Efficient Cartesian Genetic Programming-based Automatic Synthesis Framework for Reversible Quantum-Flux-Parametron Logic Circuits

Rongliang Fu, Robert Wille, Nobuyuki Yoshikawa *Fellow, IEEE*, and Tsung-Yi Ho *Fellow, IEEE*

***Abstract*—Reversible computing has garnered significant attention as a promising avenue for achieving energy-efficient computing systems, particularly within the realm of quantum computing. The reversible quantum-flux-parametron (RQFP) is the first practical reversible logic gate utilizing adiabatic superconducting devices, with experimental evidence supporting both its logical and physical reversibility. Each RQFP logic gate operates on alternating current (AC) power and features three input ports and three output ports. Notably, each output port is capable of implementing a majority function while driving only a single fan-out. Additionally, the three inputs to each gate must arrive in the same clock phase. These inherent characteristics present substantial challenges in the design of RQFP logic circuits. To address these challenges, this paper proposes an automatic synthesis framework for RQFP logic circuit design based on efficient Cartesian genetic programming (CGP). The framework aims to minimize both the number of RQFP logic gates and the number of garbage outputs within the generated RQFP logic circuit. It incorporates the specific characteristics of the RQFP logic circuit by encoding them into the genotype of a CGP individual. It also introduces several point mutation operations to facilitate the generation of new individuals. Furthermore, the framework integrates circuit simulation with formal verification to assess the functional equivalence between the parent and its offspring. Experimental results on RevLib and reversible reciprocal circuit benchmarks demonstrate the effectiveness of our framework.**

***Index Terms*—Superconducting electronics, reversible quantum-flux-parametron, reversible computing, logic synthesis, Cartesian genetic programming**

I. Introduction

THE impetus for developing reversible computing stems from the inherent limitations of traditional irreversible logic systems, especially regarding energy dissipation. Conventional logic systems, such as complementary metal-oxide-semiconductor (CMOS) logic, are prone to irreversible operations that result in information loss and subsequent energy dissipation in the form of heat. According to Landauer's principle established in 1961 [1], the loss of each bit of information incurs an energy cost quantified as $\mathrm{k_B T \ln 2}$ J, where k_B is the Boltzmann constant and T denotes the temperature of the system. This heat dissipation becomes increasingly significant with rising processor frequencies. Consequently, reversible computing has emerged as a promising paradigm for reducing energy dissipation in logic operations, drawing considerable academic interest. Theoretically, it offers the potential for near-zero energy consumption by preserving information [2]. This paradigm is particularly relevant to quantum computing, where operations are inherently reversible [3].

Nonetheless, the practical realization of reversible computing poses substantial challenges. It requires both logical and physical reversibility, along with the development of ultra-low power logic devices [4]. In this context, Takeuchi *et al.* introduced the reversible quantum-flux-parametron (RQFP) [5], marking the first practical reversible logic gate utilizing adiabatic superconducting devices. The logical and physical

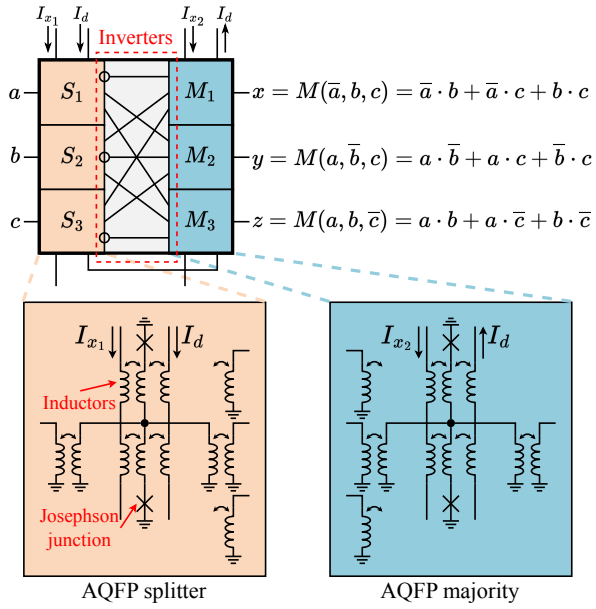

Fig. 1. Functional and structural schematic of the RQFP logic gate, where symbols employed include $\cdot$ for logical conjunction (AND), $+$ for logical disjunction (OR), and $\overline{\ }$ for logical negation (NOT).

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust. This work is supported in part by the Research Grants Council of Hong Kong SAR (No. CUHK14207523); in part by the European Union's Horizon 2020 research and innovation programme (DA QC, No. 101001318); and in part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

Rongliang Fu and Tsung-Yi Ho are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China. E-mail: {rlfu, tyho}@cse.cuhk.edu.hk.

Robert Wille is with the Chair for Design Automation, Technical University of Munich, 80333 Munich, Germany, and also with Software Competence Center Hagenberg GmbH, 4232 Hagenberg, Austria. E-mail: robert.wille@tum.de.

Nobuyuki Yoshikawa is with the Department of Electrical and Computer Engineering, Yokohama National University, Yokohama 240-8501, Japan. E-mail: nyoshi@ynu.ac.jp.

reversibility of RQFP has been validated experimentally [5], [6]. Furthermore, Yamae *et al.* designed and fabricated an RQFP logic full adder [7], further illustrating the feasibility of RQFP logic circuits. Consequently, RQFP logic has attracted increasing research interest [8]–[11].

Despite these advances, the design of RQFP circuits remains complex due to the lack of automated design tools [8]. RQFP logic gates are implemented using adiabatic quantum-flux-parametron (AQFP) technology [12], an energy-efficient superconductor logic element based on the quantum flux parametron [13]. Fig. 1 illustrates a standard RQFP logic gate structure [6], [7], consisting of three AQFP splitter gates and three AQFP majority gates. The RQFP logic gate thus provides three functional outputs, $R(a,b,c) = \{\mathrm{M}(\overline{a},b,c), \mathrm{M}(a,\overline{b},c), \mathrm{M}(a,b,\overline{c})\} = \{x,y,z\}$, where a, b, and c are inputs; x, y, and z are outputs; and $\mathrm{M}(\cdot)$ denotes a three-input majority function $\mathrm{M}(a,b,c) = a\cdot b + a\cdot c + b\cdot c$. In contrast, conventional reversible logic circuits primarily utilize the basic Toffoli [14] and Fredkin [15] gates, along with their extensions known as multiple-control Toffoli (MCT) and multiple-control Fredkin (MCF) gates. Fig. 2(a) and 2(b) show the schematics of MCT and MCF gates, respectively. They operate as multi-controlled NOT and multi-controlled SWAP gates, respectively, with output functions primarily focused on the last one or two output ports to realize the XOR-sum-of-products. Moreover, RQFP logic should adhere to constraints inherent to AQFP logic, such as the fan-out limitation and the clock-synchronized data propagation requirement. Therefore, due to these functional and structural distinctions, traditional logic synthesis methods are unsuitable for RQFP logic.

The existing exact synthesis method [10] formulates the generation of RQFP logic circuits as a Boolean satisfiability (SAT) problem. While this approach is capable of exactly synthesizing RQFP logic circuits based on a specified number of RQFP logic gates and garbage outputs, it suffers from significant scalability limitations, rendering it applicable only to very small circuits. In contrast, our previous work, RCGP [11], directly employs the Cartesian genetic programming (CGP) approach to facilitate the generation of large RQFP logic circuits. Both methods aim to optimize the number of RQFP logic gates and the number of garbage outputs, as these two metrics severely affect the energy dissipation of RQFP logic circuits. However, they do not explicitly constrain the inverter configuration of RQFP logic gates to ensure logical reversibility. Therefore, the exploration of reversible logic synthesis methods for legal RQFP logic circuits remains vital.

In response to these challenges, this paper introduces an efficient Cartesian genetic programming framework for the automatic synthesis of RQFP logic circuits, aiming to optimize both the number of RQFP logic gates and the number of garbage outputs. In summary, this paper makes the following contributions:

- This paper thoroughly analyzes the characteristics of RQFP logic and introduces the RQFP splitter and buffer to meet its design requirements.
- This paper presents a novel automatic synthesis framework based on efficient Cartesian genetic programming to effectively and efficiently generate RQFP logic circuits.

