Journal of Electronics, Electromedical Engineering, and Medical Informatics
Homepage: jeeemi.org; Vol. 7, No. 3, July 2025, pp: 663-679 e-ISSN: 2656-8632

RESEARCH ARTICLE OPEN ACCESS

Power-Efficient 8-Bit ALU Design Using Squirrel Search and Swarm Intelligence Algorithms

Ashish Pasaya, Sarman Hadia, and Kiritkumar Bhatt

Gujarat Technological University, Ahmedabad, India

Corresponding author: Sarman Hadia (e-mail: asso_s_k_hadia@gtu.edu.in), **Author(s) Email**: Ashish Pasaya (e-mail: ashishec447@gmail.com, Kiritkumar Bhatt (e-mail: krbhatt2022@gmail.com)

Abstract **The Arithmetic Logic Unit (ALU) serves as a core digital computing element which performs arithmetic functions along with logic operations. The normal operation of ALU designs leads to increased power consumption because of signal redundancy and continuous operation when new data inputs are unavailable. The research implements the Squirrel Search Algorithm (SSA) combined with Swarm Intelligence Algorithm (SIA) for 8-bit ALU optimization to achieve maximum resource efficiency alongside computational accuracy. The optimization properties of SSA and SIA make them ideal choices for digital circuit design applications because they yielded successful results in power-aware systems. The proposed method utilizes SSA-based conditional execution paired with SIA-based transition minimization to direct operations to execute only during fluctuating input data conditions thus eliminating undesired calculations. Studies confirm SSA and SIA function more effectively than distributed clock gating for power saving because they enable runtime-dependent optimization without creating significant computational overhead. The experimental Xilinx Vivado tests executed on an AMD Spartan-7 FPGA (XC7S50FGGA484) running at 100 MHz frequency established that SSA eliminates power consumption from 6 mW to 2 mW, and SIA achieves a power level of 4 mW. The SSA algorithm generates worst negative slack (WNS) values of 8.740 ns which SIA produces as 6.531 ns improving system timing performance. SSA-optimized ALU requires the same number of LUTs as the unoptimized design at 42 LUTs yet SIA uses 50 LUTs because of added logical elements. We observe no changes in flip-flop use during SSA where nine FFs remain yet SIA shows an increase in its usage up to 29 FFs due to input tracking. The study proves that bio-inspired methods create energy-efficient platforms which make them ideal for implementing ALU designs with FPGAs. Research studies demonstrate that hybrid swarm intelligence techniques represent an unexplored potential to optimize power-efficient architectures thus reinforcing their significance for future high-performance energy-efficient digital systems.**

Keywords **Power-efficient ALU; Swarm Intelligence Algorithm (SIA); Squirrel Search Algorithm (SSA); FPGA optimization; Dynamic power reduction; Low-power digital design; Heuristic techniques.**

I. Introduction

The Arithmetic Logic Unit (ALU) serves as a critical component in microprocessors, executing essential arithmetic and logical operations. However, conventional ALU architectures suffer from significant power dissipation due to continuous processing cycles and redundant signal transitions [1], [2]. Efficient power management in digital systems is crucial, particularly in embedded systems and FPGA-based designs, where power constraints directly impact performance, device longevity, and energy efficiency [3], [4], [5], [6], [7]. These factors include muscle fatigue [9], force variation [10], and forearm orientation [11]. To address the growing demand for energy-efficient digital systems, researchers have increasingly turned to nature-inspired optimization algorithms. These bio-inspired techniques have demonstrated promising results in reducing power consumption, improving battery life, and optimizing resource utilization across a range of computing and communication systems. For instance, the Improved Squirrel Search Algorithm (ISSA) enhances the balance between exploration and exploitation in optimization problems through dynamic strategies like jumping and progressive search methods [33]. In energy management applications, hybrid and traditional nature-inspired algorithms have shown effectiveness in optimizing objectives such as emission reduction, peak demand, and operational cost, particularly in smart grids and microgrids [34].

In the context of wireless sensor networks (WSNs), energy-efficient routing protocols remain a major research focus due to the challenges posed by

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

inaccessible deployment environments. Algorithms like hybrid Falcon-ACO and Penguin Search Optimization have been employed to extend network lifespans and reduce energy usage [35], [38]. Clustering techniques, cluster head selection, and data aggregation have been further enhanced by using Earthworm Optimization Algorithm (EWA), Particle Swarm Optimization (PSO), and other metaheuristic strategies [36], [37].

Furthermore, novel routing mechanisms such as HPSOPIO integrate PSO and PIO to mitigate energy consumption in biologically inspired WSNs while improving metrics like packet delivery ratio and network lifetime [39]. In the realm of wireless power transfer, bio-inspired digital pre-distortion schemes based on algorithms like Seagull Optimization and Black Widow Optimization have been introduced to tackle hardware impairments and improve power conversion efficiency [40]. Energy optimization in smartphones is another area where nature-inspired methods have made significant contributions. The NIPO (Nature Inspired Power Optimization) system, for instance, applies PSO, ACO, and Cuckoo Search with Levy Flights to optimize OLED display power usage in Android-based devices. This system not only meets user-defined battery lifetime requirements but also demonstrates the practical applicability of bio-inspired methods in real-time environments [41]. The integration of these advanced optimization algorithms into digital system design, including components like the ALU, opens up new possibilities for achieving low-power, high-efficiency architectures. By leveraging the adaptive, distributed, and intelligent behaviours observed in natural systems, engineers can develop robust solutions to manage power consumption effectively in modern electronic devices.

A. Related Work and Research Gap

Several power optimization strategies have been explored for ALU architectures[8], [9], [10], [11], [12], [13], [14], [15]. Traditional methods such as clock gating and voltage scaling improve power efficiency but introduce latency and complexity[7], [16], [17],[18]. More recent approaches leverage machine learning and heuristic algorithms to dynamically optimize power consumption. However, these techniques often require extensive computational overhead, making them unsuitable for real-time applications.

Various heuristic algorithms, including the Squirrel Search Algorithm (SSA), Particle Swarm Optimization (PSO), and Fish Swarm Optimization Algorithm (FSOA), have demonstrated remarkable optimization potential across various domains, including digital circuit design[19], [20], [21], [22], [23], [24], [25], [26], [27]. SSA, in particular, has been widely explored for its efficient convergence behavior and adaptive search mechanisms[23]. Studies on SSA highlight its effectiveness in achieving competitive power savings while maintaining computational accuracy[28]. However, most existing research does not fully explore its limitations in comparison to other heuristic techniques. Unlike conventional power-saving techniques such as clock gating, the proposed SSA, SIA hybrid approach adaptively adjusts execution based on input variations, introducing an energy-efficient framework not previously explored in FPGA-based ALUs[29], [30], [31].

In parallel, alternative power optimization frameworks, such as distributed clock gating, have shown significant energy savings (approximately 45%-50% in 32-bit ALU designs)[16]. However, these methods do not leverage swarm intelligence principles, limiting their adaptability to dynamic input conditions. Additionally, nature-inspired approaches, such as the collapse-and-evolve model, have demonstrated power and latency improvements but are yet to be integrated with swarm-based methodologies[21], [32]. This gap indicates the need for comparative studies on hybrid optimization techniques, particularly in FPGA-based ALU architectures.

B. Research Objective and Contribution

This research work introduces SIA and SSA integration into an 8-bit ALU to achieve power reduction without sacrificing execution quality. The proposed method achieves power efficiency by integrating adaptive search from SSA and SIA transition minimization features. The contributions of this work include:

1. Introducing SSA-based conditional execution, allowing the ALU to process operations only when required, reducing unnecessary signal transitions.
2. Implementing SIA-based transition minimization,
3. ensuring that computations occur only when inputs change, effectively lowering dynamic power consumption.
4. Comparing SSA and SIA with conventional power optimization strategies, such as clock gating, to evaluate their effectiveness in reducing power dissipation.
5. Analyzing power and timing improvements by comparing optimized ALU designs, reinforcing the role of nature-inspired algorithms in FPGA-based power optimization.
6. Bridging the research gap in swarm intelligence applications for ALU designs, paving the way for future developments in hybrid optimization frameworks.

By addressing these objectives, this research underscores the significance of bio-inspired algorithms in digital circuit optimization, contributing to enhanced

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

computational efficiency and sustainability in modern computing systems. Additionally, this work highlights the growing importance of interdisciplinary approaches, integrating swarm intelligence with digital logic design to develop next-generation power-aware computing architectures.

II. Materials and Methods

A. Data Set and ALU Operations

In our proposed design, the 8-bit ALU operates on two 8-bit hexadecimal inputs, referred to as A and B, and performs a range of operations defined by a 4-bit Operation Code (Op Code). The supported operations are grouped into three primary categories. Under arithmetic operations, the ALU can perform Addition (0000) and Subtraction (0001). Logical operations include standard bitwise functions such as AND (0010), OR (0011), XOR (0100), and NOT (0101). For bit manipulation, the ALU also incorporates Left Shift (0110) and Right Shift (0111) capabilities. As illustrated in Fig. 1., the ALU architecture is modular, comprising distinct functional blocks: an Arithmetic Unit, a Logical Unit, a Shifter, and a centralized Control Unit. The Control Unit interprets the Op Code and activates the corresponding block, ensuring that only one operation is performed at a time through a Multiplexer-based selection mechanism. Additionally, the ALU produces essential status flags, such as the Zero Flag and Carry-Out, to assist in interpreting computational outcomes. This design approach ensures both clarity and control in execution, while maintaining flexibility across a wide range of operations.

outcome and accuracy of each operation. This table serves as a helpful reference for verifying the ALU's behavior and functionality across different test cases.

