ULUSLARARASI 3B YAZICI TEKNOLOJİLERİ
VE DİJİTAL ENDÜSTRİ DERGİSİ

INTERNATIONAL JOURNAL OF 3D PRINTING
TECHNOLOGIES AND DIGITAL INDUSTRY

ISSN:2602-3350 (Online)

URL: https://dergipark.org.tr/ij3dptdi

IMPLEMENTATION OF GENETIC ALGORITHM ACCELERATOR ON FPGA FOR TRAVELING SALESMAN PROBLEM AND PERFORMING COMPARATIVE PERFORMANCE ANALYSIS

Yazarlar (Authors): Cem Deniz Kumral*

Bu makaleye şu şekilde atıfta bulunabilirsiniz (To cite to this article): *Kumral C. D., "Implementation of Genetic Algorithm Accelerator on FPGA for Traveling Salesman Problem and Performing Comparative Performance Analysis" Int. J. of 3D Printing Tech. Dig. Ind., 10(1): 176-191, (2026).*

DOI: 10.46519/ij3dptdi.1856787

Araştırma Makale/ Research Article

Erişim Linki: (To link to this article): https://dergipark.org.tr/en/pub/ij3dptdi/archive

IMPLEMENTATION OF GENETIC ALGORITHM ACCELERATOR ON FPGA FOR TRAVELING SALESMAN PROBLEM AND PERFORMING COMPARATIVE PERFORMANCE ANALYSIS

Cem Deniz Kumral [a*]

[a]Isparta University of Applied Sciences, Distance Education Vocational School, Computer Technologies Department, TURKEY

[*]*Corresponding Author: cemkumral@isparta.edu.tr*

(**Received:** 05.01.26; **Revised:** 12.02.26; **Accepted:** 15.03.26)

ABSTRACT

The primary objective of this study is to perform a comprehensive comparative performance analysis between traditional processor-based systems and hardware-oriented architectures for solving the Traveling Salesman Problem (TSP). On this way, a high-performance and scalable Genetic Algorithm (GA) accelerator was designed and implemented on a Xilinx Artix-7 Field-Programmable Gate Array (FPGA) using VHSIC Hardware Description Language (VHDL). The proposed architecture employs a Dual-Port Block RAM (BRAM) structure and a 4-stage parallel pipeline to overcome the computational bottlenecks and memory access latencies inherent in serial Central Processing Units (CPU) execution. The performance of the developed system was benchmarked against a Python-based CPU implementation using TSPLIB datasets. Experimental results reveal that the FPGA-based accelerator provides a big advantage, achieving a speedup ratio of up to 121.24x on used datasets. This comparative analysis demonstrates that the proposed hardware-based approach offers a superior, cost-effective, and deterministic alternative to software-based solutions for real-time optimization and Edge AI applications.

Keywords: Genetic Algorithm, FPGA, VHDL, TSP, Hardware Acceleration, Performance Analysis.

1. INTRODUCTION

The Traveling Salesman Problem and its variations appear in a wide range of industries ranging from logistics to data routing. The Travelling Salesman Problem and its variations are one of the most basic problems from the fact that they cannot be solved in polynomial time using classical means owing to the growth of the factorial number of alternatives with the increase in cities [1]. As the number of nodes gets larger, the solution space of such problems expands exponentially, which makes deterministic methods inadequate. To deal with such computational difficulties, Genetic Algorithms are amongst the most effective meta-heuristic methods that have been used for solving complex optimization problems [2]. The significant success of GA would stem from its capability to converge to a global optimum in a wide search space, which has been thoroughly studied in theory and practice from the past to the present [3]. However, the performance of GA on standard serial processors is limited by the demanding computational speed and data intensity of current applications. On this reason, parallel computing architectures that can reduce execution time through the algorithm's inherent parallelism have gained importance [4].

Field Programmable Gate Arrays have taken lead over Central Processing Units in terms of performance in today's Artificial Intelligence and Machine Learning applications. Comparative studies reveal that while CPUs are versatile, they do not offer the low latency and energy efficiency that FPGAs offer for large-scale matrix computations and deep learning models [5]. Research into embedded intelligence applications also showed that FPGA-based accelerators are more efficient than both CPU and GPU-based systems, especially on power-constrained edge devices [6]. An illustration of utilization of a machine

learning algorithm like Random Forest on FPGA shows an improvement in processing times due to parallelization at the hardware level [7].

The potential of FPGAs when it comes to computer science applications is no longer just theoretical; similar techniques have already been used with success in real-world applications which deal with data in high volumes like signal processing and image processing. FPGAs have proven to offer much higher frame rates in real-time implementations of an image processing algorithm than software implementations [8]. Likewise, FPGAs are an essential part of systems demanding custom hardware, e.g. parameter tuning of approximate image processing methods [9] and the design of acoustic cameras [10]. Moreover, FPGA-based stochastic VLSI architectures developed for training deep neural networks [11] and for hardware acceleration of Inception-Net based image classifiers [12] prove that complex computations can be efficiently performed on FPGAs.

Literature survey on different types of optimization studies using FPGA shows different architectures are developed for different types of problems. Research on concurrent implementation of GA using Vivado HLS shows hardware speedup leads to significant reduction in computation time [13]. FPGA-based applications of the meta-heuristic algorithms have demonstrated that, among others, a Kalman Filter that is simulated can operate on hardware [14]. Arithmetic optimization algorithms developed using chaotic maps [15] and GA-based approaches proposed for scheduling problems in heterogeneous multi-core systems [16] have been established in the literature. Furthermore, research exists on the evolution of heuristic functions in pathfinding algorithms [17] and parallel cloning-based GA architectures for digital circuit configurations [18]. Hardware architectures proposed for sequence alignment problems in bioinformatics [19] and advanced search algorithms used in motor speed control strategies [20] serve as significant examples demonstrating the flexibility of FPGAs. Additionally, automated software design for FPGAs [21], GA-based design approaches for DCT hardware accelerators [22], and advanced algorithms developed for embedded FPGA logic diagnostics [23] aim to enhance hardware reliability. Even in physical problems such as light focusing the use of FPGA-parameterized GA increases processing speed [24]. Similarly, the optimization of fuzzy controllers used in DC motor control via GA improves precision in control systems [25].

When existing studies are examined, although it is observed that GA applications on FPGA are becoming widespread, memory access bottlenecks and the lack of a fully parallel pipeline are particularly noticeable in many architectures proposed for the TSP. In this study, to address this deficiency in the literature, a hardware accelerator specialized for the TSP, implemented on a Xilinx Artix-7 FPGA and featuring a Dual-Port Block RAM architecture with a 4-stage fully parallel pipeline, is presented. The performance of the proposed architecture was analysed using three datasets of varying sizes. The primary objective of this study is to demonstrate that by executing the same Genetic Algorithm structure on both the FPGA hardware and a traditional CPU-based software environment, the proposed architecture maintains its stability despite increasing numbers of cities and achieves significant speedup compared to processor-based solutions.

This article is organized as follows: Section 2 details the proposed hardware architecture and the hybrid genetic algorithm methodology. Section 3 presents the experimental results obtained from datasets of varying scales, hardware resource utilization rates, and comprehensive comparative performance analyses against processor-based solutions. Finally, Section 4 provides the conclusions drawn from the findings along with future research objectives.

2. MATERIAL AND METHODS

This section describes the theoretical background of the algorithms and hardware technologies used and discusses the implementation of the proposed approach in software and hardware.

2.1. Overview of Genetic Algorithms

Genetic Algorithms use the principles of natural selection and genetics to provide adaptive heuristic search methods in the solution of complex optimization problems. As a

fundamental subclass of evolutionary computation, these algorithms convert the search in the solution space from randomness to an intelligent process. The GA working principle is based on "survival of the fittest," which is the evolutionary process either of biological systems [26].

