Journal of Brilliant Engineering 2 (2021) 25-61

Brilliant Engineering

www.acapublishing.com

RESEARCH ARTICLE

Implementation and Design of 32 Bit Floating-Point ALU on a Hybrid FPGA-ARM Platform

Bahadır Özkılbaç, Tevhit Karacalı
Department of Electrical and Electronic Engineering, Ataturk University, Erzurum, Turkey

Abstract

FPGAs have capabilities such as low power consumption, multiple I/O pins, and parallel processing. Because of these capabilities, FPGAs are commonly used in numerous areas that require mathematical computing such as signal processing, artificial neural network design, image processing and filter applications. From the simplest to the most complex, all mathematical applications are based on multiplication, division, subtraction, addition. When calculating, it is often necessary to deal with numbers that are fractional, large or negative. In this study, the Arithmetic Logic Unit (ALU), which uses multiplication, division, addition, subtraction in the form of IEEE754 32-bit floating-point number used to represent fractional and large numbers is designed using FPGA part of the Xilinx Zynq-7000 integrated circuit. The programming language used is VHDL. Then, the ALU designed by the ARM processor part of the same integrated circuit was sent by the commands and controlled.

Keywords: FPGA, ALU, IEEE754 Floating-Point Numbers, ARM, VHDL.

1.Introduction

FPGAs are integrated circuits consisting of configurable logic blocks (CLB), digital signal processing blocks (DSP), memory blocks (RAM), I / O blocks and programmable interconnects that connect these blocks [1]. FPGAs are commonly used in numerous applications requiring high performance computing due to their capabilities such as reconfiguration, low power consumption and parallel operation.

For mathematical calculations in a digital system, positive and small integers cannot always be used. There is often a need to deal with numbers that are fractional, negative, or large. Fixed-point number format or floating-point number format is used to represent negative, fractional, large numbers in a binary number system. Floating-point numbers have higher precision and wide range of number than fixed-point numbers. Fixed-point numbers have advantages such as higher speed and less usage area than floating-point numbers. Number format to be used according to the expectations of any application is preferred. Today, floating-point number format standardized by IEEE is used throughout computer systems.

1.1. IEEE754 Floating-Point Number Format

The floating-point number format, which was standardized by IEEE in 1985, has two separate representations: single precision and double precision [2]. Single precision number format consists of 32 bits and the double precision number format consists of 64 bits. The single-precision number format is 32 bits, including 1-bit sign value, 8-bit exponent value, and 23-bit fraction value. The representation of the single precision floating point number format used in this paper is given in Figure 1.

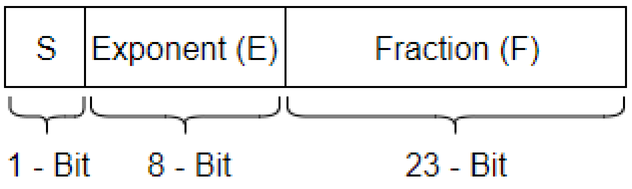


Figure 1. 32 Bit Floating-Point Representation

$$(-1)^S \, x \, 1.F \, x \, 2^{E-127} \quad (1)$$

The sign, exponent, and fraction values are then placed in a 32 bits binary number respectively.

For many algorithms that require millions of calculations per second, floating-point numbers are used because of their wide and dynamic range. In applications with floating-point numbers, bit lengths that are overused can lead to more usage and lower speed. A study involving addition, multiplication, division, and subtraction of 16 bit and 18-bit lengths similar to the IEEE754 floating-point number format standard, but different from this standard in bit length [3]. According to the application to be performed, sometimes there may be a need to deal with fixed-point or floating-point numbers. A library for hybrid applications that includes both floating-point and fixed-point number calculations was created using a fixed to floating format and floating to fixed format converter [4]. In floating point numbers, special designs with adjustable parameters have been made considering that the bit length will be flexible depending on the parameter entered by the user or it will provide flexibility in terms of usage [5]. Numerous applications have been made with floating point number units designed on FPGA. There is a study that have multiplied matrix using 64-bit floating-point number format [6]. Artificial neural network design with floating point numbers and training of this network are among these applications [7-10]. A customized single instruction multiple data (SIMD) is designed using a 16-bit floating-point number [11]. 2D Gaussian Filter, which is one of the applications where fixed point number format is used, is seen as inefficient in terms of usage area floating-point format [12].