Table 1. Operations of conventional 8-Bit ALU design with applied inputs and outputs obtained

A (Hex)	B (Hex)	Op Code	Operation	Result (Hex)	Zero Flag	CarryOut
0F	01	0000	F + 1	10	0	0
0F	0F	0001	F – F	00	1	0
0F	03	0010	F AND 3	03	0	-
0F	03	0011	F OR 3	0F	0	-
0F	03	0100	F XOR 3	0C	0	-
0F	--	0101	NOT 1F	F0	0	-
0F	--	0110	F << 1	1E	0	0
0F	--	0111	F >> 1	07	0	1

ALU processes two 8-bit inputs, A and B, based on a 4-bit operation code, executing fundamental operations such as addition, subtraction, bitwise logical functions (AND, OR, XOR, and NOT), as well as logical shift operations. For the addition operation (Op Code: 0000), the ALU computes the sum of A and B, producing a result of 10 in hexadecimal, with no overflow (CarryOut = 0) and a nonzero output (Zero Flag = 0).

The subtraction operation (Op Code: 0001) results in 00 when subtracting two identical values, setting the Zero Flag to 1, indicating a zero result, with no borrow generated (CarryOut = 0). Logical operations such as AND (Op Code: 0010), OR (Op Code: 0011), and XOR (Op Code: 0100) correctly produce expected bitwise results (03, 0F, and 0C in hexadecimal, respectively),

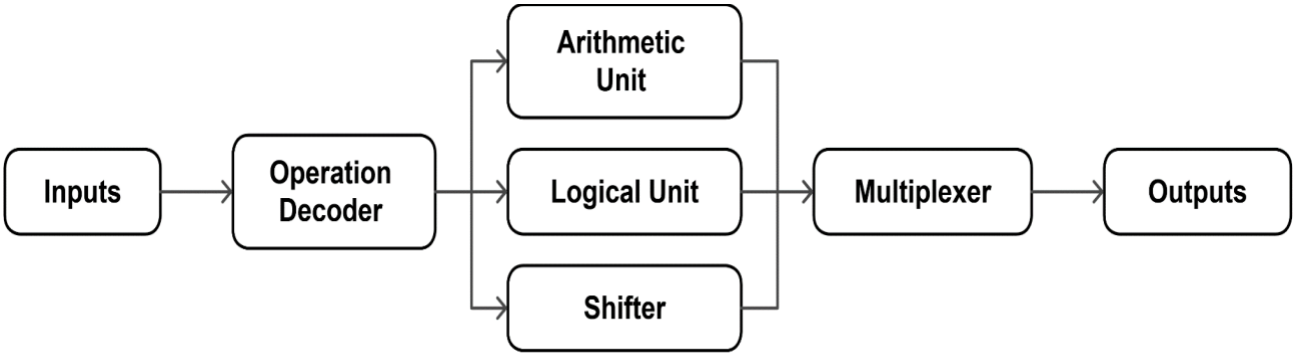

Fig. 1. Block diagram of conventional ALU design with main operational elements.

Table 1. offers a clear and organized summary of the testbench results. It displays the input combinations used, the operations performed based on their respective opcodes, and the results produced by the ALU. It also highlights the states of important flags Zero Flag and Carry-Out Flag, which help interpret the with the Zero flag remaining unset as all results are nonzero. The NOT operation (Op Code: 0101) inverts all bits of A, resulting in F0, with the Zero flag remaining 0. The shift operations demonstrate the expected behavior, where a left shift (Op Code: 0110) doubles the value (1E in hexadecimal) without setting the

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

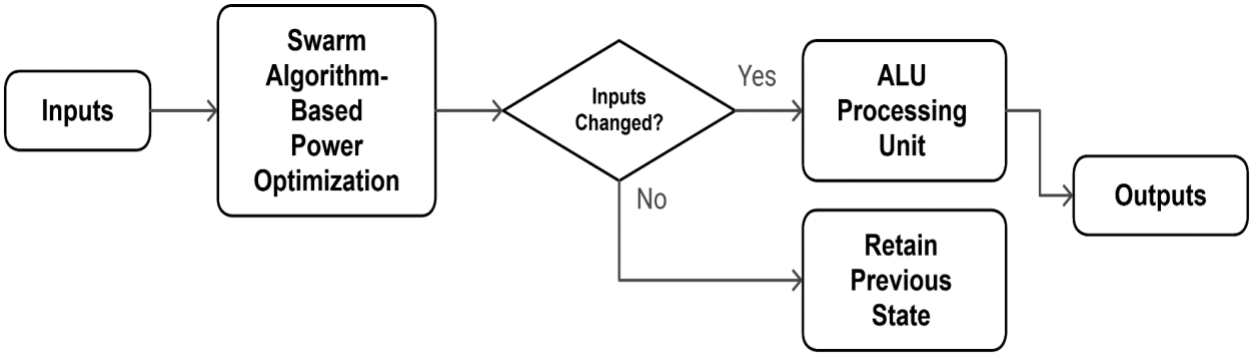

Fig. 2. Block diagram of SIA applied to conventional ALU design for dynamic power reduction.

CarryOut flag, while a right shift (Op Code: 0111) halves the value (07 in hexadecimal), capturing the least significant bit shift-out in the CarryOut flag (CarryOut = 1). These results effectively validate the ALU's capability to handle fundamental arithmetic and logical computations while correctly setting status flags based on the computed outputs.

B. Swarm Intelligence-Based Optimization

The Swarm Intelligence Algorithm (SIA) as shown in Fig. 2 further enhances power efficiency by ensuring computations occur only when inputs change.

The Swarm Algorithm-Based Power Optimization block ensures efficient ALU operation by first checking whether the inputs (A, B, Op) have changed compared to the previous state using the prev_A, prev_B, and prev_Op signals in VHDL. If the inputs remain unchanged, redundant calculations are avoided, reducing dynamic power consumption. The decision block determines whether the inputs have changed; if they have, the ALU executes the required operation, while if they remain the same, the system retains the previous state, preventing unnecessary power usage. When inputs change, the ALU processing unit performs operations such as addition, subtraction, AND, OR, XOR, NOT, shift left, and shift right based on the opcode (Op), updating the output (Result) only when computation occurs. If the inputs remain unchanged, the system holds the last computed result without reprocessing, thereby minimizing power wastage, which is implemented in VHDL by skipping execution. The output handling mechanism updates Result, CarryOut, and Zero flags only when new computations take place; otherwise, previous values are retained, effectively preventing redundant updates and optimizing power consumption.

The Swarm Intelligence Algorithm is applied in the ALU Design through the conditional check mechanism that prevents redundant computations. The key algorithm implementing this are:

Algorithm 1. Storing Previous Inputs for Power Optimization

```
Step 1  Initialize an 8-bit variable prev_A
        with all bits set to 0.
Step 2  Initialize an 8-bit variable prev_B
        with all bits set to 0.
Step 3  Initialize a 4-bit variable
        prev_Opcode with all bits set to 0.
Step 4  On each clock cycle, retrieve the
        current values of A, B, and Opcode.
Step 5  Compare current inputs with prev_A,
        prev_B, and prev_Opcode.
Step 6  If all inputs are unchanged.
Step 7  Skip the ALU operation to reduce
        unnecessary power usage.
Step 8  Else:
Step 9   Execute the ALU operation.
Step 10 Update prev_A, prev_B, and
        prev_Opcode with current values.
```

This algorithm (Algorithm 1) enables the ALU to remember previous inputs and avoid executing redundant operations when no changes have occurred. This strategy aligns with Swarm Intelligence principles, where agents (e.g., ants, birds) make efficient decisions based on memory of past environments. For example, ants avoid revisiting unproductive paths by remembering pheromone trails, and birds adjust their flight based on historical movement patterns. Similarly, the ALU conserves energy by recognizing repeated inputs and preventing unnecessary switching activity, thereby reducing dynamic power consumption.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

Algorithm 2. Checking for Input Changes

Step 1 Check if any of the following conditions are true:
A ≠ prev_A
B ≠ prev_B
Op ≠ prev_Op
Step 2 If any of the above conditions are true:
Step 3 Update prev_A with the current value of A.
Step 4 Update prev_B with the current value of B.
Step 5 Update prev_Op with the current value of Op.

This algorithm (Algorithm 2) ensures that the ALU only reacts when a change in input values occurs. If all inputs remain the same as in the previous cycle, no update is made, avoiding redundant processing and saving power. This approach is inspired by the behavior of bees in swarm intelligence systems, where scouts only search for new flower sources when the existing ones become insufficient or depleted. In the same way, the ALU minimizes unnecessary effort by executing computations only when a meaningful change occurs in the operational environment.

