FPGA-BASED ANALYSIS AND PERFORMANCE EVALUATION OF FIR FILTERS

Dr. Vidhya D S [1], Ishan Prabhu [2], Sohan Sawant [2], Melba d'souza [1], Sagar Shetkar [2]

[1] Assistant Professor, Don Bosco College of Engineering, Goa University, Goa, India
[2] Student, Don Bosco College of Engineering, Goa University, Goa, India

Corresponding Author
Dr. Vidhya D S, ds.vidhya@dbcegoa.ac.in

DOI
10.29121/shodhkosh.v5.i6.2024.5653

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

ABSTRACT

Finite impulse response (FIR) filters are widely used in digital signal processing because of their many benefits. Because these filters are naturally stable, stability issues during design and implementation are resolved. FIR filters have historically been made using ASICs or DSP processors. On the other hand, the incorporation of specialized DSP blocks and fundamental characteristics like re-configurability, reusability, cost effectiveness, and scalability have made FPGAs a preferred platform for the implementation and testing of digital signal processing systems. Optimized convolution algorithms can effectively realize FIR filters, even though they usually require a high order. The main goal of this project is to implement the FIR filter design on an FPGA. Traditional FIR filter design methods often have limitations in terms of delays, memory usage, power consumption, and overall system performance. To address these problems, this project uses Distributed Arithmetic with Offset Binary Coding (DA-OBC) technique. By enabling high-speed operation and lowering hardware resource consumption, this technique improves performance while minimizing circuit complexity. Verilog HDL was used to design the FIR filter, which was then implemented using the Xilinx ISE Design Suite 14.7. MATLAB was used to generate the filter coefficients, allowing for accurate and effective filter design.

Keywords: DSP Blocks, FPGA, FIR Filter

1. INTRODUCTION

Filters can be broadly classified into two types: analog and digital. In order to make subsequent processing more efficient, they are used to alter input signals. Among these, digital filters are more widely used due to their superior signal-to-noise ratio compared to analog filters. At each intermediate transformation stage, digital filters perform mathematical computations without making any noise. The two primary categories of digital filters are FIR (Finite Impulse Response) and IIR (Infinite Impulse Response) filters. The fact that IIR filters can only typically implement low-pass, band-pass, and high-pass responses is one of their main drawbacks. In contrast, FIR filters offer greater flexibility, allowing the realization of nearly any desired frequency response with high precision and linear phase characteristics. Most of the time, the output of FIR filters is calculated using a series of delays, multipliers, and adders.

Figure 1

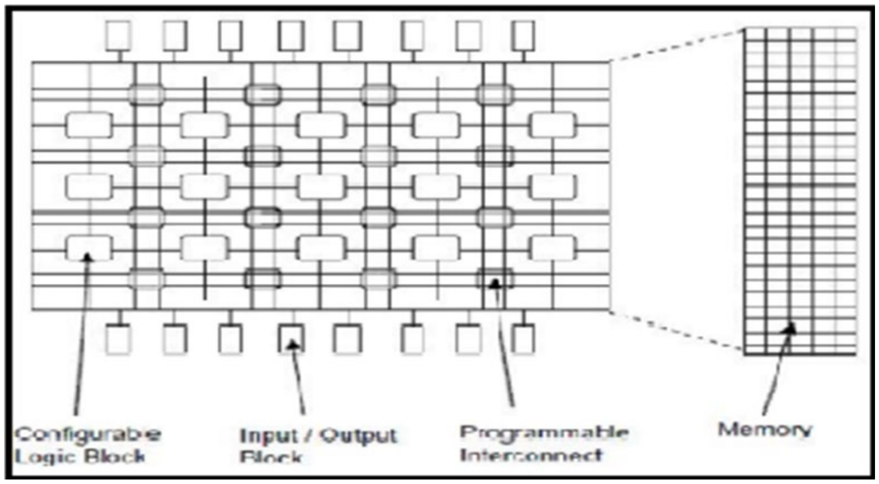


Figure 1 Architecture of FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices consisting of a matrix of configurable logic blocks (CLBs) interconnected by programmable routing. FPGAs are a significant advancement in the semiconductor industry because they provide post-manufacturing programmability, in contrast to conventional VLSI circuits, which are completely customizable during the manufacturing process. Before the advent of FPGAs, digital logic design was limited to a few options: discrete logic devices (such as SSI and VLSI), Cell-Based ASICs, programmable devices like PALs or PLDs, and Mask-Programmed Gate Arrays (MPGAs) Figure 1, while programmable devices permitted the integration of logic equivalent to dozens or even hundreds of discrete components, discrete devices could only implement a small amount of logic. Users could program these devices using specialized hardware on-site. With the added benefit of user reprogram ability, FPGAs combine the advantages of PLDs and MPGAs by enabling the implementation of thousands of logic gates on a single chip. FPGAs, in contrast to ASICs, which are designed and manufactured specifically for a given function, can be reprogrammed multiple times to meet changing application and functionality requirements. This flexibility, along with reduced time-to-market, lower development costs, and high performance, has made FPGAs an increasingly popular choice for digital signal processing (DSP), prototype development, custom computing, and logic emulation.

Digital FIR filters are widely used in various digital signal processing (DSP) applications due to their inherent stability and linear phase characteristics. Consequently, optimizing speed and power consumption is essential for achieving high-performance filter designs. However, most existing filter designs tend to focus on either high-speed operation or low power consumption, rarely achieving both. To address this, it is possible to design an FIR filter that minimizes both execution time and power usage by employing a more efficient algorithm [1].

To develop an effective VLSI architecture that makes use of offset binary coding and distributed arithmetic and to design and implement a novel finite impulse response (FIR) digital filter using FPGA technology. The objective is enhanced speed, area efficiency, and power consumption performance.

