©2020 Published in 8th International Symposium on Innovative Technologies in Engineering and Science 23-25 October 2020 (ISITES2020 Bursa - Turkey)
https://doi.org/10.33793/acperpro.03.01.59

Exponential Computing Digital Circuit Design Developed for FPGA-based Embedded Systems

*[1]Serkan Dereli and [2]Mahmut Uç
[1,2]Bilgisayar Teknolojileri ve Programlama Bölümü, Sakarya Uygulamalı Bilimler Üniversitesi, Sakarya, Türkiye

Abstract

Digital systems consist of thousands of digital circuit blocks operating in the background, working in their simplest form such as addition, subtraction, multiplication, division. In exponential expressions like square roots and cube roots, just like these circuits, it is found in many digital systems and performs tasks. Although these processes seem to be used only in circuits carrying out mathematical operations, they actually take an active role in solving many engineering problems. In this study, a digital circuit design that computes both the integer and a floating point exponent of a 32-bit floating-point number has been realized. This digital circuit, which is coded with VHDL language, can be used from beginner to advanced level in FPGA based systems. This digital circuit, which is coded with VHDL language, can be used from beginner to advanced level in FPGA based systems. In addition, three floating IP cores - logarithm, multiplication and exponent - were used in this digital circuit, and results were obtained with a total of five finite state machines in sixty-six clock pulse time.

Key words: FPGA design, embedded system, IP-Core, floating-point, exponential

1. Introduction

Nowadays, operations are carried out on the hardware due to reasons such as decreasing hardware costs, making it easier to design hardware and increasing hardware capabilities [1]. In this sense, logarithmic and exponential numbers are frequently used, especially in mathematical formulas [2]. Exponential numbers, which are frequently used in solving engineering problems, have a very complex structure in digital design, although they can easily produce results in today's computers. Especially the mathematical operations performed with floating point numbers tire digital systems quite a lot. For this reason, besides ALU, there is also an FPU unit for processing decimal numbers in processors. Floating-point numbers were first adopted as the industry standard in 1985 then they were revised in 2008 and standardized as IEEE 754-2008 [3]. Thus, until today, due to its strong representation ability, it has been widely preferred in computer systems with 32-bit length single precision and 64-bit length double precision [4]. There are two possibilities for decimal numbers especially for those using FPGA technology. One of them is floating point numbers while the other are fixed numbers. When the literature is examined in detail, we witness that both of them are widely used in many studies [5, 6, 7].

Since FPGA technology allows for hardware-based and raw design, there are currently no arithmetic functions found in many software-based applications [8]. It is already known that the most important feature of this technology is application specific. Therefore, designers need to shape their designs according to what processes they will perform [9]. However, some common applications used by everyone are converted to IP Core and made available to designers.

*Corresponding author: Address: Department of Computer Technologies and Programming Sakarya University of Applied Sciences, 54187, Sakarya TURKEY. E-mail address: dereli@subu.edu.tr, Phone: +902646160492

Therefore, instead of making a new design, the designer takes the IP Core and integrates it into its design [10]. In this sense, when we look at the literature, an extremely rich design library is encountered. So much so that there are situations in which the same designs are created with different methods. For example, Kachhwa and Route developed a square root algorithm using the Vedic Mathematical method and compared this algorithm with the square root algorithm developed by the Newton-Raphson method. In these studies, they obtained 16-bit floating point number output against 24-bit floating point input value. Thus, at the end of the study, a complex circuit was simplified, occupying less space and became simpler. [11]. Zhou and Hu scaled the similar square root operation to a value between 0-1 after obtaining a 16-bit integer output value against 16-bit integer input. So, thanks to the study, the square root operation could be calculated more simply by shifting, adding and subtracting operations. [12]. Guardia and Boemo were able to realize the cube root of floating point numbers in IEEE 754-2008 standard with Newton-Raphson method in only two iterations in their study [13]. Therefore, operations that are common for a software-based programming language and can be performed easily cannot be done directly in FPGA. This is the most important disadvantage of using FPGA and unfortunately this makes the technology very difficult to learn [14].