Algorithm 3. Avoiding Redundant Computation

Step 1 If none of the input values have changed (i.e., A = prev_A, B = prev_B, and Op = prev_Op):
Step 2 Do nothing — retain the current values of prev_A, prev_B, and prev_Op.

When the ALU detects that input values have not changed from the previous cycle, it avoids executing any operation. This prevents unnecessary logic transitions, thereby reducing dynamic power consumption. This strategy (Algorithm 3) mirrors the behavior of bird flocks in swarm intelligence systems when there are no external disturbances or threats, birds conserve energy by maintaining formation and minimizing movement. Likewise, the ALU remains idle when the computational context is stable, optimizing power usage without sacrificing performance.

Algorithm 4. Power Optimization through Conditional Execution

Step 1 Based on the value of Op, perform the corresponding operation:
Step 2 If Op = "0000" (Addition):
Step 3 Set temp_var = 0 & A_unsigned + B_unsigned.
Step 4 Assign lower 8 bits of temp_var to temp_result.
Step 5 Assign 9th bit of temp_var to carry_out_temp.
Step 6 If Op = "0001" (Subtraction):
Step 7 Set temp_var = 0 & A_unsigned - B_unsigned.
Step 8 Assign lower 8 bits to temp_result.
Step 9 Assign 9th bit to carry_out_temp.
Step 10 If Op = "0010" (AND):
Step 11 Set temp_result = A AND B.
Step 12 If Op = "0011" (OR):
Step 13 Set temp_result = A OR B.
Step 14 If Op = "0100" (XOR):
Step 15 Set temp_result = A XOR B.
Step 16 If Op = "0101" (NOT):
Step 17 Set temp_result = bitwise NOT of A.
Step 18 If Op = "0110" (Logical Shift Left):
Step 19 Shift A_unsigned left by 1 bit and assign to temp_result.
Step 20 Set carry_out_temp = original MSB of A (bit 7).
Step 21 If Op = "0111"(Logical Shift Right):
Step 22 Shift A_unsigned right by 1 bit and assign to temp_result.
Step 23 Set carry_out_temp = original LSB of A (bit 0).
Step 24 For all other cases:
Step 25 Do nothing, retain existing values of temp_result and carry_out_temp.

The ALU only executes operations when inputs change, thereby reducing unnecessary switching and power wastage. This (Algorithm 4) mimics how swarm systems dynamically allocate tasks, ensuring only necessary computations occur just as ants focus on foraging only when food sources change.

1. Hardware Architecture Modifications for SIA

The integration of SIA into the ALU design introduces several architectural enhancements to support transition-aware computation. Specifically, three 8-bit and 4-bit registers are added to store the previous cycle's values of A, B, and Op, named prev_A, prev_B, and prev_Op respectively. These are used in conjunction with comparator logic, which evaluates whether the current inputs differ from their stored counterparts. A decision control block processes the comparison results and asserts a control signal (compute_enable), allowing or bypassing ALU operation accordingly. The main ALU data path is conditionally gated based on compute_enable, ensuring the execution unit only triggers when necessary. Additionally, output control logic ensures that the result (Result[7:0]), CarryOut, and Zero flag are updated only when the operation executes, otherwise retaining their previous values. This results in a conditional execution flow that reduces unnecessary

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

toggling and thus dynamic power. These logic additions are reflected in the increased usage of flip-flops and LUTs as reported in Section III.D.

2. Configuration and Tuning Parameters for SIA Implementation

The implementation of the Swarm Intelligence Algorithm (SIA) in hardware is intentionally simplified for FPGA integration. Rather than using population-based or probabilistic metaheuristic models, the tuning of SIA is achieved through hardware control constructs.

The Block diagram in Fig. 3. illustrates the power optimization process of an ALU using the Squirrel Search Algorithm (SSA). The system receives input signals, which are processed through SSA-based power optimization to minimize energy consumption. A decision-making step evaluates whether the enable signal is active, if yes, the ALU processing units execute computations, generating outputs. If the enable signal is inactive, the system retains its previous state, preventing unnecessary power usage. This approach ensures adaptive power optimization by

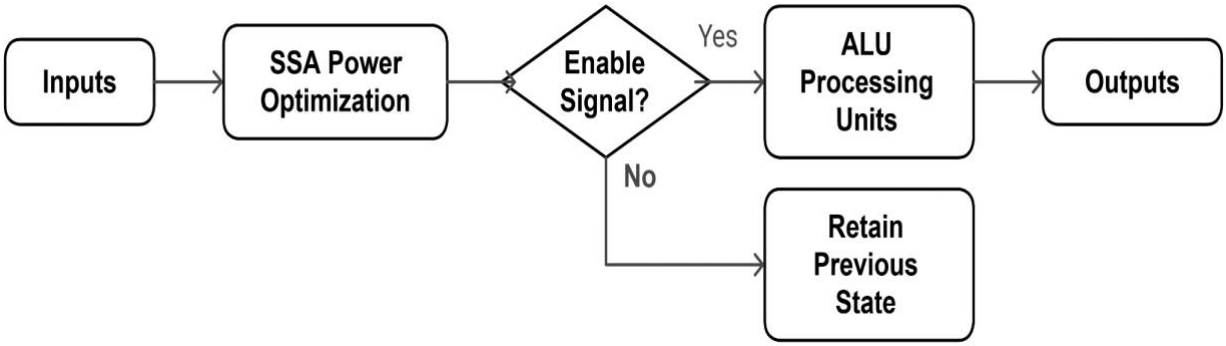

Fig. 3. Block diagram of SSA applied to conventional ALU design for dynamic power reduction.

Specifically, the prev_A, prev_B, and prev_Op registers store previous-cycle values, and their current counterparts are compared using fixed comparator logic. This comparison determines whether the compute_enable signal is asserted to trigger ALU computation.

The design ensures input change detection at the byte level for A and B, and at the nibble level for the 4-bit opcode. The configuration was validated through iterative synthesis trials to minimize unnecessary toggling and optimize power consumption without compromising functionality. No runtime-adjustable parameters or heuristic weights were used, which helps maintain deterministic and resource-efficient behavior suitable for FPGA-based environments. The optimization process guided by SIA primarily targets power minimization, while ensuring that timing slack and hardware resource usage remain within acceptable bounds.

C. SSA-Based Power Optimization

The Squirrel Search Algorithm (SSA) is an evolutionary optimization technique inspired by the foraging behavior of squirrels. In this study, SSA is integrated into the ALU to enable conditional execution and prevent unnecessary transitions, reducing dynamic power dissipation.

dynamically managing computations, leading to reduced total on-chip power consumption without compromising performance. SSA is Integrated into proposed ALU Design through the following way:

Algorithm 5. Conditional Execution to Reduce Switching Power

Step 1 Check if enable is equal to 1.
Step 2 If true:
Step 3 Check if Op, A, or B differ from their previous values.
Step 4 If any input has changed:
Step 5 Execute the ALU operation corresponding to Op.
Step 6 Update prev_Op, prev_A, and prev_B.
Step 7 Else:
Step 8 Do nothing, retain previous result to avoid unnecessary toggling.
Step 9 Else:
Step 10 Skip computation entirely, the circuit remains idle.

This algorithm (Algorithm 5) ensures that ALU operations are gated by an enable signal and only proceed if inputs have changed. It minimizes dynamic power consumption by avoiding unnecessary computation and signal toggling a common cause of energy waste in digital circuits.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

This logic aligns with the behavior of the Squirrel Search Algorithm (SSA), where squirrels adjust their foraging actions based on environmental cues and energy states. In this context, the enable signal acts like a trigger, similar to a squirrel's decision to forage only when resources or conditions demand it. This bio-inspired control helps maintain system efficiency and adaptive responsiveness in hardware systems.

Algorithm 6. Preventing Unnecessary Signal Transitions

Step 1 If reset = 1:
Step 2 Set temp_result = all 0s (reset ALU output to prevent unnecessary updates).
Step 3 Set CarryOut = 0 (clear carry out to prevent redundant transition).
Step 4 Else:
Step 5 Keep temp_result unchanged (preserve previous ALU output).
Step 6 Set CarryOut = carry_out_temp (reuse stored carry value to prevent toggling).
Step 7 Check the value of temp_result:
Step 8 If temp_result = 0:
Step 9 Set Zero flag = 1 (indicates ALU output is zero).
Step 10 Else:
Step 11 Set Zero flag = 0 (indicates ALU output is non-zero).

This algorithm (Algorithm 6) ensures the ALU avoids unnecessary changes when a reset is triggered or when retaining the same result is sufficient. By preserving previous outputs and preventing signal toggling unless strictly needed, the system significantly reduces dynamic power. This mirrors the behavior of squirrels in the Squirrel Search Algorithm (SSA), which conserve energy by minimizing movement when external conditions (like food availability or environmental risk) do not demand action. Similarly, the ALU holds state unless change is necessary, optimizing performance for low-power digital systems.

