Exact Logic Synthesis for Reversible Quantum-Flux-Parametron Logic

Rongliang Fu[†], Olivia Chen[‡], Nobuyuki Yoshikawa[‡], Tsung-Yi Ho[†]
[†]*The Chinese University of Hong Kong*
[‡]*Tokyo City University*
[‡]*Yokohama National University*
rlfu@cse.cuhk.edu.hk

***Abstract*—Reversible computing, deriving its inspiration from Landauer's principle, has captured significant interest as a promising technology for logic operations without energy dissipation. The reversible quantum-flux-parametron (RQFP) stands as the first practical reversible logic gate using adiabatic superconducting devices, whose logical and physical reversibility has been experimentally demonstrated. However, due to its unique logic function and structure, the design of RQFP logic circuits is a highly challenging task. At present, there are no automated design tools available for RQFP logic. Therefore, this paper proposes the first exact logic synthesis algorithm for RQFP logic. It formulates the synthesis problem as the Boolean satisfiability problem and subsequently constructs and calls upon the incremental propositional logic model iteratively for optimal synthesis with the least number of gates and garbage outputs. Experimental results on the reversible logic benchmark from RevLib demonstrate the effectiveness of the proposed algorithm.**

***Index Terms*—reversible computing, superconductor logic, RQFP, exact logic synthesis**

I. Introduction

The ongoing downscaling of field-effect transistors has been a primary driving force behind the enhancement in the performance and energy efficiency of microprocessors for several decades. Nonetheless, due to technological and fundamental limitations for non-adiabatic complementary metal-oxide-semiconductor (CMOS) logic [1], [2], it is impractical to attain smaller switching energy through device miniaturization, which has stimulated the development of beyond-CMOS energy-efficient alternative computing systems. Landauer stated in 1961 [3] that the minimum energy dissipation to erase one bit of information is $k_BT\ln 2$, where k_B is the Boltzmann constant and T is the temperature of the system. Furthermore, Bennett showed in 1973 [4] that there is no energy dissipation for logic operations without the reduction in information entropy under ideal physical circumstances. Subsequently, reversible computing [5], [6] has gained attention as a prospective solution for energy dissipation. Especially in quantum computing, reversible computing can play a vital role due to the inherent reversibility of quantum operations.

However, the physical implementation of reversible computing presents a formidable undertaking, as it necessitates both logical and physical reversibility, as well as ultra-low power logic devices. The reversible quantum-flux-parametron (RQFP) proposed in 2014 [7], [8] is the first practical reversible logic gate using adiabatic superconducting devices, with its logical and physical reversibility being validated through experimental demonstrations. The RQFP logic gate is realized using adiabatic quantum-flux-parametron (AQFP) logic [9] based on superconductor technology. This logic gate can execute logical operations with significantly low energy dissipation, possibly even below $k_BT\ln 2$. Besides, Yamae *et al.* [10] used RQFP logic gates to design and fabricate a reversible full adder, further revealing the reversibility and feasibility of RQFP logic circuits. Hence, RQFP logic has drawn great attention from researchers [11], [12].

However, the lack of automated design tools has resulted in the limited availability of RQFP-based reversible logic circuit designs [11]. Conventional reversible logic circuit designs primarily realize reversible Boolean functions relying on basic Toffoli [5] and Fredkin [6] 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, these two kinds of reversible logic gates can be viewed as multi-controlled NOT and multi-controlled SWAP gates, respectively. Their output functions mainly focus on the last one or two output ports. In contrast, the normal RQFP logic gate shown in Fig. 2(a) consists of three AQFP splitter gates and three AQFP majority gates. Its function can be represented by the equation $R(a,b,c) = (M(\overline{a},b,c), M(a,\overline{b},c), M(a,b,\overline{c})) = (x,y,z)$, where a, b, and c are three inputs, x, y, and z are three outputs, and $M(\cdot)$ represents the majority function, such as the three-input majority function is $M(a,b,c) = ab + ac + bc$. Given that RQFP logic is realized through AQFP logic, both share the same fan-out limitation and path-balancing requirement [13]–[15]. As a result, the logic synthesis methods used for conventional reversible logic are not applicable to RQFP logic due to its distinctive logic function and structure. Therefore, the research on the logic synthesis method of RQFP logic is of crucial significance.

Since the number of gates and garbage outputs can cause significant energy dissipation in RQFP logic, the need for RQFP logic circuit design is up to or near the optimum of these two metrics. To guarantee optimality, this paper formulates the logic synthesis problem of RQFP logic as the Boolean satisfiability problem and proposes the first exact logic syn-

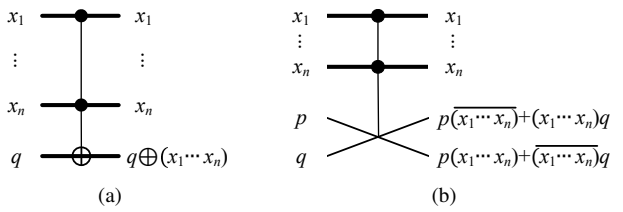
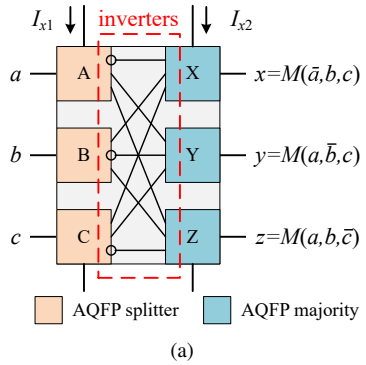


Fig. 1. Functional schematics of (a) MCT gate and (b) MCF gate, where symbols utilized are $\cdot$ for logical product, $+$ for logical sum, and $\bar{}$ for logical negation.



a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	1	1

(b)

Fig. 2. (a) and (b) show the structure schematic and the truth table of the normal RQFP logic gate, respectively, where AC currents I_{x1} and I_{x2} are clock signals, and the circles represent the inverters.

thesis algorithm for RQFP logic. The algorithm leverages the inherent characteristics of AQFP logic, and first redefines the logic function of RQFP logic gates. Subsequently, the algorithm can incrementally construct the proposition logic model by increasing the number of RQFP logic gates. After obtaining the first feasible solution, it iteratively checks the model by reducing the number of garbage outputs, thereby achieving optimal synthesis with the least number of gates and garbage outputs. Finally, the proposed RQFP buffers are inserted into the synthesis result to satisfy the path-balancing requirement, thereby generating the RQFP logic circuit.

