



GOLDS: Genetic Algorithm-based Optimization of Custom FPGA Architecture Layout Design for Secure Silicon

Pratyush Nandi
pratyush.nandi@iiitb.ac.in
IIIT-Bangalore
India

Anubhav Mishra
anubhav.mishra@iiitb.ac.in
IIIT-Bangalore
India

Madhav Rao
mr@iiitb.ac.in
IIIT-Bangalore
India

ABSTRACT

Modern FPGAs, equipped with heterogeneous blocks (H-blocks) offer design flexibility, enabling the implementation of complex designs with minimal effort. Custom-embedded fabric in System-on-a-Chip (SoC) has recently been envisioned to secure an IP block through the entire flow from design to manufacture, which involves multiple vendors. Hence, new design strategies for deducing efficient embedded-FPGA (eFPGA) architecture blocks targeted toward specific workloads need to be investigated. This work applied Genetic Algorithm (GA) to generate the most optimal H-block-defined FPGA layout to achieve the best hardware objectives, including critical path delay, resources utilized, and power consumed for the benchmark under test. The strategic placement of memory, digital signal processor (DSP) slices, and configurable logic blocks (CLBs) targeted for attaining minimum area-delay-product (ADP) for the benchmark-under-test, on the given layout forms the crux of this work. This novel framework was applied to different machine learning (ML) and non-ML benchmarks to characterize its performance against other state-of-the-art (SOTA) FPGA architectural layout designs. The proposed work showcases a maximum of 35% ADP gain for one of the benchmark design runs over the SOTA-designed layout. The ideal fabric layout architecture is a step towards securing IP on an eFPGA block, which is not disclosed to other vendors and configured as per the targeted workload post-fabrication of the chip.

CCS CONCEPTS

• **Hardware** → *Full-custom circuits*; **Reconfigurable logic and FPGAs**; **Programmable logic elements**.

KEYWORDS

FPGA, Optimization, Layout, Genetic Algorithm, Secure IP, Fabric design, Reconfigurable ASIC

ACM Reference Format:

Pratyush Nandi, Anubhav Mishra, and Madhav Rao. 2024. GOLDS: Genetic Algorithm-based Optimization of Custom FPGA Architecture Layout Design for Secure Silicon. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24)*,

June 12–14, 2024, Clearwater, FL, USA. ACM, New York, NY, USA, 6 pages.
<https://doi.org/10.1145/3649476.3658743>

1 INTRODUCTION

Field-programmable Gate Arrays (FPGAs) have an enormous advantage as a computational engine, making them one of the preferred choices for a wide range of applications. FPGAs not only serve as a power-efficient and high-performing computing setup, but primarily allow reconfigurability, and re-programmability of designs for achieving best possible results [7]. However, in the modern-day computing demands with the ever-changing technological landscape, employing a rigid layout for diverse workloads leads to ineffective execution. Especially with the rising need for embedded FPGA (eFPGA), the desire to have both reconfigurability and security on silicon has grown [6]. The reconfigurability of the fabric on silicon allows designers to flexibly change security protocols even after post production thereby, making it highly secure. Moreover, OpenFPGA has demonstrated the flow from FPGA design to silicon footprint through the regular ASIC flow [24]. Hence, design strategies to develop custom eFPGA that fits through the existing CMOS flow is critical and valuable to the overall VLSI community. This paper discusses one such optimization framework to design a highly custom fabric targeted for specific workloads. This framework is expected to generate a layout which could be further fed to OpenFPGA flow for generating secure fabric designs on silicon.