To optimize power consumption in the ALU, computations should only be triggered when necessary. Activating the ALU solely in response to valid operation codes minimizes unnecessary switching, aligning with energy-efficient conditional execution. The following algorithm illustrates this selective computation approach. Algorithm 7 ensures that the ALU performs computations only when valid operational codes are received. If no matching operation code is detected, the ALU retains its previous output, reducing unnecessary activity and switching and therefore, power consumption. This mimics Squirrel Search Algorithm (SSA) behavior, where squirrels only exert effort (such as switching trees or exploring new food zones) when a meaningful change in the environment justifies it. In the same way, the ALU avoids engaging computation hardware unless a valid Op code necessitates it, leading to smarter and more energy-efficient processing.

Algorithm 7. ALU Operations Executed Only When Necessary

Step 1 Check the value of Op to determine the ALU operation.
Step 2 If Op = "0000" (Addition):
Step 3 Extend A and B with a leading 0 to form 9-bit values.
Step 4 Compute A + B.
Step 5 Store lower 8 bits of the result in temp_result.
Step 6 Store the 9th bit (carry-out) in carry_out_temp.
Step 7 Else if Op = "0001" (Subtraction):
Step 8 Extend A and B with a leading 0 to form 9-bit values.
Step 9 Compute A − B.
Step 10 Store lower 8 bits of the result in temp_result.
Step 11 Store the 9th bit (borrow/carry-out) in carry_out_temp.
Step 12 Else:
Step 13 Do nothing, keep temp_result and carry_out_temp unchanged.

1. Hardware Architecture Modifications for SSA

The implementation of SSA in the ALU architecture involves a simpler modification compared to SIA. A global enable control signal is introduced to govern whether the ALU performs computation during a given clock cycle. If enable is deasserted, the ALU remains idle, preserving its previous output state without performing any operation. This gating mechanism is implemented via a conditional wrapper around the execution unit, which checks enable before initiating the operation based on Op, A, and B. A reset path is also included to explicitly clear the result and flags when required, preventing residual toggling. This approach eliminates unnecessary computation cycles, minimizing internal switching activity and thereby reducing dynamic power. Since the design does not require input comparison or history tracking, the resource overhead is minimal, as supported by the resource data in Section III.D.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

2. Configuration and Tuning Parameters for SSA Implementation

In the hardware abstraction of the Squirrel Search Algorithm (SSA), the conditional execution behavior is modeled through a global enable signal. This signal is asserted externally or internally based on whether the system requires ALU computation, emulating the squirrel's energy-aware response to environmental triggers. The SSA implementation does not involve iterative search, dynamic movement modeling, or weight-based decisions. Instead, it uses a gated execution mechanism with simple control logic that wraps around the ALU datapath. The control logic was configured to bypass computation when enable is low, effectively eliminating dynamic power consumption during idle cycles. This approach avoids traditional metaheuristic hyperparameters and offers a deterministic control model optimized for resource-constrained digital systems. The optimization process guided by SSA primarily targets power minimization, while ensuring that timing slack and hardware resource usage remain within acceptable bounds.

III. Results

A. Power Analysis

Power consumption was measured for three ALU configurations:

- Baseline (Unoptimized ALU)
- SSA-Optimized ALU
- SIA-Optimized ALU

Table 2. Power Analysis (in mW) of conventional 8-Bit ALU design before optimization and after optimization

ALU Design	Clocks (mW)	Signals (mW)	Logic (mW)	I/O (mW)	Dynamic Power (mW)	Static Power (mW)	Total On-chip Power (mW)
Before optimization	0.48	0.3	0.24	4.98	6	18	24
After Swarm Intelligence Algorithm	0.56	0.32	0.24	2.88	4	18	22
After Squirrel Search Algorithm	0.4	0.22	0.18	1.2	2	18	20

C. Integration into FPGA Design Flow

The SSA and SIA algorithms were integrated into the FPGA design flow using Xilinx Vivado 2023.1, targeting an AMD Spartan-7 (xc7s6csga225) FPGA operating at 100 MHz. The process began with designing a conventional 8-bit ALU in VHDL, followed by functional simulation, synthesis, and implementation to establish baseline values for power, timing, and resource utilization. Subsequently, SSA and SIA were applied sequentially to the ALU design. Each algorithm generated optimized control parameters, which were incorporated through VHDL generics and conditional logic without altering the functional behavior. These modified versions were tested using the same simulation, synthesis, and implementation steps as the baseline.

Custom XDC constraints were used to support low-power features such as clock gating and timing control. Power analysis was performed using Vivado's in-built power analyzer after implementation. This consistent, tool-supported flow allowed direct comparison of conventional, SSA-optimized, and SIA-optimized designs. The approach is portable to other toolchains by adapting constraints and configuration logic, supporting practical adoption in diverse FPGA environments.

The Table 2. and Fig. 4. presents the power reduction results of an ALU design before and after applying optimization techniques using the Swarm Intelligence Algorithm and the Squirrel Search Algorithm. Before optimization, the total on-chip power consumption was 24 mW, with dynamic power at 6 mW and static power at 18 mW. After applying the Swarm Intelligence Algorithm, dynamic power reduced to 4 mW, leading to a total on-chip power consumption of 22 mW. Further optimization using the Squirrel Search Algorithm

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

Table 3. Percentage Reduction of Dynamic Power and Total On-chip Power of ALU Design before and after optimization

ALU Design	Dynamic Power (mW)	% Reduction in Dynamic Power	Static Power (mW)	Total On-chip Power (mW)	% Reduction in Total On-chip Power
Before optimization	6	---	18	24	---
After application of Swarm Intelligence	4	33.33	18	22	8.33
After application of Squirrel Search Algorithm	2	66.66	18	20	16.66

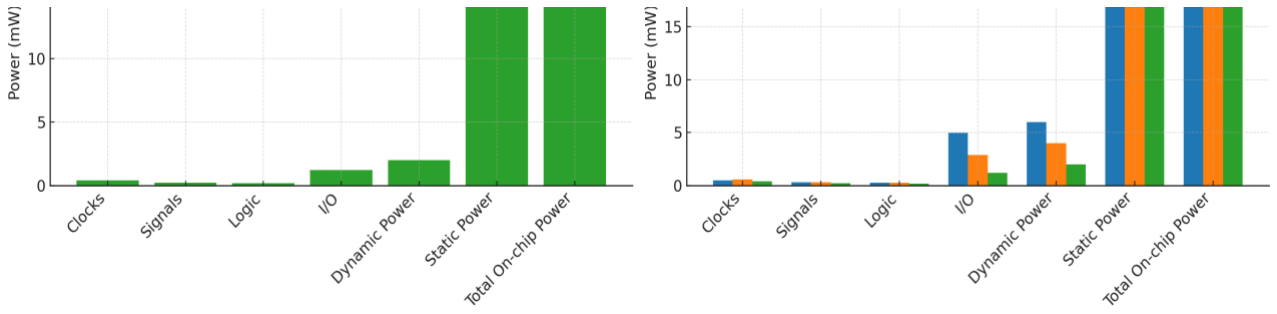

Fig. 4. a) Power dissipation of ALU before optimization b) Power dissipation of ALU after applying Swarm Intelligence Algorithm c) Power dissipation of ALU design after applying Squirrel Search Algorithm d) Power dissipation comparison across each design

resulted in a more significant reduction, with dynamic power dropping to 2 mW and total on-chip power decreasing to 20 mW. This indicates that the Squirrel Search Algorithm achieved superior power efficiency compared to the Swarm Intelligence Algorithm, making it a more effective technique for reducing ALU power consumption.

B. Functional Simulation Results

The Table 3. and Fig. 5. illustrate the power reduction effectiveness of the Swarm Intelligence Algorithm and the Squirrel Search Algorithm in optimizing ALU design. Initially, before optimization, the dynamic power consumption was 6 mW, while the total on-chip power was 24 mW. After applying the Swarm Intelligence Algorithm, dynamic power reduced to 4 mW, achieving a 33.33% reduction, while the total on-chip power dropped to 22 mW, reflecting an 8.33% reduction. The Squirrel Search Algorithm provided even greater efficiency, reducing dynamic power to 2 mW (66.66% reduction) and lowering total on-chip power to 20 mW, marking a 16.66% reduction. These results highlight

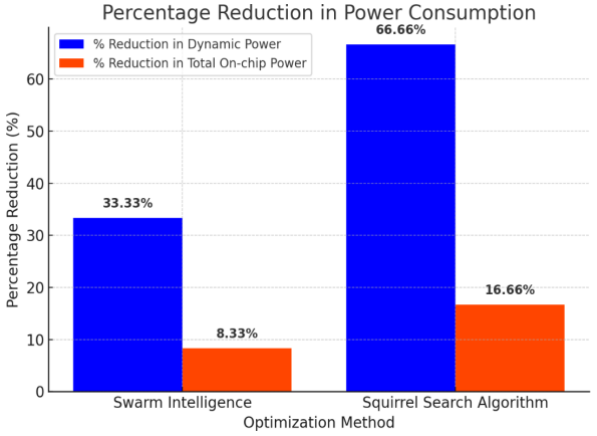

Fig. 5. Comparison of Percentage Reduction in Power Consumption of ALU Design with impact of SIA and SSA.

that while both optimization techniques improve power efficiency, the Squirrel Search Algorithm outperforms

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

the Swarm Intelligence Algorithm in minimizing dynamic and total on-chip power consumption.

