Hindawi
IET Computers & Digital Techniques
Volume 2024, Article ID 6623637, 23 pages
https://doi.org/10.1049/2024/6623637

Research Article

Accelerated and Highly Correlated ASIC Synthesis of AI Hardware Subsystems Using CGP

H. C. Prashanth and Madhav Rao

International Institute of Information Technology Bangalore, Bengaluru, India

Correspondence should be addressed to Madhav Rao; mr@iiitb.ac.in

Received 21 July 2023; Revised 26 November 2023; Accepted 6 January 2024; Published 29 January 2024

Academic Editor: Seok-Bum Ko

Unconventional functions, including activation functions and power functions, are extremely hard-to-realize primarily due to the difficulty in arriving at the hierarchical design. The hierarchical design allows the synthesis tool to map the functionality with that of standard cells employed through the regular ASIC synthesis flow. For conventional functions, the hierarchical design is structured and then supplied to the synthesis flow, whereas, for unconventional functions, the same method is not reliable, since the current synthesis method does not offer any design-space exploration scheme to arrive at an easy-to-realize design entity. The unconventional functions either take a long synthesis run-time or additional efforts are spent in restructuring the hierarchical design for the desired function to synthesizable ones. Cartesian genetic programing (CGP) allows to not only incorporate custom logic gates for synthesizing the hierarchical design but also aids in the design-space exploration for the targeted function through the custom gates. The CGP configuration evolves difficult-to-realize complex functions with multiple solutions, and filtering through desired Pareto-optimal requirements offers a unique hierarchical design. Incorporating CGP-derived hierarchical designs into the traditional synthesis flow is instrumental for implementing and evaluating higher-order designs comprising nonlinear functional constructs. Six activation functions and power functions that fall in the category of unconventional functions are realized by the CGP method using custom cells to demonstrate the capability. Further, the hierarchical design of these unconventional functions is flattened and compared with the same function that is directly synthesized using basic gates. The CGP-derived synthesis method reports 3× less synthesis time for realizing the complex functions at the hierarchical level compared to the synthesis using basic gate cells. Hardware characteristics and error metrics are also investigated for the CGP realized complex functions and are made freely available for further usage to the research and designers' community.

1. Introduction

The standard rule-based and top-down synthesis flow [1–6] is adopted regularly for realizing various arithmetic and logical functions, including higher operand bit-widths. However, these methods neither allow any room to augment custom cells toward arriving at the hierarchical design nor offer any design-space exploration to realize the hardware-reliable design. The focus of the synthesis flow has always been to map the hand-crafted structured design optimally with the functionally equivalent logical gates toward attaining minimum delay with less gate count. The heuristic design is further supplied to the standard ASIC flow to realize the hardware parameters. The whole synthesis process and the gates list are industry-protected and, hence, are not customizable for any refinement while realizing the design [3, 7, 8]. The advances in standard cell design for the ASIC flow to realize custom hardware gain are acceptable; however, this demands extra design effort and time to characterize new gates and their feasibility for realizing other designs, including traditional computational design entities, apart from the nonlinear functions which are of interest in this work [9, 10]. Hence, a need for a hierarchical design flow to represent unconventional functions that allow (i) design-space exploration to achieve optimal hardware parameters, (ii) augment custom cells for high-level synthesis, (iii) remains independent of the standard cell library that is adopted for ASIC flow, and (iv) easily integrate with standard ASIC flow to realize system designed on-the-chip. Combining the much-needed hierarchical design flow and ASIC flow into one synthesis flow is an option, but one has to be careful, considering any minor functionality upsets need fine-tuning and reiterating the overall process. The time-consuming process is still pleasantly absorbed by today's designers toward realizing linear or arithmetic functions. However, this strategy does not hold well for realizing unconventional designs such as nonlinear activation and power functions [9, 10]. Hence, keeping the hierarchical design flow separate from the regular ASIC flow is suitable, yet the process of hierarchical framework and its smooth integration to ASIC flow will be beneficial for quick characterization of the design. This paper establishes a framework to evolve the best hierarchical design constructs for nonlinear functions, such as activation and power functions, which are then seamlessly fed to the ASIC flow to extract the hardware parameters. The two-level synthesis method proposed in this work aims to simplify the unconventional function to a level that is further easy-to-synthesize through the regular ASIC synthesis flow. Other forms of realization do exist in the literature but the complete silicon realization requires more effort in structuring the design for easy-to-synthesize. Application-specific activation functions are not only difficult to design but also realize the same using traditional methods of Look-Up-Tables, and IP cores incur a heavy penalty in the hardware resources used [9, 11–13]. Hierarchical gate-level design synthesized for unconventional functions generated from a custom-defined set of gates is extremely beneficial to the designers. The proposed hierarchical synthesis flow with a quick turnaround time is likely to aid designers to not only validate the functional requirement but also offers an opportunity to alter behavioral design whenever desired. The overall synthesis process for arriving at hierarchical design is intended to be a black-box implementation, where the designers remain isolated from the details of the characterized gates till the last level of synthesis. Invariably, the hierarchical netlist generated by the proposed fast synthesis method is desired to break down the realization process to an intermediate level and then proceed further toward ASIC-level synthesis. Electronic design automation has evolved over the years, but few methods offer optimal gate-level design solutions by performing a thorough design-space exploration under tight constraints [7, 14–16]. This includes metrics-driven design methodologies, especially in the digital design domain [17, 18]. The primary objective of the logic synthesis process is to develop optimized versions of gate-level designs with a minimum number of cells and logic depth, which directly contributes to the circuit's performance on silicon [19, 20].

The existing ABC synthesis tool [21] employs directed acyclic graphs that use two-input AND nodes and their associated edges, forming an AND-Inverter Graph (AIG) to represent circuits. The optimization process entails reducing the AIG size by replacing the subgraphs with precomputed ones iteratively, while preserving the node functionality [22]. Despite being straightforward and scalable, this synthesis method is unable to represent XOR and XNOR gates with less than three AIG nodes each. As a result, the current synthesis method is considered insufficiently robust to represent real-world designs that are typically nonlinear in nature [18, 23]. The disjunctive normal form method is another alternative approach for synthesizing and realizing digital circuits incorporating all possible input combinations to achieve the required output. Although this method is accurate, it is highly exhaustive in process [14, 15] and hence is not a feasible solution for realizing higher-order circuits. An efficient two-level heuristic minimization scheme via a logical regression technique was investigated for structural synthesis in [24]. However, it fails to incorporate a high number of XOR and XNOR gates whenever required, especially for the accumulator and its associated system design. The synthesis scheme adopts a Karnaughmap-driven minimization function but does not account for realizing the function through custom cells. The evolutionary algorithm fulfills the requirement of thoroughly exploring the design-space to arrive at an optimal hierarchical structure, in low synthesis time, and it also allows to customize the list of gates to be employed for designing hierarchical design. Cartesian genetic programing (CGP) is an evolutionary algorithm that explores two-dimensional design space to evolve a list of nodes representing connected gates for the desired functions [18, 25]. As an iterative process, CGP evolves by picking gates from the predefined list and converges to a gate-level design to closely match the specified fitness [25]. The CGP approach suitably fits to express functional blocks using standard logical elements on a plane [26]. CGP considers the given truth table specified with all possible input–output combinations to render a gate-level structure under the user-configured design resolution and fitness parameters. CGP attempted to synthesize smaller approximate digital block functions in the past [27]. In the past, the evolutionary algorithm was applied to design 4-bit multipliers, 25-input median circuits [28], and 8-bit multipliers using basic logic gates [29]. The proposed work aims to establish quick realization of complex nonlinear functions of different data formats, range, and precision, using a low number of gates and yet maintaining the desired nonlinear profile. CGP has the ability to incorporate selected gates to generate the intended function and its approximate derivatives by modifying the fitness parameter. Other popular methods, including CORDIC and piecewise-linear (PWL), have shown the capability to realize a few complex functions in the past [30–35]; however, function implementations are predefined, and these methods do not allow for realizing any new functions on the fly. Hence, these methods do not satisfy toward building a generic framework for realizing new complex functions. Besides, these methods do not completely explore the design space and thereby concede more hardware resources, as shown in [36]. Hence, the CGP method works as a suitable candidate for synthesizing optimal hierarchical gate-level constructs by exploring large design space with the predefined list of custom gates and seamless integration with the ASIC flow. In this paper, CGP as a synthesis tool using a predefined set of gates is discussed and demonstrated for six activation and three power functions. A predefined set of 4-input gates comprising both basic and compound gates (MUX, AOI, and

OAI) was used for generating a hierarchical netlist, which is further flattened to 2-input gates to obtain a flattened netlist. The hierarchical results of all six activation functions and power functions are presented and compared with the results of the flattened netlist and directly CGP-synthesized basic gates. An elaborate investigation of various configurations of CGP, including a list of gates employed, desired data-format, size of the gates utilized along the nodes, and fitness-and-mutation strategies employed for realizing complex functions, is performed for the first time. The hardware characteristics and error metrics for all six activation functions and power functions are made freely available in [37] for further usage by the research community. This is the first time a comprehensive list of nonlinear and hard-to-realize functions was synthesized and characterized at a hierarchical level using predefined gates.

The contributions of this paper are listed below:

(i) Two major modifications to the traditional CGP method in a view to improve its usage for synthesizing nonlinear functions, which are (i) improve the speed of evolution using variable mutation rate and (ii) achieving circuit output to be highly correlated by using binary-weighted fitness (*BwF*) function.

(ii) Perform a comprehensive review of CGP configurations (fitness function, evolution strategy, and mutation scheme) across a wide range of bit-widths and data formats.

(iii) Benchmark the modified CGP for realizing basic nonlinear functions (x^2, x^3, x^4) and extend the same to six complex activation functions.

(iv) Explore suitable methods to integrate CGP into the standard cell synthesis flow experiment with both basic gates and popular standard cells.

(v) Highlight the improvement in fitness convergence and correlation of circuit output with expected output.

(vi) Analyze the impact of data format on the evolution process, and comparison of cell usage in different standard cell libraries.

2. Proposed Method

2.1. CGP for Synthesizing Complex Functions. CGP is a type of genetic programing that is specifically used for designing combinational digital circuits [23, 38, 39]. It creates directed acyclic graphs that function as combinational arrays of gates, which represent the digital circuits. The nodes in the graph are part of a predetermined list of gates used for building the circuit. Each node is identified by 2D coordinates on the graph. In CGP, a directed acrylic graph (DAG) consisting of an array of gates is evolved iteratively for the desired function [38]. Nodes of the DAG are gates picked from the predefined list employed to build the combinational circuit. The DAG, which is defined using an encoded genotype, is mutated repeatedly in an attempt to obtain the corresponding phenotype (digital circuits) that performs better in the desired objective functions. Typically, the objective functions are configured to the functional correctness of the circuit, hardware performance metrics (area, delay, or power of the circuit), or a multiobjective function combining both functional correctness and hardware metrics.

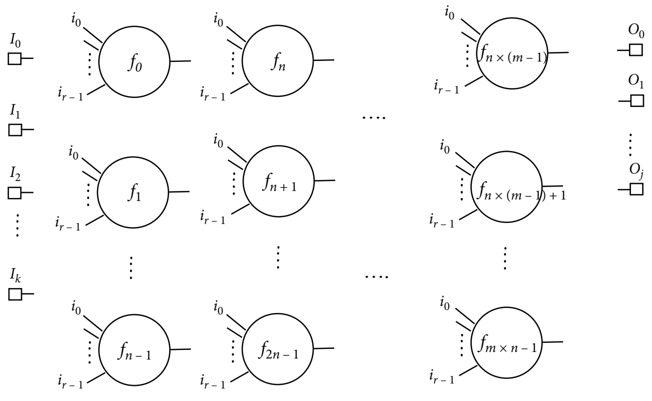


FIGURE 1: A schematic representation of Cartesian genetic programing (CGP) showing 2D DAG for inputs I, generating outputs O.

Figure 1 shows a general node-level representation of CGP DAG, where each node represents a functional block. The graph represents a circuit with $k+1$ inputs, $j+1$ outputs, and $m\times n$ computation nodes arranged in n rows and m columns. For n rows and m columns, each node is a functional gate selected from a predefined set of $f\in F$. The arity r defines the number of inputs for each node. In this work, the function set F is a combination of 4-input gates and other basic standard gates of 2-inputs. The strength of CGP lies in its ability to obtain hardware-efficient circuits through heuristic search-space exploration. However, as the complexity of intended design requirements increases, the evolutionary approach may fail for gate counts above 10K or require excessive time to generate gate-level circuits [40]. To address this issue, a cut-based method for evolutionary synthesis was proposed in [40], which offers fewer gate designs at the expense of slower convergence. Another approach involved CGP being applied to extracted subcircuits and placed back into the original circuit for global-level optimization, but selecting subcircuits for CGP-driven resynthesis can be time-consuming for larger sizes, and small subcircuits may not offer significant benefits [40]. Furthermore, a search-based strategy that adopted CGP to resynthesize circuits for FPGA system design was presented in [41]. Overall, CGP provides better tradeoff metrics between hardware design parameters and error metrics than traditional methods [38], but the synthesis runtime is exceptionally high for a large number of inputs and complex nonlinear functionalities [42]. Additionally, the increased use of machine learning and artificial intelligence techniques mandates higher-order computations in 32- or 64-bit data formats [43]. Consequently, faster genetic programing algorithms are required to evolve hardware designs with a large number of inputs without compromising fitness. However, higher-order functional design is likely to impose tight constraints on fitness because even a slight deviation can induce significant errors in processed data. Thus, there is an immediate need to speed up the CGP algorithm while maintaining fitness. In the past, CGP and its variants have

Table 1: CGP parameters.