Typically, most FPGA architectures consist of heterogeneous blocks (H-blocks) such as Block RAMs and DSPs, configurable logic blocks (CLBs), inherent routing architectures, and switch blocks. Synthesis and implementation algorithms aim to map the logical and computational designs on the above-mentioned blocks of FPGA [19]. According to the authors' knowledge, little work has been done in layout design and optimal positioning of FPGA components, specifically on the H-blocks for domain-specific applications. For instance, in general purpose AMD-Xilinx or Intel FPGAs [12], DSPs, and Block RAMs are evenly spread across the fabric, in order to satisfy performance requirements for diverse applications. In Achronix FPGAs [9], DSPs and Block RAMs are located next to each other to implement neural networks efficiently. Island-style FPGAs are most common among conventional FPGA architectures due to their simple and compact layout structure. In such FPGAs, the heterogeneous blocks occupy rows or columns and are interleaved with CLBs in between. In this style of architecture, only one type of H-block means either DSPs or BRAMs occupy the entire column or row. This approach gives a compact layout but suffers from a few critical issues. For instance, given a heavy computing workload design on the layout equipped with DSPs enclosing BRAMs and CLBs, large wire lengths are drawn to realize the design, which

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0605-9/24/06

<https://doi.org/10.1145/3649476.3658743>

leads to increased critical path delay, affecting the overall performance. Another instance is in the implementation of a finite state machine (FSM), though the resource utilized is minimal, the entire column comprising of BRAM is utilized, thus adding area overhead with little or no improvement in performance.

These reconfigurable FPGA hardware platforms have become integral component for diverse applications, ranging from high-performance computing servers to edge-computing embedded systems. In this complex and diverse landscape of FPGA design, Verilog-to-Routing (VTR), an open-source tool has allowed us to explore the architectural layout design [13, 18]. VTR tool maps high-level hardware description design defined in Verilog to intricately routed digital circuits that harness available FPGA resources. Beyond resource optimization, VTR empowers designers to explore a plethora of architectural possibilities, allowing them to strike the ideal balance between resources utilized, critical path delay, and power consumption tailored to the specific demands of the design runs. Exploring resources equipped architectural FPGA layout for specific designs targeted towards workload run is a daunting task, considering the placement of H-blocks and CLBs within the footprint defined offer a huge combination to evaluate the objectives. One can always evaluate multiple combinations of H-blocks and CLBs towards achieving the least resources and minimal critical path delay, forming an exhaustive design-space problem. However, a better and more scalable approach is to adopt an optimization algorithm for achieving the ideal H-blocks defined FPGA layout to realize the design mapped for a desired workload run. Genetic Algorithm (GA) [16] is one of the earliest and most widely used Evolutionary Algorithms (EAs) [23]. Evolutionary algorithms are a heuristic-based approach utilized for solving problems that are not easily expressed in polynomial time. GA is inspired by the process of natural selection and genetics, as it searches for the best optimal solution from a population of candidate solutions.

This paper presents a novel framework with a unique and alternative methodology to extract optimal H-blocks defined FPGA layout. The framework is powered by the GA optimizer engine, and the evolved FPGA layout is constantly assessed by the VTR tool. The area-delay-product (ADP) is considered a fitness parameter in the proposed framework to extract the best H-blocks defined layout for the application run with the underlying design mapped to the FPGA resources. Several benchmarks are applied to verify the functionality of the proposed framework. The chosen benchmarks when mapped on FPGA encompass a wide combination of resource usage, ensuring comprehensive evaluation. The proposed GA-adopted framework designed for finding the best FPGA layout is made freely available at [20] for further usage by the designers and researchers community. This is a step towards realizing optimized fabric design to run dedicated workloads for fabric-on-silicon designs efficiently.

2 EMBEDDED FPGA

FPGAs are typically considered too expensive and relegated to prototype products or time-to-market advantage for emerging standards. In modern-day computing, the applications of FPGA are now not only limited to the previously mentioned use case, but it is now increasingly being integrated with the SoC (system-on-chip).

A re-configurable fabric inside the SoC is considered a viable and valuable option because, with eFPGA, one can define the number of look-up tables (LUTs), registers, embedded memory, and DSP blocks. One can also control the aspect ratio and number of I/O ports, making trade-offs between power and performance[6] as shown in Figure 1. With our proposed framework we target to design a domain-specific FPGA architecture, integrated into a SoC as a reconfigurable fabric thus catering to the needs of the application and providing upgradability as and when required. Secure IP is protected without giving away any details to the manufacturers. In summary, the major advantages of eFPGA are:

- (1) System Performance: The programmed FPGA fabric with a domain-specific architecture can be integrated with a SoC for better performance.
- (2) Security: The reconfigurable fabric provides additional security as it remains flexible regarding programmability, unlike ASIC. The eFPGA fabric is programmed with additional security protocols to adapt to future threats. Besides, the most critical IP is not programmed till the chips are at hand.
- (3) Extended Life: The versatility of the eFPGA enables SoC to remain relevant in the market for a longer period, as it allows for accommodating newly identified use cases even after the SoC has been manufactured.