The algorithm functions on a population of chromosomes, with each chromosome referring to a solution to the domain. Evolution occurs through change over the course of many generations. In every iteration a problem specific objective function is used to evaluate the fitness of the population. The most-fit individuals from the current generation are chosen as parents to transmit genes to the next generation. As stated in literature, crossover refers to the recombination of genes, and mutation refers to the random alteration of genes [27]. Using these parents, new offspring are created. This process would continue until a termination criteria was met. The termination criteria can include reaching a maximum number of generations or achieving a desired value of fitness.

2.2. Overview of FPGA Technology

Field Programmable Gate Arrays are integrated circuits programmed by users after manufacture using Hardware Description Languages (HDL). The architecture of these devices fundamentally consists of three major components: the Configurable Logic Blocks (CLBs), which perform logical operations; Input/Output (I/O) cells, which communicate with the outside world; and programmable interconnect matrices, which handle the data transfer between these components [28]. This adaptable internal architecture allows for the physical configuration of the hardware layout of a digital circuit.

The greatest benefit of FPGAs as compared to CPUs is how they process information in different ways. According to the Von Neumann architecture, processors execute the instruction sequence one after another. In contrast, FPGAs enable the spatial distribution and full parallel execution of algorithms to the degree offered by the available hardware. As indicated in the literature, this feature of hardware parallelism offers designers more flexibility in terms of performance and reconfigurability than processor-based systems, especially in high-speed control systems and data-intensive applications [29].

2.3. Proposed Methodology

In this study, the proposed system was initially developed as a reference model within a software environment (Python). Subsequently, based on the algorithmic parameters and logical structure derived from this model, it was physically implemented on FPGA hardware.

2.3.1. Software implementation and datasets

To evaluate the accuracy, stability, and performance of the proposed hardware architecture, TSPLIB, recognized as the standard benchmark library for the Traveling Salesman Problem in the literature, was adopted as a reference [30]. In the experimental studies, three datasets of varying complexity were selected to analyse the scalability of the algorithm with respect to the increasing number of nodes: the small-scale (29 city) Western Sahara (wi29), the medium-scale (51 city) Eisenhardt (eil51), and the large-scale (70 city) St70 (st70). These datasets comprise the coordinates of cities in 2D Euclidean space and served as the common data source for both approaches during the comparative analysis of hardware and software implementations.

The experimental setup for software-based implementation was built on a mid-range PC to simulate a standard affordable PC. The system incorporates an Intel Core i5-12500H system-on-chip that boasts a powerful 12 cores and runs a Windows 11 Pro operating system. The algorithms were implemented using Python 3.9, with standard libraries such as NumPy for matrix operations and Matplotlib for visualizations. It does not use any multi-threading acceleration. To offer a reasonable benchmark against the Xilinx Artix-7 (XC7A15T), which is also an entry-to-mid-level FPGA solution, this configuration was selected to ensure that the comparison is realistic for available software and hardware.

Before the hardware design, a reference model using Python was created to test the working of the Genetic Algorithm and evaluate its efficiency as well as the performance of the algorithm on the standard CPU. The purpose of the software model is to create a performance baseline that allows measuring the speedup ratio and solution quality of the FPGA-based

accelerator in an objective manner. Given the nature of the TSP, a permutation encoding structure was adopted by the model. Each chromosome was a non-repeating ordered city. After the random generation of the initial population, the value of fitness was calculated as the reciprocal of the total distance of the route.

Classical GA operators were supported with Local Search techniques to establish a hybrid architecture. The specific operators employed are as follows:

- Selection: The Tournament Selection method was implemented due to its low computational cost and the feasibility of easy realization using parallel comparators on the hardware.
- Crossover: The Order Crossover (OX1) operator was preferred to maintain the validity of the generated routes and to prevent node repetition.
- Mutation: The Inversion Mutation operator was utilized to maintain population diversity and prevent premature convergence. This operator works by reversing a sub-sequence of the tour between two randomly selected points.
- Elitism: By preserving the top 10% of the population in each generation, a monotonic increase in solution quality was guaranteed.
- Local Search: To accelerate the convergence rate of the algorithm, 2-Opt optimization was integrated. This procedure is applied periodically (every 50 generations) and at the end of the simulation to the best individual, thereby eliminating crossing paths in the route.

Algorithm 1 presents the pseudocode showing the algorithmic flow and processing steps of the hybrid reference model developed for use in CPU performance analysis.

Algorithm 1. Algorithm structure for CPU.
Input: Distance Matrix (D), Pop_Size (N), Max_Gen (G)
Output: Best_Route (R_best), Min_Cost (C_min)

```
1:  Initialize Population P with random permutations
2:  Evaluate Fitness for all individuals in P
3:  Sort P and initialize R_best
4:  gen <- 0
6:  WHILE gen < G DO:
7:      P_new <- Empty List
8:      Add Elites (Top %10) to P_new      // Elitism
10:     // Create New Generation
11:     WHILE Size(P_new) < N DO:
12:         p1, p2 <- Tournament_Selection(P)
13:         child  <- Order_Crossover(p1, p2)
14:         child  <- Inversion_Mutation(child)
15:         Append child to P_new
16:     END WHILE
18:     P <- P_new
20:     // Periodic Local Search (Hybrid Approach)
21:     IF (gen % 50 == 0) THEN
22:         Apply_222:  Best_Individual_in_P)
23:     END IF
25:     Evaluate Fitness(P) and Update R_best
26:     gen <- gen + 1
27: END WHILE
29: // Final Refinement
30: R_best <- Apply_2Opt(R_best)
31: Return R_best, Min_Cost(R_best)
```

The optimization algorithm starts with a population of random permutations, as indicated in the pseudo-code provided. The elitism mechanism preserves the top 10% of the population in each generation for use in the next generation. The generation of offspring is done using the Order Crossover (OX1) and Inversion Mutation operators on parents chosen using the tournament method. The unique hybrid structure of the algorithm applies the local search method 2-Opt to the best individual for every 50 generations and at the process end. Using local improvements that genetic operators may overlook, this approach improves global convergence toward finding the best one. Specifically, the control parameters were configured as follows: a Crossover Rate of 0.95, a Mutation Rate of 0.01, and a Tournament Size of 5. These values were selected based on preliminary tests to maintain an optimal balance between exploration and exploitation.

2.3.2. Hardware implementation on FPGA

The implementation of the proposed hardware architecture was performed using VHDL on Xilinx Optimized Artix-7 family xc7a15tcpg236-1 FPGA chip through the Vivado Design Suite 2021.1 environment. The primary aim of the design is to reduce memory access latency and process genetic operators in a fully parallel way as against processor based. The system is designed as a modular structure managed by a central Finite State Machine (FSM). The FPGA's internal Dual-Port Block RAM resources were leveraged to hold population data. Due to the ability of this memory architecture to access the two memory addresses simultaneously in the same clock cycle, the ‘read-modify-write’ loop is accelerated, which is important for mutation

and crossover operators. In addition, a 4-stage fully parallel pipeline architecture was integrated to maximize the throughput during the fitness evaluation phase. This specialized structure enables the retrieval of inter-city distances from ROM units and their accumulation in the DSP slices at every clock cycle, effectively neutralizing the computational overhead as the number of cities increases. For Random Number Generation (RNG), a 32-bit Linear Feedback Shift Register (LFSR) was preferred due to its low hardware cost and long period. Furthermore, distance calculations were performed using high-performance DSP48 slices on the FPGA. The general architecture and data path organization of the system are illustrated in the block diagram in Figure 1, while the hardware flow control and processing steps are presented in Algorithm 2.

