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

Rongliang Fu
rlfu@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR, China

Robert Wille
robert.wille@tum.de
Technical University of Munich
Munich, Germany

Tsung-Yi Ho
tyho@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR, China

Abstract

Reversible computing has gained increasing attention as a prospective solution for energy dissipation, particularly in quantum computing. As the first practical reversible logic gate using adiabatic superconducting devices, the reversible quantum-flux-parametron (RQFP) has been experimentally demonstrated in logical and physical reversibility. However, the circuit design of RQFP logic poses enormous challenges due to its distinctive logic function and structure. Furthermore, the circuit scale severely restricts the applicability of the existing exact logic synthesis method for RQFP logic. Therefore, this paper proposes RCGP, an automatic synthesis framework based on efficient Cartesian genetic programming, to generate large RQFP logic circuits. RCGP considers the characteristics of RQFP logic circuits to minimize the number of gates and garbage outputs. Meanwhile, RCGP combines circuit simulation with formal verification to assess the functional equivalence between the parent and its offspring. Experimental results on reversible logic benchmarks demonstrate the effectiveness of RCGP.

1 Introduction

The requirement for reversible computing arises from the fundamental limitations of traditional irreversible logic systems in terms of energy dissipation. In conventional logic systems, such as complementary metal-oxide-semiconductor (CMOS) logic, irreversible operations result in information loss and energy dissipation in the form of heat. According to the Landauer principle in 1961 [1], every bit loss of information is involved in a loss of $k_B T \ln 2$ J of energy, where k_B is the Boltzmann constant and T is the temperature of the system. This heat dissipation becomes significant as the rate of the processor frequency increases. Therefore, reversible computing has gained significant attention as a promising approach to minimizing energy dissipation in logic operations. It can theoretically produce nearly energy-free computation systems by preserving the loss of information [2]. This approach also holds particular significance in quantum computing due to the inherent reversibility of quantum operations [3].

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655950

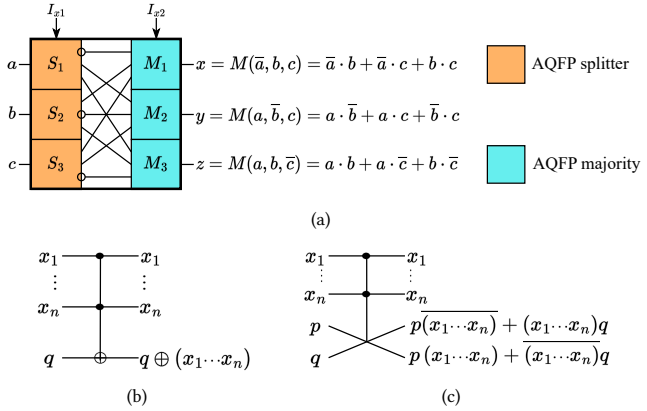

Figure 1: Functional schematics of (a) RQFP logic gate, (b) MCT gate, and (c) MCF gate, where symbols employed include · for logical conjunction, + for logical disjunction, and ¯ for logical negation.

However, the practical implementation of reversible computing presents significant challenges. It requires both logical and physical reversibility, as well as the development of ultra-low power logic devices [4]. Takeuchi *et al.* proposed the reversible quantum-flux-parametron (RQFP) [5], the first practical reversible logic gate using adiabatic superconducting devices. The logical and physical reversibility of RQFP has also been validated through experimental demonstrations [5, 6]. Moreover, an RQFP logic full adder was designed and fabricated by Yamae *et al.* [7], further revealing the reversibility and feasibility of RQFP logic circuits. Therefore, RQFP logic gains increasing attention from researchers [8, 9].