Parameter	Value	Description
No. of generations	$G > 0$	The maximum no. of generations in an evolutionary run
Nodes (n)	$n > 0$	Number of usable nodes for the algorithm
Arity	$A > 0$	Arity of each node
Function Set	—	Available functions to be used as nodes
Mu	$\mu > 0$	No. of parents passed to the next generation, also used to create children
Lambda	$\lambda > 0$	No. of children created using μ parents

```
forall $i$ such that $0 \leq i < (\lambda + 1)$ do
    Randomly generate individual $i$
end
Select $\mu$ individuals, that are promoted as the parents.
while the generation limit is not reached do
    forall $i$ such that $0 \leq i < \lambda$ do
        Mutate the $\mu$ parents based on the mutation scheme to generate $\lambda$ offsprings
    end
    Generate the fittest individual using the following rules:
    if an offspring genotype ranks better in a selection scheme then
        Offspring genotype is chosen as fittest
    else
        The parent chromosome remains the fittest
    end
end
```

Algorithm 1: $(\mu + \lambda)$ evolutionary strategy.

been extensively studied and analyzed [44, 45]. To provide a brief summary, various CGP variants, such as graph-based crossover and mutation operators, have been discussed in [44]. However, the use of multiple mutations on multiple threads or processes leads to higher computational resource consumption due to context switching, and achieving a quick graphical crossover depends heavily on the initial seeding and correlated mutation operators. Modular CGP is another version that utilizes additional mutation operators to reevaluate and resynthesize CGP-encoded subfunctions. However, evolving to a subfunctional genotype initially requires significant computational resources, and evolved subfunctions may not converge to an optimized solution, leading to functionally incorrect designs. An alternative approach involves taking the CGP phenotype to machine code for faster execution, resulting in a speedup of the CGP runs executed on hardware units such as FPGAs and application-specific virtual reconfigurable circuits [46, 47]. However, most modifications aim to add more operative dimensions or shift execution to a different platform without addressing the speedup mechanism to arrive at fitter solutions. Previously, the authors explored and demonstrated the desired modifications to the existing CGP configuration, namely *BwF* and exponentially varying mutation rate (*eVar*), which can produce functionally correct solutions at a remarkably fast pace [48]. The advantages of the modifications are showcased for basic nonlinear power functions and are verified for usage in activation functions that are otherwise difficult to achieve. This paper further elaborates on the distribution of the gates picked for hierarchical synthesis, seamless integration to the regular ASIC synthesis flow, data format impact on the synthesis, and node representation impact on the synthesis. This research provides an in-depth investigation of 12 different CGP configurations with alterations in mutation schemes and evolutionary strategies for power functions. The article presents and discusses a comparison between the benefits of *BwF* and *eVar* adopted CGP versus traditional CGP methods in detail. The parameters listed in Table 1 are utilized to configure the CGP algorithm for synthesis. Algorithm 1 presents the $(\mu + \lambda)$ evolutionary strategy steps, which return the fittest genotype in an algorithm run for randomly generated seed candidate solutions. The μ value indicates the number of solutions promoted to the next generation as parents. The λ value indicates the number of offsprings generated from the μ parents from the previous generation.

In this work, different evolutionary strategies were analyzed to identify the ideal μ, λ values for achieving faster evolution of nonlinear digital circuits. To showcase the impact of a novel mutation rate and modified fitness function, power functions were realized using the modified features of the CGP technique. The original CGP with constant mutation rate and traditional fitness function for the same functions were also evaluated and compared against the

Table 2: CGP configurations.

Configuration	Possible choices	Typical choice
Evolutionary strategy	$(\mu + \lambda)$	$(1 + 4)$
Mutation scheme	Probabilistic, probabilistic only active, point, single	Point
Selection scheme	Select fittest, Tournament	Select fittest
Reproduction scheme	Mutate every parent, Mutate random parent	Mutate every parent
Fitness	Supervised learning (SL), binary-weighted fitness (BwF)	Supervised learning
Mutation rate	Constant, variable	Constant

proposed modified CGP techniques. All possible options and typically selected configurations are listed in Table 2. The four mutation schemes vary in terms of the mutations performed on the node and chromosome level. The mutation schemes are elaborated below.

Probabilistic: Conducts probabilistic mutation on the given chromosome. Each chromosome gene is changed to a random valid allele with a specified probability.

Probabilistic only active: Conducts probabilistic mutation on the active nodes in the given chromosome. Each chromosome gene is changed to a random valid allele with a specified probability.

Point: Conducts point mutation on the given chromosome. A predetermined number of chromosome genes are randomly selected and changed to a random valid allele. The number of mutations is the number of chromosome genes multiplied by the mutation rate. Each gene has an equal probability of being selected.

Single: Conducts a single active mutation on the given chromosome.

In this work, novel *BwF* function and *eVar* are employed for running CGP. The other configurations, including evolutionary strategy, mutation scheme, selection scheme, and reproduction schemes, were also investigated for the synthesis application. Select Fittest selection scheme and Mutate Every Parent reproduction scheme were incorporated for all the evolutionary runs. As indicated in Table 2, typical CGP in the form of supervised learning (SL) and constant mutation rates were compared against *BwF* and variable mutation rate (*eVar*) CGP technique toward synthesizing digital circuits of unconventional functions.

2.2. Modified CGP.

This work proposes two new modifications to the existing CGP methodology to achieve the desired synthesized gate-level circuits with lesser generations and extract a group of fitter solutions. These two modifications are listed: (i) Applying a different mutation rate, similar to [49]. Especially for realizing nonlinear functions instead of a constant mutation rate adopted in the traditional methods, and (ii) Incorporating a *BwF* function instead of the same weight across the data format of the functions under synthesis. The proposed modifications to the CGP algorithm are particularly beneficial for designing hardware that can perform nonlinear functions, which are essential for the modern-day neural network and other computational networks. The authors demonstrate the effectiveness of their approach by implementing two types of nonlinear functions—power functions and activation functions. The work introduces two novel features, namely the exponential mutation rate (*eVar*) and the *BwF* function, for synthesizing nonlinear functions. To the best of the authors' knowledge, this is the first time that a varying mutation rate and a *BwF* function have been applied and analyzed for circuit synthesis across different data formats.

2.3. *eVar*.

In traditional CGP, the mutation rate is configured to be a constant value in a view to have the same mutation rate across the generations and continue to evolve until a desired fitness or termination criteria is achieved [38]. In this work, we intend to formulate a mechanism that intuitively selects between rough topologies in the initial generations and then mutates subparts of the best-performing circuit topologies, followed by minimal and single gate-level selection in the final generations. This mechanism is inspired by varying learning rates in various ML optimization algorithms during the training phase. The fitness function used can achieve the selection between the rough topologies in the initial generations and subsequently among the finer subparts of the designs in the later runs. We tried several types of mutation rate variations, including linear and various nonlinear decays. It is observed that an exponential variation of the mutation rate converges fastest to functionally correct circuits.

$$\text{Mutation, rate}(g) = R \times e^{\frac{-g}{0.1 \times G}}. \tag{1}$$

The exponential variation of the mutation rate used is stated in Equation (1), where G = total number of generations, g = current generation, and R = initial rate of mutation. It is observed that a decay rate of 10 is suitable based on our heuristic studies on evolving nonlinear functions. If the mutation rate reduces lower than one gene, the mutation scheme was changed to mutate at least one gene for further generations to continue the evolution process. The sequence of a first-up rough design requires high mutations to evolve to a relatively adequate design topology. Further fine-tuning the design topology with fewer mutations requires lower mutation rates. The process of following *eVars* was found to inherently reduce the evolution time to achieve similar fitness as other variations. The selection between multiple mutated designs at all generations is driven by the configured fitness function.

2.4. *BwF*.

BwF was introduced to evolve designs that are closer to the data-formatted functional output. The traditional SL fitness can either be devised to formulate a minimizing or maximizing objective function for the evolutionary algorithm. For the minimizing case, SL uses the Hamming distance between the evolved circuit output data-bits and expected output data-bits over all input combinations, indicating the total number of incorrect circuit output data-bits. In the maximizing case, the number of correct output data-bits of the evolved circuit is used as the fitness score. In this work, we use the minimizing case, as stated in Equation (2), where z is the number of inactive nodes in the solution circuits.

$$\text{SL fitness} = \begin{cases} b \text{ when } b > 0 \\ b - z \text{ otherwise} \end{cases}. \tag{2}$$

This method continues to evolve designs till the number of generations is exhausted, even if a functionally correct circuit is found while trying to minimize z. We minimize the circuit size by reducing the number of active gates. However, running the SL-configured CGP method to evolve the best-fit design mandates millions of generations, especially for realizing nonlinear functions under investigation. Besides, the SL method adopts a similar Hamming distance rule across all bits, leading to smaller refinements along all the design paths associated with the output bits. The SL process ceases to drive major topological changes for the most significant bits associated with design paths and thereby lags in evolving best-fit design solutions in consolidation with all the output bits of the investigated nonlinear functions. *BwF* attempts to improve the selection among the available solutions by considering the weighted fitness across the output bits of the design evolved instead of considering the Hamming distance across all the established output bits. The *BwF* fitness function is stated in Equations (3) and (5). The binary-weighted sum (BWS) is a weighted score, giving positional weightage to an error in individual output bits derived from the evolved circuit design (C(Oi)) when compared with the expected (E(Oi) functional output, where *i*th output bit is scaled by the magnitude of error contributed. BWS of the overall input combinations was applied to evaluate the *BwF* for a solution. We also include the term z to minimize the circuit size post a functionally correct circuit is evolved.

$$\text{BWS} = \sum_{i=0}^{n_o - 1} 2^i \times \delta_i, \tag{3}$$

$$\text{Where } \delta_i = \begin{cases} 1 \text{ if } O_{\text{Exp}}(i) \neq O_{\text{Cir}}(i) \\ 0 \text{ if } O_{\text{Exp}}(i) \neq O_{\text{Cir}}(i) \end{cases}, \tag{4}$$

$$\text{BwF fitness} = \begin{cases} \text{BWS when BWS} > 0 \\ \text{BWS} - Z \text{ otherwise} \end{cases}. \tag{5}$$

3. Experimental Results

The modifications proposed for CGP in this work are particularly useful in realizing hardware designs for nonlinear functions, which are crucial for modern neural networks and other computational networks. To demonstrate the impact of the proposed CGP method, this work showcases the hardware realization of power functions. Detailed experimental results on realizing power functions with different CGP configurations, including evolutionary strategy and the impact of gates for synthesizing three of the power functions (x^2, x^3, and x^4), along with corresponding error metrics, are highlighted in this section. In the next section, the best possible CGP configuration is employed to realize six nonlinear activation functions for different input and output data-format structures. The CGP-integrated ASIC synthesized hierarchical netlist of activation functions is evaluated with respect to the fitness of the function realized, hardware resources utilized for realizing the functions, and the impact on the data format. Besides, a section on the comparison of activation functions with other hardware implementations is added to understand the significance of the proposed method over other state-of-the-art implementations. In the early generations, *BwF* helps select the most functionally equivalent topology among the solutions. This is because *BwF* enables the selection of topologies with the least deviation in the output function from the expected function, which was not possible in the SL method due to its equal weightage rule associated with the output bits. The evolution process slows down with fewer mutations in the later generations, only modifying small subparts of the circuits. *BwF* continues to evolve the designs at the gate-level toward the final generations. The authors observed that using *BwF* along with the *eVar* mutation rate not only produces circuits with the least deviation from the expected nonlinear function but also evolves the circuit with fewer generations. To showcase the impact of *BwF* and *eVar* adoption in CGP, we synthesize circuits of power functions and activation functions, which are difficult to synthesize otherwise. For each configuration discussed in Section 3, the authors ran 20 experiments with the same generation limit for different configurations. For example, the 5-bit x^3 has 12 configurations of mutation schemes and evolutionary strategies. Both *BwF* and SL with these 12 configurations are run 20 independent times to obtain the average behavior of circuits evolved with the specific configuration. The results discussed in the following sections are just an illustrative subset of all the experiments, and the complete comprehensive set of results is available in [37].

3.1. Error Analysis. Eight different error metrics, including mean absolute error (MAE), error probability (EP), standard deviation (STD), mean relative error (MRE), median of absolute error (MeAE), mode of absolute error (MoAE), maximum absolute error (Max-AE), and minimum absolute error (Min-AE) were investigated for *eVar*, and *BwF*-modified CGP and traditional CGP configurations for the power

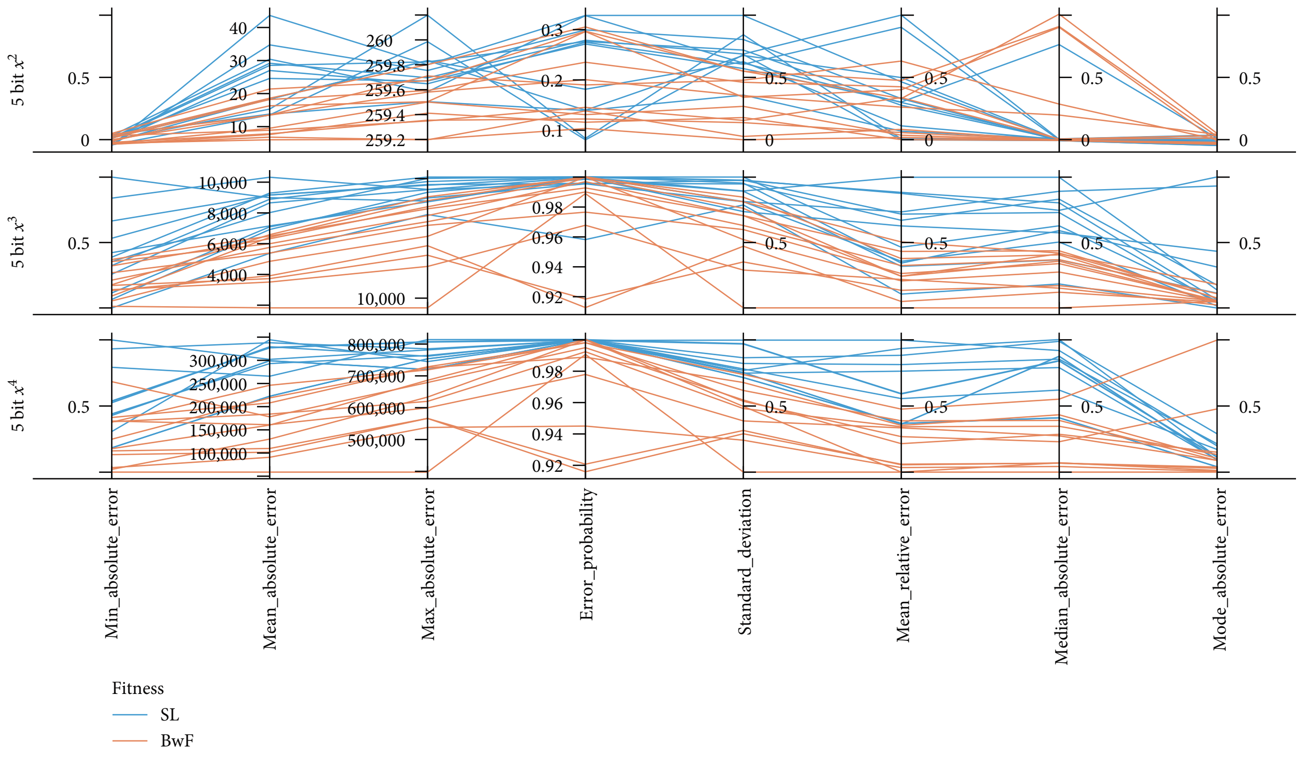


FIGURE 2: Error metrics comparisons for BwF versus SL methods, when employed with all possible CGP configurations, for generating 5-bit power functions: x^2, x^3, x^4.

and activation functions. The error metrics are defined in Equation (6).

$$\begin{cases} \mathrm{EC} = \sum\limits_{\forall x \text{ in } R} \left(O_{\mathrm{Exp}}(x) \neq O_{\mathrm{Cir}}(x) \right) \\ \mathrm{MAE} = \dfrac{\sum_{\forall x \text{ in } R} \left| O_{\mathrm{Exp}}(x) - O_{\mathrm{Cir}}(x) \right|}{2^N} \\ \mathrm{EP} = \dfrac{\mathrm{EC}}{2^N} \\ \mathrm{STD} = \sqrt{\dfrac{(\mathrm{AE}(x) - \mathrm{MAE})^2}{2^N}} \\ \mathrm{Max\_AE} = \max\left(\left| O_{\mathrm{Exp}}(x) - O_{\mathrm{Cir}}(x) \right| \right) \forall_x \text{ in } R \\ \mathrm{Min\_AE} = \min\left(\left| O_{\mathrm{Exp}}(x) - O_{\mathrm{Cir}}(x) \right| \right) \forall_x \text{ in } R \\ \mathrm{MRE} = \dfrac{\sum_{\forall x \text{ in } R} \frac{\left| O_{\mathrm{Exp}}(x) - O_{\mathrm{Cir}(x)} \right|}{\max\left(1, O_{\mathrm{Exp}}\right)}}{2^N} \end{cases} . \tag{6}$$

Figure 2 illustrates the 5-bit data-format for power-two (x^2), power-three (x^3), and power-four (x^4) functions using the proposed *eVar* and *BwF*-modified CGP. To evolve the power function for 5-bits, three mutation schemes—$(1 + 10)$, $(2 + 8)$, and $(5 + 5)$ with four different evolutionary strategies—*Probabilistic*, *Probabilistic Only Active*, *Point*, and *Single*, were independently configured. Each colored line represents the average error metrics of 20 runs for a given configuration among the 12 investigated. Compared to the SL-driven CGP method, which employs a constant mutation rate and equal-weighted fitness, most of the circuits generated by the proposed *eVar* and *BwF*-modified CGP method exhibited lower error metrics for different evolved circuits. The reduced error metrics confirm that the *eVar* and *BwF*-configured CGP approach produced fitter circuits than the SL-based CGP approach.

