Erzincan Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
2022, 15(3), 699-714
ISSN: 1307-9085, e-ISSN: 2149-4584
Araştırma Makalesi

Erzincan University
Journal of Science and Technology
2022, 15(3), 699-714
DOI: 10.18185/erzifbed.1077921
Research Article

Design of RISC Processor with IEEE754 Standard Floating-Point Instruction Set in FPGA using VHDL for Digital Signal Processing Applications

Bahadır ÖZKILBAÇ [1*], Tevhit KARACALI [1]

[1] Department of Electrical-Electronics Engineering, Faculty of Engineering, Ataturk University, Erzurum, Turkey

Received: 23/02/2022, **Revised:** 23/09/2022, **Accepted:** 19/06/2022, **Published:** 30/12/2022

Abstract

The design of RISC processors, which are the key of digital signal processing applications, are increasing in reconfigurable hardware. FPGAs are suitable reconfigurable hardware for RISC processor design, with advantages such as parallel processing and low power consumption. In this study, the design of the 32-bit RISC processor in a FPGA is presented. The designed RISC processor contains IEEE754 standard floating-point number processing unit, which is executed in one clock cycle. The verification of the processor is performed for the Zynq-7000 SoC Artix-7 FPGA chip in the Xilinx Vivado tool. Classification of an artificial neural network using the iris dataset is carried out in this designed RISC processor. In order to compare the performance, the same artificial neural network is executed in real time in the dual-core ARM Cortex-A9 processor in the operating system of the Zynq-7000 SoC. The results show that the RISC processor designed in the FPGA executes at 20x less clock cycles and 3x higher speed compared to the ARM processor.

Keywords: FPGA, ARM, RISC, reconfigurable hardware

VHDL Kullanarak, Dijital Sinyal İşleme Uygulamaları için IEEE754 Standart Kayan Noktalı Komut Kümesine Sahip RISC İşlemcinin FPGA'de Tasarımı

Öz

Dijital sinyal işleme uygulamalarının kalbi niteliğinde olan RISC işlemcilerin, yeniden yapılandırılabilir donanımlardaki tasarımları giderek artmaktadır. FPGA'ler, paralel çalışma, düşük güç tüketimi gibi avantajlara sahip, RISC işlemci tasarımı için ideal yeniden yapılandırılabilir donanımlardır. Bu çalışma 32-bit RISC işlemcinin FPGA'de tasarımını sunmaktadır. Tasarlanan RISC işlemci, tek saat darbesinde işlem yapabilecek paralellikte olan IEEE754 standartında kayan noktalı sayı işlem birimini içermektedir. İşlemcinin doğrulması, Xilinx Vivado aracında Zynq-7000 SoC Artix-7 FPGA çipi için yapılmıştır. Iris dataseti kullanılarak bir yapay sinir ağının sınıflandırma işlemleri, tasarlanan bu RISC işlemci içerisinde gerçekleştirilmiştir. Performans kıyaslaması yapabilmek için aynı yapay sinir ağı Zynq-7000 SoC'nin işletim sistemi kısmında bulunan çift çekirdekli ARM Cortex-A9 işlemcisinde de gerçek zamanlı olarak çalıştırılmıştır. Elde edilen sonuçlar, FPGA içerisinde tasarlanan RISC işlemcinin, ARM işlemciye kıyasla 20 kat daha az saat darbesinde 3 kat daha yüksek hızda bu işlemi gerçekleştirdiğini göstermektedir.

Anahtar Kelimeler: FPGA, ARM, RISC, yeniden yapılandırılabilir donanım

Corresponding Author: bahadir.ozkilbac@atauni.edu.tr
Bahadır ÖZKILBAÇ, https://orcid.org/0000-0002-3384-1565
Tevhit KARACALI, https://orcid.org/0000-0002-3647-6372

1. Introduction

With the development of technology, digital signal processing applications (DSP) are increasing significantly in space and defense industry, medicine and various commercial areas. The reduced instruction set computer (RISC) processor is at the heart of such DSP applications [1]. All processors have an instruction set architecture that perform operations according to instructions. Unlike other processors, the RISC processor architecture has a simple instruction set and each instruction is executed in one clock cycle. [2]. Through these advantages, the RISC processor offers a flexible and extensible architecture that provides maximum performance for any processing technology. RISC architectures are used today in phones, tablets and personal computers, as well as supercomputers such as Fugaku, Summit, Sierra, Sunway and TaihuLight [3]. The RISC processor has also been used in DSP applications, video decoding and image acquisition [4]-[5]. Such as applied mathematics and control systems, computations are other usage area of the RISC processor [6]-[7]. In artificial intelligence, speech recognition and motion estimation are examples of the use of the RISC processor [8]-[9].