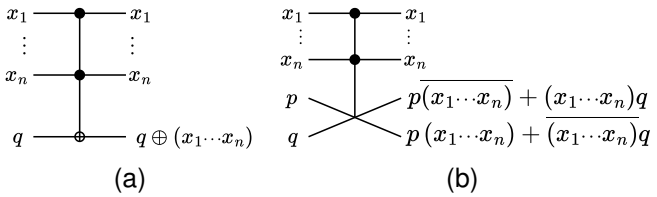

Fig. 2. Functional schematics of (a) MCT gate and (b) MCF gate.

- This paper presents a CGP encoding for RQFP logic circuit representation, three corresponding point mutation operations for individual generation, and a new shrinking operation of the genotype for search space reduction.
- This paper limits the inverter configuration of RQFP logic gates to ensure logical reversibility of generated RQFP logic circuits.

Moreover, experimental results on RevLib circuits [16] and reversible reciprocal circuits [17] demonstrate the effectiveness and efficiency of our proposed framework.

- In terms of performance, our proposed framework yields comparable effectiveness to the exact logic synthesis method [10] on small RevLib circuits [16]. Compared with two other baselines, including the heuristic method in Section V and the RCGP [11], our framework achieves an average reduction of 58.30% and 6.11% in the number of RQFP logic gates and 70.56% and 18.21% in the number of garbage outputs, respectively.
- In terms of scalability, the experimentation extends to larger-scale circuits sourced from RevLib circuits [16] and reversible reciprocal circuits [17]. Notably, the exact logic synthesis [10] fails to provide feasible solutions for these large circuits. Compared with two other baselines, our proposed framework achieves an average reduction of 47.28% and 18.30% in the number of RQFP logic gates and 63.72% and 15.12% in the number of garbage outputs, respectively. Meanwhile, our framework is efficient, showcasing an average runtime reduction of 35.84% over the RCGP [11].

The rest of this paper is organized as follows. Section II provides an overview of several preliminaries and delves into the characteristics of RQFP logic circuits. Section III introduces the problem formulation. Section IV describes our proposed automatic synthesis framework for RQFP logic circuits. Section V demonstrates the effectiveness and efficiency of our framework. Section VI summarizes this paper and discusses potential avenues for further optimizing RQFP circuit design.

II. Preliminaries

A. Reversible Computing

Reversible computing is an emerging paradigm that seeks to overcome the energy inefficiencies inherent in traditional computing systems. Conventional computing relies on irreversible operations, which result in information loss and energy dissipation, typically manifesting as heat. This inefficiency is primarily due to the loss of information during processes like

bit erasure. According to Landauer's principle [1], each erased bit incurs a specific energy cost, which becomes increasingly significant as computational demands grow.

Reversible computing offers a solution by allowing computational processes to be reversed. This means that the outputs of a computation can be used to uniquely determine its inputs, *i.e.*, logical reversibility, effectively preserving information throughout the process [2]. This approach has the potential to significantly reduce energy consumption and enable theoretically energy-free computation. The concept dates back to the 1970s, with pioneering work by researchers like Charles H. Bennett and Richard Feynman, who proposed that logical operations could be designed to be reversible [2], [14], [15], [18]. This paradigm has significant implications for the development of low-power computing systems, quantum computing, and nanotechnology, where energy efficiency is paramount. However, the physical implementation of reversible computing is challenging. It requires both logical and physical reversibility, as well as ultra-low power logic devices. Despite these challenges, by integrating reversible logic with emerging technologies, reversible computing holds promise for creating energy-efficient, high-performance computing solutions that align with the growing need for sustainability in technological advancement [3].

B. Adiabatic Quantum-Flux-Parametron Logic

AQFP logic is a superconducting digital technology renowned for its ultra-low power consumption [12], [19]. It combines the principles of adiabatic switching with the quantum-flux-parametron mechanism [13] to achieve energy-efficient computations. This makes AQFP a promising alternative to conventional CMOS technology, especially in applications demanding low power consumption. Consequently, extensive research has focused on AQFP logic circuit design. In terms of logical design, people have proposed majority-inverter graph (MIG)–based logic optimization methods to minimize circuit size and depth [20]–[22]. Meanwhile, the unique characteristics of AQFP logic, specifically fan-out limitation and clock-synchronized data propagation requirement, have also been effectively addressed [23]–[26]. In terms of physical design, multi-phase clocking-based placement methods have been proposed for AQFP logic circuits to adhere to spacing and timing constraints, particularly in relation to the 4-phase clocking scheme [27], [28] and the delay-line clocking scheme [29].

C. Reversible Quantum-Flux-Parametron Logic

The RQFP logic gate, a member of the superconducting logic family, exhibits both logical and physical reversibility [5], [6]. As illustrated in Fig. 1, it consists of three one-to-three AQFP splitter gates (S_1, S_2, S_3) and three three-input AQFP majority gates (M_1, M_2, M_3). Similar to AQFP logic, excitation currents I_{x_1} and I_{x_2} are required to drive these splitter and majority gates, respectively, along with a direct current I_d to provide an offset flux. Upon the arrival of the excitation currents, the Josephson junction (JJ) within the AQFP logic gate switches, enabling the inductor to produce the output current. Furthermore, since the RQFP logic gate is constructed using AQFP logic gates, it inherits the fundamental characteristics of AQFP logic, particularly the limitation on fan-out and the requirement for clock-synchronized data propagation.

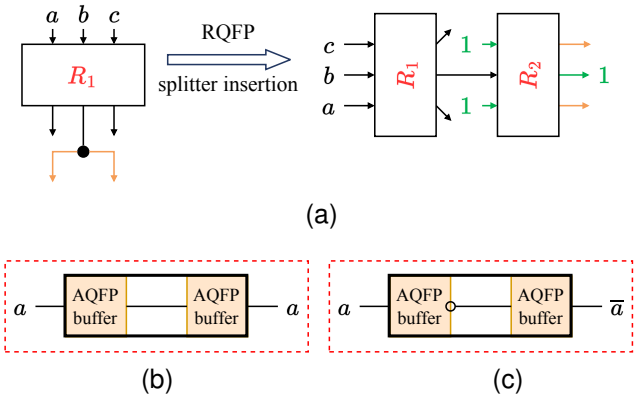

Fig. 3. Schematics of (a) splitter insertion for RQFP logic, (b) RQFP buffer, and (c) RQFP inverter. The gate R_1 has four fan-outs with three distinct types of functional outputs. After RQFP splitter insertion, the new gate R_2 can afford the same output function with two fan-outs (orange arrows).

1) RQFP Functionality: The outputs of the RQFP logic gate are derived from its three three-input AQFP majority components, thereby naturally implementing the majority function. In a standard RQFP logic gate, three inverters are positioned in fixed locations, one in front of each AQFP majority gate, *i.e.*, $R(a,b,c) = \{\mathrm{M}(\overline{a},b,c), \mathrm{M}(a,\overline{b},c), \mathrm{M}(a,b,\overline{c})\}$. This configuration establishes a one-to-one mapping between the three inputs and three outputs of the RQFP gate, rendering it logically reversible [5]. Notably, the flexibility to integrate inverters into any input of each AQFP majority gate allows for an extension of the functionality of RQFP logic gates, enabling the configuration of each RQFP logic gate's function through the inverter settings. Consequently, each output of an RQFP logic gate can have eight function choices, including $\mathrm{M}(a,b,c)$, $\mathrm{M}(a,b,\overline{c})$, $\mathrm{M}(a,\overline{b},c)$, $\mathrm{M}(a,\overline{b},\overline{c})$, $\mathrm{M}(\overline{a},b,c)$, $\mathrm{M}(\overline{a},b,\overline{c})$, $\mathrm{M}(\overline{a},\overline{b},c)$, and $\mathrm{M}(\overline{a},\overline{b},\overline{c})$.

2) RQFP Splitter: In AQFP logic, each output of each AQFP logic gate can only drive one successor, adhering to the single fan-out limitation. While the AQFP splitter gate can achieve multiple fan-outs, it cannot be directly employed in RQFP logic due to its irreversibility. To maintain the reversibility, the RQFP splitter can be constructed by the combination of an RQFP logic gate and constant inputs. For example, the one-to-two RQFP splitter can be realized by $R(1,x,1) = \{\mathrm{M}(\overline{1},x,1), \mathrm{M}(1,\overline{x},1), \mathrm{M}(1,x,\overline{1})\} = \{x,1,x\}$. Fig. 3(a) illustrates an example of splitter insertion in RQFP logic, where the second output port of gate R_1 has two fan-outs. A one-to-two RQFP splitter R_2 with two constant inputs can be inserted after gate R_1 to satisfy the fan-out limitation.

