0$ represents infinity. The fraction f represents a number in the range $[0, 1)$ and the significant S is given by $S = 1.f$ and f is in the range $[1, 2)$. The actual value of the quadruple precision floating point number is the following:

$$\text{Value} = -1^{(\text{sign bit})} \times 2^{(\text{exponent} - 16383)} \times 1.(\text{mantissa}).$$

The basic format is described in IEEE 754 format, quadruple precision using 128-bits. Floating-point arithmetic as differs in a number of ways from standard integral arithmetic. Floating-point arithmetic is almost always inexact. Only floating point numbers which are the sum of a limited sequence of powers of two may be exactly represented using the format specified by IEEE 754. This contrasts with integer arithmetic, where (for example) the sum or product of two numbers always equals their exact value sum, excluding the rare case of overflow. For example, in IEEE arithmetic $0.1 + 0.2$ is not equal to 0.3 , but rather is equal to 0.30000000000000004 . This has many subtle ramifications, with the most common being that equality comparisons should almost never be exact. They should instead be bounded by some epsilon. Another difference between floating-point and integer numbers is that floating-point numbers include the special values positive and negative infinity, positive and negative zero, and not-a-number (NaN). These values are produced in certain circumstances by calculations with particular arguments. For example, infinity is the result of dividing a large positive number by a small positive number. A third difference is that floating-point operations can sometimes be made safe against certain conditions which may be considered errors. In particular, floating-point exceptions provide

a mechanism for detecting possibly-invalid operations such as the canonical division by zero, overflow to an infinity value, underflow to a number too small to be represented or inexact when the result of a calculation isn't exactly equal to the mathematical result of the calculation.

2.1. IEEE 754 Quadruple Precision Floating-Point Encoding

The mapping from an encoding of a quadruple-precision floating-point number to the number's value is summarized in Table 2.2.

2.1.1 Normalized numbers

A floating-point number is said to be normalized if the exponent field contains the real exponent plus the bias other than 0x7FFF and 0x0000. For all the normalized numbers, the first bit just left to the decimal point is considered to be 1 and not encoded in the floating-point representation and thus also called the implicit or the hidden bit. Therefore the quadruple precision representation only encodes the lower 112 bits.

2.1.2 Denormalized numbers

A floating-point number is considered to be denormalized if the exponent field is 0x0000 and the fraction field doesn't contain all 0's. The implicit or the hidden bit is always set to 0. Denormalized numbers fill in the gap between zero and the lowest normalized number.

2.1.3 Infinity

In quadruple precision representation, positive infinity is represented by sign bit is '0', exponent field of 0x7FFF and the whole fraction field of 0's. The negative infinity is represented by sign bit is '1', exponent field of 0x7FFF and the whole fraction field of 0's.

2.1.4 Not a Number (NaN)

In quadruple precision representation, NaN is represented by at least one of the mantissa bits must be nonzero. Otherwise, it would be interpreted as infinity. The sign bit can be 1 or 0, exponent field of 0x7FFF and the fraction field that doesn't include all 0's and the lower 111 bits of the mantissa can be any value, as long as it is nonzero for the NaN.

2.1.5 Zero

In double-precision representation, zero is represented by exponent field of 0x0000 and the whole fraction field of 0's. The sign bit represents -0 and +0, respectively.

3. Hardware Implementation

The block diagram of hardware implementation of Floating Point Unit is shown the fig. 3.1. The inputs to the Unit are clock, reset, enable, rounding mode, operation mode and two operands. Based on the inputs applied the output from the one of the chosen module. At the output 128 bit result is available. The block Floating Point Unit consists of separate modules for performing addition, subtraction, multiplication and division which is shown in the fig.3.2.