Reconfigurable computing bridges the gap between software and hardware design using hardware like field programmable gate arrays (FPGAs) [10]. The role of reconfigurable processors in embedded system design has been increasing in recent years. Owing to reconfigurable hardware such as FPGA, processor architectures can be changed by programming [11]. Various RISC processor have been designed in the literature using FPGA. FPGA-based 64-bit RISC processor with self-test verificated using VHDL was designed in the Xilinx ISE tool [12]. [13] describes a 16-bit RISC processor designed using VHDL, in which behavioral model is preferred to create components. The design is a four-stage pipelined processor. According to [14] presented the design of a pipeline microprocessor without interlocked pipeline stages (MIPS) RISC processor using VHDL. 32-bit RISC processor-based MIPS was designed using VHDL [15]. Unlike the previous paper given above, it presents five-stage, pipeline processor. A from top to down approach is preferred in the design. 16-bit RISC processor design is presented using the VHDL language [16]. The design is simulated and synthesized using Xilinx ISE 13.1. Pipelining is used to speed up to the processor. In pipelining, the instruction cycle is seperated so that multiple processes can be executed in parallel. Designs of subunits of processor are also among the studies in the literature. In the first of these studies, the fetch and decode units of the RISC processor was designed [17]. The design is successfully simulated in the Quartus II tool of Intel. In [18], it is aimed to design the fetch unit and ALU, which are subunits of the RISC processor. The Fetch unit is designed to read instructions stored in memory. The ALU that performs all the computations such as arithmetic and logical operations is at the execution stage of the pipeline. The Xilinx ISE 8.1 tool is used to simulate the design with the VHDL.

When designing real-time digital systems using the RISC processor, there is usually a need to operate with fractional or large numbers. To satisfy this need, various number representations are used according to the performance and accuracy of the design. The floating-point number representation is standardized by IEEE754 and used to represent real numbers in the binary number system. The design of floating point processing units is quite important for system

performance. In this study, RISC processor with IEEE754 standard floating-point unit (FPU) is designed in FPGA. Due to FPU, unlike other RISC processor designs in the literature, the designed processor can process with fractional numbers. Thus, it becomes suitable for real-time signal processing applications. In addition, taking advantage of the parallel processing of the FPGA, the FPU is designed to operate in one clock cycle. Thus, the processor reduces the execution time of any application. This design is verificated in the Xilinx Zynq-7000 FPGA chip. Then, classification of iris flower is done by using artificial neural network in the designed processor. Finally, in order to compare the performance, the classification is executed in real time on the ARM processor in the Zynq-7000 chip in the same artificial neural network. The remainder of this article is organized as follows. In section 2, information about floating-point number arithmetic is given and the architecture of the designed RISC processor is presented. In section 3, simulation and synthesis studies of artificial neural network classifying iris flower species in RISC processor are given to verify the behavioral function of the RISC processor.

2. Material and Methods

This section presents the floating-point number representation, the floating-point arithmetic and the RISC processor designed in the FPGA, respectively.

2.1. IEEE754 Standart Floating-Point Number Representation

Floating-point number representation, which was made a standard by the IEEE in 1985, consists of three formats: single precision, double precision and extended precision [19]. A number represented as an IEEE754 floating point consists of sign, exponent and fraction bits. According to the format used, the bit lengths of the exponent and fraction changes. In Table 1, information about bit lengths for single precision, double precision and extended precision number formats is given.

Table 1. Floating-point number formats

Format	Sign Bits	Exponent Bits	Fraction Bits
Single	1	8	23
Double	1	11	52
Extended	1	15	112

2.2. Floating Point Arithmetic

Multiplication, division, addition, and subtraction are among floating-point number arithmetic operations. All operation are executed in specially designed units. Floating-point addition and subtraction is performed in only one unit given in Figure 1. This unit has the highest latency, the longest design time and the most complex among floating-point operations.

