

FPGA IMPLEMENTATION OF A 64-TAP FIR FILTER USING RADIX-8 BOOTH MULTIPLIER IN VERILOG

Nallapati Sowmya¹, Sreejeesh S G², Nanditha N Varma³

National Institute of Electronics and Information Technology, Calicut, India

¹sowmyanallapati7879@gmail.com, ²sree@calicut.nielit.in, ³nanditha@calicut.nielit.in

Abstract -In this paper, the FPGA implementation of a 64-tap FIR filter is realized by using a Radix-8 Booth multiplier for efficient multiplication and high-speed performance. Designed using Verilog HDL, and implemented on a Xilinx Artix-7 FPGA (xc7a100tcsg324-1), the prototype was implemented on a Nexys A7-100T development board, with synthesis and implementation performed in Vivado 2021.2. Considering the computational accuracy and hardware complexity, the Q1.15 fixed-point format is adopted in this design. In this work, the architecture is analyzed based on the resource utilization and operating frequency, making it appropriate for real-time FPGA-based digital signal processing applications.

Keywords: FIR filter, Radix-8 Booth multiplier, FPGA, Verilog HDL, Nexys A7-100T.

(Article history: Selected papers from NIELIT's International Conference on Communication, Electronics and Digital Technologies (NICE-DT'26), 29-30Jan, 2026, Gorakhpur, India)

1. Introduction

Finite Impulse Response filters are among the most frequently employed elements in digital signal processing systems owing to their intrinsic stability and linearity. They are employed in many applications in audio and image processing, communication systems, and biomedical signal processing, among many others. On the other hand, the complexity in FIR filters depends on the number of taps, and as such, there are significant hardware requirements and power consumption, especially in real-time systems.

In FIR filters, the process of multiplication is intrinsic, and as such, any form of efficiency in multiplication directly impacts the performance. In most cases, conventional multipliers are performance bottlenecks in FIR filters. This, in essence, provides another reason for employing efficient FIR filters. The radix-8 booth multiplication algorithms are efficient in FIR filters owing to the reduction in the number of partial products. This may be advantageous in terms of reduced logic depth and potentially lower power consumption, which may be advantageous in FIR filters implemented in FPGAs.

It is also noteworthy that FPGAs offer hardware resources that facilitate parallel computation and the customization of the data path. The resource utilization of the designed FIR filter can thus be optimized by incorporating the Radix-8 Booth multiplier into the FIR structure without compromising the high speed of the computation process. Moreover, fixed-point arithmetic formats such as Q1.15 provide a good trade-off between the precision of the computation and the hardware utilization.

In the present study, a 64-tap FIR filter with a Radix-8 Booth multiplier is implemented in Verilog HDL and implemented on a Xilinx Artix-7 FPGA platform with the help of a Nexys A7-100T development board. The implemented design is synthesized, placed, and routed with the help of the Vivado 2021.2 tool and evaluated in terms of hardware utilization and the maximum frequency that can be achieved with the designed structure.

1.1 Mathematical Background of FIR Filter

A Finite Impulse Response (FIR) filter is mathematically represented as:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

where:

$y[n]$: output signal

$x[n-k]$: delayed input samples

$h[k]$: filter coefficients

N : number of taps

The FIR filter performs a convolution operation between the input signal and the set of coefficients. The complexity increases linearly with the number of taps. The efficient implementation of the multiplier plays an important role for higher-order filters, such as the proposed 64-tap FIR filter. For efficient hardware implementation, fixed-point arithmetic in Q1.15 form is employed, in which 1 bit represents the integer part and 15 bits represent the fractional part.

2. Literature Survey

Different approaches have been discussed to implement FIR filters on FPGA platforms considering optimization of computation speed, hardware usage, and power efficiency.

