Indonesian Journal of Electrical Engineering and Computer Science
Vol. 19, No. 3, September 2020, pp. 1480~1489
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v19.i3.pp1480-1489

Design and implementation of fast floating point units for FPGAs

Mohammed Falih Hassan[1], Karime Farhood Hussein[2], Bahaa Al-Musawi[3]
[1,3]Department of Electronic and Communication Engineering, University of Kufa, Iraq
[2]Thi-Qar Oil Company, Iraq

Article Info

Article history:

Received Nov 29, 2019
Revised Feb 20, 2020
Accepted Mar 25, 2020

Keywords:

Embedded block
Floating-point unit (FPU)
FPGA architecture
IEEE floating-point

ABSTRACT

Due to growth in demand for high-performance applications that require high numerical stability and accuracy, the need for floating-point FPGA has been increased. In this work, an open-source and efficient floating-point unit is implemented on a standard Xilinx Sparton-6 FPGA platform. The proposed design is described in a hierarchal way starting from functional block descriptions toward modules level design. Our implementation used minimal resources available on the targeting FPGA board, tested on the Sparton-6 FPGA platform and verified on ModelSim. The open-source framework can be embedded or customized for low-cost FPGA devices that do not offer floating-point units.

Corresponding Author:

Mohammed Falih Hassan,
Department of Electronic and Communication Engineering,
University of Kufa, Najaf, Iraq.
Email: mohammedf.aljanabi@uokufa.edu.iq

1. INTRODUCTION

A field-programmable gate array (FPGA) is a flexible hardware computing tool that enables designers to implement difficult tasks at higher execution speed with the ability to frequently upgrade during runtime. Due to the high-speed capability of FPGAs, many existing systems and new technologies moved from software implementations to hardware-based FPGA. Current applications become more complex and require extensive Floating-Point (FP) calculations to satisfy the accuracy requirements. Digital Signal Processing (DSP) and image processing, wireless communications, and other industrial applications are examples of FPGA applications [1-9]. The requirement for speed is achieved using FPGA technologies and parallel distributed systems while accuracy is achieved by incorporating FP arithmetic in algorithms implementations [10-14].

Although embedded hardware FP blocks are available in commercial products such as [15-16], there is still a need for software-core implementations for two reasons. First, FP is a complex process and requires too many resources [17-19], as a result, it can customized software to match certain applications [20]. Second, low-cost platforms do not equip with embedded FP. Therefore, software-core provides the flexibility of optimized FP arithmetic to target different applications.

In this work, an efficient and open-source FP arithmetic for FPGA is suggested. The proposed structure used IEEE 754 FP standard, the most widely used standard for FP computation. The proposed design is described in the FPGA module level as well as at the software level. The system is implemented on the Xilinx Spartan-6 chip on the Nexys3 board, a powerful digital system design platform that is optimized for high-performance logic. We also verified the performance of our proposed module using ModelSim, the multi-language HDL simulator. The proposed structure can be used and customized in applications that require FP arithmetic.

The rest of the paper is organized as follows. Section 2 reviews the IEEE 754 FP standard. The Functional block diagram is described in Section 3. Then in Section 4 the modules level design for FPGA is proposed. Section 5 discusses the results and finally the conclusion is given in Section 6.

2. IEEE 754 FLOATING-POINT STANDARD

In this section, the IEEE 754 FP format is described [21-24]. The format consists of three parts; Sign (S), Exponent (E or exp) and Fraction (F). Figure 1 shows the three components (S , E , F) that represent the IEEE FP format. The sign part is 0 for positive numbers and 1 for negative. The exponent is either positive or negative depending on its values. The fraction represents the fraction part after FP numbers. The ranges of E and F are directly related to the precision of FP numbers.

Figure 1. IEEE 754 floating-point format

The value of FP number can be calculated from (S , E , F) as described in (1)

$$Decimal\ FP\ Number = (-1)^S \times val(1.F) \times 2^{val(E)}, \quad (1)$$

where the term $1.F$ called Significand (Sig). The implied "1" to the left of the binary point represents the normalized number while "0" represents the de-normalized number. The advantages of using FP numbers can be summarized as follows. Firstly, helping in unifying the developments of FP framework. Secondly, providing multi-level precision calculations that depend on a given application and finally making a seamless porting of FP numbers to other applications. However, the FP numbers are more expensive compared to a fixed point in hardware implementations.

According to the IEEE FP standard, there are two precision formats; single and double precision as shown in Figure 2. The single-precision uses 1-bit for sign, 8-bits for exponent and 23-bits for the fraction. While double precision format uses 1-bit for sign, 11-bits for exponent and 52-bit for the fraction.

