Expert Systems With Applications 296 (2026) 129083

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

# Automatic residue-to-binary converter circuit generation and simplification using hybrid cartesian genetic programming and simulated annealing

Amir Dadashzadeh [a], Mehdi Hosseizadeh [b,c,d,*], Amir Sabbagh Molahosseini [e], Amir Sahafi [f]

[a] Department of Computer Engineering, SR.C., Islamic Azad University, Tehran 1477893855, Iran
[b] Institute of Research and Development, Duy Tan University, Da Nang, Vietnam
[c] School of Engineering & Technology, Duy Tan University, Da Nang, Vietnam
[d] Jadara Research Center, Jadara University, Irbid 21110, Jordan
[e] School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast BT9 5AF, UK
[f] Department of Computer Engineering, ST.C., Islamic Azad University, Tehran, Iran

ARTICLE INFO

*Keywords:*
Residue Number Systems (RNS)
Reverse Converter
Cartesian Genetic Programming (CGP)
Eevolutionary Circuit Design
Simplification of RNS Converters

ABSTRACT

Residue Number Systems (RNS) provide notable advantages in digital signal processing and error detection due to their inherent parallelism and fault tolerance. However, designing efficient reverse converters, particularly those based on the Chinese Remainder Theorem, remains a complex and labour-intensive. This paper presents an innovative automated methodology for developing simplified residue-to-binary converters by integrating Cartesian Genetic Programming (CGP) with Simulated Annealing (SA). The proposed approach employs a two-phase optimization framework. In the first phase, a hybrid CGP-SA algorithm generates module-level architectures by evolving computational structures using arithmetic and bitwise operations. Then, SA is then applied to optimize operator parameters, ensuring functional accuracy and enhanced efficiency. In the second phase, the high-level architecture is refined into an optimized gate-level design using adaptive evolutionary strategies to minimize latency, area, and power consumption. Simulation results demonstrate that the proposed framework consistently outperforms conventional handcrafted methods, offering improved computational efficiency, reduced hardware complexity, and greater design flexibility. This advancement provides a promising solution for practical, high-performance RNS reverse converter implementations.

## 1. Introduction

Residue Number Systems (RNSs) provide an efficient number representation system that enables parallel computation and high-speed arithmetic while offering error resilience (Saheed et al., 2024; Shenets, 2024). By expressing numbers as sets of residues comprising pairwise co-prime moduli, RNS has gained widespread use in digital signal processing, cryptography, deep learning, and bioinformatics (Mojahed et al., 2021a). Despite these advantages, a major challenge in RNS implementation is converting between residue and binary representations (Balaji & Padmaja, 2024). In particular, residue-to-binary conversion (RNS reverse conversion) is crucial for interfacing RNS-based computing units with binary processors but remains computationally demanding and structurally complex. This complexity arises from the dependence on the Chinese Remainder Theorem (CRT), Mixed-Radix Conversion (MRC), and their modern adaptations like New-CRTs, which require modular inverses, weighted summations, and sequential arithmetic operations such as multiplication. These operations impose significant computational overhead, making hardware-efficient implementation difficult (Boyvalenkov et al., 2023; Li et al., 2024; Yu et al., 2023).

Traditional reverse conversion methods have relied on lookup tables or multiplication-based designs (Ahmadifar & Torabi, 2024). Lookup table-based methods (ROM-based) precompute conversion values, allowing fast translation from RNS to binary. However, as moduli set sizes grow, memory requirements increase exponentially, making this approach impractical for large-scale applications. Multiplication-based designs, which perform weighted summations and modular reductions, provide an alternative but depend heavily on hardware multipliers. While these methods offer speed improvements, they consume

* Corresponding author.
*E-mail addresses:* amir.dadashzadeh@iau.ac.ir (A. Dadashzadeh), mehdihosseinzadeh@duytan.edu.vn (M. Hosseizadeh), a.sabbaghmolahosseini@qub.ac.uk (A.S. Molahosseini), sahafi@iau.ac.ir (A. Sahafi).

https://doi.org/10.1016/j.eswa.2025.129083
Received 30 April 2025; Received in revised form 2 July 2025; Accepted 18 July 2025
Available online 23 July 2025
0957-4174/© 2025 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

substantial power and silicon area, restricting their use in low-power and resource-limited environments like embedded systems and energy-efficient processors. Researchers have explored memory-less and multiplier-less reverse converters to overcome these limitations, especially for moduli sets based on powers of two (Jaberipur & Ahmadifar, 2014). These designs eliminate costly memory storage and multiplication operations by taking advantage of arithmetic properties that simplify modular computations. Memory-less converters replace lookup tables with direct computational formulas, reducing memory usage while maintaining conversion efficiency. Also, multiplier-less designs avoid expensive multiplications by using bitwise operations, modular addition, and shift-based arithmetic techniques. These advancements contribute to lower power consumption and improved computational efficiency.

Nevertheless, these reverse conversion designs still rely on manual development, requiring engineers to manually devise conversion algorithms and circuit architectures based on predefined mathematical models (Turan et al., 2020). This manual design process is labor-intensive, time-consuming, and computationally inefficient, as it involves extensive adjustments to achieve an optimal balance between speed, power consumption, and hardware complexity (Prediger et al., 2023). Moreover, the vast and complex design space of reverse converters presents additional challenges, as manually developed architectures may fail to explore all possible optimizations (Babkin & Ulitin, 2024; Fernandes et al., 2024). The constraints of manual design become even more evident as moduli set sizes increase, requiring trade-offs between latency, area, and power efficiency. Therefore, in many cases, suboptimal solutions emerge due to the limited ability to exhaustively analyze all design variations.

As a result, there is a growing demand for automated design methodologies that can systematically explore the architectural space and generate optimized, high-performance, and resource-efficient converter designs. An ideal automated approach should intelligently navigate various design trade-offs, minimize manual intervention, and provide solutions that outperform traditional handcrafted designs in terms of hardware efficiency, computational speed, and energy consumption. The increasing complexity of modern computing systems necessitates such automation, making it a crucial step toward achieving more efficient and adaptable residue-to-binary converters.

Recent advances in evolutionary algorithms and genetic programming (GP) have shown promise in automating design processes of digital circuits (Miller & Turner, 2015; Miller, 2020). By utilizing evolutionary principles, these methods can explore vast design spaces, optimize multiple criteria such as area, power, delay, and fault tolerance, and discover innovative circuit architectures that might not be apparent through traditional deterministic approaches. This flexibility allows GP to generate novel solutions by effectively navigating trade-offs and harnessing stochastic search methods, which can lead to optimized and unconventional designs. As technology advances and digital circuits become increasingly complex, the role of GP in circuit design is likely to grow, driving further innovation and efficiency in the field. The inherent ability of GP to adapt and evolve designs makes it a powerful tool for handling the intricate requirements of modern digital systems (Husa & Sekanina, 2024; Prashanth & Rao, 2024). However, despite its proven potential, the application of GP to the specific challenge of designing RNS converters remains largely unexplored. RNS converter design involves unique challenges that may benefit from GP's adaptive search capabilities. Investigating this application could not only enhance the efficiency and robustness of RNS converter designs but also extend the frontiers of GP in tackling specialized and complex digital circuit design problems.

In this regard, this paper presents an evolutionary-based automatic design framework for reverse converters, integrating Cartesian Genetic Programming (CGP) and Simulated Annealing (SA). The proposed methodology follows a two-phase optimization process to systematically refine reverse converter architectures at both the module-level and gate-level. During the first phase, the reverse converter is constructed using a combination of arithmetic and bitwise operations, in which, CGP is utilized to explore a wide range of potential designs and searches for optimal module-level configurations. The optimization process is governed by a cost function that assesses correctness, latency, and resource consumption. Since certain operations, such as bitwise shifts, require parameter tuning, SA is incorporated as a local search technique to refine these parameters. Through iterative evolution, the algorithm reduces errors, shortens latency, and enhances resource efficiency, ultimately producing an optimized high-level representation of the reverse converter. In the second phase, first, the high-level design is converted into a gate-level implementation, where each operation is mapped to fundamental logic gates such as AND, OR, XOR, and NOT, along with higher-order components like multiplexers, sum, and carry gates. Next, another CGP optimization process is applied to further refine the correctness, latency, and resource efficiency of the gate-level circuit. This phase ensures that the final circuit is optimized, hardware-efficient, and suitable for practical implementation. By using hybrid evolutionary optimization techniques, the proposed approach enables automated and systematic design space exploration, producing optimized reverse converters tailored to the performance and resource constraints of modern computing systems. Specifically, the key contributions of this paper are:

- Introducing a fully automated evolutionary design methodology for RNS reverse converters that eliminates manual intervention, systematically exploring and optimizing converter architectures for enhanced performance and efficiency. To the best of our knowledge, it is the first evolutionary-based methodology that integrates CGP and SA for the automated design of RNS reverse converters.
- Proposing a two-phase optimization framework utilizing a module-level architecture generation phase and a gate-level simplification phase. A structured approach is devised where, in the first phase, module-level architecture is generated using high-level arithmetic and bitwise operations as fundamental building blocks. This is followed by gate-level refinement phase to achieve an optimal balance between hardware efficiency, resource utilization, and overall performance.
- In the first phase, a functionally efficient module-level reverse converter is designed using the CGP-SA framework, where CGP explores high-level structural configurations while SA fine-tunes critical parameters, enabling the construction of a structurally optimized reverse converter that ensures both accuracy and efficiency.
- In the second phase, gate-level circuit optimization is performed using CGP, where the gate-level representation of the reverse converter is refined by minimizing gate count, reducing latency, and improving power efficiency, ensuring that the final circuit is highly efficient and resource-optimized.
- We introduce a multi-objective cost function that balances correctness, latency, and resource usage in RNS reverse converters. Correctness is evaluated via normalized Hamming distance between obtained outputs and expected values, latency is estimated by the number of active columns in the CGP grid, and resource usage is calculated based on the sum of the active nodes' costs. This cost function guides evolutionary optimization toward accurate, high-performance, and hardware-efficient designs.
- Due to the large search space in the second phase, dynamic mutation rates and tournament sizes are incorporated into CGP to effectively balance exploration and exploitation, leading to a more optimized gate-level implementation.
- The proposed methodology surpasses traditional manual design approaches by exploring the design space more extensively and efficiently, leading to more optimized circuit architectures.

The rest of this paper is organized as follows: Section 2 reviews existing literature on RNS reverse converter design and GP for circuit synthesis, highlighting the contributions of the proposed approach.

Section 3 presents the preliminaries, covering the fundamental principles of CGP and RNS reverse conversion, establishing the theoretical foundation for the proposed methodology. Section 4 details the proposed two-phase optimization framework, explaining the module-level design using CGP-SA in Phase 1 and gate-level refinement using CGP in Phase 2. Section 5 provides simulation results, including hyper-parameter settings and a comparative analysis of CGP and SA algorithms. This section also evaluates the performance of the proposed design against conventional methods across prominent moduli sets. Finally, Section 6 concludes the paper by summarizing key findings and discussing potential directions for future research and further improvements. Additionally, detailed explanations of technical details of CGP and new CRTs, along with a case study on the {4,7,9} moduli set demonstrating the framework's ability to evolve functionally correct architectures, are provided in Appendices A, B, and C, respectively.

## 2. Literature review

This section initially provides a review of existing literature on RNS reverse converter design from 2010, focusing on various techniques employed to boost efficiency and reduce hardware complexity. It examines both algorithmic and hardware-based optimizations that have been introduced to enhance the performance of reverse converters across diverse moduli sets and applications. In addition, the section reviews the application of GP in automatic circuit synthesis, emphasizing its effectiveness in evolving optimized digital circuits without manual intervention. Finally, the unique contributions of our hybrid CGP-SA approach are presented. This innovative method automates and simplifies the design of residue-to-binary converters, offering improved efficiency and greater design flexibility compared to existing methodologies.

### 2.1. RNS reverse converter design

As mentioned above, RNS provides significant advantages in digital signal processing due to its inherent parallelism and fault tolerance. However, the design of efficient reverse converters remains a key challenge in RNS-based systems. Hence, researchers have explored various techniques to enhance speed, area efficiency, and power consumption, leading to notable advancements in the field.

One of the main efforts in this domain focused on optimizing reverse converters based on 3-moduli sets. In this regard, Molahosseini et al. (Molahosseini et al., 2008) introduced an efficient architecture for designing reverse converters based on a general 3-moduli set $\{2^{\alpha}, 2^{\beta}-1, 2^{\beta}+1\}$, significantly improving the speed and efficiency of reverse conversion. Their approach utilizes the properties of specific moduli sets to simplify the conversion process. By optimizing the residue computation, they were able to significantly enhance the speed and efficiency of reverse conversion. Building on this foundation, they proposed improvements for new 4-moduli sets ($\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ and $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$), incorporating New-CRT-based techniques (Molahosseini et al., 2009). These enhancements led to substantial reductions in computational complexity, making the designs more efficient in terms of both area and processing speed.

Addressing the reliance of RNS converters on ROMs and multipliers, Jaberipur et al. (Jaberipur & Ahmadifar, 2014) proposed a ROM-less reverse RNS converter for the moduli set $\{2^q \pm 1, 2^q \pm 3\}$. Their approach utilized a two-stage New CRT conversion scheme with conjugate grouping of moduli, effectively eliminating the need for ROMs and multipliers. The proposed architecture achieved notable improvements, including a 22 % reduction in delay, a 19 % decrease in area, and an 8 % drop in power dissipation compared to the previous ROM-less designs. Analytical gate-level comparisons and synthesis results further confirmed the superiority of their design.

An alternative strategy to enhance the efficiency of reverse converters involves careful selection of moduli sets. Mohan (Mohan, 2016) explored this concept by designing converters for the moduli set $\{2^{n+1}-3, 2^n-1, 2^{n-1}-1\}$. By capitalizing on the unique characteristics of these moduli, the proposed converter design demonstrated improved efficiency and implementation simplicity.

In another effort, Patronik and Piestrak (Patronik & Piestrak, 2018) introduced an innovative approach to simplifying reverse converters by shifting constant additions to residue data-path channels. Their method formulates the reverse conversion equation into two distinct parts: one dependent on input variables and the other a single constant. Instead of processing the constant within the reverse converter, it is shifted to the residue data paths, reducing hardware costs by eliminating an operand from the multi-operand adder. Their proposed design, tested on the 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$, demonstrated improvements in area and power efficiency.

Beyond efficiency improvements, multifunctional RNS units have been proposed to enhance the versatility of reverse converters. Mojahed et al. (Mojahed et al., 2021b) designed a multifunctional unit that combined reverse conversion and sign detection based on a 5-moduli set $\{2^{2n}, 2^n+1, 2^n-1, 2^n+3, 2^n-3\}$. By reusing hardware components, they reduced area overhead and computational complexity. This approach enhances the practicality of RNS in applications requiring signed arithmetic operations.

Meanwhile, Akbari et al. (Akbari et al., 2021) focused on a high-speed, low-area residue-to-binary converter for the 4-moduli set $\{2^{4n}, 2^{2n}+1, 2^n+1, 2^n-1\}$, employing CRT-I for optimization. Their research aimed at reducing the area and power consumption of the converters while maintaining high-speed performance. By optimizing the design using CRT-I techniques, they were able to achieve improvements in efficiency, making their converters suitable for high-performance applications.

Energy efficiency has also been a focal point in recent research efforts. Hence, Majd and Molahosseini (Majd & Molahosseini, 2022) introduced an energy-efficient residue-to-binary conversion architecture without using modulo adders for the 4-moduli set $\{2^{n+k}, 2^{2n+1}-1, 2^n+1, 2^n-1\}$. Their design eliminated the need for traditional modulo adders, resulting in a reduction in power consumption. This energy-efficient approach is particularly beneficial for low-power applications where energy consumption is a critical concern.

Building upon the principles of parallelism, an approach utilizing CRT-based processing techniques was introduced by Selianinau and Povstenko (Selianinau & Povstenko, 2022) to enhance the speed and reliability of reverse conversion. By distributing computations across independent modular channels, this method reduced conversion time while maintaining accuracy, making it an effective method for real-time applications where rapid and precise residue-to-binary transformation is essential.

Further progress in energy-efficient RNS design includes the work of Givaki et al. (Givaki et al., 2023), who proposed a generalized RNS approach for ultra-low-power arithmetic circuits utilizing deterministic bit-streams. Their method successfully reduced power consumption while preserving computational accuracy, making it suitable for energy-sensitive applications, such as energy-harvesting and bio-implantable devices, where peak power consumption and heat dissipation are critical challenges.

Moreover, advancements in nanometer-scale technology have necessitated robust error mitigation strategies. To address these challenges, Mansoor et al. (Mansoor et al., 2023) integrated multi-bit soft error correction mechanisms at the 7 nm node, enhancing the resilience and stability of reverse converters. Their approach uses majority voting and redundant RNS (RRNS) to effectively detect and correct soft errors, ensuring reliable operation in power-constrained and high-performance computing environments. This research is particularly relevant for cutting-edge technologies where error correction is a critical requirement.

Advancing further on efforts to improve flexibility and efficiency in 3-moduli RNS architectures, Patronik and Piestrak (Patronik & Piestrak, 2023) introduced a reverse converter design for the flexible moduli set $\{2^k, 2^n-1, 2^n+1\}$. Their work formulated the converter equations using the New CRT-I, separating the constant and variable components of the conversion process. This separation facilitates hardware optimizations such as shifting constant additions to the residue data-path channels to reduce delay, area, and power consumption. By developing four different versions of the converter, their approach achieved performance gains compared to other designs.

Expanding the scope of moduli set evaluations, Fernandes et al. (Fernandes et al., 2024) provided a comprehensive analysis of reverse converters for various moduli sets. Their work offered a detailed evaluation of different moduli sets, highlighting their advantages and limitations. By comparing various sets, they were able to provide valuable insights into the selection of moduli sets for different applications, contributing to a better understanding of moduli set optimization in RNS converter design.

In parallel to these developments, Ahmadifar and Torabi (Ahmadifar & Torabi, 2024) presented adder-only reverse converters for a 5-moduli set $\{2^q, 2^q-1, 2^{q+1}\pm 1, 2^{q+2}-1\}$, demonstrating a novel approach to reducing computational complexity. Their design focused on using only adders, eliminating the need for more complex arithmetic operations. This simplification resulted in converters that were both efficient and easy to implement. Analytical results and synthesis evaluations confirmed that this adder-only approach provides an efficient alternative to traditional designs, contributing to ongoing efforts toward practical RNS reverse converter solutions.

