Synthesis of SFQ Circuits with Compound Gates

Rassul Bairamkulov
Integrated Systems Laboratory
EPFL, Lausanne, Switzerland
rassul.bairamkulov@epfl.ch

Alessandro Tempia Calvino
Integrated Systems Laboratory
EPFL, Lausanne, Switzerland
alessandro.tempiacalvino@epfl.ch

Giovanni De Micheli
Integrated Systems Laboratory
EPFL, Lausanne, Switzerland
giovanni.demicheli@epfl.ch

***Abstract*—Rapid single-flux quantum (RSFQ) is one of the most advanced superconducting technologies with the potential to supplement or replace conventional VLSI systems. However, scaling RSFQ systems up to VLSI complexity is challenging due to fundamental differences between RSFQ and CMOS technologies. Due to the pulse-based nature of the technology, RSFQ systems require gate-level pipelining. Moreover, logic gates have an extremely limited driving capacity. Path balancing and clock distribution constitute a major overhead, often doubling the size of circuits. Gate compounding is a novel technique that substantially enriches the functionality realizable within a single clock cycle. However, standard logic synthesis tools do not support its specific synchronization constraints. In this paper, we build first a database of minimum-area compound gates covering all the Boolean functions up to 4 variables and all possible input arrival patterns. Then, we propose a technology mapping method for RSFQ circuits that exploits compound gates using the database as a cell library. We evaluate our framework over the EPFL and ISCAS benchmark circuits. Our results show, on average, a 33% lower logic depth with 24% smaller area, as compared to the state of the art.**

I. Introduction

Rapid Single-Flux Quantum (RSFQ) [1] is one of the most promising beyond-CMOS technologies. RSFQ systems consistently achieve operating frequencies on the order of tens of gigahertz [2]–[4], with particular cells operating at hundreds of gigahertz [5]–[7]. Furthermore, the operating power of the RSFQ systems is two to three orders of magnitude smaller than CMOS, even considering the refrigeration power [8].

However, achieving the aforementioned advantages at scale remains a challenge. Unlike CMOS, most RSFQ logic gates operate as latches with one clock input and one or more data inputs [9]. Arrival of a *single-flux quantum* (SFQ) pulse at the data input changes the internal state of the gate. The presence or absence of an SFQ pulse within the clock period represents logical `1` or `0`, respectively. The clock pulse resets the gate to initial state, potentially releasing an SFQ pulse. This reliance on the clock signal requires SFQ circuits to be pipelined at the gate level. To ensure a correct data propagation, i.e., correct data arrives in the correct time frame, path balancing is required, as shown by the two path-balancing D-flip-flops (DFF) in Fig. 1b. Furthermore, due to the quantized nature of SFQ pulses, most RSFQ primitives have a maximum driving capacity of one gate. Consequently, a special cell called *splitter* is used to duplicate signals [9], [10], as illustrated in Fig. 1.

Despite the advances in RSFQ technology mapping [11], [12], the number of path-balancing DFFs and splitters can be prohibitively large, degrading the area and yield of an integrated system [13]. Different approaches have been proposed in the literature to tackle this fundamental issue. In [11], the number of path-balancing DFFs is reduced using dynamic programming, yielding, on average a 12% smaller area. Further reductions in path-balancing overhead is achieved by using dual clocking, where high- and low- frequency clock signals are used [14]. This technique however requires relatively expensive NDRO DFFs along with the duplication of the clock distribution network.

Different techniques to reduce the number of clocked elements are proposed in the literature [15], [16]. In dynamic SFQ (DSFQ) the gates reset to the initial state after the specified period of time [17]. The design of DSFQ circuits is therefore similar to CMOS circuits where large combinational blocks can be synchronized using relatively few synchronous elements [10]. A similar approach based on clockless logic gates is proposed in [2]. Based on nondestructive readout (NDRO) flip-flops, two additional clockless cells, namely the `NIMPLY` ($\neg x_0 \land x_1$) and the `AND` functions, are efficiently realized using fewer clocked elements for synchronization. The advantages of these approaches are smaller area, lower clock network complexity, and simpler path balancing, as compared to conventional RSFQ. The timing constraints, however, constitute a major challenge. In DSFQ, the interaction between the input skew tolerance, clock frequency, and bias margins [10] complicates the circuit design. The NDRO-based clockless gates are particularly sensitive to the arrival time of the inputs, necessitating careful timing analysis [18].

The *gate compounding* technique has been recently proposed as an alternative strategy to reduce the number of clocked elements [19]. Unlike DSFQ and clockless gates, *compound gates* (logic gates obtained by gate compounding) are not sensitive to the arrival of the inputs, reducing the complexity of the system design process. The functionality achievable within a single clock cycle is enriched by exploiting RSFQ synchronization mechanisms. Gate compounding can significantly reduce the pipeline depth and number of clocked elements, not only improving the latency and area of a functional circuit, but also reducing the size of the clock

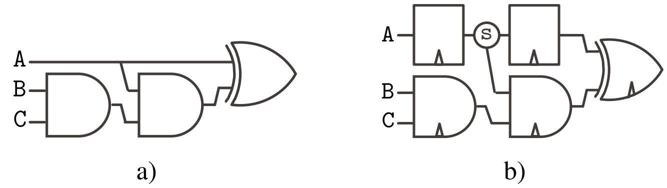

Fig. 1. a) An example of a CMOS circuit. b) Equivalent RSFQ circuit with a splitter and two path-balancing DFFs.