In summary, this paper makes the following contributions:

- This paper proposes the first exact logic synthesis algorithm to generate RQFP logic circuits for a given number of gates and garbage outputs.
- The proposed algorithm employs an incremental encoding approach to construct the model targeting minimizing the number of gates and garbage outputs.
- Furthermore, a simplification of the propositional logic model is proposed along with four optimization strategies to break the structural and functional symmetries and thereby reduce solving time.
- Experimental results on the reversible logic benchmarks provided at RevLib [16] demonstrate the effectiveness of the proposed algorithm.

II. Preliminaries

A. Reversible Quantum-Flux-Parametron

The RQFP logic gate, a type of superconductor logic gate, is both logically and physically reversible [8]. As shown in Fig. 2(a), it consists of six symmetrically interconnected AQFP logic gates, with three of them labeled A, B, and C operating as three-output splitter gates, and the remaining three labeled as X, Y, and Z operating as three-input majority gates. I_{x1} and I_{x2} are the excitation currents for the splitter gates and majority gates, respectively. When excitation currents arrive, the Josephson junction (JJ) in the AQFP logic gate switches, and then the inductor generates the output current. Fig. 2(b) describes the truth table of the RQFP logic gate, where the values '1' and '0' represent logic **true** and **false**, respectively. The input and output of the RQFP logic gate exhibit a one-to-one correspondence, which unambiguously demonstrates its logical reversibility [7].

In a normal RQFP logic gate, each of its outputs serves the function of a majority function. This attribute confers upon RQFP logic a more compact logic representation compared to the AND-inverter graph (AIG) and majority-inverter graph (MIG). However, since RQFP logic is implemented using AQFP logic gates, it must adhere to the inherent constraints of AQFP logic, including the fan-out limitation and path-balancing requirement [13]–[15]. Specifically, each output of each AQFP logic gate can only drive one successor, and all inputs to each gate must possess the same clock phases. Consequently, RQFP logic circuits must also satisfy these requirements to ensure proper functioning. Meanwhile, the inverter setting for normal RQFP logic gates remains fixed, that is, there is an inverter in front of each majority gate, and its position is firmly established. Since inverters can be integrated into any input of each AQFP majority gate, the functionality of RQFP logic gates can be extended. That is, each output of an RQFP logic gate can have eight function choices, including $M(a,b,c)$, $M(a,b,\overline{c})$, $M(a,\overline{b},c)$, $M(a,\overline{b},\overline{c})$, $M(\overline{a},b,c)$, $M(\overline{a},b,\overline{c})$, $M(\overline{a},\overline{b},c)$, and $M(\overline{a},\overline{b},\overline{c})$.

1) RQFP Buffer: To fulfill the path-balancing requirement, the insertion of buffers into RQFP logic circuits becomes necessary. One approach is to utilize the combination of an RQFP logic gate and constant inputs to generate the RQFP buffer. For instance, $R(a,0,1) = (M(a,0,1), M(a,\overline{0},1), M(a,0,\overline{1})) = (a,1,0)$ can achieve the RQFP buffer function while simultaneously producing two additional outputs, resulting in significant cost. In light of the logical and physical reversibility requirement of reversible computing, two cascaded AQFP buffers can be employed to construct an RQFP buffer, as shown in Fig. 3(a). Furthermore, based on the inverter property of AQFP logic, the RQFP inverter can be constructed as shown in Fig. 3(b), which can be used to link the primary input and primary output with complementary logic values.

2) RQFP Splitter: In AQFP logic, the splitter gate can directly achieve the multiple fan-out., but can not be directly used in RQFP logic due to its irreversibility. Considering the reversibility, the RQFP splitter can be realized by the combination of an RQFP logic gate and constant inputs. For instance, the 1-to-3 splitter can be realized by

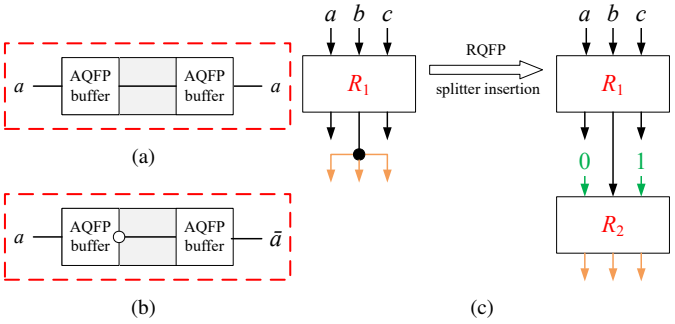


Fig. 3. Schematics of (a) RQFP buffer, (b) RQFP inverter, and (c) splitter insertion for RQFP logic. The gate R_1 has five fan-outs with three distinct types of functions. After RQFP splitter insertion, the new gate R_2 can afford the same output function with three fan-outs (orange arrows).

$R(0,a,1) = (M(0,a,1), M(0,a,1), M(0,a,1)) = (a,a,a)$. Fig. 3(c) shows an example of splitter insertion in RQFP logic. The second output port of gate R_1 has three fan-outs. An RQFP logic gate R_2 with two constant inputs can be inserted after it to satisfy the fan-out limitation.

3) RQFP Logic Circuits: RQFP logic gates have been demonstrated experimentally as effective building blocks for the design of reversible logic circuits. In general, reversible logic circuits can implement a reversible Boolean function that represents a one-to-one mapping between vectors of primary inputs and primary outputs. To render a nonreversible function reversible, additional constant inputs and outputs, known as garbage outputs, must be introduced. However, numerous garbage outputs can lead to energy waste within the RQFP logic circuit. Therefore, except for gate cost, the design of RQFP logic circuits must also take into account the number of garbage outputs.