In this paper, Arithmetic Logic Unit is designed which includes multiplication, division, addition, subtraction using IEEE754 single precision floating point number format. While designing, multiplication, division, addition operations of two binary integers of the same length were needed and multiplication, division, addition modules designed for this study, not VHDL operators which contain ready-made libraries were used. Then the ALU was controlled by the commands sent by the ARM.

2. Arithmetic Logic Unit (ALU)

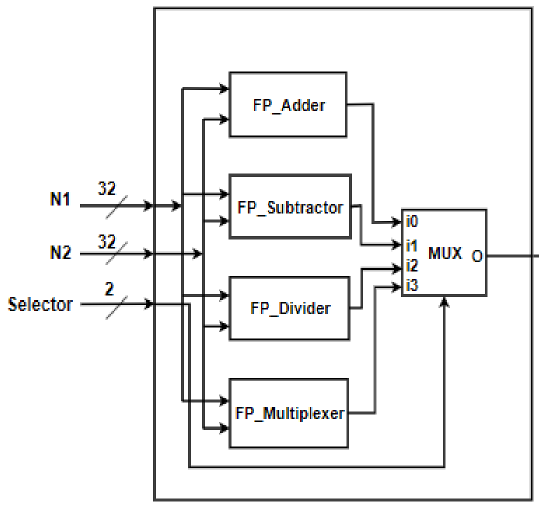


Figure 2. Arithmetic Logic Unit Architecture

ALU is unit that performs mathematical and logical operations of processor [13]. ALU designed consists of addition, subtraction, division, multiplication operation modules in IEEE754 32-bit floating-point number format and a 4 x 1 multiplexer to which the output of these modules is connected. 32-bit N1 and N2 numbers from input of ALU are transmitted to the four modules through data path. Figure 2. shows the architecture of ALU designed.

The multiplexer selects which operation result is output according to the value of the “selector” number. The value of "selector" number is assigned by control unit. In this study, the control unit is the ARM processor. Table 1. shows which operations are performed by ALU according to the value of “selector”.

Table 1. ALU Command Table

Selector	Operation
“00”	Addition
“01”	Subtraction
“10”	Division
“11”	Multiplicaiton

2.1. Floating-Point Adder and Subtractor

Addition is the most commonly used mathematical operation. Addition and subtraction have the highest latency and the most complexity among floating-point operations. It is not possible to directly add or subtract the two numbers shown as in (1). In order to add and subtract two numbers with such representations, their exponent values must be equal. Architecture of floating-point addition and subtraction module is given in Figure 3.

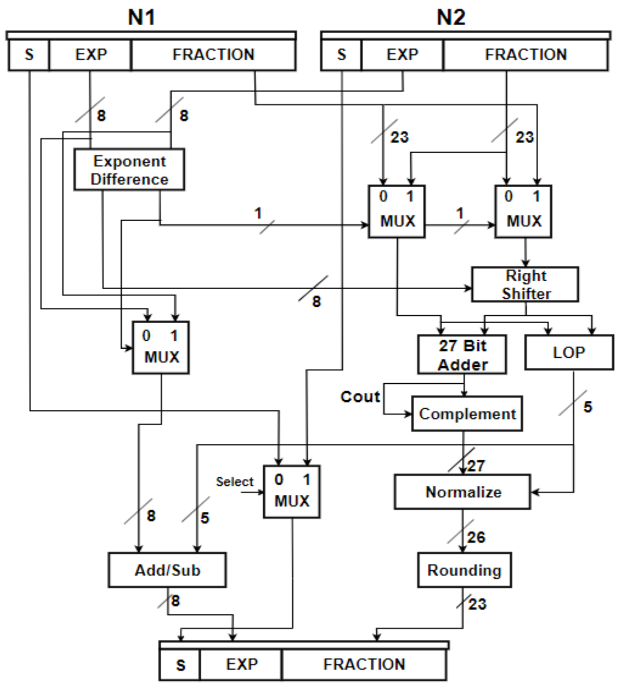


Figure 3. Floating-Point Addition Architecture

Fraction of the smaller number is shifted to the right. By adding shifting amount to exponent value of smaller number, exponent values are equalized for two numbers. In this case addition or subtraction can be performed. At the end of the process, the most significant ‘1’ for normalization is found, while the delay time is very important for speed performance. In this paper, LOP algorithm, which gives better results in terms of speed performance, is used than standard collection algorithms [14].