This research was supported by the SNF grant "Supercool: Design methods and tools for superconducting electronics", 200021 1920981.

distribution network. However, due to complex synchronization requirements, traditional technology mapping tools are not directly applicable.

In this work, inspired by [20]–[22], we present a technology mapping method for SFQ compound gates based on a pre-computed database. Using enumeration, we generate functionally correct and area-optimal compound gates for all functions up to four variables and all possible input arrival patterns. Next, we utilize these gates as cells during technology mapping to synthesize large scale SFQ circuits. In the experimental results, we show a drastic reduction in the area and logic depth by 24% and 33%, respectively, compared to the state-of-the-art.

II. Gate compounding technique

The gate compounding technique exploits differences in pulse synchronization mechanisms to reduce the pipeline depth of an RSFQ circuit. In particular, RSFQ logic gates can be divided into three categories, namely, *AA*, *AS*, and *SA*, where the first letter denotes whether input signals should arrive (a)synchronously, while the second letter indicates whether the output is released (a)synchronously.

AA elements process the inputs immediately upon arrival and the output is released without a synchronizing signal (clock). For instance, a *merger* cell, often referred to as *confluence buffer* (CB), directs signals from multiple (typically two) input branches into one output branch, i.e., implements an $\texttt{OR}$ function. Note that the merger produces two subsequent output pulses if input pulses are temporally separated, or a single pulse, if input signals arrive simultaneously.

AS elements process the input information immediately upon arrival and release the output synchronously after the arrival of the clock signal. The simplest RSFQ component of this type is *D-flip-flop (DFF)* that stores an incoming pulse and releases it upon the arrival of the clock signal. Other important AS elements are the *inverter ($\texttt{NOT}$)* and *exclusive-or ($\texttt{XOR}$)*.

SA elements require the inputs to arrive simultaneously. The result of the computation is released immediately after processing. Assuming inputs arrive simultaneously, a CB can be tuned to produce at most a single output pulse, producing an $\texttt{OR}$ element [23]. Furthermore, by adjusting the JJ size and bias current, the $\texttt{OR}$ structure can be transformed into $\texttt{AND}$ element. Note that, unlike conventional RSFQ, $\texttt{OR}$ and $\texttt{AND}$ elements are not clocked and require inputs to arrive simultaneously.

These three categories of components govern the flow of data within an RSFQ circuit. Most importantly, the SA components ensure simultaneous release of the SFQ pulses. Therefore, SA components can only be placed directly after the AS elements. To comply with these restrictions, the *gate compounding* technique was proposed in [19]. A compound SFQ logic gate can be produced by following the generic structure illustrated in Fig. 2. Inputs to a compound gate are initially processed by AA elements. The signals then flow towards the AS components where the result of a logical operation is stored until the arrival of the clock signal. The clock signal triggers the simultaneous release of the data towards the SA elements. Finally, the AA components complete the function.

The proposed structure offers two major advantages. Since the initial processing is handled by the AA or AS elements,

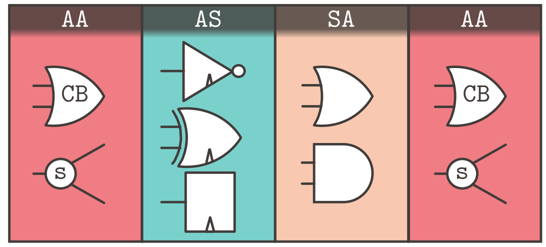

Fig. 2. Generic compound gate structure.

arbitrary order of input arrival is supported, relaxing the timing constraints of the circuit. The proposed gate compounding technique significantly expands the set of functions realizable within a single clock cycle. Using compound gates, for example, all 16 two-input functions are realized within a single clock cycle, as compared to only 13 functions in conventional SFQ [19].

III. Background and Notation

A multi-output Boolean function $f : \mathbb{B}^k \mapsto \mathbb{B}^m$ maps k input signals to m output signals. A single output Boolean function ($m = 1$) $f : \mathbb{B}^k \mapsto \mathbb{B}$ can be represented as a truth table with 2^k rows. A truth table can be conveniently encoded as a 2^k-bit string $\texttt{Y} = \overline{\texttt{y}_{2^k-1} \ldots \texttt{y}_0}$ where bit $\texttt{y}_i$ denotes the output at the i^{th} row in the truth table. For example, $f_1(\texttt{x}_1, \texttt{x}_0) = \texttt{x}_1 \oplus \texttt{x}_0$ is encoded as $\texttt{Y}_1 = \texttt{0110}_2$, since $f_1(\texttt{1},\texttt{1}) = \texttt{0}$, $f_1(\texttt{1},\texttt{0}) = \texttt{1}$, $f_1(\texttt{0},\texttt{1}) = \texttt{1}$, and $f_1(\texttt{0},\texttt{0}) = \texttt{0}$.