The functional simulation results shown in Fig. 6. provide a comprehensive validation of the ALU's

Most importantly, this functional validation confirms that despite the significant reductions in dynamic and total on-chip power consumption, as outlined in the earlier power analysis figures, the optimized ALU

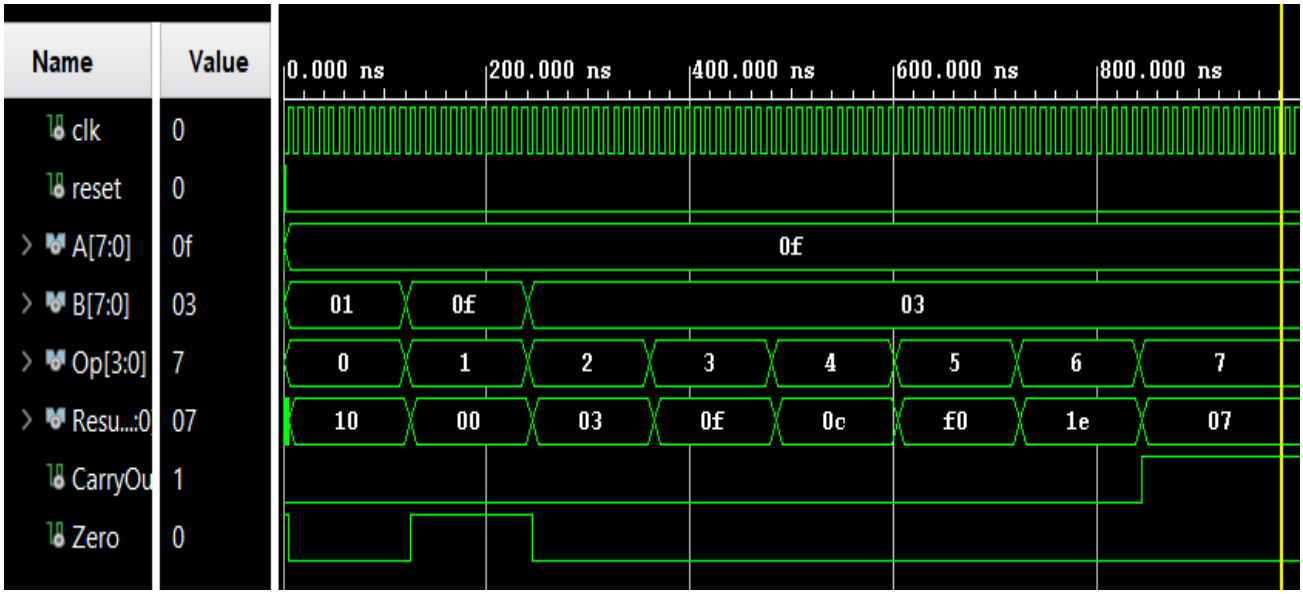

Fig. 6. Functional Simulation Result of conventional 8-Bit ALU design before and after application of SIA and SSA indicating unchanged design functionality

correct functionality following power optimization through the Swarm Intelligence Algorithm (SIA) and the Squirrel Search Algorithm (SSA). The waveform depicts a series of test vectors applied to the ALU inputs, A[7:0], B[7:0], and the operation selector Op[3:0], across multiple clock cycles. As seen in the figure, A holds a constant value of 0F, while B changes sequentially from 01 to 07, and Op increments from 0 to 7, representing various arithmetic and logic operations such as addition, subtraction, bitwise AND, OR, and shifts.

The corresponding output signal Result[7:0] accurately reflects the expected outcome of each operation. For instance, when Op = 0, the ALU might be performing addition, producing a valid result (10) for inputs A = 0F and B = 01. Similarly, in later cycles, results like 00, 0C, and F0 appear, aligning with expected logic outputs for the given operations and operands. These consistent results indicate that the core computational logic of the ALU is unaffected by the applied optimization algorithms.

In addition to correct result generation, the simulation also shows appropriate transitions in the CarryOut and Zero flags. These status flags are crucial for indicating arithmetic overflow and zero results, respectively. For example, a high CarryOut bit in certain addition operations and the Zero flag being asserted when the result is 00 serve as further proof that the control logic remains intact and fully functional.

design maintains full operational integrity. The waveform effectively demonstrates that the ALU continues to produce correct outputs under varied input conditions, thereby ensuring that power optimization does not compromise computational correctness, which is a critical requirement in low-power, high-performance digital systems.

C. Timing Analysis

The timing analysis results for the ALU design, following the application of the Squirrel Search Algorithm (SSA) and the Swarm Intelligence Algorithm (SIA), are presented in the Table 4. and visually illustrated in the accompanying Fig. 7. chart. These metrics provide not only a snapshot of the design's timing performance, but also offer valuable insight into the resilience and stability of the optimized circuits under worst-case operating conditions. In FPGA-based systems, achieving timing closure is a critical milestone, especially when targeting higher operating frequencies or integrating complex logic. The ability of an optimization algorithm to maintain or improve timing margins while reducing power consumption is a strong indicator of its practical viability. Among the evaluated metrics, the Worst Negative Slack (WNS) stands out as a key parameter that quantifies how close the design is to violating setup timing requirements. The SSA-optimized design shows a WNS of 8.740 ns, a noticeable improvement over the 6.531 ns observed

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

with the SIA-optimized version. This larger positive slack margin not only reflects better timing closure but also indicates greater immunity to timing variations due to PVT (Process, Voltage, and Temperature) fluctuations, which are especially critical in mission-critical or high-performance designs. Both optimization approaches achieve a Total Negative Slack (TNS) of 0.000 ns, confirming that there are no setup violations and that the design meets its timing requirements across all paths at the given clock constraint.

Table 4. WNS and WHS details of ALU design with SSA and SIA optimization

Metric	SSA Optimization	SIA Optimization
Worst Negative Slack (WNS)	8.740 ns	6.531 ns
Total Negative Slack (TNS)	0.000 ns	0.000 ns
Worst Hold Slack (WHS)	0.395 ns	0.250 ns
Total Hold Slack (THS)	0.000 ns	0.000 ns

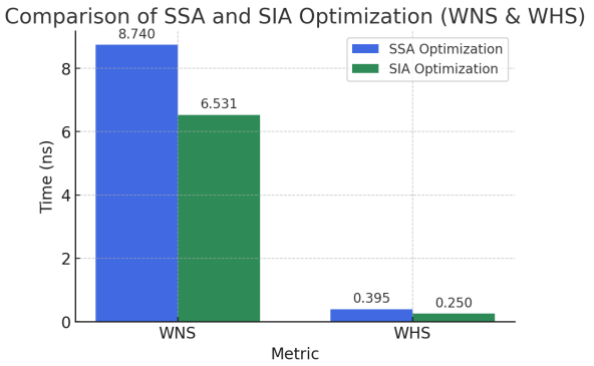

Fig. 7. Comparison Worst Negative Slack and Worst Hold Slack of ALU design for both SIA and SSA

In terms of hold analysis, the Worst Hold Slack (WHS) is also more favorable under SSA, measuring 0.395 ns versus 0.250 ns for SIA. This indicates that SSA better handles short-path conditions, which can be a common challenge in pipelined or deeply parallel architectures. The Total Hold Slack (THS) is 0.000 ns for both, showing that no hold-time violations are present in either design. Together, these results confirm that both SSA and SIA maintain timing correctness, but SSA delivers more comfortable timing margins, making the design more robust against dynamic timing uncertainties such as crosstalk, temperature drift, or slight clock jitter.

These findings are particularly significant when considered alongside the earlier power analysis, which demonstrated SSA's superior energy-saving capabilities. The combination of enhanced timing reliability and reduced power consumption makes SSA a compelling optimization approach for FPGA designs where performance, power efficiency, and operational reliability must be carefully balanced. This dual benefit is especially valuable in applications such as biomedical devices, portable health monitors, implantable systems, and real-time diagnostic platforms, where low power consumption is critical to prolong battery life, and deterministic performance is essential for ensuring accurate, timely processing of physiological signals. By achieving both energy efficiency and timing robustness, SSA-optimized designs are well-suited for modern biomedical applications that demand compact, low-power, yet high-performance computing at the edge.

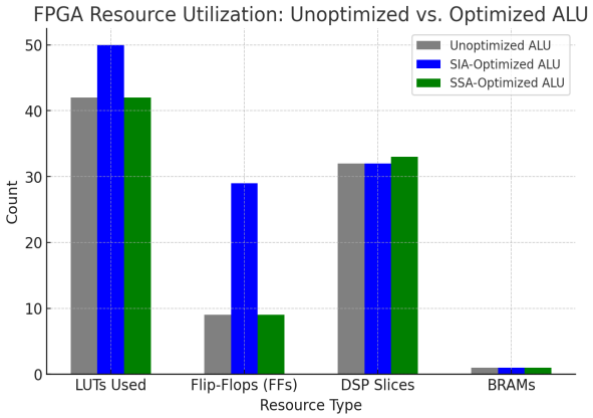

Fig. 8. FPGA Resource Utilization for conventional 8-Bit ALU, SIA and SSA optimized design

D. Resource Utilization Analysis

To evaluate the hardware efficiency of the ALU implementations after applying the Squirrel Search Algorithm (SSA) and the Swarm Intelligence Algorithm (SIA), the FPGA resource utilization was analyzed in terms of Look-Up Tables (LUTs), Flip-Flops (FFs), Digital Signal Processing (DSP) slices, and Block RAMs (BRAMs). These resources are critical indicators of how efficiently the design maps onto the FPGA fabric. The detailed results are summarized in Table 5. and visualized in Fig. 8., which compares the resource usage across the unoptimized ALU, the SIA-optimized version, and the SSA-optimized design.