The steps for the LOP algorithm are as follows;

Step 1: Exponent value of N2 is subtracted from the exponent value of N1. There are two situations.

1. If the difference is positive or zero, N1 is considered greater.
2. If the difference is negative, the second N2 is considered greater.

Step 2: For pre-normalization, ‘1’ is added to the far left of the fractions. Thus, the new fraction lengths are 24 bits.

Step 3: Fraction of the smaller number and the five least significant bits of the difference result are given as input to the right shifter module. In this module, fraction of the small number is shifted to the right by the difference amount. With the shift operation, the number of bits (27) is increased by adding guard (G), round (R), sticky (S) bits to the far right. These three bits are used for rounding in the last step.

Step 4: The 27-bit fraction from the shift operation and fraction value of the larger number are input to the 27-bit adder. If signs of the numbers are same in the 27-bit addition module, the addition is performed. If the signs of the numbers are different, subtraction is performed in 27-bit addition module.

Step 5: The leading one predictor module (LOP) is used to estimate the most significant ‘1’ for normalization when processing in a 27-bit addition or subtraction.

Step 6: signs of the numbers are the same or if the exponent difference is greater than one, one bit left or right shift is performed, if not shifting left by the estimated amount from the LOP.

Step 7: To compute exponent value of the result, exponent value of large number is set and the sign bit of the result is calculated.

Step 8: The fraction value is rounded and the result is sent to the output.

2.2. Floating-Point Multiplexer

Floating-point multiplication is used in many mathematical applications such as trigonometry, derivative, integral, logarithm, square root, and filter. When more complex designs are made using these mathematical applications, even a small delay in floating-point multiplication is of great importance. As in (1), multiplication of two numbers is sum of the exponent values and multiplication of the fraction values. To perform these two operations, an 8-bit binary addition and a 24-bit binary multiplier are required [15]. The architecture of floating-point multiplication is given in Figure 4.

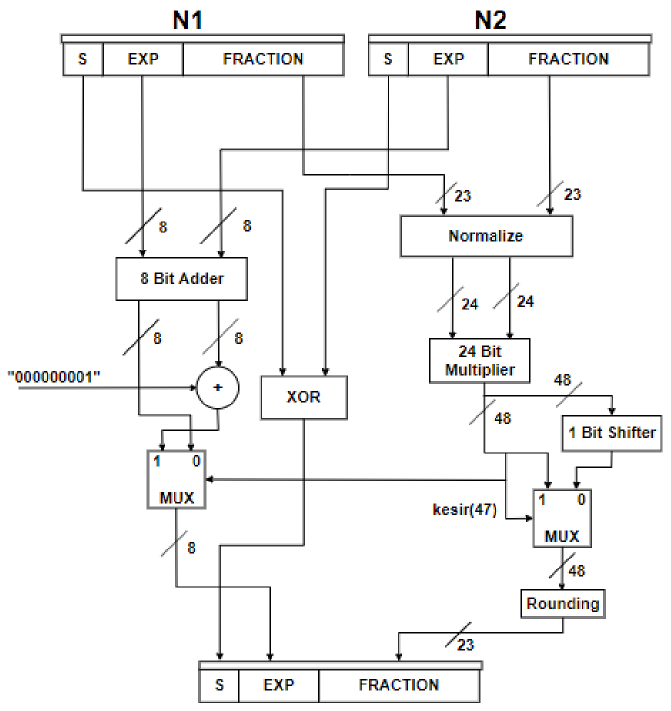


Figure 4. Floating-Point Multiplication Architecture

Floating-point multiplication is performed step by step as follows;

Step 1: It is checked whether N1 and N2 have a special case of being zero or infinity. If such a condition exists, the result is sent directly to the output.

Step 2: For pre-normalization, '1' is added to the far left of the fractions. Thus, the new fraction lengths are 24 bits.

Step 3: After the pre-normalization, the 24-bit numbers obtained are sent to the 24-bit multiplier. The 24-bit multiplier module give a 48-bit number output as a result of multiplication.

Step 4: While the 24-bit multiplication is continuing, exponent values of the numbers are added on the other hand.

Step 5: Since each bias value is attached bias value, it is necessary to subtract "01111111" from the sum of exponents.

Step 6: Sign bits of the numbers are passed through the xor gate to find the sign bits of the result.