3.2. CGP Configuration Analysis. Figure 3 shows the comparison between *BwF* and SL-generated circuits for all the configurations. The $(5 + 5)$ evolutionary strategy consistently performs better than the $(1 + 10)$ and $(2 + 8)$ strategies. The $(2 + 8)$ strategy is also found to perform better than $(1 + 10)$ strategy. This result hints that creating fewer child solutions per parent in every generation is preferred. Fewer children per parent imply a higher number of circuit topologies are being carried forward to further generations. The mutation scheme has a large impact on the generated circuits MAE when using the SL a fitness function. Furthermore, there is no clear correlation between the mutation type and MAE over the functions (x^2, x^3, x^4) and bit width. *BwF* as fitness function is found to minimize the variation in MAE with the mutation type selected. *Single* and *Point* mutation types perform better than *Probabilistic* and *Probabilistic Only Active* when *BwF* is used. *BwF* across 12 mutation schemes emerges with a clear accuracy advantage ranging from 5% to 50% of less MAE for x^3 and x^4 power functions when compared with the SL method. The same is not established for x^2 power functions, which are smaller designs. Hence, *BwF* is preferred to realize functions with higher bit-width and complexity.

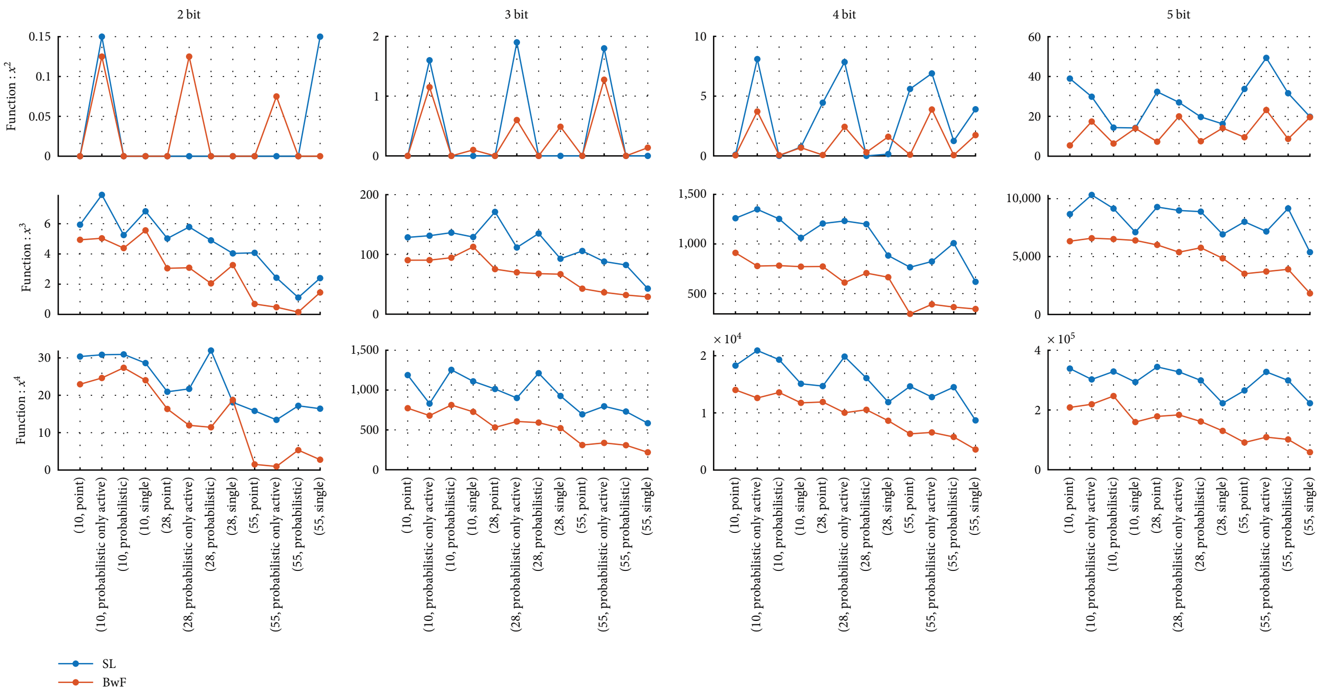


Figure 3: Comparison of CGP configurations w.r.t mean absolute error (in y-axis). Configurations are pairs of evolutionary strategy ($10 \longrightarrow (1+10)$, $28 \longrightarrow (2+8)$, $55 \longrightarrow (5+5)$) and mutation scheme. BwF performs better than SL for all CGP configurations.

3.3. CGP Configuration Impact on Synthesized Design. In the conventional configuration of a CGP run, SL is used with constant mutation rates. However, this work introduces a modified configuration that produces fitter circuits at a faster convergence rate. The modified configuration involves *BwF*, which evaluates the output bits with assigned binary weights during each evolution. This approach selects the circuits to mutate further in each generation by giving weighted importance to higher-order output bits relative to lower bits. Additionally, an *eVar* is used to reach functionally fitter circuits faster. The goal is to select the best rough topologies in the initial generation that require high mutation rates and, in later generations, prefer minimal and single gate level changes, which demand low mutation rates. The varying learning rate is inspired by various machine learning algorithms used in the training phase. The use of *eVars* was found to inherently reduce the evolution time to achieve similar fitness as other long duration configurations. The selection between multiple mutated designs in each generation is driven by the *BwF* strategy. The *BwF* and *eVAR* strategies are used to realize circuits through CGP runs for all the functions under consideration. A series of 5-bit power functions (x^2, x^3, x^4) are realized to validate the results of the adopted strategy (*BwF* + *eVAR*) with the typical approach (SL + Constant). Finally, a study on different evolutionary strategies—(1 + 10), (2 + 8), and (5 + 5)—derived combinatorial circuits, and mutation schemes *Point*, *Probabilistic*, *Probabilistic Only Active*, and *Single* types for 5-bit power functions are presented.

Post-CGP run, the obtained phenotype represents combinational circuits. The distribution of six types of gates picked from the predefined list to derive the circuit is shown in Figures 4 and 5. Figure 4 indicates that (2 + 8) and (5 + 5) evolutionary strategies have less active nodes and are more hardware-efficient than (1 + 10) strategies. Less gate sizes at the synthesis stage are likely to offer hardware benefits in terms of silicon footprint, power, and delay parameters. Within different schemes studied, preference of (5 + 5) over (1 + 10) suggests that more parents in the form of circuit topologies are carried forward to further generations during the evolutionary runs.

The distribution of gates in % remains similar throughout the three strategies investigated for all three power functions. Figure 5 shows that all mutation schemes have a similar distribution of the gates. *Probabilistic* scheme results in the largest gate size irrespective of SL or *BwF* fitness functions. An important observation is the distribution of gates in the evolved circuit. OR and NOR gates are the highest occurring, while XOR and XNOR are the least occurring. AND and NAND gates are relatively less used than OR and NOR gates. This result clearly indicates that while the circuit size remains similar, the accuracy of output improvement of *BwF* over the SL method makes *BwF*, a favorable synthesis option. Hence, *BwF* configuration with (5 + 5) mutation strategy and *eVAR* mutation rate, along with a single type is configured for realizing the functions under consideration. The proposed *BwF* and *eVar* methods led to faster synthesis of circuits and gate-level designs that produced nearly exact output. The *eVar* mutation rate facilitated faster exploration of the design space, resulting in functionally correct solutions. The *BwF* feature aided in selecting and driving evolutionary designs toward weighted fitness along output bits, resulting in reduced error metrics compared to the SL technique. Among the evolutionary strategies, (5 + 5) was preferred due to its low gate count and best fitness for realizing

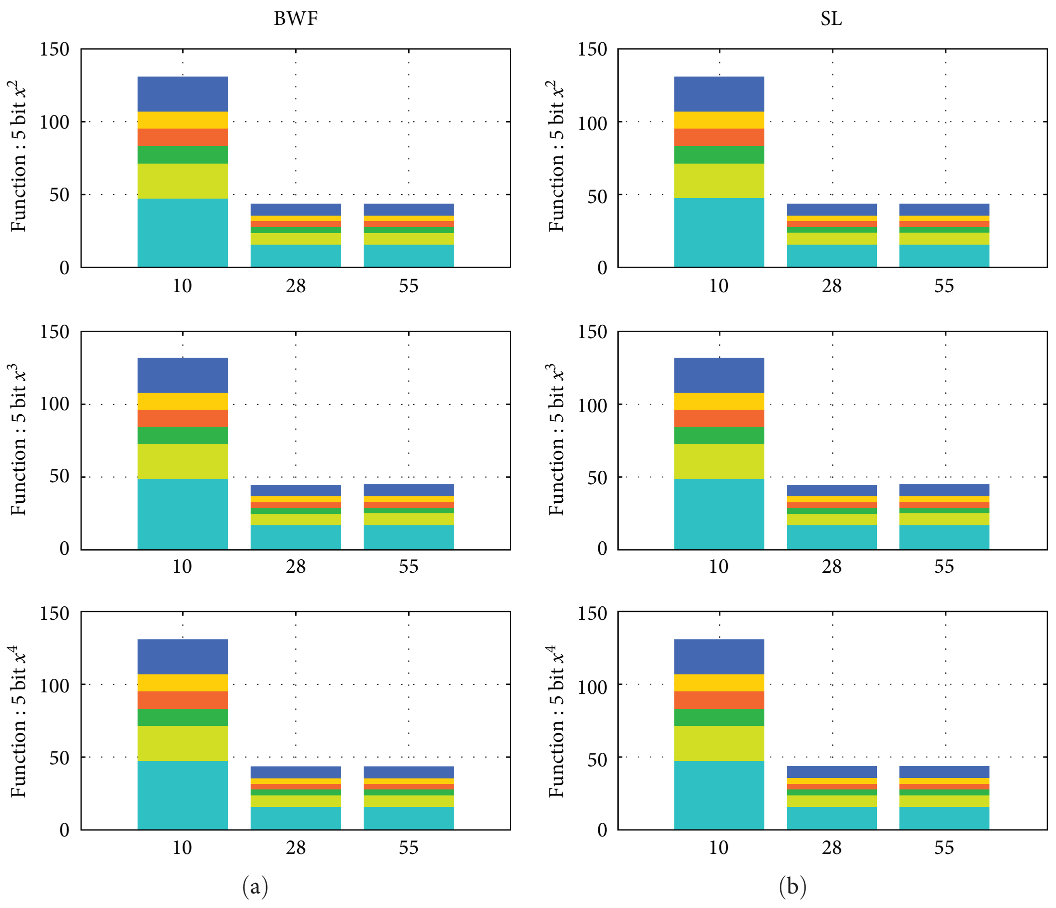


FIGURE 4: Distribution of gates in the evolved 5-bit circuits for power functions (x^2, x^3, and x^4), with (a) showing circuits obtained using BwF, and (b) showing circuits using SL for evolutionary strategy (10 $\longrightarrow$ (1 + 10), 28 $\longrightarrow$ (2 + 8), 55 $\longrightarrow$ (5 + 5)).

power functions. The *BwF* approach improved fitness by reducing MAE by 5% to 50% for x^3 and x^4 functions compared to the SL method.

With respect to CGP implementation, we have demonstrated two important modifications, such as *BwF* and *eVar*, as compared to SL, which employs the same weight across all data-bits and a constant mutation rate for evaluating the fitness parameters. Figures 2–5 depict the three power functions evolved using *eVAR* and *BwF* and compared against equal weighted fitness and constant mutation rate, which clearly demonstrates that the proposed method converges faster and yields fitter circuits. *eVar* contributes in evolving circuits faster, whereas *BwF* contributes to realizing fitter circuits. Adopting a mix of *BwF* with a constant mutation rate is likely to yield fitter circuits but demands a large convergence time, whereas *eVar* with equal weighted fitness is likely to converge faster but not necessarily comply with the required fitness.

4. Application: ASIC Synthesis of Activation Functions

4.1. CGP Integrated ASIC Synthesis. Figure 6 illustrates the proposed usage of a faster CGP synthesis process to generate a fitter hierarchical design of unconventional functions in the ASIC standard cell synthesis flow. This method produces a pure combinational circuit using a predefined set of gates. Figure 6 presents two flows to integrate CGP-generated netlists into standard ASIC synthesis. In *Path-A*, the evolved netlist is directly included as a subdesign or submodule to the original Verilog design file. In *Path-B*, CGP-evolved netlists are treated as hierarchical modules in the top-level design hierarchy. Here, CGP runs are expected to run in parallel to the synthesis of other designs, treating the instantiation as a subdesign that gets integrated into the top design during global partitioning and optimization. While *Path-A* is a better choice for obtaining the most efficient synthesis result, *Path-B* is preferred when the evolved circuits are large in size and require a long run-time to arrive at a solution.

4.2. Optimization of Evolved Circuits. The netlist obtained from CGP is not optimal in terms of cell usage. The netlists may contain gates that are either redundant or are not driving any other node. While pruning of these logic gates is performed by the synthesis tool in the *Path-A* flow, it is necessary to clean the evolved circuit when targeted for the *Path-B* flow before applying it in the top design. Following pruning techniques are applied to the evolved netlist.