In this study, a design has been carried out in which the base and power numbers are floating point numbers in the IEEE 754-2008 standard. Therefore, both input values and output values are 32-bit long.

2. Materials and Method

In this study, an FPGA-based hardware design that performs exponential function calculation is presented. This design, in which the input and output values used as numerical data are 32-bit floating point numbers, performs the operation mathematically expressed in Equation 1.

$$F = a^b \qquad \text{(Eq. 1)}$$

In the equation "F" is the output value, "a" and "b" are the input value. "a" is the base of the exponential function and "b" is the power. Single-precision floating-point numbers that support the IEEE 754-2008 standard are used as the number format in the design.

2.1. Floating Point Standard

IEEE 754 based floating-point numbers are widely used in today's computer systems due to their representation power. These numbers, which were first standardized in 1985, were finalized with a small revision in 2008. They have 32-bit length single and 64-bit length double precision and consist of three parts: Sign, Exponent and Fraction [15].

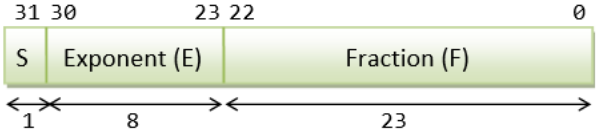

Figure 1. 32-bit single precision floating point number

Figure 1 shows a 32-bit single precision floating point number, its divisions, and the bit lengths of the segments. The most valuable bit (MSB) is called the sign bit, the next 8-bit exponent and the last remaining 23-bit is called the fraction part [16].

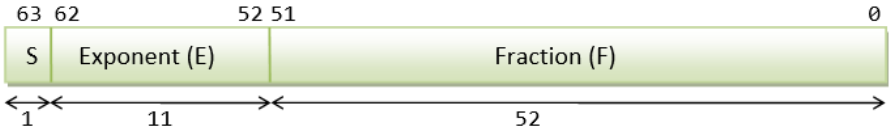

Figure 2. 64-bit double precision floating point number

Figure 2 shows a 64-bit single precision floating point number, its divisions, and the bit lengths of the segments. The most valuable bit (MSB) is called the sign bit, the next 11-bit exponent and the last remaining 52-bit is called the fraction part [16].