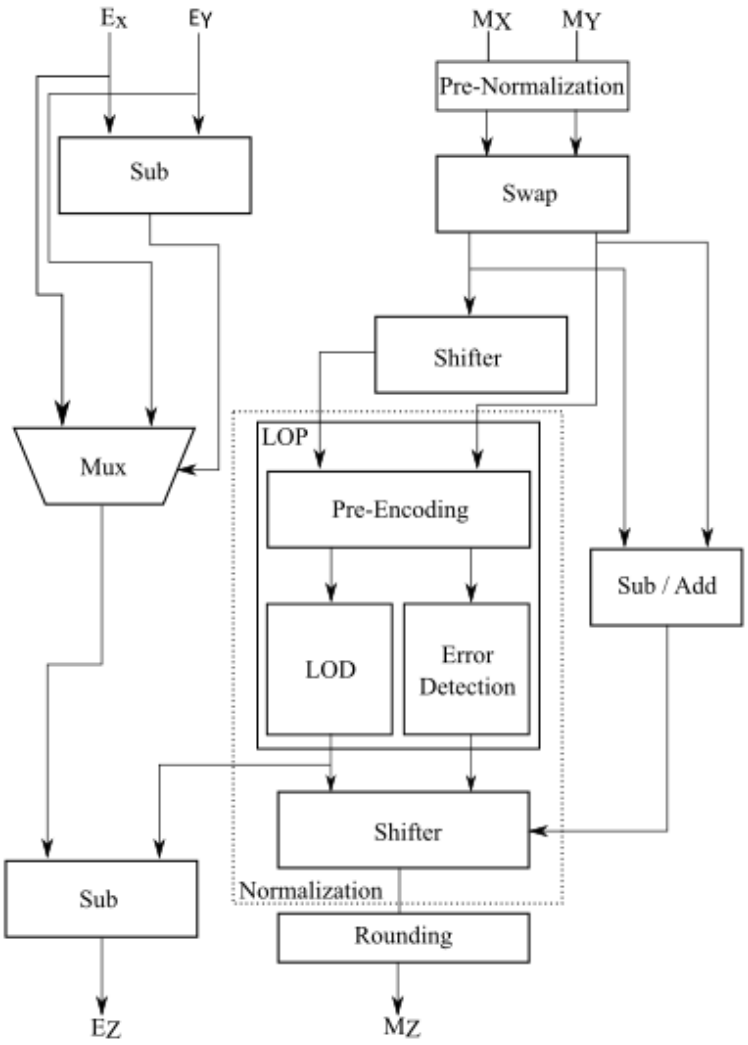

Figure 1. Adder/subtractor micro-architecture

Firstly, in the floating point addition/subtraction unit, first the exponents of the two numbers are subtracted to determine the large and small number. The '1' bit is placed to the left of the fraction bits for pre-normalization. The fraction of the smaller number is shifted to the right according to subtraction of exponents. The fraction from the shift operation and the fraction of the larger number are sent to the sub/add module. While the process is running in the sub/add module, the estimation of the position of the most significant '1' bit required for normalization is done by the leading one predictor module (LOP). Normalization is performed by shifting the numbers to the left by the estimated amount from LOP if the signs are different, or by shifting one bit to the left or right depending on the situation. The fraction obtained by normalization is rounded. To determine the exponent of the result, the exponent of the large number is set and the result is obtained.

Multiplication, another floating-point operation, is among the most used operations along with addition in digital signal processing applications. Floating-point multiplication is executed within the unit given in Figure 2.

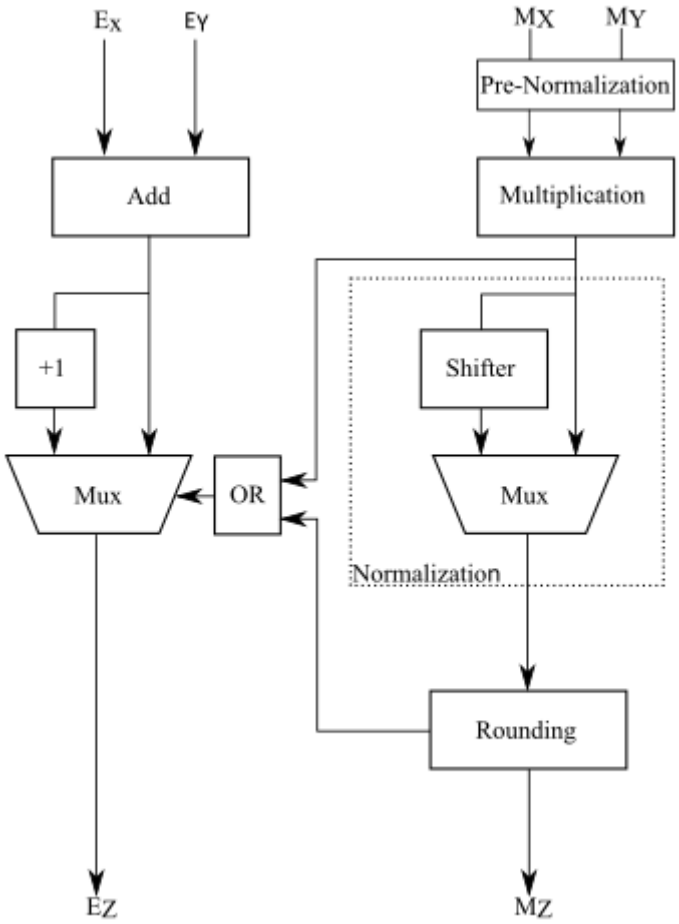

Figure 2. Multiplier micro architecture

Pre-normalization is performed by placing '1' to the left of the most significant bit of the fraction of two numbers in the floating point multiplication unit. After the pre-normalization, the fractions of the numbers are multiplied. As the multiplication continues, the exponents of the numbers are added to get the exponent of the result. The most significant bit of the multiplication result of the fractions is checked. If the corresponding bit is '1', the exponent resulting from the sum is increased by 1. The obtained fraction is rounded so that the result is found.

The floating-point division operation is similar to the multiplication. It is in the form of subtracting two numbers and dividing fractions. The micro-architecture of the floating point divider unit is given in Figure 3.

In the floating-point division operation, pre-normalization is performed by placing '1' to the left of the most significant bit of the fractions of two numbers. The fractions got by pre-normalization are divided. At this time, the exponents of the numbers are subtracted in parallel. The fraction from the division operation is rounded to form the fraction of the result, thus the result is obtained.