2. BACKGROUND

Traditionally, DSP applications were implemented using ASICs and dedicated DSP processors. However, with the emergence of FPGAs, there has been a significant shift in design trends due to their superior performance, flexibility, and re-configurability. Designing efficient architectures for DSP functions—particularly Finite Impulse Response (FIR) filters—has gained considerable attention. FIR filters are widely used in various domains, including telecommunications, wireless communication, video and audio processing, and biomedical signal processing. The core component of FIR filters is the multiply and Accumulate (MAC) unit, typically implemented using multiplier-based architectures. However, these architectures have been found to be resource-intensive and costly in terms of area and power. In FPGA-based FIR designs, a constant trade-off exists between power consumption and processing speed. Therefore, there is a growing demand for FIR filter designs that are both low-power and high-speed. Recent algorithmic advancements have demonstrated significant improvements in performance metrics such as power efficiency, speed, and overall resource utilization.

Linear-phase FIR filters optimized for high performance on FPGA platforms are the subject of the paper "FPGA Implementation of High-Performance FIR Filters." An optimal structure for effective FPGA implementation is found by comparing various well-known multiplier architectures. An interleaved memory structure is used to accommodate incoming sample values and support high data throughput. A generic VHDL-based filter model has been developed for increased flexibility. This model enables automatic synthesis—from filter specifications to FPGA deployment. It is controlled by three user-defined parameters: the number of filter taps, the signal word length, and the coefficient word length. It uses 2's complement arithmetic to operate [1]. High-speed Finite Impulse Response (FIR) filters can be implemented using only registered adders and hardwired shifters in a new method. To reduce the number of necessary adders, the method extensively employs a modified mutual sub-expression elimination algorithm. The implementations that are produced by the optimization are contrasted with those that are generated by Xilinx Coregen TM employing Distributed Arithmetic. The optimization is targeted specifically for Xilinx Virtex-II FPGAs [2]. An 8-bit, 8-tap FIR filter for noise reduction and bandpass filtering to stabilize signal frequency is designed and implemented in this paper [3]. An input signal is sampled by an ADC with a peak-to-peak amplitude of 1 mV, a sampling rate (fs) of 2000 Hz, and a frequency of 500 Hz. After that, a Field Programmable Gate Array (FPGA) uses FIR filtering to process the digitized signal [4]. In comparison to the internal analog noise, the proposed implementation reduces noise by 13% and exhibits a processing delay of only 13.42 ns, significantly less than that of conventional dynamic algorithms. A Digital-to-Analog Converter (DAC) is then used to reconstruct and verify the original analog signal from the filtered digital output.

To meet the demands of real-time signal processing, a 16-tap linear-phase low-pass FIR filter was designed using the Kaiser Window method with a cutoff frequency of 100 kHz, implemented in direct form. The signal was analyzed in both the time and frequency domains before and after filtering. On the EP2C35F672C8 FPGA, the design produced a high-speed FIR low-pass filter with a maximum frequency response of 61.1 MHz [5]. The overall design demonstrated significant improvements in both efficiency and filtering performance [6]. Digital Signal Processing (DSP) is a highly demanding domain in modern technology, with increasing requirements for enhanced performance and efficient resource utilization. FPGAs are now the preferred platform for implementing and evaluating DSP applications due to recent advancements in FPGA technology, particularly the integration of dedicated DSP functional blocks and inherent features like re-configurability, reusability, scalability, and cost-effectiveness. A low-power FIR filter was developed and implemented in this setting to meet the requirements of DSP systems that save energy [5]. An effective FIR filter design based on an FPGA is presented alongside a description of the characteristics and fundamental principles of FIR digital filters in this proposal [6]. The coefficients of the filter were made with the FDA Tool in MATLAB, and a VHDL-based 16th-order constant-coefficient FIR filter was made. Quartus II was used to simulate the design, and the results met the desired performance requirements. A novel window function is proposed for the design of FIR filters. This window is tunable, allowing its characteristics to be adjusted by modifying a variable parameter. The Hamming and Kaiser windows are contrasted with the proposed window [7]. According to simulation results, the proposed window outperforms the Hamming window (5.74 dB) and the Kaiser window (18.87 dB) in terms of side-lobe roll-off ratio.

3. METHDOLOGY

Distributed Arithmetic (DA) is a computational technique that replaces traditional multiplications with lookup table-based operations, making it particularly well-suited for implementing multiply intensive algorithms such as FIR filters. In DA, sum-of-products computations are reformulated into a series of repeated lookup table accesses, with the resulting values accumulated to produce the final output. This approach is especially effective when one of the multiplication operands is constant, as in FIR filtering. Distributed Arithmetic stores precomputed partial products in ROM or RAM in FPGA architectures based on Lookup Tables (LUTs), such as the Xilinx XC4000 series. The address lines, derived from the bits of the variable operand, are used to retrieve these stored values efficiently. Although DA operates in a bit-serial manner—potentially making it appear slower—its performance is significantly optimized when the vector length closely matches the word size. This makes Distributed Arithmetic an attractive option for high-speed, area-efficient FIR filter implementations in FPGA environments. Proposed lookup-table based Distributed Arithmetic (DA) implementation are shown below Figure 1