B. Exact Logic Synthesis

Exact logic synthesis is a crucial component of digital circuit design, targeting the generation of a logic circuit that corresponds functionally to a provided Boolean function while optimizing certain criteria such as gate count and circuit depth. Unlike heuristic-based methods, exact logic synthesis can guarantee the effective realization of an optimal or near-optimal logic circuit, which reveals its importance in building libraries of small functions that can be used to construct larger functions. Take the number of gates as an example. In a logic circuit with n primary inputs $x_1, \ldots, x_n$ and m primary outputs $o_1, \ldots, o_n$, m logic functions over n variables are represented as

$$(o_1 = f_1(x_1, \ldots, x_n), \ldots, o_m = f_m(x_1, \ldots, x_n)). \quad (1)$$

The objective of exact logic synthesis is to explore the detailed implementation of $f_i, i \in [1,m]$ using given r gates, i.e.,

$$\forall x_1, \ldots, x_n, f_i(x_1, \ldots, x_n) \underset{i\in[1,m]}{\equiv} cir_i(x_1, \ldots, x_n), \quad (2)$$

where $\{cir_i, i \in [1,m]\}$ are specified by selecting appropriate gate types and connections. Therefore, the key to exact logic synthesis becomes how to effectively and efficiently formulate the types of gates and the connections between gates, primary inputs, and primary outputs.

As a vital technique in the domain of electronic design automation, Boolean satisfiability finds extensive application in modern practical implementations of exact logic synthesis. The Boolean satisfiability (SAT) problem deals with a propositional logic formula $f(x_1, \ldots, x_n)$ that is constructed utilizing Boolean variables $x_1, \ldots, x_n$ and logic connectives, including AND ($\wedge$), OR ($\vee$), NOT ($^-$), and Implication ($\implies$). SAT aims to verify if an assignment exists for the variables $x_1, \ldots, x_n$ such that f evaluates to true. If so, f is considered satisfiable; otherwise, f is unsatisfiable. The Boolean formula f is commonly represented in conjunctive normal form (CNF), which is a conjunction of clauses where a clause is a disjunction of literals, and a literal denotes a variable in regular or complemented form.

III. Methodology

The RQFP logic gate has three input ports and three output ports, each output of which can work as the majority function. These unique characteristics of RQFP logic create substantial complexity during the circuit design of RQFP logic. This section first introduces the SAT-based encoding for the properties of RQFP logic to implement its exact logic synthesis. Subsequently, specific optimizations are presented to break the symmetries to reduce the solving time.

A. SAT Encoding for RQFP Logic

To realize the RQFP logic circuit with m functions $f_i(x_1, \ldots, x_n), i \in [1,m]$ and r RQFP logic gates, the corresponding SAT encoding inspired by Boolean chains [17] is expressed as follows.

1) Encoding for Variables: First, the Boolean variables used in the SAT formula are defined as the following:

- $o_{i[t]}, i \in [1,m]$: The t^{th} bit of primary output o_i in the truth table.
- $x_{0(0)[t]}, t \in [1, 2^n]$: Constant **true** in the truth table. Given the inverter feature, only constant **true** is used.
- $x_{i(0)[t]}, i \in [1,n], t \in [1, 2^n]$: The t^{th} bit of primary input x_i in the truth table.
- $x_{i(p)[t]}, i \in [n+1, n+r], p \in [1,3], t \in [1, 2^n]$: The t^{th} bit of the p^{th} output port of gate x_i in the truth table.
- $c_{i,j(p_1),k(p_2),l(p_3)}, i \in [n+1, n+r]$: Three inputs of gate x_i from left to right: $j(p_1), k(p_2), l(p_3)$, each of which can be an output port of any gate, constant **true**, or any primary input.
- $c_{i,j(p)}, i \in [n+r+1, n+r+m], i > j$: Primary output $o_{(i-n-m)}$ is connected to node $x_{j(p)}$, which can be an output port of any gate or constant **true**. Note that direct connections between primary inputs and primary outputs are excluded, which can be realized by cascading buffers.

To avoid introducing cycles into the circuit, the constraint $(i > j) \wedge (i > k) \wedge (i > l)$ is employed for each variable $c_{i,j(p_1),k(p_2),l(p_3)}$. Moreover, given that the majority function is insensitive to the order of its inputs, the constraint $(i >$

$l) \wedge ((l > k) \vee ((l == k) \wedge (p_3 > p_2))) \wedge ((k > j) \vee ((k == j) \wedge (p_2 > p_1)))$ is employed to ensure that three inputs to gate x_i are ordered.

Furthermore, considering the significant number of variables generated for input connections of each gate, the introduction of Boolean selection variables $c_{i,j(p)}, i \in [n+1, n+r], i > j$ in place of variables $c_{i,j(p_1),k(p_2),l(p_3)}$ can reduce the total number of variables from

$$\sum_{i=n+1}^{n+r} \binom{n+1+(i-n-1)*3}{3} \tag{3}$$

to

$$\sum_{i=n+1}^{n+r} (n+1+(i-n-1)*3), \tag{4}$$

while maintaining the same constraints for the entire problem. However, this reduction comes at the cost of augmented complexity of clauses, thus posing a trade-off between the number of variables and the complexity of clauses.

2) Encoding for Gate Functions: The RQFP logic circuit is designed around the utilization of the RQFP logic gate as its fundamental logic gate, which has three output ports, each operating as the majority function. As the output functions of each RQFP logic gate are primarily regulated by the inverter configuration that precedes the AQFP majority gates, they can be encoded through the use of nine Boolean variables $f_{i(p,j)}, p \in [1,3], j \in [1,3]$, where each variable indicates whether an inverter does not exist in front of the j^{th} input of the p^{th} majority in gate $x_i, i \in [n+1, n+r]$. Therefore, the output value $x_{i[p]}$ of the p^{th} output port of gate x_i must satisfy the following constraint

