Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 2321 – 2330

doi:10.3906/elk-1705-394

Design of area-efficient IIR filter using FPPE

Nallathambi RAMYARANI[1,*], **Veerana SUBBIAH**[2], **Prabhakaran DEEPA**[3]
[1]Department of Electrical and Electronics Engineering, Sri Krishna College of Engineering and Technology, Coimbatore, India
[2]Department of Electrical and Electronics Engineering, PSG College of Technology, Coimbatore, India
[3]Department of Electronics and Communication, Government College of Technology, Coimbatore, India

Received: 30.05.2017 • **Accepted/Published Online:** 29.11.2018 • **Final Version:** 15.05.2019

Abstract: Floating point arithmetic circuits provide wide dynamic range and high precision, and they are widely used in scientific computing and signal processing applications, but the complexity increases in hardware implementations of floating point units. In VLSI design architecture, many applications suffer in size of the components used in logical operations. The aim of reducing architecture is to gain reduction in power loss and also in area, but the reduction in size of the components leads to an increase in delay and memory. Hence, to overcome these limitations and to optimize the area, a novel design of floating point processing element (FPPE) architecture is proposed in this work with a smaller number of logical components and registers. A partially folded arithmetic function architecture is modeled for the design of an infinite impulse response (IIR) filter using FPPE and implemented on a field programmable gate array (FPGA) with efficient area. FPGAs are widely used in the implementation of floating point computing modules due to the increase in gate density and embedded arithmetic cores. Synthesis results prove that the proposed design of the IIR filter provides efficient area compared with existing works. The modules are designed in Verilog and implemented on Xilinx FPGAs.

Key words: Field programmable gate arrays, floating point arithmetic, floating point processing element, IEEE 754-2008 standard, infinite impulse response filters

1. Introduction

Floating point arithmetic and logic units are a part of computer systems. The requirements of floating point arithmetic have become very intense due to its dynamic range representation of real numbers compared to fixed point values. These floating point numbers are represented as per the IEEE 754-2008 standard [1]. This standard represents the single precision (32 bits) and double precision floating point formats (64 bits), but the complexity of these arithmetic circuits increases on hardware implementation due to increase in bit width representation [2]. Application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) are some of the current technologies used for hardware implementation. Nowadays, FPGAs are used for the implementation of floating point arithmetic units because of their increased integration density compared with ASICs and provide increased computing speed [3–6]. Most scientific computing applications, engineering fields like image processing, cryptography, network security, and signal processing rely on floating point arithmetic algorithms.

Digital filters are the basic building blocks of digital signal processing systems that are used to remove

*Correspondence: ramyaeee.skcet@gmail.com

the noises in systems. Recursive and nonrecursive filters are the two major types of filter designs. Recursive or infinite impulse response (IIR) filters depend on the past output results and have an infinite number of coefficients. These filters have feedback and nonlinear phase characteristics. Nonrecursive or finite impulse response (FIR) filters depend on only the present input sequences. FIR filters have linear phase characteristics but require more computations and hence the design of FIR filters in digital signal processors consumes more memory [7].

Nowadays, digital signal processing designs are widely embedded in FPGA devices by utilizing the inbuilt digital signal processing (DSP) blocks and embedded multipliers for transform computations and to determine the filter coefficients, but the hardware implementation of these DSP modules on FPGAs consumes more area due to the multiplication operations in transforms and filters. Hence, in this work, floating point processing element (FPPE) architecture is designed to implement IIR filters on FPGAs with reduced number of registers and logical operations.

The paper is described as follows: Section 2 describes a literature survey of floating point arithmetic units on FPGAs. Section 3 presents the design of FPPE architecture. Section 4 details the design of FPPE-based architecture for the design of area-efficient IIR filters. Section 5 presents the simulation and synthesis results with discussions. Finally, Section 6 concludes the work with future scope.

2. Literature survey

Many researchers proposed methods for the design of floating point arithmetic units and implementation of IIR filters on FPGAs. These modules were optimized for area, speed, and power in hardware implementation on FPGAs. Some of the works are presented here.

Scrofano et al. evaluated balanced and unbalanced binary tree arithmetic expressions using pipelined floating point cores. The implementation of results on FPGAs was compared in terms of area, speed, and latency [8].

Kaur et al. proposed the design and implementation of IIR filters on FPGAs. This work was compared in terms of delay and frequency for pipelined and nonpipelined architectures. The synthesis results proved that the pipelined architectures produced increased speed of computations compared with nonpipelined architectures [9].