The reviewed techniques in RNS reverse converter design highlight significant advancements in optimizing speed, area, and power efficiency. Key advantages include innovative architectures that enhance computational efficiency and versatile multifunctional units that streamline the design process. However, these techniques also face drawbacks, such as the complexity and time-consuming nature of developing and implementing new moduli sets and CRT-based methods. Despite these challenges, the progress made in this field lays a strong foundation for future innovations, aiming to further simplify and enhance the efficiency of RNS converter designs. Table 1 provides a summary of the reviewed RNS reverse converter approaches, highlighting authors, publication years, moduli sets, and the primary innovations aimed at improving efficiency, computational complexity, and hardware optimization.

**Table 1**
Summary of the reviewed studies on RNS reverse converter design.

| Authors/Ref | Year | Moduli Set | Key Contributions |
|---|---|---|---|
| Molahosseini et al. (Molahosseini et al., 2008) | 2010 | 3-moduli set $\{2^\alpha, 2^\beta-1, 2^\beta+1\}$ | Optimized reverse converters using specific moduli properties, significantly enhancing speed and efficiency. |
| Molahosseini et al. (Molahosseini et al., 2009) | 2012 | 4-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{2n+1}-1\}$ $\{2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1\}$ | Built upon earlier work by introducing New-CRT-based techniques to substantially reduce computational complexity and improve performance. |
| Jaberipur et al. (Jaberipur & Ahmadifar, 2014) | 2014 | 4-moduli set $\{2^q\pm 1, 2^q\pm 3\}$ | Proposed a ROM-less reverse converter using two-stage New CRT with conjugate moduli grouping, achieving notable reductions in delay (22 %), area (19 %), and power dissipation (8 %). |
| Mohan (Mohan, 2016) | 2016 | 3-moduli set $\{2^{n+1}-3, 2^n-1, 2^{n-1}-1\}$ | Focused on moduli set optimization to simplify implementation, balancing efficiency with reduced hardware complexity. |
| Patronik and Piestrak (Patronik & Piestrak, 2018) | 2018 | 3-moduli set $\{2^n-1, 2^n, 2^n+1\}$ | Developed reverse converters utilizing constant shifting in residue datapath channels, significantly lowering hardware costs and improving area and power efficiency. |
| Mojahed et al. (Mojahed et al., 2021b) | 2021 | 5-moduli set $\{2^{2n}, 2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ | Introduced a multifunctional reverse converter unit integrating sign detection, reducing area overhead (31 %) and delay (28 %). |
| Akbari et al. (Akbari et al., 2021) | 2022 | 4-moduli set $\{2^{4n}, 2^{2n}+1, 2^n+1, 2^n-1\}$ | Designed high-speed, low-area converters using CRT-I optimizations, reducing power and area while maintaining high performance. |
| Majd and Molahosseini (Majd & Molahosseini, 2022) | 2022 | 4-moduli set $\{2^{n+k}, 2^{2n+1}-1, 2^n+1, 2^n-1\}$ | Developed an energy-efficient residue-to-binary converter by eliminating modulo adders, achieving substantial power savings. |
| Selianinau and Povstenko (Selianinau & Povstenko, 2022) | 2022 | General CRT-based approach | Employed parallel processing techniques to accelerate conversion time, enhancing reliability and suitability for real-time applications. |
| Givaki et al. (Givaki et al., 2023) | 2023 | Generalized RNS | Created ultra-low-power arithmetic circuits using deterministic bit-streams, ideal for energy-constrained applications such as bio-implantable devices. |
| Mansoor et al. (Mansoor et al., 2023) | 2023 | RRNS (7 nm technology) | Integrated multi-bit soft error correction mechanisms, significantly enhancing converter robustness and reliability at 7 nm technology. |
| Patronik and Piestrak (Patronik & Piestrak, 2023) | 2023 | 3-moduli set $\{2^k, 2^n-1, 2^n+1\}$ | Introduced flexible moduli set converters using New CRT-I, significantly reducing delay, area, and power via optimized constant shifting. |
| Fernandes et al. (Fernandes et al., 2024) | 2023 | Various moduli sets | Conducted comprehensive comparative analyses, providing valuable insights into moduli selection for diverse RNS applications. |
| Ahmadifar and Torabi (Ahmadifar & Torabi, 2024) | 2024 | 5-moduli set $\{2^q, 2^q-1, 2^{q+1}\pm 1, 2^{q+2}-1\}$ | Proposed adder-only reverse converters, effectively reducing computational complexity and hardware implementation costs. |

### 2.2. GP for circuit synthesis

GP and its variants, specifically CGP, have become powerful tools for automating the design and optimization of digital circuits. By mimicking natural evolution, GP techniques evolve solutions over generations, leading to optimized designs that often outperform manually crafted circuits. CGP, initially developed by Miller et al. in 1997 and formalized in 2000, has significantly advanced the field of digital circuit design (Miller & Turner, 2015; Miller, 2020).

The application of evolutionary techniques to approximate circuit design has demonstrated the effectiveness of GP in balancing circuit complexity with performance. Research by Vasicek and Sekanina (Vasicek & Sekanina, 2014) introduced an approach based on CGP to automate the design of approximate digital circuits. Their method used the error-resilient nature of certain applications, enabling circuit accuracy to be traded for reduced hardware costs, lower power consumption, and improved computational efficiency. By evolving gate-level and functional-level circuits, their approach systematically explored trade-offs between accuracy and resource utilization, producing circuits that offered substantial area and energy savings compared to conventional designs. A key innovation in their work was the introduction of a heuristic population seeding mechanism, which improved both the quality of evolved circuits and the efficiency of the evolutionary process. Their

experimental results demonstrated the superiority of CGP-based approximate designs for arithmetic circuits, particularly in applications where controlled functional deviations could be tolerated in exchange for significant resource optimization.

Further extending the capabilities of evolutionary algorithms, Hrbáček (Hrbácek, 2017) introduced multi-objective parallel evolutionary circuit design, optimizing multiple factors such as power consumption, speed, and area simultaneously. His approach, based on CGP and NSGA-II, uses a highly parallelized framework to improve scalability and efficiency. By distributing computations across multiple processing units, the method enables a broader exploration of the solution space, leading to more diverse and optimized circuit designs. The effectiveness of this approach was demonstrated in various applications, including approximate combinational arithmetic circuits, bent Boolean functions, etc. Experimental results showed that this parallel multi-objective strategy outperformed conventional methods, highlighting the versatility and effectiveness of CGP in handling complex problems for automated circuit synthesis.

Adaptive population sizing has been another crucial innovation in CGP. Lima et al. (Lima et al., 2019) proposed a multi-objective CGP approach that dynamically adjusts population size based on the number of non-dominated solutions. Their method begins with a single candidate solution and allows the population to grow up to a predefined maximum, enhancing diversity while preventing excessive computational overhead. When the population size surpasses a threshold, individuals with the lowest crowding distances are removed, ensuring an efficient balance between exploration and convergence. This adaptive strategy was evaluated on combinational circuits such as adders, multipliers, and ALUs, demonstrating its effectiveness in producing approximate circuits with optimized trade-offs between error, area, power dissipation, and delay. Compared to other methods, this approach led to solutions with superior energy efficiency and computational performance, highlighting the benefits of dynamic population control in evolutionary circuit design.

Circuit resynthesis techniques have also benefited significantly from evolutionary algorithms. Kocnova and Vasicek (Kocnova & Vasicek, 2020) introduced an EA-based resynthesis approach that addresses scalability issues in logic circuit optimization. Their method employs a local resynthesis strategy, where complex circuits are decomposed into smaller sub-circuits that are individually optimized using CGP before being reintegrated. This approach mitigates the challenges of large-scale evolutionary optimization by allowing targeted improvements while preserving overall circuit functionality. The experimental evaluation demonstrated that evolutionary resynthesis achieved superior results compared to global optimization techniques, removing 25.1 % more redundant gates on average, with substantial improvements particularly in arithmetic circuits. The proposed method efficiently balances scalability and optimization quality, reinforcing the practical advantages of GP in circuit refinement and resource efficiency.

Expanding the scope of mutation strategies in evolutionary circuit design, de Souza and Bernardino (Souza & Bernardino, 2020) analyzed the impact of mutation operators in multi-objective CGP for optimizing approximate combinational logic circuits. Their study compared point mutation, the standard mutation operator in CGP, with single active mutation, which ensures that at least one active node is altered, leading to more effective evolutionary search. By integrating CGP with NSGA-II and adaptive population size, the proposed method improved the trade-offs between error, power dissipation, and delay, addressing key challenges in approximate computing. The evaluation, conducted on 15 heterogeneous benchmark circuits, demonstrated that single active mutation consistently outperformed point mutation, particularly in medium-sized circuits, resulting in faster convergence and superior optimization. Experimental results highlighted that combining single active mutation with NSGA-II and an initial feasible solution achieved the best outcomes in most cases, reinforcing the importance of advanced mutation strategies in enhancing the efficiency and adaptability of CGP for multi-objective circuit synthesis.

Convergence improvements in CGP have been another area of research focus. Milano and Nolfi (Milano & Nolfi, 2021) introduced a preferential selection mechanism that favors larger and potentially more complex solutions during the selection process, enhancing CGP's ability to explore complex designs and achieve better performance. Their study demonstrated that selecting phenotypically larger solutions among equally fit candidates increases evolvability, allowing for faster convergence and the discovery of superior solutions. This approach was validated on two distinct benchmarks: an 8-bit parity problem and the Paige regression problem, where it was shown to accelerate evolutionary progress by preserving genetic diversity and enhancing robustness. Further performance gains were achieved by integrating a self-adaptive mutation rate using the one-fifth success rule, which dynamically adjusts mutation probability based on evolutionary progress. This combination of preferential selection and adaptive mutation proved especially effective in problems where neutrality plays a lesser role, reinforcing the importance of solution size in evolutionary optimization.

Optimization of approximate circuits has also advanced with CGP-based methodologies. Berndt et al. (Berndt et al., 2022) proposed a CGP-based logic flow designed to optimize both accuracy and circuit size by evolutionary techniques. Their approach utilizes a two-step process: first, it prioritizes the selection of functionally larger solutions to enhance convergence, and then it switches to minimizing circuit size while maintaining accuracy. The proposed method refines solutions derived from other techniques, using them as a starting point for further evolutionary optimization. Evaluations on approximate circuits demonstrated that the CGP-based flow achieved at least a 22.6 % improvement in balancing accuracy and size compared to state-of-the-art approaches. The framework proved particularly valuable for error-tolerant applications, where a trade-off between precision and resource constraints is essential.

The application of CGP in AI hardware design has also been explored in the study by Prashanth and Rao (Prashanth & Rao, 2024), where they proposed an accelerated synthesis approach for AI hardware subsystems, particularly for ASIC implementations. Their method addresses the challenges of synthesizing complex nonlinear functions, such as activation and power functions, which are difficult to realize using conventional synthesis techniques. By using CGP, their approach enables hierarchical design-space exploration and the integration of custom logic gates, facilitating the generation of highly optimized AI circuits. A key innovation in their work is the introduction of binary-weighted fitness (BwF) to enhance the correlation between evolved circuit outputs and expected results, alongside an exponentially varying mutation rate (eVar) to accelerate convergence. These enhancements significantly improved synthesis efficiency, achieving a 3 $\times$ reduction in synthesis time for complex functions while maintaining hardware accuracy. Their experimental results demonstrated CGP's potential in ASIC design of AI hardware, particularly for neural network accelerators and computational subsystems that require optimized nonlinear operations.

Further innovations in GP have included semantic mutation operators, which enhance evolutionary search efficiency by incorporating domain-specific knowledge. Husa and Sekanina (Husa & Sekanina, 2024) introduced a semantic-aware mutation operator specifically designed for evolving bent Boolean functions, which are crucial in cryptographic applications. Their approach evaluates a function's nonlinearity in detail and ensures that mutations do not disrupt highly nonlinear components while adjusting input variable distribution to maximize nonlinearity. By using the property that nonlinearity remains invariant under affine transformations, the proposed mutation strategy systematically avoids detrimental mutations and directs evolution toward highly nonlinear solutions. Experimental results demonstrated that this method significantly reduces the number of evaluations required for genetic programming, achieving a threefold speedup for 12-input bent functions and nearly a sixfold speedup for 20-input functions compared to conventional mutation operators. The findings highlight

the potential of semantic-driven GP techniques in efficiently designing cryptographic boolean functions with superior properties.

Expanding the capabilities of CGP, Jamróz (Jamróz, 2024) introduced Multivalued CGP (M−CGP) for the automatic design of digital sequential circuits. Unlike traditional CGP, which struggles with sequential systems due to their dependence on previous states, M−CGP enables the evolution of circuits without requiring predefined state tables or transition graphs. This approach treats the target system as a black box, learning its behavior solely from observed input–output responses, making it particularly useful for reverse engineering unknown systems or designing circuits based on external reactions. A key innovation in M−CGP is the division of functional nodes into multiple groups, allowing simultaneous processing of logic gates and flip-flops while maintaining separate memory spaces for their operations. Experimental results demonstrated M−CGP's superiority over standard CGP in designing various sequential systems, such as counters, registers, sequence detectors, and finite state machines, proving its effectiveness for automated sequential circuit synthesis where traditional CGP is insufficient.

The reviewed techniques in applying GP for circuit synthesis showcase substantial improvements in design automation, efficiency, and performance optimization. Innovations such as multi-objective optimization and different mutation strategies have expanded GP's applicability to approximate computing, cryptographic functions, and sequential circuit design. These techniques enhance design space exploration, resource efficiency, and computational performance, surpassing traditional methods. Notably, recent contributions like M−CGP for sequential circuits and AI-specific optimizations reinforce GP's adaptability to emerging hardware challenges. These contributions have laid the groundwork for further innovations in the automated design and optimization of RNS converters. Table 2 summarizes the reviewed studies on the application of GP for circuit synthesis.

**Table 2**
Summary of the reviewed studies on the application of GP for circuit synthesis.

| Authors/Ref | Year | Applications/ Benchmarks | Key Contributions |
|---|---|---|---|
| Vasicek & Sekanina (Vasicek & Sekanina, 2014) | 2015 | Approximate arithmetic circuits | Introduced heuristic population seeding in CGP for approximate circuits, achieving significant area and energy savings with faster convergence. |
| Hrbáček (Hrbácek, 2017) | 2017 | Approximate arithmetic circuits, bent Boolean functions | Developed a parallel multi-objective CGP framework using NSGA-II, improving scalability, diversity, and optimization efficiency. |
| Lima et al. (Lima et al., 2019) | 2019 | Arithmetic circuits (adders, multipliers, ALUs) | Proposed adaptive population sizing in CGP, enhancing diversity, efficiency, and multi-objective optimization for arithmetic circuits. |
| Kocnova & Vasicek (Kocnova & Vasicek, 2020) | 2020 | Logic circuits, especially arithmetic circuits | Introduced local resynthesis in CGP for scalable circuit optimization, achieving a 25.1 % reduction in redundant gates. |
| de Souza & Bernardino (Souza & Bernardino, 2020) | 2020 | Approximate combinational logic circuits | Analyzed SAM vs. PM in MOCGP, integrating NSGA-II with APS to improve convergence and trade-offs in error, power, and delay. |
| Milano & Nolfi (Milano & Nolfi, 2021) | 2021 | 8-bit parity problem, Paige regression problem | Developed preferential selection of larger solutions in CGP, enhancing evolvability, convergence speed, and robustness. |
| Berndt et al. (Berndt et al., 2022) | 2022 | Approximate combinational circuits | Proposed a two-stage CGP logic flow prioritizing accuracy followed by size optimization, achieving a 22.6 % improvement in accuracy-size trade-offs. |
| Prashanth & Rao (Prashanth & Rao, 2024) | 2024 | AI hardware synthesis (ASIC, nonlinear functions) | Introduced BwF fitness and eVar mutation, significantly reducing synthesis time by 3 × while maintaining hardware accuracy. |
| Husa & Sekanina (Husa & Sekanina, 2024) | 2024 | Bent Boolean functions (cryptography) | Developed semantic-aware mutation for cryptographic Boolean functions, improving convergence speed by 3–6 × and increasing nonlinearity. |
| Jamróz (Jamróz, 2024) | 2024 | FSM, counters, registers | Introduced M−CGP for sequential circuit synthesis without predefined states, outperforming standard CGP in efficiency. |

### 2.3. Our contributions against previous approaches

Despite significant advancements in RNS reverse converter design and GP-based circuit synthesis, their integration remains largely unexplored. To bridge this gap, we introduce CGP-SA, an automated evolutionary framework specifically designed for residue-to-binary converter synthesis. Unlike conventional methods that rely on manual intervention, CGP-SA systematically explores and optimizes both module-level and gate-level representations. Our approach employs a structured two-phase optimization process. First, CGP constructs module-level computational architectures, while SA fine-tunes critical parameters, such as bitwise shift amounts, to enhance accuracy and efficiency. In the second phase, gate-level refinements utilize an adaptive CGP strategy, where a dynamically adjusted mutation rate promotes early exploration and gradually stabilizes through exponential decay for improved convergence. Additionally, an adaptive tournament selection mechanism progressively increases tournament sizes, balancing diversity with convergence toward optimal solutions. A key innovation lies in the cost function, which integrates three metrics: correctness, latency, and resource usage. Correctness is prioritized through the normalized average Hamming distance between candidate outputs and expected results. Latency is minimized by normalizing the number of active computational stages within the CGP grid, encouraging shorter critical paths. Resource usage is quantified based on active node costs, factoring in hardware complexity, power consumption, and area overhead. In summary, our CGP-SA methodology represents a significant advancement in the field, offering a novel, automated, and efficient solution for residue-to-binary converter design in RNS applications.

## 3. Preliminaries

This section aims to provide a comprehensive overview of the principles underlying CGP and RNS reverse conversion. A clear understanding of these principles is essential for grasping how CGP can be utilized to optimize the process of RNS reverse conversion. By exploring the theoretical foundations and practical implications of CGP and RNS, we establish a strong basis for designing efficient and scalable RNS reverse converter circuits. Additionally, this discussion highlights the inherent challenges in both domains, such as ensuring accuracy, reducing computational overhead, and achieving high performance in practical applications.