$$\begin{aligned}
&\left(c_{i,j(p_1)} \wedge c_{i,k(p_2)} \wedge c_{i,l(p_3)}\right) \implies \bigwedge_{t\in[1,2^n]} (\\
&((f_{i(p,1)} \wedge f_{i(p,2)} \wedge \underline{f_{i(p,3)}}) \implies M(x_{j(p_1)[t]}, x_{k(p_2)[t]}, x_{l(p_3)[t]}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((f_{i(p,1)} \wedge \underline{f_{i(p,2)}} \wedge \overline{f_{i(p,3)}}) \implies M(x_{j(p_1)[t]}, x_{k(p_2)[t]}, \overline{x_{l(p_3)[t]}}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((f_{i(p,1)} \wedge \underline{f_{i(p,2)}} \wedge \underline{f_{i(p,3)}}) \implies M(x_{j(p_1)[t]}, \overline{x_{k(p_2)[t]}}, x_{l(p_3)[t]}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((\underline{f_{i(p,1)}} \wedge \overline{f_{i(p,2)}} \wedge \overline{f_{i(p,3)}}) \implies M(x_{j(p_1)[t]}, \overline{x_{k(p_2)[t]}}, \overline{x_{l(p_3)[t]}}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((\underline{f_{i(p,1)}} \wedge f_{i(p,2)} \wedge \underline{f_{i(p,3)}}) \implies M(\overline{x_{j(p_1)[t]}}, x_{k(p_2)[t]}, x_{l(p_3)[t]}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((\underline{f_{i(p,1)}} \wedge \underline{f_{i(p,2)}} \wedge \overline{f_{i(p,3)}}) \implies M(\overline{x_{j(p_1)[t]}}, x_{k(p_2)[t]}, \overline{x_{l(p_3)[t]}}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((\underline{f_{i(p,1)}} \wedge \underline{f_{i(p,2)}} \wedge \underline{f_{i(p,3)}}) \implies M(\overline{x_{j(p_1)[t]}}, \overline{x_{k(p_2)[t]}}, x_{l(p_3)[t]}) \oplus \overline{x_{i[p][t]}}) \wedge \\
&((\underline{f_{i(p,1)}} \wedge f_{i(p,2)} \wedge \overline{f_{i(p,3)}}) \implies M(\overline{x_{j(p_1)[t]}}, \overline{x_{k(p_2)[t]}}, \overline{x_{l(p_3)[t]}}) \oplus \overline{x_{i[p][t]}})).
\end{aligned} \tag{5}$$

Moreover, as illustrated in Section II-A2, an RQFP logic gate can function as either an RQFP splitter or an RQFP buffer. In both cases, it necessitates two constant inputs. Consequently, if an RQFP logic gate x_i functions as a splitter or buffer, it must satisfy the following constraint

$$c_{i,0(0)} \wedge c_{i,j(p)} \wedge \left(\bigwedge_{(k==j \implies q\neq p) \wedge k\in[1,i)} \overline{c_{i,k(q)}} \right), \tag{6}$$

where $c_{i,j(p)}$ specifies that $x_{j(p)}$ serves as the source of multiple fan-outs. This constraint on gate x_i is noted as $s_{i,j(p)}$.

3) Encoding for Constraints: After creating the requisite variables to support the propositional logic model, the next step entails defining the constraints that must be met to ensure compliance with the desired properties of RQFP logic circuits.

- Circuit function constraints:

$$\bigwedge_{\substack{i\in[n+r+1,n+r+m] \\ j\in[n+1,n+r],p\in[1,3]}} \left(c_{i,j(p)} \implies \bigwedge_{t\in[1,2^n]} o_{(i-n-r)[t]} \oplus \overline{x_{j(p)[t]}} \right), \tag{7}$$

  which shows that primary output $o_{(i-n-r)}$ is connected to node $x_{j(p)}, j \in [n+1, n+r]$.
- Single fan-in constraints:

$$\bigwedge_{i\in[n+1,n+r+m]} \left(\left(\sum_{j=0}^{i-1} c_{i,j(*)} \right) == 1 \right), \tag{8}$$

  where $c_{i,j(*)}$ denotes the connections from all output ports of node x_j to node x_i.
- Single fan-out constraints:

$$\left(\bigwedge_{j\in[1,n]} \left(\left(\sum_{i=n+1}^{n+r+m} c_{i,j(0)} \right) == 1 \right) \right) \wedge \left(\bigwedge_{\substack{j\in[n+1,n+r] \\ p\in[1,3]}} \left(\left(\sum_{i=j+1}^{n+r+m} c_{i,j(p)} \right) \leq 1 \right) \right), \tag{9}$$

  which posits that except for constant **true**, every primary input has one and only one fan-out, and each output port of any logic gate has at most one fan-out. Besides, according to the fan-in and fan-out constraints of RQFP logic, the relationship between the fan-out number c of constant **true** and the number g of garbage outputs can be summarized as $n + c \equiv m + g$. Thus, the fan-out constraint of constant **true** is

$$\left(\sum_{i=n+1}^{n+r+m} c_{i,0(0)} \right) == m + g - n \tag{10}$$

- Garbage output constraints:

$$\left(\sum_{\substack{j\in[n+1,n+r] \\ p\in[1,3]}} \left(\bigwedge_{i\in[j+1,n+r+m]} \overline{c_{i,j(p)}} \right) \right) == g, \tag{11}$$

  where g is the given number of garbage outputs. Besides, since a reversible function has an equivalent number of primary inputs and primary outputs, the minimum number of garbage outputs required can be determined from the relationship $g \geq \max(0, n-m)$.

These constraints identified above require the introduction of cardinality constraints, which establish the minimum and maximum number of true clauses within a formula. In certain instances, such as for the sum constraint where a value to be equal to or less than one is required, it may be advantageous to avoid arithmetic computation altogether and instead express the constraint in a Boolean format, such as through the utilization of ITE encoding and shifter encoding [18]. This paper uses the "card2bv" command available within the Z3 solver [19] to accomplish the elimination of cardinality constraints.

B. Exact Logic Synthesis for RQFP Logic

Upon the creation of all essential variables and constraints, the propositional logic model can be solved to obtain the values of the variables, which reveal the function of each

Algorithm 1: Exact logic synthesis for RQFP logic.