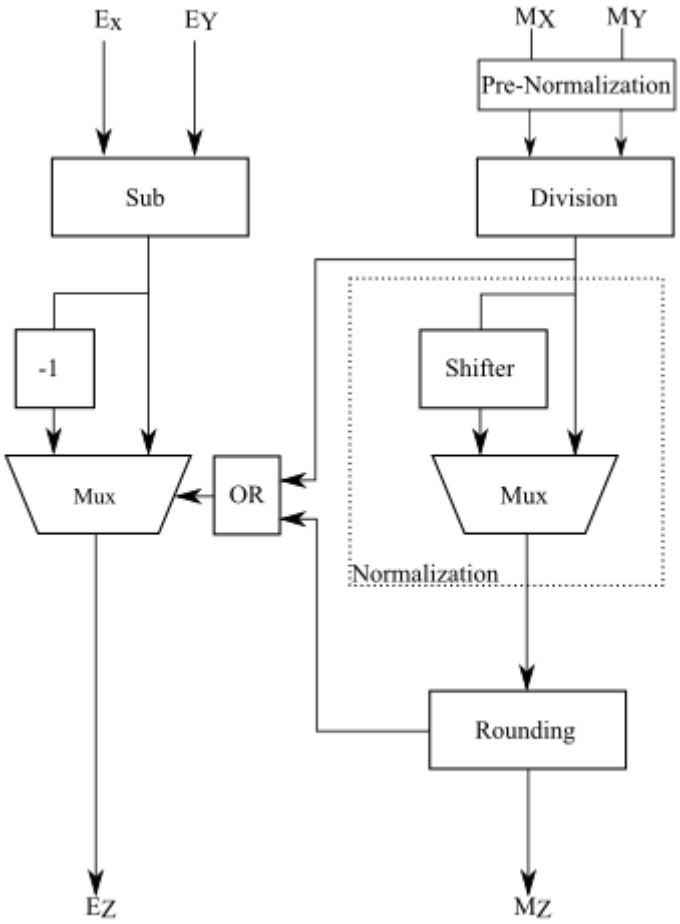

Figure 3. Floating-point divider micro architecture

2.3. 32-bit RISC Processor Design

In this section of the study, information is given about the designed RISC processor, FPU unit, CPU, registers, program counter, main memory and I / O ports.

Before any computer hardware can be designed, the instruction should be defined. Some computer hardware has very few instructions to reduce the physical size of the circuit required by the CPU. Such a design allows the CPU to execute instructions at a high frequency. This architectural approach used when designing computer hardware is called RISC [2].

In a RISC processor, one clock cycle is required to execute any instruction. It is easy to convert a RISC processor into a pipelined design, because all instructions are processed in the same time and the opcode and the operand take the same position in the number sequence. The RISC processor to be designed in this study consists of AN FPU operating in IEEE754 single precision floating-point number representation, a program memory, a data memory, a control unit and registers. The micro-architecture of the designed RISC processor is given in Figure 4.

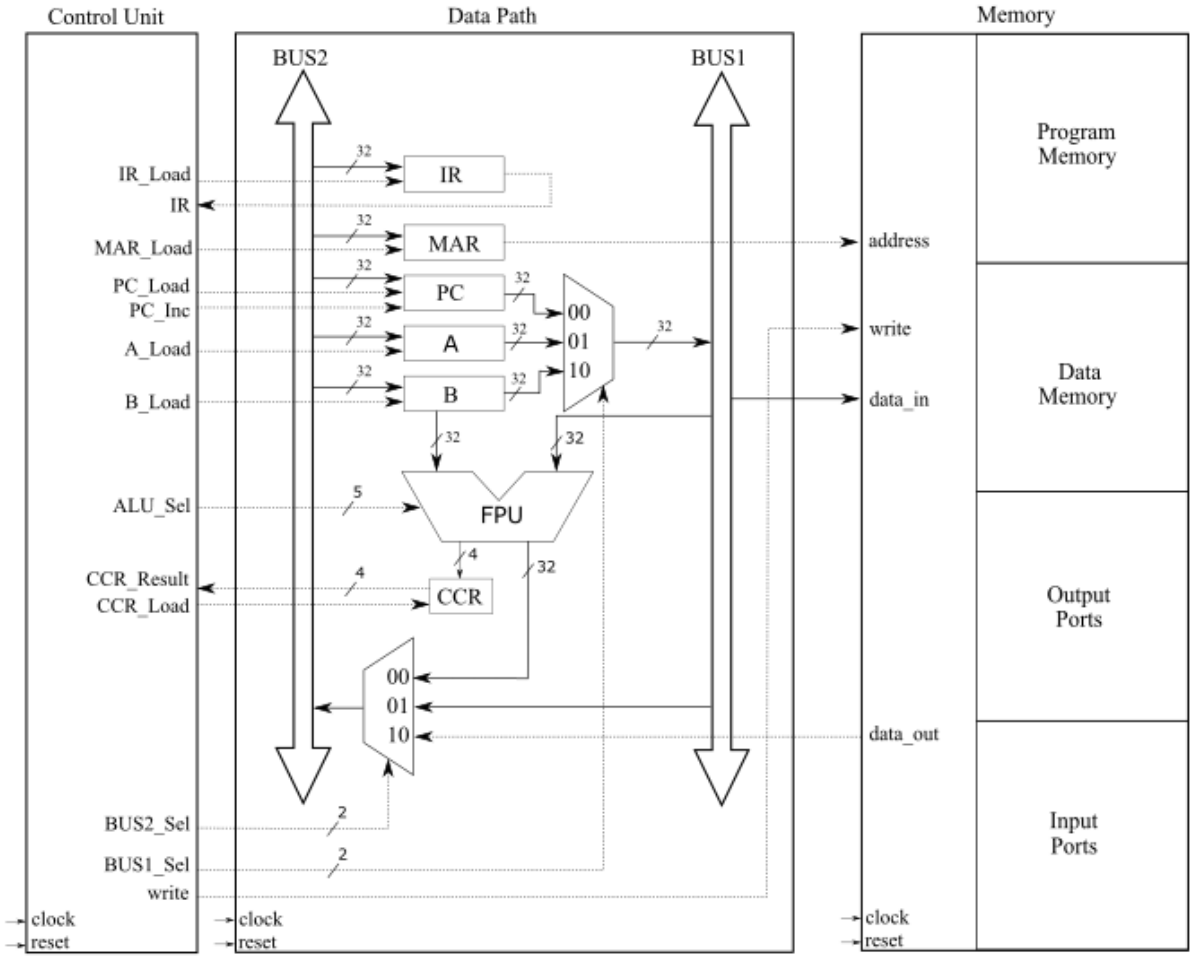

Figure 4. RISC processor micro architecture.