[1] In "Implementation of Radix-8 FIR Filter Using Booth Multiplier for Reduced Power Consumption," an FIR filter architecture was implemented, which uses a Booth multiplier in Radix-8 form to minimize partial products. This results in a faster implementation of the FIR filter. By utilizing a Spartan-3E FPGA from Xilinx and designing the architecture with Xilinx ISE, the FIR filter architecture showed a reduced hardware usage compared to conventional multipliers. However, only the results from the synthesis level were discussed. There was no implementation of post-implementation power consumption results.

[2] In "High Speed and Low Power FIR Filter Using Vedic Multiplier," a Vedic multiplication algorithm was used to implement a high-speed FIR filter. This algorithm was integrated with the FIR filter to minimize the logic depth. By utilizing a Virtex-6 FPGA, a higher maximum operating frequency was obtained compared to a Booth multiplier-based FIR filter. This implementation was totally based on the results obtained from the synthesis level. However, results from the routed design were not available.

[3] Low Power FIR Filter Design Using Shift and Add Multiplication Technique proposes a low-power implementation of an FIR filter. In this design, the authors reduced the switching activities and hence the dynamic power dissipation in the FPGA. This was achieved by replacing the multipliers in the direct form of the FIR filter with shift and add multipliers. However, this design targeted an older version of FPGAs and made use of simulation tools to estimate the power dissipation. It did not make use of hardware power estimation tools.

[4] In Design of FPGA-Based FIR Filter Using Distributed Arithmetic, the authors have already pre-computed the partial results. Hence, the multipliers were replaced. This reduced the logic utilization and improved the speed of the filter. In this design, the authors have implemented the filter in a Spartan-6 FPGA.

It is clear from the literature review that though there are various methods to optimize FIR filters in FPGA devices, most of them focus on speed and area or, in a few cases, power consumption during the simulation phase, while very few papers discuss post-implementation power breakdown for modern FPGA devices. This paper bridges this gap by designing a 64-tap FIR filter with a Radix-8 Booth multiplier and reporting routed design results, including both resources and frequency, for a Xilinx Artix-7 FPGA device.

3. System Architecture

The design utilizes a finite impulse response filter with 64 taps optimized for FPGA implementation, along with a Radix-8 Booth multipliers to support high-speed multiplication. Our design has been implemented on a Xilinx Artix-7 FPGA platform, utilizing the xc7a100tcsg324-1 device on a Nexys A7-100T development board.

The top-level datapath of the design is shown in Fig. 1. The Datapath of the design begins with the input interface, where the discrete time signals are provided to the design. In order to sample the signals in accordance with the fixed coefficients, the signals are passed through a tap delay line. Each output of the tap delay line is then multiplied by its corresponding coefficient using a Radix-8 Booth multipliers block. The output of all the multipliers' outputs are then pipelined and accumulated in an adder tree to produce the accumulated result. Debugging and hardware validation are achieved using Virtual Input/Output (VIO) and Integrated Logic Analyzer (ILA) IP cores.

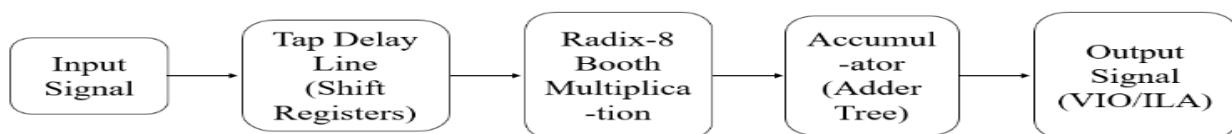


Fig. 1. Top-level architecture of the proposed FIR filter

3.1 Tap Delay Line

The tap delay line is the primary data structure that maintains the last 64 samples. In each clock cycle, a new sample is inserted into the delay line, and the previous samples are shifted one position down. The oldest sample in the delay line is removed, and the latest sample is available for multiplication with $h[0]$. This ensures that each tap has the correct delayed sample available in parallel. The delay line utilizes distributed registers instead of block RAM to avoid access latency. This ensures timing closure. In parallel, all 64 taps can be read out in the same clock cycle.