Mehra and Thakur proposed the design and implementation of digital IIR filters on FPGAs. These designs utilized the multiply and accumulate (MAC) unit for computing the filter coefficients. Parallel pipelining was also utilized in the design to provide efficient area [10].

Jeong-Hwan et al. realized the design of digital IIR and FIR filters for the removal of noise in electrocardiogram (ECG) signals monitored for patients. The design was implemented in a smart phone with Android operating system by designing the graphical user interface in MATLAB [11].

Chong and Parameswaran implemented multimode embedded floating point arithmetic units on FPGAs. The embedded floating point units included the design of the floating point adder and multiplier to perform double precision operations or two single precision operations simultaneously. Such designs provided the performance and area benefits for the implementation of single precision and double precision floating point arithmetic units on FPGAs [12].

Tsai et al. proposed the hybrid Taguchi genetic algorithm for the design of digital IIR filters. The experimental results proved that the proposed algorithm can be used to design low-pass, high-pass, and band-

pass filters with better performance characteristics compared to the existing genetic algorithm-based IIR filter designs [13].

Vazquez and Bruguera proposed an algorithm-based architecture for computing power and extraction of roots using fixed point and floating point exponents. The algorithm was designed using an optimized iterative sequence of operations including reciprocal, left-to-right carry-free multiplication, and high radix online exponential function. Logarithm and exponential functions are part of the sequence of the above operations [14].

Jaiswal et al. presented two architectures for floating point adders in multimode configuration. The quadruple double precision architecture worked in dual mode, which operated either quadruple precision or two double precision in parallel. The second quadruple double precision quadruple single precision architecture worked in tri mode. It computes either one quadruple precision, two double precision, or four single precision operations in parallel. The proposed designs provided better results in terms of area and delay [15].

Wijayaratna et al. proposed an IIR filter design used for the preprocessing stage of beam enhanced aperture arrays [16]. Basiri and Mahammad proposed VLSI architecture for a folded sixth-order IIR filter used in DSP applications [17]. The design produced 58.4% reduction in energy compared with MAC architecture.

Joldes et al. presented algorithms for precision extension using floating point expansions. New algorithms were introduced for the normalization, division, and square root of floating point expansions. The reciprocal and square root operations were designed based on the Newton–Raphson iteration method. Floating point expansions provided highly optimized hardware implementations. Accurate results were provided using error analysis procedure [18].

Akkas and Schulte proposed the design of a quadruple precision floating point multiplier. The design supported the parallel operation of the evaluation of two double precision floating point multiplications. The results provided the latency of three clock cycles for quadruple precision multiplications and latency of only two clock cycles for two double precision multiplications in parallel [19].

Li et al. proposed the design of a fully pipelined single precision floating point arithmetic unit. The design was implemented in 3 pipeline stages. Along with four basic arithmetic operations, square root calculation was also performed and implemented on the FPGA [20].

Fu et al. compared hardware designs on FPGAs using floating point and logarithmic number systems and determined the choice of format in terms of area and speed. The proposed designs were compared with Monte Carlo radiative heat transfer simulation and showed better accuracy [21].

3. Floating point processing element architecture

FPPE is an important architecture for the computation of binary arithmetic operations like addition, subtraction, and multiplication in DSP applications [22]. To design the FPPE model, several techniques are used for estimating mantissa value and an exponential term with minimum usage of logical components. There are some limitations of those techniques that lead to an increase in delay and memory. Hence, to overcome these limitations of the VLSI architecture in DSP processing, the FPPE architecture is designed with the blocks of shift adders, multipliers, and dividers as shown in the block diagram of Figure 1.

The double precision value (64 bits) is considered as the input register. This register consists of two operands, A and B, of 32 bit values. The sign bit of the two operands is computed using the XOR operation from the 63rd bit and 31st bit. The bits from [62:55] and [30:23] are considered for exponent computation. The