Step 7: The most significant bit (MSB) value of the multiplication result is checked. If the MSB bit is '1', "00000001" is added to exponent value of the result and fraction value of the result is shifted 1 bit to the right, and 0 is continued without doing anything.

Step 8: Fraction value is rounded, sign, exponent and fraction values form the relevant places in 32 bits result and the result is sent to the output.

2.3. Floating-Point Divider

It has become critical to design more optimal operation units with the increase in applications made with floating-point numbers and the use of reconfigurable devices such as FPGAs during applications. One of these operation units is the floating-point number division. The basic algorithm of the floating-point number division operation is to take the difference of the exponent values and divide the fraction values. Division of two same length binary numbers is made with comparison algorithm using magnitude comparator [16]. When designing the floating-point division operation, the probability of divided and divisor numbers being zero or infinite was taken into consideration. Figure 5. shows the architecture of division operation.

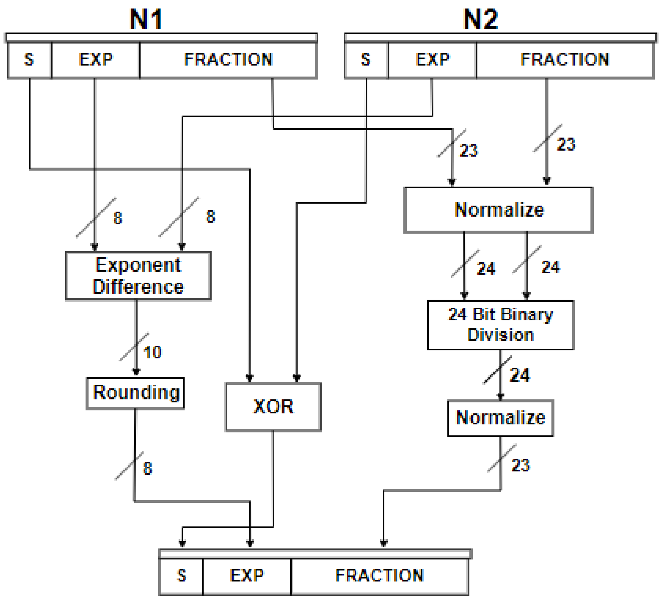


Figure 5. Floating-Point Division Architecture

The operations of the floating-point division algorithm are as follows;

Step 1: It is checked whether N1 and N2 have a special case of being zero or infinity. If such a condition exists, the result is sent directly to the output.

Step 2: Two numbers in 32-bit floating-point format are input into the designed module.

Step 3: The fraction values of numbers are normalized. Thus, the new fraction lengths are 24 bits.

Step 4: 24-bit fraction values obtained as a result of normalization are divided.

Step 5: The 24-bit number obtained from division is rounded and the fraction value of the result is generated.