3.2 Coefficient Storage

The filter coefficients are constants during operation and are stored in block RAM (BRAM) preloaded during bitstream generation. There are several advantages of using BRAM to store filter coefficients:

- Single-cycle access time
- Centralized storage, avoiding routing congestion
- Ease of coefficient updates in reconfigurable systems

Each coefficient is read in the same cycle that its delayed sample is available, resulting in zero stall cycles in this datapath.

3.3 Radix-8 Booth Multiplier

The heart of the filter is the Radix-8 Booth multiplier, which reduces the number of partial products generated during multiplication, as opposed to standard binary multiplication, by examining the multiplier bits in sets of three. This results in significant reductions in hardware for partial product generation and a drastic reduction in the critical path.

Working Principle of Radix-8 Booth Encoding: The radix-8 Booth encoding reduces the number of partial products by grouping the multiplier bits in sets of three. Multiplication is performed according to the combination of the bits as:

Table 1. Working Principle of Radix-8 Booth Encoding.

Booth Encoding	Operation
000	$0 \times M$
001	$+1 \times M$
010	$+1 \times M$
011	$+2 \times M$
100	$-2 \times M$
101	$-1 \times M$
110	$-1 \times M$
111	$0 \times M$

This reduces the partial products to $N/3$, which improves speed significantly compared to conventional multipliers.

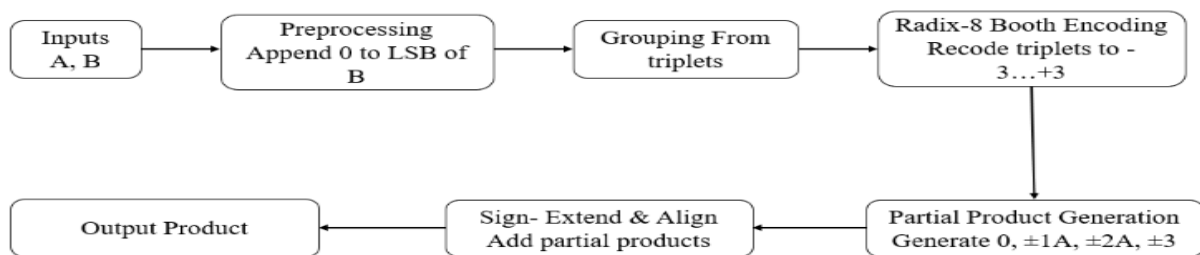


Fig. 2Block diagram of the Radix-8 Booth multiplier

Key design optimizations include:

- Parallel partial product generation to maintain high throughput
- Signed arithmetic handling to support Q1.15 fixed-point operations
- Balanced adder tree within the multiplier to achieve 100 MHz timing closure

3.4 Comparison with Other Multipliers

Table 2. Comparison of Multiplier Architectures.

Multiplier Type	Speed	Area	Power
Conventional Multiplier	Low	High	High
Booth (Radix-2)	Medium	Medium	Medium
Booth (Radix-8)	High	Low	Low
Vedic Multiplier	Very High	Medium	Medium

The Radix-8 Booth multiplier provides a fair compromise between speed and hardware usage, which makes it appropriate for use in designing high-order FIR filters.

3.5 Pipelined Accumulator (Adder Tree)

After multiplication, the tap products are signed 32-bit integers. The products are summed up by a balanced binary adder tree with pipeline registers between stages. The accumulation process is as follows:

Stage 1: 64 products $\rightarrow$ 32 sums

Stage 2: 32 $\rightarrow$ 16

Stage 3: 16 $\rightarrow$ 8

Stage 4: 8 $\rightarrow$ 4

Stage 5: 4 $\rightarrow$ 2

Stage 6: 2 $\rightarrow$ 1 (final sum)

Pipeline registers are also used to ensure that the design maintains a sample per cycle throughput while having minimal combinational delay in all stages. The final sum is also maintained in 40 bits to avoid overflow before the final rounding.