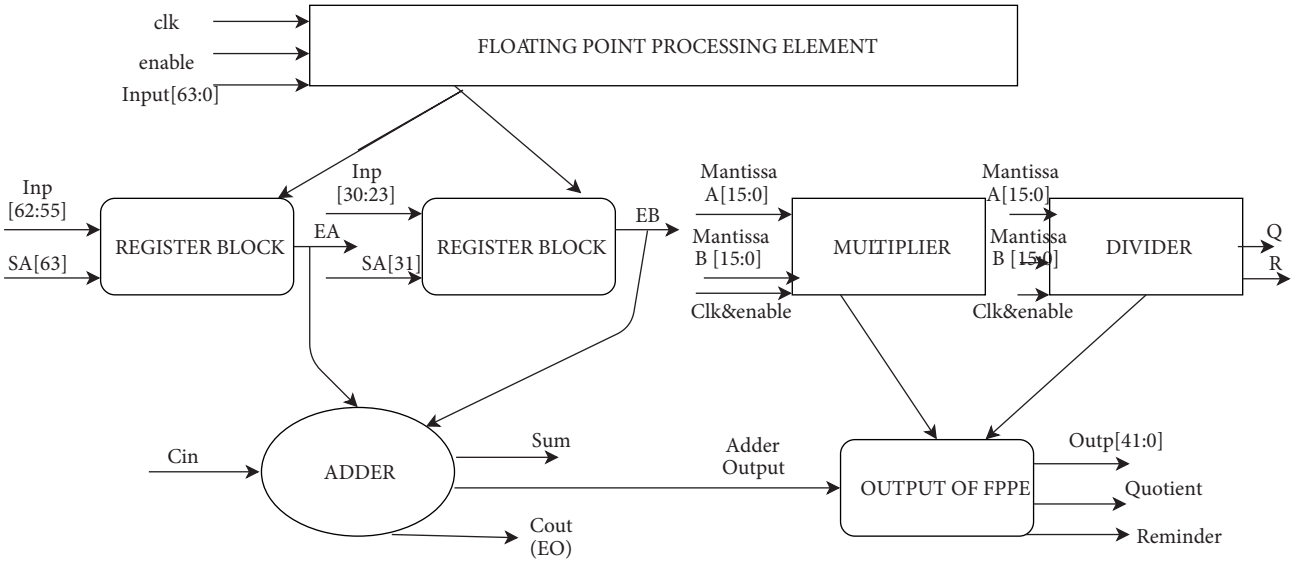

Figure 1. Block diagram of FPPE.

exponents of the two operands are computed by the adder block, resulting in output of 8-bit size. The first 16 bits from the LSB of the mantissa of the two operands is considered for multiplication and division. The multiplication of the two operand mantissas (16 bits each) produces an output of 32-bit size on repeated addition. The multiplication of two operands has been computed in 4 stages. In the first stage, the least significant bit group of the two operands is multiplied. The first input most significant bit group and second input least significant groups are multiplied in the second stage. In the third stage, the first input least significant bit group and second input most significant bit group are multiplied. The two inputs' most significant bit groups are multiplied in the fourth stage. The fourth stage's output is shifted and added to the third stage's output. The second stage's output is shifted and added to the first stage output. The resultant output of these two stages is shifted and added to produce the output. As per the floating point standard of IEEE single precision, the obtained output is normalized. The normalized output of the mantissa is 33 bits. The final resultant output of the shift adder and multiplier is 42 bits, consisting of one sign bit, 8 bits of exponent, and 33 bits of mantissa. The division of two operand mantissas produces a quotient and remainder of 16-bit size. The dividend and divisor input values are assigned to the registers. The 2's complement value of the divisor is stored in the register. In the adder block, the 16-bit values of the two operands are added. The resultant sum is added to the divisor value consecutively until the sum of the output and carry is generated. If the resultant sum is greater than or equal to the 16-bit divisor, the count value is incremented and moved to the R4 register that represents the quotient value.

4. IIR filter

Partially folded arithmetic function-based architecture for an ALU processing unit with FPPE is proposed in the application of the IIR filter as shown in Figure 2.

The pipelined register design (PRD) algorithm is used for the allocation of registers. This algorithm reduces the size of the DSP unit with a minimum number of logical components and registers. In this DSP unit, a multiplier-free 64-bit second-order IIR filter is designed by using the adder and shifter followed by accumulator