However, the design of RQFP logic circuits is complicated due to the lack of automated design tools [8]. The RQFP logic gate is implemented by adiabatic quantum-flux-parametron (AQFP) [10], an energy-efficient superconductor logic element based on the quantum flux. Fig. 1(a) shows the structure of a normal RQFP logic gate composed of three AQFP splitter gates and three AQFP majority gates. So, the RQFP logic gate has three functional outputs, i.e., $R(a, b, c) = \left\{M(\overline{a}, b, c), M(a, \overline{b}, c), M(a, b, \overline{c})\right\} = \{x, y, z\}$, where a, b, and c are three inputs, x, y, and z are three outputs, and $M(\cdot)$ represents a three-input majority function $M(a, b, c) = ab + ac + bc$. In contrast, a conventional reversible logic circuit primarily consists of basic Toffoli [11] and Fredkin [12] gates, as well as their extensions commonly known as multiple-control Toffoli (MCT) and multiple-control Fredkin (MCF) gate libraries. As shown in Fig. 1(b) and Fig. 1(c), these two kinds of reversible logic gates can be viewed

as multi-controlled NOT and multi-controlled SWAP gates, respectively. It is clear that their output functions mainly focus on the last one or two output ports and realize the XOR-sum-of-products. Furthermore, RQFP logic must also satisfy specific constraints, including the same fan-out limitation and path-balancing requirement as AQFP logic [13–15], since RQFP logic is realized through AQFP logic. Consequently, the logic synthesis methods of conventional reversible logic do not apply to RQFP logic due to these distinctions in logic function and structure. In addition, the energy dissipation of RQFP logic circuits is significantly affected by the number of gates and garbage output. Exact logic synthesis [15] is the only existing method for generating RQFP logic circuits, but it is severely limited by the circuit scale, which has been demonstrated in the experiment. Therefore, the study of practical logic synthesis methods of RQFP logic circuits is vital.

Considering these above challenges, this paper proposes RCGP, an efficient Cartesian genetic programming (CGP)-based automatic synthesis framework for RQFP logic circuits, which aims to optimize the number of RQFP logic gates and garbage outputs. In summary, this paper makes the following contributions:

- This paper proposes a complete logic synthesis framework for RQFP logic circuit generation from RTL descriptions.
- The proposed algorithm employs a dynamic CGP encoding to present an RQFP logic circuit and integrates circuit simulation with formal verification to evaluate the functional equivalence between the parent circuit and its offspring.
- Furthermore, three mutations are proposed to effectively and efficiently generate a legal RQFP logic circuit.
- Experimental results of RCGP on large RevLib circuits [16] and reversible reciprocal circuits [17] show that the number of RQFP logic gates and the number of garbage outputs are significantly reduced by 32.38% and 59.13% on average, respectively.

2 Preliminaries

2.1 Reversible Quantum-Flux-Parametron Logic

As a superconductor logic gate, the RQFP logic gate exhibits both logical and physical reversibility [6]. Fig. 1(a) illustrates its structure, composed of three three-output 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_{x1} and I_{x2} are required to drive these splitter gates and majority gates, respectively. The outputs of the RQFP logic gate source from the three three-input AQFP majority gates, making RQFP logic possess a compact logic representation. In a normal RQFP logic gate, an inverter is configured before the fixed-position input of each AQFP majority. This produces a one-to-one correspondence between the inputs and outputs of the RQFP gate to make the RQFP logic gate logically reversible [5]. Since inverters can be freely integrated into any input of each AQFP majority gate, the functionality of RQFP logic gates can be extended, which means that the function of each RQFP logic gate can be configured by the inverter setting.

Moreover, since the RQFP logic gate is realized by AQFP logic gates, it also inherits the inherent characteristics [13–15] of AQFP logic. In AQFP logic, each output of each AQFP logic gate can only drive one successor, i.e., the single fan-out limitation. The insertion of AQFP splitter gates can eliminate this limitation. Similarly, the RQFP splitter can be constructed by the introduction of constant inputs to satisfy this limitation [15]. For example, a 1-to-3 splitter can be achieved by $R(1, a, 0) = \{M(1, a, 0), M(1, a, 0), M(1, a, 0)\} = \{a, a, a\}$. Additionally, all inputs to each gate in AQFP logic must possess the same clock phases, i.e., the path-balancing requirement. This requirement can be satisfied by the insertion of AQFP buffers. Similarly, two cascaded AQFP buffers can be used to construct an RQFP buffer [15]. Meanwhile, the RQFP inverter can also be constructed by inserting an inverter in the RQFP buffer. Consequently, RQFP logic circuits must perform RQFP buffer and splitter insertion to meet these requirements, thereby ensuring proper circuit operation.