From the analysis, the SSA-optimized ALU demonstrates notable hardware efficiency by retaining the same number of LUTs (42) as the unoptimized version. This indicates that the SSA method achieves power and timing improvements without increasing the logic complexity of the design, making it ideal for resource-constrained applications. In contrast, the SIA-

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

optimized ALU consumes 50 LUTs, suggesting additional logic overhead, most likely due to the inclusion of comparison modules or internal buffers used to track or store past values during the optimization process.

Table 5. Resource Utilization details of ALU design with and without optimization methods

Resource	Unoptimized ALU	SIA-Optimized ALU	SSA-Optimized ALU
LUTs Used	42	50	42
Flip-Flops (FFs)	9	29	9
DSP Slices	32	32	33
BRAMs	1	1	1

A significant difference is observed in Flip-Flop (FF) usage. While both the unoptimized and SSA-optimized ALUs utilize only 9 FFs, the SIA-optimized ALU spikes to 29 FFs. This increase can be attributed to the internal state retention mechanisms in the SIA approach, where prior inputs or intermediate states may be stored to influence current decisions. In comparison, SSA maintains a minimalist FF footprint by reducing unnecessary transitions and avoiding additional state-holding logic, thereby preserving energy and reducing switching activity. In terms of DSP slice usage, all three designs fall within a narrow range, with the unoptimized and SIA-optimized ALUs both using 32 DSP slices, and the SSA-optimized version slightly increasing to 33.

This minor increment in SSA may stem from the use of additional control or conditional execution logic required for dynamic power optimization. However, the increase is minimal and does not substantially impact the overall resource footprint, especially when weighed against SSA's power and timing benefits.

Finally, the Block RAM (BRAM) usage remains consistent across all three ALU versions, with 1 BRAM utilized. This consistency confirms that neither optimization strategy imposes extra memory demands or alters the data storage architecture of the ALU design. Maintaining a fixed BRAM count is particularly important in FPGAs with limited on-chip memory capacity.

IV. Discussion

This study provides a detailed and practical evaluation of two bio-inspired optimization techniques, Squirrel Search Algorithm (SSA) and Swarm Intelligence Algorithm (SIA), when applied to an 8-bit ALU implemented on an FPGA platform. The primary goal was to assess the impact of these techniques on dynamic power reduction, timing performance, and hardware resource efficiency, offering a benchmark for integrating swarm-based methods into real-world low-power digital circuit design.

The findings clearly demonstrate that both SSA and SIA significantly improve energy efficiency and timing performance in FPGA-based 8-bit ALUs. SSA achieved a 66.66% reduction in dynamic power, lowering consumption from 6 mW to 2 mW, with no increase in hardware resources. This gain stems from its conditional execution strategy, which triggers computation only when new input data is detected, minimizing unnecessary transitions and switching activity. In contrast, SIA delivered a 33.33% power reduction, dropping from 6 mW to 4 mW, by leveraging input-tracking mechanisms to suppress redundant processing. However, this benefit came with increased hardware usage: 50 LUTs and 29 FFs compared to 42 LUTs and 9 FFs for both SSA and the baseline. The resource overhead in SIA results from added logic for storing and comparing input states.

Table 6. Comparative Analysis of Overhead vs. Power Savings

Metric	Unoptimized ALU	SSA-Optimized ALU	SIA-Optimized ALU
Flip-Flops (FFs)	9	9	29
LUTs	42	42	50
Dynamic Power (mW)	6	2	4
Total On-Chip Power (mW)	24	20	22
Dynamic Power Savings (%)	—	66.7%	33.3%
Resource Overhead (FF+LUT)	—	0	+28 (FF+LUT)

Timing analysis reinforces SSA's advantage, showing a Worst Negative Slack (WNS) of 8.740 ns, compared to 6.531 ns for SIA. Both designs satisfied all constraints, with zero Total Negative Slack (TNS) and Total Hold Slack (THS), confirming that setup and hold conditions were met across all timing paths. SSA's superior WNS suggests greater robustness under process-voltage-temperature (PVT) variations, making it more reliable for time-sensitive or constrained applications. These results are quantitatively

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

summarized in Table 6., highlighting the trade-off between power savings and resource overhead.

The data confirms that SSA delivers the most efficient power-to-resource trade-off, offering maximum dynamic power savings with zero additional resource cost, making it highly suitable for low-power, area-constrained FPGA implementations. Meanwhile, SIA incurs an overhead of 28 logic elements, including a 19.0% increase in LUTs and a 222.2% increase in FFs. Despite this, the overall utilization remains negligible relative to the FPGA's available capacity of 3750 LUTs and 7500 FFs. Thus, SIA remains a viable option in systems where moderate energy savings, enhanced input-state awareness, and fine-grained control are desirable, and where area constraints are less critical.

These outcomes align with trends observed in related research. A low-powered, self-testable ALU design was presented in [1], focusing on energy-efficient architecture and fault detection capabilities. While the study in [1] emphasized power efficiency, it did not explore optimization through swarm intelligence or bio-inspired algorithms. Similarly, swarm-based circuit optimization was discussed in [24], where moderate improvements in area and energy efficiency were reported. However, as with [1], the work in [24] was limited to theoretical evaluation and lacked practical implementation or post-synthesis hardware validation. In contrast, the present study bridges this gap by verifying SSA and SIA-based optimizations through synthesis and simulation on an AMD Spartan-7 FPGA (xc7s6csga225), thereby demonstrating their real-world applicability.

While the results are promising, there are several limitations to consider. The analysis was confined to an 8-bit ALU, and further validation is needed to understand how these techniques scale with more complex architectures, such as 16-bit or 32-bit ALUs or digital signal processing blocks. The adaptive principles behind SSA and SIA, conditional execution and transition minimization, are conceptually scalable to larger data paths, and thus hold promise for power savings in wider architectures. Additionally, like many bio-inspired algorithms, SSA and SIA may exhibit sensitivity to initial parameter settings, which could impact convergence speed and solution quality, especially in larger or more complex architectures. Future work will explore adaptive parameter tuning and hybrid strategies to mitigate these challenges. However, our current findings indicate that scaling may introduce challenges including increased control logic complexity, higher resource utilization, and more demanding timing closure due to longer critical paths and wider input-tracking mechanisms, especially notable in the SIA implementation. To overcome these, future work should explore hierarchical or pipelined control schemes, and investigate hybrid SSA-SIA models to balance power reduction with manageable hardware overhead, ensuring practical applicability to complex digital designs.

Moreover, power measurements were performed using static testbenches; the effectiveness of SSA and SIA under dynamically varying workloads, as encountered in real-world applications, remains unexplored. SIA also introduces additional design complexity due to its reliance on input history tracking, which may complicate timing closure in larger or high-speed pipelines. Furthermore, the study lacks hardware co-simulation or empirical power measurement using physical instrumentation, which would provide more comprehensive validation of power claims under realistic operating conditions.

While our study provides valuable insights into power reduction through SSA and SIA optimization using Vivado's static power estimation tools, it is important to acknowledge limitations in measurement accuracy. The current power analysis relies primarily on post-synthesis simulation data, which may not fully capture dynamic power variations, transient switching effects, or real-world environmental influences such as temperature fluctuations and voltage noise. To enhance the fidelity and applicability of power measurements, future work should explore the integration of real-time on-chip power monitoring capabilities. This could involve utilizing FPGA-embedded power sensors or Power Monitor IP cores to collect live power consumption data under dynamic operating conditions. Additionally, coupling these with external measurement instruments or hardware-in-the-loop testing setups would provide a more comprehensive validation framework. Such approaches would enable a deeper understanding of the effectiveness of bio-inspired optimization techniques in real-world scenarios, ensuring that power savings translate accurately from simulation to practical deployment.

Despite these limitations, the implications of this work are significant. SSA, in particular, stands out as a practical solution for ultra-low-power, resource-efficient digital systems, including wearable biomedical monitors, portable diagnostics, and IoT edge-computing platforms. These systems require both tight timing constraints and low energy budgets, conditions under which SSA demonstrates clear advantages. Additionally, this research opens pathways for future exploration into hybrid optimization models. For example, combining SSA's conditional execution with SIA's input transition detection could lead to a more adaptive and fine-tuned optimization strategy. Such hybrid approaches may prove especially useful in scenarios requiring real-time responsiveness with dynamically fluctuating workloads, including AI-based edge devices and smart health monitoring systems. These bio-inspired techniques not only extend traditional low-power design methods but offer adaptive

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

capabilities that static techniques like clock gating cannot provide, making them well-suited for real-time, data-driven digital systems.