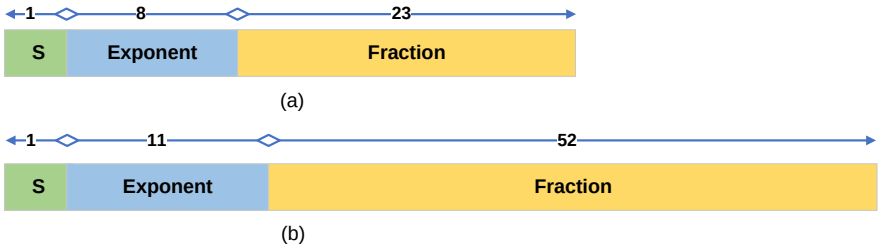

Figure 2. (a) IEEE single precision format, (b) IEEE double precision format

The exponent field value represents both positive and negative numbers using biased exponent as defined in (2)

$$val(\mathrm{E}) = E - bias, \quad (2)$$

where bias is a constant, which is 127 for single precision and 1023 for double precision. In addition, exponent values has the following properties (the numbers in the parentheses represent values for double precision)

$0 \le E \le 255(2047)$,

$E = 0$ *and* $E = 255(2047)$ *are reserved for special values* ,

$E = 1 \rightarrow 254\ (2046)$ *are used for normalized FP number*,
bias $= 127\ (1023)$ *then val*$(E) = E - 127\ (1023)$.
A decimal value for the normalized FP number for two precisions formats is defined as [24]

$$FP\ decimal\ value = (-1)^S \times (1.F)_2 \times 2^{E-bias}, \quad (3)$$

where bias=127 and 1023 for single and double precisions respectively. In addition to normalized FP numbers, there are de-normalized numbers, which is defined when the bit "1" defined in (3) changed to "0". de-normalized numbers have the advantage of providing gradual decreasing toward zero, i.e. fill the space between the smallest normalized FP number and zero as shown in Figure 3. These numbers are defined when (S, $E = 0$, $F \neq 0$) and their ranges described in (4). Table 1 summarized the ranges of FP numbers and their special values.

$$\begin{aligned} &Single\ precision\ : (-1)^S \times (0.F)_2 \times 2^{-126} \\ &Double\ precision : (-1)^S \times (0.F)_2 \times 2^{-1022} \end{aligned} \quad (4)$$

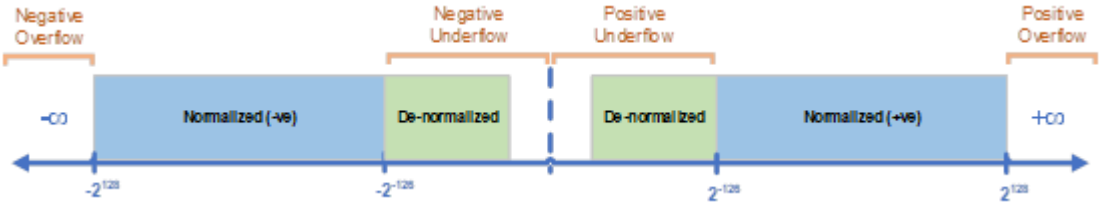

Figure 3. FP range for normalized and de-normalized numbers

Table 1. Summarized ranges of FP numbers and their special values

Single-Precision	Exponent = 8	Fraction = 23	Value
Normalized Number	1 to 254	Anything	$\pm (1.F)_2 \times 2^{E-127}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{E-126}$
Zero	0	0	±0
Infinity	255	0	±∞
Not a Number	255	nonzero	NaN
Double-Precision	Exponent= 11	Fraction = 52	Value
Normalized Number	1 to 2046	Anything	$\pm (1.F)_2 \times 2^{E-1023}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{E-1022}$
Zero	0	0	±0
Infinity	2047	0	±∞
Not a Number	2047	nonzero	NaN

3. FUNCTIONAL BLOCK DIAGRAM

Figure 4 shows the functional block diagram of the IEEE single-precision FP arithmetic implemented on Xilinx Spartan-6 FPGA chip based on Nexys-3 board while Figure 5 shows the mapping of input switches on the Neyxy3 board. The input signals to the FPGA board are:

a) SYSTEM_CLK: 100MHz system clock
b) Input switches ($Sw_0, Sw_1, \ldots\ldots Sw_7$): these switches are used to sequentially entering floating-point numbers to the system and display results on the seven-segment unit. The inputting process is implemented using a sequential state machine model.
c) UP-POINTER: this is a push-button input used to enter data sequentially in a forward direction starting from the Least Significant (LS) digit to the Most Significant (MS) digit.
d) DW-POINTER: a push-button input used to remove the previous entering data in a backward direction.
e) ADD/SUB: a push-button input used to add/subtract numbers.
f) CALCULATE: a pushbutton switch to start the addition/subtraction operation.
g) RESET: a push-button input used to reset the system to the initial state.