2.3.1. Control Unit

All operations performed within the RISC processor are managed by the control unit. Therefore, the design of the control unit is quite important so that the instruction to work properly in the right order. In this study, the control unit is designed as a finite state machine (FSM) that executes the Fetch, decode and execute stages. Two status signals, current state and next state, in the control unit are shown in Figure 5.

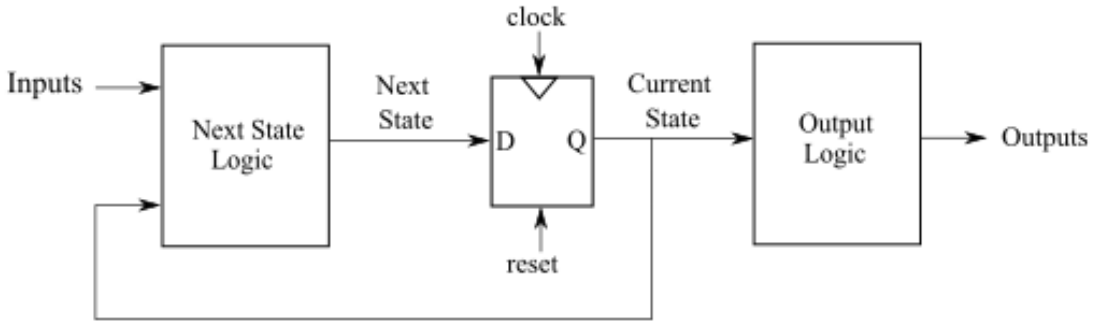

Figure 5. Control unit FSM

The current state is in a synchronous and sensitive to the clock and reset signal. Current state changes its state according to next state. Next state is a combinational structure that runs according to the current state and inputs. In addition, the output logic circuit, which is responsible for producing the relevant output in any situation, which changes depending on the current state, is also located in the control unit. The output logic circuit is only sensitive to the current state. Thus, the signals produced by the control unit can be sent to the memory or data bus asynchronously, without waiting for the next clock cycle. As can be seen in Figure 6, the current state and the next state in the control unit are three stages: Fetch, Decode and Execute.

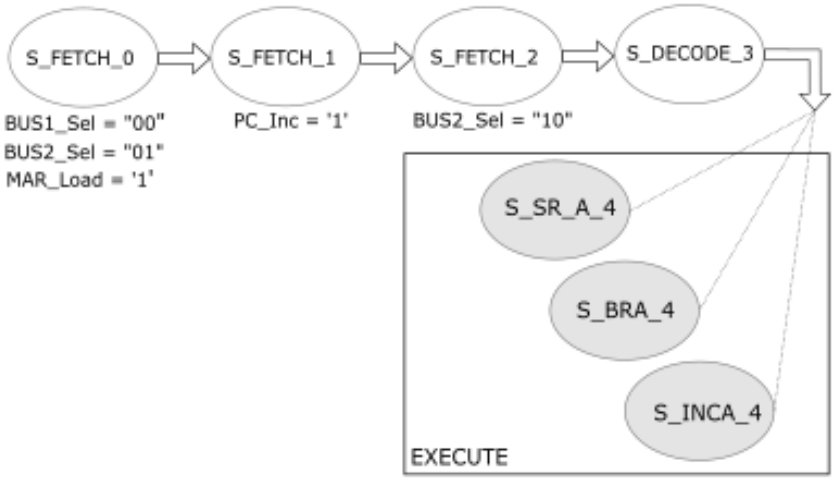

Figure 6. Fetch, Decode and Execute stages

In Fetch, the instruction in the program memory is read and written to the instruction register (IR). Fetch consists of three sub-branches. In S_FETCH_0, firstly BUS1_Sel signal is set to "00" value and PC register value is sent to BUS1. Then, the BUS2_Sel signal is set to "01" and the data in BUS1 is sent to BUS2. Finally, by making the MAR_Load signal '1', the address going to the memory is updated. Thus, the data at the corresponding address in the memory can be accessed.

The program counter is made ready by setting PC_Inc signal as '1' in S_FETCH_1 and increasing the value of PC register by 1. In S_FETCH_2, the BUS2_Sel signal is set to "10" and the value read at the relevant address in the memory is transferred to BUS2. Then, the IR_Load signal is set to '1' and the instruction in BUS2 is loaded into the IR register. In Decode, the identity of the instruction is detected. Finally, the instruction detected in the execute state is performed inside the processor. For this, the instructions given in Table 2 are used.

The instructions are executed sequentially in the program memory and the desired operation is performed by the processor. These instructions actually create a software. By correctly writing and sequencing the instructions, the software to be executed by the processor is built. Executing the software and converting it to machine code is carried out by the processor.

Table 2. Instruction Set of the Designed RISC Processor

Type	**Instruction**
Loads and Stores	*LDA_IMM, LDA_DIR LDB_IMM, LDB_DIR STA_DIR, STB_DIR*
Floating-Point Arithmetic	*FP_ADD_AB FP_SUB_AB FP_DIV_AB FP_MULT_AB*
Logical	*AND_AB, OR_AB, XOR_AB, NOT A, INCA, INCB DECA, DECB SR_A, SL_A SR_B, SL_B*
Branches	*BRA, BMI, BPL, BEQ, BNE, BVS, BVC, BCS, BCC*

The instructions for the software to be executed by the processor are written to the program memory in order. The sequential reading of these instructions in the program memory is realized by the PC register. In Figure 7, the PC register in the processor is given.