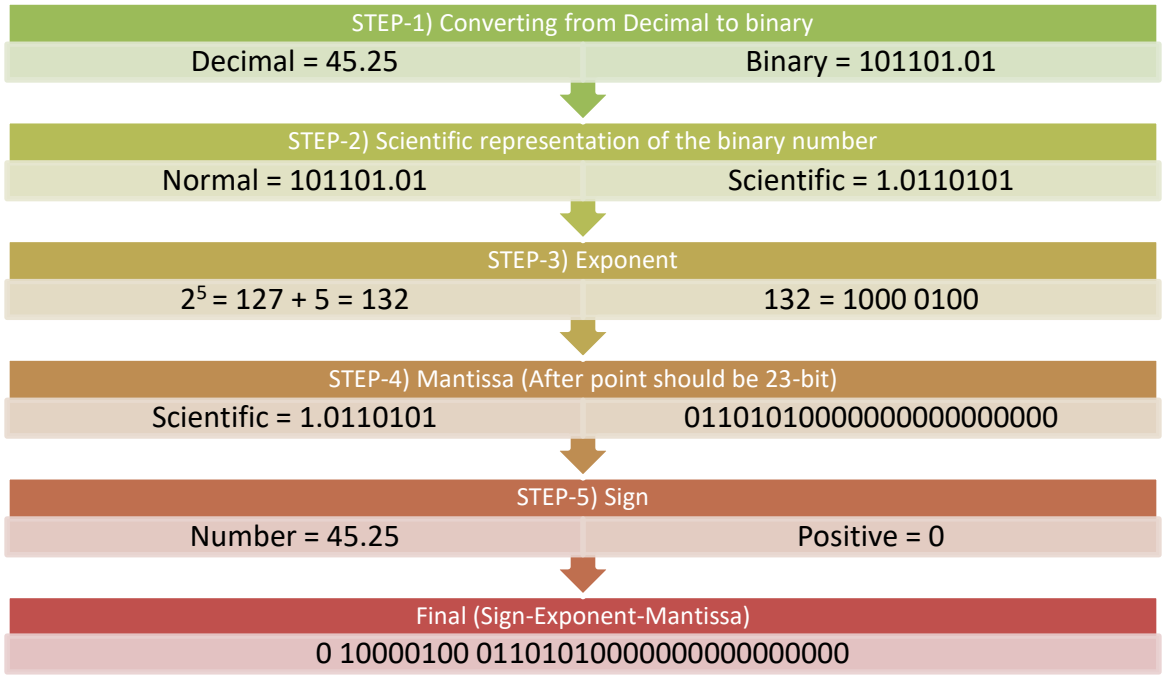

Figure 3. Step by step conversion of a real number to floating point number

Figure 3 shows the stages of the conversion of a real number to a floating point number based on an example number. The conversion process takes place in five steps. The real number is binary represented by 8-bit, while floating point is represented by 32-bit. Table 1 shows the portions and bit lengths of both single and double precision floating point numbers.

Table 1. Single and double precision floating point sections

TYPES	SIGN	EXPONENT	MANTISSA	BIAS
Single Precision	1 (31 st bit)	8 (30 - 23)	23 (22 - 0)	127
Double Precision	1 (63 rd bit)	11 (62 - 52)	52 (51 - 0)	1023

2.2. FPGA (Field Programming Gate Array) Technology

In the 1990s, the electronics world met with a new hardware technology. This was such a technology that it had both very fast processing capability and rapid prototyping on a product basis. With this technology, called FPGA, designs that can be directly reconfigured in the field could be realized [17]. However, in terms of usage, FPGAs stand out as an empty device and they have to realize all the designs they will use in every designer application. For example, ready-made functions such as mathematical functions, analog-digital conversions found in many embedded devices (raspberry, PIC, Arm) are not available in FPGAs. This is one of the major disadvantages of these devices and a major obstacle for first time users [18].

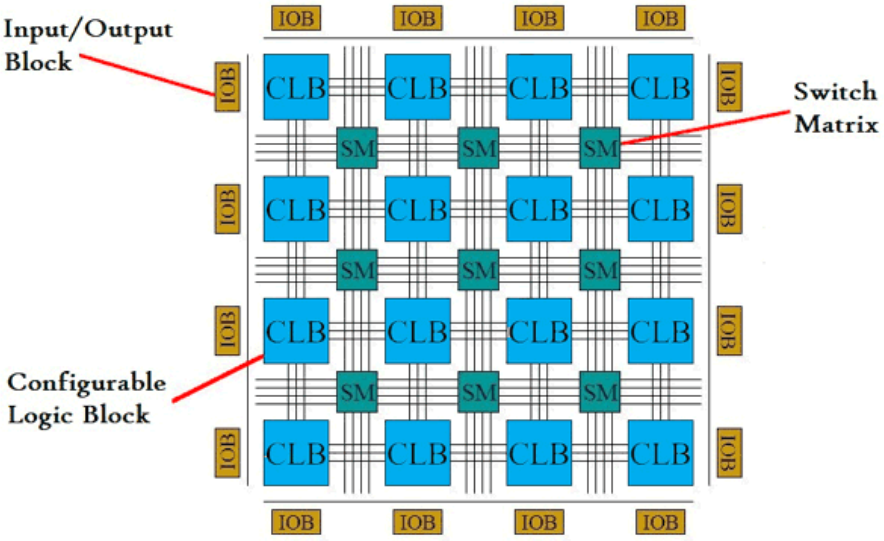

Figure 4. Architecture of FPGA

Figure 4 shows the structure of a basic FPGA and their interconnection. As it seems, an FPGA consists of three important units: CLB, IOB, SM.CLBs are the most basic unit and the designs performed are found here. It is named as configurable logical block. As can be seen from Figure 4, the more CLBs there are in an FPGA, the larger the circuits can be accommodated [19]. Already, it is the CLBs that make FPGAs stand out from other embedded systems, enabling the transactions to be hardware-based and ultra-fast.