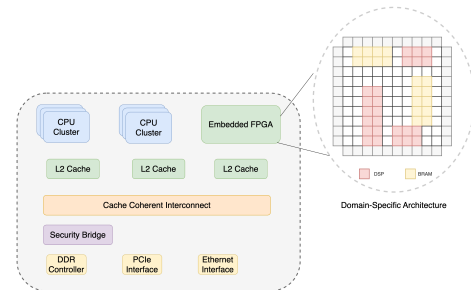


Figure 1: Schematic showing SoC with programmable accelerator block on eFPGA.

3 BACKGROUND AND RELATED WORK

3.1 Genetic Algorithm

The most widely used genetic algorithm follows traditional evolutionary methodology [4, 8, 15, 21]. A set of possible solutions (*population*) are recombined (*crossover*) and/or perturbed (*mutation* based on a probability). To execute a certain task, the new solutions are evaluated based on their performance (*fitness*). The most optimal solution is then selected and passed on to the next iteration (*generation*). In the next iteration, the new optimal solution again gets recombined and/or perturbed with a new set of candidate solutions to generate a more optimal solution than the previous one. The iteration continues until a termination condition is reached, as shown in the Figure 2. The primary assumption is that the emerged problem has some local continuity, and the solutions generated by mutation and crossover are superior to randomly generated solutions. This emulates the principles of natural Darwinian evolution, where fitness forces the selection of individuals from the population, and crossover and mutation ensure diversity and novelty in the next generation.

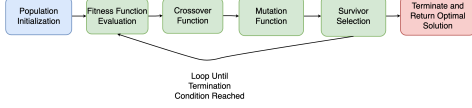


Figure 2: Overview of Genetic Algorithm.

3.2 Related Works

H-blocks play a crucial role in the contemporary design of FPGAs. These specialized blocks are tailored to perform specific tasks efficiently, such as digital signal processing (DSP), and memory access (Block RAMs). By integrating diverse processing elements within a single FPGA, designers can create custom hardware accelerators for various applications, covering machine learning (ML) runs to high-speed network switching. This flexibility enhances performance and significantly reduces power consumption compared to traditional homogeneous FPGAs. H-blocks empower FPGA designers to optimize their hardware architectures and achieve superior levels of adaptability and performance. The strategic placement of H-blocks within an FPGA is critical towards optimizing performance and minimizing interconnect delays. Careful placement saves data transfer distances, and enhances overall speed and efficiency. Additionally, it allows designers to capitalize on resource proximity, ensuring that computational tasks are allocated to the most suitable blocks for maximum throughput. In the H-blocks defined FPGA design, appropriate placement of blocks turns out to be a key component in realizing the full potential. On fixing the resources defined FPGA layout, further optimization and mapping of the design to realize on fabric, along with placement and routing schemes to achieve minimum critical path delay, forms the complete design process. The latter two are explored heavily [1, 10, 14, 22, 25], but the former - Layout engineering has not been investigated much and has the potential to provide much better design solutions on FPGA when compared with the current mechanism. The H-blocks defined resource-aware layout is likely to simplify the placement and routing task, considering resource positions are predefined appropriately to suit the workload design. Optimally placed resources are expected to reduce the FPGA compilation time for large designs, besides providing the Quality-of-Results (QoR) for the entire realization flow. The placement determines the routing delays, and hence even post-synthesis parameters are also expected to be benefited.