2.2 Cartesian Genetic Programming

Evolutionary algorithms show great potential in solving complex optimization problems. The most representative is the Cartesian genetic algorithm proposed by Julian F. Miller and Peter Thomson in 2000 [18]. Cartesian genetic programming (CGP) is a variant of genetic programming where the candidate solutions are represented as a string of fixed-length integers mapped to a directed acyclic graph. In CGP, the genotype is represented as a two-dimensional grid of nodes, where each node corresponds to a function or terminal. The genotype is then translated into a phenotype, which represents a computational structure or program. This modular representation allows for the efficient exploration of a vast search space, enabling CGP to tackle problems with high dimensionality and complexity, such as mathematical equations, computer programs, neural networks, and general digital circuits.

The CGP is the most powerful evolutionary technique in the domain of evolutionary algorithm (EA)-based logic synthesis and optimization[19, 20]. Many competitive results in general circuit design have been achieved since the introduction of the CGP. Particularly, the scalability of CGP has witnessed notable advancements with the introduction of an SAT-based CGP in 2011[21]. This approach addresses the challenge posed by the computationally expensive exhaustive circuit simulation, which is typically employed to determine the Hamming distance between a candidate solution and a given specification. A binary decision diagram (BDD)–based fitness function [22] is further used to speed up the evolution of complex circuits. Then, the combination of a circuit simulation and a formal verification [23] can support the optimization of combinational circuits with hundreds of inputs and thousands of gates. Besides, more complex real-world instances (millions of gates) can be optimized using windowing [24]. Consequently, this paper adopts CGP as the foundation for implementing an efficient automatic synthesis algorithm for RQFP logic to address the challenges associated with the generation of RQFP logic circuits.

3 RCGP

RQFP logic exhibits unique characteristics that present significant challenges in the design of RQFP logic circuits. RQFP logic gate has three output ports, each capable of independently functioning as a majority function. Additionally, RQFP logic circuits require buffer and splitter insertion to meet fan-out and path-balancing requirements. These distinct features of RQFP logic contribute to the complexity of the RQFP logic circuit design. To end this, this paper proposes an end-to-end circuit generation framework for RQFP logic, as shown in Fig. 2. Unlike the existing exact synthesis method [15] that takes truth tables as input, the proposed RCGP divides the entire process into several phases to gradually optimize the RQFP logic

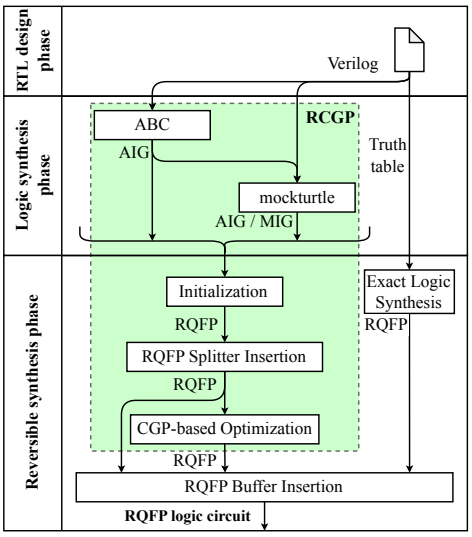

Figure 2: Design flows of proposed RCGP and existing exact logic synthesis, where lines represent interfaces between files and tools or methods.

circuit. First, existing common logic synthesis tools are employed to process the RTL input and optimize the circuit. Then, the generated network is directly converted to the RQFP logic netlist. Next, RQFP splitters are inserted into it to meet the fan-out limitation. In this way, an initial RQFP logic network is obtained. The RQFP-oriented CGP is proposed to optimize this initial RQFP logic network and minimize the number of RQFP logic gates and garbage outputs. Finally, after RQFP buffer insertion, the final RQFP logic circuit can be obtained.