3.6 Output Formatting

The final 40-bit accumulated result is then rounded and saturated to fit back into the 16-bit Q1.15 output format. The saturation logic is used to ensure that the results are clamped to the correct range to prevent wrap-around errors, which are extremely critical in many DSP applications.

3.7 Control Logic

A centralized control unit manages:

Sample input timing and delay line shifting

Coefficient read enable signals

Booth multiplier enable pulses

Pipeline register synchronization

Since the design is fully pipelined, the control logic is simple, relying on a global enable signal that ties to the system clock.

3.8 Debugging Infrastructure

To validate the design on the Nexys A7-100T board:

- VIO cores are used for injecting test input sequences and observing output samples without the need for any dedicated pins.
- Internal signals such as outputs from the multiplier, intermediate results from the adder tree, and the final output from the accumulator are monitored using ILA cores.

This allows for real-time debugging using the Vivado Hardware Manager.

3.9 Design Scalability

Although the design is currently for 64 taps, it is easily extendable to other numbers of taps. The Booth multiplier block is also easily substitutable for other word lengths and multiplier algorithms.

3.10 Implementation Considerations

The choice to use the Radix-8 Booth multiplier is based on the trade-off between DSP slice usage and LUT-based logic. While Artix-7 FPGAs contain DSP48E1 slices, Booth multipliers implemented using LUTs minimize DSP usage, freeing them for other purposes in more complex systems. Furthermore, the pipelining is important to guaranteeing latency, which is crucial for real-time DSP.

This expanded design ensures that the FIR filter is able to satisfy the two requirements of throughput and resource usage in modern FPGAs, making it suitable for many DSP applications where both are crucial.

4 Implementation and Results

The designed 64-tap FIR filter architecture with Radix-8 Booth Multiplier was coded in Verilog HDL and synthesized on an Xilinx Nexys A7-100T FPGA board.

4.1 Design Flow

The design implementation flow followed is depicted in Fig. 6 (System Architecture). Some of the most important steps of this design flow are as follows:

- 1. HDL Design and Modular Verification:** Radix-8 Booth Multiplier was designed and coded in Verilog HDL, and it accepts signed 16-bit data and produces 40-bit output. The designed 64-tap FIR filter was coded in fixed-point data type with a point of one and 15 bits, and it includes a delay line, a multiplier array, and an adder tree.
- 2. Simulation and Functional Verification:** The designed 64-tap FIR filter and Radix-8 Booth Multiplier are simulated separately to verify their functionality and accuracy.
- 3. Synthesis:** This step is followed by synthesizing the designed 64-tap FIR filter and Radix-8 Booth Multiplier using Xilinx's Vivado Synthesis tool for Xilinx Artix-7 FPGA XC7A100T-1CSG324.
- 4. Implementation:** Placement and routing of the synthesized design are performed by Vivado implementation tools, and constraints are added to set a clock period of 10 ns (i.e., 100 MHz), and performance reports are generated to analyze the design timing.
- 5. Generation of bitstream and hardware testing:** Finally, the bitstream was generated and loaded onto the Nexys A7 board. Hardware output verification was performed with Virtual Input/Output (VIO) and Integrated Logic Analyzer (ILA) cores, and 7-segment displays.

4.2 Resource Utilization

The post-synthesis and post-implementation resource utilization is depicted in Table 1. From the architecture, a large amount of DSP resources is utilized, along with a small usage of LUT and flip-flops, leaving a lot of resources available for more modules to be included.

Table 3. Post Implementation Resource Utilization.

Resource	Used	Available	Utilization (%)
LUT as Logic	7096	63400	11.19
CARRY4	1419	15850	8.95
LUT as Shift Register	95	19000	0.50
LUT as Distributed RAM	24	19000	0.13
F7/F8 Muxes	12	63400	0.02
Registers (FF)	3435	126800	2.71
Block RAM (BRAM)	1	135	0.74
I/O Pins	17	210	8.10

4.3 Performance Summary

The implemented FIR filter shows efficient usage of the resources in the FPGA while providing reasonable performance. The pipelining technique is employed to ensure continuous processing of the data, with one sample being processed per clock cycle.