A *Boolean function*[1] f can be represented by a *Boolean network*[2] $\mathcal{N} = (\mathcal{V} = \mathcal{I} \cup \mathcal{O} \cup \mathcal{G}, \mathcal{E})$ — a directed acyclic graph (DAG) representing the sequence of the Boolean operations applied to realize f. Set $\mathcal{G}$ is a set of gates, where each node $u \in \mathcal{G}$ applies a function f_u to its *fanins FI*(u) and passes the result to *fanouts FO*(u). Set $\mathcal{I}$ denotes the set of *primary inputs* (PI), i.e., nodes without fanins. Set $\mathcal{O}$ denotes the set of *primary outputs* (PO), i.e., nodes without fanouts.

A. Delay

In SFQ, the delay is typically expressed in terms of the number of clock cycles required to realize a function. In practice, input signals can often arrive at different clock cycles, as illustrated in Fig. 3a. We define the *input level pattern* $\boldsymbol{\ell}_{\mathcal{N}} = \left[\ell^0, \ldots, \ell^{k-1}\right]$ as a vector of integers describing the clock cycles during which the PI signals enter the network $\mathcal{N}$. Without loss of generality, we normalize the input patterns such that the earliest PI signal arrives at cycle 0, i.e., $\min(\boldsymbol{\ell}_N) = 0$. For example, an input level pattern $\boldsymbol{\ell}_{\mathcal{N}} = [0, 1]$ indicates that the data from the second PI is delayed by one clock cycle. A *level* l_u denotes the number of clock cycles between the earliest PI and node u. The *input arrival pattern* $\mathbf{d}_u = \left[d_u^0 \ldots d_u^{k-1}\right]$ is the number of clock cycles between u and each PI,

$$\mathbf{d}_u = [l_u - \ell^0, \ldots, l_u - \ell^{k-1}].$$

We define two operators to compare the delay patterns of any two nodes u and v:

$$\mathbf{d}_u = \mathbf{d}_v \Leftrightarrow \forall i \; d_u^i = d_v^i,$$
$$\mathbf{d}_u < \mathbf{d}_v \Leftrightarrow \exists i \; d_u^i < d_v^i \text{ and } \nexists i \; d_u^i > d_v^i.$$

[1]For brevity, we use the term *function* to represent a *Boolean function*
[2]We use the terms *network* and *circuit* to represent a *Boolean network*

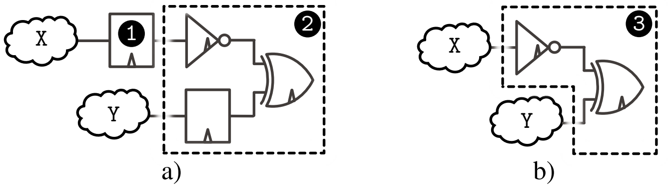

Fig. 3. Realization of an XNOR function between networks X and Y. The left network uses a path-balancing DFF (1) followed by an XNOR with equal delay pattern (2). This structure requires three clock cycles and 33 JJs. The right network uses an XNOR element with unequal delay pattern (3), requiring two clock cycles and 21 JJs.

In the former case, corresponding delays are equal. In the latter case, the delays of u are not greater than the corresponding delays of v, but for at least one PI the delay of u is smaller.

B. Cost

The most common metric to evaluate the cost of an SFQ circuit is the JJ count, which directly correlates with the area of an SFQ circuit. Let q_u be the area cost associated with the logic primitive implemented by a node u. The area cost $c(\mathcal{N})$ of a circuit $\mathcal{N}$ is the sum of costs q_u for each node $u \in \mathcal{G}$. A transitive fanin cone $TFI(u)$ is defined as the set of all nodes having a path to u. The area cost c_u of a node u is defined as the cost of its TFI. Note that c_u differs from q_u, since q_u defines the cost of a single primitive, while c_u is the sum of costs of all ancestors of u. Suppose nodes u, v are fanins of node w. The cost of the node w is,

$$c_w = q_w + S(u, v),$$

$$S(u,v) = \sum_{n \in TFI(u) \cup TFI(v)} \left[q_n + q_s \max\left(|FO(n)| - 1, 0 \right) \right],$$

where q_s is the cost of splitter.

An SFQ circuit should comply with specific technological constrains, such as path balancing and fanout constraints in SFQ. With SFQ gate compounding, gates also follow the structure described in Fig. 2 to avoid the data hazards described in the upcoming subsection.

C. Data hazards

1) Double pulse hazard. If two pulses entering a CB are sufficiently spaced in time, two subsequent SFQ pulses are generated at the output, potentially producing an error. For instance, a double pulse produced by a CB entering a XOR may trigger unwanted switching, producing incorrect result. In particular, the internal storage loop within a XOR is toggled one additional time between 0 or 1 by the input pulses. Nevertheless, if the CB has its output pin connected to a DFF or an inverter, the second pulse has no effect on the system [9].

Consider the circuit implementing $(A \vee B) \oplus C$ shown in Fig. 4. The storage loop within the XOR element is correctly switched and reset with pulses A and C. The pulse B, however, sets the storage loop to state 1, producing an incorrect result. To avoid this data hazard, the XOR component is placed after a CB only if the CB is guaranteed to produce at most one SFQ pulse, i.e., the inputs to a CB are never simultaneously equal to 1.

