Optimal IIR Filter Design and FPGA Realization

Arunjyothi Eddla
Department of EECE, Gandhi Institute of Technology and Management (Deemed to be University), Rudraram, Hyderabad, Telangana, India
aeddla@gitam.edu (corresponding author)

V. Y. Jayasree Pappu
Department of EECE, Gandhi Institute of Technology and Management (Deemed to be University), Rushikonda, Visakhapatnam, Andhra Pradesh, India
jpappu@gitam.edu

Received: 20 June 2025 | Revised: 16 August 2025, 19 October 2025, and 2 November 2025 | Accepted: 3 November 2025

 DOI: https://doi.org/10.48084/etasr.12837

ABSTRACT

An Infinite Impulse Response (IIR) filter implemented on a Field-Programmable Gate Array (FPGA) provides low-area, high-speed Digital Signal Processing (DSP) with efficient hardware utilization. However, designing an IIR filter on an FPGA can be challenging due to the recursive nature of filters in fixed-point arithmetic processes, which makes it difficult to obtain numerical stability and minimize hardware resource usage. This research proposes the Array Multiplier-Optimal Adder-IIR (AM-OA-IIR) and Array Multiplier-Kogge-Stone Adder-IIR (AM-KSA-IIR) architectures to minimize hardware usage. The Carry Lookahead Adder (CLA) and the Carry Skip Adder (CSA) are combinations of the Optimal Adder (OA), which effectively manage multi-operand addition by minimizing carry propagation delay. The CSA performs rapid intermediate additions without immediate carry computation, whereas the CLA quickly resolves final carries. This integration reduces logic depth and minimizes the overall area required for the FPGA implementation. Furthermore, the KSA offers high-speed arithmetic by minimizing the delay caused by carry propagation through parallel computation. Therefore, the proposed AM-OA-IIR and AM-KSA-IIR achieve a logic element count of 379 and 486, respectively, for the Virtex5 FPGA device at order 8, compared to the logic element count of 1,024 for a conventional FIR-based IIR design.

Keywords-Array Multiplier (AM); Carry Skip Adder (CSA); Infinite Impulse Response (IIR); Optimal Adder (OA)

I. INTRODUCTION

FPGA devices have become an effective and popular platform for DSP applications. These precise, flexible architectures of Integrated Circuit (IC) resources are used in a wide range of applications [1]. FPGAs provide reconfigurable hardware platforms that enable the effective use of IIR filters, which contain an impulse response for limitless amounts of time. IIR filters are a type of digital filter containing delay elements, adders, System-on-Chip (SoC) platforms, Microcontroller Units (MCUs), and processors [2, 3]. Compared to Finite Impulse Response (FIR) filters, IIR filters offer faster computation speeds, lower complexity, and improved magnitude responses [4]. Most signal and data processing applications rely on fundamental arithmetic operations, with multiplication and addition being the most crucial and widely used. Traditional arithmetic operations can be replaced in numerous ways to provide accurate results with less hardware for rapid operation and compactness [5, 6]. To achieve ideal system performance, it is crucial to determine the appropriate filter coefficient that minimizes the error surface between the filter's output and the output of the unknown model, thereby achieving an optimized value [7]. Various types of adders, such as carry skip, carry save, carry lookahead, and carry choose, are designed to produce faster outputs while minimizing hardware requirements [8]. The selection of lower-order and higher-selectivity filters is achieved by using IIR filter realization [9]. When two numbers are multiplied, Partial Products (PP) are formed, which are then added to obtain the final product [10]. The input data for the IIR filter are randomly generated using a True Random Number Generator (TRNG), which produces unpredictable numbers utilizing an asynchronous origin and a physical process. The lack of similarity between the most recently gathered sequences strengthens the security of the TRNG [11]. A significant benefit of FPGA-based TRNGs is the use of a large matrix of Configurable Logic Blocks (CLBs) interconnected through programmable interconnects, which enables the growth of a robust platform in digital VLSI [12, 13]. Furthermore, the system achieves greater integrity and data protection than