Although the design does not satisfy the timing constraints of 100 MHz, it operates well at lower frequencies, i.e., ~58 MHz.

4.4 Power Analysis

The power analysis after implementation, as performed in Vivado, is shown in the results below, as given in Table 2. The power consumption is mainly due to dynamic switching activity in DSP slices and interconnects.

Table 4. Power Report Summary

Power Component	Value (V)
Dynamic Power	0.122
Static Power	0.097
Total On-Chip Power	0.219

4.5 Timing Analysis

The timing summary after implementation is shown in Table 3. The result is a Worst Negative Slack (WNS) of -7.098 ns, indicating that the design is not meeting the 100 MHz timing constraint without any optimization. However, correctness is maintained for lower frequency operation.

Table 5. Timing Report Summary

Parameter	Value
Target Lock Period	10.000 ns
Target Frequency	100.000 MHz
Achieved WNS (Setup)	-7.098
Total Negative Slack	-100.050 ns
Worst Hold Slack (WHS)	0.067

4.6 Discussion on Timing Violation

The negative slack observed in the design is mainly due to:

- Deep combinational paths in the adder tree
- Delay introduced by Booth encoding logic
- Insufficient pipeline stages in critical paths

This can be improved by:

- Increasing pipeline depth
- Using DSP48 slices instead of LUT-based multipliers
- Applying retiming and register balancing

4.7 Simulation Results

1. Booth Multiplier Simulation: The Radix-8 Booth multipliers were executed with different signed input combinations. The waveform in Fig. 3 merely represents proper partial product generation (pp0 to pp5), sign extension, and alignment for different sign combinations of operands.


Fig. 3 Simulation waveform of Radix-8 Booth multiplier showing signed partial product generation and accumulation.

2. FIR Filter Simulation: The FIR filter was simulated with a fixed number of 64 taps and coefficients. The waveform in Fig. 4 represents the input sample stream (data_in), the set of coefficients, the generated partial products, the accumulated sum, and the filtered output (data_out) correctly. The alignment of the values in the intermediate stages proves the correctness of the multiplication and addition phases.