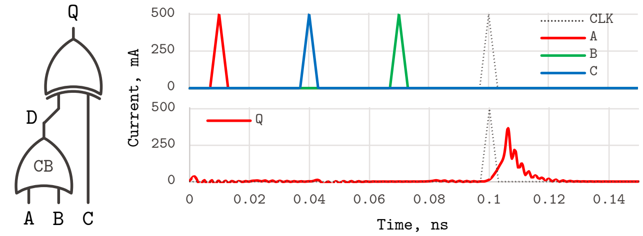

Fig. 4. Incorrect realization of $(\mathtt{A} \vee \mathtt{B}) \oplus \mathtt{C}$ function using a CB and an XOR. The main loop within an XOR element is set to 1 by A, reset to 0 by C, and subsequently set to 1 by pulse B, incorrectly producing an output pulse.

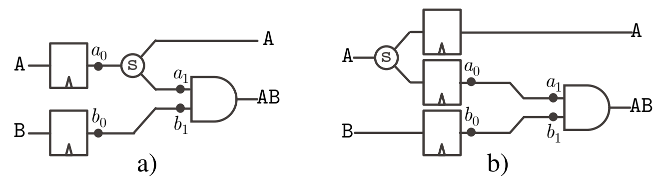

Fig. 5. a) A system violating the compound gate structure. Any AA element (splitter) between AS (DFF) and SA (AND) elements may desynchronize the input arrival. b) The issue is resolved by moving the splitter before the AS elements.

To identify the condition where a CB can produce two pulses, we assign a hazard flag $\mathtt{h}_n$ to each node n. If n is not a CB, $\mathtt{h}_n$ is 0; otherwise,

$$\mathtt{h}_n = \mathtt{h}_u \vee \mathtt{h}_v \vee \delta(\mathtt{Y}_u \wedge \mathtt{Y}_v),$$

where $u, v \in FI(n)$ and $\delta(\mathtt{Y}) = 1$ only if Y is nonzero.

For example, consider nodes u, v, w, with $\mathtt{Y}_u = 1010_2$, $\mathtt{Y}_v = 0001_2$, $\mathtt{Y}_w = 1100_2$, and $\mathtt{h}_u = \mathtt{h}_v = \mathtt{h}_w = 0$. Connecting u and v to a CB produces node p that can be used with XOR, since $\mathtt{h}_u = \mathtt{h}_v = 0$ and

$$\delta(\mathtt{Y}_u \wedge \mathtt{Y}_v) = \delta(1010_2 \wedge 0001_2) = \delta(0000_2) = 0 \Rightarrow \mathtt{h}_p = 0,$$

i.e., the u and v are never simultaneously equal to 1. In contrast, connecting u and w to a CB produces node q that cannot be used with XOR, since

$$\delta(\mathtt{Y}_p \wedge \mathtt{Y}_w) = \delta(1010_2 \wedge 1100_2) = \delta(1000_2) = 1 \Rightarrow \mathtt{h}_q = 1.$$

Suppose node r is produced by connecting q and v to a CB. Although $\delta(\mathtt{Y}_q \wedge \mathtt{Y}_v) = 0$, node r cannot be used with XOR since $\mathtt{h}_q = 1 \Rightarrow \mathtt{h}_r = 1$.

2) Desynchronization hazard. The signal desynchronization is a timing hazard where the inputs cannot simultaneously arrive to an SA element. Consider for example the circuit illustrated in Fig. 5a. The splitter is placed between the AS (DFF) and SA (AND) components. Delays $a_0 \rightarrow a_1$ and $b_0 \rightarrow b_1$ are not equal. Therefore, pulses from A and B do not arrive simultaneously, violating the input timing requirement of the AND element. Thus, the AND operates as a constant 0.

A possible correction is shown in Fig. 5b. The splitter is placed before the DFFs to equalize delays $a_0 \rightarrow a_1$ and $b_0 \rightarrow b_1$. The timing violation is therefore avoided at the cost of an additional DFF.

IV. Library construction

The fixed structure of the compound gates combined with the hazards described in the previous section complicates the technology mapping process. For example, AA elements should be prevented from being placed between AS and SA elements,

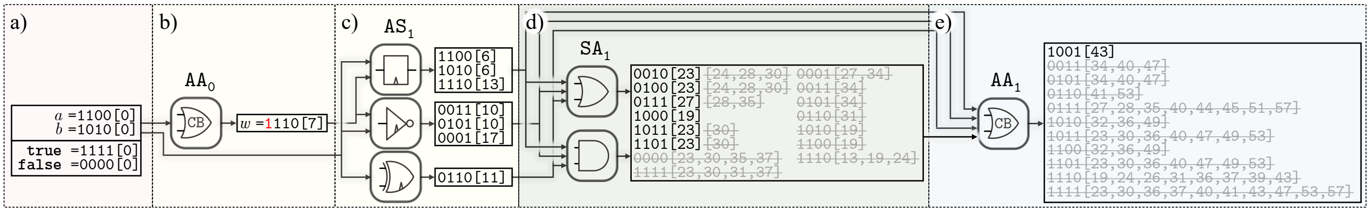

Fig. 6. Example of enumeration with 2 primary inputs represented by truth tables 1100 and 1010. The numbers in brackets represent the cost of a node (in JJ). The red 1 represents the double pulse hazard. The crossed grey numbers represent the discarded truth tables.

an issue described in section III-C2. Adapting the existing tools to consider these constraints requires to significantly modify the underlying algorithms, potentially degrading run time and quality of results.