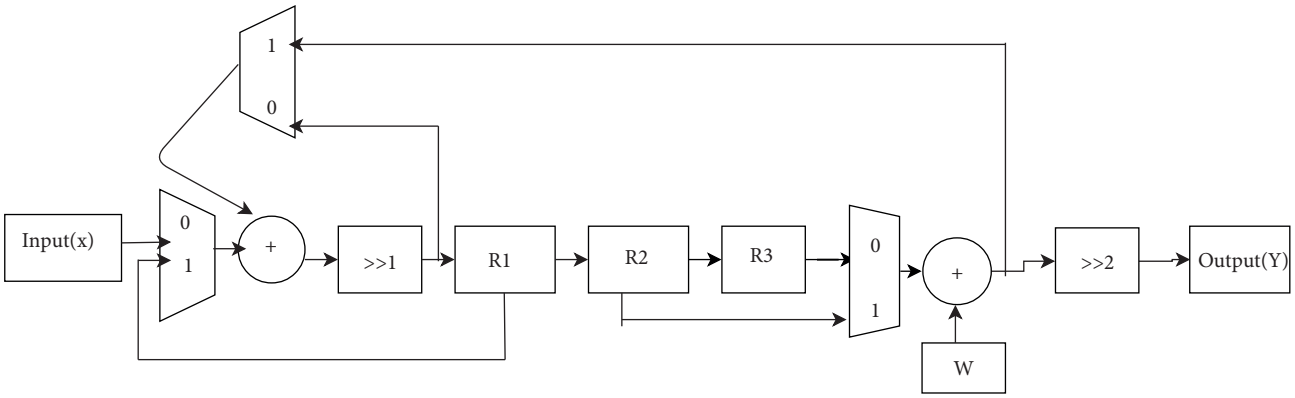

Figure 2. Partially folded arithmetic function architecture.

to proceed to the FPPE. In the design of the IIR filter, 3 registers are used for the temporary allocation of data. The delay is estimated in each data shift to the register. If the delay is less than predicted, registers are allocated. If the delay is more than predicted, the data are shifted and then sent to the pipelined register. The input is processed in the folded block that computes the filter coefficients in 4 pipelined stages. Eight forward and eight backward filter blocks are designed using the PRD algorithm.

4.1. Pipelined register design algorithm

The input values are stored in the bock RAM memory of the FPGA using a block memory generator. These values are accessed as .Coe files in the Xilinx FPGA. A lifetime chart is used for the computation of minimum registers required to implement the design in the hardware [23]. A data variable is alive from the time it is produced through the time it is consumed. Once consumed, the alive variable becomes dead. A variable requires one register until the variable remains in the alive state. Horizontal lines represent the clock cycle and vertical lines represent the lifetime of the alive variables. Eleven nodes of the IIR filter design are computed in each clock cycle as shown in Figure 3. Based on the lifetime chart, registers *R1*, *R2*, *R3* are used for the allocation of temporary results as shown in Table 1. The registers are filled with intermediate data in a pipelined manner during shifting and adding operations of the filter design as per the following steps.

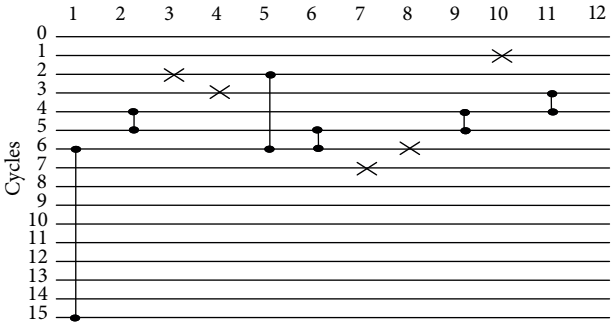

Figure 3. Lifetime chart.

Step 1: Initialize registers, $RS = \{R_1, R_2, \ldots, R_N\}$

Where $N = 3$; i.e. 3 registers are used to perform folded transform.

Step 2: for $i = 1$ to number of stages

Table 1. Register allocation.

Clock	Filter stage				Registers		
					R1	R2	R3
0	2, 0	2, 2	2, 1	2, 3			
1	2, 4	2, 6	2, 5	2, 7	2, 2	2, 0	2, 3
2	2, 8	2, 10	2, 9	2, 11	2, 6	2, 2	2, 7
3	2, 12	2, 14	2, 13	2, 15	2, 10	2, 8	2, 11
4					2, 14	2, 10	2, 15

Step 3: $t = 0$;

Step 4: for $j = 1$ to number of lines

Step 5: If Delay (i, j) = '1'

Step 6: $t = t + 1$;

Step 7: end if;

Step 8: Estimate kernel function (s).