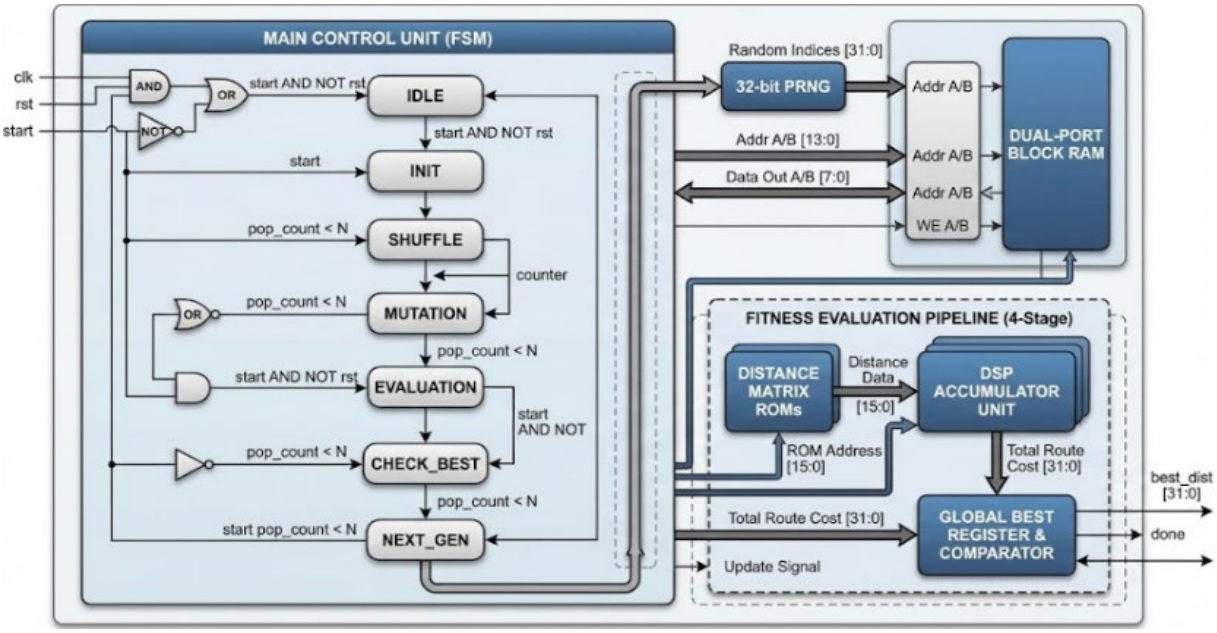

Figure 1. FPGA hardware implementation block diagram.

Algorithm 2. Algorithm structure for FPGA.

Input: Clock (clk), Reset (rst), Distance_ROMs (LUTs)
Internal: Dual-Port Block RAM (Population), 32-bit LFSR (RNG), DSP Unit

```
1:  PROCESS (Positive Clock Edge):
2:     CASE State IS:
3:        -- INITIALIZATION PHASE --
4:        WHEN S_INIT:
5:           Sequential Write: Fill RAM with City Indices [0..N-1]
7:        -- GENETIC OPERATORS PHASE (Parallel Access) --
8:        WHEN S_MUTATION / S_SHUFFLE:
9:           PARALLEL BLOCK:
10:             Thread 1: RNG generates Random Indices (Addr_A, Addr_B)
11:             Thread 2: FSM requests Read Access to RAM (Port A & Port B)
12:          END PARALLEL;
14:          -- Dual-Port RAM Read (1 Clock Cycle) --
15:          Val_A <= RAM[Addr_A];  Val_B <= RAM[Addr_B];
17:          -- Swap Operator (Combinational Logic) --
18:          Swap(Val_A, Val_B);
20:          -- Dual-Port RAM Write (Update Population) --
21:          RAM[Addr_A] <= Val_B;  RAM[Addr_B] <=  Val_A;
23:       -- FITNESS EVALUATION PHASE (4-Stage Pipeline)
24:       WHEN S_EVALUATION:
25:          LOOP (For all cities in individual):
26:             Stage 1 (Fetch): Generate Address for City_i and City_i+1
27:             Stage 2 (Read):  Retrieve City IDs from Block RAM
28:             Stage 3 (Lookup): Access ROM for Distance(City_i, City_i+1)
29:             Stage 4 (Accumulate): DSP Unit adds Distance to Total_Cost
30:          END LOOP;
32:          -- Comparator Logic --
33:          IF Total_Cost < Best_Global_Register THEN
34:             Best_Global_Register <= Total_Cost; -- Upd Best
35:          END IF;
37:       -- CONTROL UNIT --
38:       WHEN S_NEXT_GEN:
39:          Increment Generation_Counter;
40:          IF Generation_Counter == MAX_GEN THEN State <= S_DONE;
41:          ELSE State <= S_MUTATION; -- Loop Back
42:          END IF;
43:    END CASE;
44: END PROCESS
```

As detailed in Algorithm 2, the hardware workflow is distinguished by its pipeline structure, particularly during the fitness evaluation phase. The '4-Stage Fully Parallel Pipeline' (Lines 23-30) ensures that the distance calculation of a route is executed with maximum throughput. In Stage 1 (Fetch), memory addresses for consecutive cities are generated. Stage 2 (Read) retrieves city indices from the BRAM. In Stage 3 (Lookup), inter-city distance data is fetched from pre-calculated ROMs (Read-Only Memories). Finally, in Stage 4 (Accumulate), the total distance is

computed using the DSP unit. This pipelining technique effectively neutralizes the complexity introduced by an increasing number of cities by enabling the processing of a new distance value at every clock cycle. Furthermore, the S_MUTATION state (Lines 8-21) leverages the dual-port nature of the memory to execute the swap operation within minimum clock cycles, thereby demonstrating the superiority of hardware-level parallelism.

The hardware datapaths dedicated to the genetic operators are designed to achieve maximum parallelism. In contrast to function calls in software which occur in sequence, these operators utilize special memory addressing and logic techniques to execute very fast, as mentioned below.

Mutation Operator (Swap Logic): In the Python reference model, the Inversion Mutation operator was used due to the robustness of the algorithm. In FPGA, however, the Swap Mutation operator was preferred. By leveraging the Dual-Port feature of BRAM, this strategic modification reduces the clock cycles. The RTL circuit diagram of the Swap Mutation operator created in the VHDL design is given in Figure 2.

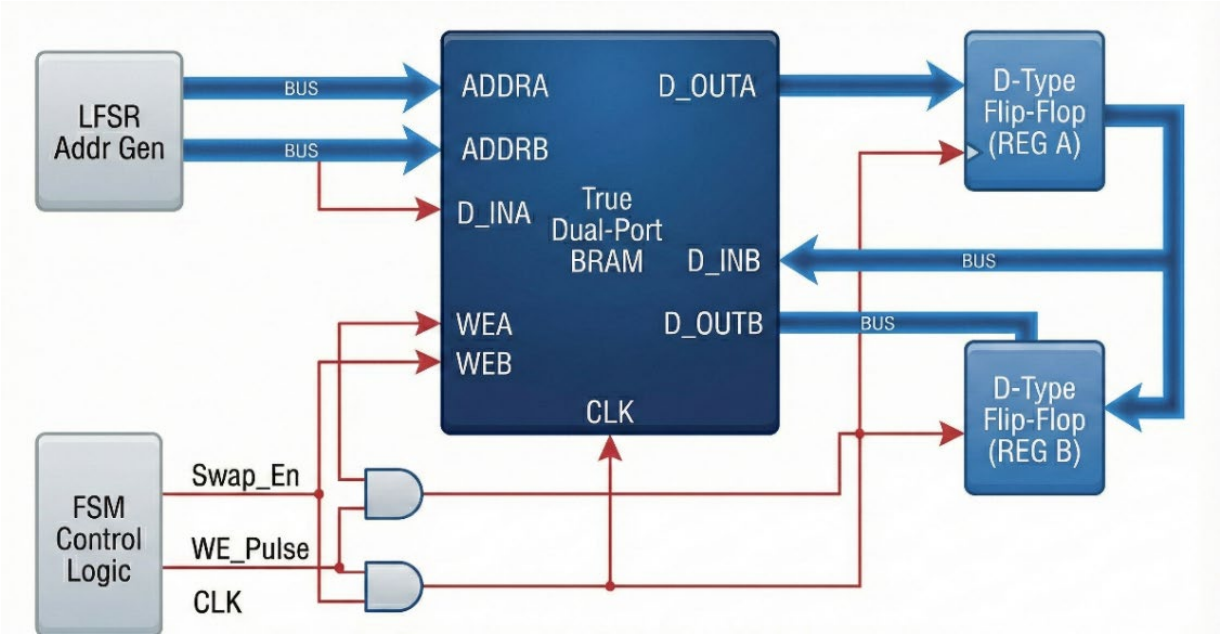

Figure 2. RTL schematic of the Swap Mutation operator.