The current implementation of SSA and SIA algorithms is based on static analysis, where the optimization process is performed offline using representative input scenarios. While this enables effective design-time power minimization, the algorithms do not adapt dynamically to variations in input patterns, workload fluctuations, or environmental conditions. In practical real-time systems, such adaptability could enhance robustness and efficiency. Future research can explore integrating real-time monitoring and adaptive optimization strategies, enabling SSA and SIA to respond dynamically to operational changes.

However, integrating SSA and SIA, or extending them into more complex hybrid models, also introduces potential challenges that must be carefully considered. Combining multiple bio-inspired techniques increases the design complexity, as each optimization layer adds its own control logic and state management requirements. For instance, simultaneously implementing SSA's enable-based conditional execution and SIA's input change detection may lead to overlapping or redundant logic paths, complicating RTL design and timing coordination. Additionally, these hybrid methods may face convergence challenges, especially when optimization goals compete or interact nonlinearly, potentially requiring careful tuning of parameters and heuristics to avoid suboptimal or unstable behavior. The overhead introduced by combined logic may also impact resource utilization and timing closure, particularly in high-speed or resource-constrained environments. As current FPGA synthesis tools are primarily optimized for conventional low-power techniques, integrating hybrid swarm-based logic may demand custom synthesis constraints, testbenches, and verification strategies to ensure correct and efficient implementation. Future research should therefore focus not only on demonstrating the effectiveness of such models but also on developing scalable and maintainable frameworks that balance power savings with implementation feasibility.

V. Conclusion

The aim of this study was to design a power-efficient 8-bit ALU by applying the Swarm Intelligence Algorithm (SIA) and the Squirrel Search Algorithm (SSA) for optimization. The goal was to reduce power dissipation while maintaining performance in terms of resource utilization and timing. The main findings of the study showed a significant reduction in power dissipation with the application of both optimization algorithms. The Swarm Intelligence Algorithm (SIA) reduced dynamic power dissipation to 4 mW and the total on-chip power dissipation to 22 mW, while the Squirrel Search Algorithm (SSA) achieved even better results with dynamic power at 2 mW and total on-chip power at 20 mW. These results demonstrate a notable improvement in power efficiency compared to the unoptimized ALU, which had a dynamic power dissipation of 6 mW and total on-chip power of 24 mW.

In terms of resource utilization, the SSA-optimized ALU required fewer LUTs and flip-flops (42 LUTs and 9 flip-flops) compared to the SIA-optimized ALU (50 LUTs and 29 flip-flops). Despite this, the SIA-optimized ALU exhibited better timing performance, as indicated by a smaller Worst Hold Slack (WHS) value of 0.250 ns, compared to 0.395 ns for the SSA-optimized ALU. Both optimization methods resulted in Total Negative Slack (TNS) and Total Hold Slack (THS) values of 0.000 ns, indicating no timing violations.

Overall, both optimization methods successfully reduced power dissipation, improved timing, and utilized resources effectively, with the SSA optimization providing the best overall results in terms of power efficiency. Future research can focus on developing hybrid models that combine SSA and SIA, leveraging the strengths of both techniques to achieve enhanced power efficiency while optimizing resource utilization. By integrating the conditional execution strategy of SSA with the redundant computation minimization of SIA, a more balanced and efficient power optimization framework can be established. Additionally, further investigations can explore the scalability of SSA and SIA in larger ALU architectures, such as 16-bit or 32-bit designs, to evaluate performance trade-offs, computational efficiency, and power savings in more complex digital systems. Another promising direction involves AI-assisted adaptive optimization, where machine learning techniques can be integrated with swarm intelligence algorithms to enable dynamic, real-time power management. This would allow FPGA-based ALUs to intelligently adjust power consumption based on workload variations, leading to more energy-efficient and adaptive computing architectures.

References

[1] M. Nagharjun and V. Ravi, "Low Powered Self-Testable ALU," *IOP Conf Ser Mater Sci Eng*, vol. 1012, no. 1, p. 12048, Jan. 2021, doi: 10.1088/1757-899x/1012/1/012048.

[2] Y. Swami, "Design Guidelines for Low Power Embedded Systems using Low Power Electronics," *WSEAS TRANSACTIONS ON ELECTRONICS*, vol. 14, pp. 65–70, Nov. 2023, doi: 10.37394/232017.2023.14.8.

[3] R. Cheour, S. Khriji, D. El Houssaini, M. Baklouti, M. Abid, and O. Kanoun, "Recent Trends of FPGA Used for Low-Power Wireless Sensor Network," *IEEE Aerospace and Electronic*

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

Systems Magazine, vol. 34, no. 10, pp. 28–38, Oct. 2019, doi: 10.1109/maes.2019.2901134.

[4] S. Tamimi, Z. Ebrahimi, B. Khaleghi, and H. Asadi, "An Efficient SRAM-Based Reconfigurable Architecture for Embedded Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 3, pp. 466–479, Mar. 2019, doi: 10.1109/tcad.2018.2812118.

[5] A. Razzaq, S. R. Sani, and A. G. Ye, "Designing efficient FPGA tiles for power-constrained ultra-low-power applications," *Integration*, vol. 78, pp. 124–134, May 2021, doi: 10.1016/j.vlsi.2021.02.004.

[6] A. Shrivastava *et al.*, "VLSI Implementation of Green Computing Control Unit on Zynq FPGA for Green Communication," *Wirel Commun Mob Comput*, vol. 2021, no. 1, Jan. 2021, doi: 10.1155/2021/4655400.

[7] D. Haripriya, K. Kumar, A. Shrivastava, H. M. R. Al-Khafaji, V. Moyal, and S. K. Singh, "Energy-Efficient UART Design on FPGA Using Dynamic Voltage Scaling for Green Communication in Industrial Sector," *Wirel Commun Mob Comput*, vol. 2022, pp. 1–9, May 2022, doi: 10.1155/2022/4336647.

[8] M. A. M. El-Bendary and F. Amer, "Based on FS-GDI Approach with 65 nm Technology: Low Power ALU Design," *International Journal of Electronics*, vol. 110, no. 5, pp. 915–933, May 2022, doi: 10.1080/00207217.2022.2068195.

[9] U. Penchalaiah and V. G. S. Kumar, "Design and Implementation of Low Power and Area Efficient Architecture for High Performance ALU," *Parallel Process Lett*, vol. 32, no. 01n02, Oct. 2021, doi: 10.1142/s0129626421500171.

[10] N. K. Kabra and Z. M. Patel, "Low-Power and High-Speed Configurable Arithmetic and Logic Unit," in *Innovations in Electronics and Communication Engineering*, Springer Singapore, 2019, pp. 355–363. doi: 10.1007/978-981-13-3765-9_37.

[11] S. Thakral and D. Bansal, "High functionality reversible arithmetic logic unit," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 3, p. 2329, Jun. 2020, doi: 10.11591/ijece.v10i3.pp2329-2335.

[12] M. Vahabi, P. Lyakhov, A. N. Bahar, A. Otsuki, and K. A. Wahid, "Novel Reversible Comparator Design in Quantum Dot-Cellular Automata with Power Dissipation Analysis," *Applied Sciences*, vol. 12, no. 15, p. 7846, Aug. 2022, doi: 10.3390/app12157846.

[13] B. Safaiezadeh, E. Mahdipour, M. Haghparast, S. Sayedsalehi, and M. Hosseinzadeh, "Novel design and simulation of reversible ALU in quantum dot cellular automata," *J Supercomput*, vol. 78, no. 1, pp. 868–882, Jun. 2021, doi: 10.1007/s11227-021-03860-y.

[14] R. Roy, S. Sarkar, and S. Dhar, "Design and testing of a reversible ALU by quantum cells automata electro-spin technology," *J Supercomput*, vol. 77, no. 12, pp. 13601–13628, Apr. 2021, doi: 10.1007/s11227-021-03767-8.

[15] D. Rebecca Florance, B. Prabhakar, and M. K. Mishra, "Design and Implementation of ALU Using Graphene Nanoribbon Field-Effect Transistor and Fin Field-Effect Transistor," *J Nanomater*, vol. 2022, no. 1, Jan. 2022, doi: 10.1155/2022/3487853.

[16] R. R. Kulkarni and S. Y. Kulkarni, "Power Optimization of a 32-Bit ALU Using Distributed Clock Gating Technique," in *Advances in Electrical and Computer Technologies*, Springer Singapore, 2020, pp. 831–841. doi: 10.1007/978-981-15-5558-9_71.

[17] T. Sharma and L. Kumre, "Energy-Efficient Ternary Arithmetic Logic Unit Design in CNTFET Technology," *Circuits Syst Signal Process*, vol. 39, no. 7, pp. 3265–3288, Dec. 2019, doi: 10.1007/s00034-019-01318-4.

[18] C. Jose, T. D. Subash, and S. P. Thomas, "FPGA Implementation Of Dynamic Power, Area Optimized Reversible ALU For Various DSP Applications," *Mater Today Proc*, vol. 24, pp. 2044–2053, 2020, doi: 10.1016/j.matpr.2020.03.635.