Figure 2

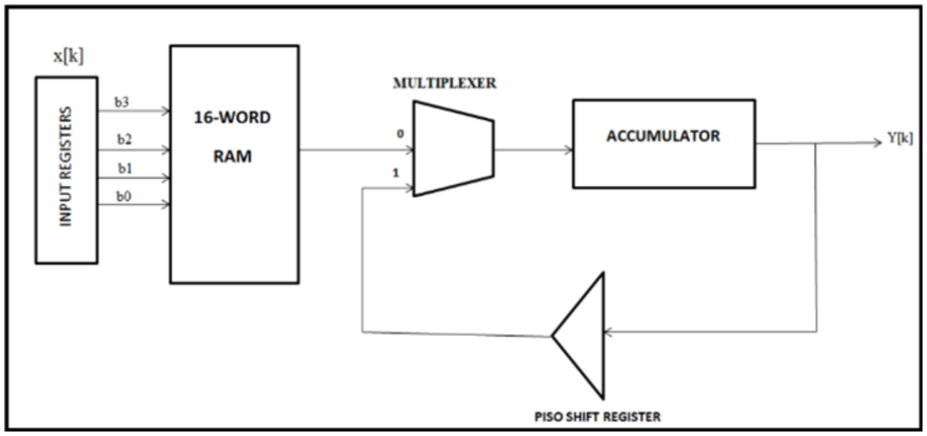


Figure 2 Proposed lookup-table based DA

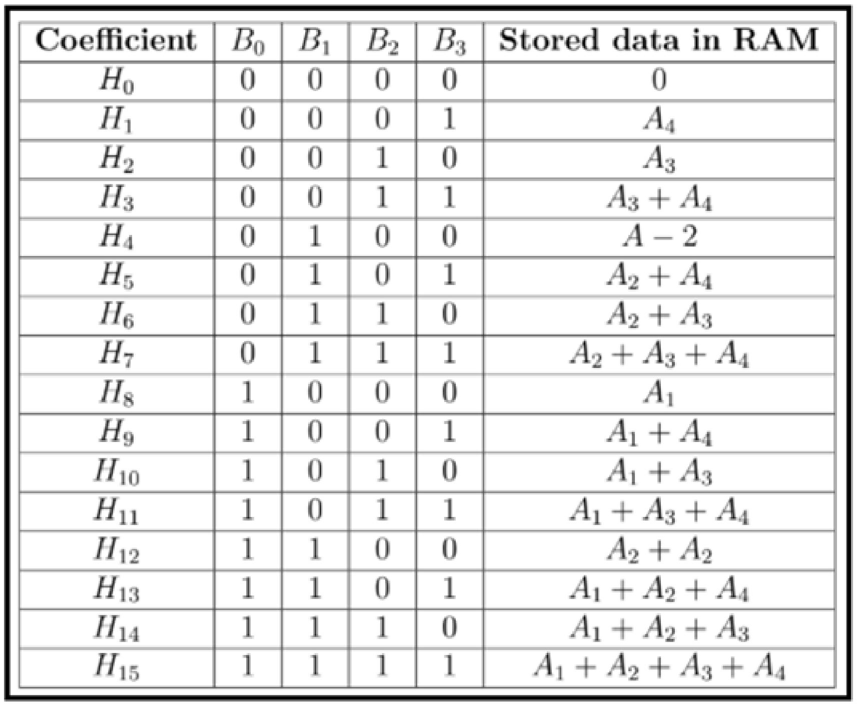
1) Register Input

The input samples X[k], each with a data width of N, are stored in the input register Figure1. Since the input samples are initially in parallel form, they must be converted to serial form to generate the appropriate addresses for the Look-Up Table (LUT). This conversion is achieved by shifting the parallel input samples to the right with each clock cycle, thereby producing the serial bit stream required to form the LUT address.

2) 16-Word RAM

To implement a 4-input LUT unit, we utilized a predefined LUT table that contains all possible sum combinations of the filter coefficients. Verilog HDL was used for the implementation. The lookup table's contents were created and saved for use during the filtering procedure prior to execution.

Table 1 LUT table in DA Algorithm



Coefficient	B_0	B_1	B_2	B_3	Stored data in RAM
H_0	0	0	0	0	0
H_1	0	0	0	1	A_4
H_2	0	0	1	0	A_3
H_3	0	0	1	1	$A_3 + A_4$
H_4	0	1	0	0	$A - 2$
H_5	0	1	0	1	$A_2 + A_4$
H_6	0	1	1	0	$A_2 + A_3$
H_7	0	1	1	1	$A_2 + A_3 + A_4$
H_8	1	0	0	0	A_1
H_9	1	0	0	1	$A_1 + A_4$
H_{10}	1	0	1	0	$A_1 + A_3$
H_{11}	1	0	1	1	$A_1 + A_3 + A_4$
H_{12}	1	1	0	0	$A_2 + A_2$
H_{13}	1	1	0	1	$A_1 + A_2 + A_4$
H_{14}	1	1	1	0	$A_1 + A_2 + A_3$
H_{15}	1	1	1	1	$A_1 + A_2 + A_3 + A_4$

3) Accumulator

This stage includes an accumulator, implemented using a 33-bit register. Since the addition of two 32-bit values may produce a carry, an extra bit is allocated to accommodate the overflow. By adding the shift register's output and the LUTs' output, partial products are created in the accumulator. These partial products are accumulated over successive clock cycles, and the final result from the accumulator represents the output of the FIR filter.

4) Shifter Unit

A 33-bit Parallel-In Serial-Out (PISO) shift register is employed in this design. The input to the shift register is the output of the accumulator. During the filtering process, the least significant bit (LSB) of the shift register is fed back to the accumulator, allowing for sequential accumulation of partial products.

5) Control Unit

The control unit is responsible for managing the behaviour of individual circuit components as well as the overall system operation. The control unit of Distributed Arithmetic (DA) typically consists of a counter, whose upper limit is determined by the input precision and has a direct impact on the throughput of the circuit. One key advantage of DA-based systems is that their throughput remains independent of the filter order, which distinguishes them from other implementation methods.

6) Distributed Arithmetic using Offset Binary Coding

A novel technique utilizing Offset Binary Coding (OBC) is proposed to reduce the memory requirements of the Look-Up Table 2 (LUT) by half. In this architecture, a combination of LUTs and multiplexers (MUXs) is employed for computation.