traditional pseudo-random number generators by leveraging the physical randomness and inherent unpredictability of TRNGs [14-16]. The reconfigurable nature of FPGA-based TRNGs enables rapid customization and scalability to adapt to various application-specific needs while maintaining low area and power usage [17, 18]. However, designing an IIR on an FPGA reveals difficulties in achieving numerical stability and minimizing hardware resource usage due to the recursive nature of the filters in the fixed-point arithmetic process. The AM-OA-IIR and the AM-KSA-IIR minimize this problem by integrating optimized adder logic. Authors in [14] created a parallel-pipelined, FIR-based, IIR filter with an eight-tap parallel FIR structure and a scaling factor. This design achieved a high IIR response and shortened cycle time. However, achieving the requisite IIR sharpness required higher FIR lengths, which increased computational load and delay. Authors in [15] used Distributed Arithmetic (DA) to reduce the hardware requirements and the number of taps in their design of an FPGA-based Butterworth low-pass IIR filter of different orders. Low-order architectures benefited from improved speed and smaller footprints; however, delay and logic complexity increased due to the necessary recursive feedback. Authors in [16] used hybrid multipliers and adders, proposing a hybrid IIR filter to increase speed. However, including multiplexers and 2-bit adders made the circuit more complex, resulting in the wasteful use of FPGA resources and routing congestion. Authors in [17] decreased the area and power consumption while increasing the speed of a DA-based 4-bit FIR filter. However, one of the main drawbacks was that the pre-computed PP increased exponentially for higher-order filters, greatly increasing the necessary memory. Authors in [18] introduced an optimization-dependent Linear Phase Finite Impulse Response (LPFIR) filter implementing a Hybrid Namib Beetle and Battle Royale Optimization Algorithm (HNBOA) to minimize power consumption. The transition bandwidth of the LPFIR filter adheres to the designated frequency, which minimizes the filter's sparsity. Moreover, LPFIR-HNBOA enhances filter performance by decreasing passband and stopband ripples, as well as transition bandwidth, leading to a more precise frequency response. Nevertheless, the fine-tuning of LPFIR coefficients introduces significant design complexity and additional switching activity, which varies the intended power-saving benefits in high-order filter designs. Overall, existing filters have limitations, such as longer filter lengths and additional circuit complexity. This results in significant design complexity and additional switching activity, leading to increased routing congestion and suboptimal resource usage. To improve speed, area, and power balance, the AM-OA-IIR design combines an AM with an OA, such as a CLA or CSA, which is appropriate for low-power VLSI. The primary function of the OA is to minimize carry propagation delay by estimating and avoiding carry paths. This results in a faster, more organized, and more parallel layout with less gate consumption and connection complexity. In the second design, the AM-KSA-IIR, the AM is combined with a high-speed KSA, which offers reduced latency and a regular hardware structure that is ideal for embedded applications. The KSA uses parallel prefix carry computation to quickly and effectively add binary numbers, minimizing path delay and ensuring that carry signals are generated simultaneously across all bit positions. Together, these designs provide a flexible, performance-conscious approach to IIR filter implementation.

II. PROPOSED METHODOLOGY

This study proposes the use of an OA, a KSA, and an Array Multiplier (AM) in an IIR Butterworth filter design to decrease area and delay. The multiplier is replaced by an array, and the adder is replaced by an OA and a KSA. The OA helps minimize the delay associated with carry propagation via long chains of bits. Furthermore, the KSA is a parallel-prefix adder that rapidly and effectively adds binary numbers. Figure 1 shows the block diagram of the proposed IIR filter.

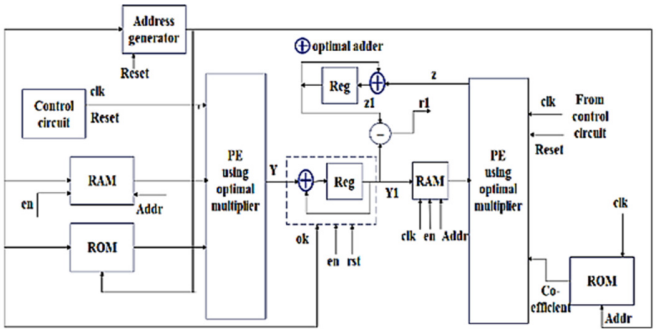

Fig. 1. Structure of the proposed filter.