Step 9: If $W_{i,j} = D$

Step 10: $R(W_{i,j}) = 1$; //where 'i' and 'j' are the size of weight matrix at each stage and $k = 1, 2 \ldots N$.

Step 11: $AR = R_k$; //Allocate registers.

Step 12: end 'if'.

Step 13: end 'j' loop.

Step 14: end 'i' loop.

Initially the count value is initialized to zero. For every stage, the delay value is computed for shifting of values between the registers and the count value gets incremented. When the filter coefficients of real and imaginary parts get matched with the delay value, registers are allocated for the data bits.

This type of merging technique reduces the delay rate of a proposed model of the DSP unit with minimum number of logical components. In each condition, one register is used to produce the logical output that holds paring of blocks at each stage. The crossing of the bit sequence from input to output optimizes the delay block due to the link formation in the register.

The IIR filter is designed with clock and reset input. The input size of each filter block is considered as 8 bits in parameter order of 8. The intermediate results are stored in temporary registers. The eight forward and eight backward filter blocks are designed as shown in the block diagram of Figure 4. In each filter block, two inputs of 8-bit size are considered and manipulated in four stages.

In the first forward filter block, the first input of size [3:0] and second input of size [3:0] are given as input to the ripple carry adder. This results in an output called sum1. In stage 2, the first input of size [7:4] is ripple carry added with the second input of size [3:0] to obtain the output called sum2. In stage 3, the first input of size [3:0] and the second input of size [7:4] are input to the ripple carry adder and output sum3 is obtained. The first input of bit size [7:4] and the second input of bit size [7:4] is ripple carry added and output sum4 is obtained. In the shifter and adder block, sum2 is shifted and added with sum1 to produce the output sum5. Similarly, output sum6 is obtained by shifting sum4 and adding it with sum3. The resultant output

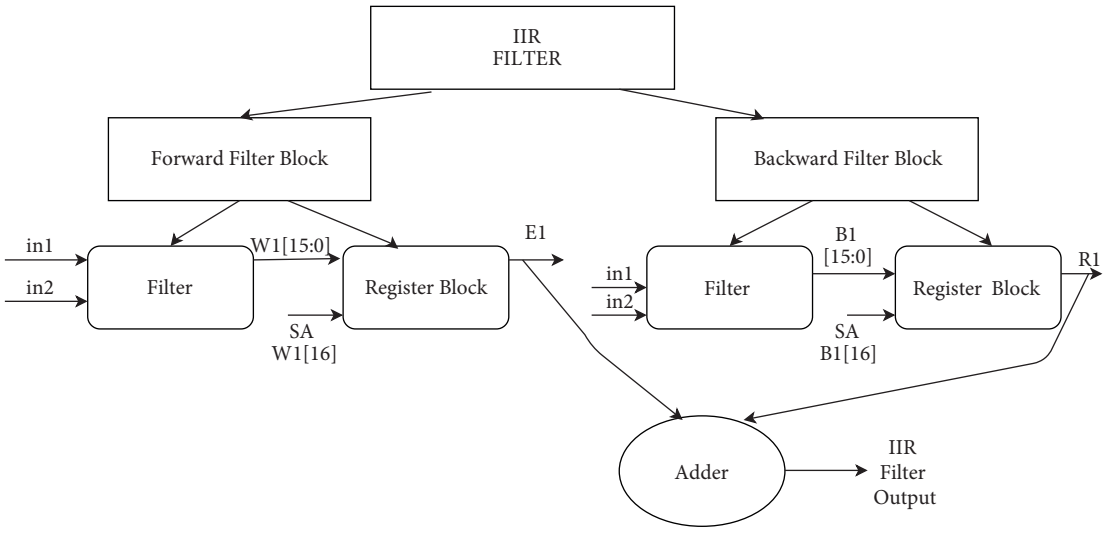

Figure 4. Block diagram of IIR filter showing forward and backward blocks.

E1 is obtained by shifting sum6 and adding it with sum5. In the same way, the remaining seven forward and backward filter blocks are computed. The forward and backward filter blocks' outputs are added to yield the final filter output.

5. Experimental results

This section describes the simulation and synthesis report of the proposed work. The design has been verified by simulating the design in the Modelsim simulator tool. The hardware utilization of the resources has been analyzed from the synthesis report implemented in the Xilinx Virtex 6 FPGA.