Area- or delay-optimal SFQ circuits can be created using exact synthesis methods, such as Boolean satisfiability [24], [25] and enumeration [20]. However, exact methods are limited to small sizes (≤ 16 nodes) and few variables (≤ 6), due to the computational intractability of the problem. Nevertheless, exact synthesis can be applied to create a database of optimal small-scale structures. Since the number of Boolean functions grows double exponentially with the number of variables (2^{2^k}), complete databases are typically limited to 4 variables. These locally-optimal networks are subsequently used to produce larger networks [20]–[22]. Library-driven approaches have been successfully applied to MIG resynthesis [21], [22] and AQFP logic synthesis [20]. The database-driven mapping offers several advantages:

- *Functional correctness.* Each circuit block within a database describes a realization of a logic function complying with the specific technological constraints. Thus, technology mapping can safely proceed at the block level, since the technological requirements are satisfied during the database creation.
- *Local optimality.* The logic blocks in the database can be optimized for area or delay.
- *Performance.* The parameters of each logic block, such as area and delay, are computed in advance and can be accessed in constant time during mapping.
- *Reuse.* Once created, the database can be used multiple times to synthesize arbitrary SFQ circuits.

In this section, we present the procedure to create a database of area-optimal compound gate structures for each of the k-input, single-output Boolean function.

A. Enumeration procedure

The algorithm constructs a Boolean network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$, where nodes represent a particular realization of a logic function using compound gates. Each node $u = (\mathtt{Y}_u, l_u, c_u, \mathtt{h}_u) \in V$ is a 4-tuple of a truth table, level, cost, and hazard flag. The procedure is initialized with k nodes representing the PIs. For example, Fig. 6a describes the initialization for $k = 2$:

$$a = (\mathtt{1100}_2, 0, 0, \mathtt{0}) \qquad b = (\mathtt{1010}_2, 0, 0, \mathtt{0})$$

For completeness, constant `true` and `false` are also included. After initialization, the algorithm cycles through three subroutines, following the compound gate structure in Fig. 2.

1) AA. The stage $\mathtt{AA}_i$ implements the addition of AA elements to a compound gate at level i. For each pair of nodes $u = (\mathtt{Y}_u, i, c_u, \mathtt{h}_u)$ and $v = (\mathtt{Y}_v, i, c_v, \mathtt{h}_u)$, a new node $w = (\mathtt{Y}_u \vee \mathtt{Y}_v, i, q_{\mathtt{CB}} + S(c_u, c_v), \mathtt{h}_w)$ is produced. Consider the $\mathtt{AA}_1$ stage, illustrated in Fig. 6b, where the new node $w = (\mathtt{1110}_2, 0, 7, \mathtt{1})$ is discovered. The 7-JJ cost of the node is the cost of a CB used to realize this function.

2) AS. For each node $u = (\mathtt{Y}_u, i-1, c_u, \mathtt{h}_u)$, stage $\mathtt{AS}_i$ produces two new nodes,

$$p = (\mathtt{Y}_u, i, c_u + c_{\mathtt{DFF}}, \mathtt{0}) \quad \text{and} \quad q = (\neg \mathtt{Y}_u, i, c_u + c_{\mathtt{NOT}}, \mathtt{0}),$$

corresponding to addition of a DFF and NOT element. Note that the hazard flag is reset to 0, since only a single pulse is produced by the AS elements. In addition, for each pair of nodes $u = (\mathtt{Y}_u, i-1, c_u, \mathtt{0})$ and $v = (\mathtt{Y}_v, i-1, c_v, \mathtt{0})$, whose hazard flag is 0, a new node is produced

$$r = (\mathtt{Y}_u \oplus \mathtt{Y}_v, i, q_{\mathtt{XOR}} + S(c_u, c_v), \mathtt{0}).$$

In Fig. 6c, three new nodes are produced by a DFF, while four new truth tables are discovered by applying NOT and XOR operations. Note that the node w is not used with XOR due to the hazard flag $\mathtt{h}_w = 1$.

3) SA. After the AS stage, inputs are synchronized enabling the use of SA gates. At stage $\mathtt{SA}_i$, each pair of nodes $u = (\mathtt{Y}_u, i, c_u, \mathtt{0})$ and $v = (\mathtt{Y}_v, i, c_v, \mathtt{0})$ produces 2 new nodes

$$p = (\mathtt{Y}_u \wedge \mathtt{Y}_v, i, q_{\mathtt{AND}} + S(c_u, c_v), \mathtt{0}), \text{ and}$$
$$q = (\mathtt{Y}_u \vee \mathtt{Y}_v, i, q_{\mathtt{OR}} + S(c_u, c_v), \mathtt{0}).$$

In Fig. 6d, the outputs of the $\mathtt{AS}_1$ stage proceed to the $\mathtt{SA}_1$ stage where the logical AND and OR are applied to the outputs of the previous stage. The 6 nodes implementing previously undiscovered functions with smallest cost are added to the network, while 36 nodes are discarded.

The algorithm repeats these three stages ($\mathtt{AA}_{i-1} \rightarrow \mathtt{AS}_i \rightarrow \mathtt{SA}_i \rightarrow \mathtt{AA}_i \rightarrow \ldots$) until all 2^{2^k} k-input functions are realized. In our example for $k = 2$, after stage $\mathtt{SA}_1$ the algorithm proceeds to stage $\mathtt{AA}_1$, where 78 nodes are produced, of which only a single node implements the remaining function $\mathtt{1001}_2$. After this stage, all of the $2^{2^2} = 16$ two-input truth tables are discovered and the enumeration process is terminated.