Although IOBs (Input/output Block) vary from application to application, they can also be used as output pins as input pins. These pins each of which is 1-bit, come together to form data groups in the form of 4, 8, 16, 32 bits [20].

SMs actually carry out the function of managing and routing the paths that provide the connection between input / output pins and logical blocks (CLB). Because, if a design is too large, it is placed in more than one logical block (CLB) and the communication of these logical blocks with each other is completely carried out by SMs [21].

Table 2 contains the comparison of FPGA and other devices used in some studies in the

literature. Even from these studies, it can be seen that FPGAs are actually ideal for performance and real-time systems.

Table 2. Comparison of some literature

STUDY	YEAR	PROCESS	FPGA	OTHER DEVICE	SPEED
[22]	1998	(512x512) Image Processing	1.31 ms	DSP - 42 ms	42x
[23]	2008	(640x480) Optical Flow Algorithm	320 µs	GPU - 3.85 ms	12x
[24]	2009	UAV Real Time Path Planning – Genetic Algorithm (selection and crossover)	8.85 µs	Computer – 94 ms	10.000x
[25]	2017	K-Nearest Neighbour Algorithm	0.346 W	GPU GTX960 - 30 W	87x
[26]	2016	Jacobi algorithm (N=256 – row vector)	44.58 ms	CPU - 1326.88 ms	30x
[27]	2016	Robot manipulator inverse kinematics	0.101 µs	GPU - 32.384 µs	320x
[28]	2019	Large-Scale Computing System (Data Center)	Power Reduction 89%	Power Reduction GPU – 8%	11x
[29]	2018	Particle Swarm Optimization	0.48 ms	CPU - 2.2 ms	4.5x

2.3. Mathematical Expression of Design

The mathematical model of the operation performed within the scope of this study is given in Equation 2. This equation is also the formula for exponential computation in mathematics. So what is actually meant to be done is the formula given in Equation 1. Equation 2 is the mathematical expression of this formula.

$$F = e^{b\,(\ln a)} \qquad \text{(Eq. 2)}$$

2.4. Hardware Design of the Exponent Computation

Xilinx Nexys 4 DDR device was taken as a base in the digital design realized within the scope of this study and VHDL hardware description language was used. As seen in Equation 2, there are "Logarithmic" and "Exponential" functions in the formula. For these functions and multiplication operation, IP Cores in the Xilinx library are used. Since first logarithm, then multiplication and finally exponential computation must be performed respectively, finite state machines are used in the design. The algorithm used in the design is shown in Figure 5.