A. True Random Number Generator

The TRNG stores the input data for the IIR filter in random order in RAM. Meanwhile, MATLAB's FDA tool constructs the Butterworth filter coefficients in ROM. The filter processes these data over 17 clock cycles. In each cycle, the Processing Element (PE) efficiently multiplies the current input data by its appropriate coefficient using an AM. The ideal adder combines the CLA and CSA, called the OA, with the AM in the first method. In the other method, the adder combines the KSA with the AM for speed. Then, the PE's output is added to the current accumulator value, and the result is stored in the register for that particular tap. The seventeen taps of the filter repeat this procedure, combining the stored value from the previous tap with the result from each cycle.

B. Butterworth Filter

After input data are obtained, they are passed through a ROM to generate coefficients using a Butterworth filter with the FDA tool in MATLAB. The Butterworth filter offers a maximally flat frequency response in the passband, ensuring a smooth response with minimal ripple. Additionally, the simple Butterworth design simplifies the calculation of the coefficients, leading to effective hardware realization with less complexity. This results in a more reliable and efficient FPGA-based filter system. In this module, one input value is used per clock cycle to obtain an output formulated by:

$$O[k] = aI[k-1] + \gamma O[k-2] + \delta O[k-3] + \varepsilon O[k-4] \quad (1)$$

where $I[k]$ is the sample value at instant k, $O[k]$ denotes the output value. The constant values are acquired from transfer function of filter based on cut-off and filter order.

Subsequently, the input data and coefficient value are fed into PE using AM for multiplication process.

C. Array Multiplier

The PE uses an AM to calculate the product of the first input data and the first coefficient during the first clock cycle after the coefficients are generated. This AM is crucial to the design of IIR filters because it reduces area by using fewer logic gates. It also enhances speed through parallel multiplication, operating as a combinational multiplier, employing shift-and-add logic to shift and add the PP produced by the AND gates using adders. The $n \times n$ multiplier uses $n(n-2)$ full adders, n^2 AND gates, and n half-adders. Assuming $A = a(1)a(0)$ and $B = b(1)b(0)$, the final product term P is:

$$P(0) = a(0)b(0) \quad (2)$$

$$P(1) = a(1)b(0) + b(1)a(0) \quad (3)$$

$$P(2) = a(1)b(1) + C1 \quad (4)$$

where $C1$ is the carry generated in addition of P (1) term, P (3) = $C2$, and $C2$ indicates the carry generated for the P (2) term. A set of four AND gates is used to generate the partial product terms, such as a (0) b (0), and then, an adder array calculates sums involving these PP and their corresponding carry combinations. This process yields the final product bits, ensuring accurate and efficient binary multiplication. The output from the PE (Y) is fed into the input as the accumulator, where it is added to the initial accumulator value. The result is $0 + Y = Y$. Later, this sum is stored in the register of the first tap, and the inputs are passed through the OA process to enhance the filter's performance.

D. Optimal Adder Using the Carry Lookahead Adder and the Carry Skip Adder

After multiplication, the output, Y, is routed to the adder block to enhance the IIR filter's performance. The significant carry propagation latency of the RCA, caused by stages waiting sequentially for the previous carry, must be avoided for large input additions. However, half and full adders are suitable for small inputs (3-bit, 4-bit). An OA that combines the CSA for rapid binary addition and the CLA, which generates carries in parallel, addresses this problem. The CLA and CSA operating together provide the high-speed, effective performance required for sophisticated IIR filter designs. Figure 2 displays a CLA, which is used in the circuit to add the output of an 8-bit adder to an 8-bit accumulator. The sum produced is then saved back into the accumulator, which was previously set to zero. Due to its speed architecture, the CLA uses propagate p and generate g signals to derive its carry signals from the inputs of each complete adder stage. The propagate signal (6) decides whether to pass an incoming carry (c_i) forward to the next stage as c_{i+1}, whereas the generate signal (5) is active (set to 1) only when both input bits (a_i and b_i) are 1, requiring a carry-out:

$$G_i = A_i B_i \quad (5)$$

$$P_i = A_i \oplus B_i \quad (6)$$

The mathematical equation of sum Si and carry Ci+1 uses the generate signal Gi and the propagate signal Pi:

$$S_i = P_i \oplus G_i, C_{i+1} = P_i C_i + G_i \quad (7)$$

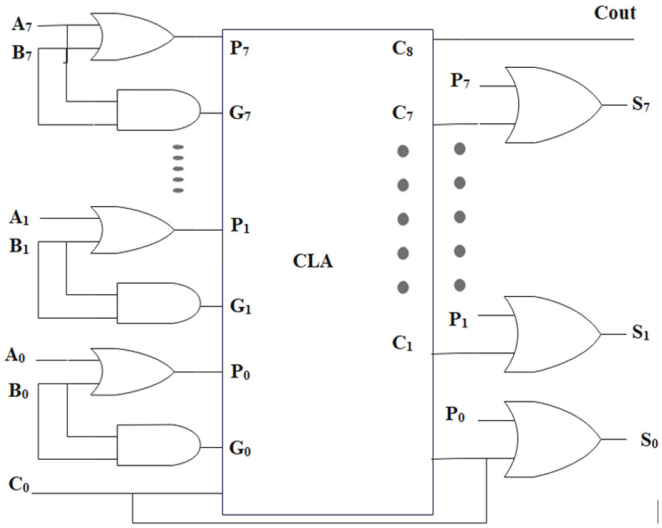

Fig. 2. 8-bit carry lookahead adder.

Since the carry-out $C_{i+1} = P_i$ depends directly on the initial carry C_0 rather than on intermediate carries, the CLA greatly reduces propagation delay and enables quicker parallel addition. The CSA, known as the carry-bypass adder, contains special circuitry that increases speed and minimizes carry propagation delay. The propagating signals are $P0$, $P1$, $P2$, and $P3$ depending on two inputs of each full adder (A, B, A, B, A, B, A, B). The first 8 bits from the adder are fed into the CLA. If no carry propagation occurs, the carry input C_{in} is directly passed to the output. Otherwise, the carry spreads through various stages before reaching the output. This approach effectively minimizes the delay associated with carry propagation. In CSA, carry skip circuitry is used depending on the carry skip mechanism. This circuitry has two primary blocks, the AND block and the multiplier block, which together are referred to as the propagate block. Here, the RCA with the carry-skip circuit is designed to establish the CSA circuit along with four full adder blocks:

$$P_i = A_i XOR B_i \quad (8)$$

The propagate signal is set to a logic 1 when $A_i \neq B_i$ and to a logic 0 when $A_i = B_i$. Later, the propagate signals from each full adder block are fed into an AND block, which performs a logical AND operation. The calculated AND output is then passed through a multiplexer block based on the data selector. Depending on the input, the multiplexer selects the corresponding value. If the input is a logic 1, the multiplexer selects $C0$ as the output. Instead of propagating through all the blocks, the carry is bypassed to the output. If the multiplexer input is a logic zero, the multiplexer selects the carry from the last full adder, which is transmitted from the initial full adder. To increase the speed of addition by using a carry-skip mechanism to decrease propagation delay, the design combines an optimum adder with a CLA and a CSA. Figure 3 shows that two 8-bit CLA1 and CLA2 and a Skip Logic block (SKIP1) are used to construct this 16-bit CSA. CLA1 processes the lower eight bits, a [7:0] and b [7:0], and performs a block carry and

partial sum calculation. SKIP1 rapidly determines whether a carry can propagate down all 8 bits of the bottom block through an OR gate, which is the key to the speedup. If so, the carry bypasses the internal stages and skips straight to the higher block's adder (CLA2), which lowers the overall carry propagation delay for the upper eight bits (*a* [15:8] and *b* [15:8]). This makes it faster than a traditional ripple-carry adder.

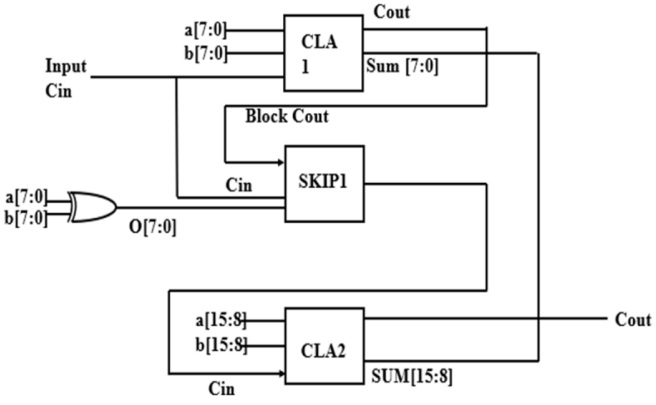

Fig. 3. Architecture of OA.