As detailed in the RTL schematic, the mutation process is orchestrated by a Finite State Machine (FSM) that interacts with a True Dual-Port Block RAM to execute the operation in constant time complexity. Upon the generation of two distinct random addresses (Addr_A and Addr_B) by the LFSR unit, the genes located at these addresses are read simultaneously from Port A and Port B in the first clock cycle. In the subsequent write cycle, the data lines are physically cross-wired within the FPGA logic fabric; the data read from Port A is routed directly to the data input of Port B and conversely, the data from Port B is routed to Port A. This direct routing technique eliminates the need for intermediate storage registers, allowing the swap operation to be completed instantly without CPU intervention.

Crossover Operator (Order Crossover Logic): The Order Crossover is implemented as a sequential hardware process to ensure the generation of valid Hamiltonian cycles. Unlike the mutation operator, this module requires validity checking logic to prevent duplicate cities in the offspring tour. The RTL circuit diagram of the Order Crossover operator created in the VHDL design is given in Figure 3.

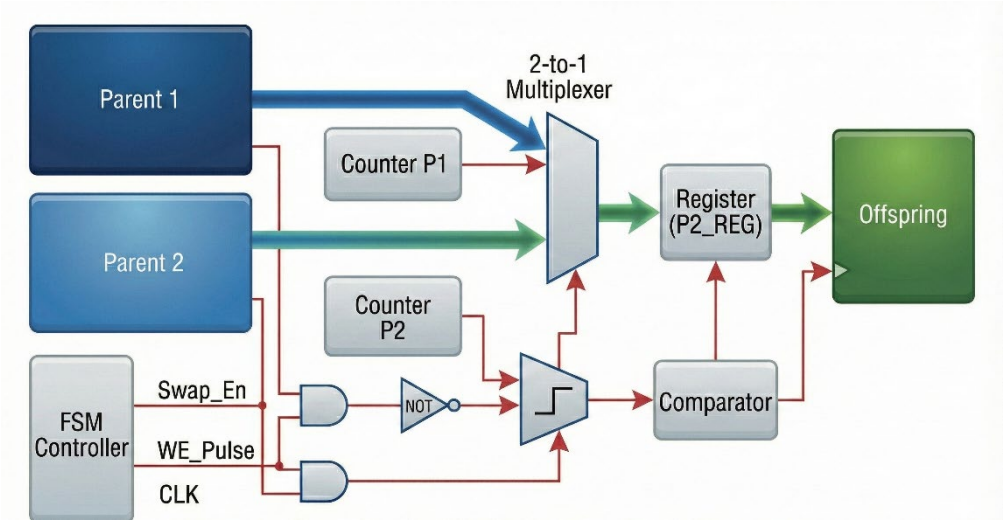

Figure 3. RTL schematic of the Order Crossover operator.

As illustrated in the RTL schematic, the data flow of the crossover operator is managed by the FSM through two sequential phases to guarantee valid Hamiltonian cycles. In the initial Segment Copy phase, a specific sub-sequence of cities is transferred directly from Parent 1 to the Offspring BRAM using synchronized counters. Subsequently, in the Fill and Validate phase, the system iterates through the gene sequence of Parent 2 while a hardware Comparator Unit operates in parallel to verify the existence of the current city within the Offspring memory. Based on this real-time comparison, the Write Enable (WE) signal is conditionally activated; if the comparator signals that the city is missing (logic '0'), the signal triggers a write operation to the next available slot, whereas the signal is suppressed if the city already exists (logic '1'). This hardware-level filtering instantly prevents duplication ensuring strict adherence to TSP constraints without requiring iterative software checks.

To ensure that benchmarking was performed in an equitable environment, the computational cost of generating distance matrix was carefully excluded from the run-time of both platforms. In the Python implementation, the distance matrix is treated as off-line pre-calculated data. Its generation time is not included in the execution time. In a similar vein, to guarantee the proper functioning of the hardware and to ensure data consistency, the same distance matrices generated in Python were translated into VHDL packages (.vhd files) and instantiated as ROM blocks for each dataset. As a result, during this phase, both architectures use a specialized form of memory lookup for evaluating fitness rather than performing the complex, derivative computations of Euclidean distance in coordinate space at runtime. The timing measurements did not include any additional operations related to the software–file parsing and to the hardware UART communication.

3. RESULTS AND DISCUSSION

3.1. PC-Based Implementation Results

A Python-based reference implementation was first developed on a PC environment in order to showcase the acceleration performance obtained by the proposed hardware-based GA design. The software implementation aims to find differences in execution time with FPGA results and carry out a performance analysis in this section. Moreover, through this process, the algorithm's logical correctness and solution quality were guaranteed before hardware implementation took place. For the comparative analysis, TSP datasets with 29, 51 and 70 cities were used, and results of convergence and runtime time of algorithms were obtained.

For the small-scale WI29 dataset, the algorithm exhibited rapid convergence, reaching the optimum solution with a cost value of 27,603 at the 84th generation. The total execution time required to complete 1,500 generations for this dataset was recorded as 13,710 ms. The results are presented in milliseconds to facilitate a more comprehensible comparison with the execution times on the FPGA side. The convergence results obtained for the wi29 dataset are presented in Figure 4.

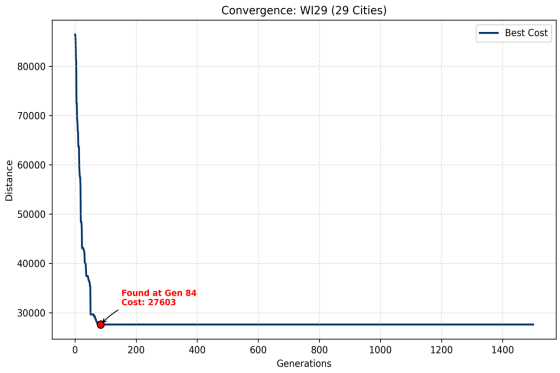

Figure 4. Convergence graph for wi29.

For the medium-scale eil51 dataset, the search process progressed steadily, and the best cost value of 440 units was achieved at the 1143rd generation. The total execution time to complete 2,000 generations for this dataset was recorded as 39,120 ms. The convergence results obtained for the eil51 dataset are presented in Figure 5.

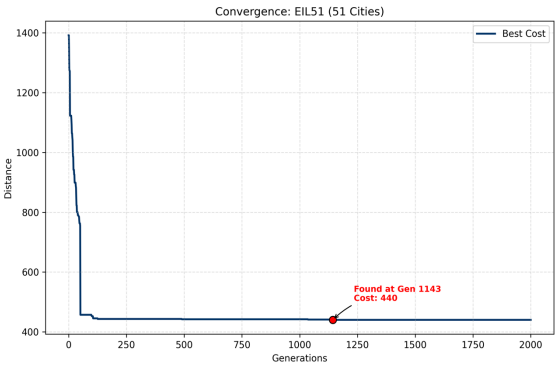

Figure 5. Convergence graph for eil51.

When the large-scale ST70 dataset is examined, it is observed that the solution quality improved steadily over the generations despite the increase in problem size. This optimization process was completed by reaching the best cost value of 683 units at the 3840th generation, with the algorithm exhibiting a stable structure. The total execution time to complete 4,000 generations for this dataset was recorded as 109,120 ms. The convergence results obtained for the ST70 dataset are presented in Figure 6.