One such example is studied in [11]. The work in the literature focuses on exploring different types of FPGA architecture with alternate physical locations of H-blocks. It makes use of the VTR tool to comprehend the number of H-blocks in the critical path of VTR benchmark designs and analyzes the impact of grouping H-blocks together. An improvement of 3.45% along the critical path delay was noted for the designs mapped from the benchmark run. Furthermore, an average improvement of 5.66% in routing delay was reported as a direct result of the H-block grouping. Additionally, a new custom architecture was proposed for deep learning (DL) benchmark suite Koios and its impact was evaluated in [17]. This new architecture was referred to as scattered cluster architecture, which consists of a cluster of H-blocks scattered across the FPGA layout, showcasing an average improvement of 6.4% in critical path delay over the best-performing architectural topology. It is important to note that this enhancement was solely attributed to the repositioning of H-blocks within the fabric, representing architectural adjustments, without any alterations to the underlying

optimization algorithms for logic realization placement and routing schemes. However, this work relies on optimizing the positioning of H-blocks in the layout to minimize the critical path delay and, thereby, not necessarily achieve the compact form of resources laid in FPGA. Hence, a better option is to appreciate both objectives and possibly evolve the solution for the product of area and delay (ADP).

4 PROPOSED FRAMEWORK

The proposed (GOLDS) framework comprises an automated flow that is supplied with an initial set of solutions representing H-blocks defined FPGA layout as shown in the Figure 3. The framework evolves the most optimal solution for the fitness function defined. The optimal solution consists of positions of CLBs, DSPs, and BRAMs on the FPGA layout. The XML file is auto-generated with the positions specified by the optimal solution. The VTR architectural exploration tool is supplied with the XML file to generate the FPGA architectural layout and runs a specific benchmark file to characterize performance parameters. The fitness function is continuously evaluated for new solution sets, as directed by the evolving GA runs until the predefined termination condition is met. The termination criterion is satisfied when a predetermined generational limit is reached or a convergent fitness result is achieved.

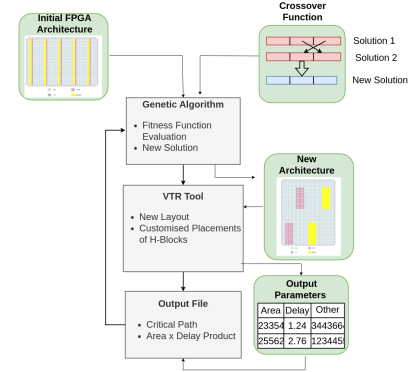


Figure 3: Flowchart of the proposed GOLDS Framework involving Genetic Algorithm for evolving heterogeneous resourced FPGA architectural layout.

The evolutionary algorithms and their techniques [5] are modified accordingly to establish the runs for extracting architectural layout comprising of heterogeneous resources. The algorithm consists of the following steps:

- (1) Generate a random initial solution i.e. parent solution P and evaluate its fitness f , which in this case is set to ADP for the architectural layout.
- (2) Inside the main algorithm:
 - a) Generate α children $C_1, C_2, \dots, C_\alpha$ by crossover and mutation functions.
 - b) Evaluate children's fitness values $f_1, f_2, \dots, f_\alpha$.
 - c) If any child C_i has fitness value $f_i < f$ then the parent solution is replaced by the child. If multiple children satisfy the condition, the child with the best fitness value is chosen and a new generation g is created.

- (3) Termination condition is reached either when G generations of populations are created, or the fitness function of a particular population remains unchanged across β generations.

4.1 Solution Representation

In the proposed framework, the H-blocks defined FPGA layout as a solution is represented as an array of arrays. One such solution is defined as $[10, [[5,2,12],[6,6,12]], [[10,8,21],[11,4,21]]]$ is depicted in the Figure 4 (b). Its corresponding XML file snapshot specifying the locations in terms of X-Y grids for DSPs and BRAMs, along with the priority order for all the resources, including CLB is shown in the Figure 4 (a). Here, the first element of the array is the priority with which CLBs are filled. The next set of elements is again an array of arrays representing the positions of DSPs and BRAMs, respectively. The first one refers to the positions of two DSPs, while the subsequent one reflects the positions of two BRAMs. Individual DSP positions and BRAMs positions are defined in the specific array entity. Figure 4 (b) depicts the positions of DSPs and BRAMs as per the two-dimensional (2D) coordinates that are used to define the H-blocks in the FPGA layout. The 2D coordinates are followed by priority order in both DSPs and BRAMs positions such as 12, and 21. CLBs are defined with only priority order. The priority order ensures that the particular H-block is given preference for the desired space over other blocks. The higher priority order is preferred over others. Each population is mutated and applied for crossover, among the θ solutions to generate a new set of solutions.