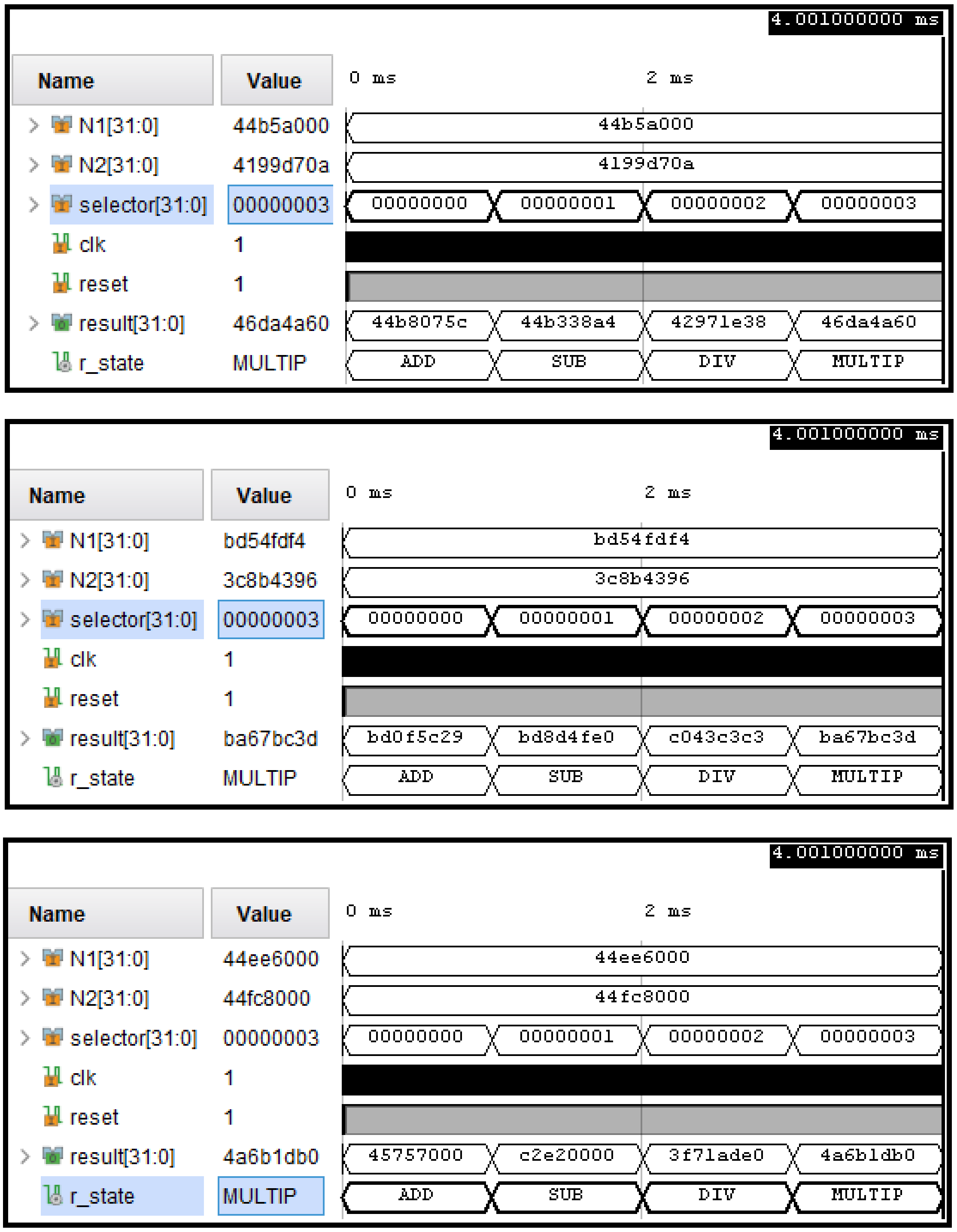
Step 6: To calculate sign bit of the result, the sign bits of the two numbers taken as input are passed through xor gate.

Step 7: The exponent values of N1 and N2 are subtracted, 127 (bias value) is added to the difference. Exponent, fraction, and sign values are placed in the required 32-bit result, and the result is sent to the output.

3. Results

3.1. Simulation Results

The simulation of ALU, which was designed using the Vivado 2019.1 by Xilinx, was successfully realized [17]. Two randomly selected number values are given as input to ALU and the value of selector number is changed. These two 32-bit floating-point numbers is given as input to the ALU, and the value of "selector" number is "00", "01", "10", "11" respectively. Thus, the results of each process are given to the output. The hexadecimal and decimal equivalents of numbers used in simulation in three different condition are shown in Table 2. Figure 6 shows the waveforms of all numbers.



```
                                                              4.001000000 ms
Name              Value        0 ms                    2 ms
N1[31:0]          44b5a000     44b5a000
N2[31:0]          4199d70a     4199d70a
selector[31:0]    00000003     00000000   00000001   00000002   00000003
clk               1
reset             1
result[31:0]      46da4a60     44b8075c   44b338a4   42971e38   46da4a60
r_state           MULTIP       ADD        SUB        DIV        MULTIP
```

```
                                                              4.001000000 ms
Name              Value        0 ms                    2 ms
N1[31:0]          bd54fdf4     bd54fdf4
N2[31:0]          3c8b4396     3c8b4396
selector[31:0]    00000003     00000000   00000001   00000002   00000003
clk               1
reset             1
result[31:0]      ba67bc3d     bd0f5c29   bd8d4fe0   c043c3c3   ba67bc3d
r_state           MULTIP       ADD        SUB        DIV        MULTIP
```

```
                                                              4.001000000 ms
Name              Value        0 ms                    2 ms
N1[31:0]          44ee6000     44ee6000
N2[31:0]          44fc8000     44fc8000
selector[31:0]    00000003     00000000   00000001   00000002   00000003
clk               1
reset             1
result[31:0]      4a6b1db0     45757000   c2e20000   3f71ade0   4a6b1db0
r_state           MULTIP       ADD        SUB        DIV        MULTIP
```