B. Filtering

During the enumeration process, the size of the network grows rapidly with each additional stage. In Fig. 6, for example, only 7 nodes are produced at stage $\mathtt{AS}_1$, while 62 nodes are produced at stage $\mathtt{AA}_1$. The number of nodes considered during enumeration drastically increases with k, with several billions of nodes processed while enumerating four-input functions. To limit the number of nodes and prevent inferior nodes from being added to the database, the dominance relationship is used. Suppose, the node u implements a Boolean function f with

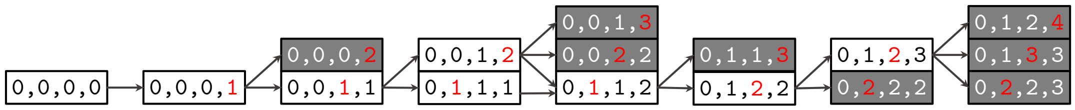

Fig. 7. Level patterns considered during enumeration. Due to permutation symmetry, only the sorted level patterns are considered. The process starts with the pattern $(0,0,0,0)$. In subsequent iterations, the level of one of the PIs is incremented (marked red). If the iteration does not yield any cost- or area-optimal nodes, the pattern is not incremented (shaded gray).

input arrival pattern $\mathbf{d}_u$ and area c_u. Also, suppose another node v implementing the same function f with input arrival pattern $\mathbf{d}_v$ and area c_v has previously been discovered. The node v is said to *dominate* the node u in two cases,

- faster delay: $\mathbf{d}_v < \mathbf{d}_u$ and $c_v \leq c_u$;
- lower cost: $\mathbf{d}_v = \mathbf{d}_u$ and $c_v < c_u$.

In these cases, the node u is not created.

C. Input arrival patterns

During initial enumeration, all PIs are placed at equal levels $\boldsymbol{\ell} = (0, \dots, 0)$. To consider different input arrival patterns, the enumeration process is repeated with PIs introduced at different levels $\boldsymbol{\ell} = \left(\ell^0, \dots, \ell^{k-1}\right)$. The number of input level patterns considered during the enumeration process can be reduced based on dominance relationship. Suppose that, while considering the pattern $\boldsymbol{\ell}_a = (\ell^1, \dots, \ell^q, \dots, \ell^k)$, all nodes were dominated by or equivalent to previously discovered nodes. The pattern $\boldsymbol{\ell}_b = (\ell^1, \dots, \ell^q + 1, \dots, \ell^k)$ is therefore unlikely to yield a non-dominated node, due to inferior delay and cost.

V. Technology Mapping

We propose a three-stage technology mapping flow to synthesize arbitrary Boolean networks using SFQ compound gates. First, we employ a delay-driven technology mapper that uses the computed database as a cell library. Due to path balancing, delay optimization is essential for area reduction in SFQ circuits. Intuitively, longer critical paths require more DFF elements due to longer paths to balance [28].

Next, our flow inserts path-balancing DFFs and minimizes their number using minimum-area retiming [29], which provides an optimal solution. Note that retiming preserves the path-balancing constraint since each path traverses the same number of DFFs before and after retiming.

Finally, splitter cells are inserted to satisfy the driving capacity constraint. Our synthesis flow has been implemented using the open-source logic synthesis library `mockturtle` [30].

VI. Experimental Results

We employed a computing cluster with 48 2.5GHz Intel Xeon E5-2680 CPUs and 256GB of RAM to create the database. Due to the computational complexity, we limited the number of inputs to four, i.e., $k = 4$. The enumeration process starts from pattern $\boldsymbol{\ell}_0 = (0,0,0,0)$, i.e., all of the PIs are at the same level. During the subsequent iterations, the level of one of the PIs is incremented and the enumeration process is repeated. If the enumeration does yield to non-dominated nodes, a new PI level is incremented. Fig. 7 illustrates possible level patterns considered by the enumeration process.

The computation for the input level pattern $\boldsymbol{\ell}_0 = (0,0,0,0)$ required seven hours, evaluating over 13 billion nodes. Other delay patterns required between one to five hours. The resulting database was created in 52 hours and consisted of 488,636 entries. Next, we filtered entries based on input-permutation equivalences (P-classes) [31]. Our final database contains 28,258 non-dominated implementations for all the 3,984 P-classes of Boolean functions up to 4 variables. Each entry represents a valid RSFQ compound gate. Note again that the considerable initial runtime for database creation is amortized by repeated use.

We apply our final database to synthesize a subset of EPFL [30] and ISCAS [27] benchmark circuits. We compare our results against PBMap [11], the state-of-the-art dynamic programming algorithm for path balancing. The results are shown in Table I. Compared to the state of the art, gate compounding technique drastically reduces logic depth by an average of 33%. Due to the use of more expressive compound gates, the area of the circuits (expressed as total JJ count) is reduced by an average of 24%, despite 53% larger number of path-balancing DFFs.