3) RQFP Buffer: Furthermore, all inputs to each gate in AQFP logic must arrive in the same clock phases, fulfilling the requirement for clock-synchronized data propagation. This requirement can be addressed by the insertion of AQFP buffers. Similarly, the insertion of RQFP buffers into RQFP logic circuits becomes essential, ensuring that all data input

paths to the logic gate from primary inputs pass through the same number of logic gates (namely *logic level*). One method involves utilizing a combination of an RQFP logic gate and constant inputs to construct the RQFP buffer. For instance, $R(x,1,1) = \{\mathrm{M}(\overline{a},1,1), \mathrm{M}(a,\overline{1},1), \mathrm{M}(a,1,\overline{1})\} = \{1,a,a\}$ can achieve the RQFP buffer function while producing two additional outputs, which may result in significant cost. Given the requirements for logical and physical reversibility in reversible computing, two cascaded AQFP buffers can be utilized to construct an RQFP buffer [10], as shown in Fig. 3(b). Furthermore, the RQFP inverter can also be constructed by placing an inverter in front of the latter RQFP buffer, as shown in Fig. 3(c). It can connect the primary input (PI) and primary output (PO), which have complementary logic values.

4) RQFP Logic Circuits: RQFP logic gates have been experimentally validated as effective components for the design of reversible logic circuits. To achieve reversible logic circuit design by RQFP logic gates, we need to consider the following three requirements:

(a) the realization of given functions by RQFP logic gates;
(b) single fan-out limitation;
(c) clock-synchronized data propagation.

The first ensures the functional legalization of generated circuits. The last two ensure the legalization of generated circuits on the structure inherited from RQFP logic. That means that all inputs to each logic gate have the same logic level, and all output ports of each logic gate have at most one fan-out. Notably, the last two depend on the result of the first and can be satisfied by inserting RQFP splitters and RQFP buffers.

Generally, a reversible logic circuit can implement a reversible Boolean function that represents a one-to-one mapping between vectors of primary inputs and primary outputs. Additional constant inputs and outputs must be introduced to convert an irreversible function into a reversible one. These additional outputs are called garbage outputs. However, excessive garbage outputs can lead to energy inefficiencies. Therefore, the design of RQFP logic circuits must consider both the number of RQFP logic gates and the number of garbage outputs.

D. Cartesian Genetic Programming

Evolutionary algorithms have demonstrated significant efficacy in addressing complex optimization problems. A prominent example is Cartesian genetic programming (CGP), introduced by Julian F. Miller and Peter Thomson in 2000 [30]. CGP is a specialized form of genetic programming that employs a directed acyclic graph (DAG) structure, visualized as a two-dimensional array of computational nodes in r rows and c columns. Each node in the graph represents a specific function with n_i inputs and n_o outputs and is encoded by n_i+1 integer genes. For the j^{th} node, the function gene F_j records the index of its corresponding computational function in a predefined function look-up table Γ, while the remaining n_i connection genes $C_{j,0}\cdots C_{j,k}\cdots C_{j,n_i-1}$ specify the sources of its input data, where $C_{j,k}$ indicates that the k^{th} input comes from the node with index $C_{j,k}$. When the problem requires n_{po} program outputs, the genotype will increase n_{po} output genes $O_0\cdots O_k\cdots O_{n_{po}-1}$, where O_k records the address of the node from which the k^{th} output data comes. Hence, the genotype in CGP is represented as a string of fixed-length integers, as follows:

$$C_{0,0}\cdots C_{0,n_i-1}\underline{F_0}C_{1,0}\cdots C_{1,n_i-1}\underline{F_1}\cdots\cdots$$
$$C_{r\times c-1,0}\cdots C_{r\times c-1,n_i-1}\underline{F_{r\times c-1}}O_0O_1\cdots O_{n_{po}-1},$$

where the genotype length is $L = r\times c\times(n_i+1)+n_{po}$. To control the connectivity of the graph encoded, the parameter l, called levels-back, restricts the columns from which a node can receive its inputs. For instance, with $l = 1$, a node can only get its inputs from the immediately adjacent left column or from primary inputs. When one wishes to allow nodes to connect to any nodes to their left, then $l = c$.

The genotype is subsequently translated into the corresponding phenotype, thereby yielding a computational structure or program. This modular representation facilitates the efficient exploration of a vast search space, enabling CGP to effectively address problems with high dimensionality and complexity, including mathematical equations, software programs, neural networks, and digital circuits. Taking the digital circuit as an example, Fig. 4 illustrates the process of decoding the CGP genotype for the circuit to the corresponding phenotype. To achieve efficient search, $(1+\lambda)$–evolutionary strategy is typically applied in CGP, as outlined in Algorithm 1. The key point of this strategy is to mutate a single parent genotype to generate λ offspring. First, the parent individual p is initialized (line 1). Then, the population P of the size λ is generated by mutating the parent p (line 3), and the offspring individual c with the best fitness is selected (line 4). If the offspring c's fitness is equal to or better than the parent p, the offspring

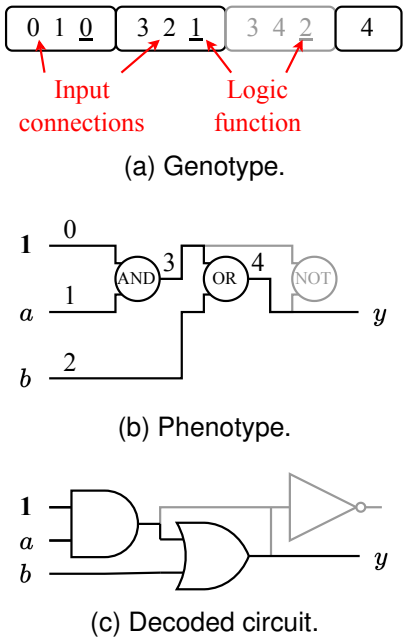

Fig. 4. Illustration of the decoding process of the CGP genotype to the corresponding phenotype with parameters $n_{pi} = 3, n_{po} = 4, n_i = 2, n_o = 1$, c = 3, r = 1, l = 3, Γ = {0:AND, 1:OR, 2:NOT}. This CGP individual presents a circuit with the function $y = a + b$, where the identifiers **1**, a, and b refer to the three primary inputs with indices 0, 1, and 2, and the identifier y stands for the primary output of the circuit.

Algorithm 1: Standard $(1+\lambda)$-CGP search algorithm.

Input: CGP parameters.
Output: The best individual p.
1 Initialize the parent p.
2 **do**
3 $P \leftarrow \{p'_1 = mutate(p), ..., p'_\lambda = mutate(p)\}$.
4 $c \leftarrow$ select the individual with the best fitness in P.
5 **if** $fitness(c) \geq fitness(p)$ **then**
6 $p \leftarrow c$.
7 **while** *termination criterion not met*;
8 **return** p.

c is chosen as the parent. Finally, the above steps (lines 3-6) are repeated until the termination criterion is met, such as up to the maximum population size N. In this way, the best individual can be obtained.

CGP has emerged as a highly effective evolutionary technique within the realm of evolutionary algorithm-based logic synthesis and optimization [31], [32]. Since its introduction, it has produced numerous competitive outcomes in general circuit design. Notably, the scalability of CGP has improved significantly with the advent of SAT-based CGP in 2011 [33], which addresses the computationally intensive nature of exhaustive circuit simulations needed to evaluate the Hamming distance between candidate solutions and specified targets. To expedite the evolution of complex circuits, a binary decision diagram (BDD)–based fitness function has been employed [34]. Furthermore, the integration of circuit simulation with formal verification [35] facilitates the optimization of combinational circuits comprising hundreds of inputs and thousands of gates. In addition, more complex real-world scenarios involving millions of gates can be optimized using windowing techniques [36]. Consequently, this paper utilizes CGP as a foundational framework for developing an efficient automatic synthesis algorithm for RQFP logic, aimed at addressing the challenges inherent in generating RQFP logic circuits.