Figure 5 presents the simulation results of the FPPE architecture. The double precision floating point value is given as input to the architecture. The output 42 bits [41:0] were obtained by cascading multiplied outputs of a mantissa of size [32:0], output exponent of size [7:0], and output sign bit obtained by the XOR operation of the two input sign bits. The divider block output was obtained as quotient Q [15:0] and remainder R [15:0].

Partially folded arithmetic architecture with FPPE architecture was designed for the application of an IIR filter. The simulation result of the IIR filter using FPPE was obtained as shown in Figure 6. The forward filter block output was obtained in E1, E2, E4, E6, E8, E10, E12, and E14. The backward filter block output was obtained in R1, R2, R4, R6, R8, R10, R12, and R14. The addition of forward and backward filters output values yields the IIR filter output Dout [17:0]. Based on the clock and reset input, the output enable bit (Oen) has been enabled and disabled corresponding to the manipulated output.

Table 2 represent the comparison of the proposed design of FPPE with other floating point architectures implemented on the FPGA. This table proves that the proposed FPPE designed in this work provided an efficient area in terms of slices and maximum operating speed (MHz) compared with other architectures. The Xilinx auxiliary processor unit (APU) is a floating point coprocessor consisting of single precision and double precision units. The APU and Nallatech core processor designs occupied more area compared with the proposed FPPE. Also, the operating speed of the proposed FPPE is maximum compared with the Xilinx APU and Nallatech core processor architectures.

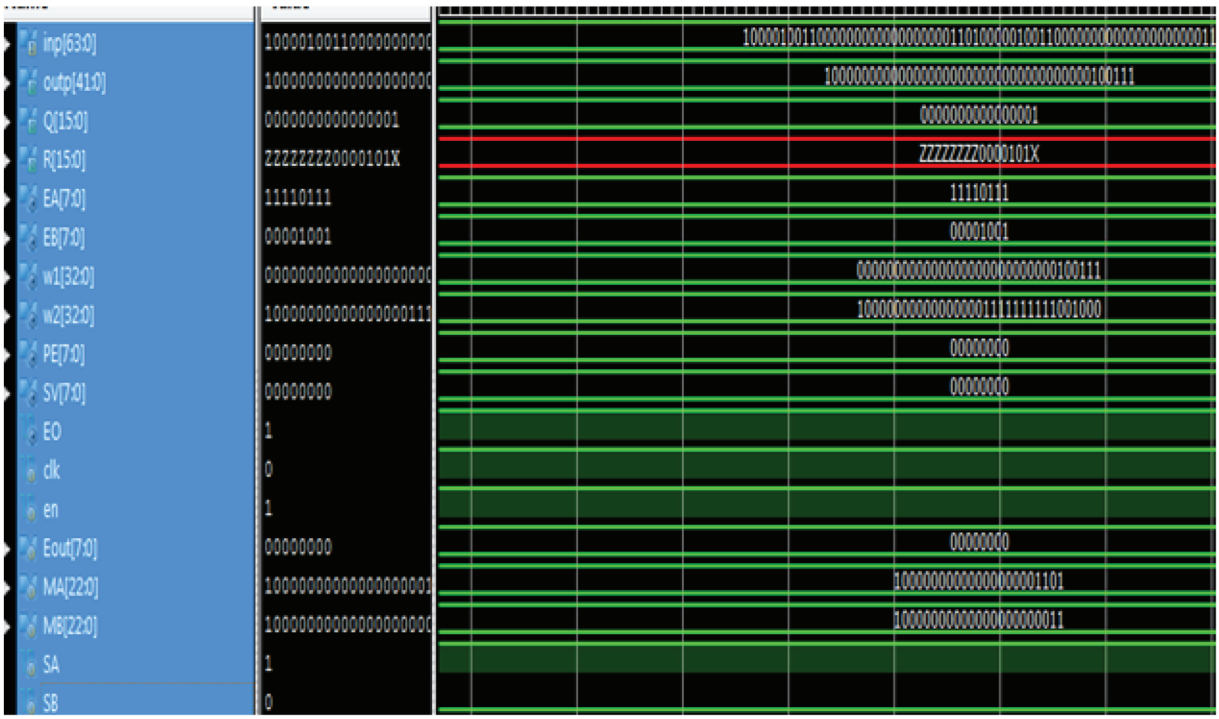

Figure 5. Simulation output of FPPE.