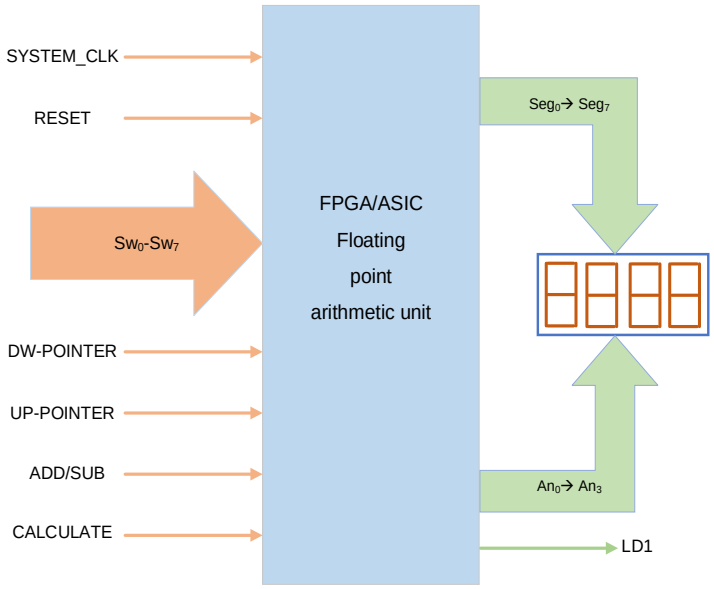

Figure 4. Functional block diagram for FP arithmetic unit

The output signals from the FPGA board are:

a) Seven segment output ($seg_0, seg_1, \ldots, seg_7$): These signals are used to send data to the seven-segment display.

b) ($An_0, An_1, ..., An_3$): Signals used to activate seven-segment digits.

c) LD_1 : an LED signal used to show the arithmetic operation type (i.e. addition or subtraction).

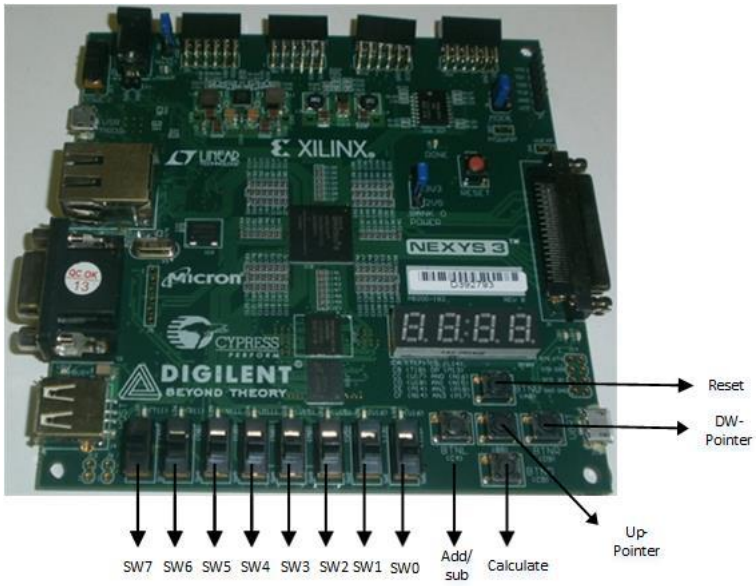

Figure 5. Input signals description of the Sparton-6 FPGA board

4. MODULE DESIGN

The details of our module-level design for theblock (Figure (4)) is shown in Figure 6. It is important to mention that we are designing a 5-bits word for each digit. This allows us to display more patterns on the seven-segment compare to 4-bits word. These patterns hel FPGA functional p us to identify special cases shown in Table 1. Below we describe the function of each module in detail.