3.1 Initialization

To support the RTL description inputs with multiple standard formats, such as the Verilog format, .aig format, and .blif format, RCGP first automatically integrates current popular open-source logic synthesis tools to process them, including ABC [25] and mockturtle [26]. Both ABC and mockturtle can optimize the AND-inverter graph (AIG)-based network, and mockturtle also can optimize the majority-inverter graph (MIG)-based network. The selection of both these tools and corresponding methods is configurable. After logic synthesis using these tools, AIG or MIG-based optimized networks can be obtained. Since the introduction of constant inputs can make the RQFP logic gate realize AND, OR, NOT, and majority functions easily, the AIG or MIG-based network can directly be converted into the RQFP logic netlist. Take the AND function as an example. Introducing one constant **1** can realize it, i.e., $R(a, b, 1) = \{M(\overline{a}, b, 1), M(a, \overline{b}, 1), M(a, b, 0)\} = \{\overline{a} + b, a + \overline{b}, ab\}$, where the last output achieves the AND function.

After generating the initial RQFP logic netlist, 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.

3.2 CGP-based Optimization

3.2.1 CGP Encoding and Evaluation. To optimize the RQFP logic circuit, CGP encoding is first introduced to represent the RQFP logic circuit. CGP usually encodes a candidate solution using an integer

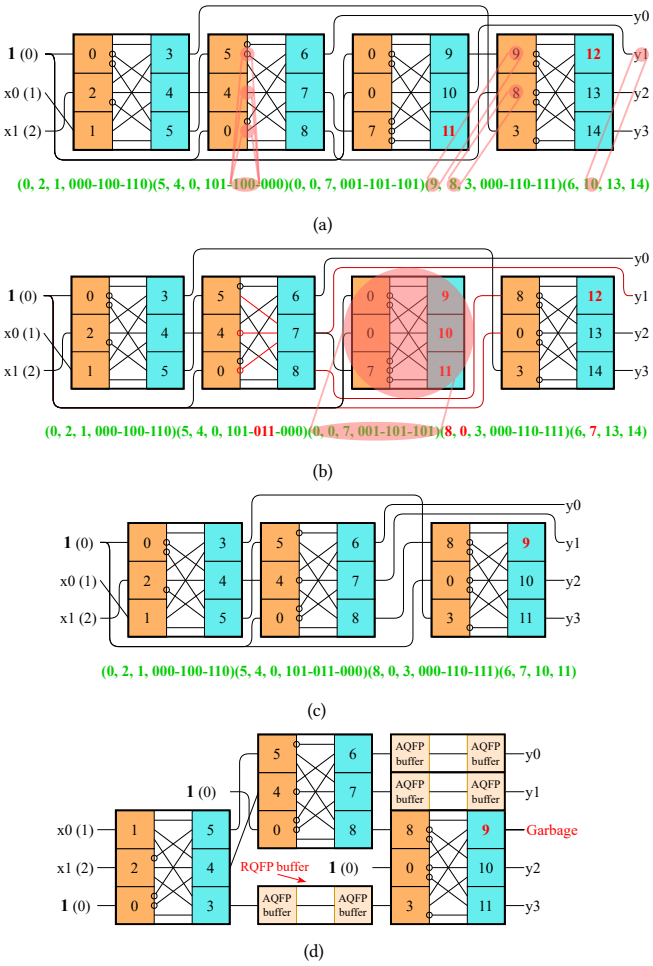

Figure 3: Schematics of CGP individuals encoding a 2-to-4 decoder circuit with $n_{pi} = 2$ primary inputs and $n_{po} = 4$ primary outputs. (a) shows a CGP individual with $n_r = 4$ RQFP logic gates and $n_g = 2$ garbage outputs. (b) is the individual with $n_r = 4$ RQFP logic gates (including one useless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

array consisting of $n_C \cdot n_R$ programmable nodes, where n_C and n_R determine the number of columns and rows, respectively. As for the circuit optimization, a linear form of CGP is usually preferred, i.e., $n_R = 1$. Each programmable node has a fixed number of inputs n_i and outputs n_o and can implement one of n_f predefined primitive functions. So, since each node represents an RQFP logic gate in RCGP, both n_i and n_o are set to 3. Besides, since the function of each RQFP logic gate depends on its inverter configuration, there are $2^9 = 512$ kinds of functions, i.e., $n_f = 512$. Any input to each node can be connected to the output of a node placed in the previous n_l columns or to one of n_{pi} primary inputs. Due to the complicated evaluation, the cycle is not allowed in the standard version of CGP, which means there is no feedback to the node from its successors. Therefore, the candidate solution is encoded as a CGP chromosome

with $n_L = n_C \times n_R \times (n_i + 1) + n_{po}$ integers, where n_{po} is the number of primary outputs of the circuits.