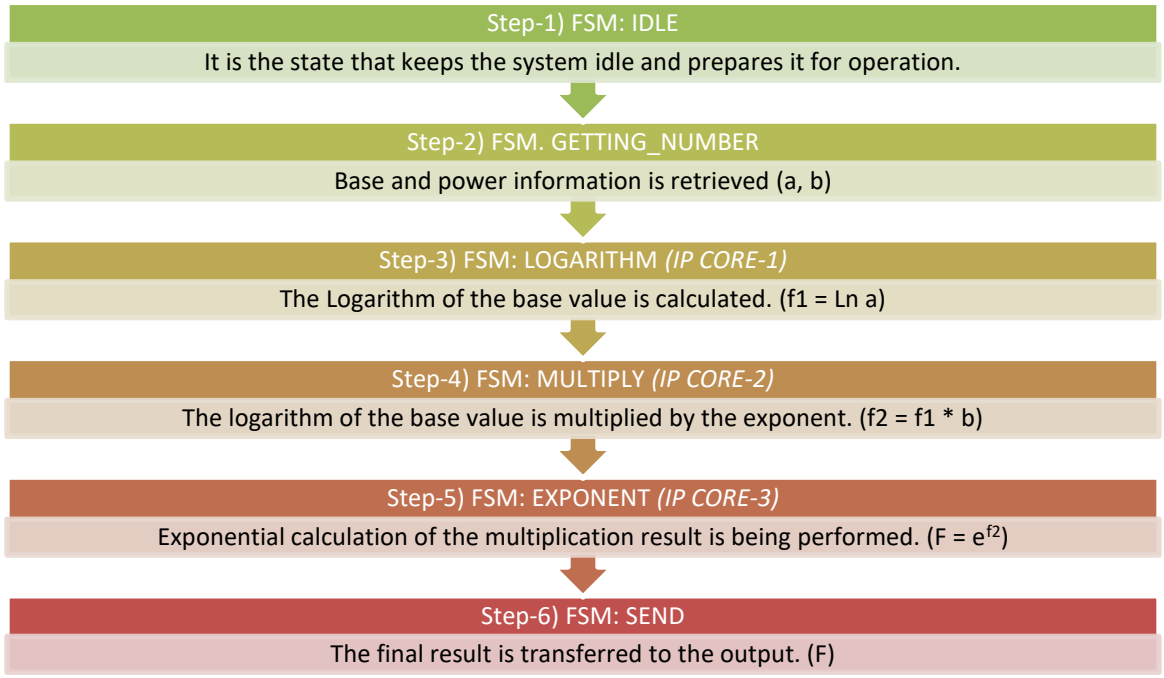

Figure 5. Step by step design algorithm

3. Results

Operations are performed with 32-bit floating point numbers and six finite state machines in total. The decimal part of real numbers is infinite in the real world. However, this situation is kept at a certain number in floating point numbers. This situation is clearly seen in the simulation results.

Table 3. Input and output values (CPU)

	DECIMAL	HEX	BIN
INPUT (base)	4.2	40866666	0 10000001 00001100110011001100110
INPUT (pow)	2.156	4009fbe7	0 10000000 00010011111101111100111
OUTPUT (Result)	22.06609	41b0875a	0 10000011 01100001000011101011010

Table 3 and Table 4 show the computer and FPGA representations of the numbers used as examples. It seems clear that the 32-bit constraint in representation affects the value of the numbers used in the FPGA, albeit very little.

Table 4. Input and output values (FPGA)

	DECIMAL	HEX	BIN
INPUT (base)	4.19999	40866666	0 10000001 00001100110011001100110
INPUT (pow)	2.15599	4009fbe7	0 10000000 00010011111101111100111
OUTPUT (Result)	22.06608	41b0875a	0 10000011 01100001000011101011010