III. Problem Formulation

The primary focus of this paper is how to generate a legal RQFP logic circuit for a given function. Our objective is to minimize both the number of RQFP logic gates and the number of garbage outputs while meeting the requirements of RQFP circuit design. Therefore, the problem for RQFP circuit generation can be formulated as follows:

- Input:
  1) A given function with n_{pi} primary inputs and n_{po} primary outputs.
- Output:
  A legal RQFP logic circuit $G(V, E)$ with n_r RQFP logic gates and n_g garbage outputs, where V is the gate set, and E is the edge set.
- Constraints:
  1) Function equivalence: The generated circuit G can achieve the given function.

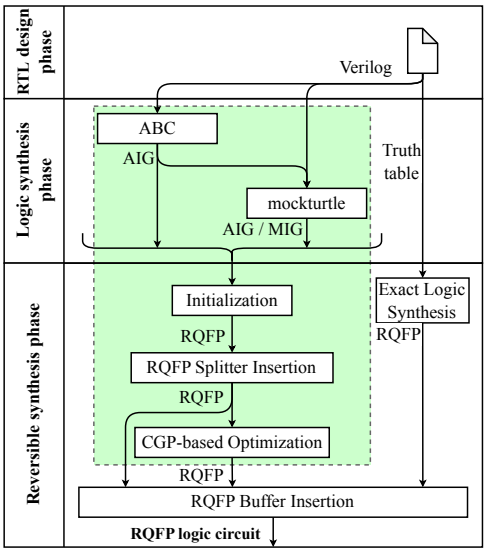

Fig. 5. The flow of RQFP logic circuit generation, where lines represent interfaces between files and tools or methods.

2) Fan-out limitation:

$$\forall e \in E, |e_t| = 1, \tag{1}$$

which ensures that each edge in circuit G has only two pins, including one source and one sink.

3) Clock-synchronized data propagation:

$$\forall u \in \mathrm{FI}(v), v \in V, L(v) = L(u) + 1, \tag{2}$$

which ensures that all inputs to each gate within circuit G have the same logic level.

Hence, Equations (1) and (2) guarantee the final circuit is legal after RQFP splitter and buffer insertion.

- Goal:

$$\min_G \langle n_r, n_g \rangle, \tag{3}$$

which means minimizing the number n_g of garbage outputs on the basis of minimizing the number n_r of RQFP logic gates for the generated circuit G.

IV. Automatic Synthesis Framework

RQFP logic is characterized by distinctive characteristics that pose significant challenges in the design of RQFP logic circuits. Specifically, an RQFP logic gate possesses three output ports, each capable of independently executing a majority function. In addition, RQFP logic circuits must adhere to the single fan-out limitation and the requirement for clock-synchronized data propagation, which necessitates the insertion of RQFP splitters and buffers. These distinctive features inherently contribute to the complexity involved in the design of RQFP logic circuits. In light of these challenges, this section proposes a comprehensive end-to-end framework for the generation of RQFP logic circuits, as illustrated in Fig. 5. In contrast to the existing exact synthesis method [10] that generates the corresponding RQFP logic circuit directly from a truth table in a single step, the proposed framework decomposes the entire process into multiple phases. This

phased approach facilitates the gradual optimization of the RQFP logic circuit, thereby enhancing its scalability. Initially, conventional logic synthesis tools are utilized to process the RTL input and optimize the intermediate circuit. The generated circuit is then directly transformed into an RQFP logic netlist. Subsequently, RQFP splitters are inserted to address the fan-out limitation, yielding an initial RQFP logic network without multiple fan-outs. To optimize this network, an RQFP-oriented CGP method is then proposed, aiming to minimize both the number of RQFP logic gates and the number of garbage outputs. Finally, after the insertion of RQFP buffers, a legal RQFP logic circuit can be achieved.

A. Initialization

To accommodate the RTL description inputs in various standard formats, such as `Verilog`, `AIGER`, and `BLIF`, our framework initially integrates several widely-used open-source logic synthesis tools, specifically ABC [37] and mockturtle [38], to process these inputs automatically. Both mockturtle and ABC are adept at optimizing networks based on the AND-inverter graph (AIG), while mockturtle additionally supports optimization for networks based on the majority-inverter graph (MIG). The choice of both these tools and their corresponding optimization methods is configurable within the framework.

Following the application of these logic synthesis tools, optimized networks based on either AIG or MIG are generated. Given that the introduction of constant inputs can facilitate the realization of AND, OR, NOT, and majority functions through RQFP logic gates, the AIG or MIG-based network can be straightforwardly transformed into an RQFP logic netlist. For instance, the introduction of a constant **1** enables the realization of the AND function, specifically expressed as $R(a,b,1) = \{\mathrm{M}(\overline{a},b,1), \mathrm{M}(a,\overline{b},1), \mathrm{M}(a,b,\overline{1})\} = \{\overline{a}+b, a+\overline{b}, ab\}$, where the last output corresponds to the AND function.

Furthermore, not all inverter configurations for an RQFP logic gate can make the gate logically reversible. To ensure logical reversibility, there must be a one-to-one mapping between vectors of primary inputs and primary outputs for each RQFP logic gate. This requires that any two outputs in the truth table of the RQFP logic gate with inverter configuration ic must be different. Specifically, for any two distinct data input vectors d_1 and d_2 with respective indices $x_1 \in [0,8)$ and $x_2 \in [0,8)$, $x_1 \neq x_2$ in the truth table, the following equation must be true:

$$\bigvee_{i\in[0,3)} \left(\mathop{\mathrm{M}}_{j\in[0,3)} (ic[i][j] \oplus d_1[j]) \oplus \mathop{\mathrm{M}}_{j\in[0,3)} (ic[i][j] \oplus d_2[j]) \right), \tag{4}$$

where $d_1 = \{x_1 \wedge 1, (x_1 \gg 1) \wedge 1, (x_1 \gg 2) \wedge 1\}$, $d_2 = \{x_2 \wedge 1, (x_2 \gg 1) \wedge 1, (x_2 \gg 2) \wedge 1\}$, and $ic[i][j]$ represents whether an inverter exists in front of the j^{th} input port of the i^{th} majority within RQFP logic gate.

B. RQFP Splitter Insertion

After generating the initial RQFP logic netlist using RQFP logic gates with logical reversibility, a lot of multiple fan-outs may exist. To mitigate the complexity of RQFP logic optimization, RQFP splitters are inserted into the initial RQFP logic netlist ahead of time to meet the fan-out limitation. In this way, the initialization process of RQFP logic circuits is completed.

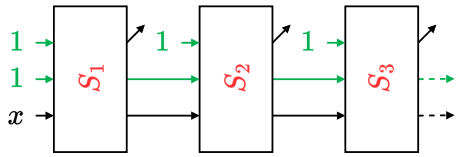

Fig. 6. Schematic of the chained RQFP splitter tree for a multi-fan-out signal x, where the green (black) line represents the transmission of the constant (signal x), and gates S_1,S_2, and S_3 have the same inverter configuration, *i.e.*$R(1,1,x) = \{\mathrm{M}(\overline{1},1,x), \mathrm{M}(1,1,\overline{x}), \mathrm{M}(1,\overline{1},x)\} = \{x,1,x\}$.

Theorem 1. *Given an RQFP logic gate, no matter what inverter configuration it has, it cannot function as a one-to-three RQFP splitter with logical reversibility.*

Proof. Assume that there exists an RQFP logic gate g, which functions as a one-to-three RQFP splitter for its data input x. So, all three output ports of gate g output x. For simplicity, let the other two inputs of gate g be constant **1**, *i.e.*, $R(1,x,1) = \{x,x,x\}$. According to the definition of the majority function, to make each majority gate within gate g output x, its other two inputs must be complementary, such as $R(1,x,1) = \{\mathrm{M}(\overline{1},x,1), \mathrm{M}(\overline{1},x,1), \mathrm{M}(\overline{1},x,1)\}$. In this inverter configuration, we cannot distinguish $R(0,0,1)$ from $R(0,1,1)$, *i.e.*, $R(0,0,1) = \{1,1,1\} = R(0,1,1)$, which contradicts with the requirement of logical reversibility. Therefore, the one-to-three RQFP splitter constructed by an RQFP logic gate does not exist. □

However, to ensure logical reversibility, the one-to-three RQFP splitter cannot be constructed using the current RQFP logic gate, as illustrated in Theorem 1. Hence, we can only use one-to-two RQFP splitters to implement multiple fan-outs, where each splitter introduces one additional garbage output. To reduce the number of garbage outputs caused by the insertion of RQFP splitters, a chained RQFP splitter tree structure is adopted, as illustrated in Fig. 6. In this structure, the constant output of the parent node serves as a constant input of its successor, which makes each chained RQFP splitter tree introduce at most one garbage output in total.