### 3.1. Cartesian genetic Programing

As mentioned before, CGP originated as a method for evolving digital circuits and was formalized as a general genetic programming approach in the late 1990 s. Unlike tree-based genetic programming, CGP employs a two-dimensional grid of computational nodes to represent solutions.

This grid-based structure allows for compact and flexible genotype-to-phenotype mapping, offering a significant advantage in the optimization of complex systems. The term "Cartesian" reflects the structured grid representation, analogous to Cartesian coordinates, which facilitates the modeling of solutions as directed acyclic graphs (DAGs). This allows CGP to naturally support parallelism and reuse of intermediate computations within evolved solutions. Fig. 1 provides a visualization of a CGP grid, illustrating how nodes are connected to inputs and outputs. The genotype structure is illustrated below the grid in this figure. Each function gene, $F_{i,j}$ is represented as an integer index referencing a function in a predefined table. Similarly, the connection genes, $C^k_{i,j}$, serve as data addresses and are integers ranging from 0 to the last node in the preceding columns of nodes.

Each node in the grid performs a function selected from a predefined set, which may include logical and arithmetic operations, and receives inputs either from program inputs or from outputs of earlier nodes, depending on a user-defined levels-back parameter. These features collectively allow CGP to produce highly expressive and compact representations. A defining characteristic of CGP is its support for inactive or non-coding genes, which are nodes present in the genotype but not expressed in the final phenotype. Although they do not directly contribute to the output, these non-coding genes enable neutral mutations that help preserve genetic diversity and allow the evolutionary process to escape local optima.

CGP also benefits from the reusability of nodes, allowing them to serve as inputs to multiple other nodes. This supports implicit reuse of computations and contributes to both solution efficiency and structural compactness. The evolutionary search is primarily driven by mutation, which can modify function, connection, or output genes, potentially altering the behavior or structure of the resulting phenotype. Several key parameters, including grid size, node arity, and levels-back, can be configured to control the expressiveness and performance of the evolved circuits. A more detailed explanation of CGP's encoding structure, functional parameters, mutation dynamics, and its application to digital circuit design is provided in Appendix A.

### 3.2. RNS reverse converter

The Residue Number System (RNS) is a non-positional number system defined by a set of pairwise relatively prime integers, referred to as the moduli $\{p_1, p_2, \ldots, p_n\}$. A number $X$ within the dynamic range $P = \prod_{i=1}^{n} p_i$ can be uniquely represented by its residues:

$$X \rightarrow (x_1, x_2, \ldots, x_n), \text{ where } x_i = X \bmod p_i \text{ for } i = 1, 2, \ldots, n \quad (1)$$

This representation enables carry-free and fully parallel arithmetic, as all computations can be performed independently for each residue channel. Due to this inherent parallelism, RNS is widely utilized in high-performance domains such as digital signal processing, cryptography, and fault-tolerant computing.

While the forward conversion of a number $X$ into its residue representation is straightforward, reverse conversion, which reconstructs $X$ from its residues, poses significant computational challenges. The efficiency and accuracy of reverse conversion algorithms directly affect the practicality of RNS-based systems. Several methods have been developed for reverse conversion in RNS, each offering different trade-offs in terms of computational complexity, hardware requirements, and efficiency. The most common techniques include CRT and its modifications i.e., New CRT-I and New CRT-II.

The traditional CRT reconstructs $X$ from residues $(x_1, x_2, \ldots, x_n)$ using Equation (2):

$$X = \left| \sum_{i=1}^{n} (P_i . (P_i^{-1} \bmod p_i) . x_i) \right|_P, \quad (2)$$

where, $P_i = P/p_i$, and $P_i^{-1}$ is the multiplicative inverse of $P_i$ mod $p_i$, satisfying $P_i . P_i^{-1} \equiv 1 \bmod p_i$. Although the traditional CRT is mathematically elegant and straightforward to implement in software, it poses challenges for hardware implementation because it requires managing large numerical ranges, executing full-width modular multiplications, and calculating multiple modular inverses.

To overcome these limitations, New CRT-I and New CRT-II were developed with a focus on hardware efficiency. These methods reduce arithmetic complexity, lower resource usage, and enhance throughput. New CRT-I uses a sequential reconstruction strategy that incrementally combines residues through intermediate computations and modular inverses. It reduces the number of large modular operations and is well suited for area-constrained designs. New CRT-II adopts a divide-and-conquer strategy to enable parallelism. It partitions the moduli set into smaller groups, computes partial results independently, and combines them using weighted modular operations. This structure improves scalability and is ideal for high-speed applications where latency is critical. These enhancements make the New CRT methods highly effective for real-time RNS implementations in areas such as cryptography and digital signal processing. Additional details on both algorithms are provided in Appendix B.

## 4. Proposed methodology

This section provides a detailed explanation of the methodology employed to design optimized RNS reverse converters using the

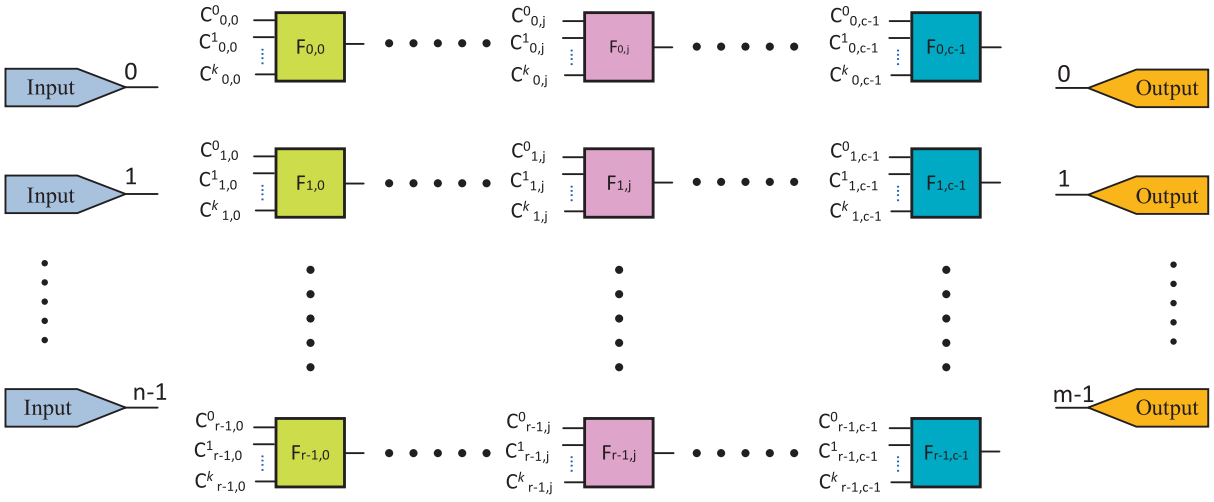


**Fig. 1.** Visualization of a CGP grid comprising of input, output and function nodes.

proposed hybrid CGP-SA approach. As shown in Fig. 2, the approach consists of a two-phase process, where the first phase focuses on developing a module-level architecture of the reverse converter using a set of high-level operations, and the second phase emphasizes gate-level optimization to enhance hardware efficiency.

In Phase 1, the design begins with a function set comprising arithmetic operations such as addition, subtraction, and modular addition, as well as bit-level operations including logical shifts, circular shifts, complement, and concatenation. These operations serve as the fundamental building blocks for constructing an RNS reverse converter at the module level. To explore a diverse range of candidate designs, CGP is employed, where each solution is encoded as a chromosome within a CGP grid. The optimization process is guided by a cost function that evaluates three main factors: correctness, latency, and resource usage. Correctness is determined by comparing the candidate design's output against the expected values from a truth table. Latency is assessed by counting the number of active columns in the CGP grid, which represents the critical path length. Eventually, resource usage is quantified by counting the active nodes in the grid. Since certain operations, such as bitwise shifts, require parameter tuning for optimal performance, SA is incorporated into the CGP process which serves as a local search mechanism that refines these parameters. Through iterative evolution, the algorithm progressively minimizes errors, reduces latency, and optimizes resource consumption, ultimately generating an efficient high-level description of the reverse converter.

In Phase 2, the high-level design is translated into a gate-level representation, where each functional operation is mapped to a network of basic logic gates such as AND, OR, XOR, and NOT, as well as higher-order components like multiplexers, sum, and carry gates. This phase initializes a new CGP optimization process, but unlike Phase 1, it does not require SA since the gate-level design consists of fixed Boolean primitives that do not involve parameter tuning. The cost function remains the same, evaluating correctness by verifying outputs against the truth table, minimizing latency by reducing the number of active columns in the gate-level grid, and optimizing resource usage by minimizing the number of gates. Since the number of nodes and connections is significantly larger in this phase compared to the previous one, adaptive mutation rates and tournament selection sizes are introduced to enhance the balance between exploration and exploitation in CGP. This phase ultimately simplifies the circuit while maintaining functional accuracy, ensuring that the final design is both hardware-efficient and implementation-ready.

By incorporating these two phases, the proposed methodology effectively balances the objectives of correctness, latency, and resource usage. The high-level architecture generated in the first phase provides a foundation for efficient design, while the gate-level optimization in the second phase ensures the final implementation is hardware-efficient and ready for practical use. This approach harnesses the power of CGP, augmented by SA for precise fine-tuning of operator-specific parameters, enabling a comprehensive exploration of the design space and the generation of optimized RNS reverse converters that meet the specific performance needs and resource constraints.

In this study, we adopt a one-shot, two-phase approach to independently validate and isolate the effects of each stage in the CGP-SA framework. This decoupled design enables a modular evaluation of both the high-level architecture generation and the low-level logic optimization processes, making it easier to analyze the contribution of each phase. Moreover, developing a reliable, unbiased mapping between these two distinct genome representations is nontrivial and risks introducing structural bias or prematurely constraining the search space. By avoiding tight coupling between phases, the framework maintains greater flexibility within each design space and ensures that Phase 2 is free to discover novel gate-level simplifications that may diverge from the original high-level architecture.

### 4.1. Phsse 1: Module-Level design

As mentioned above, the first phase of the methodology develops a high-level architecture for a RNS reverse converter using hybrid CGP-SA algorithm. This involves defining functional building blocks, generating a reference truth table, encoding designs in a CGP-SA framework, and optimizing them through an evolutionary process. A cost function guides the optimization by prioritizing correctness while also considering latency and resource usage. The overall flowchart of the CGP-SA algorithm is shown in Fig. 3.

The design of the RNS reverse converter is constructed using a set of predefined operations, referred to as primitives, which serve as the fundamental building blocks for candidate designs. The primitive set includes arithmetic operations such as addition, subtraction, and modular addition, as well as bitwise operations, including logical shifts, circular left and right shifts, complement, and concatenation. These operations are specifically selected to ensure compatibility with RNS-specific requirements.

Certain primitives, such as bitwise shift and modular arithmetic operations, incorporate a set of parameters (e.g., shift amount or modulo base) that govern their behavior, ensuring that candidate designs can effectively capture the complex relationships necessary for accurate reverse conversion. The parameters of certain operators, such as the modulo base in modular addition, are always defined as $2^{(operand\ bits)}-1$. However, the correct functionality of an RNS reverse converter depends on selecting an appropriate shift amount ($n$) for bitwise shift operators in a CGP grid.

To determine the optimal values for these parameters, this study proposes a hybrid CGP-SA algorithm, which integrates SA as a local search mechanism within the CGP optimization framework. In this approach, CGP is responsible for evolving the overall computational structure of the RNS converter by the selection of operators and their connectivity within the reverse converter design. Meanwhile, SA is employed to tune operator-specific parameters (i.e. the shift amount ($n$) in bitwise shift operations), ensuring that the evolved designs can achieve functional correctness. SA was chosen for its simplicity, flexibility, and well-established effectiveness as a single-solution metaheuristic in

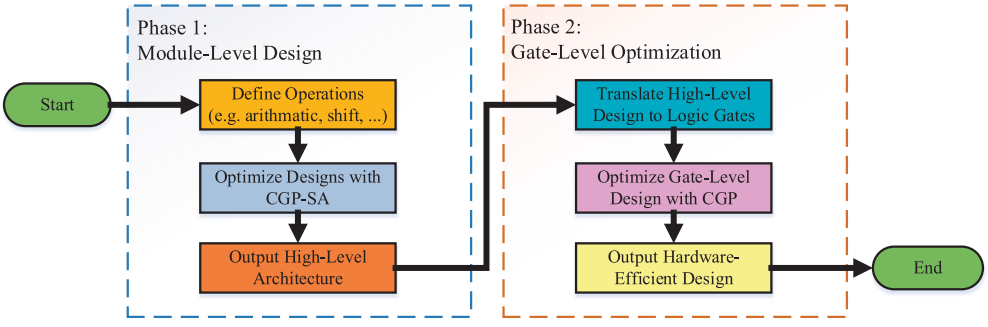


**Fig. 2.** The overall flowchart of the proposed methodology.

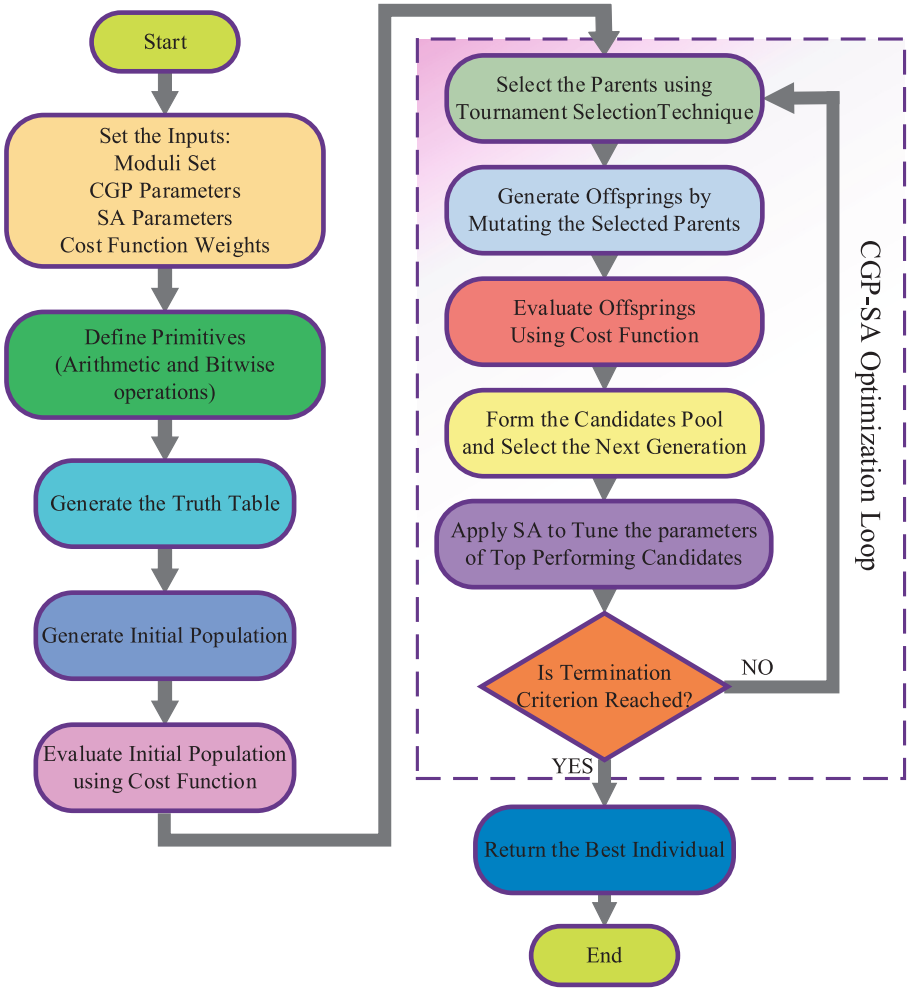


**Fig. 3.** The CGP-SA genome representation.

local search tasks. Its probabilistic acceptance criterion enables it to escape local optima and refine discrete parameters that standard CGP mutation operators often fail to optimize efficiently. In the proposed hybrid CGP-SA framework, the genome of a candidate solution is encoded as a vector that represents the computational structure and associated parameters of the reverse converter. Each node in the genome is encoded as shown in Fig. 4.

According to the Fig. 4, The CGP-SA Genome Representation (top section) illustrates a candidate solution as a vector containing both the computational structure and the operator-specific parameters of the reverse converter. Each functional gene ($F_i$) represents an operation, while the connection genes define the flow of data between these nodes. Additionally, the parameter genes ($P_i$) are included in the genome, allowing SA to refine their values for improved performance. The bottom-left section shows the CGP Genome and the SA Solution separately. The CGP Genome (bottom-left section) represents a traditional CGP structure, consisting of function nodes and their respective connections. In this approach, CGP is responsible for evolving the architecture of the RNS converter by selecting appropriate operators and determining their connectivity. However, it does not explicitly handle the fine-tuning of node-specific parameters, which may limit the solution's efficiency. On the other hand, the SA Solution (bottom-right section) focuses exclusively on the parameter optimization aspect. It represents a set of numerical parameters that are fine-tuned using SA, ensuring that each operator performs optimally within the evolved CGP structure. By integrating CGP and SA, this hybrid approach optimizes both the structural framework and operator-specific parameters, ensuring an efficient computational structure, functional correctness,

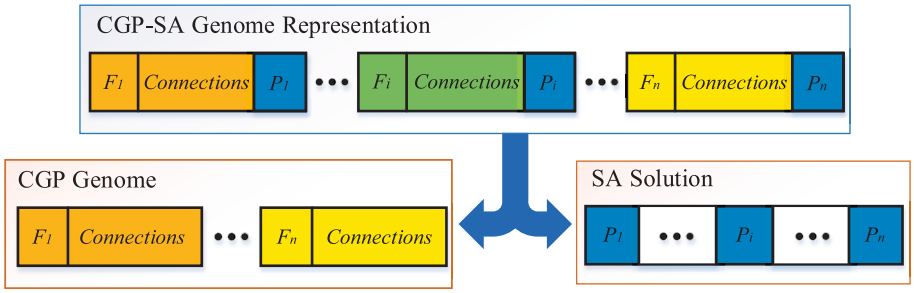


**Fig. 4.** The CGP-SA genome representation.

and enhanced performance.