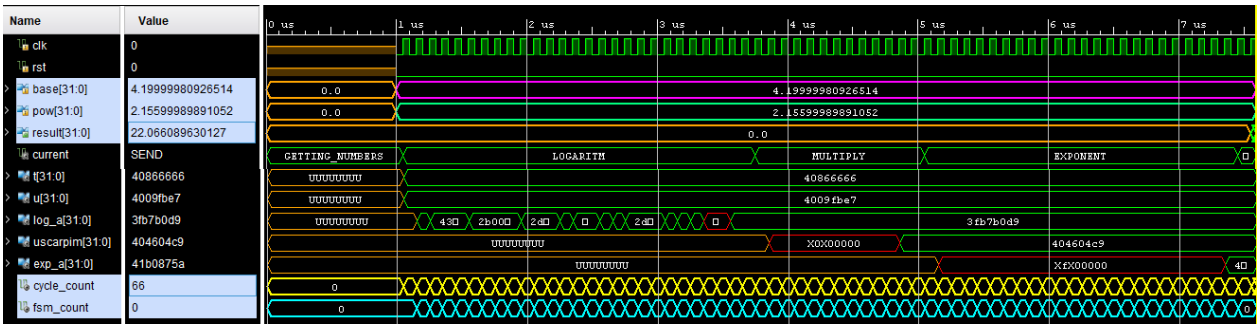

Figure 6. Simulation results of design

Figure 6 shows the simulation results performed with the sample numbers given in Table 3. After six finite state machines, results are obtained in 66 clock pulses (the signal is cycle_count in figure). The number of clock pulses required for each finite state to occur is shown in Table 5.

Table 5. FSM cycle count in design

ORDER	FINITE STATE MACHINE	COUNT
1	IDLE	0
2	GETTING_NUMBER	2
3	LOGARITHM	26
4	MULTIPLY	12
5	EXPONENT	23
6	SEND	3
TOTAL	-	66

One of the most important elements of FPGA-based digital circuits is circuit cost, that is, the dimensions of the circuit. Since the design is based on floating point numbers, three different IP Cores are used.

Table 6. Cost of design

IP CORE	LUT	FF	DSP
LOGARITHM	781	1200	4
MULTIPLIER	159	308	2
EXPONENTIAL	882	726	1
TOTAL (Design)	1822	2234	7
TOTAL (FPGA Nexyx DDR 4)	95100	126800	240
PERCENTAGE	2%	1.8%	3%

Conclusions

In this study, a design that performs exponential calculation of floating point numbers with IEEE 754 standard has been realized. Since the operations had to be performed one after the other, it was concluded with finite state machines sequentially. In total, the output value was obtained at the end of 66 cycles with six finite state machines. Since the floating-point numbers used are 32-bit single-precision, there are differences in the numbers by one tenth or one hundred thousandth. The most important advantage of the design is that it reaches results in a short time, the circuit cost is low, and it uses floating point numbers in both the base and the power. In addition, it can be used easily as a structural sub-circuit in any design.

References

[1] Kösten MM, Efe MÖ. Implementation of Discrete Time Sliding Mode Control with Floating Point Arithmetic on an FPGA. Otomatik Kontrol Ulusal Toplantısı 2015.

[2] Koyuncu İ, Çetin Ö, Katırcıoğlu F, Tuna M. Edge dedection application with FPGA based Sobel operator. 23nd IEEE Signal Processing and Communications Applications Conference (SIU) 2015; 1829-1832.

[3] Çavuşlu MA, Karakuzu C, Şahin S, Karakaya F. Yapay sinir ağı eğitiminin IEEE 754 kayan noktalı sayı formatı ile FPGA tabanlı gerçeklenmesi. İstanbul: İstanbul Teknik Üniversitesi (GOMSİS) 2008.

[4] Kamm L, Willemson J. Secure floating point arithmetic and private satellite collision analysis. Int. J. Inf. Secur. 2015; 14:531-548.

[5] Higham NJ, Pranesh S. Simulating low precision floating-point arithmetic. SIAM Journal on Scientific Computing 2019; 41:585-602.

[6] Brain M, Tinelli C, Rümmer P, Wahl T. An automatable formal semantics for IEEE-754 floating-point arithmetic. IEEE 22nd Symposium on Computer Arithmetic 2015; 160-167.

[7] Catrina O. Round-efficient protocols for secure multiparty fixed-point arithmetic. International Conference on Communications (COMM) 2018; 431-436.

[8] Nane R, Sima VM, Pilato C, Choi J, Fort B, Canis A, Anderson J. A survey and evaluation of FPGA high-level synthesis tools. EEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2015; 35:1591-1604.

[9] Zhang C, Prasanna V. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays 2017.

[10] Tolba MF, Fouda ME, Hezayyin HG, Madian AH, Radwan AG. Memristor FPGA IP core implementation for analog and digital applications. IEEE Transactions on Circuits and Systems II: Express Briefs 2018; 66:1381-1385.