Figure 6. Simulation Results

Table 2. Operation Results

Number	Hexadecimal	Decimal
N1	44B5A000	1453
N2	4199D70A	19.23
Multiplication Result	46DA4A60	27941.188
Division Result	42971E38	75.55902
Adder Result	44B8075C	1472.23
Subtractor Result	44B338A4	1433.77

Number	Hexadecimal	Decimal
N1	BD54FDF4	-0.052
N2	3C8B4396	0.017
Multiplication Result	BA67BC3D	-0.000884
Division Result	C043C3C3	-3.0588
Adder Result	BD0F5C29	-0.035
Subtractor Result	BD8D4FE0	-0.069

Number	Hexadecimal	Decimal
N1	44EE6000	1907
N2	44FC8000	2020
Multiplication Result	4A6B1DB0	3852140
Division Result	3f71ADE0	0.944
Adder Result	45757000	3927
Subtractor Result	C2E20000	-113

3.2. Synthesizing Results

The design written in VHDL hardware description language is synthesized in Xilinx Vivado. The hardware model of ALU is obtained. The summary of usage area in FPGA of floating-point operation units obtained by synthesis is shown in Table 3.

Table 3. Summary Information Usage Area

Modül	LUT	FF	IO	BUFG
Adder/Sub	708	32	98	1
Multiplier	657	96	98	1
Divider	597	96	98	1
ALU	2207	312	130	1
FPGA	53200	106400	200	32

3.3. Implementation Results on Zedboard

Real time application was made on Xilinx Zedboard Development Kit. Zedboard have a Zynq-7000 integrated circuit and many peripherals such as UART, Audio Codec, Ethernet, HDMI, VGA, XDC Header, FMC Connector, Oled [18]. Zedboard is shown in Figure 7.

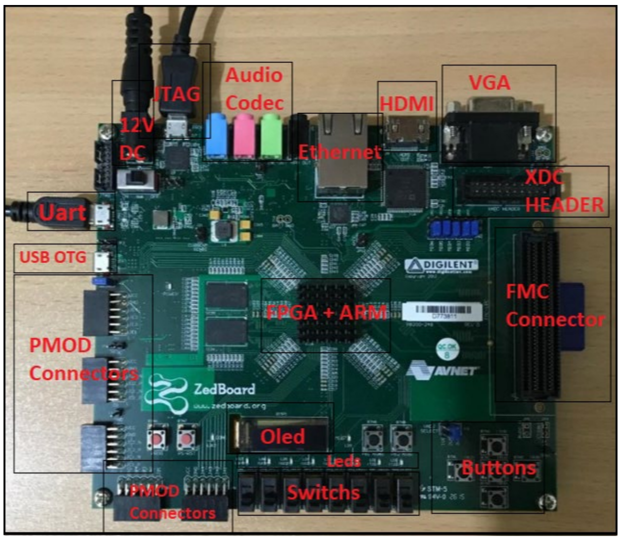


Figure 7. Zedboard Development Kit

The Zynq-7000 combines FPGA technology with a dual-core ARM Cortex-A9 [18]. With the dual core ARM Cortex-A9 processor, the FPGA exchanges data with each other over AXI interconnects. The N1, N2, selector numbers are sent to the ALU designed by the ARM in the FPGA through AXI interconnect. Result is sent back to ARM processor through AXI interconnect.

N1 and N2 numbers to be processed in the system are entered by the user as decimal and these numbers are sent to the system through UART. In the ARM processor, the decimal number is converted to the IEEE754 32-bit floating-point number format and transmitted to the FPGA. The numbers are processed by the FPGA in floating-point format and output to the ARM processor in the same format. The floating-point result received in the ARM is converted to a decimal number and sent to the PC through UART. The result is shown in the Tera Term VT serial port program. The connection between PC and Zedboard is shown in Figure 8. The images of the results shown in the Tera Term VT program are given in Figure 9.