E. Kogge-Stone Adder (KSA)

The KSA architecture is a parallel-prefix adder employed in digital circuit design to rapidly and effectively add binary numbers. It is a type of CLA designed to minimize propagation delay when calculating carry bits during binary addition. In the preprocessing stage, the P and G signals are produced for each bit pair of the input operands *A* and *B*, which play a crucial role in determining carry propagation. The *P* signal indicates whether a carry from a previous bit position is carried over to the present bit position, and the *G* signal indicates whether the current bit position generates a carry independently of the previous carry. The *P* and *G* signals are calculated and sent to the next stage, where the sum and carry are obtained:

- Pre-processing stage:

$$P_i = A_i \oplus B_i \quad (9)$$

$$G_i = A_i B_i \quad (10)$$

- Generation of carry:

$$G_i = P_i G_i^* + G_i \quad (11)$$

where G_i^* is the previous stage G_i value and G_i is the present value.

$$P_i = P_i P_i^* \quad (12)$$

where P_i^* is the previous stage P_i value and P_i is the present value.

- Final processing stage:

$$C_i = G_i \quad (13)$$

$$S_i = P_i \oplus C_{i-1} \quad (14)$$

Figure 4 illustrates the KSA architecture, where the carry generation stage leverages the P and G signals from the pre-processing stage to calculate the carry bits for each output bit position in parallel. This stage contains a tree-like structure of logic gates that effectively integrates the *P* and *G* signals to generate the carries. The complexity of this network increases with the number of bits in the adder. The final stage uses the carry bits generated in the previous stage and the P signals to compute the sum bits for each output position. For each position, the carry (C) and sum (S) are found using (13) and (14). Here, C_{i-1} represents the carry bit from the previous position of the bit. The KSA uses a parallel carry generation method that minimizes the adder's delay significantly compared to the RCA, where the carry bit propagates sequentially through each stage. This reduction in delay results in a rapid addition operation, making the KSA an ideal choice for high-performance VLSI applications.

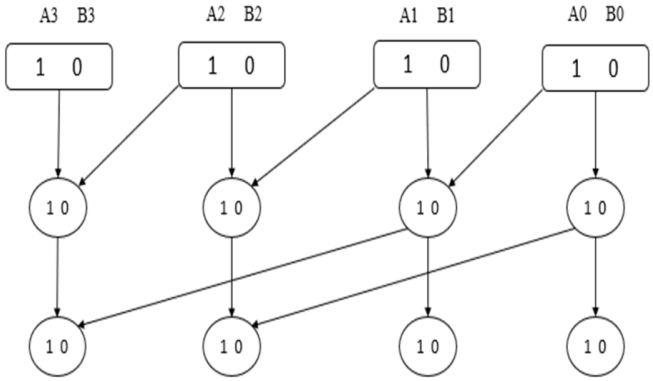

Fig. 4. Architecture of KSA.

III. SIMULATION RESULTS

The simulation results were gathered by executing the Verilog code of the IIR filter in Xilinx ISE with different coefficient values generated in MATLAB. The results are generated for an order of 8 in the Virtex 5 FPGA family, which shows significant reductions in area and delay. Metrics, such as the number of slice registers and Look-up Tables (LUTs), also known as logic elements, are used to calculate the model performance in IIR design.