Subsequently, the truth table is generated that serves as the ground truth for evaluating the correctness of candidate designs. It is generated by systematically enumerating all possible residue combinations based on the chosen moduli. For a given set of moduli, each residue can take values ranging from 0 to (modulus − 1), and the total number of unique residue combinations is equal to the product of the moduli, ensuring a complete representation of the input space. To establish the integer equivalence of each residue combination, the CRT or alternative modular reconstruction methods are applied. These methods compute the corresponding integer representation of each residue set, forming a reference mapping that defines the expected outputs of the RNS reverse converter. This truth table serves as the benchmark for assessing the accuracy of candidate designs during the CGP optimization process, ensuring that evolved solutions correctly reconstruct integers from their modular residues.

As previously mentioned, in the CGP process, CGP genomes are translated to grids consisting of input nodes, functional nodes, and output nodes. Here, the input nodes correspond to the residues of the moduli, while the functional nodes perform operations selected from the primitive set, and the output nodes generate the final converted value of the reverse conversion. The grid is parameterized to support both sequential and parallel computations, where the number of input nodes equals the number of moduli, and the output nodes typically consist of a single node representing the final integer result. The grid structure includes multiple rows for parallel computations and columns that define computation stages. Connections between nodes are governed by the "levels back" parameter, which determines the range of preceding nodes that each functional node can access, influencing connectivity and computational depth. The genome encoding specifies the operations assigned to each node and their interconnections, allowing the CGP framework to systematically explore and optimize a wide range of RNS reverse converter architectures.

The cost function is central to guiding the evolutionary process by evaluating candidate designs based on three key criteria: correctness, latency, and resource usage. Correctness is the primary objective and is determined by comparing the outputs of a candidate design with the expected outputs from the truth table. The correctness cost is calculated as:

$$Cost_{correctness} = \frac{1}{N_{entries} \times B} \sum_{i=1}^{N_{entries}} HD(y_i, y_i^{\text{expected}}) = \frac{1}{N_{entries} \times B} \sum_{i=1}^{N_{entries}} \sum_{j=1}^{B} \mathrm{I}(y_i \neq y_i^{\text{expected}}) \tag{3}$$

where $N_{entries}$ is the total number of truth table entries, $B$ is the number of bits in the output for each entry, and $HD(y_i, y_i^{\text{expected}})$ represents the hamming distance between the candidate solution's output ($y_i$) and the expected output ($y_i^{\text{expected}}$) for the $i$-th entry. This expression computes the average bit-level error across all entries in the truth table by summing the number of bit differences for each entry, and then normalizing the total error by the overall number of bits evaluated. The indicator function $\mathrm{I}(y_i \neq y_i^{\text{expected}})$ returns 1 if the $j$-th bit of the candidate output does not match the corresponding bit in the expected output, and 0 otherwise. As a result, this cost function provides a normalized measure of the candidate circuit's correctness, where a value of 0 indicates a perfect match and a value closer to 1 indicates significant deviation from the expected outputs.

In this study, latency is measured by counting the number of active columns in the CGP grid. Active columns represent the stages of computation that contribute to the final output. The latency cost is expressed as:

$$Cost_{latency} = \frac{N_{Active\_Columns}}{N_C} \tag{4}$$

where $N_{Active\_Columns}$ represents the number of active columns in the CGP grid. To normalize this value, it is divided by the total number of columns ($N_C$) in the CGP grid. By minimizing this cost, the optimization encourages efficient designs that require fewer computational stages, thus a shorter critical path length.

Resource usage is quantified by counting the active nodes in the CGP grid, where each active node contributes to the computational complexity and hardware resource consumption of the designed RNS reverse converter. To systematically assess the resource efficiency of a given solution, a cost function is defined that accounts for the cumulative impact of the operations performed by these active nodes. Specifically, the resource cost is formulated as:

$$Cost_{resources} = \frac{\sum_{i \in \text{ active nodes}} C_{op}(O_i)}{N_C \times N_R \times \mathrm{Max}(C_{op})} \tag{5}$$

where $C_{op}(O_i)$ represents the individual cost associated with each operation ($O_i$) performed by an active node. Here, $\mathrm{Max}(C_{op})$ represents the cost of the most expensive operator, and the denominator accounts for the worst-case scenario where every grid cell contains the most resource-intensive operator. This ensures that the resource cost is always within a normalized range. The assigned cost to each operation ($O_i$) reflects the implementation complexity, hardware footprint, and energy consumption of the operation, ensuring that designs relying on computationally expensive operations are appropriately penalized. For example, a bitwise shift operation requires a minimal area and power consumption and is assigned a low cost, whereas a modular addition or multiplication involves multi-stage arithmetic with increased logic depth and power consumption, resulting in a higher penalty. This formulation encourages the evolutionary process to favor resource-efficient designs, guiding CGP toward optimized RNS reverse converters that balance computational performance with hardware constraints.

The total cost integrates the mentioned three criteria using adjustable weights to balance accuracy, latency, and resource efficiency:

$$Cost_{total} = w_c.Cost_{correctness} + w_l.Cost_{latency} + w_r.Cost_{resources} \tag{6}$$

where $w_c$, $w_l$, and $w_r$ are weight parameters that determine the relative importance of each objective. The correctness weight ($w_c$) is assigned a significantly higher value compared to the latency ($w_l$) and resource usage ($w_r$) weights to ensure accuracy remains the dominant factor during optimization. By adjusting these weights, CGP can explore different trade-offs between performance and hardware efficiency, leading to an optimized RNS reverse converter that meets specific design constraints. So, Domain experts can tune these weights based on application goals, such as prioritizing resource usage for low-power systems or latency for time-critical tasks.

The evolutionary process for designing an RNS reverse converter begins with the initialization of hyperparameters for hybrid CGP-SA algorithm. The CGP framework is defined by the following parameters: population size ($N_{pop}$), the number of generations ($G$), mutation rate ($M_{rate}$), tournament selection size ($TS$), grid dimensions including the number of rows ($N_R$) and columns ($N_C$), and the "levels back" parameter ($L_b$) that determines how far back in the grid a node can connect. The algorithm also defines cost function weights for correctness ($w_c$), latency ($w_l$), and resource efficiency ($w_r$).

The process starts by generating a random population of CGP candidates, each representing a possible architecture for the reverse converter. These candidates are immediately evaluated using the cost function, ensuring that correctness, latency, and resource efficiency are considered. Once the initial population is assessed, the evolutionary loop begins, iterating over multiple generations to refine the design. In each generation, a tournament selection method is applied, where a subset of candidates competes, and the one with the lowest cost is chosen as a parent. These selected parents undergo mutation, which

modifies the structure and connections within the CGP grid. The resulting offspring are then evaluated and added to a candidate pool alongside their parents to execute an evolutionary process using the $\mu+\lambda$ strategy. Hence, the next generation is formed by selecting the best candidates from the pool based on their cost.

In the proposed algorithm, a fraction of the next generation ($F_{SA}$) is designated for refinement using SA. This fraction represents the top-performing candidates from each generation, selected to undergo further optimization through SA. The SA process is controlled by an initial temperature ($T_0$), a cooling rate ($\beta$), a minimum temperature ($T_{min}$), and the number of iterations per SA run ($L$). SA plays a crucial role in optimizing function node parameters by actively searching for the correct parameters that will allow the entire CGP grid to generate the expected outputs for the RNS reverse converter. At each iteration, SA modifies function node parameters and evaluates whether the changes improve the overall correctness of the system. If the new configuration yields a lower cost, it is immediately accepted. Otherwise, the modification may still be accepted based on a probabilistic function that allows exploration of potential improvements. The probability of accepting a suboptimal solution is controlled by the temperature parameter ($T$), which gradually decreases throughout the process. As $T$ decreases, the probability of accepting worse solutions diminishes, allowing the algorithm to converge toward an optimal configuration.

After all generations are processed, the best-performing candidate from the final population is selected as the optimal RNS reverse converter architecture. By combining CGP's structural evolution with SA's targeted search for function node parameters, this hybrid approach ensures an efficient and accurate design that balances correctness, latency, and resource utilization. The output of this phase is a high-level architecture of the RNS reverse converter which serves as the foundation for Phase 2, where it will be refined into a gate-level implementation. By prioritizing correctness and balancing other objectives, Phase 1 ensures a robust and efficient design. The pseudo-code of this Phase is shown in Algorithm 1.

**Algorithm 1.** Module-Level RNS Reverse Converter Design Using Hybrid CGP_SA

**Inputs:**
Moduli set: $\{m_1, m_2, \ldots, m_k\}$
Initialize CGP parameters: ($N_{pop}$, $N_R$, $N_C$, $L_b$, $G$, $M_{rate}$, $TS$)
Initialize SA parameters: ($F_{SA}$, $T_0$, $\beta$, $T_{min}$, $L$)
Set cost function weights: ($w_c$, $w_l$, $w_r$)
**Outputs:**
Best candidate solution for the module-level RNS reverse converter
**Begin:**
***1. Define primitives:***
Arithmetic operations: addition, subtraction, modular addition
Bitwise operations: circular shifts, logical shifts, complement, concatenation
***2. Generate the truth table:***
Compute all possible residue combinations for the moduli set $\{m_1, m_2, \ldots, m_k\}$
For each residue combination:
compute the corresponding integer using *CRT*
store residues and their corresponding integer in the truth table
***3. Generate initial population:***
population ← InitializePopulation($N_{pop}$, $N_R$, $N_C$, $L_b$)
***4. Evaluate initial population:***
For each candidate in population:
candidate.total_cost ← EvaluateCandidateCost(candidate, Truth Table, $w_c$, $w_l$, $w_r$)
***5. Perform evolutionary optimization loop:***
For generation = 1 to $G$:
selected_parents ← TournamentSelection(population, $TS$)
offspring ← MutatePopulation(selected_parents, $M_{rate}$)
***6. Evaluate offspring***
For each child in offspring:
child.total_cost ← EvaluateCandidateCost(child, Truth Table, $w_c$, $w_l$, $w_r$)
7. ***Form candidate pool and select next generation***
candidate_pool ← Merge(parents, offspring)
candidate_pool ← SortByCost(candidate_pool) // ***Sort to prioritize best candidates***
population ← SelectNextGeneration(candidate_pool, $N_{pop}$) // ***Select top $N_{pop}$ candidates***
8. ***Apply SA refinement to a fraction of top candidates***
top_fraction ← SelectTopFraction(population, $F_{SA}$)
For each candidate in top_fraction:

(*continued on next column*)

(*continued*)

**Algorithm 1.** Module-Level RNS Reverse Converter Design Using Hybrid CGP_SA

candidate ← SimulatedAnnealing(candidate, $T_0$, $\beta$, $T_{min}$, $L$)
***9. Return the best candidate***
best_solution ← GetBestCandidate(population)
Return best_solution
***10. Function for cost evaluation:***
Function EvaluateCandidateCost(candidate, Truth Table, $w_c$, $w_l$, $w_r$)
correctness_cost ← ComputeCorrectnessCost(candidate, Truth Table)
latency_cost ← CountActiveColumns(candidate)
resource_cost ← ComputeResourceCost(candidate)
total_cost ← ($w_c$ × correctness_cost) + ($w_l$ × latency_cost) + ($w_r$ × resource_cost)
Return total_cost
***11. Function for simulated annealing:***
Function SimulatedAnnealing(candidate, $T_0$, $\beta$, $T_{min}$, $L$)
$T \leftarrow T_0$
best_solution ← candidate
best_cost ← EvaluateCandidateCost(candidate)
While ($T > T_{min}$ AND iteration < $L$):
neighbor ← GenerateNeighbor(candidate)
neighbor_cost ← EvaluateCandidateCost(neighbor)
If neighbor_cost < best_cost:
best_solution ← neighbor
Else:
acceptance_probability ← exp(−(neighbor_cost − best_cost) / $T$)
If Random(0, 1) < acceptance_probability:
best_solution ← neighbor
$T \leftarrow T * \beta$
Return best_solution

### 4.2. Phase 2: Gate-Level optimization

The Gate-Level Optimization Phase aims to derive an efficient, optimized gate-level implementation of an RNS reverse converter using CGP. As in the first phase, this stage seeks to minimize hardware resource usage and latency while ensuring output correctness and generating designs suitable for practical hardware implementation. To achieve these goals, the optimization process uses evolutionary techniques to explore a vast design space while dynamically adapting to constraints and performance requirements.

The optimization process begins with the initialization of the genome structure, which defines the layout and parameters of the Cartesian grid. In this phase, the grid consists of rows and columns, where each node represents a logic gate. The number of inputs corresponds to the RNS residues or intermediate signals, while the outputs map to the binary representation of the converter's result. Additionally, RCGP is employed in this phase to enable CGP to optimize RNS gate-level circuits by efficiently integrating EAC feedback loops, which are essential for modular arithmetic operations. Also, the grid's flexibility is improved through the "levels back" parameter, which controls node connectivity depth.

The set of allowed primitives defines the types of gates used in the design, including basic logic gates such as NOT, AND, OR, and XOR, as well as higher-order components like multiplexers (MUX) and arithmetic gates such as SUM and CARRY. Notably, in Phase 2, primitives consist exclusively of logic gates, which operate with fixed Boolean logic and require no adjustable parameters. This contrasts with the high-level primitives in Phase 1, where the SA algorithm was used for parameter tuning. As a result, SA is unnecessary in Phase 2, and this phase solely relies on the CGP-based evolutionary process to optimize logic gate arrangements for an efficient RNS reverse converter. Following this, A population of candidate solutions is initialized, where each individual represents a potential gate-level design encoded as a genome. The genome describes the connections, types of gates, and overall structure of the design.

As mentioned before, the cost evaluation plays a central role in guiding the evolutionary process. Hence, in this phase, again each individual is evaluated against three key metrics: correctness cost, latency, and resource usage. Just like in Phase 1, correctness cost is evaluated by comparing the candidate design's output with the expected output

defined in the truth table of the RNS reverse converter. For every input in the truth table, the candidate's output is checked against the expected result, and the correctness cost is computed as the number of mismatches between the candidate's output and the expected result. Minimizing this cost ensures that the design achieves the desired functionality. Also, latency cost is determined by measuring the critical path length, represented as the number of active columns in the Cartesian grid.

Beyond correctness and latency, resource cost is calculated based on the active gates in the design, where each gate type has an associated cost reflecting its power, delay and area (PDA). Each primitive is characterized by its Boolean functionality and associated costs, such as area and power consumption, which are considered during optimization. For example, simple gates like AND and OR have lower associated costs due to their minimal PDA. In contrast, more complex gates like MUX, or SUM are assigned significantly higher costs because they consume more resources and involve greater design complexity. The resource usage for a design is computed as the sum of the costs for all active gates in the CGP graph. This resource usage calculation serves as a penalty in the composite cost function, discouraging overly complex designs. By assigning higher penalties to designs that utilize resource-intensive gates, the optimization process is biased towards solutions that achieve the required functionality with simpler, more efficient gates. This balance is essential in ensuring that the final design is not only functionally correct but also optimized for real-world constraints such as hardware implementation and energy consumption.

Furthermore, once again, correctness, latency, and resource costs are combined, as shown in Equation (6), with adjustable weights to reflect their relative importance. Also, correctness cost remains the top priority, with a substantially higher weight than latency and resource costs. This prioritization ensures that the designs satisfy functional requirements while also being optimized for latency and resource efficiency. This adjustment is critical for producing designs that are not only accurate but also practical for real-world constraints, such as hardware implementation and energy efficiency.

Similar to the Phase 1, the evolutionary process employs a $\mu+\lambda$ strategy, where the population undergoes mutation and reproduction. Mutation introduces variations by altering gate connections, types, or input combinations, allowing the algorithm to explore new design possibilities. Since the CGP grids in this phase are larger than those in Phase 1, the evolutionary process requires careful control over the mutation rate to ensure an effective search strategy and balance exploration and exploitation. Larger grids introduce more complexity, increasing the number of potential connections and functional nodes, which can lead to a vast search space. To efficiently navigate this space while avoiding excessive randomness or premature convergence, the mutation rate is dynamically adjusted over generations. This adjustment follows an exponential decay based on the generation number, as expressed in Equation (7).

$$M_{rate}(g) = M_{rate}^{init} \cdot e^{-\alpha \cdot g}, \tag{7}$$

Where, the variable $g$ denotes the current generation number, and the $M_{rate}^{init}$ represents the starting value for the mutation rate at the beginning of the optimization process. The parameter $\alpha$ is the decay rate, a positive constant that controls the rate at which the mutation rate decreases over time. This approach ensures that the mutation rate is higher during the early stages of the algorithm, encouraging greater diversity and exploration across the solution space. As the optimization progresses, the mutation rate gradually decreases, emphasizing exploitation of the most promising solutions discovered so far. To prevent the mutation rate from diminishing excessively and causing stagnation in the search process, a minimum threshold is imposed. The mutation rate at any generation is given by:

$$M_{rate} = max(M_{rate}, M_{rate}^{\min}), \tag{8}$$

Where, the $M_{rate}^{\min}$ is a predefined lower bound that maintains a baseline level of diversity, ensuring that the algorithm does not entirely lose the ability to explore new solutions. By dynamically adapting the mutation rate in this manner, the algorithm effectively balances the need for exploration in the early stages with the precision required for fine-tuning solutions in later generations. This dynamic adjustment enhances the algorithm's ability to discover and converge on optimal designs efficiently.

Also, the tournament size is dynamically adjusted during the evolutionary process, starting with a smaller size in the early generations to promote diversity and gradually increasing in later generations to refine the best solutions. The dynamic adjustment of tournament size *(TS)* follows a stepwise strategy defined as:

$$TS(g) = \begin{cases} TS_{init} \text{if } g \leqslant g_1, \\ TS_1 \text{if } g_1 < g \leqslant g_2, \\ TS_2 \text{if } g_2 < g \leqslant g_3, \\ TS_{max} \text{if } g > g_3, \end{cases} \tag{9}$$

where $g$ represents the current generation, $TS_{init}$ is the initial tournament size, $TS_{\max}$ is the maximum tournament size, and $g_1$,$g_2$, and $g_3$ are generation thresholds that control the stepwise increase of the tournament size. This adaptive mechanism encourages exploration in the early stages and focuses on exploiting promising solutions as the algorithm progresses, balancing exploration and exploitation effectively.