Input: The number n of primary inputs $x_1, \ldots, x_n$, the number m of primary outputs $o_1, \ldots, o_m$, and the truth table of $x_1, \ldots, x_n, o_1, \ldots, o_m$.
Output: The generated RQFP logic circuit

```
1  r = ⌈max(n, m)/3.0⌉
2  g_lb = max(0, n − m)
3  g = −1
4  sol ← Solver()
5  create variables for primary inputs, primary outputs,
     constant true, r gates, and connections.
6  create clauses for function constraints.
7  sol.push() // create a snapshot
8  while true do
9      if g < 0 then
10         create clauses for fan-out constraints and
             primary outputs' fan-in constraints.
11         sol.push() // create a snapshot
12     else
13         create clauses for garbage output constraints.
14     if sol.check() == SAT then
15         create RQFP logic circuit cir by sol.model().
16         calculate the garbage number g_c of cir.
17         if g_c ≤ g_lb then
18             break
19         else
20             g = g_c − 1
21             sol.pop() // remove all clauses added after
                 the last snapshot
22     else
23         if g ≥ 0 ∧ g ≤ g_lb then
24             break
25         else if g > 0 then
26             g −= 1
27             sol.pop() // remove all clauses added after
                 the last snapshot
28         else
29             sol.pop(2) // remove all clauses added after
                 the penultimate snapshot
30             r += 1 // new a RQFP logic gate
31             create variables for new gate x_{n+r}.
32             create clauses for constraints of x_{n+r}.
33             sol.push() // create a snapshot
34 insert RQFP buffers for cir
35 return cir
```

gate and the connection relationship between primary inputs, primary outputs, and gates. Subsequently, the corresponding RQFP logic circuit can be generated. Algorithm 1 summarizes the complete flow of exact logic synthesis for RQFP logic. This algorithm iteratively constructs the propositional logic model in an incremental manner.

For an input circuit with n primary inputs and m primary outputs, the initial number of RQFP logic gates, subject to the single fan-in and single fan-out constraints, can be calculated as $r = \lceil \max(n, m)/3.0 \rceil$. To begin with, the number g of garbage outputs is initialized to -1, indicating that garbage output constraints are not taken into account until the first feasible solution is obtained. Next, variables and constraints are established for r gates based on the initial parameters (lines 4-7), followed by the iterative process (lines 8-33). In cases where the first feasible solution is not obtained, fan-out constraints and primary outputs' fan-in constraints are added (lines 9-11). Conversely, garbage output constraints are introduced (line 13) if a feasible solution has already been obtained (lines 15-16). After the verification of the model's satisfiability, if a feasible solution exists, an RQFP logic circuit is generated, and the value g is reduced by one to optimize the garbage outputs (lines 15-21). In the absence of a feasible solution, if a feasible solution has been previously discovered, the value g also decreases by one (lines 25-27); Otherwise, a new RQFP logic gate is introduced (lines 29-33). Ultimately, the iterative process ends when the value of g is less than or equal to the specified lower limit g_{lb} of garbage outputs (lines 17-18,23-24) under the given number r of RQFP logic gates.

After the generation of the RQFP logic circuit, some RQFP logic gates may not still conform to the path balancing requirement. Hence, RQFP buffers presented in Section II-A1 must be inserted to ensure that all inputs to each gate possess the same clock phases. Fig. 4 shows the buffer insertion result for the generated 3-8 decoder.

C. Optimizations

All works described above focus on finding a correct solution. However, since symmetries pervade an extensive search space [20], many equivalent solutions exist. To mitigate this, four symmetry-breaking optimization strategies are proposed to reduce the solving time.

1) Splitter Limitation (SL): As mentioned previously, an RQFP logic gate requires two constant inputs if it functions as a splitter. However, the introduction of two constant inputs gives rise to a significant degree of redundancy. For instance, $R(1,1,a) = (M(\overline{1},1,a), M(\overline{1},1,a), M(\overline{1},1,a)) = (a,a,a)$ and $R(1,1,a) = (M(\overline{1},1,a), M(1,\overline{1},a), M(\overline{1},1,a)) = (a,a,a)$ are both capable of representing a 1-to-3 splitter. Besides, when an RQFP splitter possesses merely one fan-out or sources from an RQFP logic gate with less than three fan-outs, it can lead to a substantial increase in the search space. Hence, the inverter configuration, input choice, and fan-out limitation of the RQFP splitter need the following constraints.

- Input constraints:

$$s_{i,j(p)} \implies \bigwedge_{q\in[1,3]\wedge q\neq p} \left(\bigvee_{k=[j+1,n+r+m]} c_{k,j(q)} \right), \quad (12)$$

which means that only if the fan-out limitation is violated can RQFP splitters be applied, i.e., the predecessor RQFP logic gate x_j of each splitter x_i must have three outputs.