Since the positions of input ports in each RQFP logic gate are fixed and are directly related to the gate function output, the CGP encoding needs to adjust to apply to RQFP logic. Take a 2-to-4 decoder as an example. Fig. 3(a) illustrates its CGP encoding, where $n_C = 4$ and $n_R = 1$. The 2-to-4 decoder has $n_{pi} = 2$ primary inputs $x0$ and $x1$ and $n_{po} = 4$ primary outputs $y0$, $y1$, $y2$ and $y3$. Considering the requirement of RQFP splitters for constant inputs, constant **1** is introduced and indexed to 0. The primary inputs are indexed from 1 to n_{pi}. The integer substring within each pair of parentheses in the long green string at the bottom is the CGP encoding of the corresponding RQFP node. For example, "(5, 4, 0, 101-100-000)" is the CGP encoding of the second RQFP logic gate, where "5, 4, 0" denotes the interconnection of its input ports with the output ports indexed 5, 4, 0 (the constant input), respectively, and "101-100-000" denotes the inverter configuration in front of its three AQFP majority gates. The inverter configuration is represented by an integer with 9 bits, each of which indicates whether an inverter exists in the corresponding input port. For instance, "100" indicates whether an inverter exists before the second input port of each AQFP majority gate within the second RQFP logic gate. Besides, the last item "(6, 10, 13, 14)" represents the indexes of output ports connected to primary outputs, where "10" represents that the primary output $y1$ is connected to the second output port of the third RQFP logic gate.

After encoding a candidate solution, its evaluation is also required. The fitness value evaluation of the CGP chromosome in RCGP contains two phases. The first is the function evaluation, which calculates the success rate of the simulation-based equivalence checking. The second is the performance evaluation, which calculates the number of RQFP logic gates and garbage outputs. The detail is defined as follows:

(1) Only when the success rate reaches 100%, the performance evaluation will be performed, thereby ensuring the function legitimacy of the solution.
(2) Then, the performance evaluation gives priority to ensuring the optimal number of RQFP logic gates.
(3) In addition, while ensuring the optimal number of RQFP logic gates and garbage outputs, RCGP will also consider reducing the number of RQFP buffers that need to be inserted for the path balancing requirement.

3.2.2 Mutation. After constructing the CGP chromosome, CGP mutation must occur to produce its offspring. Point mutation is typically preferred due to its high efficiency. Point mutation randomly modifies up to m genes (integers) of a parent genotype to create an offspring, where m depends on the mutation rate $\mu, \mu \in [0, 1]$. To ensure that each gene has a chance to be modified randomly within each mutation, RCGP sets the maximum of m to $\mu * n_L$, where n_L is the length of the chromosome.

Considering the CGP encoding, a single mutated gene causes either reconnection of a node, reconnection of a primary output, or change in the inverter configuration of a node. Due to the fan-out limitation of RQFP logic, there are two situations for the node reconnection. Notably, the initialization process has ensured the single fan-out limitation of the parent genotype.

Algorithm 1: The flow of CGP-based optimization.

Input: Initial chromosome c, total number N of generations, mutation rate μ, number λ of offspring.
Output: Optimized chromosome.

calculate the functional fitness f of c.
$f_n = f, c_n = c, n_r = \infty, n_g = \infty, n_b = \infty$.
if $f == 1$ **then**
  remove useless nodes to shrink c and update c to c_n.
  calculate the number (n_r, n_g, n_b) of gates, garbage outputs, and buffers in c_n.
for $i = 1 \rightarrow N$ **do**
  $c = c_n$.
  for $j = 1 \rightarrow \lambda$ **do**
    perform the mutation on c to generate a offspring c'.
    calculate the functional fitness f' of c'.
    if $f' == 1$ **then**
      remove useless nodes to shrink c' and update c'.
      calculate the number (n'_r, n'_g, n'_b) of gates, garbage outputs, and buffers in c'.
      if $(n_r, n_g, n_b) > (n'_r, n'_g, n'_b)$ **then**
        $(f_n, c_n, n_r, n_g, n_b) = (f'_n, c'_n, n'_r, n'_g, n'_b)$.
    else if $f' > f_n$ **then**
      $(f_n, c_n) = (f'_n, c'_n)$.
return c_n.