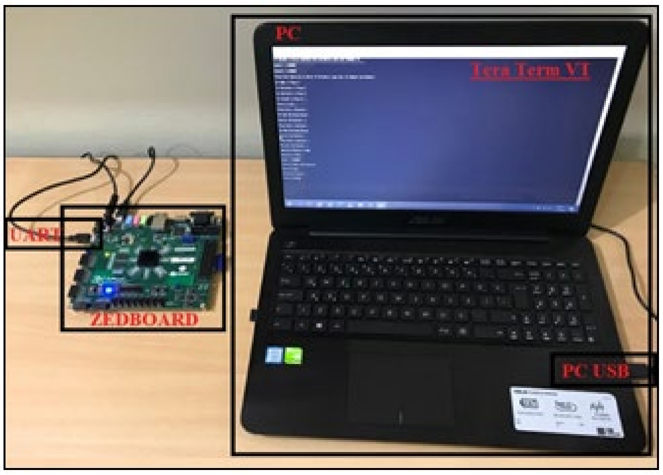
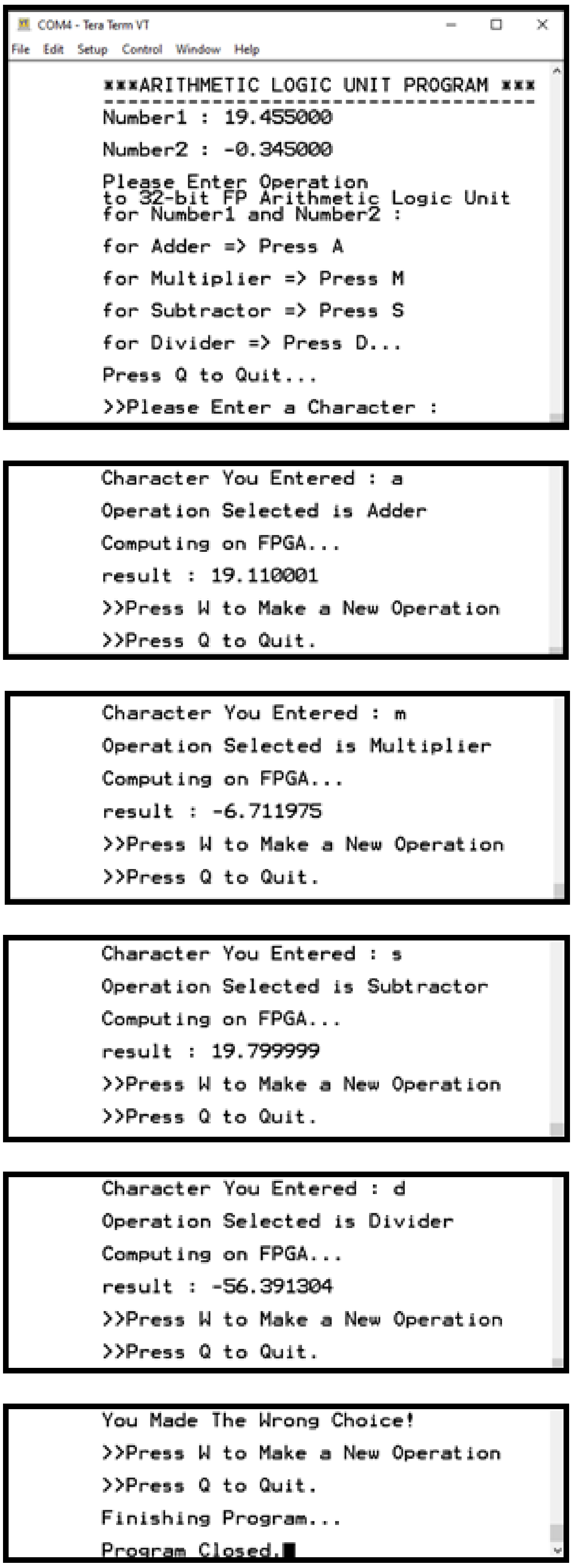