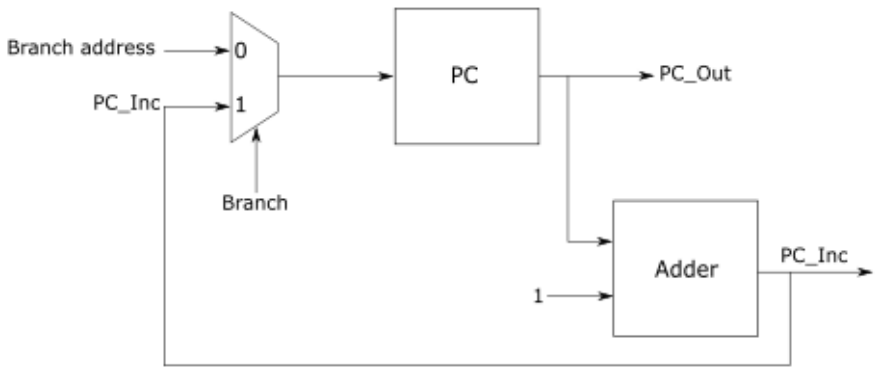

Figure 7. PC register structure

The PC register actually runs like a counter. The value of the PC register is zero in the first stage for any software to be implemented. Then the value of the PC register is incremented by 1 in each time a read is made from the memory. With the branches instructions listed in Table 2, the PC register jumps to the address value in the operand with or without negative, zero, overflow and carry flag conditions. This ensures that the instructions are executed correctly.

2.3.2. Program Memory

The program memory where the instructions are stored is located in the memory given in Figure 8 in the processor.

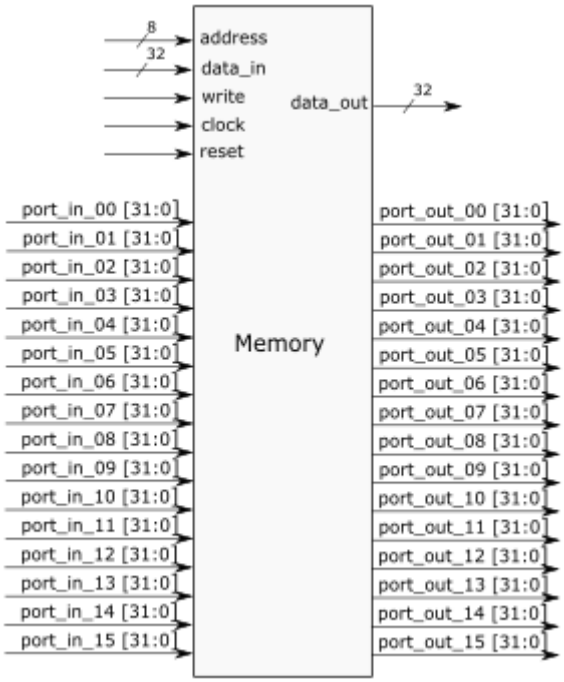

Figure 8. Processor memory

Memory consists of program memory, data memory and input and output ports. The program memory has a capacity of 512 bytes, and the data memory has a capacity of 384 bytes. Any instruction or data from the memory is read and written with the MAR register. The counter value in the PC register is sent to BUS2 in order to read or write. Then, the MAR_Load signal is set '1' and the address value in BUS2 is sent to the MAR output and thus to the memory.

2.3.3. Floating-Point Unit (FPU)

In the FPU, logical operations and shift operations are carried out as well as floating-point arithmetic operations. Figure 9 shows the FPU.

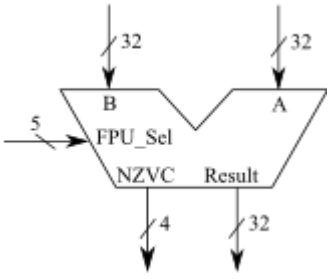

Figure 9. Floating-point unit

Which operation will be executed in the FPU is selected by FPU_Sel. The operations corresponding to FPU-Sel bits are listed in Table 3.

Table 3. FPU operations

FPU_Sel	Operation
00000	*A = A + B in floating-point*
00001	*A = A - B in floating-point*
00010	*A = A * B in floating-point*
00011	*A = A / B in floating-point*
00100	*A = A and B*
00101	*A = A or B*
00110	*A = A xor B*
00111	*A = not A*
01000	*B = not B*
01001	*A = A + 1 in floating-point*
01010	*B = B + 1 in floating-point*
01011	*A = A – 1 in floating-point*
01100	*B = B – 1 in floating-point*
01101	*Shifts A one bit to the right*
01110	*Shifts A one bit to the left*
01111	*Shifts B one bit to the right*
10000	*Shifts B one bit to the left*

The 4-bit NZVC number got from the FPU output is used to store negative, zero, overflow, carry exceptions. These exceptions are loaded into the CCR register and sent to the control unit. The corresponding NZVC values for the exceptions are listed in Table 4.

Table 4. NZVC exceptions

ALU_Sel	Operation
Negative Results	*1000*
Zero Result	*0100*
Overflow Result	*0010*
Carry Result	*0001*

3. Design Verification and Simulation Results

The processor is designed using the very high-speed integrated circuit hardware description language (VHDL). Behavioral verification of the designed processor is done on Xilinx Zynq-7000 using Vivado Design Suite. In order to perform the behavioral verification, the machine

code for the artificial neural network (ANN) given in Figure 10 is programmed and placed in the program memory.