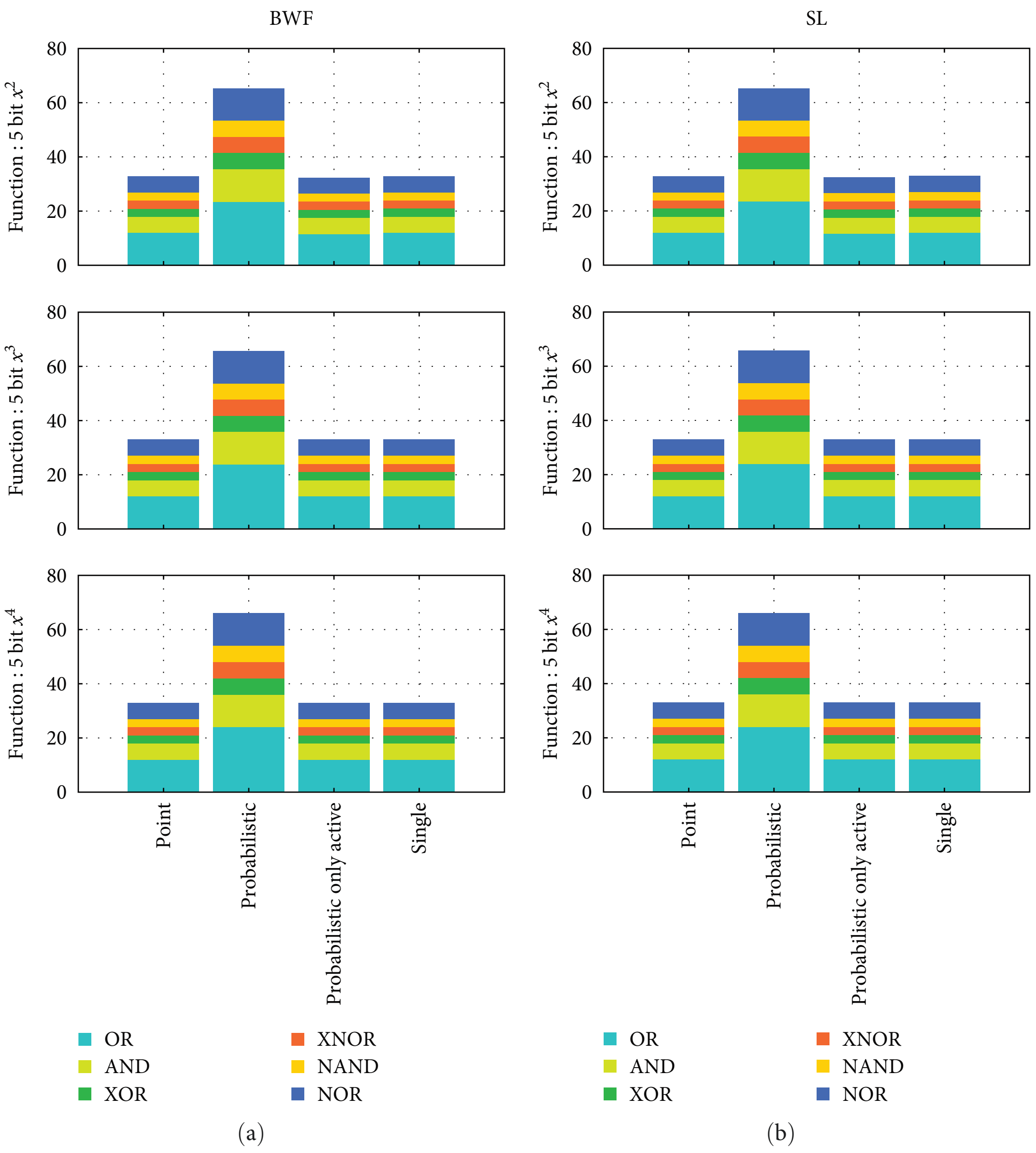


FIGURE 5: Distribution of gates in the evolved 5-bit circuits for power functions (x^2, x^3, and x^4), with (a) showing circuits obtained using BwF, and (b) showing circuits using SL for different mutation rates. Similar distributions are observed for 2-bit to 4-bit circuits.

(i) Noncoding nodes: The final genotype obtained from CGP may contain noncoding nodes that are needed for the evolution process [50] but are not utilized to drive other nodes or outputs. Figure 7(a) shows an example with a noncoding node ("MXI2_4"), which is not utilized elsewhere in the circuit for generating the output.

(ii) Ungrouping gates inside cells: When using the selected gates from standard cells as the function set for nodes, the final genotypes resulting from CGP become good candidates for ungrouping techniques. A few optimization techniques, such as logic sharing, boolean optimization, and redundancy removal, as shown in Figures 8(a) and 8(b), were implemented to refine the hierarchical design. One of the 2- AND gates is redundant, hence removed, and the input is directly fed to the next stage, as depicted in Figure 8(a). Logic sharing between the two outputs is shown in Figure 8(b), where the two 2-AND gates followed by 2-OR are repeated, and hence the same is shared to generate the output.

4.3. Six Activation Functions. Activation functions have nonlinear output profiles and are considered unconventional constructs for traditional synthesis. Hence, they were considered appropriate to showcase the capability of the novel proposed CGP synthesis tool. In addition to using classical 2-input basic gates, the following list of 20 different and popular 4-input gates was used as node functions to synthesize activation functions: AND, OR, NAND, NAND4B, NAND4BB, NOR, NOR4B, NOR4BB, XOR, XNOR, MUX, MUXI, AOI211, AOI22, AOI2BB2, AOI31, OAI211, OAI22, OAI2BB2, and

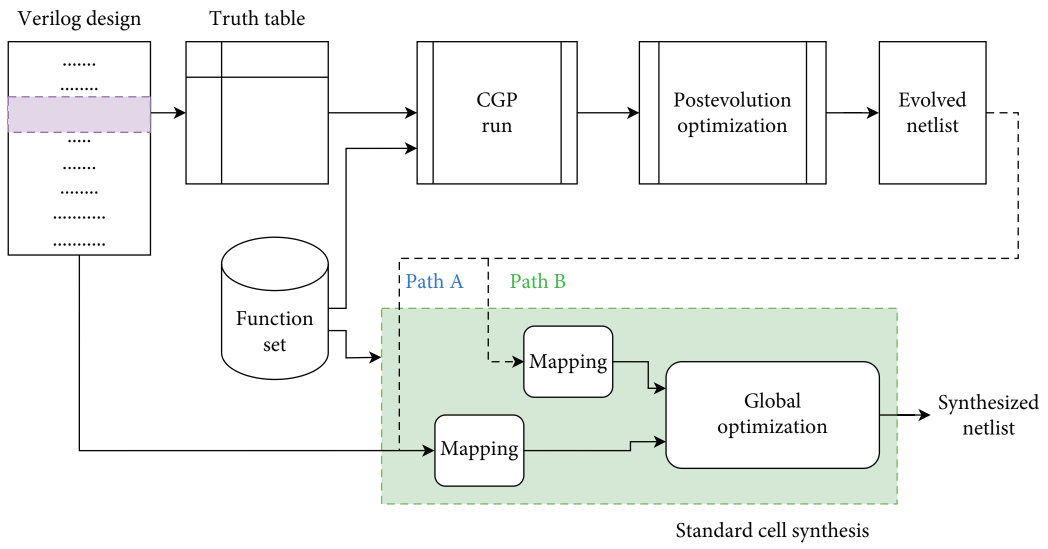


FIGURE 6: Schematic showing a synthesis framework for integrating the hierarchical design of unconventional functions using the CGP method to ASIC flow.

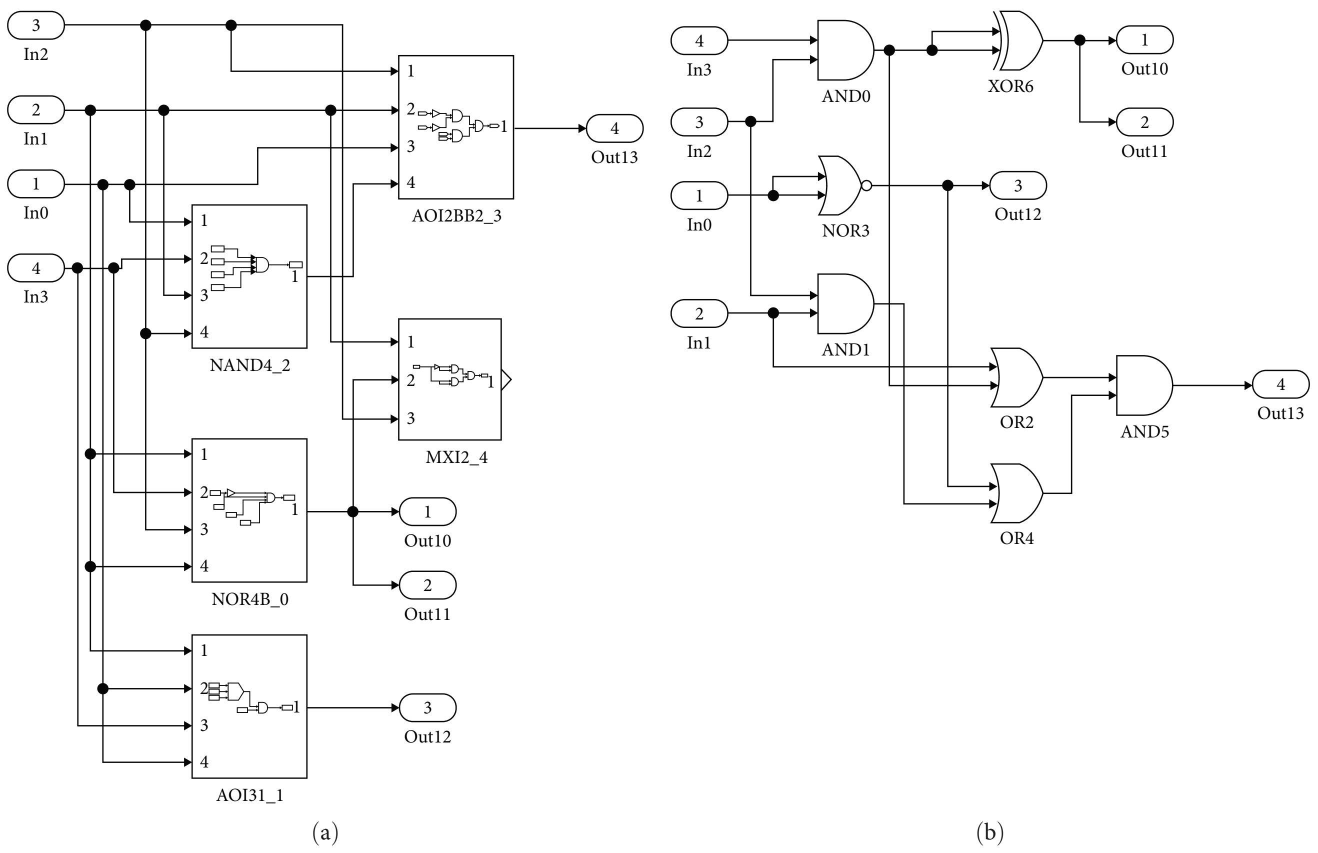


FIGURE 7: Example showing CGP evolved circuit for 4-bit sigmoid function using (a) selected gates from standard cells and (b) basic gates.

OAI31. These sets of gates were selected in a view that most of the library, including skywater 130 nm, GPDK 45 nm, UMC 65 nm, and nanGate 15 nm, consists of these gates and are likely to directly map to the cells available in the library. To validate the effectiveness of the modifications, the proposed *eVar* and *BwF*-modified CGP were applied to activation functions such as Gaussian, Sigmoid, hyperbolic-tangent, ReLU, GeLU, and Softplus. Due to the space constraint, results of Gaussian, Sigmoid, and hyperbolic-tangent are presented, and all other results are made freely available in [37].

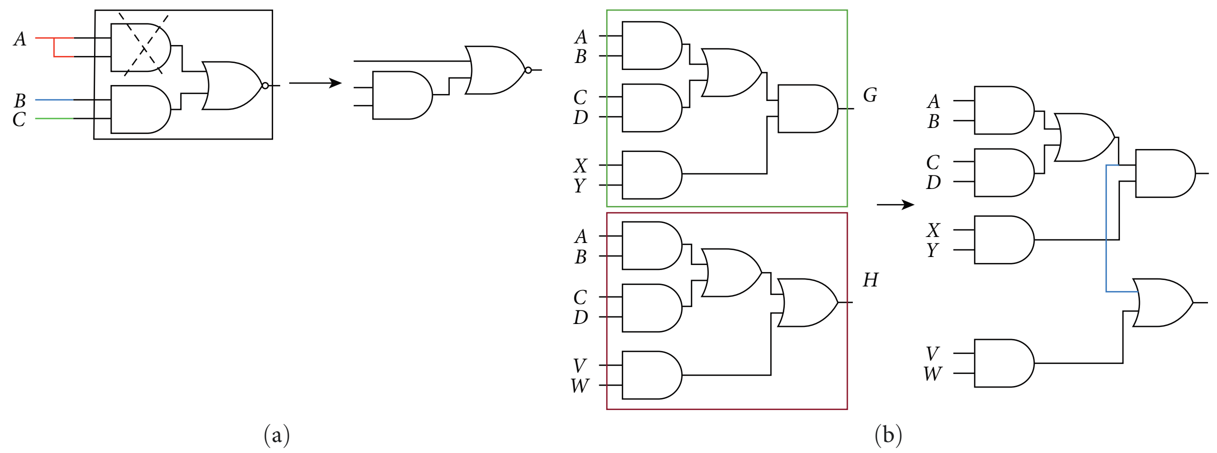


FIGURE 8: Circuit optimizations: (a) redundancy removal and (b) logic sharing, applied to the proposed hierarchical synthesis.

$$\begin{cases} \text{Sigmoid}(x) = \dfrac{1}{(1+e^{-x})} \\ \tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}} \\ \text{Gaussian}(x) = e^{-x^2} \\ \text{ReLU}(x) = \begin{cases} 0 & x<0 \\ x & x \geq 0 \end{cases} = (x \times 1)_{x>0} \\ \text{GeLU} = \dfrac{1}{2}x\left(1+\text{erf}\left(\dfrac{x}{\sqrt{2}}\right)\right) \\ \text{Softplus}(x) = \ln(1+e^x) \end{cases} . \tag{7}$$

Sigmoid, Gaussian, Hyperbolic-tangent, ReLU, GeLU, and Softplus are expressed as stated in Equation (7). Fixed-point data-format is used to represent the input and outputs of the activation function. Round-to-floor quantization scheme was used to obtain the fixedpoint representation of the continuous activation functions. Figure 9 shows the input fixed-point data format used. The output data format for the Sigmoid, Hyperbolic-Tangent, and Gaussian activation functions was configured similarly to the input data format of the Gaussian function since it offers adequate integer bits to represent the full range of output values between −1 and 1. A typical evolutionary strategy of $(1+4)$, along with a point mutation scheme, was adopted for this analysis. For invoking the CGP process to evolve the hierarchical circuit for a required design, the authors suggest a demarcation in the HDL source file to define the specification, such as:

```
// CGP {
// Function = ' Sigmoid ';
// QuantizationScheme = ' Floor ';
// WordLength = '7 ';
// DataFormat = 'FP';
// RadixPoint = '3';
// FcnSet = ' StdCells ';
// }
```

This example shows a call for running the CGP process to evolve the circuit for a 7-bit Sigmoid implementation. The Sigmoid function is quantized using round to floor quantization scheme, and data are represented using 7-bit fixed point numbers with 3 fractional bits, 3 integer bits, and 1 sign bit. CGP is also fed with the custom cells consisting of 20 different cells of 4-input gates, as listed above.