Despite substantial improvements in many benchmarks, our approach yields a weaker result in `dec` circuit. The increase in JJ count can be attributed to two factors. First, the logic depth of this circuit is only 4 cycles, limiting the impact of compound gates. Second, the JJ cost of each primitive in the RSFQ library used in [11] is not openly available at the reference. Likely, the CONNECT cell library [32] used in this work has a higher

TABLE I
Number of path-balancing DFFs, JJs, and logic depth in a subset of EPFL [26] and ISCAS [27] benchmarks

	#DFF			#JJ			Delay			
Benchmark	Baseline	Our work	Ratio	Baseline	Our work	Ratio	Baseline	Our work	Ratio	Runtime, s
`sin`	13,666	17,627	1.29	215,318	126,694	0.59	182	86	0.47	0.399
`cavlc`	522	987	1.89	16,339	15,098	0.92	17	11	0.65	0.009
`dec`	8	16	2.00	5,469	6,324	1.16	4	4	1.00	0.006
`int2float`	270	443	1.64	6,432	5,616	0.87	16	10	0.63	0.004
`priority`	9,064	14,754	1.63	102,085	95,370	0.93	127	125	0.98	0.013
`c499`	476	512	1.08	7,758	5,593	0.72	13	8	0.62	0.040
`c880`	774	1,179	1.52	12,909	8,359	0.65	22	13	0.59	0.013
`c1908`	696	799	1.15	12,013	5,553	0.46	20	11	0.55	0.025
`c3540`	1,159	1,556	1.34	28,300	22,231	0.79	31	18	0.58	0.034
`c5315`	2,908	3,727	1.28	52,033	33,524	0.64	23	13	0.57	0.091
`c7552`	2,429	4,744	1.95	48,482	28,900	0.60	19	13	0.68	0.115
Average			1.53			0.76			0.67	

TABLE II
COMPARISON WITH DCM [14] WITH 1/7 THROUGHPUT ON A SUBSET OF EPFL [26] AND ISCAS [27] BENCHMARKS

	DCM (1/7) [14]		Our Work			
benchmark	#DFF	#JJ	#DFF	Ratio	#JJ	Ratio
int2float	117	7,770	440	3.76	5,973	0.77
priority	8,562	257,252	14,754	1.72	68,177	0.27
voter	7,204	447,044	8,357	1.16	189,622	0.42
c432	224	10,734	1,180	5.27	6,905	0.64
c880	362	14,658	1,176	3.25	8,650	0.59
c1355	193	8,739	448	2.32	5,703	0.65
c1908	282	13,169	799	2.83	5,497	0.42
c3540	776	43,437	1,554	2.00	20,820	0.48
Average				2.79		0.53

JJ cost for logic primitives compared to [11], contributing to the area increase.

We also compare our results with the dual clock methodology [14]. A logic circuit is partitioned into separate clocking domains using the NDRO flip flops. Subcircuits within each partition are clocked at high frequency, while the NDRO flip flops operate at a frequency 7 times smaller than the high frequency. The throughput of the system is therefore reduced by a factor of 7. The results are compared in Table II. Despite 7 times smaller throughput and 64% fewer DFFs, the dual clocking method requires almost 2 times more JJs as compared to gate compounding. In addition, DCM systems require relatively expensive NDRO DFFs, pulse repeaters and an additional low-frequency clock distribution network, further degrading the area of the system.

VII. CONCLUSIONS

RSFQ technology has the potential to enhance power and speed of the mainstream computing systems by several orders of magnitude. The gate compounding technique is a novel method to reduce the logic depth by exploiting the synchronization mechanisms of RSFQ technology. With more expressive logic gates, area of the circuits is considerably reduced. In this paper, we proposed a scalable technology mapping method that leverages SFQ compound gates. We generated a database of functionally correct and area-optimal compound gates for all functions up to 4 variables. Then, we applied a delay-driven technology mapping using the pre-computed database as a cell library. In the experimental results, we showed a substantial reduction in the area and logic depth by 24% and 33%, respectively, compared to the state-of-the-art.

REFERENCES

[1] K. Likharev, O. Mukhanov, and V. Semenov, "Resistive Single Flux Quantum Logic for the Josephson-Junction Digital Technology," *Proc. International Conference on Superconducting Quantum Devices*, Vol. 85, pp. 1103–1108, June 1985.

[2] T. Kawaguchi, M. Tanaka, K. Takagi, and N. Takagi, "Demonstration of an 8-Bit SFQ Carry Look-Ahead Adder Using Clockless Logic Cells," *International Superconductive Electronics Conference*, July 2015.

[3] D. Gupta, A. A. Inamdar, D. E. Kirichenko, A. M. Kadin, and O. A. Mukhanov, "Superconductor Analog-to-Digital Converters and Their Applications," *Proc. IEEE MTT-S Int. Microwave Symp.*, 2011.

[4] S. Yang, X. Gao, R. Yang, J. Ren, and Z. Wang, "A Hybrid Josephson Transmission Line and Passive Transmission Line Routing Framework for Single Flux Quantum Logic," *IEEE TASC*, Vol. 32, No. 9, 2022.

[5] W. Chen, A. Rylyakov, V. Patel, J. Lukens, and K. Likharev, "Rapid Single Flux Quantum T-Flip Flop Operating up to 770 GHz," *IEEE TASC*, Vol. 9, No. 2, pp. 3212–3215, 1999.

[6] Q. P. Herr, A. D. Smith, and M. S. Wire, "High Speed Data Link between Digital Superconductor Chips," *Appl. Phys. Lett.*, Vol. 80, No. 17, pp. 3210–3212, 2002.