A. Performance Analysis

The simulation and power results for the 8-order IIR filter using Butterworth coefficients are: {2.6527, 3.9149, 3.6088, 2.2597, 0.9570, 0.2663, 0.0440, 0.0033}; and FF: {0.0574, 0.4596, 1.6086, 3.2171, 4.0214} for a 100 kHz passband and a 300 kHz stopband frequency. Increasing the number of taps and coefficients in an 8-order IIR filter increases computational activity and enables efficient management of feedforward and feedback routes through accumulators and data transmission. Figure 5 demonstrates that the system is operating properly. Intermediate outputs are generated by multiplying the inputs obtained from RAM and the coefficients retrieved from ROM. After combining the feedforward and feedback products, the difference is calculated to determine the final output. A pipelined Multiply-Accumulate (MAC) process was shown by

the signals in the waveform. The entire procedure controls the flow of the 8-bit input data (add [7:0]) and is synchronized by the clock (clk) signal. It is activated when the enable (en) signal is high. A chain of pipeline registers, ranging from $m0_q$ [15:0] to $m4_q$ [15:0], temporarily stores 16-bit partial results as the data move through several stages. The $macc_q$ [15:0] signal continuously adds up the outputs from these stages and serves as the primary accumulator. The final output, add_q [15:0], displays the sum result after a small processing delay and verifies its accuracy.

Fig. 5. Simulation results for order 8 IIR filter design.

Figure 6 shows the power report from the Xilinx ISE Simulator for the order 8 filter, which is 0.56 W. The Virtex5 (XC5VSX50T) FPGA architecture's total on-chip power consumption is 0.563 W, with the quiescent (static) power comprising the remainder, mostly from the dynamic power used by the logic components. The filter is optimized for fixed-point digital implementation in low-area, high-speed VLSI and communication systems (e.g., RFID and AM radio) and is characterized by the feedback and feedforward coefficients. The architecture ensures excellent processing speed, while the OA structure reduces carry propagation using a CLA and CSA. The AM-KSA-IIR uses the KSA for parallel carry. This results in a simpler, more effective IIR filter.

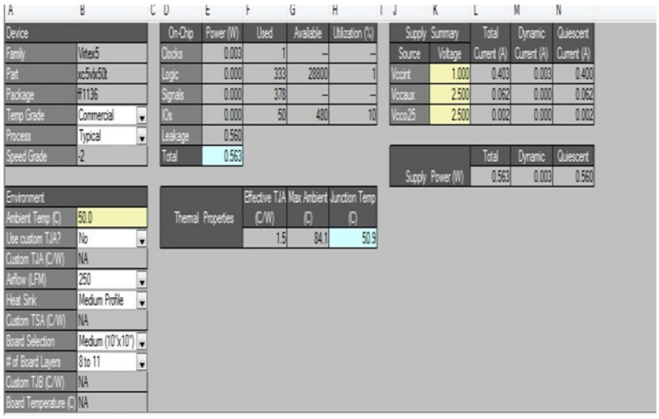

Fig. 6. Power report Virtex5.

B. Comparative Analysis

Table I presents a comparative analysis of the proposed filter on different FPGA devices. Compared to the existing filter, the proposed filter uses fewer logic elements, 379 and 486 for the AM-OA and AM-KSA filters, respectively, due to optimized adder and simplified multiplier designs. This process achieves rapid processing by minimizing significant path delay through effective carry computation, ensuring its suitability for high-speed, low-resource VLSI integration. Emphasizing propagation delay and logic usage across different FPGA families, Table I provides a comparative study of several FPGA-based IIR filter and multiplier designs. According to the research, filter designs that use a higher-order low-pass filter usually result in substantial resource usage and lengthy delays. Conversely, Cyclone IV platform implementations that use Vedic multipliers and arrays are more efficient. The proposed AM-OA-IIR design on the Virtex-5 platform outperforms all other designs, achieving the lowest delay of 9.62 ns using only 379 logic elements, proving superior in terms of speed and area optimization.

TABLE I. COMPARATIVE ANALYSIS OF THE PROPOSED IIR FILTER FOR DIFFERENT FPGA FAMILIES

Methods	Family	Delay(ns)	Logic elements
Ripple hybrid adder [10]	Virtex5	10.42	8872
Higher order IIR filter [19]	Virtex5	14.306	16,094
LPF algorithm [20]	Virtex5	55.107	4906
AM [21]	Cyclone IV	10.51	839
Vedic Multiplier [21]	Cyclone IV	9.38	661
NF IIR [22]	Zynq family	12.048	5400
VM-OA-IIR (proposed)	Virtex5	9.96	350
AM-KSA-IIR (proposed)	Virtex5	10.79	486
VM-KSA-IIR (proposed)	Virtex5	10.87	486
AM-OA-IIR (proposed)	Virtex5	9.62	379