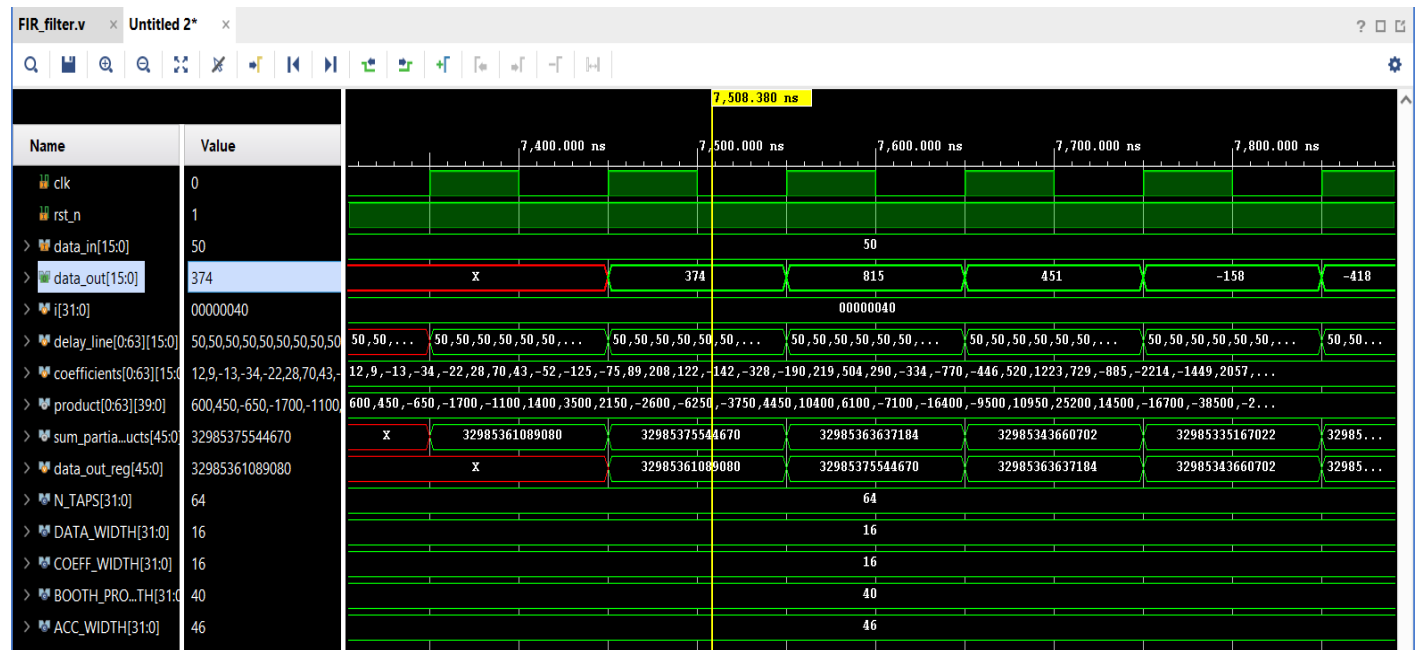


Fig. 4 FIR filter simulation output showing coefficient multiplication, partial products, and final result.

4.8 Hardware Implementation

Finally, this resulting bitstream was loaded into the Nexys A7-100T FPGA board, and the output filtered values were implemented on an array of LEDs, where all LED patterns were representative of binary representations of signals processed for the input samples.