Figure 8. Connection PC-UART



```
***ARITHMETIC LOGIC UNIT PROGRAM ***
-----------------------------------
Number1 : 19.455000

Number2 : -0.345000

Please Enter Operation
to 32-bit FP Arithmetic Logic Unit
for Number1 and Number2 :

for Adder => Press A

for Multiplier => Press M

for Subtractor => Press S

for Divider => Press D...

Press Q to Quit...

>>Please Enter a Character :
```

```
Character You Entered : a

Operation Selected is Adder

Computing on FPGA...

result : 19.110001

>>Press W to Make a New Operation

>>Press Q to Quit.
```

```
Character You Entered : m

Operation Selected is Multiplier

Computing on FPGA...

result : -6.711975

>>Press W to Make a New Operation

>>Press Q to Quit.
```

```
Character You Entered : s

Operation Selected is Subtractor

Computing on FPGA...

result : 19.799999

>>Press W to Make a New Operation

>>Press Q to Quit.
```

```
Character You Entered : d

Operation Selected is Divider

Computing on FPGA...

result : -56.391304

>>Press W to Make a New Operation

>>Press Q to Quit.
```

```
You Made The Wrong Choice!

>>Press W to Make a New Operation

>>Press Q to Quit.

Finishing Program...

Program Closed.
```

Figure 9. Tera Term VT Results

4. Conclusions

In this study, an arithmetic logic unit in IEEE754 32 bit floating point number format is designed using VHDL programming language. Designed ALU has the capacity to be used in all kinds of applications. The dual-core ARM Cortex-A9 processor in the Zynq-7000 chip is programmed to control the ALU in FPGA with the C programming language. A system that communicates with the PC via UART is designed using Zedboard Development Kit. IEEE754 floating-point number format is used today in computer systems. Because floating point numbers are easy to process. Further, it is useful for high-precision calculations.

References

[1] SEALS, Richard C.; WHAPSHOTT, G. F. Programmable Logic: PLDs and FPGAs. Macmillan International Higher Education, 1997.

[2] IEEE Standards Board , “IEEE standard for binary floating-point arithmetic”, The Institute of Electrical and Electronics Engineers. 1985.

[3] SHIRAZI, Nabeel; WALTERS, Al; ATHANAS, Peter. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In: Proceedings IEEE Symposium on FPGAs for Custom Computing Machines. 1995; p. 155-162.

[4] BELANOVIĆ, Pavle; LEESER, Miriam. A library of parameterized floating-point modules and their use. In: International Conference on Field Programmable Logic and Applications. 2002; p. 657-666.

[5] GAFFAR, Altaf Abdul, et al. Automating customisation of floating-point designs. In: International Conference on Field Programmable Logic and Applications. 2002; p. 523-533.

[6] DOU, Yong, et al. 64-bit floating-point FPGA matrix multiplication. In: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. 2005; p. 86-95.

[7] ÇAVUŞLU, Mehmet Ali, et al. Neural network training based on FPGA with floating point number format and it’s performance. Neural Computing and Applications. 2011; 20.2: 195-202.

[8] SARTIN, Maicon A.; DA SILVA, A. C. ANN in hardware with floating point and activation function using hybrid methods. Journal of Computers. 2014; 9.10: 2258-2265.

[9] AYALA, Helon Vicente Hultmann, et al. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. Control Engineering Practice. 2017; 60: 124-132.

[10] FERREIRA, Pedro, et al. Artificial neural networks processor–a hardware implementation using a fpga. In: International Conference on Field Programmable Logic and Applications. Springer, Berlin, Heidelberg. 2004; p. 1084-1086.

[11] ETIEMBLE, Daniel; BOUAZIZ, Samir; LACASSAGNE, Lionel. Customizing 16-bit floating point instructions on a NIOS II processor for FPGA image and media processing. In: 3rd Workshop on Embedded Systems for Real-Time Multimedia. 2005; p. 61-66.

[12] CABELLO, Frank, et al. Implementation of a fixed-point 2D Gaussian Filter for Image Processing based on FPGA. In: 2015 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA). 2015; p. 28-33.

[13] ABD-EL-BARR, Mostafa, EL-REWINI, Hesham. Fundamentals of computer organization and architecture. John Wiley & Sons, 2005.

[14] BRUGUERA, Javier D., LANG, Tomas. Leading-one prediction with concurrent position correction. IEEE Transactions on Computers. 1999; 48.10: 1083-1097.

[15] MANO, M. Morris. Digital logic and computer design. Pearson Education India, 2017.

[16] SHAW, Robert F. Arithmetic operations in a binary computer. Review of scientific instruments, 1950; 21.8: 687-693.

[17] FEIST, Tom. Vivado design suite. White Paper. 2012; 5: 30.

[18] CROCKETT, Louise H.; ELLIOT, Ross A.; ENDERWITZ, Martin A. The zynq book tutorials for zybo and zedboard. Strathclyde Academic Media, 2015.