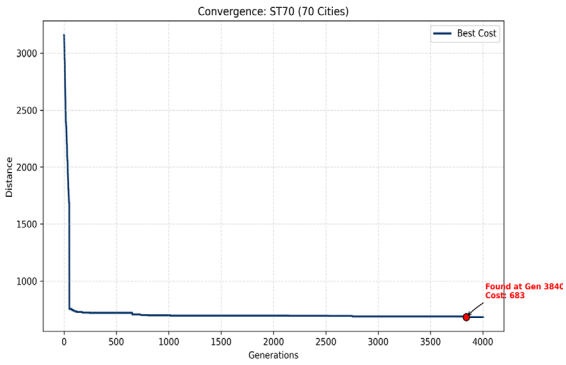

Figure 6. Convergence graph for st70.

In addition to the convergence analyses, the outputs of the algorithm were visualized, and the best tour routes were successfully generated for each test scenario. Upon examining these obtained routes, it was verified that the connections between nodes constitute valid Hamiltonian cycles and that the routes are geometrically optimized. These findings demonstrate the consistency of the Python-based reference design and establish a solid foundation for the performance comparison. The best tour routes obtained for each dataset are presented in Figures 7-9.

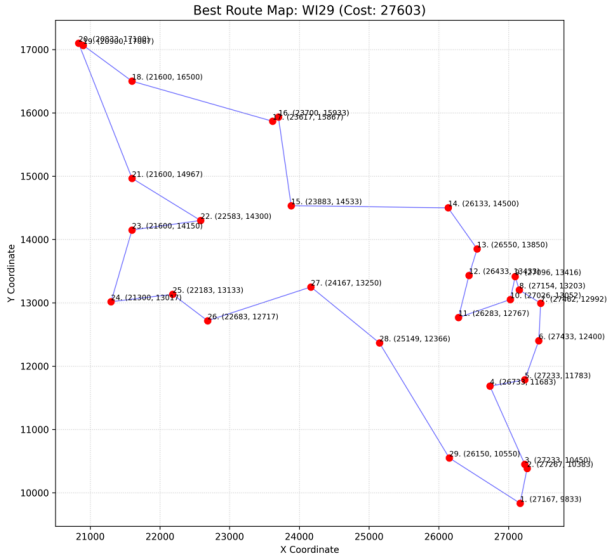

Figure 7. Best tour route for wi29.

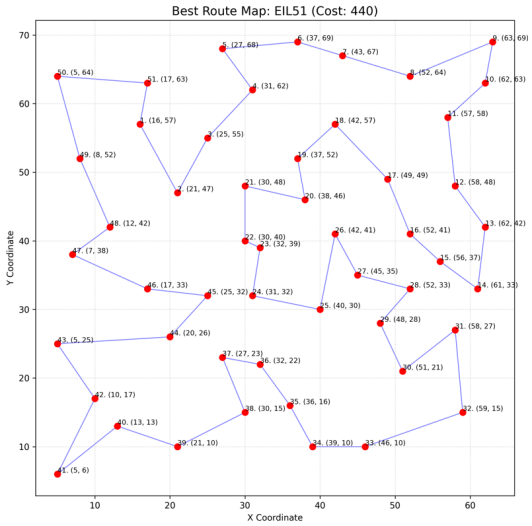

Figure 8. Best tour route for eil51.

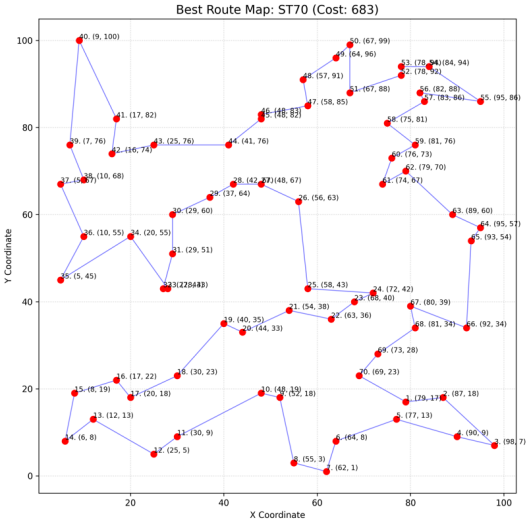

Figure 9. Best tour route for st70.

3.2. FPGA Implementation Results

The performance and resource utilization of the proposed hardware architecture were evaluated within the Vivado Design Suite environment, targeting the Xilinx Artix-7 (xc7a15t) FPGA chip.

3.2.1. Algorithm simulation results

The physical synthesis and place & route processes of the design were successfully completed, and it was verified that the system operates at a frequency of 100 MHz (10ns period) without violating any timing constraints. Performance measurements were conducted with clock cycle precision using post-implementation timing simulations, which account for the physical delays of the design. The obtained simulation waveforms reflect the actual execution times on the hardware due to the deterministic nature of the FPGA. Thanks to the 4-stage pipeline architecture, it has been recorded that operations taking seconds on the processor (PC) side are completed within the order of milliseconds on the FPGA for each dataset. This confirms that the FPGA-based design delivers pure hardware performance, completely devoid of (OS overhead) and context-switching costs.

WI29 Simulation: In the small-scale dataset, the hardware completed the process of 1,500 generations in exactly 199.965 ms by consuming 19,996,500 clock cycles. The algorithm updated the best_dist register by reaching the optimum cost of 27,603 at the 25,329 ms.

EIL51 Simulation: In the 51-city problem, the computational load of 2,000 generations was processed in 500.020 ms. It is observed that the optimum cost of 446 was achieved at the 225.259 ms instant of the simulation.

ST70 Simulation: In the 70-city scenario, which possesses the highest computational load, the completion of 4,000 generations took only 900.040 ms. The fact that this process, which took approximately 109 seconds on the PC side, was completed in under 1 second on the FPGA demonstrates the success of hardware acceleration. The best cost value of 687 was achieved at the 574.000 ms instant of the simulation. The FPGA simulation results obtained for the three datasets are presented in Figures 10-12.

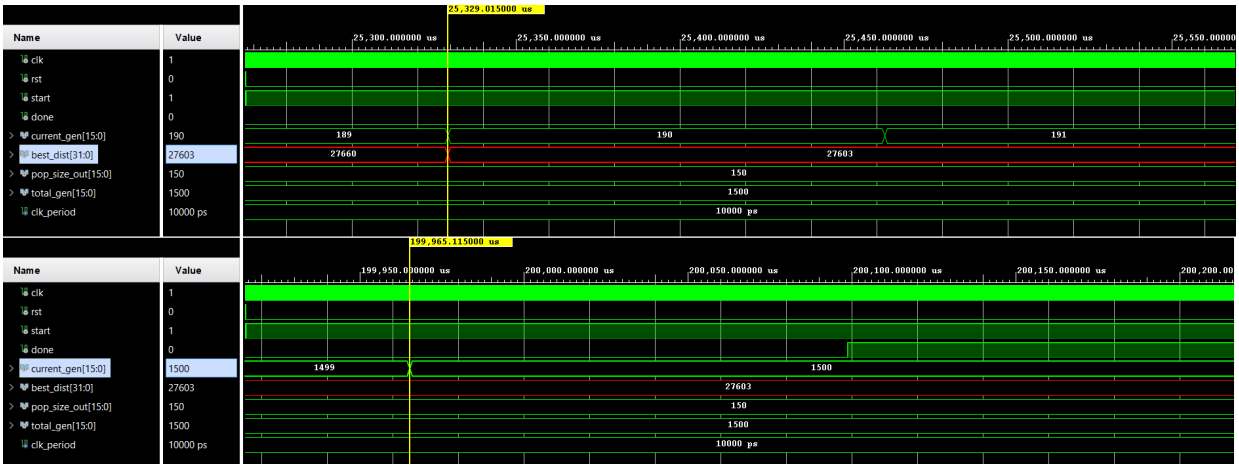

Figure 10. FPGA simulation results for wi29.