Figure 3

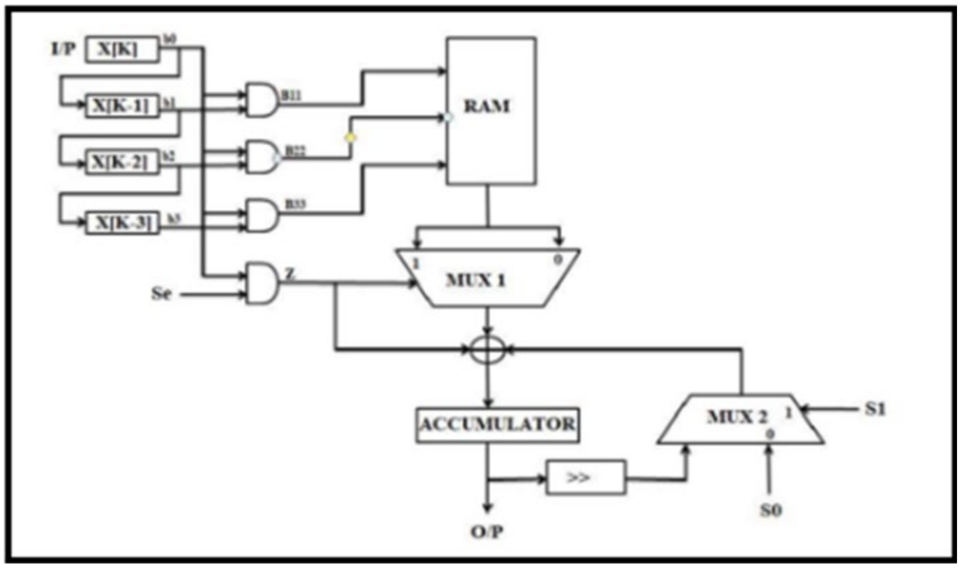


Figure 3 Proposed based offset binary implementation

Typically, the memory demand for a Distributed Arithmetic (DA)-based FIR filter implementation grows exponentially with the filter order N, requiring 2N words. By applying OBC, this requirement is effectively reduced to 2N−1 words. In OBC, input data is represented as −1 for logic '0' and +1 for logic '1'. A new LUT is constructed based on this encoding, forming the foundation for a more memory-efficient DA architecture using Offset Binary Coding.

Table 2 LUT table in Offset Binary Coding Algorithm

Coefficient	B_{1N}	B_{2N}	B_{3N}	B_{4N}	C_{1N}	C_{2N}	C_{3N}	C_{4N}	Stored data in RAM
H_0	0	0	0	0	-1	-1	-1	-1	$-\frac{1}{2}(A_1 + A_2 + A_3 + A_4)$
H_1	0	0	0	1	-1	-1	-1	1	$-\frac{1}{2}(A_1 + A_2 + A_3 - A_4)$
H_2	0	0	1	0	-1	-1	1	-1	$-\frac{1}{2}(A_1 + A_2 - A_3 + A_4)$
H_3	0	0	1	1	-1	-1	1	1	$-\frac{1}{2}(A_1 + A_2 - A_3 - A_4)$
H_4	0	1	0	0	-1	1	-1	-1	$-\frac{1}{2}(A_1 - A_2 + A_3 + A_4)$
H_5	0	1	0	1	-1	1	-1	1	$-\frac{1}{2}(A_1 - A_2 + A_3 - A_4)$
H_6	0	1	1	0	-1	1	1	-1	$-\frac{1}{2}(A_1 - A_2 - A_3 + A_4)$
H_7	0	1	1	1	-1	1	1	1	$-\frac{1}{2}(A_1 - A_2 - A_3 - A_4)$
H_8	1	0	0	0	1	-1	-1	-1	$-\frac{1}{2}(-A_1 + A_2 + A_3 + A_4)$
H_9	1	0	0	1	1	-1	-1	1	$-\frac{1}{2}(-A_1 + A_2 + A_3 - A_4)$
H_{10}	1	0	1	0	1	-1	1	-1	$-\frac{1}{2}(-A_1 + A_2 - A_3 + A_4)$
H_{11}	1	0	1	1	1	-1	1	1	$-\frac{1}{2}(-A_1 + A_2 - A_3 - A_4)$
H_{12}	1	1	0	0	1	1	-1	-1	$-\frac{1}{2}(-A_1 - A_2 + A_3 + A_4)$
H_{13}	1	1	0	1	1	1	-1	1	$-\frac{1}{2}(-A_1 - A_2 + A_3 - A_4)$
H_{14}	1	1	1	0	1	1	1	-1	$-\frac{1}{2}(-A_1 - A_2 - A_3 + A_4)$
H_{15}	1	1	1	1	1	1	1	1	$-\frac{1}{2}(-A_1 - A_2 - A_3 - A_4)$

Using MATLAB's FDA tool kit, the filter parameters meeting the above requirements were calculated out.

Table 3 Design Specification on MATLAB

Filter	Data set	Function	Cutoff Frequency	Sampling Frequency
High pass	1	Equiripple	9600Hz	48000Hz
High pass	2	Equiripple	12000Hz	60000Hz
High pass	3	Equiripple	16000Hz	72000Hz

New LUT table will be formed with new architecture for
Distributed Arithmetic using Offset Binary Coding.

Table 4 Data Sets/Coefficients

Coefficient	Value 1	Value 2	Value 3
H_0	bf88	3f90	3f90
H_1	3fa0	bfaf	bf99
H_2	bf9d	3fb3	bf94
H_3	bf7b	bf81	3fba
H_4	3f95	bfb0	bfbc
H_5	bf96	3f7e	bf5c
H_6	bf92	3fba	3fb8
H_7	3f99	bf7b	3f9c
H_8	3f9f	bfd4	bfd4
H_9	bf9c	3fc0	3fde
H_{10}	bfaa	bfd4	bfd4
H_{11}	3f9c	bf7b	3f9c
H_{12}	3fb9	3fba	3fb8
H_{13}	bfa0	3f7e	bf5c
H_{14}	bfd4	bfb0	bfbc
H_{15}	3fe1	bf81	3fba

Let the input sample xi in offset binary coding be represented as:

$$y = \sum_{n=0}^{N-1} \varrho(c_{1n}c_{2n}\cdots c_{Kn})2^{-n} + 2^{-(N-1)}\varrho(0)$$

4. RESULT AND DISCUSSION

The Xilinx ISE Design Suite was used for the synthesis and implementation of the designed FIR filter. The design process made use of coefficients for fixed high-pass filters. To evaluate performance and conduct detailed analysis, the filter was initially designed using MATLAB's Filter Design Tool. The resulting coefficients were then scaled to 16-bit precision and converted into hexadecimal format for hardware implementation. As illustrated in the figure, three distinct frequency ranges were considered, each with different sampling and cut-off frequencies. Accordingly, three separate data sets were used for the design and analysis of the FIR filter.