IV. CONCLUSIONS

This study presents the Array Multiplier-Optimal Adder-IIR (AM-OA-IIR) and Array Multiplier-Kogge-Stone Adder-IIR (AM-KSA-IIR) filters to reduce delay and area usage. This design's primary objective is to reduce propagation delay and hardware resource usage (LUTs, flip-flops, slices, and IOBs) by combining an area-efficient Array Multiplier (AM) with high-speed adders, such as Carry-Skip (CSA) [23], Carry-Lookahead (CLA), and Kogge-Stone (KSA). Butterworth coefficients are necessary to maintain a precise, ripple-free response. The AM-OA-IIR design uses only 61 registers, whereas the AM-KSA-IIR design uses 93 registers, in contrast to FIR-based IIR filters. To increase area efficiency and optimize the coefficient generation process, future work may employ machine learning techniques.

REFERENCES

[1] K. Achtenberg, R. Szplet, and Z. Bielecki, "Advanced, Real-Time Programmable FPGA-Based Digital Filtering Unit for IR Detection Modules," *Electronics*, vol. 13, no. 22, Jan. 2024, Art. no. 4449, https://doi.org/10.3390/electronics13224449.

[2] K. Hong, R. Duan, L. Zhao, and Y. Zhou, "Field Programmable Gate Array (FPGA) Implementation of a Multi-Symbol Detection Algorithm with Reduced Matching Branches and Multiplexed Finite Impulse

Response (FIR) Filters," *Applied Sciences*, vol. 15, no. 4, Jan. 2025, Art. no. 2199, https://doi.org/10.3390/app15042199.

[3] K. Ariunaa, U. Tudevdagva, and M. Hussai, "FPGA based Digital Filter Design for faster operations," *Journal of VLSI Circuits and Systems*, vol. 5, no. 2, pp. 56–62, 2023, https://doi.org/10.31838/jvcs/05.02.09.

[4] H. Yi *et al.*, "Design of infinite impulse response maximally flat stable digital filter with low group delay," *Scientific Reports*, vol. 15, no. 1, Apr. 2025, Art. no. 11074, https://doi.org/10.1038/s41598-025-87175-5.

[5] R. Randriatsiferana, J. Lorandel, R. Salvador, and C. Moy, "Novel Digital NGD Methodology for FPGA-Based Embedded Systems," *IEEE Access*, vol. 12, pp. 71520–71534, 2024, https://doi.org/10.1109/ACCESS.2024.3403033.

[6] B. Zhao, H. Liu, T. Bi, and S. Xu, "Synchrophasor Measurement Method Based on Cascaded Infinite Impulse Response and Dual Finite Impulse Response Filters," *Journal of Modern Power Systems and Clean Energy*, vol. 12, no. 5, pp. 1345–1356, Sept. 2024, https://doi.org/10.35833/MPCE.2023.000824.

[7] S. Dey, P. K. Roy, and A. Sarkar, "Adaptive IIR model identification using chaotic opposition-based whale optimization algorithm," *Journal of Electrical Systems and Information Technology*, vol. 10, no. 1, July 2023, Art. no. 33, https://doi.org/10.1186/s43067-023-00102-4.

[8] M. Sakthimohan, J. Deny, K. Umapathi, and H. H. Fayek, "Enhanced FPGA linear phase FIR filter with amalgam multiplier," *International Journal of Electronics*, vol. 111, no. 10, pp. 1653–1678, Oct. 2024, https://doi.org/10.1080/00207217.2024.2349972.

[9] K. V. Gowreesrinivas, S. Srinivas, and P. Samundiswary, "FPGA Implementation of a Resource Efficient Vedic Multiplier using SPST Adders," *Engineering, Technology & Applied Science Research*, vol. 13, no. 3, pp. 10698–10702, June 2023, https://doi.org/10.48084/etasr.5797.