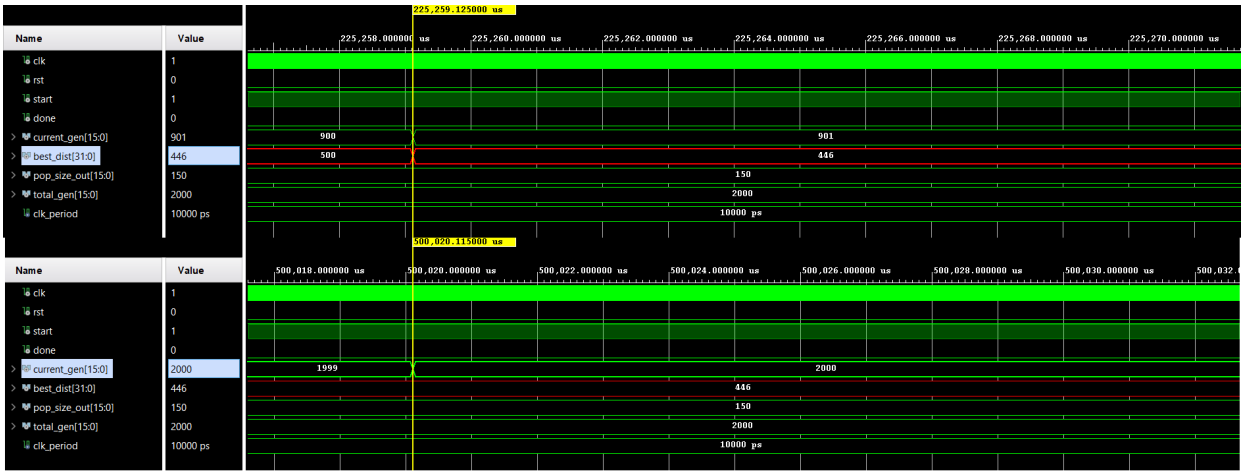

Figure 11. FPGA simulation results for eil51.

Figure 12. FPGA simulation results for st70.

When comparing the simulation results with the PC-based reference values, it was observed that the results matched exactly for the small-scale WI29 dataset (27,603), whereas marginal deviations of approximately 1.3% and 0.6% occurred in the FPGA outputs for the larger EIL51 and ST70 datasets, respectively. The best value was recorded as 446 on hardware for EIL51 (Reference: 440) and 687 for ST70 (Reference: 683). The primary reason for this is the preference for an integer-based data path instead of the high-precision floating-point arithmetic used in the PC environment to maximize throughput and minimize latency within the FPGA architecture. Furthermore, the structural differences between the stochastic

properties of the LFSR (Linear Feedback Shift Register)-based random number generator used on hardware and the Python-based Mersenne Twister algorithm minutely affected population diversity in vast search spaces. Quantitatively, the average deviation across all tested datasets is calculated as approximately 0.65%, with a worst-case deviation of only 1.36% observed in the EIL51 dataset. Considering the speedup achievement of over 100 times in computation times, this marginal difference in solution quality is evaluated as an acceptable 'speed-precision trade-off' widely recognized in hardware-accelerated heuristic studies [9, 13].

3.2.2. FPGA resource utilization

The resource utilization values obtained as a result of the physical implementation of the proposed hardware architecture were obtained from the post-implementation reports generated by the Vivado Design Suite. To analyse hardware costs under the most demanding scenario, the data presented in Table 1 were derived based on the ST70 (70 cities) configuration, which is the largest-scale dataset used in the study. It was observed that resource consumption is lower than these presented values for datasets with lower complexity, such as WI29 and EIL51. The summary of these maximal hardware costs on the targeted Xilinx Artix-7 (xc7a15t) FPGA chip is presented Table 1.

Table 1. FPGA resource utilization.

Resource Element	Available	Used	Utilization (%)
Slice LUTs (Logic)	10,400	4,295	41.30 %
Slice Registers (FF)	20,800	472	2.27 %
LUTRAM (Distributed RAM)	9,600	3,072	32.00 %
Block RAM (BRAM)	25	2	8.00 %
DSP48E1 (Arithmetic)	45	4	8.89 %
BUFG (Clock Buffers)	32	1	3.13 %

Upon examining the post-synthesis resource utilization data presented in Table 1, it is observed that the proposed architecture exhibits a highly efficient layout, even on the entry-level XC7A15T FPGA. The utilization of Slice LUTs, where logical operations are executed, occurred at 41.30%; this reflects the combinational logic density required by the genetic operators (crossover, mutation) and address generation units. Conversely, the fact that Slice Register (Flip-Flop) usage remained at a negligible level of 2.27% indicates that the sequential complexity of the design has been minimized and the pipeline structure has been efficiently constructed.

The most critical finding regarding the scalability of the design is that the Block RAM (BRAM) utilization is only 8.00% (2 units). This low memory consumption demonstrates that a substantial reserve area remains on the chip for storing population data and distance matrices. Consequently, the system possesses the flexibility to be adapted to much larger problem sizes (e.g., 200+ cities) or larger populations without necessitating hardware modifications. Similarly, the utilization rate of DSP48E1 blocks used in fitness function calculations remained limited to 8.89% (4 units). This resulting compact hardware footprint confirms that the proposed GA accelerator can be integrated as a low-cost co-processor in resource-constrained edge devices (Edge AI) and embedded systems.

3.3. Comparative Performance Analysis

The performance data of the Python-based reference implementation and the FPGA-based hardware accelerator were compared based on the execution time parameter.

The logarithmic scale graph presented in Figure 13 illustrates the timing performance exhibited by both platforms depending on the dataset size. Upon examining the graph, it is observed that as the number of cities and computational load increase, the FPGA solution keeps the processing time in a much narrower band, providing a significant speed advantage over the processor-based solution. While the execution times on the PC side rise with a steep momentum as the dataset expands, the fact that the increase on the FPGA side remains much more controlled proves the operational stability

of the hardware-based architecture. This divergence on the logarithmic scale demonstrates that the bottlenecks encountered by software-based solutions on processors in large-scale problems are effectively overcome with the proposed hardware accelerator.

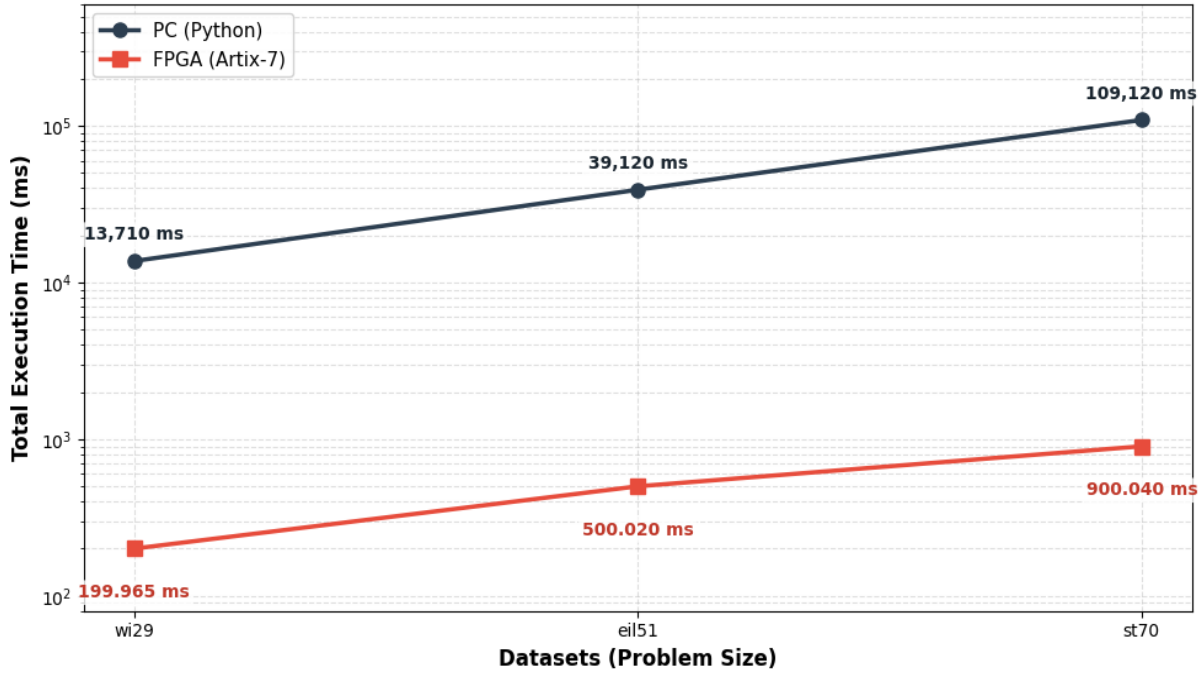

Figure 13. Algorithm execution time comparation.