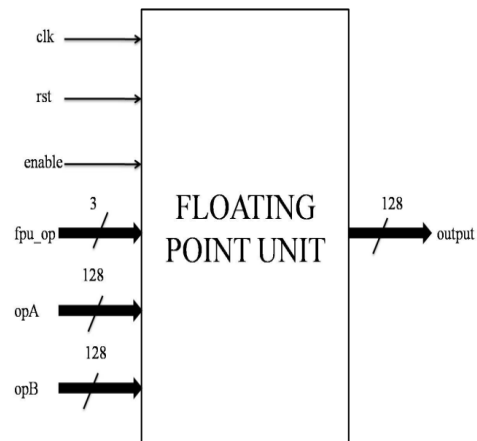


Fig. 3.1: Hardware Implementation

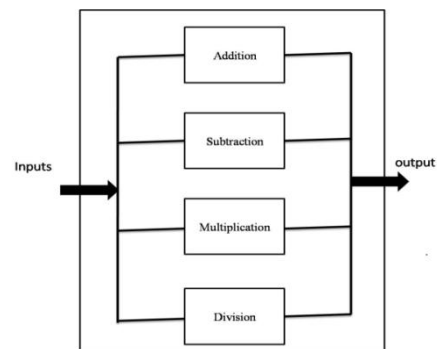


Fig. 3.2: Floating Point Unit.

3.1. Block Diagram of the Project

The input operands A and B are given from the keyboard via PS/2 interface available on FPGA board. Since operands A and B are of 128 bits it takes long time to give binary input of 256 bits serially. So, the hexadecimal input of operands A and B is given through the keyboard. By giving hexadecimal input we need to enter only 64 keys which reduce the time to enter the input. The time gap between the two

consecutive key presses is 3 seconds. The block diagram is shown in the fig. 3.3. The input from the keyboard which reaches the FPGA is respective scan code of the key pressed. A module is designed which converts scan code into hexadecimal format. A 128 bit Memory which consists of 32 locations of each 4 bits width is used to store the input operands. A reset switch is provided which can reset the memory. After entering either

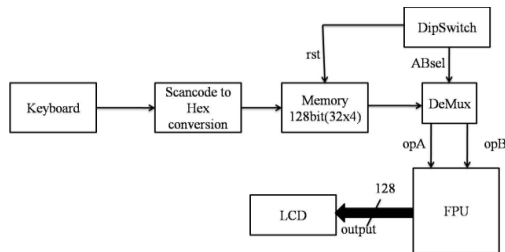


Fig. 3.3: Block Diagram of the Project

Operand A or operand B we can reset the memory so that the next input operand of 128 bit can be stored in the memory. Based on the select input from dip switch the 128 bit data can be passed to either operand A or operand B. The result which is obtained from the Floating Point Unit is displayed using LCD available on the FPGA. The result is also in the hexadecimal format. There are 32 locations available in the LCD so, the output of 128 bit is displayed in these 32 locations in hexadecimal format. The input operands are given through keyboard. The signals enable, ABsel, fpu_op, memoryrst are given through dip switches available on FPGA board. The signal rst is given through push button.

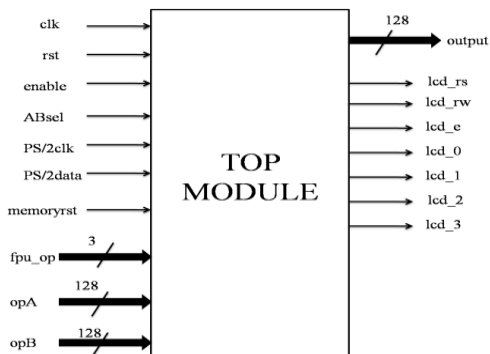


Fig. 3.4: The Top Module the Project

The output of 128 bit data is displayed on LCD in hexadecimal format. All the 32 locations present on LCD are used.

4. Results and Comparison

The code is written using Very High Speed Integrated Hardware Description Language (VHDL). VHDL code for floating point addition, subtraction, multiplication, division is written and simulation results are obtained using Xilinx 12.2 ISE tool. Clock, reset, enable, two input operands and output can be seen in the simulation plots. The hexa decimal representation is used in simulation plots. The clock period used is 10 ns which is same as Virtex5 (XC5VLX110T) FPGA.

4.1. Addition

The simulation result for the addition operation is shown in the fig. 5.1. The signal fpu_op is 000 for addition operation. The operands A and B are represented in hexadecimal format. The operand A is $(2^{265} * 1.5)$ and operand B is $(2^{281} * 1.5)$. The result of the addition operation is $(2^{281} * 1.500022888)$. The result shown in the simulation plot is in hexadecimal format.