[7] H. Akaike, T. Yamada, A. Fujimaki, S. Nagasawa, K. Hinode, T. Satoh, Y. Kitagawa, and M. Hidaka, "Demonstration of a 120 GHz Single-Flux-Quantum Shift Register Circuit Based on a 10 kA cm^{-2} Nb Process," *Superconductor Science and Technology*, Vol. 19, No. 5, pp. S320, 2006.

[8] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-Efficient Superconducting computing—Power Budgets and Requirements," *IEEE TASC*, Vol. 23, No. 3, pp. 1701610–1701610, 2013.

[9] P. Bunyk, K. Likharev, and D. Zinoviev, "RSFQ Technology: Physics and Devices," *International Journal of High Speed Electronics and Systems*, Vol. 11, No. 01, pp. 257–305, 2001.

[10] G. Krylov and E. G. Friedman, *Single Flux Quantum Integrated Circuit Design*, Springer, 2022.

[11] G. Pasandi and M. Pedram, "PBMap: A Path Balancing Technology Mapping Algorithm for Single Flux Quantum Logic Circuits," *IEEE TASC*, Vol. 29, No. 4, June 2019.

[12] N. Kito, K. Takagi, and N. Takagi, "Logic-Depth-Aware Technology Mapping Method for RSFQ Logic Circuits With Special RSFQ Gates," *IEEE TASC*, Vol. 32, No. 4, 2021.

[13] R. Bairamkulov, T. Jabbari, and E. G. Friedman, "QuCTS — Single-Flux Quantum Clock Tree Synthesis," *IEEE TCAD*, Vol. 41, No. 10, pp. 3346–3358, 2022.

[14] G. Pasandi and M. Pedram, "Depth-Bounded Graph Partitioning Algorithm and Dual Clocking Method for Realization of Superconducting SFQ Circuits," *ACM JETCAS*, Vol. 17, No. 1, October 2020.

[15] J.-H. Yang *et al.*, "Distributed Self-Clock: A Suitable Architecture for SFQ Circuits," *IEEE TASC*, Vol. 30, No. 7, 2020.

[16] X. Li *et al.*, "Optimization of Delay Time Stabilization for Single Flux Quantum Cell Library," *IEEE TASC*, Vol. 30, No. 7, 2020.

[17] S. V. Rylov, "Clockless Dynamic SFQ and Gate with High Input Skew Tolerance," *IEEE TASC*, Vol. 29, No. 5, 2019.

[18] T. Kawaguchi, K. Takagi, and N. Takagi, "Static Timing Analysis for Single-Flux-Quantum Circuits Composed of Various Gates," *IEEE TASC*, Vol. 32, No. 5, 2022.

[19] R. Bairamkulov and G. De Micheli, "Compound Logic Gates for Pipeline Depth Minimization in Single Flux Quantum Integrated Systems," *Proc. GLSVLSI*, June 2023.

[20] D. S. Marakkalage, H. Riener, and G. De Micheli, "Optimizing Adiabatic Quantum-Flux-Parametron (AQFP) Circuits Using an Exact Database," *Proc. NANOARCH*, November 2021.

[21] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P.-E. Gaillardon, J. Olson, R. Brayton, and G. De Micheli, "Enabling Exact Delay Synthesis," *Proc. ICCAD*, pp. 352–359, 2017.

[22] A. Tempia Calvino, H. Riener, S. Rai, A. Kumar, and G. De Micheli, "A Versatile Mapping Approach for Technology Mapping and Graph Optimization," *Proc. ASP-DAC*, pp. 410–416, 2022.

[23] O. Mukhanov, V. Semenov, and K. Likharev, "Ultimate Performance of the RSFQ Logic Circuits," *IEEE Trans. Magn.*, Vol. 23, No. 2, pp. 759–762, 1987.

[24] M. Soeken, W. Haaswijk, E. Testa, A. Mishchenko, L. G. Amarù, R. K. Brayton, and G. De Micheli, "Practical Exact Synthesis," *Proc. DATE*, pp. 309–314, 2018.

[25] H.-T. Zhang, J.-H. R. Jiang, L. Amarú, A. Mishchenko, and R. Brayton, "Deep Integration of Circuit Simulator and SAT Solver," *Proc. DAC*, pp. 877–882, 2021.

[26] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," *Proc. IWLS*, 2015.

[27] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Des. Test. Comput.*, Vol. 16, No. 3, pp. 72–80, 1999.

[28] A. Tempia Calvino and G. De Micheli, "Depth-Optimal Buffer and Splitter Insertion and Optimization in AQFP Circuits," *Proc. ASP-DAC*, p. 152–158, 2023.

[29] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, Vol. 6, No. 1–6, pp. 5–35, June 1991.

[30] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, D. S. Marakkalage, and G. De Micheli, "The EPFL Logic Synthesis Libraries," *arXiv:1805.05121v3*, 2018.

[31] L. Benini and G. De Micheli, "A Survey of Boolean Matching Techniques for Library Binding," *ACM Trans. Design Autom. Electr. Syst.*, July 1997.

[32] S. Yorozu *et al.*, "A Single Flux Quantum Standard Logic Cell Library," *Physica C: Superconductivity*, Vol. 378-381, pp. 1471–1474, 2002.