Table 2 presents a comparative benchmark analysis to examine in more detail the numerical values of the timing data in the graph and the net speed-up rates provided by the hardware architecture.

Table 2. Comparation benchmark analysis

Dataset and Parameters	Measured Metric	Python (CPU)	FPGA (Artix-7)	Speedup Ratio
wi29 (Pop:150, Gen:1500)	Total execution time	13,710 ms	199.96 ms	**68.56**×
	Time-to-best	~730 ms	~25.00 ms	-
eil51 (Pop:150, Gen:2000)	Total execution time	39,120 ms	500.02 ms	**78.24**×
	Time-to-best	~16,290 ms	~225.00 ms	-
st70 (Pop:200, Gen:4000)	Total execution time	109,120 ms	900.04 ms	**121.24**×
	Time-to-best	~105,210 ms	~574.00 ms	-

The performance data presented in Table 2 provides numerical evidence of the effective superiority of the proposed FPGA-based architecture over traditional processor-based solutions. The acceleration ratio, which stands at 68.56x for the small-scale wi29 dataset, escalates to 121.24x as the problem complexity increases in the st70 dataset, highlighting the scalability of the hardware accelerator and its increasing efficiency with data size. To ensure convergence as the search space expands factorially, the algorithmic parameters were scaled according to the problem size: 1,500 generations with a population of 150 for WI29, 2,000 generations with a population of 150 for EIL51, and 4,000 generations with a population of 200 for ST70 were utilized. These parameters (population size, generation count, and genetic operator rates) were kept the same on both the FPGA and PC platforms to ensure a strictly fair benchmark for each dataset.

Furthermore, when the 'Time-to-best' metric is examined, the performance gap reaches an even

more striking dimension. For the most complex scenario, st70, the best-route search process that lasts approximately 105 seconds on the PC side is reduced to a mere 574 ms on the hardware. An important detail to note here is that Time-to-best durations may vary across different runs due to the stochastic nature of the genetic algorithm. Consequently, unlike the total execution time, a fixed speed-up ratio for Time-to-best has not been included in the table. The primary objective of presenting these values is to demonstrate how rapidly the proposed hardware architecture converges to solutions and to illustrate that high-quality results can be obtained within the order of milliseconds rather than seconds, rather than establishing a constant acceleration ratio.

It is important to note that the results presented in Table 2 represent the 'best solutions' obtained from multiple experimental runs. Since Genetic Algorithms are stochastic in nature, slight variations in solution quality may occur across different runs. However, throughout the experiments, the algorithm consistently converged to values very close to the reported best solutions and the known global optima from TSPLIB, demonstrating the stability and robustness of the proposed architecture.

At the heart of this tremendous difference is the Dual-Port BRAM macro architecture that overcomes the Von Neumann bottleneck by allowing parallel access to parent chromosome and offspring chromosome data within every clock cycle. The four-stage pipeline has a fully parallel structure which helps in latency hiding and maximizing throughput.

3.4. Comparison with Existing Studies and Contribution to the Literature

To validate the architecture in detail, a comparison with recent studies on FPGA-based Genetic Algorithms was made with the literature. This assessment specifically aims to compare the proposed hardware implementation with existing hardware implementation and demonstrate its advantage of achieving fast processing with low resource overhead. Table 3 provides a concise summary of the comparison of FPGA Models based on performance metrics of hardware platform, problem domain, speedup ratio w.r.t software/CPU, and resource utilization efficiency.

Table 3. Comparison with existing studies

Study	Year	FPGA Model	Problem Domain	Speedup Ratio	Resource Utilization
[7]	2022	Xilinx Spartan (XCS500E)	Random Forest	4×	98% LUT
[12]	2023	Intel Cyclone V (5CGXFC7D6F31C7ES)	Image Classification (GA)	N/A	65.7% ALM
[13]	2020	Xilinx Artix-7 (XA7A100T FGG484)-2i	Knapsack Problem (GA)	43×	41% LUT
[19]	2025	Xilinx Virtex-7	Sequence Alignment (GA)	N/A	8.69% LUT
[24]	2022	Xilinx Virtex-5	Light Focusing (GA)	150×	N/A
This Study	2026	Xilinx Artix-7 (XC7A15T)	Traveling Salesman Problem (GA)	121.24×	41.3% LUT

In Table 3, several key abbreviations are utilized to represent hardware metrics: LUT (Look-Up Table) and ALM (Adaptive Logic Module) refer to the primary logic resources of FPGAs, respectively, while N/A (Not Available) indicates that the specific metric was not explicitly reported in the original study.

As indicated in competing studies, a major finding is that none of the recent studies in the literature cover the hardware acceleration of TSP using Genetic Algorithms on FPGA specifically. The already existing implementations focus on Random Forest [7], Image Classification [12], Knapsack Problem [13], Sequence Alignment [19] and Light Focusing [24]. This gap evidence the contribution of this work, as it proposes a architecture specialised in a combinatorial optimization problem for which modern FPGA acceleration literature is silent and has a factorial complexity.

In addition, a hardware-centric analysis reveals that all existing works utilize high-end or mid-range FPGA platforms (e.g., Intel Cyclone V, Virtex-5, or Virtex-7) whose logic and memory capacities are significantly larger than that of the entry-level Artix-7 (XC7A15T) employed in this work. Implementation on a much more constrained and cheaper device yet the proposed architecture achieves a speedup ratio of 121.24×. The performance of this implementation improves performance over almost all other general GA implementations (for example, the performance of [7] was improved at 4× and the performance of [13] was improved at 43×). Also, it remains competitive in performance with the study's implementation [24] that achieved a 150× speedup on a much larger Virtex platform.

The proposed architecture is further distinguished by its efficiency, which demonstrates its strength. Despite Artix-7(XC7A15T) being the most resource constrained FPGA in Table 3, it has a very effective resource-performance ratio. While there are some implementations released on high-end platforms who report near saturation logic usage (as in Kumral [7] at 98% LUT usage) or omit exact usage figures [24], the proposed design is compact at only 41.3% LUT and 8% BRAM usage. The 1st entry-level hardware can get a speed up of 121.24 x through architectural enhancements like the 4-stage fully parallel pipeline and efficient memory management. Consequently, this research presents a hardware co-processor for combinatorial optimization that is resource-effective and scalable. It provides a suitable trade-off between speed, area, and power consumption for constrained edge computing applications.

4. CONCLUSION AND FUTURE WORKS

In this paper, a high-performance and scalable hardware architecture for solving the Traveling Salesman Problem was designed and physically implemented on the Xilinx Artix-7 FPGA platform. In-depth studies have proved that the design of the hardware-based accelerator happens to possess a conclusive advantage in timing and efficiency over standard processor-based software. Especially for the ST70 configuration (the largest dataset tested in this study), the acceleration ratio against a PC-based solution was approximately 121 times faster. This suggests that the architecture is robust and efficient as the workload scales up and is strongly adaptive to large-scale problems.