ANN structure is quite simple. Inputs and weights are dot multiplication within the processor, and each product is summed. The result is calculated by adding bias to the summed value. The result obtained is passed through a decision mechanism. Also, the activation function used is a linear function.

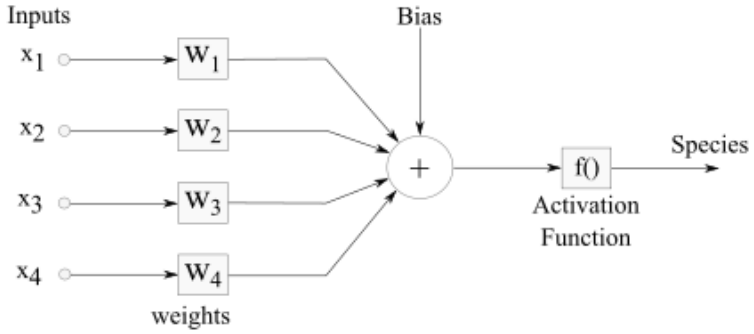

Figure 10. An artificial neural network architecture for Iris classification

100 data taken from the iris dataset, 67 are used for training the network and 33 are used for testing. Some test data used as input are given in Table 5 [20].

Table 5. Test data for iris flower

Sepal Length (x_1)	**Sepal Width (x_2)**	**Petal Length (x_3)**	**Petal Width (x_4)**	**Species**
6	*2.7*	*5.1*	*1.6*	*Versicolor*
4.8	*3*	*1.4*	*0.3*	*Setosa*
5.5	*2.3*	*4*	*1.3*	*Versicolor*
5.9	*3.2*	*4.8*	*1.8*	*Versicolor*
5.1	*3.8*	*1.9*	*0.4*	*Setosa*
5.1	*3.4*	*1.5*	*0.2*	*Setosa*
4.6	*3.6*	*1*	*1.1*	*Setosa*

These features are sent as input to the network and the species of iris flower is obtained at the output. ANN is simulated in Vivado Design Suite to verify the designed RISC processor. The simulation results for the features in the first four rows given in Table 5 are as in Figure 11.

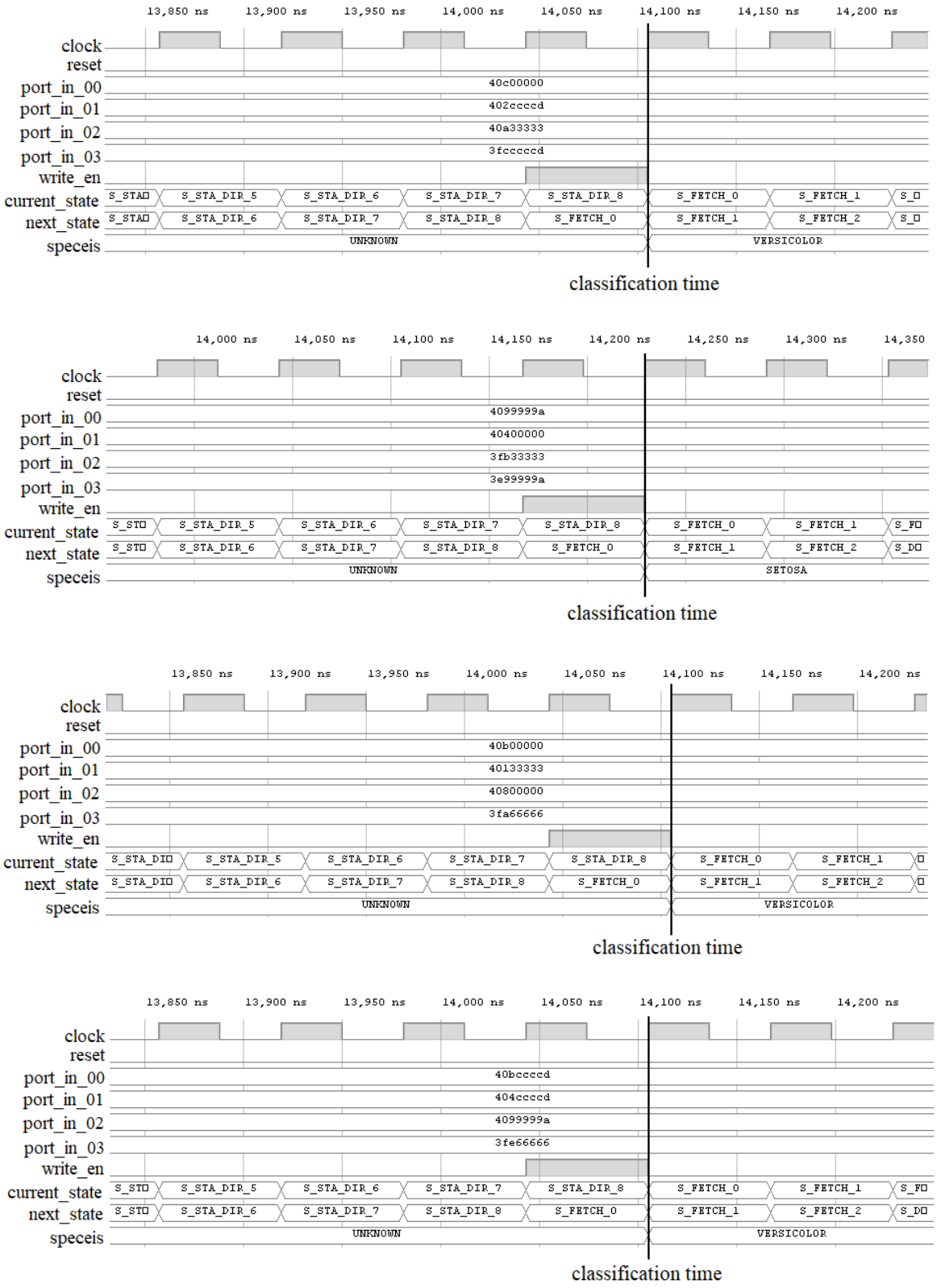

Figure 11. Simulation results for iris classification

In order to compare performance, the software of the same artificial neural network is carried out in real time in the dual-core ARM Cortex A9 processor in the same chip, the Zynq-7000, and the results given in Table 6. Results are obtained by Xilinx Vivado tool.