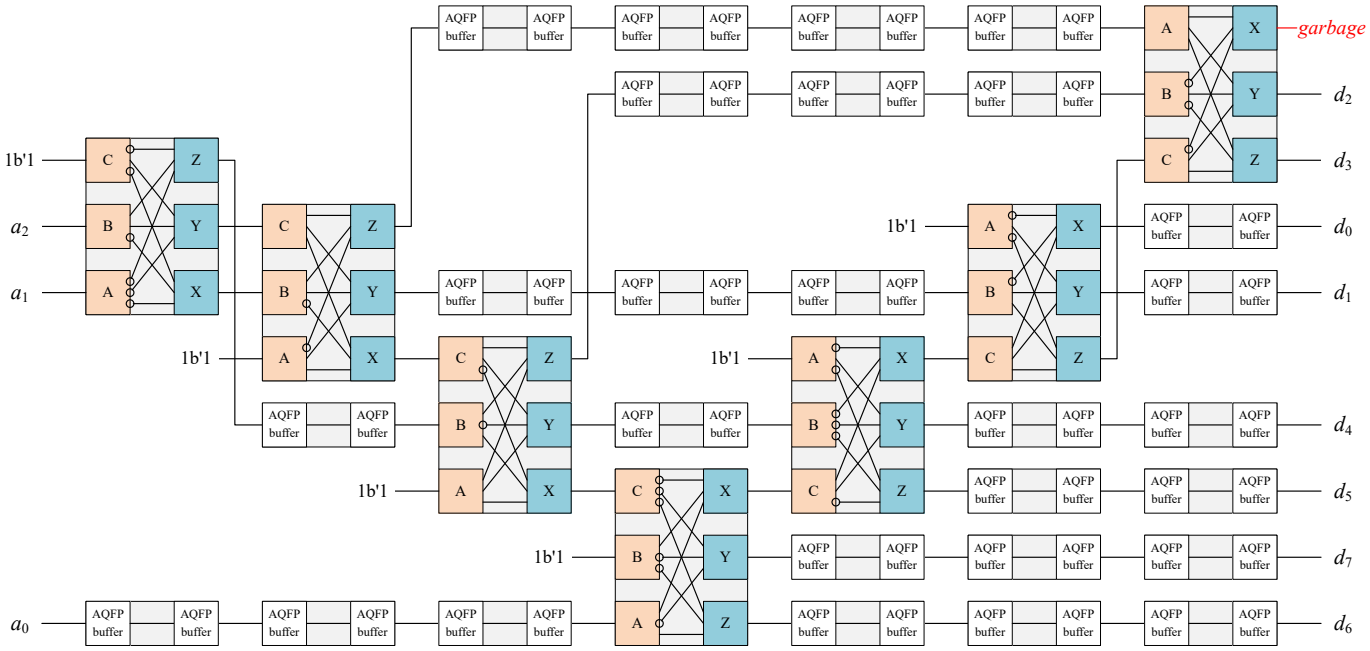


Fig. 4. Schematic of the generated RQFP-based 3-to-8 decoder with seven RQFP logic gates and one garbage output after the buffer insertion.

- Function constraints:

$$s_{i,j(p)} \implies \bigwedge_{q\in[1,3]} \left(f_{i(q,1)} \wedge \overline{f_{i(q,2)}}\right), \tag{13}$$

which ensures that none of the outputs of each splitter can be a constant. Besides, if the q^{th} output port of splitter x_i is not connected to any primary output, its output function can be further restricted to operate solely as a buffer by

$$s_{i,j(p)} \wedge \left(\bigwedge_{k\in[n+r+1,n+r+m]} \overline{c_{k,i(q)}}\right) \implies f_{i(q,3)}. \tag{14}$$

- Fan-out constraints:

$$s_{i,j(p)} \implies \bigwedge_{q\in[2,3]} \left(\bigvee_{k=[i+1,n+r+m]} c_{k,i(q)}\right), \tag{15}$$

which ensures that the last two output ports of each splitter must have a fan-out, meaning that gate x_i does indeed function as a splitter.

2) Order Limitation of Primary Outputs (OLPO): In the circuit design, the order of primary output is commonly not fixed, which poses a challenge to exact logic synthesis. For instance, the circuit only needs to swap these two output ports' inverter setting if $(o_1, o_2, o_3) = R(x_1, x_2, x_3)$ is replaced with $(o_1, o_3, o_2) = R(x_1, x_2, x_3)$. To address this issue, this paper employs a strategy that initially sorts all primary outputs in terms of the topological order of the circuit before the construction of the propositional logic model. Then, the following constraint is imposed on adjacent primary outputs.

$$\bigwedge_{i\in[n+r+2,n+r+m]} \left(c_{i,j(p)} \implies \bigvee_{\substack{(k==j \implies q<p)\\ \wedge k\le j}} c_{i-1,k(q)}\right), \tag{16}$$

which ensures that a primary output with a bigger index possesses a predecessor with a bigger index.

Moreover, there is a tip to further limit the search space, i.e., let the last primary output o_m connected to the last output port of the last RQFP logic gate x_{n+r} by $c_{n+r+m,(n+r)(3)} \oplus \overline{\textbf{true}}$.

3) Fan-in and Fan-out Order Limitation (FIFO): Since the RQFP logic gate has three input ports and three output ports, symmetries exist in its fan-ins and fan-outs. If gate x_i is not connected to gate x_{i+1}, they can be executed in any order, e.g., the circuit remains the same functionality if $(x_{7(1)}, x_{7(2)}, x_{7(3)}) = R(x_4, x_5, x_6)$ and $(x_{8(1)}, x_{8(2)}, x_{8(3)}) = R(x_1, x_2, x_3)$ are replaced with $(x_{7(1)}, x_{7(2)}, x_{7(3)}) = R(x_1, x_2, x_3)$ and $(x_{8(1)}, x_{8(2)}, x_{8(3)}) = R(x_4, x_5, x_6)$. To address this issue, the fan-in order of adjacent RQFP logic gates is limited by

$$\left(c_{i,j(p_1)} \wedge c_{i,k(p_2)} \wedge c_{i,l(p_3)}\right) \implies \bigwedge_{p\in[1,3]} \left(\bigwedge_{\substack{t\ge\max(l,n)\wedge p\in[1,3]\\ \wedge(t==l \implies p\ge p_3)}} \overline{c_{i-1,t(p)}}\right), \tag{17}$$

which ensures that the indexes of all fan-ins of the former gate must be smaller than the fan-in index of the last output port of the later gate. Therefore, only the second order in the aforementioned example is applied due to $3 < 6$.

Furthermore, similar to fan-ins of the RQFP logic gate, fan-outs of the RQFP logic gate also exhibit a comparable symmetry. Specifically, the three majorities in the RQFP logic gate have the same inputs, and their functions depend on the inverter configuration, enabling them to produce the same function output. Consequently, the output order of these three

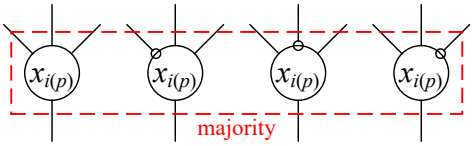


Fig. 5. Four unique majority functions considering the inverter transitivity.

output ports can be constrained by the following constraint

$$\begin{aligned}&\left(\bigvee_{i=[j+1,n+r+m]} c_{i,j(3)}\right) \quad \wedge \\ &\left(\bigvee_{i=[j+1,n+r+m]} c_{i,j(1)} \implies \bigvee_{i=[j+1,n+r+m]} c_{i,j(2)}\right) \quad \wedge \\ &\left(\bigwedge_{i=[j+1,n+r+m]} \overline{c_{i,j(2)}} \implies \bigwedge_{i=[j+1,n+r+m]} \overline{c_{i,j(1)}}\right) \quad ,\end{aligned} \tag{18}$$

which ensures that the first output port of gate x_j has a fan-out only if the second output port of gate x_j has a fan-out, and as soon as the second output port of gate x_j has no fan-out, the first output port of gate x_j also has no fan-out. Moreover, to ensure that each RQFP logic gate is utilized, the last output port of each RQFP logic gate must have a fan-out.