The Dual-Port BRAM architecture helps break the Von Neumann bottleneck, while the four-stage parallel pipeline structure helps to mask this delay resulting in high performance. These architectural features resulted in the time-to-best metric going from seconds to milliseconds, while still preserving the deterministic operational nature of the system. Moreover, using a low-level FPGA chip to execute minimal resource utilization says that the proposed design is not just performance efficient but also cost and energy-efficient which is the need of the hour. The small and powerful hardware footprint has the potential to enable integration of the accelerator into edge AI applications including autonomous systems and real-time logistics planning.

In future studies, it is planned to analyse the performance of the proposed hardware architecture on much larger-scale datasets. Furthermore, it is aimed to optimize the current design to fully align with the inherently parallel structure of FPGAs and to utilize it in more comprehensive algorithmic solutions where numerous parallel cores operate simultaneously. Explore the integration of more advanced, hardware-compatible Random Number Generator (RNG) designs to further enhance population diversity and algorithmic robustness. Consequently, it is envisioned that the developed system will be integrated into different optimization problems requiring high-density computation and expanded into industrial-scale applications.

REFERENCES

1. Pop, P.C., Cosma, O., Sabo, C. and Sitar, C.P., "A comprehensive survey on the generalized traveling salesman problem", European Journal of Operational Research, Vol. 314, Issue 3, Pages 819-835, 2024.

2. Alam, T., Qamar, S., Dixit, A. and Benaida, M., "Genetic algorithm: Reviews, implementations, and applications", arXiv preprint arXiv:2007.12673, 2020.

3. Katoch, S., Chauhan, S.S. and Kumar, V., "A review on genetic algorithm: past, present, and future", Multimedia Tools and Applications, Vol. 80, Issue 5, Pages 8091-8126, 2021.

4. Harada, T. and Alba, E., "Parallel genetic algorithms: a useful survey", ACM Computing Surveys (CSUR), Vol. 53, Issue 4, Pages 1-39, 2020.

5. Zhou, P., "Study on CPU and FPGA in Artificial Intelligence and Machine Learning", 2024 13th International Conference of Information and Communication Technology (ICTech), Pages 1-5, Wuhan, 2024.

6. Seng, K.P., Lee, P.J. and Ang, L.M., "Embedded Intelligence on FPGA: Survey, Applications and Challenges", Electronics, Vol. 10, Issue 8, Pages 895, 2021.

7. Kumral, C.D., Topal, A., Ersoy, M., Çolak, R. and Yiğit, T., "Performing Performance Analysis by Implementing Random Forest Algorithm on FPGA", El-Cezeri, Vol. 9, Issue 4, Pages 1315-1327, 2022.

8. Koyuncu, İ., Taşdemir, M.F., Alçın, M., Tuna, M. and Coşgun, E., "Real time realization of image processing algorithms on FPGA", Journal of Balikesir University Institute of Science and Technology, Vol. 24, Issue 1, Pages 125-137, 2022.

9. Doan, N.A.V., Manuel, M., Conrady, S., Kreddig, A. and Stechele, W., "Parameter Optimization of Approximate Image Processing Algorithms in FPGAs", 2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW), Pages 74-80, Naha, 2020.

10. Grubeša, S., Stamać, J., Suhanek, M. and Petošić, A., "Use of Genetic Algorithms for Design an FPGA-Integrated Acoustic Camera", Sensors, Vol. 22, Issue 8, Pages 2851, 2022.

11. Chakraborty, A., Dutta, S., Chakrabarti, I. and Banerjee, A., "VLSI architecture of stochastic genetic algorithm for real-time training of deep neural network", Sādhanā, Vol. 49, Issue 175, Pages 1-7, 2024.

12. Kaziha, O., Bonny, T. and Jarndal, A., "Genetic Algorithm Augmented Inception-Net based Image Classifier Accelerated on FPGA", Multimedia Tools and Applications, Vol. 82, Pages 45097–45125, 2023.

13. Alqudah, E. and Jarrah, A., "Parallel implementation of genetic algorithm on FPGA using Vivado high level synthesis", Int. J. Bio-Inspired Computation, Vol. 15, Issue 2, Pages 90-99, 2020.

14. Noordin, N.H., Eu, P.S. and Ibrahim, Z., "FPGA Implementation of Metaheuristic Optimization Algorithm", e-Prime - Advances in Electrical Engineering, Electronics and Energy, Vol. 6, Pages 100377, 2023.

15. Zermani, M.A., Manita, G., Chhabra, A., Feki, E. and Mami, A., "FPGA-based hardware implementation of chaotic opposition-based arithmetic optimization algorithm", Applied Soft Computing, Vol. 154, Pages 111352, 2024.

16. Jiang, Q., Xu, J. and Chen, Y., "A Genetic Algorithm for Scheduling in Heterogeneous Multicore System Integrated with FPGA", 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Pages 1-8, New York, 2021.

17. Yiu, Y.F. and Mahapatra, R., "Heuristic Function Evolution for Pathfinding Algorithm in FPGA Accelerator", 2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Pages 1-6, Laguna Hills, 2020.

18. Malhotra, G., Duraiswamy, P. and Kishore, J.K., "FPGA Accelerated Parallel HsClone GA for Digital Circuit Configuration in CGP Format", J. Inst. Eng. India Ser. B, Vol. 104, Issue 5, Pages 1079–1089, 2023.

19. Garcia, L.H., Alkamil, A., Mohsin, M.A., Menzel, J. and Perera, D.G., "FPGA-Based Hardware Architecture for Sequence Alignment by Genetic Algorithm", 2025 IEEE International Symposium on Circuits and Systems (ISCAS), Pages 1-5, London, 2025.

20. Wu, C., Zhang, K. and Zhang, X., "FPGA-Based Speed Control Strategy of PMSM Using Improved Beetle Antennae Search Algorithm", Energies, Vol. 17, Issue 8, Pages 1870, 2024.

21. Doroshenko, A., Shymkovych, V., Yatsenko, O. and Mamedov, T., "Automated Software Design for FPGAs on an Example of Developing a Genetic Algorithm", Proceedings of the 11th International Conference on DESSERT, Pages 1-12, Kyiv, 2021.

22. Barbareschi, M., Barone, S., Bosio, A., Han, J. and Traiola, M., "A Genetic-algorithm-based Approach to the Design of DCT Hardware Accelerators", ACM Journal on Emerging Technologies in Computing Systems, Vol. 18, Issue 3, Pages 1-22, 2022.

23. Danilova, E.Y. and Kovylyaev, D.A., "Advanced Genetic Algorithm for the Embedded FPGA Logic Diagnostic", 2021 International Conference on Information and Digital Technologies (IDT), Pages 93-97, Zilina, 2021.

24. Guo, S., Stern, R., Zhang, H. and Pang, L., "Speedy light focusing through scattering media by a cooperatively FPGA-parameterized genetic algorithm", Optics Express, Vol. 30, Issue 20, Pages 36414-36423, 2022.

25. Lotfy, A., Kaveh, M., Mosavi, M.R. and Rahmati, A.R., "An enhanced fuzzy controller based on improved genetic algorithm for speed control of DC motors", Analog Integrated Circuits and Signal Processing, Vol. 105, Issue 1, Pages 141–155, 2020.

26. Holland, J.H., "Genetic Algorithms", Scientific American, Vol. 267, Issue 1, Pages 66-73, 1992

27. Whitley, D., "A genetic algorithm tutorial", Statistics and Computing, Vol. 4, Issue 2, Pages 65-85, 1994.

28. Rose, J., El Gamal, A. and Sangiovanni-Vincentelli, A., "Architecture of field-programmable gate arrays", Proceedings of the IEEE, Vol. 81, Issue 7, Pages 1013-1029, 1993.

29. Monmasson, E. and Cirstea, M.N., "FPGA Design Methodology for Industrial Control Systems—A Review", IEEE Transactions on Industrial Electronics, Vol. 54, Issue 4, Pages 1824-1842, 2007.

30. Reinelt, G., "TSPLIB—A Traveling Salesman Problem Library", ORSA Journal on Computing, Vol. 3, Issue 4, Pages 376-384, 1991.