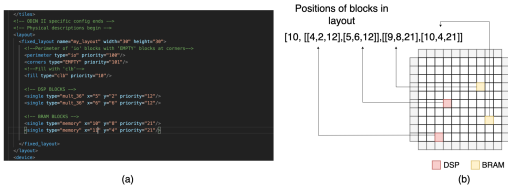


Figure 4: (a) Picture of XML file describing the positions of DSPs and BRAMs, and (b) FPGA layout representation for the above XML file.

4.2 Genetic Operators

The proposed *GOLDS* framework powered by a genetic algorithm runs primarily on three operators: initialization, crossover, and mutation.

Initialization: The initialization is the first set of solutions, which is parameterized by the number of DSPs and BRAMs. Initially, positions of all DSPs and BRAMs are randomly generated with no overlap. After this step, a random priority is assigned to the filling of CLBs, DSPs, and BRAMs.

Crossover: Crossover is performed among the solutions. The arrays containing the same number of elements are exchanged. Essentially the positions of DSPs and BRAMs are swapped with the positions of DSPs and BRAMs belonging to different solutions. For example, assume a simple solution [1, 2, 3] and another solution [4, 5, 6]. Now, if the crossover function is applied based on a probability p , then swapping is executed between the second element of the first solution and the third element of the second solution, giving rise to the new solution [1, 6, 5].

Mutation: Mutation on solutions is performed based on a pre-decided probability value p . For instance, given a solution of $[1,$

2, 3], post mutation, the new solution of [1, 4, 3] evolved is quite possible. While performing mutation, it is ensured that the randomly allocated positions for resources do not overlap and remain within the bounds of the FPGA fabric. The algorithm ensures that the mutation operation is carried out separately on both resources in order to prevent any overlapping between them. It also ensures that the same set of solutions is not generated while performing mutation.

4.3 Fitness Function

The proposed framework’s fitness function is configured to the product of area and delay (ADP). Area implies the coverage of the logic blocks in the layout formed. This includes the number of LUTs and DSP slices utilized among the arranged ones in the layout. Delay refers to the most critical path latency for the resources configured. The GA aims to minimize the fitness value (ADP) for a given design that is mapped for benchmark workload. For the proposed framework, the terminating condition is succeeded when no further improvement in fitness score over β successive generations is attained or when the generations surpass the pre-defined number G . The best solution is considered optimal on exiting given the objectives forming the fitness function.

4.4 Automation and Parsing

Each solution representing the resource-defined FPGA layout is supplied as input to the VTR flow. The VTR runs the specified benchmark on the architectural layout defined by the XML file. The output generated by the VTR tool involves implementation metrics and a successful benchmark run on the architectural layout. The total logic area of the block and critical path delay is extracted from the output file to evaluate the fitness score and are supplied to the GA for generating the subsequent set of solutions in the following iterations.

5 RESULTS AND DISCUSSIONS

5.1 Selection of Benchmarks and Baseline Architecture

The benchmarks employed in this paper for FPGA resources-based architecture layout evaluation are listed in Tables 1, and 2. These include the set of standard medium-sized benchmarks, which are supported by the VTR tool [13]. All these 8 benchmarks have different sets of resource requirements, including IOs, BRAMs, DSPs, CLBs, and hence, the proposed framework is evaluated for a wide range of resource positioning. Additional experimental validation and evaluation with Koios benchmarks were performed. The Koios [2, 3] is a suite of 19 Deep Learning (DL) acceleration benchmarks. They present a diverse range of computational designs to execute DL workloads. These benchmarks place a heavy demand on DSP, BRAM, Flipflops, and Adders resources in comparison to VTR-compliant benchmarks. Additionally, the designs demand increased data parallelism, deeper pipelining, and heavier utilization of FPGA routing resources, including wider buses. These benchmarks differ in design complexity, implementation approaches, and numerical