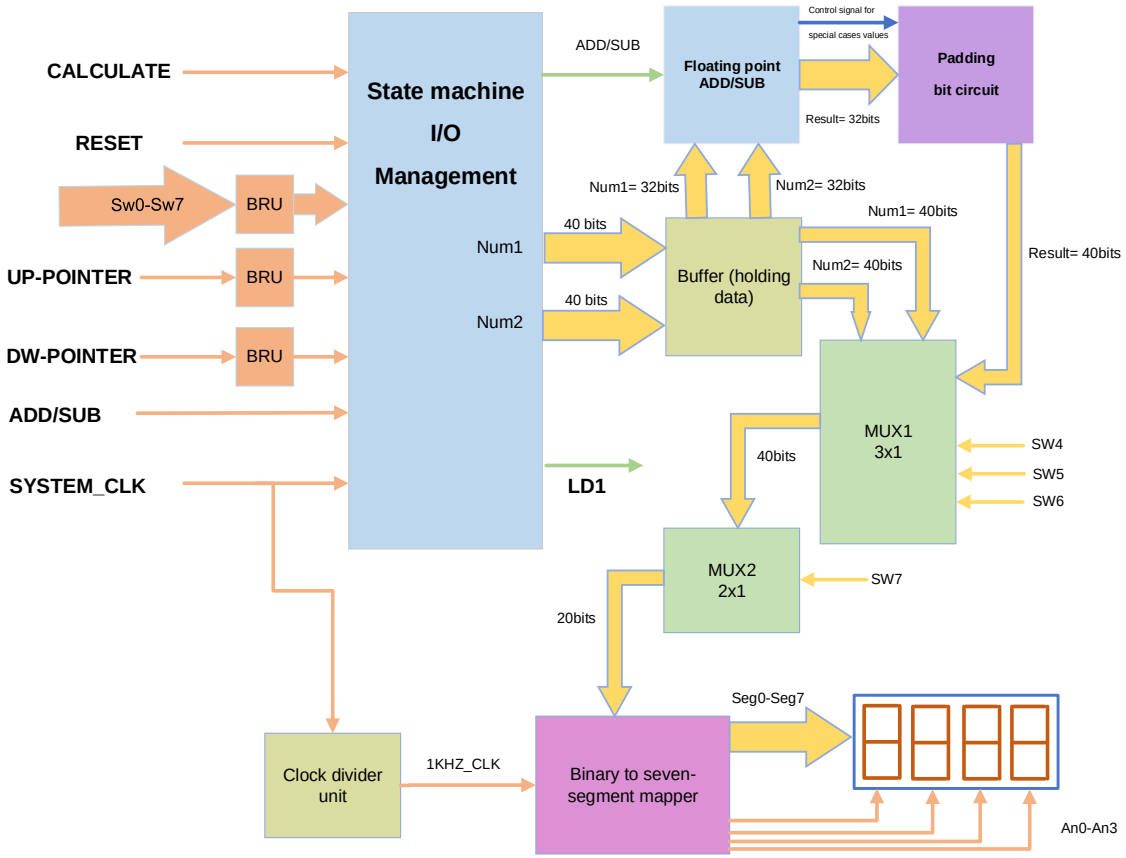

Figure 6. Module level design for the FPGA functional block

A. Bouncing removing unit (BRU): This part is considered as an essential part of any system using input signals from mechanical switches. The problem arises when a switch starts to connect, a fluctuating output between one and zero for few milliseconds before settled to a steady-state level. It can remove this effect by sampling the input signal by a 1 kHz clock that has a duration larger than the bouncing period.

B. State machine I/O management unit: This unit is used to sequentially entering FP numbers into the system. A state machine model is used to manage I/O operations as shown in Figure 7. The system initially starts with the state S_0 waiting for the input response. When UP-POINTER is active, the system displays the current switches values on the seven-segment display and transit to the state S_1. When the UP-POINTER is active, the state displays the second digit and transit to the state S_2 and so forth. If the system at state S_2 and the DW-POINTER is active, the system will remove the second digit and transit to state S_1 and so forth.

C. Buffer unit: This unit is used to hold the output data of the state machine I/O unit and prevent displaying the current switches inputs on the seven-segment display unit.

D. Floating-point addition/subtraction unit: The IEEE 754 FP standard is implemented to perform the addition/subtraction of a single-precision FP numbers. Figure 5 shows a detailed flowchart of FP arithmetic operations. It is straightforward to upgrade the proposed structure to perform arithmetic operations for a double-precision format. It can summarize the basic idea of the flowchart as shown in Figure 7. The process is started by comparing the exponent (E) of each number. After that, the difference in exponent between the two numbers is used to calculate the amount of shifting to make two exponents equal. The sign bit of the two numbers is checked. If the sign of the two numbers is the same, the addition operation is done else a subtraction operation is done.

E. Padding bit unit: This unit used for converting the word length of floating-point add/subtract unit from 4-bits to 5-bits to match the overall data bus size.