C. CGP-based Optimization

Definition 1 (Functional Symmetry)**.** *If two RQFP logic gates produce the same functional outputs but differ in their corresponding output port indices, then the two RQFP logic gates have functional symmetry. For instance, two gates* $R_1(a,b,c) = \{x,y,z\}$ *and* $R_2(a,b,c) = \{x,z,y\}$ *have functional symmetry.*

1) CGP Encoding.: To optimize an RQFP logic circuit, the process begins with the introduction of CGP encoding to represent an RQFP logic circuit. Typically, CGP encodes a candidate solution using an integer array that represents programmable nodes organized in c columns and r rows. For circuit optimization purposes, a linear form of CGP, where

$r = 1$, is usually preferred. Each programmable node in the CGP representation possesses a fixed number of inputs, n_i, and outputs, n_o, and can implement one of $n_f = |\Gamma|$ predefined primitive functions. Hence, in the context of RQFP circuits, each node corresponds to an RQFP logic gate, with both n_i and n_o set to 3. Inputs to each node can be connected to the output ports of any node within the preceding l columns or to one of the n_{pi} primary inputs. Notably, due to the complexity of evaluation, cycles are not allowed in the standard version of CGP, thereby preventing feedback loops from a node to its successors. In addition, since the function of each RQFP logic gate depends on its inverter configuration, there are a total of $2^9 = 512$ possible functions, of which 192 are logically reversible. Furthermore, due to the functional symmetry existing in RQFP logic gates, illustrated in Definition 1, the permutation of connection order can yield equivalent results, thereby resulting in a final selection of 32 unique functions, *i.e.*, $n_f = 32$. Therefore, a candidate circuit is encoded as a CGP genotype consisting of $L = c \times r \times (n_i + 1) + n_{po}$ integers, where 1 represents each node requires one integer to record its function index and the last n_{po} integers record the input indices corresponding to the primary outputs.

Since the positions of input ports within each RQFP logic gate are fixed and directly affect its functional output, the CGP encoding must adapt to suit RQFP logic. For instance, Fig. 7(a) illustrates the CGP encoding of a 2-to-4 decoder, where $c = 4$ and $r = 1$. The decoder has $n_{pi} = 2$ primary inputs, x_0 and x_1, and $n_{po} = 4$ primary outputs, y_0, y_1, y_2 and y_3. Due to the requirement of RQFP splitters for constant inputs, a constant **1** is introduced and indexed at 0, with primary inputs indexed from 1 to n_{pi}. Specifically, the long green integer string at the bottom of Fig. 7(a) represents the CGP encoding corresponding to an initial RQFP logic circuit for the 2-to-4 decoder above it. The integer substring within each pair of parentheses in the CGP encoding signifies an RQFP node. For example, "(5, 4, 0, 101-100-000)" represents the CGP encoding of the second RQFP logic gate, where "5, 4, 0" specifies the interconnection of its input ports with the output ports indexed 5, 4, 0 (the constant input), respectively, and "101-100-000" specifies the inverter configuration before its three AQFP majority components. The inverter configuration is represented by a 9-bit integer, where each bit indicates the presence of an inverter at the corresponding input port. For instance, "100" indicates an inverter exists before the first input port of the second AQFP majority within the second RQFP logic gate. Additionally, the last item "(6, 10, 13, 14)" details the indices of output ports connected to primary outputs, where "14" represents that the primary output y_3 is connected to the third output port of the third RQFP logic gate.

2) CGP Evaluation.: Once a candidate circuit has been encoded, its evaluation is imperative. In our framework, the evaluation of the fitness value of a CGP individual consists of two distinct phases. The first phase involves function evaluation, which determines the number of correct primary output bits produced in response to all possible assignments of the primary inputs. The second phase is performance evaluation, which assesses the number of RQFP logic gates and garbage outputs. The evaluation process is detailed as follows:

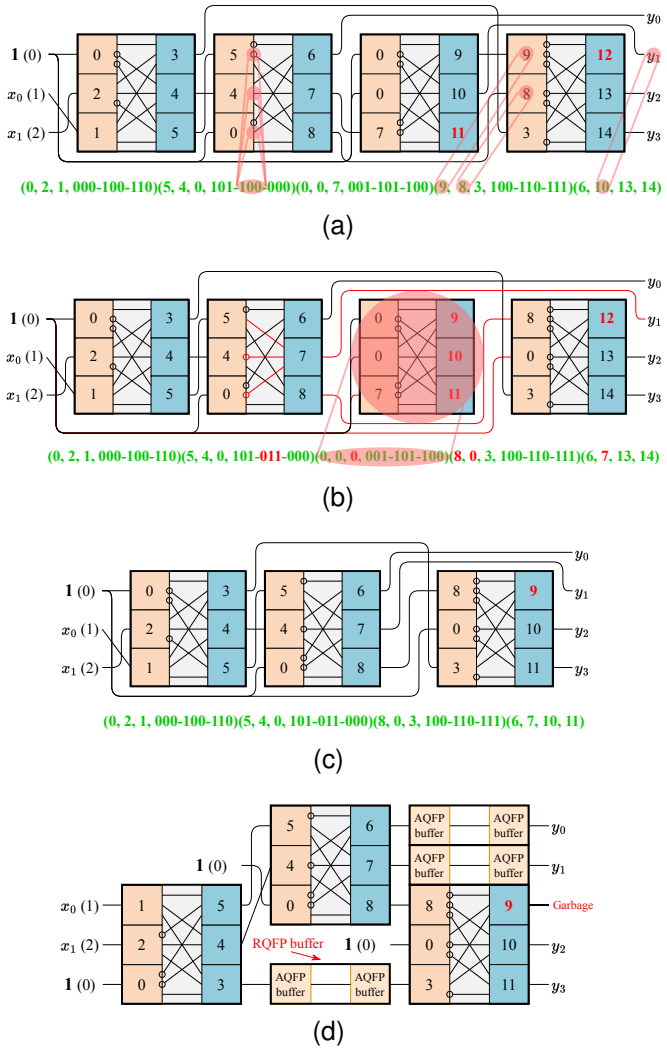

Fig. 7. Schematics of CGP individuals encoding a 2-to-4 decoder circuit with $n_{pi} = 2$ primary inputs and $n_{po} = 4$ primary outputs. (a) shows a CGP individual with $n_r = 4$ RQFP logic gates and $n_g = 2$ garbage outputs. (b) is the individual with $n_r = 4$ RQFP logic gates (including one useless gate) and $n_g = 4$ garbage outputs after the mutation of (a). (c) is the final individual with $n_r = 3$ RQFP logic gates and $n_g = 1$ garbage outputs after the removal of useless gates in (b). (d) displays the final RQFP logic circuit after RQFP buffer insertion for (c).

(a) The performance evaluation is conducted only when the circuit function is satisfied, specifically when the number of correct primary output bits equals $n_{po} \times 2^{n_{pi}}$. This condition ensures the functional validity of the solution.
(b) Subsequently, the performance evaluation prioritizes optimizing the number of RQFP logic gates, with a secondary objective of minimizing the number of garbage outputs.

Additionally, while maintaining the optimal number of RQFP logic gates and garbage outputs, our framework also considers the reduction of RQFP buffers required for the clock-synchronized propagation of data.

3) CGP Mutation.: Following the construction of the CGP individual, CGP mutation must occur to produce its offspring. Point mutation is typically preferred due to its high efficiency. This mutation process randomly alters up to m genes (integers) of a parent genotype to create an offspring, where m is determined by the mutation rate $\mu, \mu \in [0, 1]$. To ensure that

each gene has an opportunity for random modification during each mutation, our framework sets the maximum of m to $\mu * L$, where L is the length of the genotype.