Table 6. Frequency, clock cycle and execution time for FPGA and ARM

	Frequency (MHz)	**Clock Cycle**	**Execution Time (µs)**
RISC Processor Designed in FPGA	*16*	*456*	*14.11*
Software in ARM Processor	*100*	*9140*	*45.7*

4. Conclusion

FPGAs are widely used in processor design due to their advantages such as parallel processing, low power consumption and reconfiguration. This study focuses on the design of a RISC processor in a FPGA. Unlike other RISC designs, the RISC processor in this study includes an FPU. Through the FPU, various complex digital signal processing applications such as ANN, image processing, object recognition and video decoders can be implemented simply by software on this processor. Classification of Iris flower species is executed using ANN in the designed processor. In this way, both hardware design and software programming of the RISC processor in FPGA are included and its verification is also provided. The classification is implemented using the same ANN on the ARM processor to compare the performance. The results show that the processor designed in FPGA operates faster and with fewer clock cycles.

Ethics in Publishing

There are no ethical issues regarding the publication of this study.

Author Contributions

B.Ö: programming, design of the study, T.K: presented idea of the study. All authors discussed the results and contributed to the final manuscript.

References

[1] Palekar, S., & Narkhede, N. (2016) 32-Bit RISC processor with floating point unit for DSP applications, Paper presented at the 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT).

[2] LaMeres, B. J. (2019) Introduction to logic circuits & logic design with VHDL. Springer.

[3] Amit, S. (2006) Mac OS X internals: a systems approach, Addison-Wesley Professional.

[4] Yamada, K., Kojima, M., Shimizu, T., Sato, F., & Mizuno, T. (2002) A new RISC processor architecture for MPEG-2 decoding, IEEE Transactions on Consumer Electronics, 48(1) 143-150.

[5] Kumar, P. S., Shashidhar, B., & Bhargav, J. S. (2010) Image acquisition from CMOS Active Pixel Sensor using RISC processor. Paper presented at the 2010 International Conference on Signal and Image Processing.

[6] Garbey, M. (2005) Acceleration of the Schwarz method for elliptic problems. SIAM Journal on Scientific Computing, 26(6) 1871-1893.

[7] Bhakti, T. L., Susanto, A., Santosa, P. I., & Widayati, D. T. (2012) Design of Bovine SemenTemperature ControllerUsing PID. Int. J. of Comp. Eng. Res, 2(7) 52-58.

[8] Chang, C.-T., Chang, C.-T., Yang, H.-L., & Chang, H.-T. (1996) Real-time implementation of speech recognition using RISC processor core. Paper presented at the Proceedings Ninth Annual IEEE International ASIC Conference and Exhibit.

[9] Bilal, M., & Masud, S. (2007) Efficient color space conversion using custom instruction in a risc processor. Paper presented at the 2007 IEEE International Symposium on Circuits and Systems.

[10] Hauck, S., & DeHon, A. (2010) Reconfigurable computing: the theory and practice of FPGA-based computation: Elsevier.

[11] Ball, J. (2007) Designing soft-core processors for FPGAs. In Processor Design (pp. 229-256): Springer.

[12] Mohammad, I., Ramananjaneyulu, K., & Veeraswamy, K. (2012) FPGA implementation of a 64-bit RISC processor using VHDL. International Journal of Engineering Research and Applications (IJERA), 2(3) 2544-2549.

[13] Thakor, K. P., & Pal, A. (2017) Design of a 16-bit RISC Processor Using VHDL. Int. J. Eng. Res. Technol (IJERT), 6.

[14] Valli, B., Kumar, A. U., & Bhaskar, B. V. (2012) FPGA Implementation and Functional Verification of a Pipelined MIPS Processor 1.

[15] Mane, P. S., Gupta, I., & Vasantha, M. (2006) Implementation of RISC Processor on FPGA, Paper presented at the 2006 IEEE International Conference on Industrial Technology.

[16] Kadam, S. U., & Mali, S. (2016) Design of risc processor using VHDL, 2016 International Journal of Research Granthaalaya, 4(6).

[17] Luker, J. D., & Prasad, V. B. (2001) RISC system design in an FPGA. Paper presented at the Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems. MWSCAS 2001 (Cat. No. 01CH37257).

[18] Ghaturle, M. S., & Kadam, R. (2017) Review Paper on 32-Bit RISC Processor with Floating Point Arithmetic. Int. Research Journal of Engineering and Technology (IRJET), 4.

[19] Electrical, I. o., Committee, E. E. C. S. S., & Stevenson, D. (1985) IEEE standard for binary floating-point arithmetic: IEEE.

[20] Fisher, R. A. and Marshall M. (1988) "UCI repository of machine learning databases". University of California.