In the *Path-A* option of Figure 6, the evolved netlist can be instantiated as a submodule in the original Verilog file. If the *Path-B* option is used, the evolved hierarchical netlist is either directly mapped to standard cells and treated as a subdesign (which is independently synthesized) and integrated into the top design or flattening the evolved netlist before integrating it to the top design. Flattening the evolved netlist requires a final incremental optimization step to enable optimizations such as logic sharing and pruning. Four different data formats were used to analyze the effect of data format choice on the proposed CGP synthesis run and are presented below:

(i) Configuration 1, referred to as *Config-1*, has the highest range, with $(N-1)$ bits used to represent the integer part. A representation for *Config-1* is shown in Figure 10(a). The figure clearly depicts the format of integer data for N ranging from 4 to 8-bits.

(ii) Configuration 2, referred to as *Config-2*, was designed to allocate the maximum possible number of available bits to the fractional part. This is to achieve maximum precision while maintaining the full output range for the corresponding activation function. Hence, circuits representing the Sigmoid function have dlog2(5)$e = 3$ bits for the integer part. Circuits representing Gaussian and Hyperbolic-Tangent functions have dlog2(2)$e = 1$ bit and dlog2(4)$e = 2$ bits, respectively, for representing the integer part. Hence, based on the maximum output range, the integer part and maximum bits to represent the fractional part are allocated. Different data formats representing *Config-2* for three activation functions are shown in Figure 9(a)–9(c).

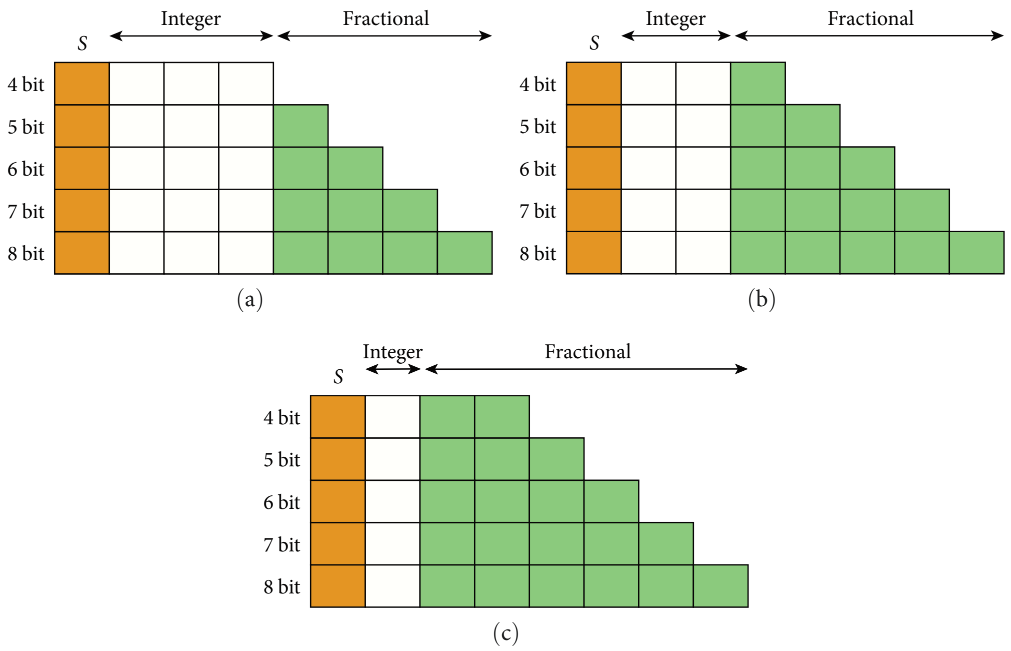


FIGURE 9: Input data format representing Config-2, that is, employed for (a) sigmoid, (b) hyperbolic tangent, and (c) Gaussian activation functions.

(iii) Configuration 3, referred to as *Config-3*, allocates N 2 bits for the fractional part, and the same is shown in Figure 10(b). Apart from the sign bit, the fractional part either occupies one bit extra or has a similar bit-width as that of the integer component.

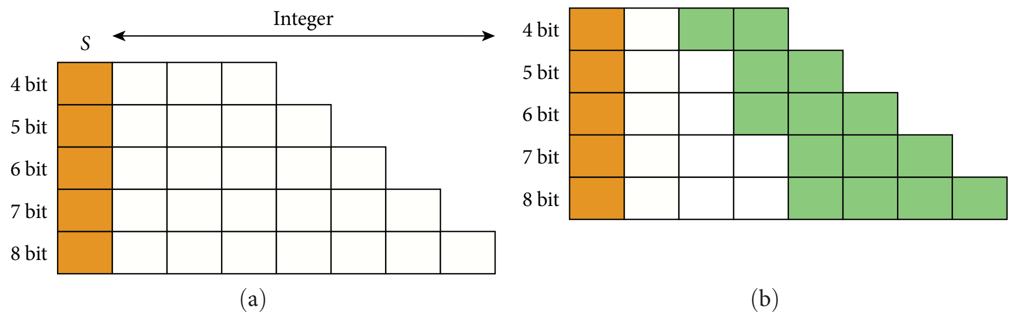
(iv) Configuration 4, referred to as *Config-4*, allocates N $2-1$ bits for the fractional part, and the same is depicted in Figure 10(c). In this configuration, the fractional part allocation is always 1-bit less than the integer component.

The four different data-formats allow to evaluate the effect of fast CGP synthesis run on the configured data-formats. The output data format for Sigmoid, Hyperbolic-Tangent, and Gaussian activation functions was configured similarly to Config-2 of the Gaussian function since it offers adequate integer bits to represent a full range of output values between −1 and 1. ReLU, GeLU, and Softplus are configured to have the same output configuration as their corresponding input configurations.

The proposed CGP synthesis flow was used to validate a comprehensive set of design variations covering a wide range of bit-widths (4–8-bits) and four different configurations applied to six different nonlinear (activation) functions. To realize this set of functions, three different synthesis runs were conducted. The first was a hierarchical synthesis run on the specified functions without flattening, referred to as hierarchical. The second was a flattened synthesis run, labeled as flattened, which was performed after flattening the hierarchical design. The third synthesis run used basic gates and was referred to as basic-gates. These three different synthesis runs allowed for a discussion on optimization and synthesis time for the selected complex functions.

Figures 7(a) and 7(b) show the circuit synthesized for the Sigmoid function using selected gates from the custom gates and basic gates, respectively. The selected gates generated circuit was further flattened using basic gates. In this work, 20 different runs for each design variant were performed to benchmark the computation effort, since the aim was to characterize the proposed synthesis flow to achieve fully-fit circuits. The fully-fit circuit refers to the one that reproduces the desired truth-table for the complex function under consideration. Due to the size constraint of this manuscript, results presented in the form of tables and figures are only a small subset of all the experiments. Complete comprehensive results are made available at [37].

4.4. Fitness Convergence. Figure 11 is the plot of percentage-of-fitness versus generations in logarithm scale for Gaussian, Sigmoid, and hyperbolic-tangent functions of 4–8-bit data-formats when configured with (blue runs) constant mutation rate with SL and (red runs) *eVar* mutation rate with *BwF*. We report the percentage of fitness since the order of magnitude of SL and *BwF* values are different. The graph also allows to recognize potential termination when a functionally perfect solution is found. The *eVar* mutation rate evolves to a similar percentage of fitness attained by constant mutation rate by at least a decade generation less for lower order bit-widths and at least 105 generations less for 7, and 8-bit activation functions. Additionally, the best-evolved design for *eVar*-based CGP consistently achieved maximum fitness for a similar number of generations throughout the three activation functions investigated. Considering the evolved design at the last generation, the percentage of fitness achieved by the *eVar*



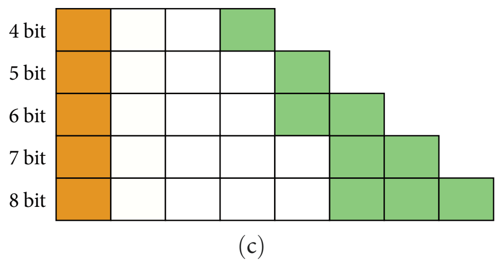


FIGURE 10: Different configurations of input data representing (a) Config-1, (b) Config-3, and (c) Config-4 are depicted, where green colored bits represent fractional values, orange colored indicates the signed position and white colored bits is assigned for integer representation. These plots represent all 5 activation functions of Config-1, Config-3, and Config-4.

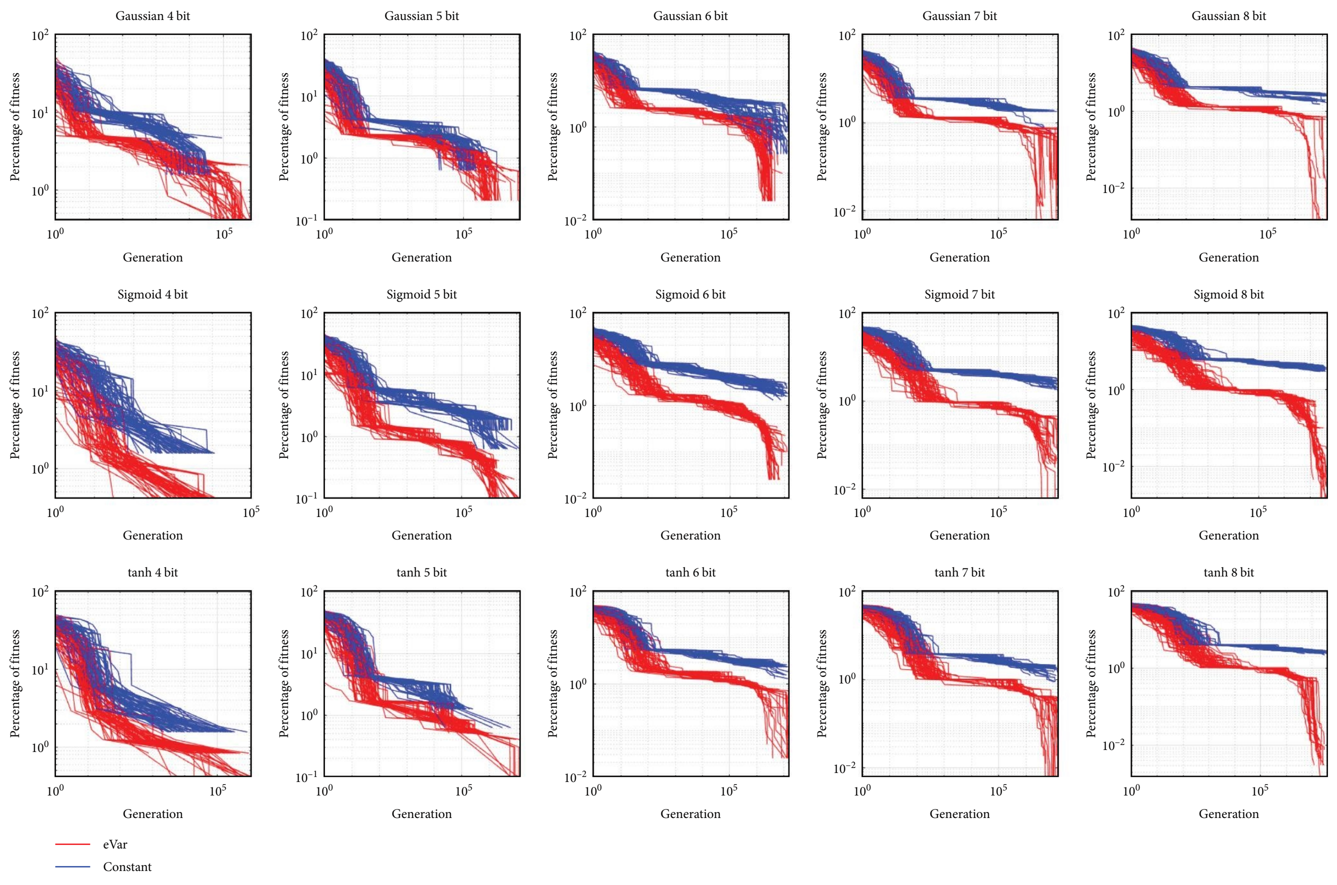


FIGURE 11: Comparison of convergence to full-fitness of constant mutation rate and exponentially varying mutation rate. Most eVar-BwF runs converge to functionally perfect solutions within the generation plotted, while SL-constant mutation rate runs require more generations.

applied CGP is better by at least a decade for lower order bit-width functions and at least 100 times better for the 7 and 8-bit activation functions.

4.5. Correlation of Circuit Output. Figure 12(a) shows the output profile of the circuit generated from the proposed scheme, which closely resembles the expected output nonlinear profile. The absolute error remains zero for most of the input data, with dominant error values close to 0. Figure 12(b) depicts the output profile of the circuit generated from the traditional SL method. It demonstrates erroneous output for different input data. The absolute error is not only spread across the input data range but also higher in magnitude. The error rate is also higher compared to the proposed *BwF-eVar* modified CGP approach. The error rates for both methods are indicated next to the absolute error graphs. For 7-bit and 8-bit activation functions, *eVar* enabled the circuit design to evolve faster by at least 105 times fewer generations, and *BwF* produced output profiles close to the required profile compared to SL. *BwF* and *eVar* enabled CGP-evolved circuits to closely adhere to the nonlinear profile of the activation functions studied in this work. Overall, *BwF* and *eVar* are two essential tools for setting up CGP to evolve complex nonlinear functions with significantly less computational effort. The designs are freely available in [37] for further use by the research and design community.

4.6. Dataformat Impact on Evolutionary Run. The choice of data format involves a tradeoff between precision and range for a given fixed bit-width, which affects the size of the CGP-derived circuits. When using a fixed-point format, a higher range implies a lower bit space for representing the fractional part, resulting in a loss of precision. As a result, the continuous nonlinear function is approximated more heavily, and a larger number of input values are mapped to the same output value. Conversely, more precision in the fixed-point format reduces the number of bits available for representing integer data, thereby limiting the range. However, this also means that fewer input values are mapped to the same output, and the circuit is expected to produce a higher number of unique output values.