(1) When the output port corresponding to the mutated value of a gene has been connected to another gene, RCGP will swap the values of these two genes. As shown in Fig. 3(a), assume that '9' in the last second item "(9, 8, 3, 000-110-111)" mutates to '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, i.e., "(8, 9, 3, 000-110-111)".
(2) When the output port corresponding to the mutated value of a gene is the constant input **1** or has no output, this value is directly assigned to this gene. After mutating to "(8, 9, 3, 000-110-111)", assume that '9' continues to mutate to '0', the genotype of the last node becomes "(8, 0, 3, 000-110-111)". This means that the second input port of the last node is connected to the constant input **1**, as shown in Fig. 3(b).

For the reconnection of a primary output, its gene is directly updated to the mutated value, such as $y1$ in Fig. 3(b) changed from 10 to 7. Furthermore, for the change of an inverter configuration f, the mutation will produce an integer $\beta \in [0, 9)$ to update f to $f' = f \oplus (1 \ll \beta)$, which means that the $(1 + \beta)^{\text{th}}$ inverter is inserted or removed. As shown in Fig. 3(a), initial inverter configuration of the second node is "101-100-000" (352) and then is updated to "101-011-000" ($344 = 352 \oplus ((1 \ll 3) + (1 \ll 4) + (1 \ll 5))$) through three mutations, as shown in Fig. 3(b). In this way, the output function of the second majority in the second node is updated.

3.2.3 Shrink. After CGP mutation, some useless nodes may exist, such as the third node "(0, 0, 7, 001-101-101)" in Fig. 3(b). This directly indicates a primary advantage of CGP encoding that even if the size of the chromosome is fixed, the size of the phenotype is variable since some nodes need not be used. Since RCGP aims to minimize

Table 1: Experimental results on small circuits from the RevLib benchmark [16].

Testcase	Original			Initialization					Exact logic synthesis						RCGP					
	n_{pi}	n_{po}	g_{lb}	n_r	n_b	JJs	n_d	n_g	n_r	n_b	JJs	n_d	n_g	T (s)	n_r	n_b	JJs	n_d	n_g	T (s)
1-bit full adder	3	2	1	6	2	152	3	7	3	3	84	3	2	41.19	3	2	80	3	2	75.69
4gt10	4	1	3	3	3	84	3	6	3	4	88	3	5	76.01	3	4	88	3	5	75.43
alu	5	1	4	12	10	328	5	17	4	7	124	4	5	1893.54	4	6	120	4	5	232.51
c17	5	2	3	11	7	292	4	16	5	14	176	5	5	106167.29	5	10	160	4	5	321.17
decoder_2_4	2	4	0	8	3	204	3	10	3	3	84	3	1	24.77	3	3	84	3	1	236.36
decoder_3_8	3	8	0	20	12	528	4	23				\			7	25	268	7	1	978.53
graycode4	4	4	0	15	7	388	4	21				\			7	10	208	5	3	835.74
ham3	3	3	0	16	5	404	4	18	5	5	140	5	1	2216.02	5	4	136	5	2	326.41
mux4	6	1	5	11	10	304	5	16				\			7	19	244	6	7	769.14

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

the number of gates and garbage outputs, these useless nodes can be removed to shrink the size of the chromosome, thereby reducing the search space, as shown in Fig. 3(c). Now, the chromosome length of the 2-to-4 decoder is reduced from 20 to 16.

3.2.4 Flow of CGP-based Optimization. Algorithm 1 shows the complete process of CGP-based optimization, which employs a typical $(1 + \lambda)$ evolutionary strategy [18]. It mutates one best parent genotype to create λ offspring within each generation (lines 8-10). An offspring with a fitness better or equal to the parent becomes the new parent for the next generation (lines 11-17). After N generations, the optimal offspring is returned (18).