The addition operation is taking 25.5 clock cycles to complete the operation and produce the result. The clock period is 10 nano seconds. The two markers shown in the plot indicates start of enable signal and result obtained.

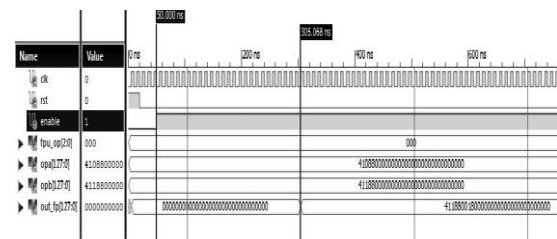


Fig. 4.1: Simulation plot for Addition

4.2. Subtraction

The simulation result for the subtraction operation is shown in the fig. 5.2. The signal fpu_op is 001 for subtraction operation. The operands A and B are represented in hexadecimal format. The operand A is $(2^{265} * 1.5)$ and operand B is $(2^{281} * 1.5)$. The result of the subtraction operation is $(- 2^{281} * 1.499977111)$. The result shown in the simulation plot is in hexadecimal format. The result is negative so, the sign bit is 1 in this case.

The subtraction operation is taking 26.5 clock cycles to complete the operation and produce the result. The clock period is 10 nano seconds. The two markers

shown in the plot indicates start of enable signal and result obtained.

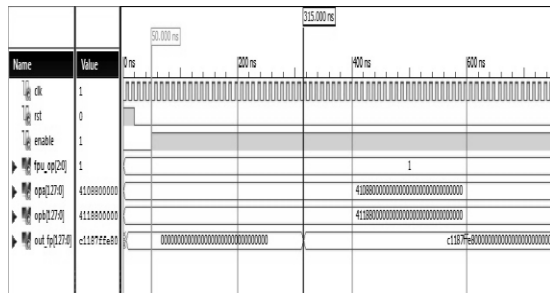


Fig. 4.2: Simulation plot for Subtraction

4.3. Multiplication

The simulation result for the multiplication operation is shown in fig. 5.3. The signal fpu_op is 010 for multiplication operation. The operands A and B are represented in hexadecimal format. The operand A is ($2^{265} * 1.5$) and operand B is ($2^{15} * 1.75$). The result of the multiplication operation is ($2^{251} * 1.3125$). The result shown in the simulation plot is in hexadecimal format. The multiplication operation is taking 29.5 clock cycles to complete the operation and produce the result. The clock period is 10 nano seconds. The two markers shown in the plot indicates start of enable signal and result obtained.

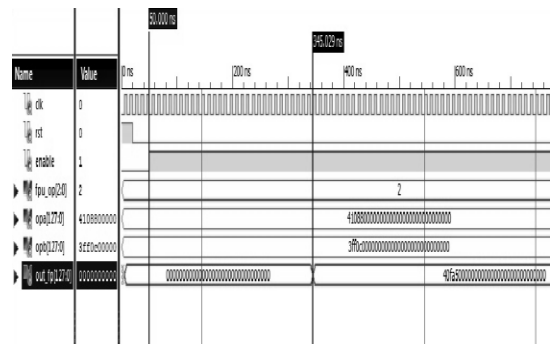


Fig. 4.3: Simulation plot for Multiplication

4.4. Division

The simulation result for the division operation is shown in the fig. 5.4. The signal fpu_op is 011 for division operation. The operands A and B are represented in hexadecimal format. The operand A is ($2^{265} * 1.5$) and operand B is ($2^{15} * 1.75$). The result of the division operation is ($2^{279} * 1.714285671$). The result shown in the simulation plot is in hexadecimal format.

The division operation is taking 128 clock cycles to complete the operation and produce the result. The clock period is 10 nano seconds. The two markers shown in the plot indicates start of enable signal and result obtained.

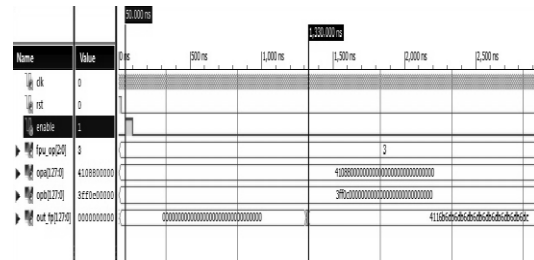


Fig. 4.4: Simulation plot for Division

4.5. Device Utilization Summary using FPGA

The design summary of the top level module is shown in the Table 4.1.