Figure 4

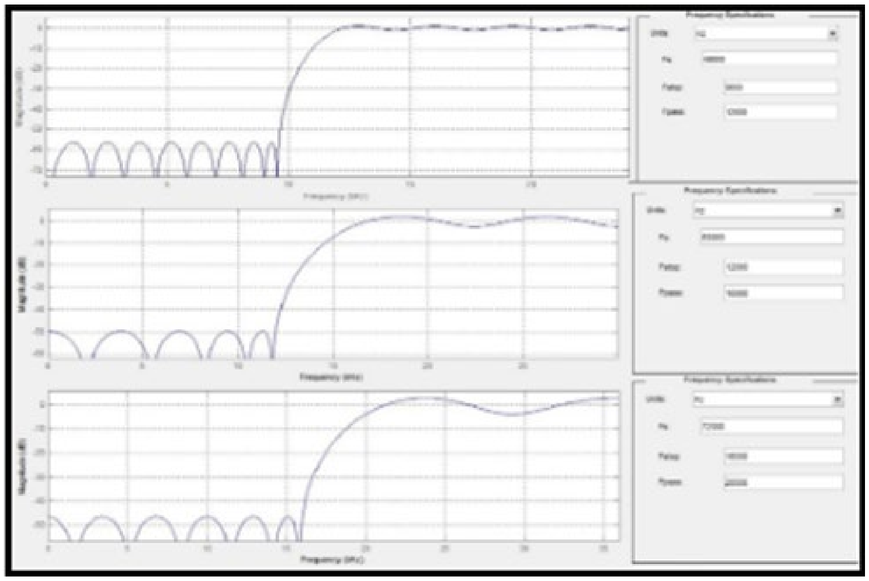


Figure 4 Magnitude response of FIR filter on MATLAB for varying frequencies

1) Simulation of Distributed Arithmetic Algorithm

The Distributed Arithmetic (DA) algorithm was synthesized using Xilinx Vivado, and the RTL schematic of the DA top module was generated. Behavioral simulation of the DA algorithm was performed, and the corresponding simulation results were analysed Figure 4.

Figure 5

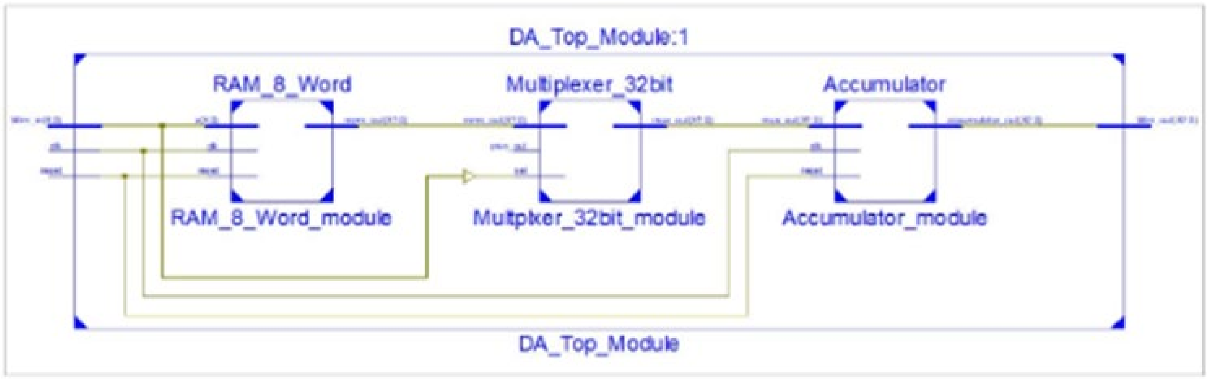


Figure 5 Schematic of DA Algorithm

As expected, signal components with frequencies lower than the specified cut-off frequency were effectively attenuated and did not pass through the filter, confirming the correct functionality of the FIR filter design.

2) Simulation of DA-OBC Algorithm

Xilinx Vivado was used to synthesize the Distributed Arithmetic with Offset Binary Coding (DA-OBC) algorithm and successfully generate the RTL schematic for the DA top module. Behavioural simulation of the DA-OBC algorithm was conducted, and the resulting waveforms were analysed Figure 5. The generated bit stream file was loaded and programmed onto the device using the appropriate configuration settings. A confirmation dialog box was displayed upon successful programming the code.