3.3 RQFP Buffer Insertion

Following the CGP-based optimization, it is possible that certain RQFP logic gates within the generated RQFP logic circuit may not satisfy the path balancing requirement. Consequently, the insertion of RQFP buffers becomes necessary to ensure uniform clock phases for all inputs to each gate. That is, a corresponding number of RQFP buffers must be inserted into each edge according to the clock phase difference between its connected RQFP logic gates. Fig. 3(d) shows the buffer insertion result of the RQFP logic circuit shown in Fig. 3(c) generated for the 2-4 decoder.

4 Experimental Results

The proposed automatic RQFP logic synthesis framework RCGP was implemented in C++ and Python. The experiments used the RevLib benchmark circuits [16] and the reversible reciprocal circuits [17] to evaluate RCGP. The logic synthesis phase of RCGP first used the "resyn2" command in the ABC [25] to obtain an optimized AIG network and then employed the "aqfp_resynthesis" command in the mockturtle to resynthesis it into an AQFP-oriented MIG network. The "aqfp_resynthesis" command implements a state-of-the-art MIG-based logic optimization method [27] of AQFP logic. Besides, the number N of generations and the mutation rate μ in RCGP was 50,000,000 and 1, respectively. The number n_C of columns was the number of RQFP logic gates in the generated initial RQFP logic netlist. n_l kept the same as the n_C and n_R was 1. The experiments were executed on the machine with Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz and 256.0 GB memory running CentOS 7.

There were two baselines to compare with RCGP. The first was designed by a heuristic method. As shown in Fig. 2, after the initialization and RQFP splitter insertion, the baseline directly carries out the RQFP buffer insertion to meet the path-balancing requirement, thereby generating RQFP logic circuits. The second was the existing exact logic synthesis method [15] using Z3 solver. According to the input truth table, it can directly generate the RQFP logic circuit with a given number of RQFP logic gates and garbage outputs. To ensure the consistency of each experimental result in handling the path balancing requirement for primary inputs and primary outputs of the generated circuit, both primary inputs and primary outputs require RQFP buffer insertion so that they are in the same clock stage, respectively.

Table 1 shows the experimental results on small circuits from the RevLib benchmark. The "Original" part shows the attributes 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. The "Initialization" part shows the results of the first baseline method. n_r represents the number of used RQFP logic gates, and n_b represents the number of RQFP buffers inserted for the path-balancing requirement. "JJs" represents the number of Josephson junctions (JJs). Since JJs are a critical component of AQFP circuits and are used to create the fluxons used for computation, their number can estimate the complexity and energy efficiency of a given AQFP circuit. So, 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 [15] for RQFP logic, where "T (s)" represents the runtime of the method and its unit is the second. The "RCGP" part shows the results of the proposed algorithm RCGP. Compared to the first baseline, the proposed RCGP significantly reduced the number of RQFP logic gates, JJs, and garbage outputs, specifically by 50.80%, 43.53%, and 71.55%, respectively. Meanwhile, although the exact logic synthesis can generate the RQFP logic circuit with the optimal number of RQFP logic gates and garbage outputs, the results of RCGP can be near even up to its results while using less runtime. Besides, the exact logic synthesis could not also find the solution for circuits "decoder_3_8", "graycode4", and "mux4" under sufficiently given 240,000 seconds. Therefore, these experimental results show that the proposed RCGP can generate RQFP logic circuits with a near-optimal number of RQFP logic gates and garbage outputs on small benchmark circuits.

Table 2: Experimental results on large circuits from the RevLib benchmark [16] and the reversible reciprocal circuits [17].