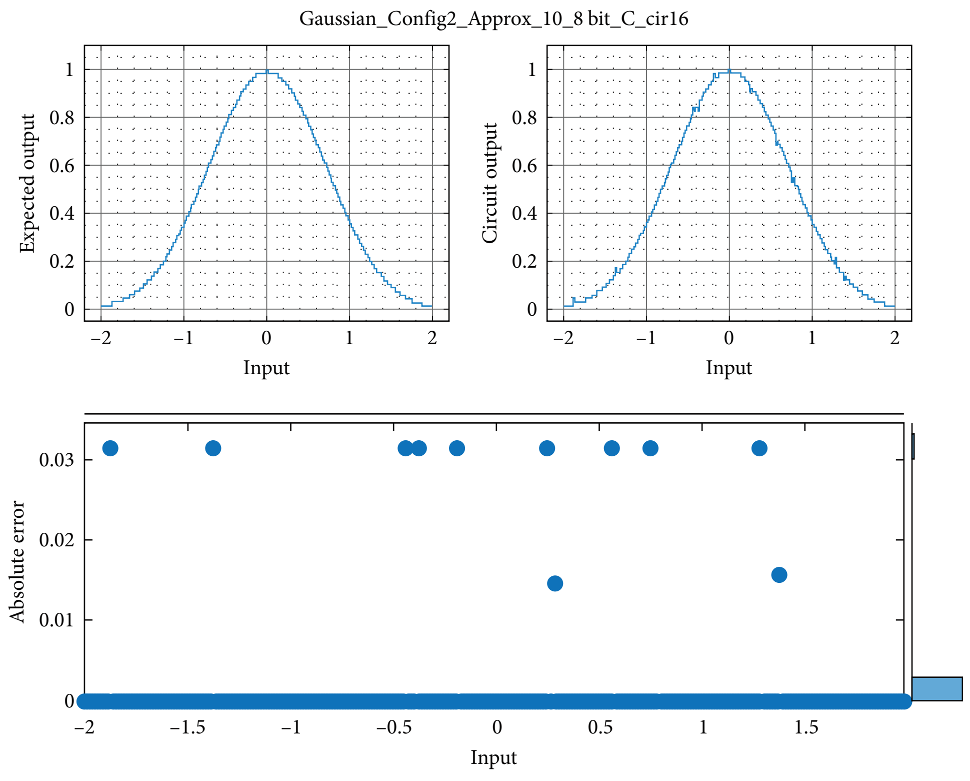
Table 3 shows four data formats chosen to benchmark CGP, with Config-2 offering the highest precision and Config-1 offering the highest range. Config-3 and Config-4 are designed to allocate half the available bits for the fractional part, with Config-3 allocating one extra bit for the fractional part.

The fully-fit circuit complexity in CGP runs depends on the input and output bit-widths. An increase in the bit-width of the input data-format results in a doubling of the truth-table size, requiring additional runs to achieve full-fitness. Similarly, an additional output bit widens the truth-table by a column, necessitating extra evolutionary steps for achieving full-fitness for the additional output bit. The impact of the data format choice on the evolutionary run is apparent from Figure 13 and Table 4. Higher precision configurations necessitate a significantly higher number of generations and nodes to achieve a zero-error circuit. For a complicated circuit like Gaussian, selecting one extra bit for the fractional part nearly doubles the genotype size. The number of generations needed to achieve convergence to full fitness also increases by a factor of 10. As shown in Figure 13, the convergence profile exhibits substantial variation before reaching 1% on full-fitness (100 on the y-axis). Among the 20 runs, the fitness convergence trend suggests that approximately correct circuits (circuits that reach 1% on full-fitness) evolve faster, requiring 1/100 of the generations necessary for full-fitness. Due to the manuscript's size restriction, only a small subset of the experiments' results are presented in upcoming figures and tables, with comprehensive results available at [37].

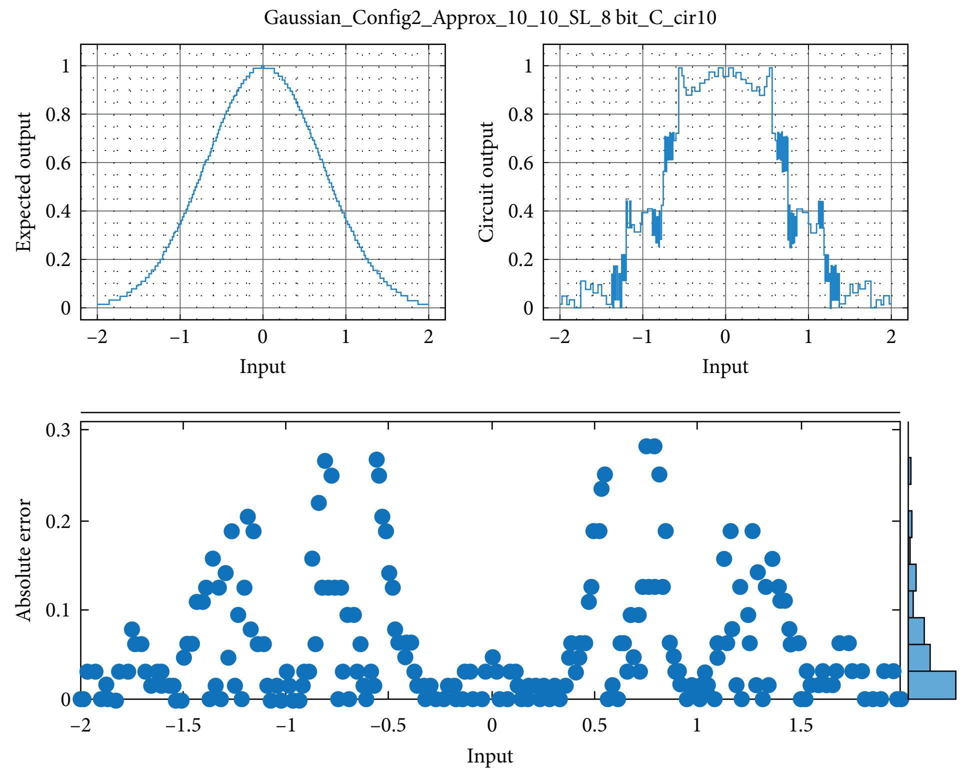
4.7. Impact on Synthesis. The CGP evolved designs from both basic gates representation and custom gates were synthesized using the Cadence Genus synthesis tool. The evolved netlists were mapped to four different standard cell libraries: Cadence gpdk45 (480 cells), Skywater 130 nm (386 cells), UMC 65 nm (1077 cells), and nanGate OCL 15 nm (76 cells). Figure 14 shows the Gaussian activation function realized in four different standard cell libraries across three different synthesis topologies: basic, hierarchical, and flattened. All the cells in the evolved netlist and synthesized designs were categorized into eight groups of incorporated gates. The majority of cells in the basic gates synthesized design are of OR, AND, and XOR type, while the hierarchical design utilizes around 53% cells from AOI, OAI, MUX, and the remaining 47% cells are basic gates AND, OR, XOR, XNOR. Post flattening, the netlist shows a higher percentage of AOI, OAI, and MUX. Figure 15 shows the difference in power, delay, and area of the design postsynthesis. Config-2 utilizes the highest resources, while Config-1 circuits concede the lowest resources. This is expected as the number of nodes in the evolved design is lower in the reduced precision configurations, as stated in Table 4. On flattening, the synthesized design improves the resources slightly but is not as significant as adopting basic gates in the CGP run.

The richness of the library (number of available cells) used for synthesis directly affects the type of cells in synthesized netlists. Nangate library, with just 76 cells, has the lowest percentage of AOI, OAI, and MUX, while the UMC library, with 1,077 cells, has around 52% of AOI, OAI, and MUX. As an inverter is not included in the function set, netlists obtained from CGP runs do not use inverters. CGP netlists have the lowest average number of cells and a uniform distribution in the type of cell, while the netlists postsynthesis introduces inverters and does not have even distribution of type of cells. On inspecting the timing reports, it was learned that the synthesis tool picks the cells to improve the timing. The distribution of type of cells shows no significant difference between hierarchical and flattened netlist synthesis.

4.8. Nodes Representation Impact on Evolutionary Algorithm. The impact of the choice of CGP node-gates is evident from Tables 4 and 5 and Figure 16. Choosing CGP nodes from the standard cells greatly reduces the size of the genotype (number of nodes) and the number of computations in each generation. Besides standard cells, selected CGP nodes show faster convergence to achieve full fitness. Using standard cells as nodes reduces computations required per run, thereby



(a)



(b)

Figure 12: Circuit and expected output 8-bit Gaussian CGP run example: (a) BwF helps to obtain circuits that closely resemble nonlinear functions; (b) SL only considers Hamming distance without assigning weightage to large errors.

TABLE 3: Four data format configurations used for inputs in the six activation functions represented as (integer part, fractional part) bits in a n-bit number.

Function	Config-1	Config-2	Config-3	Config-4
Sigmoid, ReLU, GeLU, Softplus	$(n-1, 0)$	$(3, n-4)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor - 1)$
Tanh	$(n-1, 0)$	$(2, n-3)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor - 1)$
Gaussian	$(n-1, 0)$	$(1, n-2)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor)$	$(n - \lfloor n/2 \rfloor - 1, \lfloor n/2 \rfloor - 1)$

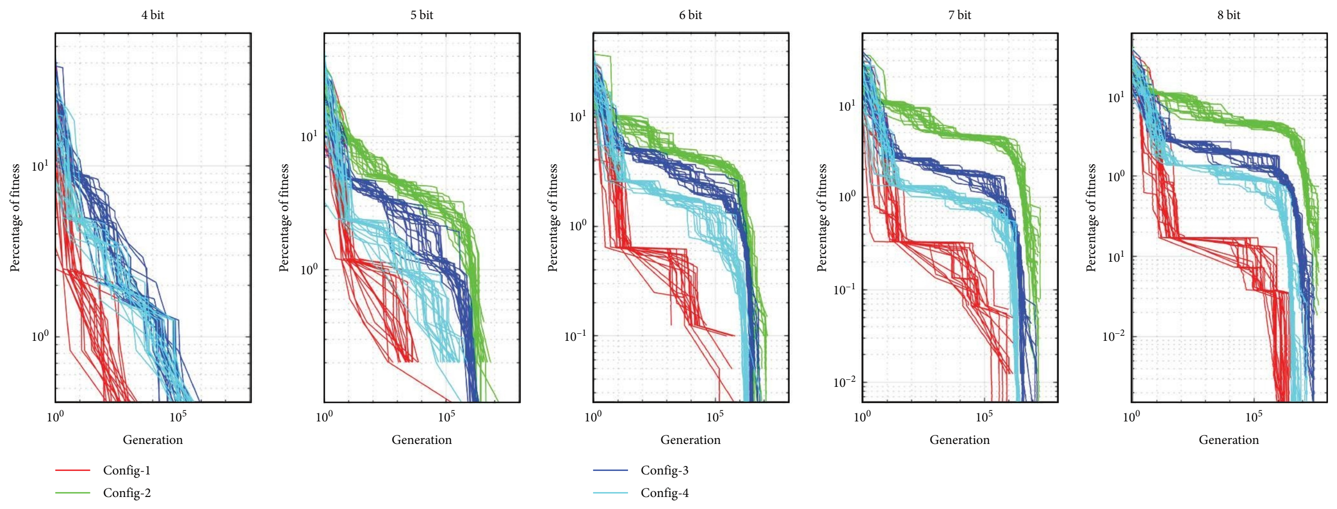


FIGURE 13: The percentage of fitness (zero being full-fitness) achieved versus generation for 20 independent runs each, targeted for 8-bit Gaussian function over four data formatted configurations (in log scale).

TABLE 4: Average number of generations (gens) and nodes required to achieve full fitness for 8-bit activation function synthesized circuits for all configurations.

Function	CGP nodes	Config-1 (gens, nodes)	Config-2 (gens, nodes)	Config-3 (gens, nodes)	Config-4 (gens, nodes)
Gaussian	Basic gates	9.33E + 06, 19	6.05E + 07, 144	2.96E + 07, 91	4.80E + 06, 44
	Std cells	2.94E + 06, 9	4.63E + 07, 105	2.42E + 07, 55	9.75E + 06, 33
GeLU	Basic gates	4.58E + 06, 23	1.76E + 07, 54	1.76E + 07, 54	2.23E + 07, 49
	Std cells	9.81E + 05, 17	7.86E + 06, 36	7.86E + 06, 36	4.63E + 06, 32
Sigmoid	Basic gates	5.57E + 06, 26	3.48E + 07, 98	3.48E + 07, 98	3.72E + 07, 80
	Std cells	6.18E + 06, 17	3.31E + 07, 61	2.30E + 07, 62	2.03E + 07, 46
Softplus	Basic gates	8.84E + 04, 9	3.85E + 07, 90	3.85E + 07, 90	2.12E + 07, 51
	Std cells	2.74E + 05, 8	1.60E + 07, 49	1.60E + 07, 49	3.97E + 06, 31
tanh	Basic gates	5.21E + 06, 22	5.92E + 07, 131	4.49E + 07, 104	1.98E + 07, 66
	Std cells	4.65E + 06, 11	3.46E + 07, 82	2.21E + 07, 61	9.55E + 06, 37

enhancing the scalability to realize larger bit-widths and higher precision data formatted designs. While standard cell nodes improve the speed of convergence, it does not necessarily produce efficient designs postsynthesis. Figures 14 and 15 show the disadvantage in terms of power, delay, and area. This result is due to the limitation of optimizations that can be performed by the synthesis tool, as circuits implemented with basic gates contain largely active nodes that are driving other nodes, while standard cell representation may contain a large number of redundant gates.