[10] B. Sindhuri Kandula, S. Patchala, K. Surapaneni, M. Aravind kumar, C. Ram Grandhi, and P. R. Desamala, "FPGA design of an efficient divide-and-conquer multiplier based on multiple stage fast ripple hybrid adders for peak cancellation with IIR filters," *International Journal of Electronics*, pp. 1–22, 2025, https://doi.org/10.1080/00207217.2025.2500779.

[11] H. B. Meitei and M. Kumar, "FPGA design and implementation of TRNG architecture using ADPLL based on fir as loop filter," *Analog Integrated Circuits and Signal Processing*, vol. 122, no. 1, Nov. 2024, Art. no. 2, https://doi.org/10.1007/s10470-024-02295-8.

[12] M. Nithin, K. Ganesan, M. Premsai, and S. Ponnambalam, "Design and Optimization of an Improved FIR Filter for Advanced Signal Processing." Social Science Research Network, Rochester, NY, Nov. 15, 2024, https://doi.org/10.2139/ssrn.5091234.

[13] V. Tayal and M. Jhamb, "Partial product based improved reconfigurable FIR filter with control logic for automated guided vehicles on virtex-7 FPGA," *International Journal of Information Technology*, vol. 15, no. 4, pp. 2077–2088, Apr. 2023, https://doi.org/10.1007/s41870-023-01266-y.

[14] E. M. Barhoumi, Y. Charabi, and S. Farhani, "FPGA Application: Realization of IIR filter based Architecture," *Journal of VLSI Circuits and Systems*, vol. 5, no. 02, pp. 29–35, 2023, https://doi.org/10.31838/jvcs/05.02.05.

[15] A. Yadav and R. Mehra, "FPGA based IIR Filter Design Analysis for Different Orders," *Journal of Basic and Applied Engineering Research*, vol. 1, no. 12, pp. 106–109, Dec. 2014.

[16] V. Thamizharasan and N. Kasthuri, "FPGA implementation of high performance digital FIR filter design using a hybrid adder and multiplier," *International Journal of Electronics*, vol. 110, no. 4, pp. 587–607, Apr. 2023, https://doi.org/10.1080/00207217.2022.2098387.

[17] A. Rai *et al.*, "Modeling and simulation of FIR filter using distributed arithmetic algorithm on FPGA," *Multimedia Tools and Applications*, vol. 83, no. 31, pp. 75855–75868, Sept. 2024, https://doi.org/10.1007/s11042-024-18637-7.

[18] P. Selvaprasanth, R. Karthick, P. Meenalochini, and A. M. Prabaharan, "FPGA implementation of hybrid Namib beetle and battle royale optimization algorithm fostered linear phase finite impulse response filter design," *Analog Integrated Circuits and Signal Processing*, vol. 123, no. 2, Mar. 2025, Art. no. 33, https://doi.org/10.1007/s10470-025-02385-1.

[19] M. Y. Ladekar, Y. V. Joshi, and R. R. Manthalkar, "Performance Analysis in Higher-Order IIR Filter Structures with Application to EEG Signal," *Circuits, Systems, and Signal Processing*, vol. 40, no. 8, pp. 4047–4063, Aug. 2021, https://doi.org/10.1007/s00034-021-01662-4.

[20] V. Sharma, "Design and Implementation of Efficient IIR Low Pass Filter Based On Vedic Multiplier Algorithm," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 1, pp. 115–117, Jan. 2024, https://doi.org/10.22214/ijraset.2024.57881.

[21] S. H. Ibrahim, I. S. Hameed, and H. H. Ali, "High Performance IIR Filter Design Based on Fast Multiplier," *Diyala Journal of Engineering Sciences*, pp. 192–202, Mar. 2025, https://doi.org/10.24237/djes.2025.18111.

[22] P. Chandel and S. D'Souza, "Digital Filter Designing: Minimum-Order IIR Filter Design with MATLAB," in *Data Science and Applications*, Singapore, 2025, pp. 115–125, https://doi.org/10.1007/978-981-96-1188-1_9.

[23] A. E. Pappu Dr VY Jayasree, "FPGA Implementation Of VM-CSA Fir Filter With Reduced Area And Delay Using Optimal Designs," *International Journal of Engineering Trends and Technology - IJETT*, vol. 69, no. 9, pp. 203–211, 2021.