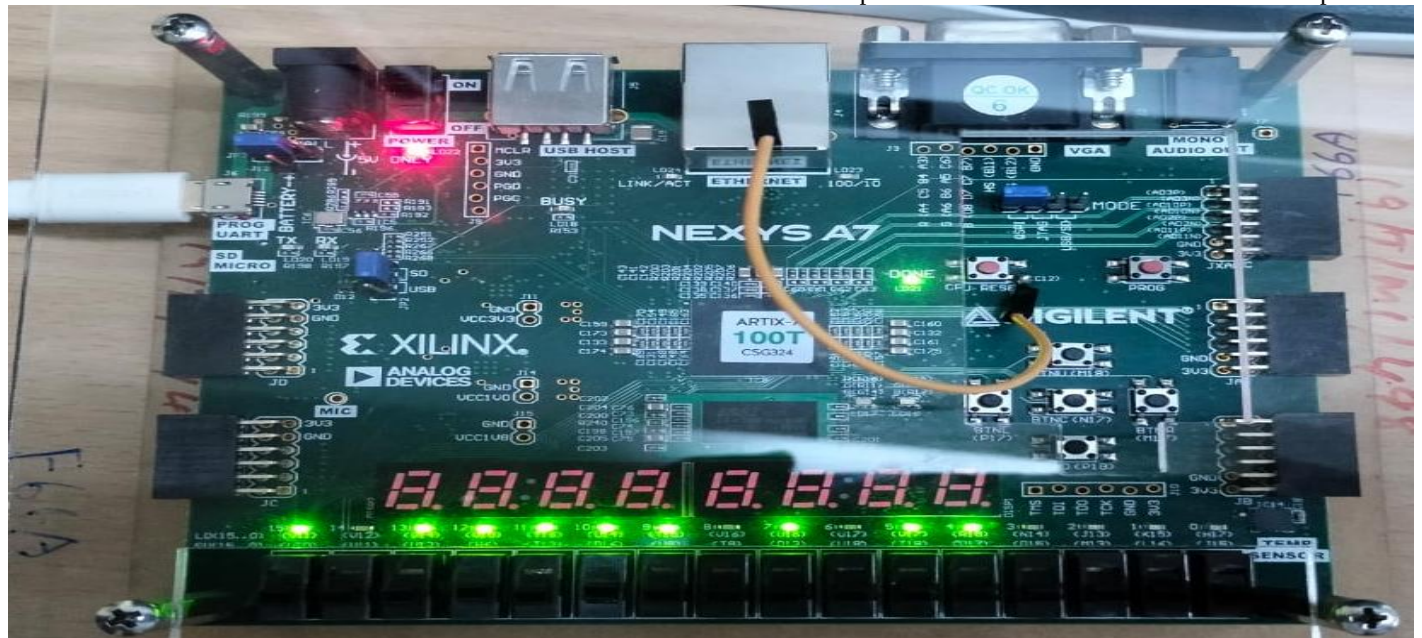


Fig. 5 Indicates the hardware output for test case 1, which is a fixed input pattern.

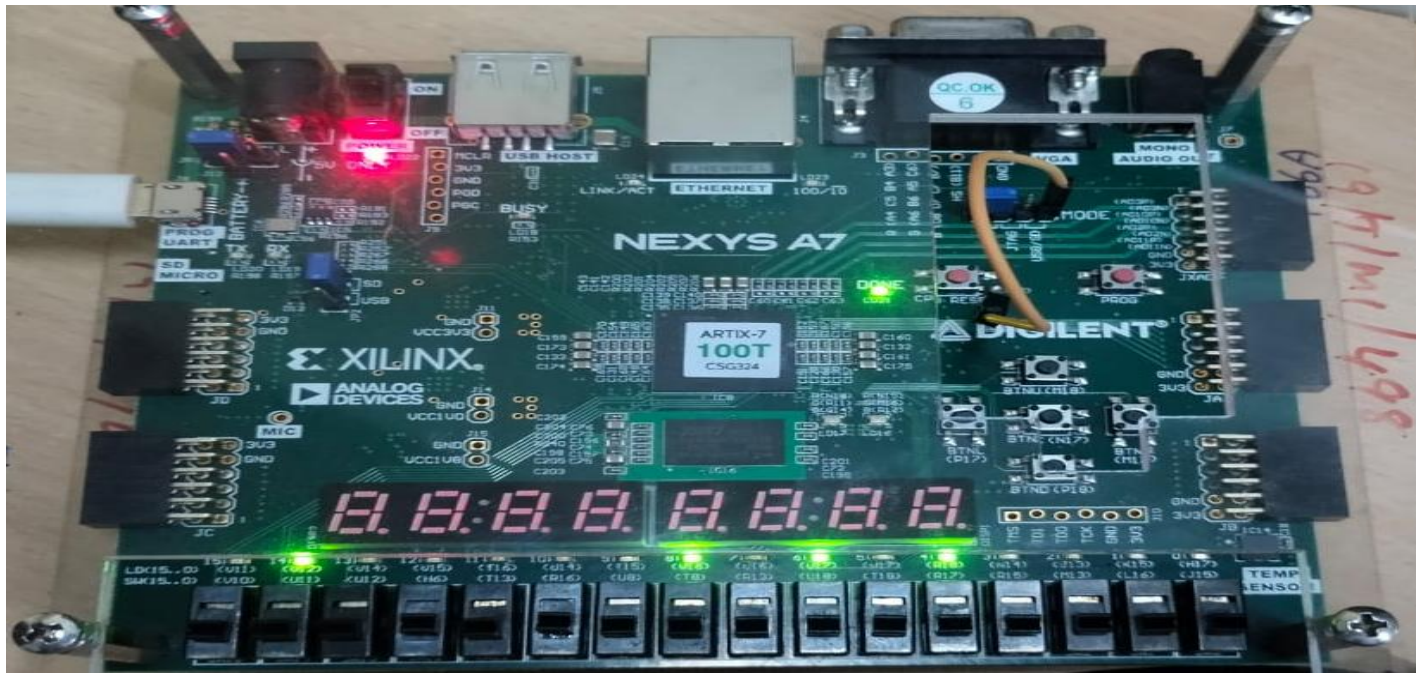


Fig. 6 Indicates the hardware output for test case 2, confirming the design functionality using a varying set of input values.