5. Comparison with Other Hardware Implementations

Generally, hardware implementations of activation functions in accelerators follow clocked-sequential designs, utilizing register files, cache, or RAM memories. Traditional LUT-based approaches demand substantial memory resources, typically on the order of 2^N words, each storing N-bit values [51, 52]. Other methods, such as series expansions and the CORDIC algorithm, involve hardware-intensive components

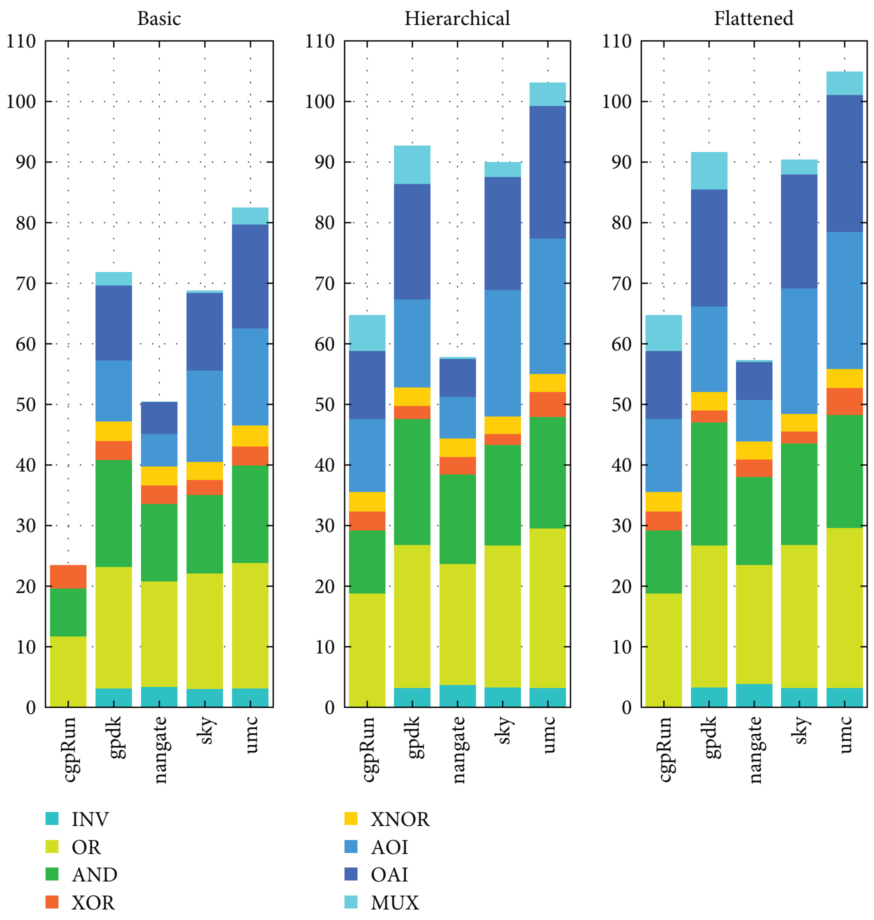


FIGURE 14: Average distribution of the kind of cells in Gaussian 8 bit circuits.

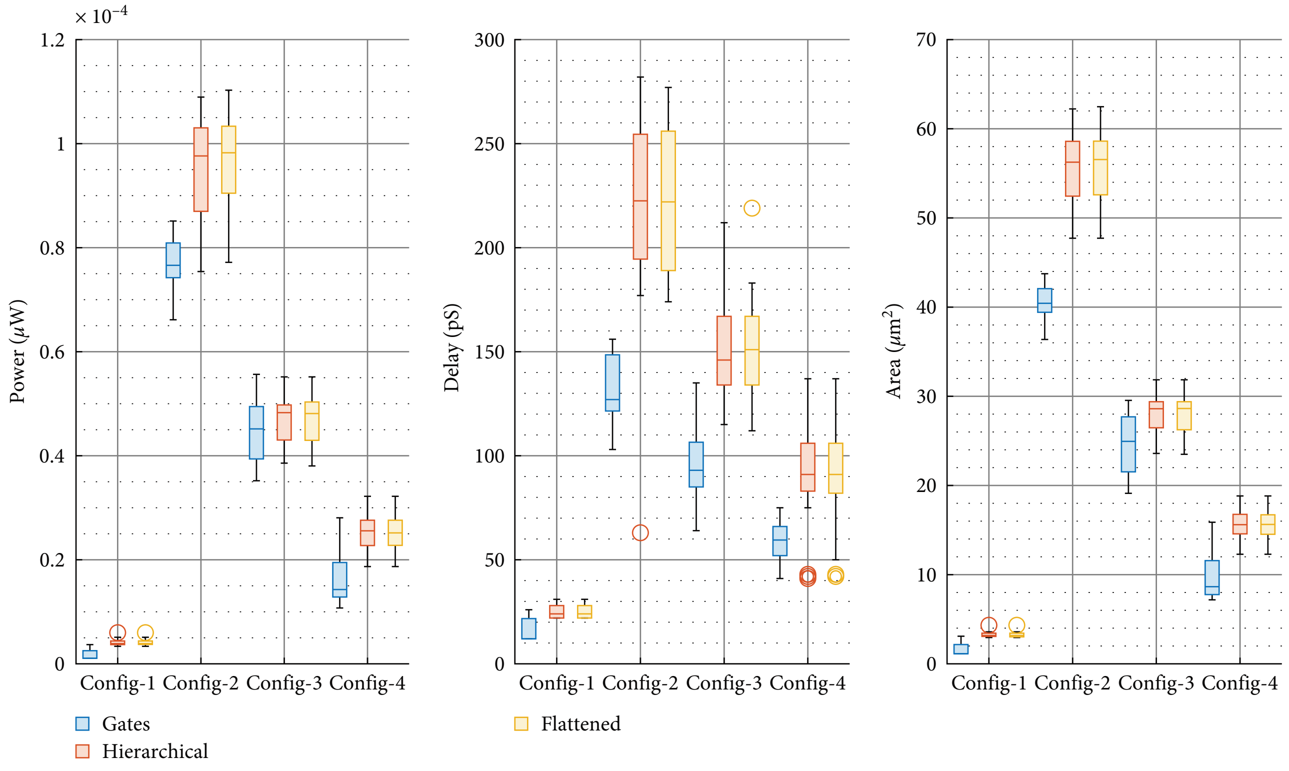


FIGURE 15: Hardware resources using basic, flattened, and hierarchical netlist in various data format configurations in Gaussian 8 bit circuits.

TABLE 5: Average number of nodes for CGP synthesized circuits under Config2 format, which is required to achieve full fitness.

Function	CGP nodes	4 bit	5 bit	6 bit	7 bit	8 bit
Gaussian	Basic gates	10	20	39	80	144
	Std cells	7	14	28	56	105
GeLU	Basic gates	10	14	18	31	54
	Std cells	7	8	12	22	36
Sigmoid	Basic gates	7	14	27	47	98
	Std cells	4	8	15	28	61
Softplus	Basic gates	5	8	23	47	90
	Std cells	4	6	14	28	49
tanh	Basic gates	6	15	35	64	131
	Std cells	4	10	20	42	82

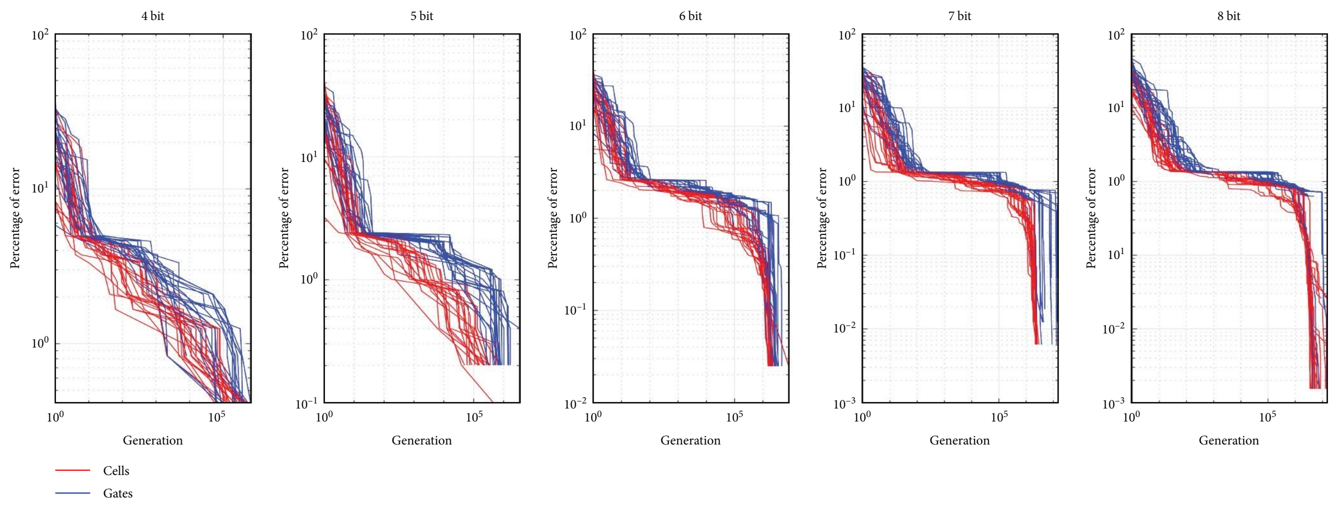


FIGURE 16: The percentage of fitness (zero being full-fitness) achieved versus generation for the 20 independent CGP runs of Gaussian Config-4 circuits synthesized using standard cells and basic gates.

like multipliers and adders. While it may be impractical to directly compare the synthesis results of unconstrained combinatorial designs in this study with timing-constrained designs in terms of PPA, we do discuss the resource advantages. The circuits obtained from CGP offer a wide range of hardware-accuracy tradeoffs. One of the largest circuit presented, 8-bit configurations under Config-2 data format, exhibit minimal error with the highest precision, comprising 150–160 basic gates. This gate count is significantly lower than any other implementation and requires fewer resources than an 8-bit multiplier. Additionally, pure combinational implementation offers the advantage of further optimization during synthesis, a capability generally unavailable when using standard memory blocks. Circuits from different configurations are likely to have fewer gates than those selected from Config-2.

The novel hardware implementations of activation functions were evaluated for hardware advantages when deployed in FPGA using the DNNWeaver accelerator [53] and BRAM in hls4ml [54]. The Xilinx CORDIC-IP method for Hyperbolic-Tangent implementation was also included in the comparison. Additionally, the PWL method with 16 segments [34] was compared with the proposed CGP evolved activation function designs. The FPGA utilization was assessed on Artix-7 FPGA fabric, and ASIC results were extracted using a 45 nm gpdk technology library. For the comparison, 8-bit Config2 circuits of these activation functions, featuring the maximum gates and generating the least error, were considered. Memory-table-based implementations utilized 256 bytes with an 8-bit bus RAM and a 2-way set associative DDR3 cache, both modeled in 45 nm technology. It is important to note that register-based memory introduced an additional clock cycle, while cache and RAM implementations required an overhead access time of 0.15–0.2 ns. Furthermore, PWL circuits with 16 segments were implemented for all the 8-bit activation functions.

The synthesis results of circuits generated through the CGP method are depicted in Figure 17, alongside other implementations. The 8-bit activation functions evolved by CGP demonstrate remarkably low LUT utilization when compared to alternative methods, including DNNweaver-LUT, hls4ml, CORDIC, and PWL, within the FPGA flow. Specifically, CGP designs require approximately (1/7) the number of LUTs as DNNweaver-LUT and hls4ml, which are an order of magnitude less than CORDIC and nearly 100 times fewer than PWL implementations. Furthermore,

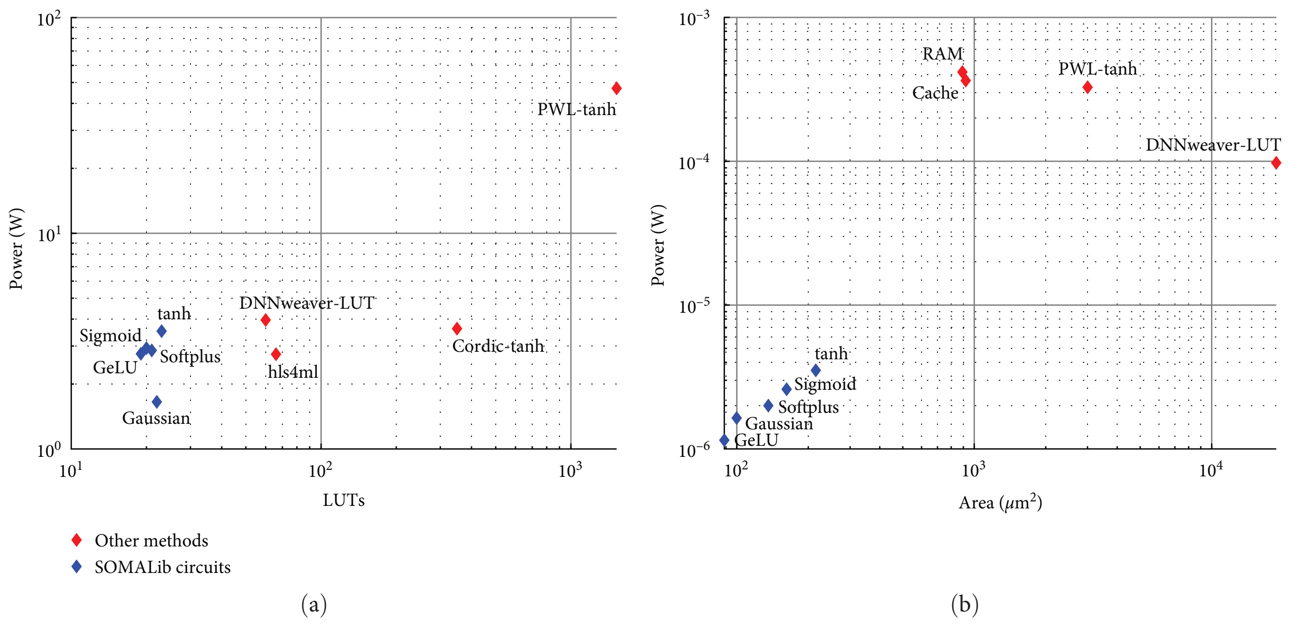


Figure 17: Comparison of the CGP circuits of Config2 8-bit activation functions with conventional implementations for the same data format in (a) FPGA flow and (b) ASIC flow. Please note that hls4ml utilizes an additional 16 BRAMs that are not captured here.

CGP implementations exhibit favorable power consumption, comparable to hls4ml, and superior to the other three methods. While hls4ml also has similar power requirements, it necessitates seven times as many LUTs. In the ASIC flow, CGP circuits offer substantial benefits in terms of footprint and power efficiency when compared to other implementations. RAM and Cache implementations consume approximately ten times more silicon space than the CGP-based activation function implementation. Meanwhile, PWL method implementations require nearly 10 times more silicon space, and DNNweaver-LUT takes up more than two decades' worth of silicon space compared to the proposed CGP method. The power consumption of CGP-based activation functions is exceptionally low when compared to other methods.

In summary, even the largest CGP-generated circuits, which have the highest gate count and produce minimal errors, offer substantial hardware advantages when compared to other implementations. Relaxing error performance requirements in error-tolerant designs enables approximate designs to provide additional hardware and performance benefits beyond existing implementations. CGP-evolved circuits offer a spectrum of hardware-error performance tradeoffs. Being purely combinational enables single-clock-cycle calculations for activations at each neural network layer. This results in savings in memory-access time and a reduction in dynamic power per computation.

The time complexity of the proposed method to evolve and realize the complex functions may be more than other methods such as CORDIC, PWL, DNNweaver-LUT, and other memory-based RAM and CACHE methods. However, postevolution and synthesis of circuits from the modified CGP method, these circuits are extremely power, performance, and footprint efficient. A direct comparison with other methods in terms of time complexity is difficult because of the heuristic nature of the algorithm. A majority of evaluations (9 out of 10 in the case of $(1+10)$ strategy) do not necessarily contribute to improvement every generation. Noncoding nodes in the solutions are also a major source of redundant computations in the algorithm, but they are crucial for the evolutionary process. Memory-based methods are primarily tool-based, full-custom derived CORDIC or PWL implementations for each function are designer-dependent.