As the optimization progresses, the cost of each individual is evaluated, and the design with the lowest cost is identified as the champion. The optimization process terminates when the maximum number of generations is reached. The final output includes the optimized genome of the champion individual, a symbolic Boolean representation of the circuit, and graphical visualizations of the design. The pseudo-code of this Phase is shown in Algorithm 2.

**Algorithm 2.** Gate-Level RNS Reverse Converter Optimization Using CGP

**Inputs:**
Initialize CGP parameters: ($N_{pop}$, $N_R$, $N_C$, $L_b$, $G$, $M_{rate}^{init}$, $M_{rate}^{min}$, $\alpha TS_{init}$, $TS_1$, $TS_2$, $TS_{max}$, $g_1$, $g_2$, $g_3$)
Set cost function weights: ($w_c$, $w_l$, $w_r$)
**Outputs:**
Best candidate solution for the gate-level RNS reverse converter
**Begin:**
***1. Define primitives:***
Logic gates: AND, OR, NOT, XOR, MUX, SUM, CARRY
***2. Generate the binary truth table:***
Generate the Truth Table for the RNS reverse converter
Convert the inputs and outputs to binary numbers
***3. Generate initial population:***
population ← InitializePopulation($N_{pop}$, $N_R$, $N_C$, $L_b$)
***4. Evaluate initial population:***
For each candidate in population:
candidate.total_cost ← EvaluateCandidateCost(candidate, Truth Table, $w_c$, $w_l$, $w_r$)
***5. Perform evolutionary optimization loop:***
For generation = 1 to $G$:
selected_parents ← TournamentSelection(population, $TS$)
offspring ← MutatePopulation(selected_parents, $M_{rate}$)
***6. Evaluate offspring***
For each child in offspring:
child.total_cost ← EvaluateCandidateCost(child, Truth Table, $w_c$, $w_l$, $w_r$)
***7. Form candidate pool and select next generation***
candidate_pool ← Merge(parents, offspring)
candidate_pool ← SortByCost(candidate_pool) // ***Sort to prioritize best candidates***
population ← SelectNextGeneration(candidate_pool, $N_{pop}$) // ***Select top $N_{pop}$ candidates***
***8. Dynamically adjust mutation rate***
$M_{rate}$ ← max($M_{rate}^{init}$ × exp(−$\alpha$ × generation), $M_{rate}^{min}$)
***9. Dynamically adjust tournament size:***
If generation ≤ $g_1$:
$TS$ ← $TS_{init}$
Else If $g_1$ < generation ≤ $g_2$:
$TS$ ← $TS_1$
Else If $g_2$ < generation ≤ $g_3$:
$TS$ ← $TS_2$

(continued on next page)

(*continued*)

**Algorithm 2.** Gate-Level RNS Reverse Converter Optimization Using CGP

Else:
$TS \leftarrow TS_{max}$
***7. Return the best candidate***
best_solution ← GetBestCandidate(population)
Return best_solution
***8. Function for cost evaluation:***
Function EvaluateCandidateCost(candidate, Truth Table, $w_c$, $w_l$, $w_r$)
correctness_cost ← ComputeCorrectnessCost(candidate, Truth Table)
latency_cost ← CountActiveColumns(candidate)
resource_cost ← ComputeResourceCost(candidate)
total_cost ← ($w_c$ × correctness_cost) + ($w_l$ × latency_cost) + ($w_r$ × resource_cost)
Return total_cost

Overall, the Gate-Level Optimization Phase employs CGP to evolve compact, efficient, and correct designs for RNS reverse converters. By prioritizing correctness and systematically refining solutions, the process delivers practical gate-level implementations that meet hardware constraints and performance requirements.

## 5. Experimental results

This section presents the empirical results obtained by applying the proposed hybrid framework for designing optimized RNS reverse converters. The evaluation begins with a detailed specification of the hyperparameter settings used during both phases of the optimization process. Proper tuning of these parameters is essential for ensuring efficient exploration of the solution space and achieving high-quality convergence in both structural evolution and parameter refinement. Subsequently, a comprehensive analysis of the hybrid CGP-SA algorithm is presented. This includes quantitative comparisons between the CGP-only and CGP-SA configurations to analyze the contributions of SA in refining parameter-sensitive operations. The evaluation also investigates the effects of adaptive mutation rates and dynamic tournament selection in Phase 2, emphasizing their roles in improving convergence, reducing solution variability, and enhancing scalability in gate-level optimization. Finally, a comparative performance analysis is conducted to benchmark the proposed CGP-based reverse converters against state-of-the-art manually designed counterparts reported in the literature. The evaluation criteria include critical design metrics including the number of gates, as well as overall conversion delay and circuit area, both reported in equivalent unit-gate counts. The results confirm that the proposed framework consistently achieves more compact designs with comparable or better performance, validating its potential for efficient and scalable RNS converter synthesis. To further illustrate the practical application and effectiveness of the proposed methodology, we conduct a case study using the commonly cited moduli set {4,7,9} in Appendix C. This example demonstrates how the CGP-SA framework automatically evolves an efficient and simplified architecture for reverse conversion. The case study highlights the ability of the framework to derive functionally correct and resource- efficient designs without manual intervention. All experiments were implemented using Python and executed on a Windows-based platform equipped with an AMD Ryzen 9 7940 CPU and 64 GB of RAM. The reported results are based on multiple independent runs to ensure statistical reliability and robustness of the findings.

### 5.1. Hyperparameter settings for the proposed approach

Table 3 outlines the empirically defined hyperparameters used in the two-phase optimization framework for RNS reverse converters. As previously described, Phase 1 applies a hybrid CGP-SA method for module-level design, whereas Phase 2 shifts focus to gate-level optimization using adaptive CGP. Each phase utilizes distinct parameter settings aligned with its specific optimization objectives.

The population size is set to 50 in Phase 1 and doubled to 100 in Phase 2 to better handle the increased complexity inherent to gate-level design. Similarly, the number of generations is 200 in Phase 1 and extended to 400 in Phase 2 to facilitate more thorough exploration. The CGP grid dimensions (including the number of rows, columns, and the levels-back parameter) are left problem-dependent, as they vary according to the moduli set being optimized.

**Table 3**
Hyperparameters of the proposed approach.

| Parameter | Description | Phase 1 Value | Phase 2 Value |
|---|---|---|---|
| $N_{pop}$ | Population size | 50 | 100 |
| $N_R$ | Number of rows in the CGP grid | Problem dependent | Problem dependent |
| $N_C$ | Number of columns in the CGP grid | Problem dependent | Problem dependent |
| $L_b$ | Levels back parameter (controls node connectivity depth) | $N_C$ | $N_C$ |
| $G$ | Number of generations | 200 | 400 |
| $F_{SA}$ | Fraction of top candidates refined via SA (Phase 1 only) | 0.1 | N/A |
| $T_0$ | Initial temperature for SA (Phase 1 only) | 0.1 | N/A |
| $\beta$ | SA cooling rate (Phase 1 only) | 0.95 | N/A |
| $T_{min}$ | Minimum temperature for SA (Phase 1 only) | 0.01 | N/A |
| $L$ | Number of iterations for SA refinement (Phase 1 only) | 50 | N/A |
| $w_c$ | Weight for correctness cost | 0.9 | 0.9 |
| $w_l$ | Weight for latency cost | 0.05 | 0.05 |
| $w_r$ | Weight for resource usage cost | 0.05 | 0.05 |
| $M_{rate}/M_{rate}^{init}$ | Initial mutation rate | 0.05 | 0.1 |
| $M_{rate}^{min}$ | Minimum mutation rate (ensures lower bound; Phase 2 only) | N/A | 0.01 |
| $\alpha$ | Decay factor for mutation rate adaptation (Phase 2 only) | N/A | 0.005 |
| $TS/TS_{init}$ | Tournament size (initial selection pressure) | 10 | 10 |
| $TS_1$ | Tournament size for intermediate generations (g1 < generation ≤ g2; Phase 2 only) | N/A | 20 |
| $TS_2$ | Tournament size for later generations (g2 < generation ≤ g3; Phase 2 only) | N/A | 50 |
| $TS_{\max}$ | Maximum tournament size for final generations (Phase 2 only) | N/A | 100 |
| $g_1$ | Generation threshold for initial tournament size adjustment (Phase 2 only) | N/A | 100 |
| $g_2$ | Generation threshold for intermediate tournament size adjustment (Phase 2 only) | N/A | 200 |
| $g_3$ | Generation threshold for final tournament size adjustment (Phase 2 only) | N/A | 300 |

Mutation and selection strategies differ significantly between the two phases to reflect their respective needs. In Phase 1, a fixed mutation rate of 0.05 and a constant tournament size of 10 are maintained throughout. In contrast, Phase 2 begins with a higher initial mutation rate of 0.1, which decays gradually using a factor of 0.005 until it reaches a minimum of 0.01, promoting exploration in early generations and convergence in later stages. The tournament size in Phase 2 is also adaptive: it starts at 10, then increases to 20, 50, and 100 at generation thresholds of 100, 200, and 300 respectively, to balance selection pressure over time.

A key feature of Phase 1 is the integration of SA for fine-tuning operator-specific parameters. Specifically, 10 % of the top-performing individuals are selected for refinement using SA. The SA procedure starts with an initial temperature of 0.1, a cooling rate of 0.95, and runs for 50 iterations per candidate, terminating at a minimum temperature of 0.01. This process helps enhance solution quality by locally optimizing parameter settings. Phase 2, in contrast, does not employ SA, as it operates at the gate level using fixed Boolean operators with no tunable

parameters. Instead, its optimization relies solely on adaptive CGP mechanisms. Both phases share a unified cost function that emphasizes correctness, latency, and resource usage, with weights of 0.9, 0.05, and 0.05, respectively. This prioritization ensures that correctness remains the primary objective, while also encouraging designs that are efficient in terms of speed and hardware utilization. Together, these tailored hyperparameter settings enable the two-phase framework to first construct a robust module-level architecture and then refine it through detailed gate-level optimization.

### 5.2. Analysis of CGP and SA algorithms

To evaluate the effectiveness of the proposed optimization methodology, we analyze the contributions of SA in Phase 1 and the impact of adaptive mutation and selection mechanisms in Phase 2. This analysis provides insight into how these components improve convergence and solution quality in the design of the RNS reverse converter.

#### 5.2.1. Impact of SA in phase 1

In Phase 1 of the proposed framework, SA is integrated into CGP to refine operator-specific parameters, particularly those associated with bitwise operations such as shift amounts. This hybridization addresses a fundamental limitation in standard CGP, which struggles to efficiently optimize parameter values using mutation alone. To evaluate the contribution of SA, we compare the performance of CGP with that of the hybrid CGP-SA approach across a range of representative moduli sets. The selected moduli sets include three-moduli, four-moduli, and five-moduli configurations, all of which are commonly found in RNS arithmetic. The three-moduli sets used for evaluation include $\{2^{(n-1)}, 2^n-1, 2^n+1\}$, $\{2^n, 2^{2n}-1, 2^{2n}+1\}$, and $\{2^{(2n-1)}, 2^{(2n+1)}-1, 2^{(2n+1)}+1\}$. The four-moduli sets considered in this analysis are $\{2^n, 2^{2n+1}, 2^n+1, 2^n-1\}$, $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ and the five-moduli set used for evaluation is $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$.

Table 4 presents the mean, standard deviation, minimum, and maximum values of the cost function obtained over 30 independent runs for each moduli set and value of $n$, with all results rounded to four decimal places. As mentioned before, the cost function integrates correctness, latency, and resource consumption, where lower values indicate better performance. Across all tested moduli configurations and parameter settings, the CGP-SA approach consistently outperforms CGP, yielding lower mean costs, reduced variability, and tighter min–max bounds. These improvements clearly demonstrate the significant role of SA in guiding the local refinement of parameter-sensitive operations, thereby accelerating convergence and enhancing solution feasibility.

Building on this observation, the detailed statistics provide valuable insight into how CGP-SA enhances both performance and stability across these diverse RNS configurations. For instance, in the three-moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$, CGP-SA not only achieves significantly lower mean costs (e.g., reducing the cost from 0.2672 to 0.1517 for $n=2$) but also consistently lowers the standard deviation, indicating more reliable convergence behavior. This trend continues across larger $n$ values, highlighting that SA becomes increasingly beneficial as the design complexity increases. Similarly, for the more computationally intensive moduli set like $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$, the CGP-SA hybrid demonstrates superior performance by achieving narrower performance ranges. Notably, the five-moduli set exhibits one of the lowest standard deviations under CGP-SA, suggesting that the efficient parameter tuning provided by SA not only enhances quality but also promotes consistency in larger, more irregular search spaces. These detailed comparisons reinforce the advantage of hybridizing CGP with SA, particularly in parameter-sensitive domains such as reverse converter design.

The bar chart in Fig. 5 offers a clear visual comparison of the average mean-cost across six moduli sets (labeled MS1 to MS6) for both the CGP-only and CGP-SA hybrid approaches. Each pair of bars represents the average cost function values over different $n$ values as presented in Table 4, quantifying the overall performance of each method in terms of correctness, latency, and resource usage. In each case, the average mean-cost achieved by CGP-SA is noticeably lower, with the most significant improvements observed in MS2 and MS3 moduli sets. These reductions highlight the effectiveness of SA in fine-tuning parameter-dependent operations that CGP alone struggles to optimize efficiently. The bar chart once again clearly demonstrates that the integration of SA into CGP consistently yields more optimized and hardware-efficient reverse converter designs.

Finally, in order to gain a deeper understanding of the consistency and robustness of the proposed CGP-SA approach, a box-and-whisker plot is shown in Fig. 6, for the moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$ across different values of $n$. This visualization highlights the distribution of cost function values over 30 independent runs per case, providing insight into both performance and variance. From this figure, it is evident that the CGP-SA method consistently demonstrates lower median costs, tighter interquartile ranges, and fewer outliers, indicating more stable convergence and higher reliability. In contrast, the CGP-only approach shows broader variability, with wider whiskers and several extreme

**Table 4**
Comparative Performance of CGP and CGP-SA in Phase 1 for Various Moduli Sets.

| Moduli Set | Method | n | Mean Cost | Std Dev | Min | Max |
|---|---|---|---|---|---|---|
| $\{2^{(n-1)}, 2^n-1, 2^n+1\}$ | CGP | 2 | 0.2672 | 3.78E-2 | 0.1297 | 0.3318 |
| | | 3 | 0.2592 | 5.12E-2 | 0.1161 | 0.3151 |
| | | 4 | 0.2049 | 6.51E-2 | 0.1346 | 0.2614 |
| | | 5 | 0.1767 | 8.39E-2 | 0.1411 | 0.2328 |
| | CGP-SA | 2 | 0.1517 | 1.76E-2 | 0.1297 | 0.1964 |
| | | 3 | 0.1483 | 3.45E-2 | 0.1161 | 0.2358 |
| | | 4 | 0.1726 | 4.81E-2 | 0.1066 | 0.2107 |
| | | 5 | 0.1283 | 5.78E-2 | 0.0994 | 0.1911 |
| $\{2^n, 2^{2n}-1, 2^{2n}+1\}$ | CGP | 2 | 0.2747 | 5.11E-2 | 0.1834 | 0.3162 |
| | | 3 | 0.2413 | 6.04E-2 | 0.1572 | 0.2841 |
| | CGP-SA | 2 | 0.1659 | 2.45E-2 | 0.1250 | 0.2399 |
| | | 3 | 0.1483 | 3.16E-2 | 0.1045 | 0.2216 |
| $\{2^{(2n-1)}, 2^{(2n+1)}-1, 2^{(2n+1)}+1\}$ | CGP | 2 | 0.2514 | 6.73E-2 | 0.1605 | 0.3362 |
| | CGP-SA | 2 | 0.1468 | 2.23E-2 | 0.1248 | 0.2076 |
| $\{2^n, 2^{2n+1}-1, 2^n+1, 2^n-1\}$ | CGP | 2 | 0.2612 | 3.82E-2 | 0.1976 | 0.2970 |
| | | 3 | 0.2492 | 4.31E-2 | 0.1884 | 0.2795 |
| | CGP-SA | 2 | 0.1825 | 1.98E-2 | 0.1637 | 0.2311 |
| | | 3 | 0.1734 | 2.09E-2 | 0.1499 | 0.2120 |
| $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ | CGP | 3 | 0.2572 | 3.21E-2 | 0.2033 | 0.2867 |
| | | 4 | 0.2315 | 2.29E-2 | 0.1977 | 0.2572 |
| | CGP-SA | 3 | 0.2107 | 1.84E-2 | 0.1908 | 0.2534 |
| | | 4 | 0.1893 | 1.27E-2 | 0.1759 | 0.2163 |
| $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$ | CGP | 3 | 0.2138 | 1.53E-2 | 0.1836 | 0.2249 |
| | CGP-SA | 3 | 0.1797 | 7.64E-3 | 0.1705 | 0.1973 |

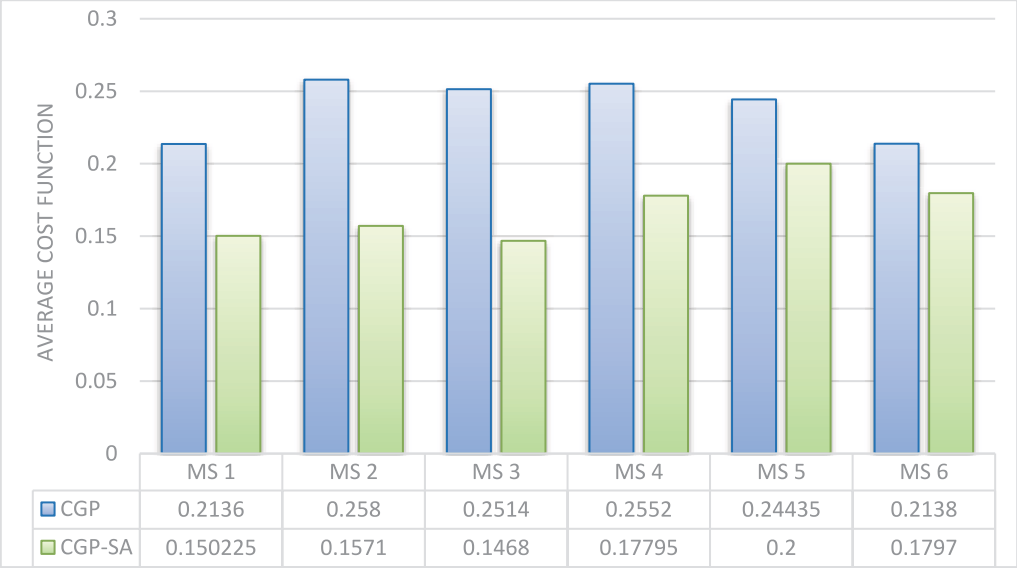


**Fig. 5.** Comparison of average mean-cost for the moduli sets in Table 4.

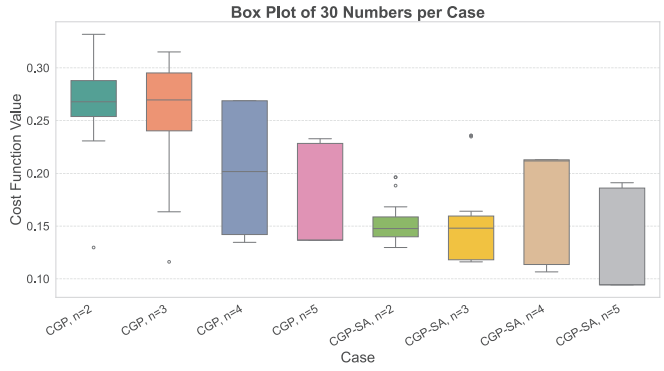


**Fig. 6.** The box-and-whisker plot of CGP and CGP-SA algorithms for the moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$.

values, particularly for especially for n = 2, n = 3, and n = 5, suggesting less predictable behavior and a higher likelihood of suboptimal solutions. These findings emphasize that the integration of SA not only improves average solution quality but also enhances the robustness and repeatability of the evolutionary design process. While CGP explores the topological space of arithmetic and bitwise operator arrangements, SA fine-tunes internal operator parameters to avoid premature convergence and local optima. This synergy leads to designs that require fewer logic gates, reduced operator depth, and improved structural regularity, which are beneficial for the downstream gate-level optimization phase.