In the context of CGP encoding, a single gene mutated from g to g' causes either a node reconnection or an inverter configuration change. Given the fan-out limitation of RQFP logic, three scenarios may arise concerning node reconnection. Notably, the initialization process has ensured the single fan-out limitation of the parent genotype.

(a) When the output port corresponding to the mutated value g' has been connected to another node v, and the node with output port index g is on the left of node v, the values of these two genes need to be swapped. For instance, in Fig. 7(a), assume that $g = 9$ in the last second item "(9, 8, 3, 100-110-111)" mutates to $g' = 8$, and then the output port corresponding to '8' is already connected to the second input port of the last node. Therefore, the swap operation is executed, resulting in "(8, 9, 3, 100-110-111)".

(b) When the output port corresponding to the mutated value g' is the constant input **1** or has no fan-out, g' is directly assigned to this gene g. For example, after mutating to "(8, 9, 3, 100-110-111)", assume that $g = 9$ further mutates to $g' = 0$ (the constant input **1**), the genotype of the last node becomes "(8, 0, 3, 100-110-111)". This indicates that the second input port of the last node is now connected to the constant input **1**, as shown in Fig. 7(b).

(c) When the output port corresponding to the mutated value g' has been connected to another node v, and the node with output port index g is not on the left of node v, g' is directly assigned to this gene g, and the gene of the corresponding input port of node v is updated to '0', meaning this input port is connected to the constant input **1**. As shown in Fig. 7(a), assume that $g = 10$ in the last item "(6, 10, 13, 14)" mutates to $g' = 7$, and then the output port corresponding to '7' is already connected to the third input port of the last second node v while the node with output port index g is not on the left of node v. Therefore, let $g = g'$, and the third input port of the last second node is connected to the constant input **1**, as shown in Fig. 7(b).

Moreover, for the change of an inverter configuration f, the mutation will produce an integer $\beta \in [0, n_f)$ to update f to the function with index β in the function look-up table. As shown in Fig. 7(a), the initial inverter configuration of the second node is "101-100-000" with index 352 and then is updated to "101-011-000" with index 344 through one mutation, as shown in Fig. 7(b). In this way, the output function of the second majority within the second node is updated.

4) CGP Contraction.: Subsequent to CGP mutation, it is possible for certain useless nodes to persist in the phenotype. The removal of these useless nodes does not alter the functionality associated with the phenotype. These nodes can be classified into two primary categories:

(a) Nodes that exhibit no fan-outs;

(b) Nodes for which all three output ports serve as constants.

For nodes belonging to the second category, their corresponding fan-outs are redirected to the constant input **1**, effectively converting them into first-category nodes without any fan-out connections. Since first-category nodes have no fan-outs, they are useless and can then be directly eliminated, thereby reducing the genotype size without affecting the inputs of subsequent nodes. For example, the third node with genes "(0, 0, 7, 001-101-100)" in Fig. 7(b), has no fan-out and is thus removed in Fig. 7(c). This elimination of useless nodes highlights a significant advantage of CGP encoding: the phenotype size can vary despite a fixed genotype size, as not all nodes are necessary. Since our framework is designed to minimize both the number of RQFP logic gates and garbage outputs, these useless nodes can be discarded to compact the genotype, thereby narrowing the search space. As a result, the genotype length for the 2-to-4 decoder is reduced from 20 to 16, as illustrated in Fig. 7(c).

Algorithm 2: The flow of CGP-based optimization.

Input: Initial individual p, total number N of generations, mutation rate μ, number λ of offspring.

Output: Optimized individual.

1 Calculate the functional fitness f of c.
2 $f_n \leftarrow f, p_n \leftarrow p, n_r \leftarrow \infty, n_g \leftarrow \infty, n_b \leftarrow \infty$.
3 **if** $f = n_{po} \times 2^{n_{pi}}$ **then**
4 Remove useless nodes to shrink p_n.
5 Calculate the number (n_r, n_g, n_b) of RQFP logic gates, garbage outputs, and RQFP buffers in p_n.
6 **for** $i \in [1, N]$ **do**
7 $p \leftarrow p_n$.
8 **for** $j \in [1, \lambda]$ **do**
9 Perform the mutation on p to generate an offspring p'.
10 Calculate the functional fitness f' of p'.
11 **if** $f' = 1$ **then**
12 Remove useless nodes to shrink p'.
13 Calculate the number (n'_r, n'_g, n'_b) of gates, garbage outputs, and buffers in p'.
14 **if** $(n_r, n_g, n_b) > (n'_r, n'_g, n'_b)$ **then**
15 $(f_n, p_n, n_r, n_g, n_b) \leftarrow (f', p', n'_r, n'_g, n'_b)$.
16 **else if** $f' > f_n$ **then**
17 $(f_n, p_n) = (f', p')$.
18 **return** p_n.

5) Flow of CGP-based Optimization.: Algorithm 2 shows the comprehensive process of CGP-based optimization, which employs a typical $(1 + \lambda)$ evolutionary strategy [30]. Initially, it evaluates the initial individual p and regards p as the current best parent p_n (lines 2-5). Subsequently, it mutates the best parent genotype to generate λ offspring within each generation (lines 7-9). An offspring exhibiting a fitness equal to or superior to that of the parent becomes the new parent for the subsequent generation (lines 10-17). After N generations, the optimal offspring is returned (line 18).

TABLE I
EXPERIMENTAL RESULTS ON SMALL CIRCUITS FROM THE REVLIB BENCHMARK [16].

Testcase	Original			Initialization					Exact logic synthesis [10]						RCGP [11]						Ours					
	n_{pi}	n_{po}	g_{lb}	n_r	n_b	n_j	n_d	n_g	n_r	n_b	n_j	n_d	n_g	t (s)	n_r	n_b	n_j	n_d	n_g	t (s)	n_r	n_b	n_j	n_d	n_g	t (s)
1-bit full adder	3	2	1	8	7	220	5	10	3	3	84	3	3	14.51	3	3	84	3	3	103.39	3	3	84	3	3	113.04
4gt10	4	1	3	3	3	84	3	6	3	3	84	3	6	18.44	3	3	84	3	6	120.54	3	3	84	3	6	98.36
alu	5	1	4	14	17	404	6	17	4	6	120	4	5	96.16	4	6	120	4	5	173.56	4	6	120	4	5	134.47
c17	5	2	3	13	12	360	5	17				\			7	6	192	4	8	301.46	7	20	248	6	6	221.84
decoder_2_4	2	4	0	10	5	260	4	10	3	5	92	3	1	10.22	3	4	88	3	2	124.47	3	3	84	3	1	128.46
decoder_3_8	3	8	0	29	33	828	8	31	7	25	268	7	1	93237.87	10	22	328	7	3	524.42	7	16	232	5	1	264.68
graycode4	4	4	0	18	13	484	5	22				\			6	13	196	6	2	283.37	6	4	160	4	2	216.86
ham3	3	3	0	23	23	644	7	24	5	3	132	5	1	192.7	5	3	132	5	1	231.45	5	3	132	5	1	233.13
mux4	6	1	5	13	14	368	6	16				\			8	13	244	6	9	354.79	6	3	156	4	7	200.55

* '\' represents that the exact logic synthesis method can not find a feasible solution within 240,000 seconds.

D. RQFP Buffer Insertion

Following the CGP-based optimization, it is possible that certain RQFP logic gates within the generated RQFP logic circuit may not fulfill the requirement for clock-synchronized data propagation. Consequently, the insertion of RQFP buffers becomes essential to ensure uniform clock phases for all inputs to each gate. Specifically, a requisite number of RQFP buffers must be inserted into each edge, corresponding to the logic level gap between its connected RQFP logic gates. Fig. 7(d) illustrates the buffer insertion result for the RQFP logic circuit shown in Fig. 7(c) generated for the 2-4 decoder.