Custom implementation using memory blocks (LUTs and PWL methods) is dependent on the technology node, and memory blocks available. For FPGA implementation, we had to optimize the physical implementation using BRAM blocks in Xilinx. The major overhead of this method involves the usage of a large number of LUTs, Registers, and Multiplexers for saving results in 8-bits. Here, the optimization is toward lower usage of memory blocks and other hardware resources to map the output and achieve results in low latency. However, the complexity and intricacies in realization are minimal, considering the 8-bit result is fed to the LUT by the designers. Similarly, the PWL method saves coefficients of every segment in the LUTs; thereby, the complexity in realizing the output is of lower order, but the hardware resources utilized are fairly high, whereas the proposed technique has the vast combinatorial design space to evolve a group of gates that maps the output. The complexity of the proposed technique is in realizing the output and later minimizing the number of cells employed. For realizing the evolved gate-level design in the FPGA system, these sets of cells are mapped to the hardware resources in-built into the system.

Implementing the activation function as a shared system-level subsystem in an SoC involves designing a memory controller for the custom SRAM blocks. This is required for both LUT and PWL implementations. For data sizes that are not power-of-two result in unused parts of the memory block that occupy hardware space. Hence, the tradeoff in realizing the activation functions in LUTs or PWL methods is in designing additional memory controller blocks besides living with unused memory blocks. The proposed technique evolves to a purely combinatorial design and neither worries about any controller overhead design nor on unused memory blocks. The combinatorial design representing the activation functions is realized on SoC as a subsystem block by mapping to the defined standard cells.

The CGP synthesis requires high computing resources over other traditional methods, such as LUTs and PWL, where the output of the function is either directly stored or approximated to extract a linear profile. On the other hand, CGP implementation runs through the entire exhaustive design space search and attempts to find node-level design representing the function under consideration. Hence, time complexity to arrive at a design solution is always easy with other implementations; however, the proposed approach is demonstrated to offer a fitter circuit that is hardware and power-efficient than other existing implementations. For linear functions, one can always opt for other existing methods since CGP is likely to scan a large design space before arriving at a solution. However, for realizing AI subsystems, which are generally in the form of nonlinear functions, the proposed CGP synthesis approach is valuable.

The activation functions studied are a few of the most complex circuits to custom-implement. The main aim was to target quantized implementations (sub-8 bit) of neural networks, as they are crucial for low-power edge implementations. The potential limitations will be with the runtime of the CGP algorithm to obtain circuits with acceptable error-hardware performance. Our implementation is purely CPU-based C code utilizing multiple threads to speed up runtime; GPU or FPGA implementation of the algorithm itself will help address the long runtimes.

6. Conclusions

CGP, an evolutionary design methodology, was implemented to synthesize hierarchical designs mapped to 20 common standard cells of 4-input gates. CGP allowed to explore design-space search for realizing complex functions. The paper employs CGP to realize and analyze popular nonlinear activation and power functions that are hard to realize for different configurations of data format and bit-widths. Additionally, CGP runs using the *BwF* fitness scheme and *eVAR* mutation rate offer quicker evolution and fitter circuits. The evolved netlist from the CGP run was further flattened to characterize the hardware metrics through the ASIC flow. CGP evolved netlist for activation functions mapped to basic gates were also generated for comparison purposes. The synthesis run through the selected standard cells saves run-time and computational effort compared to the synthesis run using classical basic gate cells. The CGP-derived synthesis method reported 3× less synthesis time for realizing the complex functions at the hierarchical level compared to using basic gates. Further flattening of the hierarchical design to their constituent basic gates showed no significant improvement in hardware metrics. The scope of the CGP-generated circuits using standard cells is easily extendable to the regular ASIC synthesis flow in two ways: *Path-A*, as referred previously, waits for the CGP synthesis run to complete and hence suffers from high turnaround time, but this flow is expected to produce hardware-efficient design solutions. Alternately, *Path-B* produces the design solution at a shorter turnaround time but is likely to present functionally suboptimal designs.

Data Availability

All the results of this work are made freely available for further benchmarking and easy usage to the researchers and designers community at https://sites.google.com/view/evolutionarysynthesis/home.

Disclosure

This manuscript is a comprehensive work extending on the basic exploration of Binary Weighted Fitness (BwF) and exponentially varying mutation rate (eVar), which was presented at the 14th International Conference on Evolutionary Computation Theory and Applications Conference 2022 under the title of Improving Digital Circuit Synthesis of Complex Functions using Binary Weighted Fitness and Variable Mutation Rate in Cartesian Genetic Programming, and is published in the proceedings of 14th International Joint Conference on Computational Intelligence, 2022.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

All the work was sponsored by the employer—International Institute of Information Technology Bangalore.

References

[1] B. C. Schafer, D. Aledo, and F. Moreno, "Application specific behavioral synthesis design space exploration: artificial neural networks. A case study," in *2017 Euromicro Conference on Digital System Design (DSD)*, pp. 129–136, IEEE, 2017.

[2] M. Anwar, S. Saha, M. M. Ziegler, and L. Reddy, "Early scenario pruning for efficient design space exploration in physical synthesis," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pp. 116–121, IEEE, Kolkata, India, 2016.

[3] T. Saito, H. Sugimoto, M. Yamazaki, and N. Kawato, "A rule-based logic circuit synthesis system for CMOS gate arrays," in *Proceedings of the 23rd ACM/IEEE Design Automation Conference (DAC '86)*, pp. 594–600, IEEE Press, 1986.

[4] J. Jung, I. H.-R. Jiang, J. Chen et al., "DATC RDF: an academic flow from logic synthesis to detailed routing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–4, IEEE Press, 2018.

[5] Y. Decoudu, K. Morin-Allory, and L. Fesquet, "A high-level design flow for locally body biased asynchronous circuits," in *2021 IFIP/IEEE 29th International Conference on Very Large*

Scale Integration (VLSI-SoC), pp. 1–6, IEEE, Singapore, Singapore, 2021.

[6] C. M. d. Oliveira Conceição and R. A. d. L. Reis, "Netlist optimization by gate merging," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 236-237, IEEE, Cuzco, Peru, 2019.

[7] S. Venkataramani, V. J. Kozhikkottu, A. Sabne, K. Roy, and A. Raghunathan, "Logic synthesis of approximate circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2503–2515, 2020.

[8] M. Awais and M. Platzner, "MCTS-based synthesis towards efficient approximate accelerators," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 384–389, IEEE, Tampa, FL, USA, 2021.

[9] C. Zhengbo, T. Lei, and C. Zuoning, "Research and design of activation function hardware implementation methods," *Journal of Physics: Conference Series*, vol. 1684, no. 1, Article ID 012111, 2020.

[10] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," in *2009 IEEE International Symposium on Circuits and Systems*, pp. 2117–2120, IEEE, Taipei, Taiwan, 2009.

[11] Z. Hajduk, "Hardware implementation of hyperbolic tangent and sigmoid activation functions," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 66, pp. 563–577, 2018.

[12] K. Basterretxea, J. M. Tarela, and I. del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *IEE Proceedings - Circuits, Devices and Systems*, vol. 151, pp. 18–24, 2004.

[13] H. Chen, L. Jiang, H. Yang et al., "An efficient hardware architecture with adjustable precision and extensible range to implement sigmoid and tanh functions," *Electronics*, vol. 9, no. 10, Article ID 1739, 2020.

[14] N. A. Avdeev and P. N. Bibilo, "Logical optimization efficiency in the synthesis of combinational circuits," *Russian Microelectronics*, vol. 44, no. 5, pp. 338–354, 2015.

[15] C.-Y. Huang, C.-A. R. Wu, T.-Y. Lee, C.-J. J. Hsu, and K.-Y. Khoo, "2019 CAD contest: logic regression on high dimensional boolean space," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, IEEE, 2019.

[16] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: a new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.

[17] H. Norouzi and M. E. Salehi, "Evolutionary design for energy-efficient approximate digital circuits," *Microprocessors and Microsystems*, vol. 57, pp. 52–64, 2018.

[18] P. Fišer, J. Schmidt, Z. Vašíček, and L. Sekanina, "On logic synthesis of conventionally hard to synthesize circuits using genetic programming," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pp. 346–351, IEEE, 2010.

[19] A. Berndt, I. S. Campos, B. Lima et al., "Accuracy and size trade-off of a cartesian genetic programming flow for logic optimization," in *34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–6, IEEE, 2021.

[20] J. Kocnova and Z. Vasicek, "EA-based resynthesis: an efficient tool for optimization of digital circuits," *Genetic Programming and Evolvable Machines*, vol. 21, no. 3, pp. 287–319, 2020.

[21] R. K. Brayton and A. Mishchenko, "Abc: an academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds., pp. 24–40, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[22] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Dag-aware aig rewriting a fresh look at combinational logic synthesis"," in *Proceedings of the 43rd Annual Design Automation Conference*, pp. 532–535, Association for Computing Machinery, New York, NY, USA, 2006.

[23] L. Sekanina, O. Ptak, and Z. Vasicek, "Cartesian genetic programming as local optimizer of logic networks," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2901–2908, IEEE, 2014.

[24] A. Stempkovsky, D. Telpukhov, and R. Solovyev, "Synthesis of approximate combinational circuits based on logic regression approach," in *2020 IEEE East-West Design Test Symposium (EWDTS)*, pp. 1–5, IEEE, 2020.

[25] V. Mrazek, Z. Vasicek, and R. Hrbacek, "Role of circuit representation in evolutionary design of energy-efficient approximate circuits," *IET Computers & Digital Techniques*, vol. 12, no. 4, pp. 139–149, 2018.

[26] J. F. Miller, *Cartesian Genetic Programming. Natural Computing Series*, pp. 17–34, Springer, Berlin, Heidelberg, 2011.

[27] L. Sekanina and Z. Vasicek, "Approximate circuit design by means of evolvable hardware," in *2013 IEEE International Conference on Evolvable Systems (ICES)*, pp. 21–28, 2013.

[28] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[29] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2016.

[30] V. Tiwari and N. Khare, "Hardware implementation of neural network with Sigmoidal activation functions using CORDIC," *Microprocessors and Microsystems*, vol. 39, no. 6, pp. 373–381, 2015.

[31] H. Chen, L. Jiang, Y. Luo et al., "A cordic-based architecture with adjustable precision and flexible scalability to implement sigmoid and tanh functions," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[32] Y. Chang, P. Jokic, S. Emery, and L. Benini, "An ultra-low-power serial implementation for sigmoid and tanh using cordic algorithm," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-2, IEEE, 2023.

[33] F. Lyu, Z. Mao, J. Zhang, Y. Wang, and Y. Luo, "PWL-based architecture for the logarithmic computation of floating-point numbers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 7, pp. 1470–1474, 2021.

[34] H. Dong, M. Wang, Y. Luo et al., "Plac: piecewise linear approximation computation for all nonlinear unary functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 2014–2027, 2020.

[35] H. Sun, Y. Luo, Y. Ha et al., "A universal method of linear approximation with controllable error for the efficient implementation of transcendental functions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 177–188, 2020.

[36] H. C. Prashanth and M. Rao, "Somalib: library of exact and approximate activation functions for hardware-efficient neural network accelerators"," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pp. 746–753, IEEE, 2022.

[37] H. C. Prashanth and Results website, "Accelerated and highly correlated ASIC synthesis of AI hardware subsystems using CGP," 2023.

[38] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *10th Annual Conference Companion on Genetic and Evolutionary Computation*, pp. 2701–2726, Association for Computing Machinery, New York, NY, USA, 2008.

[39] H. C. Prashanth and M. Rao, "Evolutionary standard cell synthesis of unconventional designs," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, pp. 189–192, Association for Computing Machinery, New York, NY, USA, 2022.

[40] Z. Vasicek, "Cartesian gp in optimization of combinational circuits with hundreds of inputs and thousands of gates," in *Genetic Programming*, P. Machado, M. I. Heywood, and J. McDermott, et al., Eds., pp. 139–150, Springer International Publishing, Cham, 2015.

[41] Z. Vasicek and L. Sekanina, "Search-based synthesis of approximate circuits implemented into fpgas," in *26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, IEEE, 2016.

[42] D. Hodan, V. Mrazek, and Z. Vasicek, "Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design"," in *2020 Genetic and Evolutionary Computation Conference*, pp. 940–948, Association for Computing Machinery, New York, NY, USA, 2020.

[43] Y. Miyasaka, X. Zhang, M. Yu, Q. Yi, and M. Fujita, "Logic synthesis for generalization and learning addition," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1032–1037, IEEE, 2021.

[44] J. F. Miller, "Cartesian genetic programming: its status and future," *Genetic Programming and Evolvable Machines*, vol. 21, no. 1-2, pp. 129–168, 2020.

[45] A. Manazir and K. Raza, "Recent developments in cartesian genetic programming and its variants," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–29, 2019.

[46] Z. Vasicek and L. Sekanina, "Hardware accelerator of cartesian genetic programming with multiple fitness units," *Computing and Informatics*, vol. 29, no. 6+, pp. 1359–1371, 2012.

[47] M. Šikulová and L. Sekanina, "Coevolution in Cartesian Genetic Programming," in *Genetic Programming*, A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, Eds., vol. 7244 of *Lecture Notes in Computer Science*, pp. 182–193, Springer, 2012.

[48] H. C. Prashanth and M. Rao, "Improving digital circuit synthesis of complex functions using binary weighted fitness and variable mutation rate in cartesian genetic programming," in *Proceedings of the 14th International Joint Conference on Computational Intelligence, IJCCI 2022*, T. Bäck, B. van Stein, and C. Wagner, et al., Eds., pp. 112–120, Scitepress, Valletta, Malta, October, 2022.

[49] D. Thierens, "Adaptive mutation rate control schemes in genetic algorithms," in *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 980–985, IEEE, 2002.

[50] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in Cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[51] D. Larkin, A. Kinane, V. Muresan, and N. O'Connor, "An efficient hardware architecture for a neural network activation function generator," in *Advances in Neural Networks - ISNN 2006*, J. Wang, Z. Yi, J. M. Zurada, B. L. Lu, and H. Yin, Eds., vol. 3973 of *Lecture Notes in Computer Science*, pp. 1319–1327, 3973, Berlin, Heidelberg, 2006.

[52] T. Nguyen, T. K. Luong, L. D. Han, and V.-P. Hoang, "An efficient hardware implementation of activation functions using stochastic computing for deep neural networks," in *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 233–236, IEEE, 2018.

[53] H. Sharma, J. Park, D. Mahajan et al., "From high-level deep neural models to fpgas," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, Taipei, Taiwan, 2016.

[54] J. Duarte, S. Han, P. Harris et al., "Fast inference of deep neural networks for real-time particle physics applications," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*, Association for Computing Machinery, New York, NY, USA, 305, 2019.