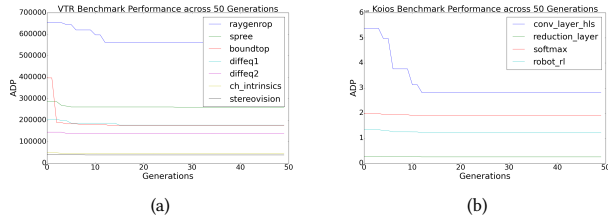
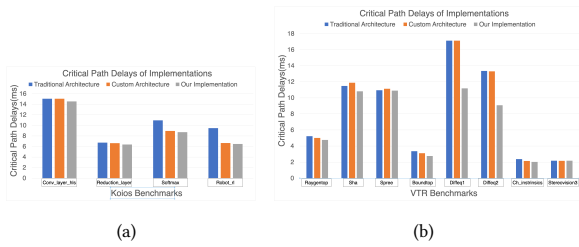
Table 1: VTR Tool Benchmark runs and the corresponding Resource Requirements.

Benchmarks	IOs	CLBs	DSPs	BRAMs
Raygentop	214	110	8	0
Sha	38	154	0	0
Spree	45	65	1	3
Boundtop	142	94	0	0
Diffeq1	162	32	5	0
Diffeq2	66	22	5	0
Ch_intrinsics	99	64	0	1
Stereovision3	11	14	0	0

precision. The Koios benchmarks and their resource utilization demands are presented in Table 2.

Table 2: Koios suite workload showing a layer of Convolution, Softmax, and Robot and its corresponding resource required to complete the task.

Benchmarks	Description	DSPs	BRAMs
conv_layer_hls	Sliding Window Convolution	12	21
Reduction_layer	Add Max/Min Reduction Tree	0	52
Softmax	Softmax Classification Layer	53	0
robot_rl	Robot + Maze Application	18	96

**Figure 5: (a) ADP of evolved solutions for VTR Benchmarks across 50 Generations and (b) ADP of evolved solutions for Koios Benchmarks across 50 Generations.****Figure 6: (a) Critical path delays of Koios benchmarks against different implementations, and (b) Critical path delays of VTR benchmarks against different implementations**

5.2 Performance Evaluation

The primary goal of this work is to generate an efficient resource-bound FPGA layout. The framework is first evaluated on VTR benchmarks, which are listed in the Table 1. Figure 5 shows the GA runs with the fitness parameter of ADP, which continues to decrease for the initial set of generations and converges much earlier to the set of 50 generations for all the benchmark workloads. GA maintains the local continuity in each iteration while furnishing the subsequent architectural design solution. The first set of randomly generated architectures has a certain ADP, however, as the algorithm continues, the mutation and crossover function allows exploration of different positions for the H-blocks on the architecture. At every iteration, a group of solutions is extracted, with, only the best-fit solution is carried forward. A significant decrease in ADP is prominently seen for Raygenrop, BoundTop, and conv_layer_hls benchmark runs.

Figure 6 clearly reports that the proposed GA-engineered framework furnished H-blocks defined FPGA architectural layout offers the least critical path delay over the other SOTA-based FPGA layouts, including traditional island-style and recently investigated customized (cluster and fractured) FPGA approaches described earlier. The proposed framework extracted 12 different FPGA architectural layouts for the corresponding workload, and all the layouts consistently offered better delay parameters for the design runs when compared with the same benchmark runs on SOTA-based FPGA layouts. The maximum delay improvement of 34.9% and 31.60% was observed for VTR tool benchmarks and Koios Suite of benchmarks respectively for the GA derived layout over the SOTA-based island style (Traditional) FPGA architectural layout. A maximum delay improvement of 34.65% is established for the proposed GA enabled FPGA layout engineering over the recently established custom approach [17]. The fractured layout and clustered format were reported in the custom approaches while positioning H-blocks along the critical path flow as discussed in [17] for VTR benchmarks and Koios suite of benchmarks respectively. This clearly suggests the benefits of the proposed *GOLDS* framework for designing the H-blocks defined FPGA layout. Figure 7 showcases that the architecture is evolving towards a more compact structure across the generations, eventually attaining a reduced Manhattan distance for a *Spree* benchmark run. The design confirms that the 2 H-blocks that lie in the critical path of the circuit are closer to the CLBs and each other. A similar trend was observed for Koios deep learning benchmarks as well. Overall a maximum ADP gain of 35% was reported for one of the benchmark design runs (diffeq1) over the traditional FPGA layout. The area gain in the form of resources utilized stays in the range of 1% to 2% for the proposed method over the SOTA implementations. The overall performance for all the 12 selected benchmark runs on the newly derived FPGA architectures over the other two SOTA implementations is listed in the Table 3. The newly formed FPGA layout consistently improves overall benchmark performance, showcasing the effectiveness of the GA-powered H-Blocks arrangement for enhanced design realization. This work underscores the necessity for a more resource-aware architectural layout in design realization for dedicated workloads. The optimized custom fabric layout to run a specific set of workloads forms a primitive step in the overall design