Figure 6

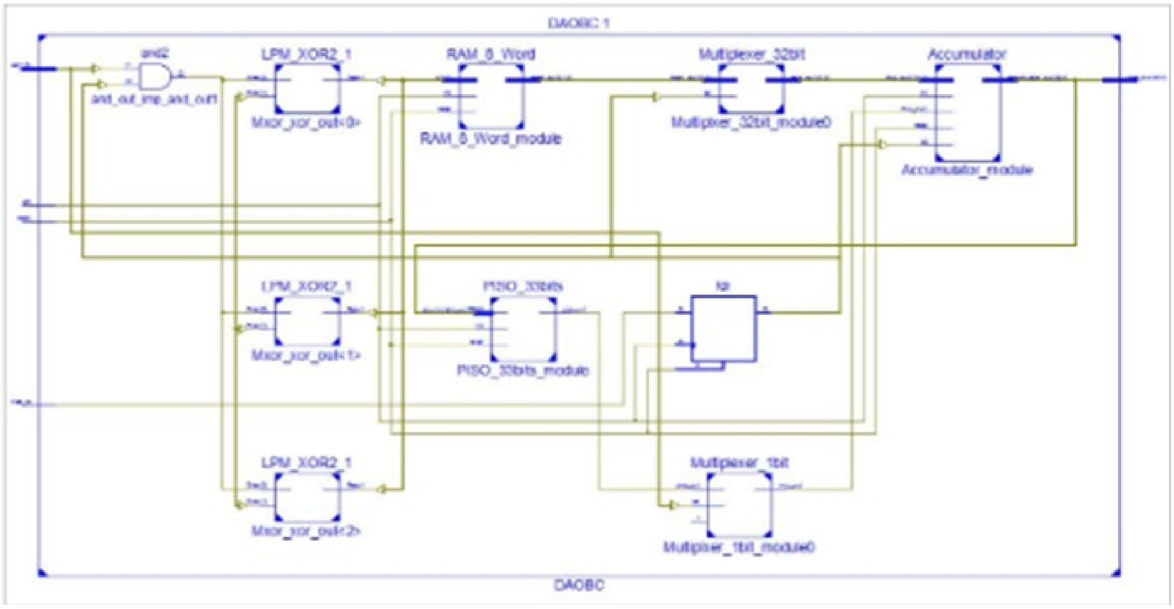


Figure 6 Schematic of DA-OBC Algorithm

Chip Scope Pro was used to analyse the output waveform following the upload of the algorithm to the FPGA. This made it possible to observe the filter's performance in real time Figure 6

Figure 7

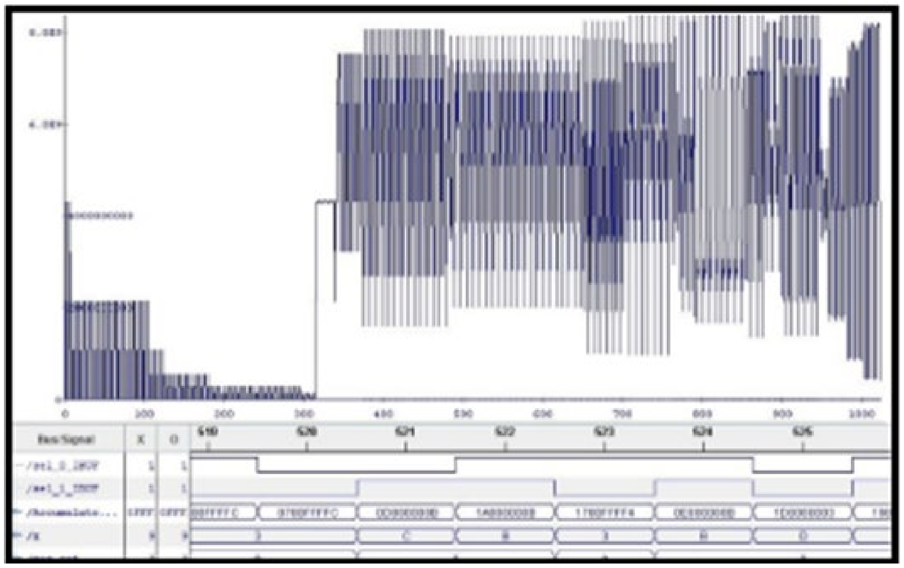


Figure 7 Bus Plot and Waveform obtained on Chip Scope

Chip Scope Pro was utilized for the design analysis. After launching the software, the JTAG chain was scanned, and the target device XC3S400 was selected. The CDC (Chip Scope Definition Core) file corresponding to the implemented algorithm was imported.

Table 5 Logic Utilization-DA

Logic Utilization	Data Set 1	Data Set 2	Data Set 3	Average
Slice Registers	349	349	326	350
Slice LUTs	326	323	326	325
Flip Flops	348	348	351	349
Number of bonded IOBs	39	39	39	39
BUFGMUXs	2	2	2	2

After pressing the Run button to start the analysis, the bus plot was used to show the output signals, and the data vs. time graph was chosen to see how the output waveform was sampled. The Summary option was selected from the Design Overview, and the corresponding Logic Utilization report was generated.

The Logic Utilization for three different data sets was recorded for both the Distributed Arithmetic (DA) and DA with Offset Binary Coding (DA-OBC) algorithms. The average logic utilization for each algorithm was then calculated for comparison. Additionally, the Timing Constraints section was accessed from the Design Overview, and the corresponding Timing Summary was obtained to evaluate the performance of the implemented designs.

Table 6 Logic Utilization-DA-OBC

Logic Utilization	Data Set 1	Data Set 2	Data Set 3	Average
Slice Registers	375	375	374	374.6
Slice LUTs	346	348	344	346
Flip Flops	374	374	373	373.6
Number of bonded IOBs	38	38	38	38
BUFGMUXs	2	2	2	2

Timing Summary of 3 data sets for DA Algorithm are observed and their averages are calculated.

Table 7 Timing Summary-DA-OBC

Timing Summary	Data Set 1	Data Set 2	Data Set 3	Average
Minimum Period (ns)	1.444	1.485	1.485	1.471
Maximum Frequency (MHz)	692.521	673.401	673.401	679.74
Setup Time (ns)	1.444	1.485	1.485	1.471

Power analysis of 3 data sets for DA and DA-OBC Algorithm are calculated, and their averages are found.

Table 8 Power Analysis-DA-OBC and DA

Power Analysis	Data Set 1	Data Set 2	Data Set 3	Average
Total On-chip power (W)	11.436	11.394	10.807	11.21
Dynamic Power (W)	11.321	11.280	10.695	11.09

Power Analysis	Data Set 1	Data Set 2	Data Set 3	Average
Total On-chip power (W)	31.744	31.7	31.64	31.694
Dynamic Power (W)	31.424	31.381	31.322	31.341

The average values of the data from the two algorithms were compared and analysed. Due to the increased number of components in the DA-OBC architecture, the number of slices utilized per LUT is higher compared to the standard DA architecture. As a result, the DA algorithm exhibits lower overall logic utilization than the DA-OBC algorithm.