5 Applications of Proposed Design

The proposed 64-tap FIR filter architecture can be used in:

- Audio signal processing (noise removal, equalization)
- Image processing (edge detection, smoothing)
- Communication systems (channel filtering, modulation)
- Biomedical signal processing (ECG, EEG filtering)
- Software-defined radio (SDR) systems

6 Conclusions

In this paper, an introduction to the implementation of a 64-tap finite impulse response filter implemented on an FPGA platform utilizing a Radix-8 Booth multiplier was given. The implementation was done in Verilog HDL targeting a Xilinx Artix 7 FPGA board, Nexys A7-100T. Using the Vivado 2021.2 tool, implementation results were obtained, which showed that the implementation consumed a total power of 0.219 W. This power consumption included a dynamic power component, which accounted for 55.7% of the total power consumption. From the implementation results, low resource usage was obtained, with 11.19% of look-up table usage and 2.71% usage of registers. Even though functionality was verified, results from the timing analysis showed a Worst Negative Slack of -7.098 ns. This gives an estimated maximum frequency of ~58.48 MHz, which is much lower than the required 100 MHz.

The proposed architecture shows that optimized design of multipliers is an important aspect in achieving efficient implementation of FIR filters. Radix-8 Booth encoding is an efficient technique in achieving this goal with reduced complexity while ensuring accuracy.

Future Work: Further improvement of pipelining in the stages of the multiplier and accumulator, or by applying retiming optimization to reduce the length of the critical path, is possible. Adding functionality to support adaptive filtering is also a possible direction to make this design more useful in a broader field of DSP algorithms.

7 References

1. P. Kumar, S. Reddy, and R. Gupta, "Implementation of Radix-8 FIR Filter Using Booth Multiplier for Reduced Power Consumption," *Int. J. Eng. Appl. Sci. Technol.*, vol. 5, no. 2, pp. 55–59, Mar.–Apr. 2020.
2. A. Sharma and M. Singh, "High-Speed and Low-Power FIR Filter using Vedic Multiplier," in *Proc. IEEE Int. Conf. Advances in Computing, Communication Control and Networking (ICACCCN)*, Greater Noida, India, Dec. 2018, pp. 201–205.
3. R. Patel and K. Meena, "Low Power FIR Filter Design Using Shift and Add Multiplication Technique," *Int. J. Comput. Appl.*, vol. 172, no. 6, pp. 18–22, Aug. 2017.
4. D. Verma and S. Gupta, "Design of FPGA-Based FIR Filter Using Distributed Arithmetic," *Int. J. Electron. Commun. Eng.*, vol. 9, no. 1, pp. 23–28, Jan. 2016.
5. A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
6. K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ, USA: Wiley, 1999.
7. M. Ercegovac and T. Lang, *Digital Arithmetic*, 2nd ed. San Mateo, CA, USA: Morgan Kaufmann, 2020.
8. V. K. Sharma, S. K. Sharma, and N. Sharma, "Design and implementation of low power FIR filter using modified Booth multiplier," *Proc. IEEE Int. Conf. Emerging Trends in Computing, Communication and Nanotechnology*, Tirunelveli, India, Mar. 2013, pp. 99–103.
9. S. R. Srikanth and P. K. Meher, "High-speed and low-complexity implementation of FIR filters using distributed arithmetic," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 1, pp. 149–160, Jan. 2014.
10. S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4–19, July 1989.
11. C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
12. A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27(4), pp. 473–484, Apr. 1992.
13. Xilinx Inc., "7 Series FPGAs Data Sheet: Overview," DS180, v2.4, Jan. 2021.
14. Digilent Inc., "Nexys A7 Reference Manual," Accessed: Aug. 2025. [Online]. Available: <https://digilent.com/reference>
IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019, Jul. 2019.