V. EXPERIMENTAL RESULTS

The proposed framework for automatic RQFP logic circuit generation was implemented using C++ and Python. For evaluation, we utilized circuits from the RevLib benchmark [16] and the reversible reciprocal circuits [17], both of which are specifically designed for reversible logic. For parameter configuration, in the logic synthesis phase, our framework initially employed the "`resyn2`" command from the ABC [37] to generate an optimized AIG network according to the truth table of a given circuit, and then applied the "`aqfp_resynthesis`" command from the mockturtle to convert the AIG network into an AQFP-oriented MIG network. This command utilizes a cutting-edge MIG-based logic optimization technique for AQFP logic [21]. In the reversible synthesis phase, the number N of generations was set to 50,000,000, and the mutation rate μ was set to 1. The number c of columns was set to the number of RQFP logic gates in the generated initial RQFP logic netlist, while l was equal to c, and r was set to 1. The experiments were conducted on a machine equipped with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and 256.0 GB memory, with Ubuntu 22.04 as the operating system.

We compared our framework against the following three baselines:

- The first baseline employed a transformation approach and was a part of our framework. As shown in Fig. 5, after the logic synthesis phase, it involved initialization, followed by RQFP splitter and buffer insertion, to meet clock-synchronization requirements and generate RQFP logic circuits.
- The second baseline utilized an existing exact logic synthesis method [10] with the Z3 solver [39]. It can directly generate the RQFP logic circuit with a given number of RQFP logic gates and garbage outputs from the input truth table. In addition, it incorporates specific constraints as illustrated in Equation (4) to ensure the logical reversibility of generated RQFP logic circuits.
- The third baseline, RCGP [11], represents our framework prior to optimizations.

In addition, to maintain consistency in all experimental results regarding the clock-synchronized data propagation requirement for primary inputs and primary outputs, both primary inputs and primary outputs require RQFP buffer insertion to align them within the same clock stage, respectively.

TABLE I presents the experimental results for small circuits from the RevLib benchmark. The "`Original`" part shows the characteristics of input circuits, including the number n_{pi} of primary inputs, the number n_{po} of primary outputs, and the lower bound $g_{lb} = \max(0, n_{pi} - n_{po})$ of garbage outputs [10]. The "`Initialization`" part shows the results from the first baseline, where n_r denotes the number of used RQFP logic gates, and n_b denotes the number of RQFP buffers inserted for the clock-synchronized data propagation requirement. n_j denotes the number of Josephson junctions (JJs). Since JJs are critical components of AQFP circuits and are used to create the fluxons used for computation, their count serves as an estimate of the complexity and energy efficiency of the AQFP circuit. Hence, the number of JJs can also be used as a cost metric of RQFP logic circuits realized by AQFP logic. Since both a buffer and a splitter have 2 JJs, and a 3-input MAJ has 6 JJs in current AQFP logic circuits, the numbers of JJs for each RQFP logic gate and RQFP buffer are 24 and 4, respectively. n_d and n_g represent the circuit depth and the number of garbage outputs in generated RQFP logic circuits, respectively. The "`Exact logic synthesis`" part shows the results of the exact logic synthesis method [10] for RQFP logic, including the runtime "t (s)" in seconds. The "`RCGP`" and "`Ours`" parts show the results of RCGP [11] and our proposed framework, respectively.

TABLE II
EXPERIMENTAL RESULTS ON LARGE CIRCUITS FROM THE REVLIB BENCHMARK [16] AND THE REVERSIBLE RECIPROCAL CIRCUITS [17].

Testcase	Original			Initialization					RCGP [11]						Ours					
	n_{pi}	n_{po}	g_{lb}	n_r	n_b	n_j	n_d	n_g	n_r	n_b	n_j	n_d	n_g	t (s)	n_r	n_b	n_j	n_d	n_g	t (s)
4_49_7	4	4	0	48	103	1564	11	42	23	101	956	16	15	1024.35	24	45	756	9	13	707.07
graycode6_11	6	6	0	30	17	788	5	36	12	47	476	10	3	542.12	11	17	332	6	3	333.25
mod5adder_66	6	6	0	202	1242	9816	34	195	133	1763	10244	60	78	6114.78	101	288	3576	16	61	3267.38
hwb8_64	8	8	0	2129	40254	212112	128	2037	2118	46673	237524	116	1846	124268.44	2027	10602	91056	39	1818	96009.00
intdiv4	4	4	0	36	35	1044	9	37	18	54	648	11	10	721.05	14	18	408	6	9	685.99
intdiv5	5	5	0	71	140	2264	13	73	47	289	2284	25	26	1910.51	37	79	1204	10	22	1071.14
intdiv6	6	6	0	154	538	5848	21	152	83	788	5144	46	41	3928.42	73	184	2488	14	45	2354.22
intdiv7	7	7	0	293	1947	14820	37	282	197	3222	17616	79	115	9460.69	140	511	5404	18	86	4945.98
intdiv8	8	8	0	562	5701	36292	59	554	400	5928	33312	70	266	19339.97	293	1459	12868	25	187	10554.15
intdiv9	9	9	0	1054	13670	79976	78	1024	812	10896	63072	72	617	47004.98	507	1920	19848	24	380	22717.25
intdiv10	10	10	0	1815	32275	172660	115	1784	1402	18506	107672	71	1211	127957.25	1119	5972	50744	33	960	100127.44

Compared to the first baseline, our proposed framework achieved a notable reduction in RQFP logic gates, JJs, and garbage outputs, specifically by 58.30%, 56.33%, and 70.56%, respectively. Although the exact logic synthesis can generate the RQFP logic circuit with the optimal number of RQFP logic gates and garbage outputs, our framework can yield near-optimal results with less runtime. Moreover, the exact logic synthesis failed to find feasible solutions for the circuits "c17", "graycode4", and "mux4" within sufficiently given 240,000 seconds. Furthermore, compared with RCGP [11], our framework also demonstrated reductions in RQFP logic gates, JJs, and garbage outputs by 6.11%, 6.56%, and 18.21%, respectively, while achieving an 18.95% reduction in runtime. These results suggest that our proposed framework can effectively generate RQFP logic circuits with a near-optimal number of RQFP logic gates and garbage outputs for small benchmark circuits.

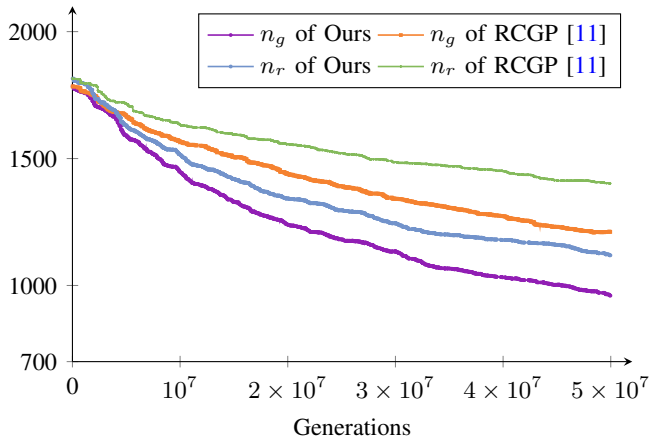

Fig. 8. The number of RQFP logic gates, n_r, and garbage outputs, n_g, for `intdiv10` generated by RCGP [11] and our framework varies with generations.

Table II presents the experimental results for large circuits from the RevLib benchmark and reversible reciprocal circuits. Notably, the exact logic synthesis method was unable to provide feasible solutions for these circuits under a given sufficient time, underscoring the importance of our framework. Compared to the first baseline, our proposed framework significantly reduced the number of RQFP logic gates and the number of garbage outputs, specifically by 47.28% and 63.72%, respectively. When compared with the RCGP [11], our proposed framework achieved reductions in these counts by 18.30% and 15.12%, respectively. Furthermore, we also compared the search processes of RCGP and our framework. Taking `intdiv10` as an example, Fig. 8 shows the number of RQFP logic gates and garbage outputs for `intdiv10` varies across generations for RCGP [11] and our framework. It is evident that our framework can find superior solutions more rapidly. Therefore, these experimental results demonstrate that our proposed framework can effectively and efficiently generate large RQFP logic circuits with fewer RQFP logic gates and garbage outputs.