flow of reconfigurable fabric on silicon. The desired configuration of the logic on the custom fabric-on-silicon is expected to run the target workload more efficiently.

Table 3: Performance gain of the benchmark (VTR, and Koios) runs on the layout extracted from the Proposed GOLDS framework over two SOTA implementations.

Category	Benchmarks	Traditional	Custom [17]
Koios	conv_layer_hls	3.34%	3.32%
Koios	Reduction_layer	5.2%	3.77%
Koios	Softmax	20.4%	2.57%
Koios	robot_rl	31.6%	2.7%
VTR	Raygentop	8.81%	4.8%
VTR	Sha	5.75%	9%
VTR	Spree	0.5%	1.55%
VTR	Boundtop	17.85%	10.96%
VTR	Diffeq1	34.9%	34.65%
VTR	Diffeq2	32%	31.8%
VTR	Ch_intrinsics	14.76%	5.16%
VTR	Stereovision3	0%	-0.46%

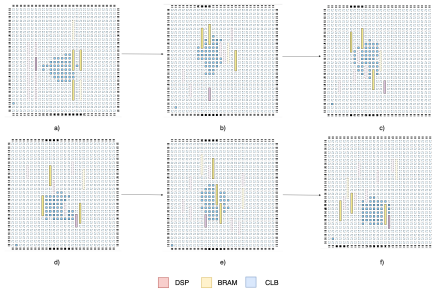


Figure 7: Architectural layout of *spree* benchmark across 50 generations.

6 CONCLUSIONS

This paper proposes an automated framework for generating an optimal FPGA resource-equipped architectural layout. GA enabled with mutation and crossover capability was adopted to optimize the layout for achieving the lowest ADP. The proposed framework generates a layout solution outperforms traditional island-style FPGAs and SOTA custom architectures, including scatter and fractured designs. This methodology of extracting the best resource-positioned FPGA layout is a step toward developing a custom fabric support for realizing secure IP on silicon. The automated framework is expected to revolutionize the custom FPGA hardware accelerator designs for modern-day computing requirements. The framework and all the supporting files are made freely available at [20] for easy adoption and further usage.