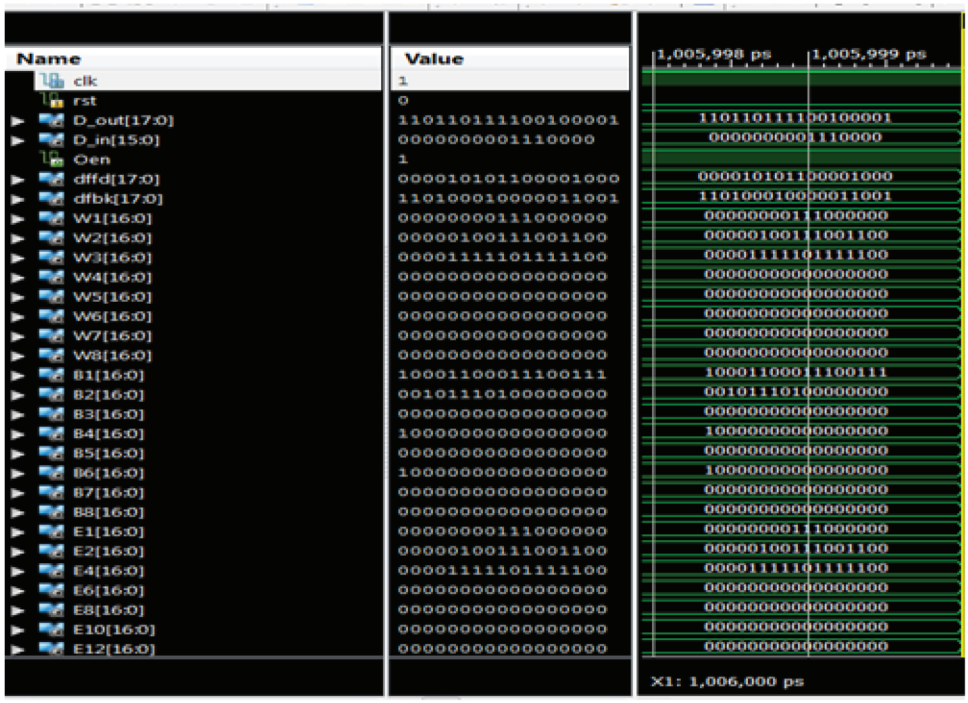

Figure 6. Output of IIR filter.

Table 3 proves that the proposed FPPE-based IIR filter design occupied fewer registers and lookup tables (LUTs) for hardware implementation on the FPGA. Also, the design provided faster computations and occupied

Table 2. Performance analysis of proposed FPPE with other floating point implementations on FPGA.

Synthesis results (Xilinx Virtex FPGA)	Proposed FPPE	Xilinx APU	Nallatech core
Area (slices)	987	2620	1600
Frequency (MHz)	658.588	200	120

a smaller number of DSP blocks of an FPGA. Hence, the proposed FPPE-based IIR filter architecture provided efficient area and speed compared with the IIR filter architectures designed for beam enhancement RF aperture arrays.

Table 3. Performance analysis of proposed FPPE IIR filter with IIR filter design of beam enhancement RF aperture arrays.

Design of second-order IIR filter Virtex-6 FPGA (n = 25)	Existing	Proposed FPPE IIR filter
Slice registers	19,204	17,283
Slice LUTs	17,866	16,079
Latency (ns)	9.8	7.6
Max. frequency (MHz)	102.041	131.579
DSP blocks	425	212

6. Conclusion and future work

This work proposed the design of floating point arithmetic units for signal processing applications. Nowadays, digital IIR filters are widely used in audio applications using floating point values and FPGA-based digital signal processing architectures have been gaining importance. Hence, in this work, the FPPE architecture was designed for the IIR filter and implemented on FPGA devices to provide efficient area. The designed FPPE provided efficient area compared to the APU and Nallatech core processor architecture. The PRD algorithm along with folding operations allowed the design to be implemented with reduced registers and logical components. The proposed architecture provided efficient area and speed compared with the digital IIR filters designed for beam enhancement RF aperture arrays.

In the future, the designed IIR filter can be utilized for the removal of noise signals from ECG and electromyography signals in the field of biomedical processing and a part of this work is under progress. Various audio input signals can be analyzed for filtering operations using IIR filters.

References

[1] IEEE. IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-2008. New York, NY, USA: IEEE, 2008.

[2] Chong YJ, Parameswaran S. Custom floating-point unit generation for embedded systems. IEEE T Comput Aid D 2009; 28: 638-650.