Testcase	Original			Initialization					Exact logic synthesis	RCGP					
	n_{pi}	n_{po}	g_{lb}	n_r	n_b	JJs	n_d	n_g		n_r	n_b	JJs	n_d	n_g	T (s)
4_49_7	4	4	0	35	17	908	5	41	\	21	83	836	13	12	1244.71
graycode6_11	6	6	0	25	9	636	4	35	\	13	31	436	7	7	853.09
mod5adder_66	6	6	0	139	137	3884	10	165	\	105	663	5172	29	63	11102.79
hwb8_64	8	8	0	1427	2727	45156	20	1662	\	1397	2729	44444	20	1533	157468.63
intdiv4	4	4	0	26	15	684	5	32	\	15	40	520	9	9	876.90
intdiv5	5	5	0	51	46	1408	8	63	\	35	119	1316	14	20	1859.56
intdiv6	6	6	0	107	95	2948	9	128	\	76	292	2992	18	45	5192.59
intdiv7	7	7	0	200	202	5608	11	234	\	128	764	6128	30	80	7562.12
intdiv8	8	8	0	381	534	11280	15	453	\	236	1681	12388	31	164	17786.66
intdiv9	9	9	0	720	944	21056	16	859	\	483	1859	19028	25	414	64670.10
intdiv10	10	10	0	1225	1986	37344	20	1453	\	833	2877	31500	26	817	146310.78

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

Table 2 shows the experimental results on large circuits from the RevLib benchmark and reversible reciprocal circuits. It is evident that exact logic synthesis could not find the solution of any testcase under a given sufficient time, further enhancing the essentiality of RCGP. In addition, compared to the first baseline, the proposed RCGP has a significant reduction in the number of RQFP logic gates and the number of garbage outputs, specifically by 32.38% and 59.13%, respectively. Therefore, these experimental results demonstrate that the proposed RCGP can effectively and efficiently optimize large RQFP logic circuits regarding the number of gates and garbage outputs.

5 Conclusion

This paper proposed RCGP, a Cartesian genetic programming-based automatic synthesis framework for RQFP logic. RCGP employs CGP encoding to present the RQFP logic circuit and combines circuit simulation with formal verification to assess the functional equivalence between the parent and offspring. After the RQFP-oriented mutation, the RQFP logic circuit can be optimized for the number of gates and garbage outputs. The experimental results on the RevLib benchmark circuits and reversible reciprocal circuits show the effectiveness of RCGP on the generation of RQFP logic gates. Furthermore, the experimental results demonstrated the applicable limitation of exact logic synthesis on larger RQFP logic circuits, further revealing the significance of the proposed RCGP.

Acknowledgments

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 and was 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.

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 *Proc. ASPDAC*, p. 669–674, 2019.
[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," *Supercond. Sci. Technol.*, 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] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An adiabatic quantum flux parametron as an ultra-low-power logic device," *Supercond. Sci. Technol.*, vol. 26, no. 3, p. 035010, 2013.
[11] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, vol. 85, pp. 632–644, 1980.
[12] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.
[13] S.-Y. Lee, H. Riener, and G. De Micheli, "Beyond local optimality of buffer and splitter insertion for AQFP circuits," in *Proc. DAC*, pp. 445–450, 2022.
[14] 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 *Proc. ASPDAC*, pp. 769–774, 2023.
[15] R. Fu, O. Chen, N. Yoshikawa, and T.-Y. Ho, "Exact logic synthesis for reversible quantum-flux-parametron logic," in *Proc. ICCAD*, pp. 1–9, 2023.
[16] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Proc. ISMVL*, pp. 220–225, 2008. 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 *Proc. DATE*, pp. 470–475, 2017.
[18] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*, pp. 121–132, 2000.
[19] J. F. Miller, "Cartesian genetic programming: its status and future," *Genet Program Evolvable Mach*, vol. 21, pp. 129 – 168, 2019.
[20] A. Manazir and K. Raza, "Recent developments in cartesian genetic programming and its variants," *ACM Comput. Surv.*, vol. 51, no. 6, 2019.
[21] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genet Program Evolvable Mach*, vol. 12, pp. 305–327, 2011.
[22] Z. Vasicek and L. Sekanina, "How to evolve complex combinational circuits from scratch?," in *Proc. ICES*, pp. 133–140, 2014.
[23] Z. Vasicek, "Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates," in *Genetic Programming*, pp. 139–150, 2015.
[24] J. Kocnova and Z. Vasicek, "EA-based resynthesis: An efficient tool for optimization of digital circuits," *Genet Program Evolvable Mach*, vol. 21, no. 3, p. 287–319, 2020.
[25] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, pp. 24–40, 2010.
[26] 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.
[27] 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 *Proc. DATE*, pp. 34–39, 2022.