### 5.2.2. *Effect of adaptive mutation and tournament selection in phase 2*

Phase 2 of the proposed framework addresses the gate-level optimization of RNS reverse converters. This stage involves a higher degree of structural complexity due to the detailed arrangement of logic gates, requiring a sophisticated evolutionary strategy to efficiently explore and refine the solution space. To this end, in this section we examine the impact of adaptive mutation rate control and dynamic tournament selection within CGP algorithm. Unlike fixed-parameter configurations, the adaptive strategy adjusts key evolutionary parameters over time. As mentioned above, in this phase the mutation rate begins at a higher value to encourage diversity and broad exploration, then gradually decreases to support focused refinement in later generations. Simultaneously, the tournament size used in selection increases progressively, intensifying selection pressure while preserving sufficient variation in the population. The effectiveness of the suggested adaptive approach is evaluated using the same six moduli sets introduced in the previous section ($\{2^{(n-1)}, 2^n-1, 2^n+1\}$, $\{2^n, 2^{2n}-1, 2^{2n}+1\}$, $\{2^{(2n-1)}, 2^{(2n+1)}-1, 2^{(2n+1)}+1\}$, $\{2^n, 2^{2n+1}-1, 2^n+1, 2^n-1\}$, $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$, and $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$), each selected to represent a different level of arithmetic complexity and dynamic range. The results, summarized in Table 5, are based on 30 independent runs per configuration and evaluated using four key performance metrics: mean cost, standard deviation, minimum, and maximum values of the cost function.

Across all tested configurations, the adaptive strategy consistently outperforms the fixed-parameter approach, delivering lower mean costs, reduced variability, and narrower performance bounds. These improvements reflect both enhanced optimization efficiency and more reliable convergence behavior. For example, in the three-moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$, the mean cost for $n = 2$ decreases from 0.3428 under fixed parameters to 0.3204 with the adaptive strategy, accompanied by zero standard deviation, indicating perfectly stable convergence across all runs. A similar trend is observed in the moduli set $\{2^n, 2^{2n}-1, 2^{2n}+1\}$, where the mean cost at $n = 3$ drops from 0.3973 to 0.3625, along with a noticeable reduction in variance. Improvements are also evident in more complex configurations. In the four-moduli set $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$, the adaptive strategy reduces the mean cost from 0.5883 to 0.5561 for $n = 4$, while producing a tighter distribution of results. The benefits are even more pronounced in the five-moduli set $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$, where adaptive CGP achieves a significantly lower mean cost of 0.3056, compared to 0.3839 using fixed parameters. These results confirm that dynamically adapting evolutionary parameters during the optimization process improves both the efficiency and consistency of gate-level circuit design.

Fig. 7 provides a comparative overview of the average mean cost for six different moduli sets (MS1–MS6), showcasing the performance difference between the fixed-parameter CGP and the adaptive CGP strategies. Each bar pair illustrates the average cost values aggregated across multiple values of $n$, as detailed in Table 5. The chart clearly shows that adaptive CGP consistently achieves lower average costs across all moduli sets. The performance gap is especially notable in MS2 and MS3, where adaptive CGP yields the most substantial reductions, underscoring its ability to more effectively navigate parameter-sensitive design challenges. These results reaffirm that incorporating adaptive mechanisms into CGP enhances its capability to generate more efficient and optimized gate-level reverse converter architectures.

**Table 5**
Comparative Performance of CGP with Fixed and Adaptive Parameters in Phase 2.

| Moduli Set | Strategy | n | Mean Cost | Std Dev | Min | Max |
|---|---|---|---|---|---|---|
| $\{2^{(n-1)}, 2^n-1, 2^n+1\}$ | Fixed Parameters | 2 | 0.3428 | 9.62E-3 | 0.3204 | 0.3561 |
| | | 3 | 0.3573 | 1.37E-2 | 0.3321 | 0.3627 |
| | | 4 | 0.3622 | 1.65E-2 | 0.3385 | 0.3814 |
| | | 5 | 0.3858 | 2.08E-2 | 0.3495 | 0.4035 |
| | Adaptive Parameters | 2 | 0.3204 | 0 | 0.3204 | 0.3204 |
| | | 3 | 0.3362 | 2.58E-3 | 0.3321 | 0.3428 |
| | | 4 | 0.3416 | 3.86E-3 | 0.3385 | 0.3528 |
| | | 5 | 0.3497 | 7.57E-3 | 0.3424 | 0.3704 |
| $\{2^n, 2^{2n}-1, 2^{2n}+1\}$ | Fixed Parameters | 2 | 0.3789 | 1.66E-2 | 0.3476 | 0.3953 |
| | | 3 | 0.3973 | 1.89E-2 | 0.3674 | 0.4176 |
| | Adaptive Parameters | 2 | 0.3512 | 7.63E-3 | 0.3476 | 0.3645 |
| | | 3 | 0.3625 | 1.07E-2 | 0.3588 | 0.3812 |
| $\{2^{(2n-1)}, 2^{(2n+1)}-1, 2^{(2n+1)}+1\}$ | Fixed Parameters | 2 | 0.3926 | 1.85E-2 | 0.3573 | 0.4075 |
| | Adaptive Parameters | 2 | 0.3508 | 9.74E-3 | 0.3418 | 0.3759 |
| $\{2^n, 2^{2n+1}-1, 2^n+1, 2^n-1\}$ | Fixed Parameters | 2 | 0.2695 | 3.49E-2 | 0.2197 | 0.3216 |
| | | 3 | 0.2852 | 4.08E-2 | 0.2371 | 0.3460 |
| | Adaptive Parameters | 2 | 0.2363 | 2.10E-2 | 0.2086 | 0.2843 |
| | | 3 | 0.2517 | 2.47E-2 | 0.2229 | 0.3078 |
| $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ | Fixed Parameters | 3 | 0.5224 | 2.76E-2 | 0.4962 | 0.5731 |
| | | 4 | 0.5883 | 3.01E-2 | 0.5538 | 0.6319 |
| | Adaptive Parameters | 3 | 0.4937 | 1.25E-2 | 0.4822 | 0.5274 |
| | | 4 | 0.5561 | 1.48E-2 | 0.5417 | 0.5926 |
| $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$ | Fixed Parameters | 3 | 0.3839 | 3.37E-2 | 0.3269 | 0.4180 |
| | Adaptive Parameters | 3 | 0.3056 | 1.78E-2 | 0.2892 | 0.3533 |

To further assess the stability and reliability of the optimization process, Fig. 8 presents a box-and-whisker plot comparing the cost function distributions of the fixed-parameter CGP and the adaptive CGP approaches across values of $n = 2$ to $n = 5$, for the moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$. Each case summarizes results from 30 independent runs, offering insight into both central tendency and variability. The plot clearly illustrates that the adaptive CGP strategy yields lower median cost values, narrower interquartile ranges, and fewer outliers across all $n$ values, reflecting more predictable and consistent performance. In contrast, the fixed-parameter CGP method exhibits greater dispersion, with broader boxes and longer whiskers, and is more susceptible to outliers (especially evident in the $n = 2$ and $n = 3$ cases) indicating higher variance and reduced convergence reliability. Most notably, for $n = 2$, the adaptive method exhibits zero standard deviation, with all 30 runs converging to the same cost value, clearly indicating highly stable and deterministic convergence.

The results of this section confirm that the proposed adaptive mechanisms in CGP enable broad exploration in early generations and precise refinement in later stages. As circuit complexity increases with larger values of $n$, the performance gap between fixed and adaptive approaches becomes more pronounced, underscoring the scalability and effectiveness of adaptive CGP in optimizing complex RNS reverse converter architectures. Moreover, these findings demonstrate that adaptive parameter control not only enhances average optimization performance but also significantly improves the robustness and consistency of the evolutionary design process.

### 5.3. Comparative performance analysis against established designs

To further assess the effectiveness of the proposed two-phase CGP-based reverse converter design approach, we conducted a comparative analysis with several well-established RNS converter designs in the literature. This evaluation focuses on both hardware complexity and conversion delay across various moduli sets. Table 6 presents a detailed comparison between the proposed CGP-based implementations and handcrafted designs, considering key hardware metrics such as the number of inverters (INV), AND, OR, XOR, and XNOR gates, as well as the overall conversion delay and area, both expressed in equivalent unit-gate counts.

The results clearly demonstrate that the CGP-SA framework consistently yields more efficient designs across all tested moduli sets. In particular, the optimized converters show a notable reduction in AND and XOR gate usage (components typically associated with area and power overhead) while maintaining the same counts for OR, XNOR, and inverter gates. Importantly, these hardware reductions are achieved without increasing propagation delay, ensuring that performance in terms of speed is fully preserved.

For example, in the widely studied three-moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$, the proposed design for $n = 2$ reduces the number of AND gates from 15 to 16 and XOR gates from 15 to 13, resulting in an overall area reduction from 56 to 53 unit-gates compared to the design reported in (Hiasat & Sweidan, 2003). While the number of inverters and OR gates remain consistent, the optimized logic structure leads to a more compact implementation. Similar patterns are evident for larger values of $n$, where the CGP-SA designs systematically achieve lower area costs (e.g., 134 to 131 for $n = 5$) without any trade-off in delay. This trend is consistent across other moduli sets cited from references (Hariri et al., 2008), (Molahosseini et al., 2008), (Molahosseini et al., 2009), (Mojahed et al., 2021a), and (Ahmadifar & Torabi, 2024). In complex configurations such as the four-moduli set $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ (Mojahed et al., 2021a), the CGP-SA design reduces the total area from 514 to 472 for $n = 3$, and from 770 to 728 for $n = 4$, while keeping the delay unchanged. Likewise, in the five-moduli configuration $\{2^n, 2^n-1, 2^{n+1}-1, 2^{n+1}+1, 2^{n+2}-1\}$ (Ahmadifar & Torabi, 2024), the proposed method reduces the area from 682 to 668, once again without increasing delay.

These consistent improvements are a direct result of the proposed framework's ability to automatically and intelligently explore the design space, selectively evolving only those architectures that offer the best trade-off between functional correctness, hardware efficiency, and speed. By decoupling structural evolution from parameter fine-tuning, the framework is capable of minimizing redundant logic, avoiding excessive gate usage, and promoting highly regular, low-latency architectures. From a practical perspective, the observed reductions in hardware complexity translate into several tangible benefits, including smaller silicon footprint, lower power consumption, shorter design and verification cycles, and reduced fabrication costs. These advantages position the CGP-generated converters as highly suitable for performance-critical applications, particularly in domains such as embedded systems, digital signal processing, cryptographic computing, and high-speed arithmetic units, where efficiency, scalability, and reliability are paramount.

Eventually, we analyze how CGP–SA designs scale with increasing Dynamic Range (DR) by plotting total circuit area against the number of representable bits (i.e. $\log_2$ of the product of the moduli). Fig. 9

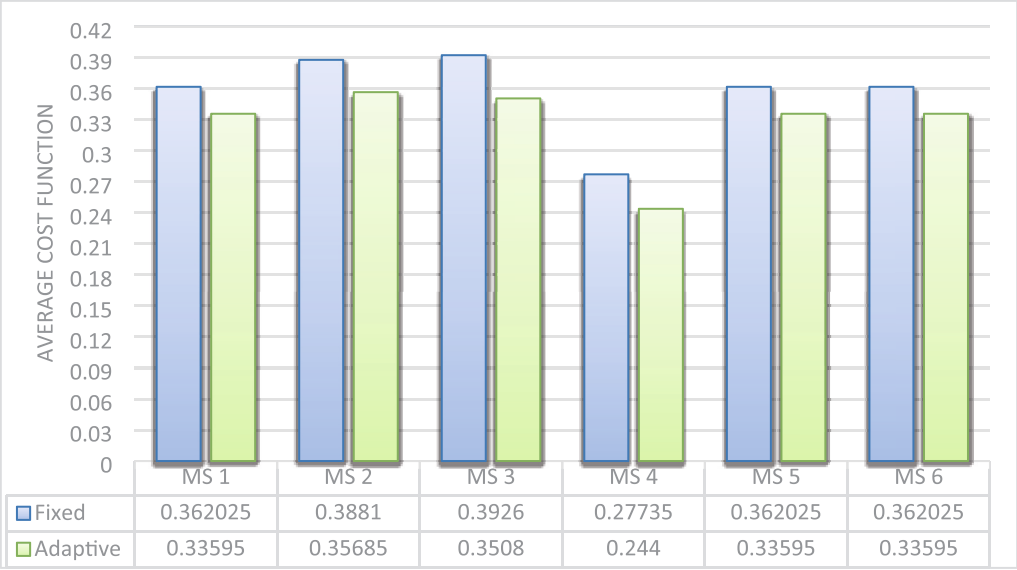


**Fig. 7.** Comparison of average mean-cost for the moduli sets in Table 5.

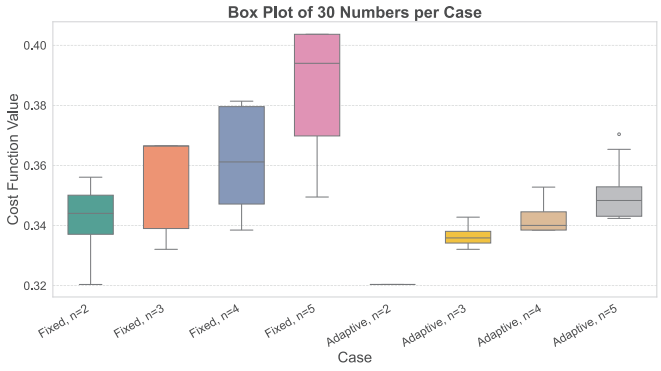


**Fig. 8.** The box-and-whisker plot of fixed and adaptipe mechanisms of CGP for the moduli set $\{2^{(n-1)}, 2^n-1, 2^n+1\}$.

illustrates this relationship, where each pair of adjacent bars represents a specific moduli configuration. The left bar in each pair corresponds to the reference design, while the right bar shows the area of the corresponding CGP-SA design, with both areas measured in unit gates. This visualization enables a direct comparison of area efficiency across a wide range of dynamic ranges. The chart clearly illustrates that the CGP-SA approach consistently achieves a lower hardware area than the reference methods across all evaluated dynamic ranges. The most substantial absolute reductions are observed in higher-range configurations, particularly at 12, 14, and 15.94 bits, where the CGP-SA design reduces the area by 42 gates or more compared to the reference implementations. All values are labeled above each bar, facilitating direct and precise comparison. For instance, at a dynamic range of 4.91 bits, the area decreases from 56 to 53 unit gates. At 14 bits, the area drops from 770 to 728 gates. At the highest tested dynamic range of 15.94 bits, the CGP-SA design achieves an area of 668 gates, compared to 682 in the reference design. These results reinforce the observation that CGP-SA not only produces compact architectures for smaller dynamic ranges but also scales more efficiently as the range increases, offering consistent benefits in hardware resource optimization.

The comparative results provide strong evidence of the practical benefits and design efficiency offered by the proposed CGP-SA framework. Across a variety of moduli sets and values of $n$, the framework consistently generates reverse converter architectures with lower area requirements and comparable or identical delay relative to conventional handcrafted designs. While the improvements in individual gate counts may appear incremental, their cumulative effect contributes to more compact and resource-conscious hardware implementations, which is especially valuable in constrained or performance-sensitive environments. These outcomes highlight the CGP-SA framework's potential as a reliable and automated design methodology for RNS reverse converters, offering a promising alternative to traditional manual approaches. Its ability to balance structural evolution with parameter fine-tuning makes it particularly well-suited for scalable, modular arithmetic systems in modern digital hardware design.

## 6. Conclusion

This paper presented an automatic approach for RNS reverse converter circuit generation and simplification using a hybrid evolutionary framework that combines CGP with SA. The proposed two-phase optimization targets both module-level and gate-level design. In the first phase, CGP explores high-level arithmetic and bitwise operator structures, while SA refines parameter-sensitive operations. In the second phase, the architecture is translated into a gate-level circuit, which is further optimized using adaptive CGP with dynamic mutation and selection control.

The effectiveness of the CGP-SA framework was validated through a series of simulations and comparative experiments. Performance comparisons between CGP and CGP-SA highlighted the role of SA in improving cost values during module-level synthesis, while box-and-whisker plots confirmed more stable convergence. In the gate-level phase, adaptive strategies enhanced both average performance and consistency compared to fixed-parameter configurations. Further comparisons with handcrafted designs showed that CGP-SA often achieves modest area reductions while maintaining similar delay. Although individual gate savings are typically incremental, they contribute to more compact and resource-conscious implementations. These results were consistent across different moduli sets and values of n, supporting the suitability of CGP-SA for a wide range of RNS configurations. Its ability to separate structural evolution from parameter tuning enables systematic design space exploration, providing a practical and automated alternative to manual circuit synthesis.