[11] Kachhwal P, Rout BC. Novel Square Root Algorithm and its FPGA Implementation. International Conference on Signal Propagation and Computer Technology (ICSPCT) 2014:158-162.

[12] Zhou Z, Hu J. A Novel Square Root Algorithm and its FPGA Simulation. Journal of Physics: Conference Series 2019.

[13] Guardia CM, Boemo E. FPGA implementation of a binary32 floating point cube root. IEEE Southern Conference on Programmable Logic (SPL) 2014.

[14] Wang T, Wang C, Zhou X, Chen H. A Survey of FPGA Based Deep Learning Accelerators: Challenges and Opportunities. Distributed, Parallel, and Cluster Computing 2018:1-10.

[15] Malík P. High throughput floating point exponential function implemented in FPGA. IEEE Computer Society Annual Symposium on VLSI 2015:97-100.

[16] Rao YS, Kamaraju M, Ramanjaneyulu DVS. An FPGA implementation of high speed and area efficient double-precision floating point multiplier using Urdhva Tiryagbhyam technique. IEEE Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG) 2015.

[17] Perera DG. Analysis of FPGA-Based Reconfiguration Methods for Mobile and Embedded Applications. 12th FPGA world Conference 2015:15-20.

[18] Tolba MF, AbdelAty AM, Soliman NS, Said LA, Madian AH, Azar AT, Radwan AG. FPGA implementation of two fractional order chaotic systems. AEU-International Journal of Electronics and Communications 2017; 78:162-172.

[19] Abdelkrim H, Othman SB, Saoud SB. Reconfigurable SoC FPGA based: Overview and trends. IEEE International Conference on Advanced Systems and Electric Technologies (IC_ASET) 2017.

[20] Dereli S. FPGA ile Gömülü Sistemler ve Sayısal Devre Tasarımı. 1st ed. Ankara: NOBEL Akademik Yayıncılık; 2020.

[21] Dereli S. Yüksek Hızlı FPGA ile Yeni Bir LFSR Tabanlı 32-Bit Kayan Noktalı Rastgele Sayı Üreteci Tasarımı. International Journal of Advances in Engineering and Pure Sciences 2020; 32:219-228.

[22] Bilsby DCM, Walke RL, Smith RWM. Comparison of a programmable DSP and a FPGA for real-time multiscale convolution. IEE Colloquium on High Performance Architectures for Real-Time Image Processing, 1998.

[23] Chase J, Nelson B, Bodily J, Wei Z, Lee DJ. Real-time optical flow calculations on FPGA and GPU architectures: a comparison study. 16th IEEE International Symposium on Field-Programmable Custom Computing Machines 2008:173-182.

[24] Allaire FC, Tarbouchi M, Labonté G, Fusina G. FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning. Journal of Intelligent and Robotic Systems 2008; 54:495–510.

[25] Muslim FB, Ma L, Roozmeh M, Lavagno L. Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis. IEEE Access 2017; 5:2747-2762.

[26] Torun MU, Yilmaz O, Akansu AN. FPGA, GPU, and CPU implementations of Jacobi algorithm for eigenanalysis. Journal of Parallel and Distributed Computing 2016; 96:172-180.

[27] Rizvi STH, Cabodi G, Patti D, Gulzar MM. Comparison of GPGPU based robotic manipulator with other embedded controllers. 2016 International Conference on Development and Application Systems (DAS) 2016:10-15.

[28] Gizopoulos D, Papadimitriou G, Chatzidimitriou A, Reddi VJ, Salami B, Unsal OS, Leng J. Modern Hardware Margins: CPUs, GPUs, FPGAs Recent System-Level Studies. IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS) 2019:129-134.

[29] Lee H, Kim K, Kwon Y, Hong E. Real-time particle swarm optimization on FPGA for the optimal message-chain structure. Electronics 2018;7:274.