Apart from whether the output port has fan-out, the successor order of three output ports of each RQFP logic gate also needs to constrain by the following constraint

$$\bigwedge_{p\in[2,3]} \left(\bigwedge_{i\in[j+1,n+r]} \left(c_{i,j(p)} \implies \bigwedge_{k\in[i+1,n+r]} \overline{c_{k,j(p-1)}}\right)\right), \tag{19}$$

which ensures that the successor index of the p^{th} output port of gate x_j must be larger than that of the $(p-1)^{\text{th}}$ output port of gate x_j. Note that the primary outputs are not in these successors since each RQFP logic gate can be connected to a primary output.

4) Inverter Limitation (IL): Thus far, the optimization strategies discussed have centered on the topological structure and connection relationships within the RQFP logic circuit. In fact, symmetry also exists in the functionality of RQFP logic gates, as inverters enable these gates to possess flexible functionality, which increases the complexity of exact logic synthesis. Each output port of the RQFP logic gate can generate eight different output functions by configuring the inverters. Nonetheless, due to the inverter transitivity, these functions can be condensed into four unique functions, as shown in Fig. 5. To address this issue, the constraint concerning the inverter configuration is introduced, expressed as

$$\begin{aligned}&c_{i,j(p_1)} \wedge c_{i,k(p_2)} \wedge c_{i,l(p_3)} \wedge \left(\bigvee_{t=[i+1,n+r]} c_{t,i(p)}\right) \implies \\ &\left(\overline{f_{i(p,1)}} \vee f_{i(p,2)} \vee \overline{f_{i(p,3)}}\right) \wedge \left(f_{i(p,1)} \vee \overline{f_{i(p,2)}} \vee f_{i(p,3)}\right) \wedge \\ &\left(f_{i(p,1)} \vee f_{i(p,2)} \vee \overline{f_{i(p,3)}}\right) \wedge \left(f_{i(p,1)} \vee f_{i(p,2)} \vee f_{i(p,3)}\right),\end{aligned} \tag{20}$$

which applies specifically to the p^{th} output port of RQFP logic gate x_i not connected to any primary outputs.

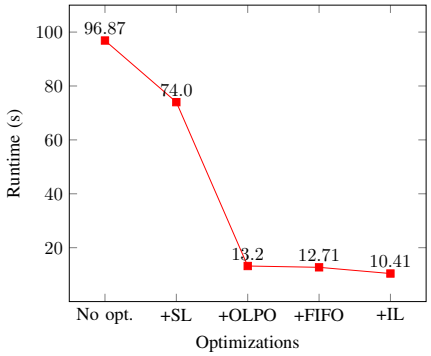


Fig. 6. Variation in the runtime with different optimization methods applied to the exact logic synthesis for the 2-to-4 decoder, where each method is cumulatively applied based on all previous ones.

Besides, when an RQFP logic gate x_i has the same two inputs $x_{j(p_1)}, x_{k(p_2)}$ on the value, i.e., $\bigwedge_{t\in[1,2^n]} x_{j(p_1)[t]} \oplus \overline{x_{k(p_2)[t]}}$, its output function can take two forms. If the inverters corresponding to the two inputs are the same, the gate output depends on these two inputs, in which case another inverter is useless. Conversely, the gate output depends on another input, in which case these two inverters are useless. So, the following constraint concerning the inverter configuration is introduced:

$$\begin{aligned}&c_{i,j(p_1)} \wedge c_{i,k(p_2)} \wedge c_{i,l(p_3)} \wedge \left(\bigwedge_{t\in[1,2^n]} x_{j(p_1)[t]} \oplus \overline{x_{k(p_2)[t]}}\right) \implies \\ &\bigwedge_{p\in[1,3]} \left(\left(\bigvee_{t=[i+1,n+r]} c_{t,i(p)}\right) \implies f_{i(p,3)} \wedge \left(f_{i(p,1)} \implies f_{i(p,2)}\right)\right).\end{aligned} \tag{21}$$

Similarly, when an RQFP logic gate x_i has the complementary inputs $x_{j(p_1)}, x_{k(p_2)}$ on the value, the inverters can also be limited by the following constraint

$$\begin{aligned}&c_{i,j(p_1)} \wedge c_{i,k(p_2)} \wedge c_{i,l(p_3)} \wedge \left(\bigwedge_{t\in[1,2^n]} x_{j(p_1)[t]} \oplus x_{k(p_2)[t]}\right) \implies \\ &\bigwedge_{p\in[1,3]} \left(\left(\bigvee_{t=[i+1,n+r]} c_{t,i(p)}\right) \implies f_{i(p,3)} \wedge \left(f_{i(p,1)} \implies \overline{f_{i(p,2)}}\right)\right).\end{aligned} \tag{22}$$

Moreover, when the p^{th} output port of gate x_i has no fan-out, the inverter setting for this port is insensitive, and thus, there is no inverter for this port set by default, as specified by the following constraint

$$\bigwedge_{t=[i+1,n+r+m]} \overline{c_{t,i(p)}} \implies \bigwedge_{j=[1,3]} f_{i(p,j)}. \tag{23}$$

As illustrated in Fig. 6, when all optimization methods are progressively applied to the exact logic synthesis for the 2-to-4 decoder, the runtime decreases from 96.87 s without any optimization applied to 10.41 s with all optimizations applied. Among them, the effects of SL and OLPO are very significant.

IV. Experimental Results

The proposed exact logic synthesis algorithm for RQFP logic is achieved by Python, and the Z3 solver [19] is used to solve the model. After the model construction of each iteration, the "simplify" tactic is first used to simplify the

TABLE I
EXPERIMENTAL RESULTS OF EXACT LOGIC SYNTHESIS FOR RQFP LOGIC.