Finally, future research could explore scalability to higher-dimensional moduli sets, ensuring the CGP-SA framework remains efficient for large-scale RNS applications. Another promising direction is to investigate inter-stage knowledge transfer, whereby module-level

**Table 6**
Performance Comparison of The Proposed CGP-Based Reverse Converter Design Approach Against Existing Prominent Methods.

| Moduli Set | Ref | n | INV | AND | OR | XOR | XNOR | Delay (Unit Gate) | Area (Unit Gate) |
|---|---|---|---|---|---|---|---|---|---|
| $\{2^{(n-1)}, 2^n-1, 2^n+1\}$ | (Hiasat & Sweidan, 2003) | 2 | 4 | 15 | 7 | 15 | 0 | 37 | 56 |
| | | 3 | 6 | 22 | 10 | 22 | 0 | 53 | 82 |
| | | 4 | 8 | 29 | 13 | 29 | 0 | 69 | 108 |
| | | 5 | 10 | 36 | 16 | 36 | 0 | 85 | 134 |
| | Ours | 2 | 4 | 16 | 7 | 13 | 0 | 37 | 53 |
| | | 3 | 6 | 23 | 10 | 20 | 0 | 53 | 79 |
| | | 4 | 8 | 30 | 13 | 27 | 0 | 69 | 105 |
| | | 5 | 10 | 37 | 16 | 34 | 0 | 85 | 131 |
| $\{2^n, 2^{2n}-1, 2^{2n}+1\}$ | (Hariri et al., 2008) | 2 | 7 | 27 | 13 | 27 | 1 | 69 | 103 |
| | | 3 | 10 | 39 | 19 | 39 | 2 | 101 | 150 |
| | Ours | 2 | 7 | 28 | 13 | 25 | 1 | 69 | 100 |
| | | 3 | 10 | 40 | 19 | 37 | 2 | 101 | 147 |
| $\{2^{(2n-1)}, 2^{(2n+1)}-1, 2^{(2n+1)}+1\}$ | (Molahosseini et al., 2008) | 2 | 9 | 34 | 16 | 34 | 1 | 85 | 129 |
| | Ours | 2 | 9 | 35 | 16 | 32 | 1 | 85 | 126 |
| $\{2^n, 2^{2n+1}-1, 2^n+1, 2^n-1\}$ | (Molahosseini et al., 2009) | 2 | 17 | 41 | 29 | 37 | 9 | 122 | 179 |
| | | 3 | 25 | 60 | 42 | 54 | 13 | 170 | 261 |
| | Ours | 2 | 16 | 39 | 28 | 35 | 9 | 122 | 171 |
| | | 3 | 24 | 58 | 41 | 52 | 13 | 170 | 253 |
| $\{2^n+1, 2^n-1, 2^n+3, 2^n-3\}$ | (Mojahed et al., 2021a) | 3 | 35 | 137 | 65 | 137 | 0 | 139 | 514 |
| | | 4 | 50 | 276 | 114 | 165 | 0 | 179 | 770 |
| | Ours | 3 | 38 | 170 | 66 | 99 | 0 | 139 | 472 |
| | | 4 | 47 | 196 | 93 | 196 | 0 | 179 | 728 |
| $\{2^n, 2^n-1, 2^{n+1}\pm1, 2^{n+2}-1\}$ | (Ahmadifar & Torabi, 2024) | 3 | 14 | 209 | 87 | 186 | 0 | 221 | 682 |
| | Ours | 3 | 14 | 200 | 86 | 184 | 0 | 221 | 668 |

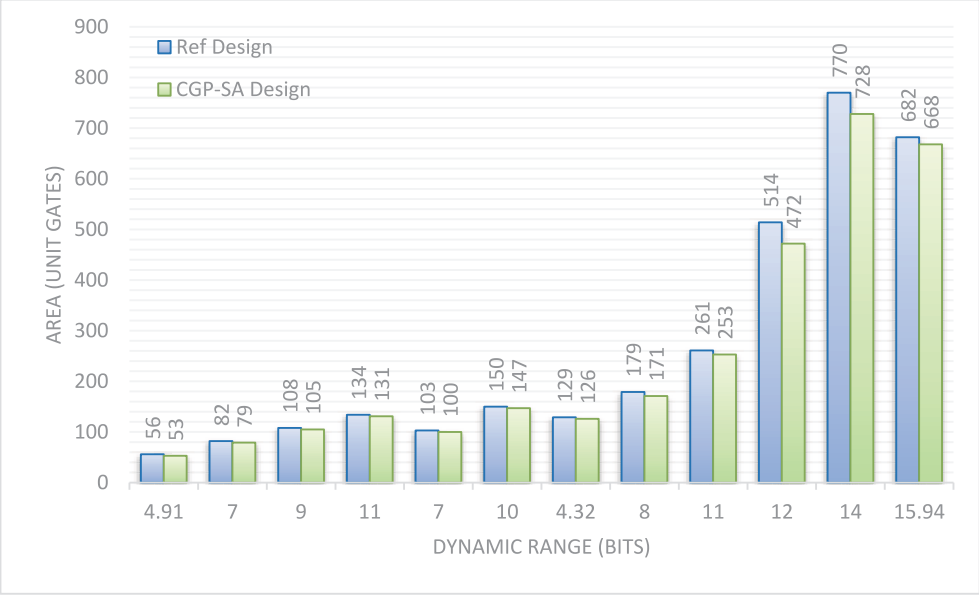


**Fig. 9.** Circuit area comparison between reference and CGP-SA designs across increasing dynamic ranges.

architectural insights (e.g., preferred low-latency operator patterns) inform gate-level search strategies. Additionally, incorporating dynamic weight adjustment into the multi-objective cost function could allow the optimization process to shift its focus over time (for example, prioritizing correctness in early generations and gradually emphasizing latency and resource efficiency as solutions stabilize), thereby improving convergence toward well-balanced hardware architectures. Moreover, incorporating Pareto-based multi-objective optimization methods (e.g., NSGA-II, SPEA2, MOEA/D) into the CGP-SA framework could offer a more principled way to explore trade-offs between latency and resource usage. By treating correctness as a strict constraint and applying Pareto selection within valid solutions, this extension would enhance transparency and flexibility in navigating conflicting hardware design objectives. Future research could also explore alternative metaheuristics, such as genetic algorithms, particle swarm optimization, or their more recent variants, to replace SA during parameter tuning. These methods may offer different advantages in navigating complex or high-dimensional parameter spaces and could further improve design quality and convergence efficiency.

## Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve the language and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Funding

Not Applicable.

### *Ethical approval*

This article does not contain any studies with human participants or animals performed by any of the authors.

### *Informed consent*

Informed consent was obtained from all individual participants included in the study.

## Appendix 1

*Appendix A:. Technical details of cartesian genetic programing*

In CGP, solutions are encoded as fixed-length genotypes that are mapped to computational graphs. Each genotype consists of a sequence of integers representing computational nodes and their connections. A node in the genotype specifies a function it performs, selected from a predefined function set, and the sources of its input data. Functions can range from basic arithmetic operations, such as addition and multiplication, to logical operations like AND and OR. The function genes define the operation for each node, while the connection genes determine the data flow by linking the node to either a program input or the output of another node in the grid. The phenotype, derived from decoding the genotype, is the active computational graph used for the solution. Some nodes in the genotype may remain inactive if they are not connected to the outputs. The two-dimensional grid structure is organized into rows and columns, with user-specified parameters determining its size and connectivity. The grid's topology impacts the computational efficiency and flexibility of the solutions.

The topology and connectivity of a CGP system are controlled by several user-defined parameters. The number of rows and columns in the grid determines the maximum number of nodes and their arrangement. For instance, a tall, narrow grid prioritizes sequential processing, while a shorter, wider grid supports parallel computations. The levels-back parameter restricts the connectivity of nodes by limiting how far back in the grid a node can connect to its inputs. A higher levels-back value increases the potential for long-range connections, enabling more complex data flows. The arity of a node, which specifies the maximum number of inputs it can have, is another critical parameter. This is dictated by the most complex function in the predefined function set. Adjusting these parameters allows for fine-tuning of the CGP structure to suit specific applications. The genotype representation in CGP offers numerous advantages that make it particularly effective for evolutionary computation. Its compactness ensures that all possible connections and functions are encoded in a single, fixed-length structure, simplifying the evolutionary process. Another significant advantage is the reusability of nodes. In CGP, nodes can be used multiple times as inputs to other nodes, allowing for implicit reuse of computations. This feature not only reduces computational overhead but also enables the creation of more complex solutions.

Mutation is the primary mechanism driving CGP evolution, introducing variation by altering function, connection, or output genes. Changes to function genes modify a node's operation, while mutations in connection or output genes redirect data sources and outputs. The mutation rate, defined as a percentage of total genes, controls the extent of these changes. Mutations can significantly impact the phenotype, activating previously unused nodes or altering program behavior, or may have no effect if they occur in non-coding genes. The inclusion of non-coding genes allows for the exploration of neutral mutations, which can help the evolutionary process escape local optima by preserving genetic diversity. This redundancy enhances robustness and adaptability, with mutations in non-coding regions potentially activating new solutions. Through this process, CGP effectively balances compactness, flexibility, and exploratory power, optimizing solutions over generations and making it highly effective for diverse computational tasks.

CGP has proven to be a powerful tool for a wide range of applications, including digital circuit design, symbolic regression, and image processing. Its mutation-centric evolutionary strategy enables gradual improvements, while its ability to represent solutions as DAGs provides flexibility in addressing diverse computational challenges. The inclusion of non-coding genes and feed-forward connectivity makes CGP particularly effective in domains requiring fault tolerance and adaptability. Fig. A.1 presents an example of a digital circuit (a two-bit multiplier circuit) designed using CGP, illustrating its capability to generate efficient and compact solutions. In this figure, the CGP genotype and its corresponding phenotype are depicted, showcasing the transformation from the encoded structure to the realized functional circuit. The genotype consists of a sequence of nodes, each defined by a set of genes that encode the node's function, chosen from a function look-up table, and its input connections. The look-up table includes logical operations such as AND (0), OR (1), and XOR (2), which are referenced by the function genes. The connection addresses indicate the flow of data through the circuit, describing how nodes interact to produce the desired outputs. Additionally, negative numbers are used in CGP's genome for the representation of circuit inputs and outputs. For example, in this figure, a function gene equal to $-3$ indicates an input node, meaning the node directly receives values from the circuit's primary inputs. Also, a function gene equal to $-2$ designates an output node, meaning the node is used to determine the final circuit output. Similarly, negative values ($-1$ and $-3$) are used in connection genes of circuit inputs and outputs to reference external connections. The phenotype visualizes the circuit's active components and connections, effectively representing the genotype's functional implementation. Additionally, inactive (non-coding) nodes, which are not utilized in the final solution, are shaded in blue dashes (e.g. nodes 8 and 11), emphasizing CGP's inherent ability to simplify and optimize circuits by excluding unnecessary components. This example highlights CGP's effectiveness in designing compact and functional digital circuits tailored to specific tasks.

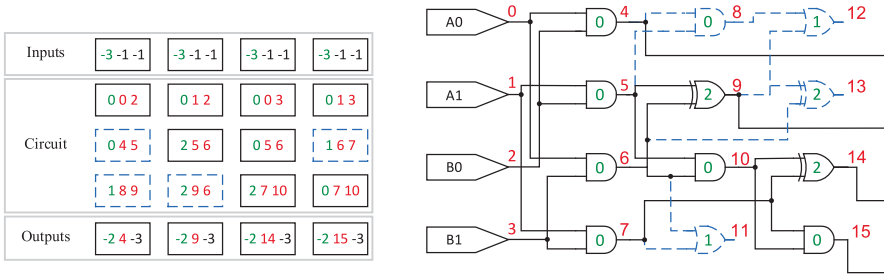


**Fig. A1.** (a) genotype and (b) phenotype of a two-bit multiplier circuit.

Recurrent CGP (RCGP) extends the principles of traditional CGP by incorporating recurrent connections, enabling the representation of cyclic graphs. This extension introduces feedback mechanisms, allowing solutions to store and utilize internal state information. RCGP is particularly advantageous for solving partially observable tasks, where outputs depend not only on current inputs but also on historical context. By lifting the acyclic constraint of CGP, RCGP creates opportunities for addressing time-dependent and dynamic problems that require memory. In RCGP, connection genes are no longer restricted to feed-forward links. Instead, nodes can connect to any other node in the program, including themselves, or to program inputs. This flexibility allows for the creation of cyclic dependencies, enabling the retention of state across iterations. Fig. A.2 illustrates an example RCGP graph, showcasing both recurrent (red lines) and feed-forward (black lines) connections.

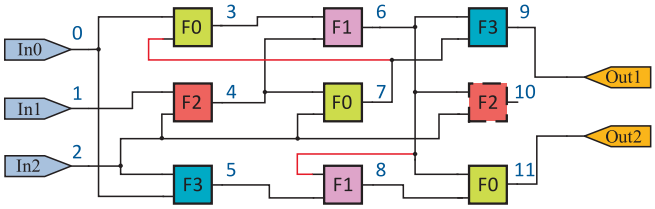


**Fig. A2.** Example of a recurrent CGP grid.

The inclusion of recurrent connections significantly expands the range of problems that can be addressed by CGP. For partially observable tasks, such as time-series prediction or reinforcement learning, the ability to incorporate memory is crucial. RCGP facilitates this by enabling nodes to store and process information from previous iterations. For digital circuit design, RCGP introduces groundbreaking potential by enabling the creation of circuits with feedback connections. Specifically, for RNS circuits, RCGP provides a unique advantage by facilitating the efficient design of Adders with end-around carry (EAC) connections, which are integral components in modular arithmetic operations. The EAC adders utilize feedback loops to ensure correct carry propagation within the modular bounds. By utilizing RCGP, designers can design RNS circuits that excel in power consumption, area, and speed.

However, the increased flexibility, achieved by incorporating feedback in RCGP, also introduces challenges. For instance, it is possible for mutations to create disconnected or purely recurrent graphs that do not utilize program inputs effectively. These configurations typically result in high cost and are filtered out during evolution. Another consideration is the potential for overfitting in tasks where recurrent connections dominate. This can be mitigated by carefully tuning the recurrent connection probability and incorporating validation techniques to ensure generalization. Despite these challenges, RCGP offers a powerful framework for tasks requiring feedback connections.

## Appendix B:. New Chinese Remainder Theorems

The New CRT-I and New CRT-II are efficient methods for reconstructing a number $X$ from its residue representation. These methods improve upon the classical CRT by reducing computational complexity, minimizing hardware requirements, and optimizing performance for specific applications such as high-speed digital systems and cryptographic circuits. The New CRT-I is based on a sequential approach to compute the original number $X$ from its residues $(x_1, x_2, ..., x_n)$. For an $n$-moduli set $\{p_1, p_2, ..., p_n\}$, the number $X$ is reconstructed as:

$$X = \left| x_1 + k_1(x_2 - x_1)p_1 + k_2(x_3 - x_2)p_1p_2 + \ldots + k_{n-1}(x_n - x_{n-1}) \prod_{i=1}^{n-1} p_i \right|_P, \tag{B1}$$

where, $P = \prod_{i=1}^{n} p_i$ is the dynamic range, and $k_j$ is the multiplicative inverse of $\prod_{i=1}^{j} p_i$.

The New CRT-II provides an alternative reconstruction method that emphasizes parallelism and reduces sequential dependencies. It emphasizes parallelism and minimizes sequential dependencies by dividing the moduli set into smaller groups and combining intermediate results using a divide-and-conquer strategy. For a 4-moduli set $\{p_1, p_2, p_3, p_4\}$ and residues $(x_1, x_2, x_3, x_4)$, we compute the first intermediate result ($Z$) as follows:

$$Z = x_1 + p_1 |k_1(x_2 - x_1)|_{p_2}, \text{ where } |k_1 p_1|_{p_2} = 1 \tag{B2}$$

Next, the second intermediate result ($Y$) is calculated as follows:

$$Y = x_3 + p_3 |k_2(x_4 - x_3)|_{p_4}, \text{ where } |k_2 p_3|_{p_4} = 1 \tag{B3}$$

Finally, these intermediate result ($Z$ and $Y$) are combined based on the following equation, in order to create the final result $X$:

$$X = Z + p_1p_2 |k_3(Y - Z)|_{p_3p_4}, \text{ where } |k_3 p_1 p_2|_{p_3p_4} = 1 \tag{B4}$$

New CRT-II is highly efficient for residue-to-binary conversion, using parallelism by allowing independent computations of $Z$ and $Y$ for partitioned groups, enabling efficient hardware implementations. It is scalable, accommodating larger moduli sets through further partitioning, and reduces computational complexity by focusing on modular operations within smaller dynamic ranges, avoiding large modulo calculations. Due to its reduced hardware complexity, modular efficiency, and support for parallelism, New CRT-II is particularly well-suited for high-speed applications such as digital signal processing, modular arithmetic units, and secure cryptographic systems.

## Appendix C:. Case Study: Reverse converter design for {4, 7, 9} moduli set

In order to illustrate the effectiveness of the proposed two-phase CGP optimization methodology, we present a detailed case study on designing an RNS reverse converter for the well-known moduli set {4,7,9}. Conventional methods, such as the CRT or MRC methods are typically used for this purpose. However, these approaches often lead to high computational complexity and increased hardware resource consumption. In contrast, the proposed CGP-based method systematically evolves an optimized architecture, efficiently balancing accuracy, latency, and resource utilization. The following sections provide a detailed breakdown of the two-phase design process, starting with the development of a high-level module architecture in Phase 1, followed by gate-level optimization in Phase 2.

***Phase 1: High-Level Module Design***.

As mentioned above, the first phase of the CGP-based design process focuses on constructing an efficient module-level architecture for the RNS reverse converter. To achieve this, the process begins with defining a set of arithmetic and bitwise operations that form the fundamental building blocks of the converter. The arithmetic operations include addition, subtraction, modular addition, and multi-operand modular arithmetic. Bitwise operations such as logical shifts, circular left and right shifts, concatenate, and bitwise complement are also included.