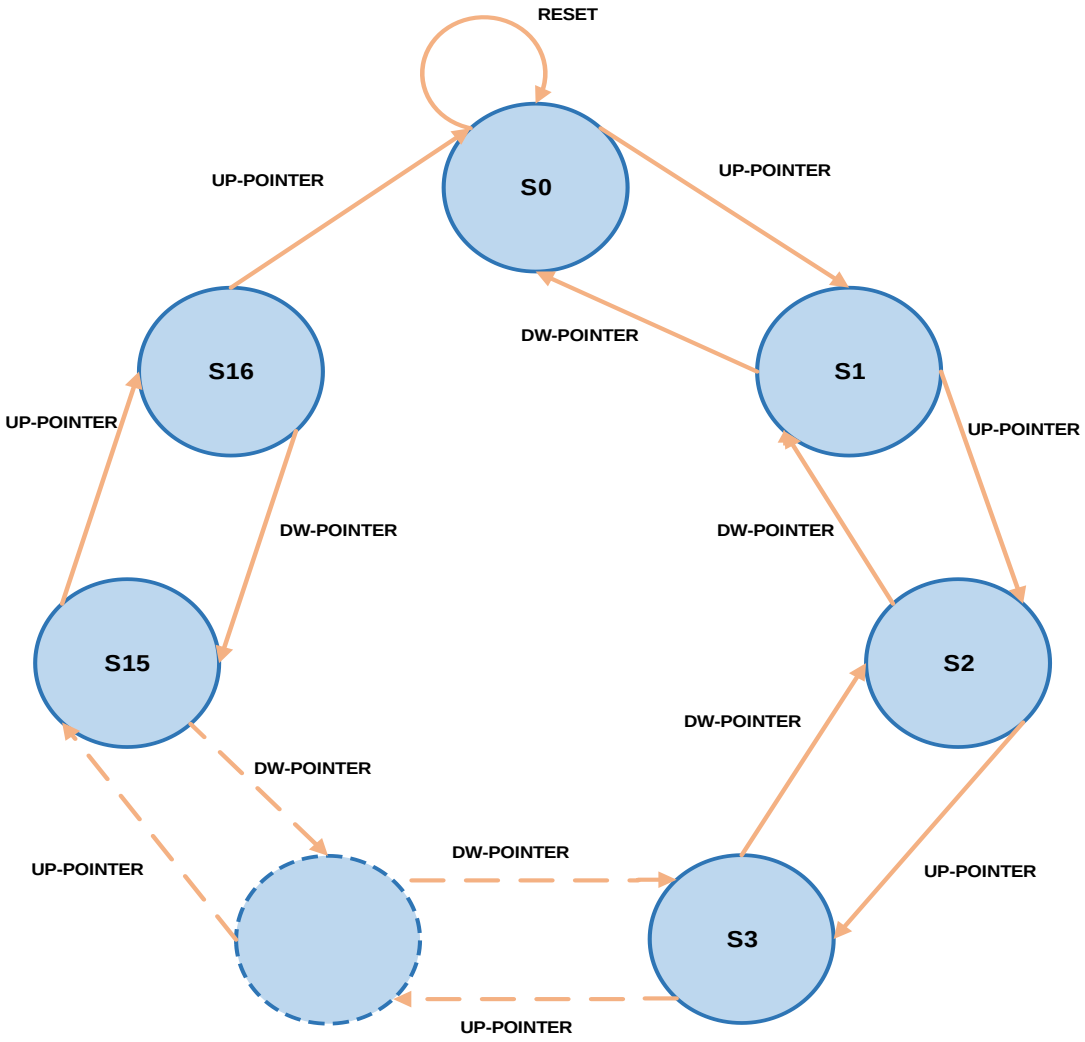

Figure 7. The state machine diagram for I/O unit

MUX1: 3×1-multiplexer is used to forward one (40-bits) word to the MUX2 unit according to the following truth table shown in Table 2.

MUX2: 2×1 Multiplexer used to organize the seven-segment operation to display only four digits at the time. The multiplexer passes the least significant four digits of an FP number when $Sw_7 = 0$ and the most significant four digits when $Sw_7 = 1$.

Seven segment display: This unit accepts 20-bits input organized as 5-bits per digit and mapped it to the seven-segment output through the seven lines ($seg_0, seg_1, \ldots, seg_7$). In addition, this unit generates four signals ($An_0, \ldots, An_3$) which are used to activate the digits sequentially at 1kHz frequency rate.