Testcase	Original			Baseline					Exact logic synthesis					
	PI	PO	g_{lb}	RQFP	Buffer	JJ	Depth	Garbage	RQFP	Buffer	JJ	Depth	Garbage	Speedup ratio
1-bit full adder	3	2	1	15	7	388	5	21	3	2	80	3	2	4.18
4gt10	4	1	3	4	3	108	3	7	3	4	88	3	5	4.93
alu	5	1	4	12	10	328	6	19	4	7	124	4	5	1.07
c17	5	2	3	8	6	216	4	13	5	14	176	5	5	/
decoder_2_4	2	4	0	8	3	204	3	10	3	3	84	3	1	11.50
decoder_3_8	3	8	0	20	10	520	4	23	7	27	276	7	1	/
graycode4	4	4	0	15	7	388	4	21	6	4	160	4	2	/
ham3	3	3	0	17	1	412	4	22	5	5	140	5	1	53.48
mux4	6	1	5	12	12	336	6	20	6	3	156	4	7	/

* '/' represents that the proposed algorithm without optimizations can not find a feasible solution within 240000 seconds.

model by applying simplification rules. Then, the "card2bv" tactic is used to convert pseudo-Boolean constraints to bit-vectors, especially for cardinality constraints. Next, the "bit-blast" tactic is used to reduce bit-vector expressions into SAT. Finally, the "aig" tactic is used to simplify Boolean structure using AIG, and the simplified model is solved by the "sat" tactic, from whose results the corresponding RQFP logic circuit is obtained. Besides, to ensure the consistency of each experimental result in terms of processing the path balancing requirement for primary inputs and primary outputs of the generated circuit, RQFP buffers are inserted into primary inputs and primary outputs such that all primary inputs and all primary outputs are respectively in the same clock stage.

The experiments use the RevLib benchmark circuits [16] and are executed on the machine with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, GeForce RTX 3090, and 256.0 GB memory running Ubuntu 22.04. The glucose-syrup solver is used to speed up the solving process, which is implemented by a library named GpuShareSat [21] and uses the GPU via CUDA to help different CPU threads to share clauses with each other. After using the "aig" tactic to simplify the model, the "tseitin-cnf" tactic in the Z3 solver can convert the model into the CNF using tseitin-like encoding [22]. Finally, the glucose-syrup solver is applied to solve the CNF, further obtaining the corresponding RQFP logic circuit. The baseline is designed by a heuristic method. This method first does MIG-based logic optimization by the LSOracle [23], a state-of-the-art logic optimization tool that supports AIG and MIG-based optimization. Then, it converts each majority node into an RQFP logic gate. If the fan-out of the majority node exceeds three, RQFP splitters will be inserted. Finally, it processes the buffer insertion to meet the path-balancing requirement.

Table I shows the experimental results. The "Original" part shows the attributes of input circuits, including the number "PI" of primary inputs, the number "PO" of primary outputs, and the lower bound "g_{lb}" of garbage outputs. The "Baseline" part shows the results of the baseline method. "RQFP" represents the number of used RQFP logic gates, and "Buffer" represents the number of RQFP buffers inserted for the path-balancing requirement. "JJ" represents the JJ count. Since JJs are a vital component of AQFP circuits and are used to create the fluxons that are used for computation, their count can give an estimate of the complexity and energy efficiency of a given AQFP circuit, thereby becoming an important metric of AQFP logic circuits. So, the JJ count can also be used as a cost metric of RQFP logic circuits realized by AQFP logic. In current AQFP logic circuits, both buffer and splitter have 2 JJs, and 3-input MAJ has 6 JJs. So, there are 24 JJs per RQFP logic gate and 4 JJs per RQFP buffer. "Depth" and "Garbage" represents 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 proposed exact logic synthesis algorithm for RQFP logic. The "Speedup ratio" indicates the runtime ratio between the proposed algorithm without and with optimizations, where the reason why the speedup of the "alu" circuit is not obvious is that the proposed algorithm with optimizations finds an extra feasible solution than one without optimizations. Compared to the baseline, the proposed algorithm has a significant reduction in the number of RQFP logic gates, JJ count, and the number of garbage outputs, specifically by 57.47%, 51.41%, and 76.76%, respectively. The experimental results show that the proposed algorithm can generate RQFP logic circuits with a smaller number of RQFP logic gates and garbage outputs.

V. CONCLUSION

This paper introduced the logic synthesis problem of RQFP logic, including how to construct RQFP-based buffers and splitters to address the path-balancing requirement and fan-out limitation. Meanwhile, this paper first proposed a novel exact logic synthesis algorithm for RQFP logic, which can generate the corresponding RQFP logic circuit for an RTL description. The experimental results on the RevLib benchmark circuits [16] show the effectiveness of the proposed algorithm, making the number of garbage outputs close to its lower bound when the number of RQFP logic gates is the least. Besides, due to the applicable limitation of exact logic synthesis by the complexity of Boolean functions, the logic synthesis methods for larger RQFP logic circuits will be further explored.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Professor Robert Wille at the Technical University of Munich, Germany, for his valuable guidance and support throughout the research process. The research work described in this paper was conducted in the Intelligent Design Automation Lab funded by The Hong Kong Jockey Club Charities Trust.

REFERENCES

[1] D. Mamaluy and X. Gao, "The fundamental downscaling limit of field effect transistors," *Applied Physics Letters*, vol. 106, no. 19, 2015.

[2] A. P. Jacob, R. Xie, M. G. Sung, L. Liebmann, R. T. P. Lee, and B. Taylor, "Scaling challenges for advanced CMOS devices," *International Journal of High Speed Electronics and Systems*, vol. 26, no. 01n02, p. 1740001, 2017.

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

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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 *2021 IEEE/ACM International Conference On Computer Aided Design*, pp. 1–8, 2021.

[14] S.-Y. Lee, H. Riener, and G. De Micheli, "Beyond local optimality of buffer and splitter insertion for AQFP circuits," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 445–450, 2022.

[15] 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 *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 769–774, 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 *Proceedings of the 38th International Symposium on Multiple Valued Logic*, pp. 220–225, 2008. RevLib is available at http://www.revlib.org.

[17] D. E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.

[18] A. Sülflow, R. Wille, G. Fey, and R. Drechsler, "Evaluation of cardinality constraints on SMT-based debugging," in *Proceedings of the 39th International Symposium on Multiple-Valued Logic*, pp. 298–303, 2009.

[19] 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*, pp. 337–340, 2008.

[20] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015.

[21] N. Prevot, M. Soos, and K. S. Meel, "Leveraging GPUs for effective clause sharing in parallel SAT solving," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 471–487, 2021.

[22] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic Part 2*, Seminars in mathematics, pp. 115–125, Springer US, 1970.

[23] "LSOracle." https://github.com/lnis-uofu/LSOracle. Accessed on April 2023.