[19] K. B. Maji, R. Kar, D. Mandal, and S. P. Ghoshal, "Optimal design of low power high gain and high speed CMOS circuits using fish swarm optimization algorithm," *International Journal of Machine Learning and Cybernetics*, vol. 9, no. 5, pp. 771–786, Oct. 2016, doi: 10.1007/s13042-016-0606-z.

[20] T. Zheng and W. Luo, "An Improved Squirrel Search Algorithm for Optimization," *Complexity*, vol. 2019, no. 1, Jan. 2019, doi: 10.1155/2019/6291968.

[21] M. Jain, V. Singh, and A. Rani, "A novel nature-inspired algorithm for optimization: Squirrel search algorithm," *Swarm Evol Comput*, vol. 44, pp. 148–175, Feb. 2019, doi: 10.1016/j.swevo.2018.02.013.

[22] M. A. Bakar, "Understanding of Collective Decision-Making in Natural Swarms System, Applications and Challenges," *ASEAN Journal of Science and Engineering*, vol. 1, no. 3, pp. 161–170, Aug. 2021, doi: 10.17509/ajse.v1i3.39637.

[23] A. D. Boursianis *et al.*, "Emerging Swarm Intelligence Algorithms and Their Applications in Antenna Design: The GWO, WOA, and SSA Optimizers," *Applied Sciences*, vol. 11, no. 18, p. 8330, Sep. 2021, doi: 10.3390/app11188330.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

[24] Q. Wu, H. Liu, and X. Yan, "An improved design optimisation algorithm based on swarm intelligence," *International Journal of Computing Science and Mathematics*, vol. 5, no. 1, p. 27, 2014, doi: 10.1504/ijcsm.2014.059382.

[25] Nallathambi, "A PARTICLE SWARM OPTIMIZATION APPROACH FOR LOW POWER VERY LARGE SCALE INTEGRATION ROUTING," *J Math Stat*, vol. 10, no. 1, pp. 58–64, Jan. 2014, doi: 10.3844/jmssp.2014.58.64.

[26] P. Kaushal, M. Khurana, and K. R. Ramkumar, "A Systematic Review of Swarm Intelligence Algorithms to Perform Routing for VANETs Communication," *ECS Trans*, vol. 107, no. 1, pp. 5027–5035, Apr. 2022, doi: 10.1149/10701.5027ecst.

[27] T. A. Khan and S. H. Ling, "A survey of the state-of-the-art swarm intelligence techniques and their application to an inverse design problem," *J Comput Electron*, vol. 19, no. 4, pp. 1606–1628, Aug. 2020, doi: 10.1007/s10825-020-01567-6.

[28] X. Zhang, K. Zhao, L. Wang, Y. Wang, and Y. Niu, "An Improved Squirrel Search Algorithm With Reproductive Behavior," *IEEE Access*, vol. 8, pp. 101118–101132, 2020, doi: 10.1109/access.2020.2998324.

[29] S. Asaithambi and M. Rajappa, "Swarm intelligence-based approach for optimal design of CMOS differential amplifier and comparator circuit using a hybrid salp swarm algorithm," *Review of Scientific Instruments*, vol. 89, no. 5, May 2018, doi: 10.1063/1.5020999.

[30] R. A. Thakker, M. S. Baghini, and M. B. Patil, "Automatic Design of Low-Power Low-Voltage Analog Circuits Using Particle Swarm Optimization with Re-Initialization," *J Low Power Electron*, vol. 5, no. 3, pp. 291–302, Oct. 2009, doi: 10.1166/jolpe.2009.1030.

[31] H. A. R. Akkar and H. S. Khairy, "Design of a Field Programmable Gate Array for Swarm Intelligent Controller Based on a Portable Robotic System: Review Study," *Journal of Cases on Information Technology*, vol. 23, no. 2, pp. 65–75, Apr. 2021, doi: 10.4018/jcit.20210401.oa6.

[32] R. Vinay and M. P. R. Prasad, "Latency and Power Improvement of Hardware Sequences Using Collapse and Evolve Approach: Nature-Inspired Methodology," in *Proceedings of First International Conference on Computational Electronics for Wireless Communications*, Springer Nature Singapore, 2022, pp. 291–302. doi: 10.1007/978-981-16-6246-1_25.

[33] Y. Wang and T. Du, "An Improved Squirrel Search Algorithm for Global Function Optimization," *Algorithms*, vol. 12, no. 4, p. 80, Apr. 2019, doi: 10.3390/a12040080.

[34] M. P. S. and L. V., "Nature-Inspired Algorithms for Energy Management Systems: A Review," *International Journal of Swarm Intelligence Research*, vol. 14, no. 1, pp. 1–16, Mar. 2023, doi: 10.4018/ijsir.319310.

[35] K. P. R. Krishna and R. Thirumuru, "Enhanced QOS energy-efficient routing algorithm using deep belief neural network in hybrid falcon-improved ACO nature-inspired optimization in wireless sensor networks," *Neural Network World*, vol. 33, no. 3, pp. 113–141, 2023, doi: 10.14311/nnw.2023.33.008.

[36] V. R. Pasupuleti and Ch. Balaswamy, "Efficient Cluster Head Selection and Optimized Routing in Wireless Sensor Networks Using Bio-inspired Earthworm Optimization Algorithm," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 11, no. 12-SPECIAL ISSUE, pp. 372–382, Dec. 2019, doi: 10.5373/jardcs/v11sp12/20193233.

[37] R. Yadav, I. Sreedevi, and D. Gupta, "Bio-Inspired Hybrid Optimization Algorithms for Energy Efficient Wireless Sensor Networks: A Comprehensive Review," *Electronics (Basel)*, vol. 11, no. 10, p. 1545, May 2022, doi: 10.3390/electronics11101545.

[38] S. Saxena and D. Mehta, "An Adaptive Fuzzy-Based Clustering and Bio-Inspired Energy Efficient Hierarchical Routing Protocol for Wireless Sensor Networks," *Wirel Pers Commun*, vol. 120, no. 4, pp. 2887–2906, May 2021, doi: 10.1007/s11277-021-08590-1.

[39] R. D. Joshi and S. Banu, "Bio-inspired wireless sensor networks - a protocol for an enhanced hybrid energy optimization routing," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 35, no. 3, p. 1808, Sep. 2024, doi: 10.11591/ijeecs.v35.i3.pp1808-1816.

[40] A. R. Nair and S. Kirthiga, "Nature Inspired Approach Toward Elimination of Nonlinearities in SWIPT Enabled Energy Harvesting Networks," *IEEE Access*, vol. 10, pp. 100837–100856, 2022, doi: 10.1109/access.2022.3208157.

[41] N. Pandey, O. P. Verma, and A. Kumar, "Nature Inspired Power Optimization in smartphones," *Swarm Evol Comput*, vol. 44, pp. 470–479, Feb. 2019, doi: 10.1016/j.swevo.2018.06.006.

Acknowledgment

We would like to thank Gujarat Technological University for the support that helped us carry out this work. The environment provided by the university played an important role in this research.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822

Author Biography

Ashish Pasaya is currently a research scholar at Gujarat Technological University, Ahmedabad, India. He received his Bachelor of Engineering (B.E.) degree in Electronics and Communication Engineering from Sardar Patel University, Anand, Gujarat, India, in 2010. He later obtained his Master of Engineering (M.E.) degree in the same discipline from Gujarat Technological University in 2012. He is a Senior Member of the IEEE, actively contributing to the academic and research community. His primary research interests include electronics, low-power VLSI design, and embedded systems. He is passionate about innovation in energy-efficient technologies and continues to engage in scholarly research and development activities.

Dr. Sarman Hadia is currently serving as an Associate Professor at Gujarat Technological University – School of Engineering and Technology, since November 2019. He earned his Bachelor of Engineering degree in 1997 from Bhavnagar University, India. Later, he completed his Master of Engineering in Communication System Engineering in 2008 from Gujarat University. He further pursued his academic career by obtaining a PhD in Electronics and Communication Engineering from CHARUSAT University, India. With a strong academic background, his research interests lie in the areas of Electronics, wireless communication, and sensor networks. He is actively involved in teaching, research, and academic development.

Dr. Kiritkumar Bhatt is currently serving as Professor and Principal at the Engineering College of Tuwa, Godhra, Gujarat, India. He earned his B.E., M.E., and Ph.D. degrees in Electronics Engineering from The M. S. University of Baroda. With over 27 years of experience in academia, he has held various leadership roles and contributed significantly to research and education. His core research areas include embedded systems, VLSI design, and low-power designs. He has successfully guided six research scholars, with three already awarded Ph.D. degrees. Dr. Bhatt has published more than 60 technical papers in reputed national and international journals and conferences. His book titled *Sand to Solar Module* is being published. He has received numerous awards and recognitions from organizations around the world. A well-recognized global speaker, he is associated with several national and international bodies as a resource person and continues to inspire through his talks and contributions.

Manuscript received March 10, 2025; Revised May 12, 2025; Accepted May 20, 2025; date of publication May 28, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i3.822