Table 2. MUX1 input-output switching

Sw6	Sw5	Sw4	Output
0	0	1	Number1
0	1	0	Number2
1	0	0	Result

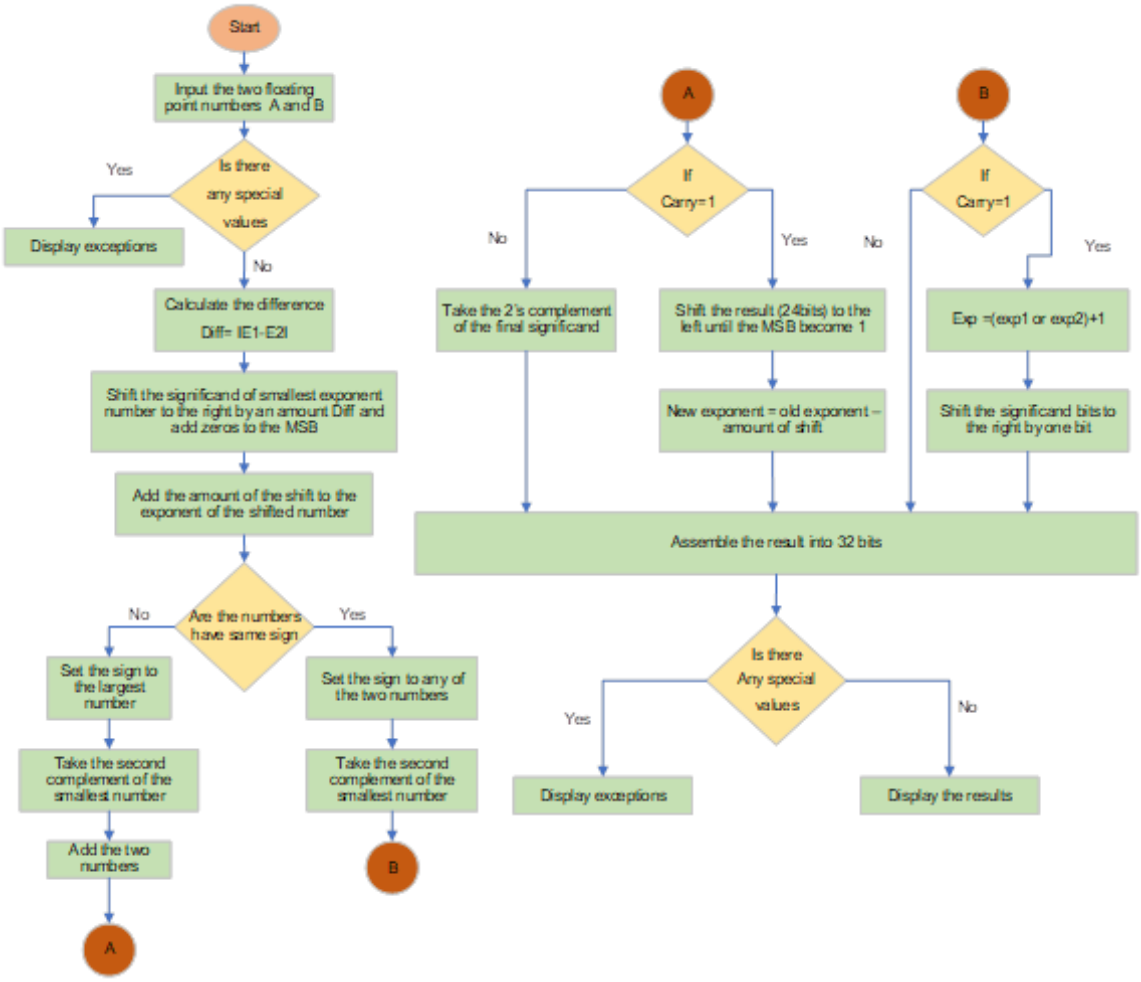

Figure 8. Flowchart of IEEE single-precision floating point for add/subtract operation

5. RESULTS AND DISCUSSION

In this section, the modules level design of FP arithmetic is implemented on Diglent Nexys3 Sparton-6 FPGA board. The language used to implement the FP algorithm and I/O mapping is VHDL. We made the implementation codes as well as other programming resources under GNU license and available on the GitHub link [25]. To make VHDL code readable we highlighted the modules-level code design by inserting comments that match the descriptions given in Section 3. The Integrated Development Environment (IDE) used to run VHDL scripts is called Integrated Synthesis Environment (ISE) design suite. Table 3 shows device summary generated by ISE design suite. The first column described the slice logic descriptions as well as the available number of these resources and how much is used in our implementation. All resources usage falls within reasonable range. The "Number of LOCed IoBs" usage is 100% since we used all input/output resources of FPGA board for validation purposes. To verify the performance of the proposed module level design we used ModelSim which is multi-language HDL simulation tool by Mentor Graphics. Figure 9 shows the tests for all special cases of FP arithmetic such as $\pm\infty$, NaN and de-normalized number states. Figure 10 and Figure 11 show examples for FP add/subtract operations according to IEEE FP standard format. Other examples and tests are also available on the GitHub link [25].

Table 3. Device utilization summary

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	368	18,224	2%
Number used as Flip Flops	366		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	2		
Number of Slice LUTs	1,179	9,112	12%
Number used as logic	1,173	9,112	12%

Slice Logic Utilization	Used	Available	Utilization
Number using O6 output only	988		
Number using O5 output only	15		
Number using O5 and O6	170		
Number used as ROM	0		
Number used as Memory	0	2,176	0%
Number used exclusively as route-thrus	6		
Number with same-slice register load	5		
Number with same-slice carry load	1		
Number with other load	0		
Number of occupied Slices	451	2,278	19%
Number of MUXCYs used	148	4,556	3%
Number of LUT Flip Flop pairs used	1,303		
Number with an unused Flip Flop	942	1,303	72%
Number with an unused LUT	124	1,303	9%
Number of fully used LUT-FF pairs	237	1,303	18%
Number of unique control sets	14		
Number of slice register sites lost to control set restrictions	58	18,224	1%
Number of bonded IOBs	27	232	11%
Number of LOCed IOBs	27	27	100%
Number of RAMB16BWERs	0	32	0%
Number of RAMB8BWERs	0	64	0%
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	3	16	18%
Number used as BUFGs	3		
Number used as BUFGMUX	0		