Next, a truth table is generated by considering all possible residue combinations corresponding to the given moduli set. For {4,7,9}, the total number of unique residue sets is computed as $4 \times 7 \times 9 = 252$. Each residue combination is mapped to its corresponding integer using the CRT algorithm. The next step involves CGP encoding and evolutionary optimization. Candidate designs are encoded within a CGP grid, where nodes represent operations selected from the predefined function set. Input nodes correspond to the residues, and output nodes generate the final converted value. The cost function evaluates candidate designs based on three key criteria. Correctness is assessed by comparing output values against the reference truth table. Latency is minimized by reducing the number of active computation columns in the CGP grid. Resource usage is optimized by minimizing the number of active functional nodes. After multiple generations of mutation and selection, the CGP optimization process converges toward an efficient high-level representation of the RNS reverse converter. This optimized design serves as the input for the next phase.

Fig. C.1 represents the final output of this phase, illustrating the optimized CGP-based module-level architecture for the RNS reverse converter. Fig. C.1 (a) illustrates the full CGP grid, displaying all operations and connections, including both active and inactive components in the final solution. Fig. C.1 (b) focuses on the active CGP grid, where inactive nodes and connections have been removed, presenting only the active elements that contribute to the final computation. This architecture strategically employs key operations such as LeftShift, ModuloAdder, CircularLeftShift, Complement, and Concatenate to process input residues and compute the final converted output. The optimized active grid shows how the design achieves an effective balance between correctness, latency, and resource efficiency, ensuring a highly efficient solution.

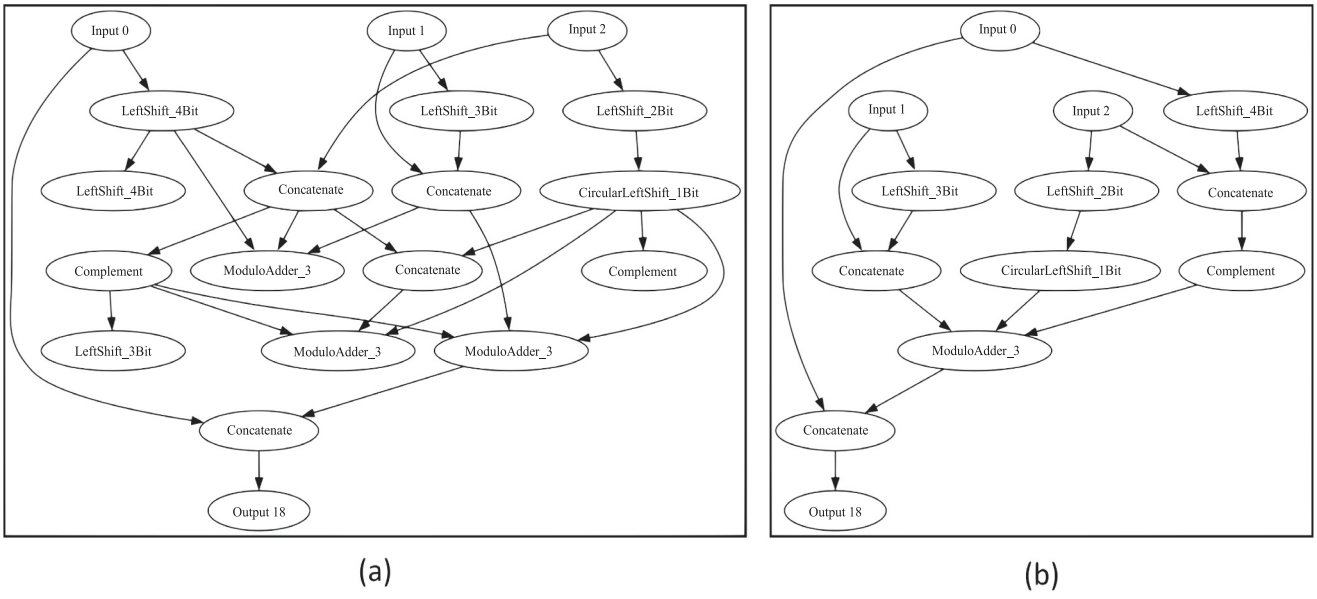


**Fig. C1.** Optimized CGP-based module-level architecture for the reverse converter: (a) Full CGP grid, (b) Active CGP grid.

***Phase 2: Gate-Level Optimization***

The second phase of the proposed methodology focuses on refining the high-level design obtained in the first phase into a gate-level circuit. This phase ensures efficient hardware implementation by minimizing delay, power consumption, and resource utilization, making the design suitable for practical use in RNS-based systems.

The process begins with gate-level mapping, where each function in the high-level representation is translated into a network of basic logic gates. The gate set used for this phase includes fundamental elements such as AND, OR, NOT, and XOR gates. Additionally, more complex elements like multiplexers (MUX) and arithmetic gates such as SUM and CARRY are incorporated. These gates serve as the building blocks of the gate-level design. The CGP evolutionary process applied at this level operates on a grid of logic gates obtained from gate-level mapping and iteratively improves the

circuit. Each node in the grid represents a logic operation, with input nodes corresponding to binary residue values of the RNS reverse conversion, and output nodes representing the binary result of the conversion.

To guide the optimization process, a cost function evaluates candidate designs by balancing accuracy with efficiency. Correctness is prioritized by ensuring the circuit produces outputs consistent with the reference truth table. At the same time, design efficiency is applied based on minimizing the cumulative cost of three key performance metrics. First, critical path delay is reduced by minimizing the number of active columns in the gate-level grid, ensuring a shorter and faster computational path. Second, resource utilization is minimized by decreasing the total number of active gates in the circuit, which directly correlates with hardware area and power consumption. Lastly, circuit efficiency is improved by penalizing active nodes with higher costs, thereby encouraging the use of less resource-intensive components. In this regard, each component is characterized by a specific hardware cost, derived by the product of the power, delay and area requirements (called PDAP), to ensure resource-efficient mapping. Simpler gates like AND and OR have lower costs, while more complex gates such as SUM and CARRY incur higher costs due to their complexity. These cost parameters are integrated into the optimization process through the cost evaluation, where circuits using resource-intensive gates are penalized. For example, designs relying excessively on SUM and CARRY are assigned higher resource penalties compared to those using simpler, functionally equivalent configurations. This approach ensures that the CGP framework prioritizes designs that minimize resource utilization without compromising functionality.

Throughout the evolutionary process, additional mechanisms such as dynamic mutation rate decay and tournament selection adjustments are applied to improve convergence. The mutation rate is reduced exponentially across generations to balance exploration and exploitation, starting with a higher rate to explore the design space and gradually focusing on fine-tuning promising solutions. Tournament size is dynamically adjusted based on the generation count, starting smaller to maintain diversity and increasing in later stages to refine high-quality designs.

The final optimized gate-level circuit for the {4, 7, 9} moduli set reverse converter is depicted in Fig. C.2. The figure shows a detailed mapping of the gate-level architecture, with distinct modules corresponding to Inverters, Full Adders (FA) and Half Adders (HA). Input bits (e.g., $X_{32}$, $X_{31}$, $X_{30}$, …) are processed through interconnected adder blocks, representing the 3-operand modulo adder corresponding to ModuloAdder_3 block in the module-level diagram. These adder blocks compute partial sums ($S_i$) and carries ($C_i$) at each stage, which are systematically propagated and combined to produce the final binary results ($O_7$, $O_6$, …, $O_0$).

As shown in Fig. C.2, the CGP evolutionary process has successfully simplified the circuit to enhance efficiency. More specifically, in the green block, the SUM circuit has been removed from the FA, reducing complexity of this block while maintaining the overall functionality of the reverse converter. Additionally, in the purple blocks, the FA circuits have been converted to HA circuits, which have a lower hardware cost compared to FA, further optimizing resource utilization. These modifications demonstrate how the CGP framework intelligently reduces hardware costs without compromising the accuracy of the design. This optimized architecture effectively balances output correctness with power efficiency, computational speed and reduced hardware requirements, making it a practical solution for RNS-based systems. The design highlights the impact of the CGP evolution process in refining the circuit to achieve optimal performance.

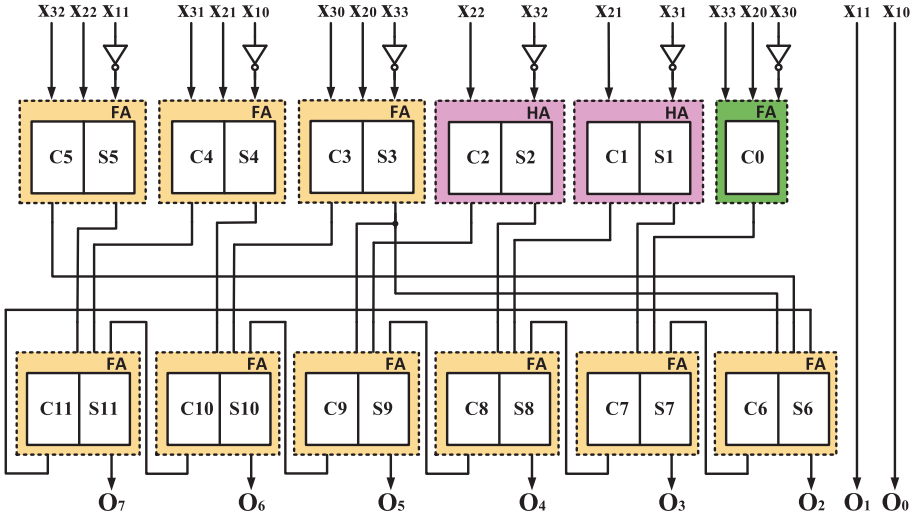


**Fig. C2.** Optimized CGP-based gate-level circuit for the {4, 7, 9} moduli set reverse converter.

## Data availability

The data that support the findings of this study are available from the corresponding author, upon reasonable request.

## References


Ahmadifar, H., & Torabi, Z. (2024). Adder-Only Reverse Converters for 5-Moduli Set {2 q, 2 q− 1, 2 q+ 1±1, 2 q+ 2− 1}. *IETE Journal of Research, 70*(9), 7346-7353.

Akbari, A., Hosseinzadeh, M., TaghipourEivazi, S., & Jassbi, S. J. (2021). A New High-Speed, Low-Area Residue-to-Binary Converter For the Moduli Set {2 4 n, 2 2 n+ 1, 2 n+ 1, 2 n-1} Based on CRT-1. *Circuits, Systems, and Signal Processing, 40*(11), 5773-5786.

Babkin, E., & Ulitin, B. (2024). Ontology-Based Evolution of Domain-Oriented Languages: Models, Methods and Tools for User Interface Design in General-Purpose Software Systems. Springer Nature.

Balaji, M., & Padmaja, N. (2024). Area and delay efficient RNS-based FIR filter design using fast multipliers. *Meas.: Sens., 31*, Article 101014.

Berndt, A. A. S., Abreu, B., Campos, I. S., Lima, B., Grellert, M., Carvalho, J. T., & Meinhardt, C. (2022). A CGP-based logic flow: Optimizing accuracy and size of approximate circuits. *Journal of Integrated Circuits and Systems, 17*(1), 1–12.

Boyvalenkov, P., Lyakhov, P., Semyonova, N., Valueva, M., Boyvalenkov, G., Minenkov, D., & Kaplun, D. (2023). Residue number systems with six modules and efficient circuits based on power-of-two diagonal modulus. *Comput. Electr. Eng., 110*, Article 108854.

Fernandes, G., Vassoler, A., Bezerra, E., Sousa, L., & Pettenghi, H. (2024). A Comprehensive Approach and Analysis of reverse Converters for a Class of Moduli Sets. *2024 IEEE 15th Latin America Symposium on Circuits and Systems (LASCAS)*.

Givaki, K., Khonsari, A., Gholamrezaei, M., Gorgin, S., & Najafi, M. H. (2023). A Generalized Residue Number System Design Approach for Ultralow-Power Arithmetic Circuits based on Deterministic Bit-Streams. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., 42*(11), 3787–3800.

Hariri, A., Navi, K., & Rastegar, R. (2008). A new high dynamic range moduli set with efficient reverse converter. *Comput. Math. Appl., 55*(4), 660–668.

Hiasat, A., & Sweidan, A. (2003). Residue number system to binary converter for the moduli set (2n− 1, 2n− 1, 2n+ 1). *journal of Systems Architecture*, *49*(1-2), 53-58.

Hrbácek, R. (2017). *Automated Multi-Objective Parallel Evolutionary Circuit Design and Approximation* PhD thesis, Department of Computer Systems, Faculty of Information Tech …].

Husa, J., & Sekanina, L. (2024). Semantic mutation operator for a fast and efficient design of bent Boolean functions. *Genet. Program Evolvable Mach., 25*(1), 3.

Jaberipur, G., & Ahmadifar, H. (2014). A ROM-less reverse RNS converter for moduli set {2q±1, 2q±3}. *IET Computers & Digital Techniques, 8*(1), 11–22.

Jamróz, D. (2024). Using Multivalued Cartesian Genetic programming (M-CGP) for Automatic Design of Digital Sequential Circuits. *Appl. Sci., 14*(23), 11153.

Kocnova, J., & Vasicek, Z. (2020). EA-based resynthesis: An efficient tool for optimization of digital circuits. *Genet. Program Evolvable Mach., 21*, 287–319.

Li, Z., Liu, Y., Lu, X., Wang, R., Wei, B., Chen, C., & Wang, K. (2024). Faster Bootstrapping via Modulus Raising and Composite NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2024*(1), 563–591.

Lima, L. S., Bernardino, H. S., & Barbosa, H. J. (2019). Designing combinational circuits using a multi-objective cartesian genetic programming with adaptive population size. *Machine Learning, Optimization, and Data Science: 5th International Conference, LOD 2019*.

Majd, K. M., & Molahosseini, A. S. (2022). Energy-Efficient Residue-to-Binary Conversion based on a Modulo-Adder-Free Architecture. *2022 30th International Conference on Electrical Engineering (ICEE)*.

Mansoor, A., Fazeli, M., Rahmani, A. M., & Reshadi, M. (2023). Optimized reverse converters with multibit soft error correction support at 7nm technology. *Comput. Electr. Eng., 107*, Article 108654.

Milano, N., & Nolfi, S. (2021). Enhancing Cartesian genetic programming through preferential selection of larger solutions. *Evol. Intel., 14*(4), 1539–1546.

Miller, J., & Turner, A. (2015). Cartesian genetic programming. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*.

Miller, J. F. (2020). Cartesian genetic programming: Its status and future. *Genet. Program Evolvable Mach., 21*(1), 129–168.

Mohan, P. A. (2016). Reverse Converters for the Moduli Set {2 n+ 1− 3, 2 n− 1, 2 n− 1− 1}. *IETE Journal of Education*, *57*(1), 31-43.

Mojahed, M., Molahosseini, A. S., & Zarandi, A. A. E. (2021a). Magnitude Comparison and Sign Detection based on the 4-Moduli Set {2n+ 1, 2n− 1, 2n+ 3, 2n− 3}. *Majlesi Journal of Electrical Engineering*, *15*(3), 93-103.

Mojahed, M., Molahosseini, A. S., & Zarandi, A. A. E. (2021b). Multifunctional unit for reverse conversion and sign detection based on five-moduli set {2 2n, 2 n+ 1, 2 n− 1, 2 n+ 3, 2 n− 3}. *Computer Science*, *22*, 101-121.

Molahosseini, A. S., Navi, K., Dadkhah, C., Kavehei, O., & Timarchi, S. (2009). Efficient Reverse Converter Designs for the New 4-Moduli Sets $\{2^{n}-1, 2^{n}, 2^{n}+ 1, 2^{2n+ 1}-1\} $ and $\{2^{n}-1, 2^{n}+ 1, 2^{2n}, 2^{2n}+ 1\} $ Based on New CRTs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *57*(4), 823-835.

Molahosseini, A. S., Navi, K., Hashemipour, O., & Jalali, A. (2008). An efficient architecture for designing reverse converters based on a general three-moduli set. *J. Syst. Archit., 54*(10), 929–934.

Patronik, P., & Piestrak, S. J. (2018). Design of RNS reverse converters with constant shifting to residue datapath channels. *Journal of Signal Processing Systems, 90*, 323–339.

Patronik, P., & Piestrak, S. J. (2023). Design of reverse converters for the general RNS 3-moduli set {2 k, 2 n− 1, 2 n+ 1}. *EURASIP Journal on Advances in Signal Processing, 2023*(1), 92.

Prashanth, H., & Rao, M. (2024). Accelerated and highly correlated ASIC synthesis of AI hardware subsystems using CGP. *IET Computers & Digital Techniques, 2024*(1), Article 6623637.

Prediger, V., Bairros, F., Seman, L. O., Bezerra, E. A., & Pettenghi, H. (2023). RNS processor using moduli sets of the form 2n±1. *Int. J. Circuit Theory Appl., 51*(7), 3432–3442.

Saheed, Y. K., Kehinde, T. O., Ayobami Raji, M., & Baba, U. A. (2024). Feature selection in intrusion detection systems: A new hybrid fusion of Bat algorithm and Residue Number System. *Journal of Information and Telecommunication, 8*(2), 189–207.

Selianinau, M., & Povstenko, Y. (2022). An Efficient Parallel reverse Conversion of Residue Code to Mixed-Radix Representation based on the Chinese Remainder Theorem. *Entropy, 24*(2), 242.

Shenets, N. N. (2024). On modular (CRT-based) secret sharing. *Journal of Computer Virology and Hacking Techniques*, 1–18.

Souza, L., & Bernardino, H. (2020). On the analysis of mutation operators in multiobjective cartesian genetic programming for designing combinational logic circuits. *Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*.

Turan, F., Roy, S. S., & Verbauwhede, I. (2020). HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. *IEEE Trans. Comput., 69*(8), 1185–1196.

Vasicek, Z., & Sekanina, L. (2014). Evolutionary approach to approximate digital circuits design. *IEEE Trans. Evol. Comput., 19*(3), 432–444.

Yu, H.-B., Zheng, Q.-X., Liu, Y.-J., Bi, J.-G., Duan, Y.-F., Xue, J.-W., Wu, Y., Cao, Y., Cheng, R., & Wang, L. (2023). An improved method for predicting truncated multiple recursive generators with unknown parameters. *Des. Codes Crypt., 91*(5), 1713–1736.