[3] Yu CW, Smith AM, Luk W, Leong PHW, Wilton SJE. Optimizing floating point units in hybrid FPGAs. IEEE T VLSI Syst 2012; 20: 1295-1303.

[4] Karlström P, Ehliar A, Liu D. High-performance, low-latency ?eld-programmable gate array-based ?oating-point adder and multiplier units in a Virtex 4. IET Comput Digit Tec 2008; 2: 305-313.

[5] Kuang S, Wang J, Hong H. Variable latency floating point multipliers for low power applications. IEEE T VLSI Syst 2010; 10: 1493-1497.

[6] Galal S, Horowitz M. Energy-efficient floating-point unit design. IEEE T Comput 2011; 60: 913-922.

[7] Savadi A, Yanamshetti R, Biradar S. Design and implementation of 64 bit IIR filters using Vedic multipliers. In: Elsevier 2016 Proceedings of International Conference on Computational Modeling and Security; 11–13 February 2016; Bangalore, India. pp. 790-797.

[8] Scrofano R, Zhuo L, Prasanna VK. Area efficient arithmetic expression evaluation using deeply pipelined floating point cores. IEEE T VLSI Syst 2008; 2: 167-176.

[9] Ravinder K, Ashish R, Hardev S, Jagjit M. Design and implementation of high speed IIR and FIR filter using pipelining. International Journal of Computer Theory and Engineering 2011; 3: 292-295.

[10] Rajesh M, Bharti T. Field programmable gate array based infinite impulse response filter using multipliers. International Journal of Electronics and Communication Engineering 2015; 9: 1457-1461.

[11] Jeong-Hwan K, Sang-Eun P, Jeong-Whan L, Kyeong-Seop K. Design and implementation of digital filters for mobile health care applications. International Journal of Electronics and Electrical Engineering 2014; 2: 75-79.

[12] Chong YJ, Parameswaran S. Con?gurable multimode embedded floating-point units for FPGAs. IEEE T VLSI Syst 2011; 19: 2033-2044.

[13] Jinn-Tsong T, Jyh-Horng C, Tung-Kuan L. Optimal design of digital IIR filters by using hybrid Taguchi genetic algorithm. IEEE T Ind Electron 2006; 53: 867-879.

[14] Vazquez A, Bruguera JD. Iterative algorithm and architecture for exponential, logarithm, powering and root extraction. IEEE T Comput 2013; 9: 1721-1731.

[15] Jaiswal MK, Sharat Chandra Varma B, Hayden KH, Balakrishnan M, Koli P, Cheung CC. Configurable architectures for multi-mode floating point adders. IEEE T Circuits Syst 2015; 8: 2079-2090.

[16] Wijayaratna S, Madanayake A, Wijenayake C, Bruton LT. Digital VLSI architectures for beam-enhanced RF aperture arrays. IEEE T Aero Elec Sys 2015; 51: 1996-2011.

[17] Basiri MMA, Mahammad SKN. Configurable folded IIR filter design. IEEE T Circuits Syst 2015; 12: 1144-1148.

[18] Joldes M, Marty O, Muller JM, Popescu V. Arithmetic algorithms for extended precision using floating-point expansions. IEEE T Comput 2016; 4: 1197-1210.

[19] Akkas A, Schute MJ. A quadruple precision and dual double precision floating-point multiplier. In: IEEE 2003 Proceedings of the Euromicro Symposium on Digital System Design; 1–6 September 2003; Antalya, Turkey. pp. 65-71.

[20] Li Z, Zhang X, Li G, Zhou R. Design of a fully pipelined single-precision floating point unit. In: IEEE 2007 Proceedings of 7th IEEE International Conference on Application Specific Integrated Circuits; 22–25 October 2007; Guilin, China. pp. 60-63.

[21] Fu H, Mencer O, Luk W. Comparing floating-point and logarithmic number representations for reconfigurable acceleration. In: IEEE 2006 International Conference on Field Programmable Technology; 13–15 December 2006; Bangkok, Thailand. pp. 337-340.

[22] Purohit S, Calamalasetti RS, Margala M, Vanderbauwhede WA. Design and evaluation of high-performance processing elements for reconfigurable systems. IEEE T VLSI Syst 2013; 21: 1915-1927.

[23] Keshabparhi K. VLSI Digital Signal Processing Systems. Design and Implementation. New Delhi, India: Wiley India, 2008.