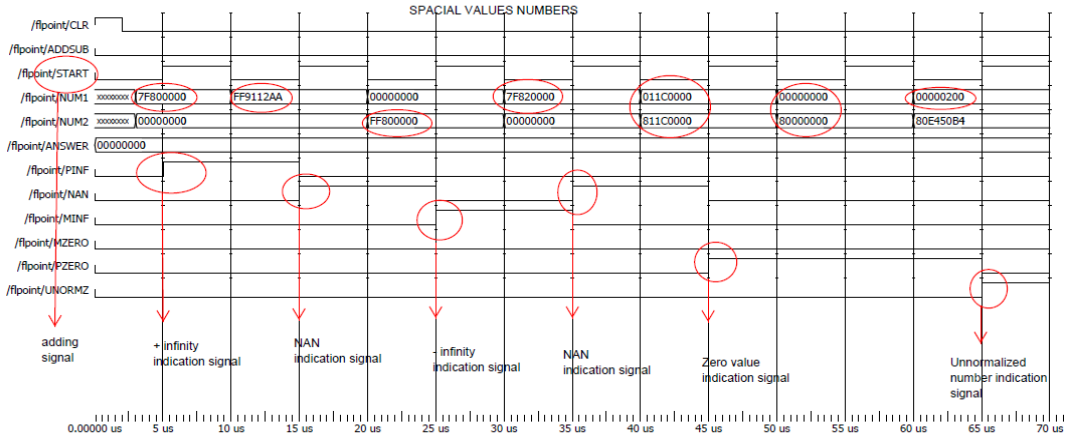

Figure 9. Simulation results for special cases FP numbers

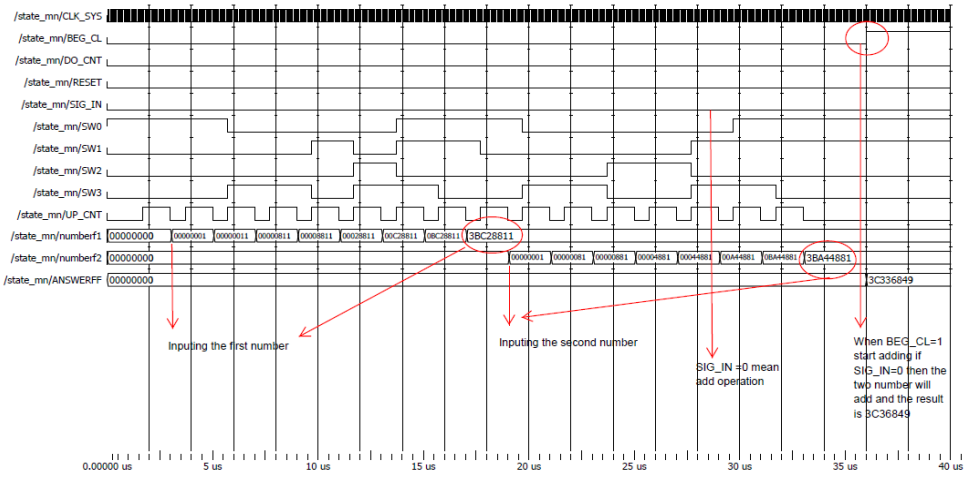

Figure 10. Simulation results for addition of two FP numbers

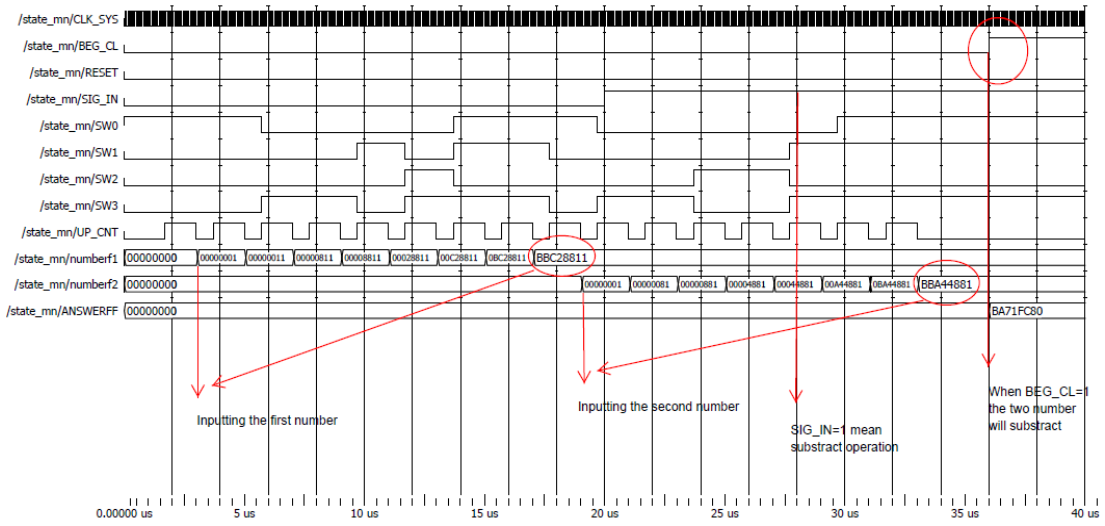

Figure 11. Simulation results for subtraction of two FP numbers

6. CONCLUSION

In this work, an open-source for IEEE single-precision floating-point algorithm is implemented on the Xilinx Nexys 3 Spartan-6 FPGA board. The overall system consists of three main parts. The first part is the I/O management unit; the purpose of this unit is to organize entering data to the board. The second part is the implementation of a single-precision FP ALU and the third part, is the sub-system that organized displaying data on the seven-segment display. We implemented the system using VHDL code and tested it successfully on the Nexys-3 board as well as on ModelSim. The proposed work can be used for test and development of FP arithmetic on low-cost FPGA platforms that do not support FP framework.

REFERENCES

[1] A. Awab, A. Ab Rahman, M. Rusli, U. Sheikh, I. Kamisian and G. Meng, "HEVC 2D-DCT architectures comparison for FPGA and ASIC implementations", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 17, no. 5, p. 2457, 2019. Available: 10.12928/telkomnika.v17i5.12815.