Table 4.1: Design Summary of Quadruple Precision Floating Point Unit

Device utilization summary:			

Selected Device : 5v1x110tff1136-1			
Slice Logic Utilization:			
Number of Slice Registers:	8415	out of 69120	12%
Number of Slice LUTs:	31859	out of 69120	46%
Number used as Logic:	31858	out of 69120	46%
Number used as Memory:	1	out of 17920	0%
Number used as SRL:	1		
Slice Logic Distribution:			
Number of LUT Flip Flop pairs used:	34911		
Number with an unused Flip Flop:	26496	out of 34911	75%
Number with an unused LUT:	3052	out of 34911	8%
Number of fully used LUT-FF pairs:	5363	out of 34911	15%
Number of unique control sets:	226		
IO Utilization:			
Number of IOs:	531		
Number of bonded IOBs:	147	out of 640	22%
Specific Feature Utilization:			
Number of BUFG/BUFGCTRLs:	6	out of 32	18%
Number of DSP48Es:	44	out of 64	68%

4.6. Comparison with different FPGA device resources

This design implemented on Virtex5 FPGA. Using with advanced architecture and effective utilization

of the LUTs and DSP48 in the design helped in reducing area and complexity.

Table 4.2: Comparison of different FPGA resources

FPGA	Slice	LUT	Frequency
Virtex 4	29274	57045	10.154MHz
Virtex 5	8415	31859	9.873MHz

Above table shown the comparison between Virtex 4 which contain two 4-input LUTs and Virtex5 in which four 6-input LUTs. It is usually the number of LUT's that is a bottleneck for FPGA design. From this perspective, it could be stated that one Virtex 5 slice can substitute 8 Virtex 4 slices. However in real-world designs it is impossible to utilize all the resources. But we succeeded by using optimum utilization of the FPGA resource with Keyboard, LCD and LED Interfacing.

5. Conclusion

A Quadruple Precision Floating Point Arithmetic Unit is implemented on FPGA (Field Programmable Gate Array) kit Xilinx Virtex5 (XC5VLX110TFF1136). Floating Point Arithmetic circuits can be extremely useful in the FPGA based implementation of complex systems that benefit from the re-programmability and parallelism of the FPGA device. The testing of the core is done by providing the facility to give input in the hexadecimal format through the keyboard. The output can be observed on the LCD in the hexadecimal format. The project can be extended to 256 bit floating point arithmetic unit. And the operations can be increased by including square root operation, etc.

References

- [1] Yong Dou, Yuanwu Lei, Guiming Wu, Song Guo, Jie Zhou, Li Shen, "FPGA Accelerating Double/Quad-Double High Precision Floating-Point Applications for ExaScale Computing", ICS'10, June 2-4, 2010, Tsukuba, Ibaraki, Japan.
- [2] Akkas, A., Schulte, M.J., "A Quadruple Precision and Dual Double Precision Floating-Point Multiplier", IEEE Proc. Digital System Design, pp.76-81, 2003.
- [3] Virtex-5 FPGA User GuideUG190 (v5.3) May 17, 2010, www.xilinx.com.
- [4] D. H. Bailey., "High-precision floating-point arithmetic in scientific computation", Computing in Science and Engineering, 7(3):54-61, January 2005.



Geedimatle Shekar was born in Kompally, Andhra Pradesh, on 9th November 1988. I completed my M.Tech in embedded design from National Institute of Technology (NIT)-Calicut. Presently working as an Assistant Professor at Amrita Vishwa Vidyapeetham University- Amritapuri

Campus. His field of interest includes Embedded and VLSI design.



Mamidi Nagaraju was born in Pale Annavaram, Andhra Pradesh, on 15th May 1985. Presently, working with SMDP-II Project at National Institute of Technology, Calicut. He completed his P.G Diploma in VLSI Design from DOEACC-Calicut. He was successfully published six papers in IEEE National

and International Conferences. His field of interest includes FPGA and ASIC Design.