Table 9 Logic Utilization

Logic Utilization	DA Algorithm	DA-OBC Algorithm
Slice Registers	350	374.6
Slice LUTs	325	346
Flip Flops	349	373.6
Number of bonded IOBs	39	38
BUFGMUXs	2	2

Figure 8

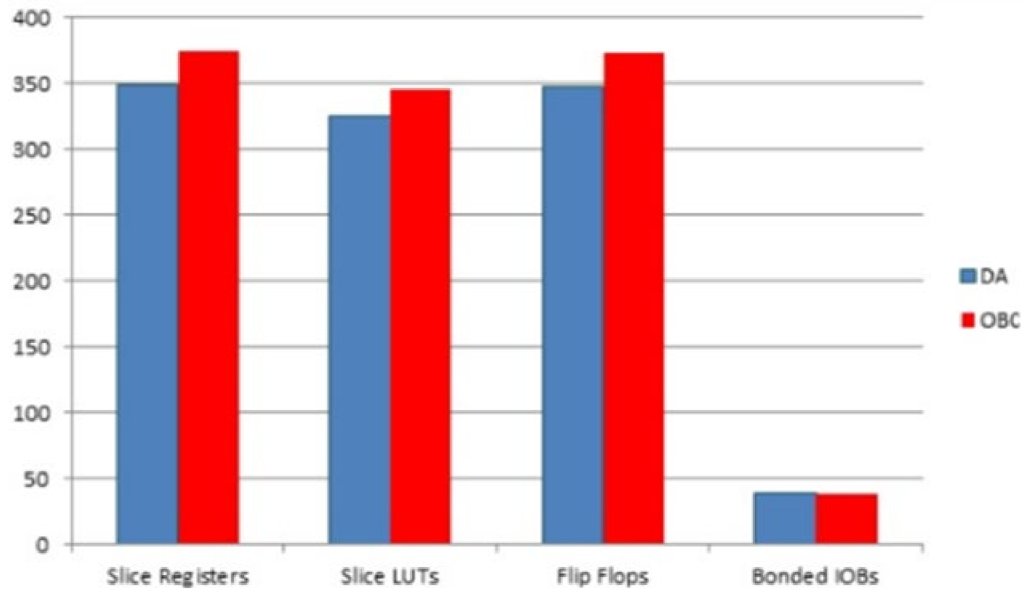


Figure 8 Logic Utilization

The data from both algorithms were analyzed and their average values compared. The DA-OBC architecture contains a greater number of functional blocks than the standard DA architecture.

Table 10 Timing Summary-DA

Timing Summary	DA Algorithm	DA-OBC Algorithm
Minimum Period (ns)	2.058	1.471
Maximum Frequency (MHz)	543.951	679.64
Setup Time (ns)	2.058	1.147

Consequently, the number of slices utilized by each LUT component is higher in the DA-OBC design. Therefore, the DA algorithm demonstrates lower overall logic utilization compared to the DA-OBC algorithm.

Figure 9

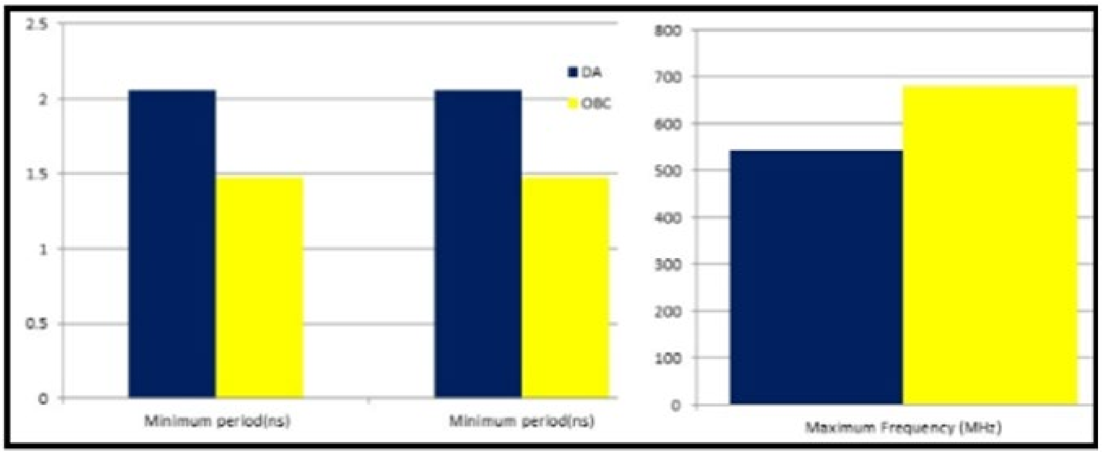


Figure 9 Timing Summary

The DA-OBC algorithm's minimum period and setup time are significantly lower than those of the standard DA algorithm, as shown by the timing summary graphs. This indicates that DA-OBC can operate at higher frequencies, making it a more time-efficient and high-performance design compared to the conventional DA approach.