[2] M. Jasim Fadhil, R. Ali Fayadh and M. K. Wali, "Design and implementation a prototype system for fusion image by using SWT-PCA algorithm with FPGA technique", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 1, p. 757, 2020. Available: 10.11591/ijece.v10i1.pp757-766.

[3] Hormigo, J. and Villalba, J. "HUB Floating Point for Improving FPGA Implementations of DSP Applications." *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 3, pp.319-323, 2017.

[4] A. Hamdoon, Z. Mohammed and E. Mohammed, "Design and implementation of single bit error correction linear block code system based on FPGA", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 17, no. 4, p. 1785, 2019. Available: 10.12928/telkomnika.v17i4.12033.

[5] J. Hormigo and J. Villalba, "Simplified floating-point units for high dynamic range image and video systems," in *Consumer Electronics (ISCE 2015), 19th IEEE Int. Symp. on*, pp. 1–2, 2015.

[6] F. Silaban, S. Budiyanto and W. Raharja, "Stepper motor movement design based on FPGA", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 1, pp. 151-159, 2020. Available: 10.11591/ijece.v10i1.

[7] B. Kurniadhani, S. Hadiyoso, S. Aulia and R. Magdalena, "FPGA-based implementation of speech recognition for robocar control using MFCC", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 17, no. 4, p. 1914, 2019. Available: 10.12928/telkomnika.v17i4.12615.

[8] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *Design Test of Computers, IEEE*, vol. 28, no. 4, pp.18–27, 2011.

[9] J. Hormigo and J. Villalba, "Optimizing DSP circuits by a new family of arithmetic operators," in Signals, Systems and Computers, Asilomar Conference on, pp. 871–875, 2014.

[10] E. Nurvitadhi, et al., "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA* 17, 2017.

[11] A. Abdelkareem, S. Mohammed Saleh and A. D. Jasim, "Design and Implementation of an Embedded System for Software Defined Radio", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 6, p. 3484, 2017. Available: 10.11591/ijece.v7i6.pp3484-3491.

[12] S. D. Mu˜noz and J. Hormigo, "Improving fixed-point implementation of QR decomposition by rounding-to-nearest," *in Consumer Electronics (ISCE 2015), 19th IEEE Int. Symp. on*, pp. 1–2, 2015.

[13] A. Ehliar, "Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics," in *Field-Programmable Technology (FPT), 2014 International Conference on*, pp. 131–138, 2014.
[14] M. Langhammer and B. Pasca, "Design and implementation of an embedded FPGA floating point DSP block," in *Computer Arithmetic (ARITH), 2015 IEEE 22nd Symposium on*, pp. 26–33, 2015.
[15] S. Churiwala, Designing with Xilinx FPGAs: using Vivado. Cham: Springer, 2017.
[16] J. S. Parab, R. S. Gad, and G. M. Naik, "Hands-on experience with Altera FPGA development boards," *New Delhi, India: Springer*, 2018.
[17] S. Amin-Nejad, K. Basharkhah, and T. A. Gashteroodkhani, "Floating Point versus Fixed point Tradeoffs in FPGA Implementations of QR Decomposition Algorithm," *European Journal of Electrical Engineering and Computer Scienc*e, vol. 3, no. 5, 2019.
[18] R. Solovyev, A. Kustov, D. Telpukhov, V. Rukhlov and A. Kalinin, "Fixed-Point Convolutional Neural Network for Real-Time Video Processing in FPGA," *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia*, pp. 1605-1611, 2019.
[19] D. L. N. Hettiarachchi, V. S. P. Davuluru and E. J. Balster, "Integer vs. Floating-Point Processing on Modern FPGA Technology," *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, pp. 0606-0612, 2020.
[20] J. Villalba, J. Hormigo and S. González-Navarro, "Fast HUB Floating-Point Adder for FPGA," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 6, pp. 1028-1032, 2019.
[21] R. P. Rao, N. D. Rao, K. Naveen and P. Ramya, "Implementation Of The Standard Floating Point Mac Using Ieee 754 Floating Point Adder," *2018 Second International Conference on Computing Methodologies and Communication (ICCMC), Erode*, pp. 717-722, 2018.
[22] M. Shirke, S. Chandrababu and Y. Abhyankar, "Implementation of IEEE 754 compliant single precision floating-point adder unit supporting denormal inputs on Xilinx FPGA," *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai*, pp. 408-412, 2017.
[23] B. Mathis and J. Stine, "A Novel Single/Double Precision Normalized IEEE 754 Floating-Point Adder/Subtracter," 2019 *IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA*, pp. 278-283, 2019.
[24] IEEE Task P754, IEEE 754-2008, Standard for Floating-Point Arithmetic, 2008.
[25] Hassan, M. (2019). Design-and-Implementation-of-Floating-Point-Units-for-FPGAs. [online] GitHup. Available at: https://github.com/mufalh/Design-and-Implementation-of-Floating-Point-Units-for-FPGAs [Accessed 14 Nov. 2019].