REFERENCES

- [1] Ziad Abuowaimar, Dani Maarouf, Timothy Martin, Jeremy Foxcroft, Gary Gréwal, Shawki Areibi, and Anthony Vannelli. 2018. GPlace3.0: Routability-Driven Analytic Placer for UltraScale FPGA Architectures. 23, 5, Article 66 (oct 2018), 33 pages.
- [2] Aman Arora, Andrew Boutros, Seyed Alireza Damghani, Karan Mathur, Vedant Mohanty, Tanmay Anand, Mohamed A. Elgammal, Kenneth B. Kent, Vaughn Betz, and Lizy K. John. 2023. Koios 2.0: Open-Source Deep Learning Benchmarks for FPGA Architecture and CAD Research. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 11 (2023), 3895–3909.
- [3] Aman Arora, Andrew Boutros, Daniel Rauch, Aishwarya Rajen, Aatman Borda, Seyed Alireza Damghani, Samidh Mehta, Sangram Kate, Pragnesh Patel, Kenneth B Kent, et al. 2021. Koios: A deep learning benchmark suite for fpga architecture and cad research. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 355–362.
- [4] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Evolving graphs by graph programming. In *European Conference on Genetic Programming*. Springer, 35–51.
- [5] Thomas Bäck and Hans-Paul Schwefel. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation* 1, 1 (1993), 1–23.
- [6] Brian Bailey. 2019. The Case For Embedded FPGAs Strengthens And Widens. <https://semiengineering.com/embedded-fpga-becomes-a-viable-option/>. [Online; accessed 25-April-2019].
- [7] Andrew Boutros and Vaughn Betz. 2021. FPGA Architecture: Principles and Progression. *IEEE Circuits and Systems Magazine* 21, 2 (2021), 4–29.
- [8] Markus Brameier, Wolfgang Banzhaf, and Wolfgang Banzhaf. 2007. *Linear genetic programming*. Vol. 1. Springer.
- [9] Product Brief. 2008. Speedster® fpga family. *Achronix Semicond., San Jose, CA, USA, Tech. Rep. PB001* (2008).
- [10] Marcel Gort and Jason H. Anderson. 2012. Analytical placement for heterogeneous FPGAs. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. 143–150. <https://doi.org/10.1109/FPL.2012.6339278>
- [11] David Lewis, Elias Ahmed, David Cashman, Tim Vanderhoeck, Chris Lane, Andy Lee, and Philip Pan. 2009. Architectural enhancements in Stratix-III™ and Stratix-IV™. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 33–42.
- [12] David Lewis, Gordon Chiu, Jeffrey Chromczak, David Galloway, Ben Gamsa, Valavan Manohararajah, Ian Milton, Tim Vanderhoeck, and John Van Dyken. 2016. The Stratix™ 10 highly pipelined FPGA architecture. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 159–168.
- [13] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, et al. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7, 2 (2014), 1–30.
- [14] Timothy Martin, Dani Maarouf, Ziad Abuowaimar, Abeer Alhyari, Gary Grewal, and Shawki Areibi. 2019. A Flat Timing-Driven Placement Flow for Modern FPGAs. In *Proceedings of the 56th Annual Design Automation Conference 2019 (Las Vegas, NV, USA) (DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages.
- [15] Julian Francis Miller and Simon L Harding. 2008. Cartesian genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. 2701–2726.
- [16] Seyedali Mirjalili and Seyedali Mirjalili. 2019. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications* (2019), 43–55.
- [17] Anubhav Mishra, Nanditha Rao, Ganesh Gore, and Xifan Tang. 2023. Architectural Exploration of Heterogeneous FPGAs for Performance Enhancement of ML Benchmarks. In *2023 IEEE Asia Pacific Circuits and Systems (APCCAS)*.
- [18] Kevin E Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron G Graham, Jean Wu, Matthew JP Walker, et al. 2020. Vtr 8: High-performance cad and customizable fpga architecture modelling. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13, 2 (2020), 1–55.
- [19] Kevin E. Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. 2013. Titan: Enabling large and complex benchmarks in academic CAD. In *2013 23rd International Conference on Field programmable Logic and Applications*. 1–8.
- [20] Pratyush Nandi. 2024. <https://github.com/pratyush48/GOLD> Accessed on January 10, 2024.
- [21] Riccardo Poli et al. 1997. Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming. In *ICGA*. Citeseer, 346–353.
- [22] Chak-Wa Pui, Gengjie Chen, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Peishan Tu, Hang Zhang, Evangeline F. Y. Young, and Bei Yu. 2016. RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.
- [23] Adam Slowik and Halina Kwasnicka. 2020. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications* 32 (2020), 12363–12379.
- [24] Xifan Tang, Edouard Giacomin, Aurélien Alacchi, Baudouin Chauviere, and Pierre-Emmanuel Gaillardon. 2019. OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 367–374.
- [25] Dries Vercruyce, Elias Vansteenkiste, and Dirk Stroobandt. 2017. Liquid: High quality scalable placement for large heterogeneous FPGAs. In *2017 International Conference on Field Programmable Technology (ICFPT)*. 17–24. <https://doi.org/10.1109/FPT.2017.8280116>