VI. CONCLUSION

This paper presented a novel automatic synthesis framework for RQFP logic based on Cartesian genetic programming. Our framework leverages CGP encoding to effectively present RQFP logic circuits and integrates circuit simulation with formal verification to evaluate the functional equivalence between the parent and offspring circuits. Through RQFP-oriented mutations, our framework optimizes the number of gates and garbage outputs, while ensuring the logical reversibility of the generated RQFP logic circuits by imposing restrictions on inverter configurations. The experimental results on both the RevLib benchmark and reversible reciprocal circuits demonstrated the effectiveness of our framework in generating RQFP logic gates. Moreover, our framework approaches the optimality of the exact synthesis method but with greater efficiency, particularly for large circuits where the exact synthesis method becomes impractical due to computational constraints. These findings underscore the potential of our framework as a robust tool for optimizing RQFP logic circuits, addressing the challenges of scalability and efficiency in logic synthesis.

Furthermore, due to the random perturbations involved in

the mutation process, our framework still requires a significant runtime. To enhance the efficiency and effectiveness of the mutation process, future work will integrate advanced machine learning (ML) techniques into the framework. This integration aims to accelerate the exploration of feasible solutions by leveraging the predictive and adaptive capabilities of ML methods.

REFERENCES

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.

[2] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[3] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.

[4] A. Zulehner, M. P. Frank, and R. Wille, "Design automation for adiabatic circuits," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019, pp. 669–674.

[5] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Reversible logic gate using adiabatic superconducting devices," *Scientific Reports*, vol. 4, p. 6354, 2014.

[6] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Reversibility and energy dissipation in adiabatic superconductor logic," *Scientific Reports*, vol. 7, p. 75, 2017.

[7] T. Yamae, N. Takeuchi, and N. Yoshikawa, "A reversible full adder using adiabatic superconductor logic," *Superconductor Science and Technology*, vol. 32, no. 3, p. 035005, 2019.

[8] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Recent progress on reversible quantum-flux-parametron for superconductor reversible computing," *IEICE Transactions on Electronics*, vol. E101.C, no. 5, pp. 352–358, 2018.

[9] N. Takeuchi, T. Yamae, C. L. Ayala, H. Suzuki, and N. Yoshikawa, "Adiabatic quantum-flux-parametron: A tutorial review," *IEICE Transactions on Electronics*, vol. E105.C, no. 6, pp. 251–263, 2022.

[10] R. Fu, O. Chen, N. Yoshikawa, and T.-Y. Ho, "Exact logic synthesis for reversible quantum-flux-parametron logic," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–9.

[11] R. Fu, R. Wille, and T.-Y. Ho, "RCGP: An automatic synthesis framework for reversible quantum-flux-parametron logic circuits based on efficient cartesian genetic programming," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[12] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An adiabatic quantum flux parametron as an ultra-low-power logic device," *Superconductor Science and Technology*, vol. 26, no. 3, p. 035010, 2013.

[13] M. Hosoya, W. Hioe, J. Casas, R. Kamikawai, Y. Harada, Y. Wada, H. Nakane, R. Suda, and E. Goto, "Quantum flux parametron: a single quantum flux device for josephson supercomputer," *IEEE Transactions on Applied Superconductivity (TASC)*, vol. 1, no. 2, pp. 77–89, 1991.

[14] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, vol. 85, 1980, pp. 632–644.

[15] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.

[16] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *International Symposium on Multiple Valued Logic (ISMVL)*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.

[17] M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "Design automation and design space exploration for quantum computers," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2017, pp. 470–475.

[18] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6-7, pp. 467–488, 1982.

[19] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Adiabatic quantum-flux-parametron cell library adopting minimalist design," *Journal of Applied Physics*, vol. 117, no. 17, p. 173912, 05 2015.

[20] R. Cai, O. Chen, A. Ren, N. Liu, N. Yoshikawa, and Y. Wang, "A buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits," in *IEEE International Conference on Computer Design (ICCD)*, 2019, pp. 429–436.

[21] G. Meuli, V. Possani, R. Singh, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage, P. Vuillod, L. Amaru, S. Chase, J. Kawa, and G. De Micheli, "Majority-based design flow for AQFP superconducting family," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2022, pp. 34–39.

[22] R. Fu, J. Huang, M. Wang, Y. Nobuyuki, B. Yu, T.-Y. Ho, and O. Chen, "BOMIG: A majority logic synthesis framework for aqfp logic," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2023, pp. 1–2.

[23] C.-Y. Huang, Y.-C. Chang, M.-J. Tsai, and T.-Y. Ho, "An optimal algorithm for splitter and buffer insertion in adiabatic quantum-flux-parametron circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021, pp. 1–8.

[24] S.-Y. Lee, H. Riener, and G. De Micheli, "Beyond local optimality of buffer and splitter insertion for AQFP circuits," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 445–450.

[25] R. Fu, M. Wang, Y. Kan, N. Yoshikawa, T.-Y. Ho, and O. Chen, "A global optimization algorithm for buffer and splitter insertion in adiabatic quantum-flux-parametron circuits," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2023, pp. 769–774.

[26] R. Fu, M. Wang, Y. Kan, O. Chen, N. Yoshikawa, B. Yu, and T.-Y. Ho, "Buffer and splitter insertion for adiabatic quantum-flux-parametron circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[27] Y.-C. Chang, H. Li, O. Chen, Y. Wang, N. Yoshikawa, and T.-Y. Ho, "ASAP: An analytical strategy for AQFP placement," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–7.

[28] P. Dong, Y. Xie, H. Li, M. Sun, O. Chen, N. Yoshikawa, and Y. Wang, "TAAS: A timing-aware analytical strategy for AQFP-capable placement automation," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1321–1326.

[29] R. Fu, O. Chen, B. Yu, N. Yoshikawa, and T.-Y. Ho, "DLPlace: A delay-line clocking-based placement framework for AQFP circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–8.

[30] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 121–132.

[31] J. F. Miller, "Cartesian genetic programming: its status and future," *Genetic Programming and Evolvable Machines*, vol. 21, pp. 129 – 168, 2019.

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

[33] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, pp. 305–327, 2011.

[34] Z. Vasicek and L. Sekanina, "How to evolve complex combinational circuits from scratch?" in *IEEE International Conference on Evolvable Systems (ICES)*, 2014, pp. 133–140.

[35] Z. Vasicek, "Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates," in *Genetic Programming*, 2015, pp. 139–150.

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

[37] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer-Aided Verification (CAV)*, 2010, pp. 24–40.

[38] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, and G. Marakkalage, Dewmini Sudara De Micheli, "The EPFL logic synthesis libraries," 2022, arXiv:1805.05121v3.

[39] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

Rongliang Fu received his BS degree in software engineering from the Northwestern Polytechnical University, Xi'an, China, in 2018 and his MS degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently studying for his Ph.D degree in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation and computer architecture.

Robert Wille is a Full and Distinguished Professor at the Technical University of Munich, Germany, and Chief Scientific Officer at the Software Competence Center Hagenberg, Austria. He received the Diploma and Dr.-Ing. degrees in Computer Science from the University of Bremen, Germany, in 2006 and 2009, respectively. Since then, he worked at the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), the University of Applied Science of Bremen, the University of Potsdam, and the Technical University Dresden. From 2015 until 2022, he was Full Professor at the Johannes Kepler University Linz, Austria, until he moved to Munich. His research interests are in the design of circuits and systems for both conventional and emerging technologies. In these areas, he published more than 400 papers and served in editorial boards as well as program committees of numerous journals/conferences such as TCAD, ASP-DAC, DAC, DATE, and ICCAD. For his research, he was awarded, e.g., with Best Paper Awards, e.g., at TCAD and ICCAD, an ERC Consolidator Grant, a Distinguished and a Lighthouse Professor appointment, a Google Research Award, and more.

Nobuyuki Yoshikawa (F'24) is a professor at the Institute of Advanced Sciences (IAS) at Yokohama National University (YNU), where he leads the superconductivity electronics group. He earned his Ph.D. in Electrical and Computer Engineering (ECE) from YNU in 1989 and has since been affiliated with YNU's ECE Department. His research primarily centers on superconductive devices and their integration into digital and analog circuits. Presently, his focus is on developing highly energy-efficient superconducting digital circuits, particularly those that employ Adiabatic Quantum-Flux Parametron (AQFP) and Single Flux Quantum (SFQ) logic, with an aim towards high-performance computing applications. In 2023, he was honored with the IEEE Council on Superconductivity (CSC) Award for Continuing and Significant Contributions in the field of Applied Superconductivity. He is also a Fellow of the IEEE.

Tsung-Yi Ho (F'24) is a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.