Table 11 Power Analysis

Power Analysis	DA Algorithm	DA-OBC Algorithm
Total On-chip power (W)	31.694	11.21
Dynamic Power (W)	31.341	11.09

Figure 10

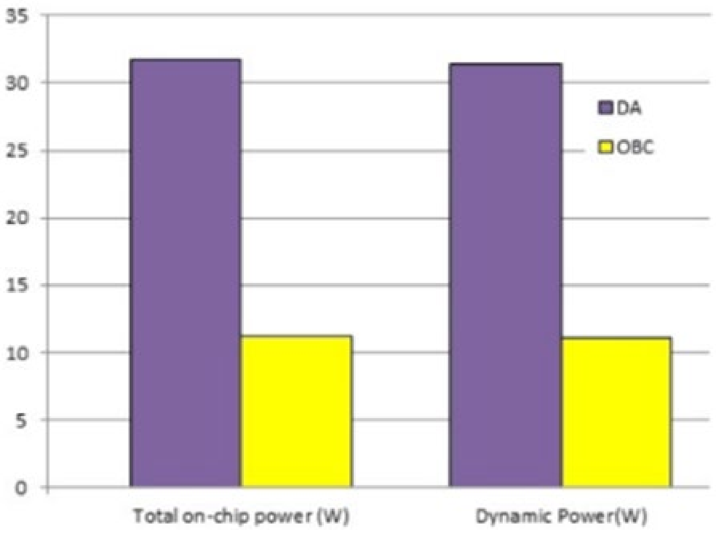


Figure 10 Power Analysis

The power consumption of the DA algorithm is 31.694 W, whereas the DA-OBC algorithm consumes significantly less power at 11.21 W. Furthermore, the dynamic power of DA-OBC is only 11.09 W, which is substantially lower than the 31.341 W consumed by the standard DA design Fig 9. Based on these findings, DA-OBC is a more energy-efficient option because it makes use of Offset Binary Coding to implement FIR filter design.

Table 12 Memory Usage

Memory Usage	DA Algorithm	DA-OBC Algorithm
Peak memory usage (MB)	151	148

Figure 11

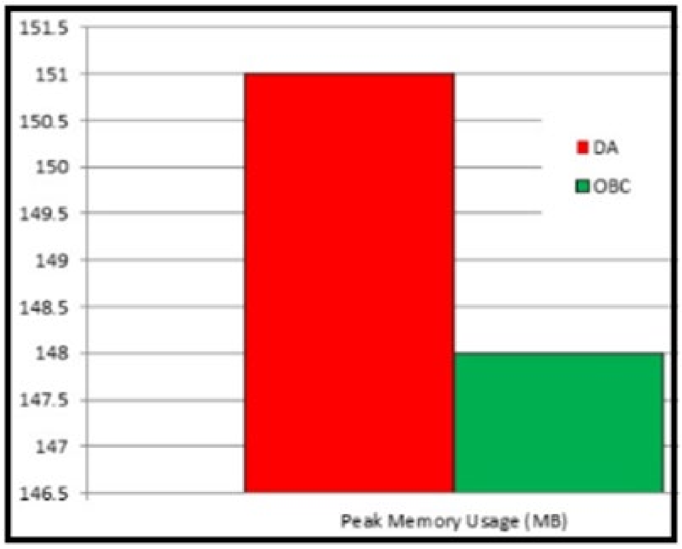


Figure 11 Memory Usage

Since the DA-OBC algorithm utilizes only half the LUT size compared to the standard DA algorithm, the overall memory consumption observed in the memory usage chart is significantly lower for DA-OBC. This confirms that DA-OBC is a more memory-efficient design than the conventional DA approach.

5. CONCLUSION

Numerous design implementations of FIR filters have been developed using various algorithms—some demanding high computational power, while others involve more basic operations. There is frequently a trade-off between important design parameters like power, timing, and area in all of these implementations. Typically, an improvement in one or two of these parameters comes at the expense of the others.

Distributed Arithmetic (DA) and Distributed Arithmetic with Offset Binary Coding (DA-OBC) emerged as the most suitable algorithms for FPGA-based implementations among the ones that were evaluated. DA relies on the use of precomputed Look-Up Tables (LUTs) for multiplication operations, replacing traditional logic-based computation. Offset Binary Coding, on the other hand, is a biased method of representing numbers that makes optimal use of memory by using unsigned numbers to represent signed values. Due to their high availability of LUTs, FPGAs are ideal for such LUT-intensive algorithms. A clear trade-off exists between logic utilization, timing, and power consumption in the DA and DA-OBC implementations. While DA significantly enhances system speed and conserves hardware resources, the inclusion of Offset Binary Coding further improves timing and power efficiency.

In conclusion, DA performs better than DA-OBC in terms of logic utilization, while DA-OBC performs better in terms of timing and power consumption. As a result, the overall findings show that DA-OBC is the best option for implementing a FIR filter on FPGA platforms due to its superior power and time efficiency.

CONFLICT OF INTERESTS

None.

ACKNOWLEDGMENTS

None.

REFERENCES

P. Kollig, B. Al-Hashimi, and K. Abbott, "Fpga implementation of high performance fir filters," in 1997 IEEE International Symposium on Circuits and Systems (ISCAS),vol. 4, 1997, pp. 2240–2243 vol.4.

S. Mirzaei, A. Hosangadi, and R. Kastner, "Fpga implementation of high speed fir filters using add and shift method," in 2006 International Conference on Computer Design, 2006, pp. 308–313.

T.-T. Do, H. Kropp, C. Reuter, and P. Pirsch, "A flexible implementation of high-performance FIR filters on Xilinx FPGAs," in Field-Programmable Logic and Applications (FPL), Lecture Notes in Computer Science, vol. 1482, Springer, 1998, pp. 441–445.

R. R. Sudharsan, "Synthesis of fir filter using adc-dac: A fpga implementation," in 2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development (INCCES), 2019, pp. 1–3.

G. Jinding, H. Yubao, and S. Long, "Design and fpga implementation of linear fir low-pass filter based on kaiser window function," in 2011 Fourth International Conference on Intelligent Computation Technology and Automation, vol. 2, 2011, pp. 496–498.

S. Bhattacharjee, S. Sil, and A. Chakrabarti, "Evaluation of power efficient fir filter for fpga based dsp applications," Procedia Technology, 2013 vol. 10, pp.856–865

X. Jiang and Y. Bao, "Fir filter design based on fpga," in 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), vol. 13, 2010, pp. V13–621–V13–624.

M. Shil, H. Rakshit, and H. Ullah, "An adjustable window function to design an fir filter," in 2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR), 2017, pp. 1–5.