A Variation-aware Hold Time Fixing Methodology for Single Flux Quantum Logic Circuits

XI LI, SOHEIL NAZAR SHAHSAVANI, XUAN ZHOU, MASSOUD PEDRAM, and PETER A. BEEREL, University of Southern California

Single flux quantum (SFQ) logic is a promising technology to replace complementary metal-oxide-semiconductor logic for future exa-scale supercomputing but requires the development of reliable EDA tools that are tailored to the unique characteristics of SFQ circuits, including the need for active splitters to support fanout and clocked logic gates. This article is the first work to present a physical design methodology for inserting hold buffers in SFQ circuits. Our approach is variation-aware, uses *common path pessimism removal* and incremental placement to minimize the overhead of timing fixes, and can trade off layout area and timing yield. Compared to a previously proposed approach using fixed hold time margins, Monte Carlo simulations show that, averaging across 10 ISCAS'85 benchmark circuits, our proposed method can reduce the number of inserted hold buffers by 8.4% with a 6.2% improvement in timing yield and by 21.9% with a 1.7% improvement in timing yield.

CCS Concepts: • **Hardware → Emerging technologies;**

Additional Key Words and Phrases: Single-flux quantum (SFQ), common path pessimism removal (CPPR), clock tree synthesis, legalization, placement, timing uncertainty, hold time, Gaussian distribution, Monte Carlo simulation

ACM Reference format:
Xi Li, Soheil Nazar Shahsavani, Xuan Zhou, Massoud Pedram, and Peter A. Beerel. 2021. A Variation-aware Hold Time Fixing Methodology for Single Flux Quantum Logic Circuits. *ACM Trans. Des. Autom. Electron. Syst.* 26, 6, Article 47 (June 2021), 17 pages.
https://doi.org/10.1145/3460289

1 INTRODUCTION

Superconductive electronics, and **single-flux quantum (SFQ)** [18] in particular, is a promising replacement for **complementary metal-oxide-semiconductor (CMOS)** technology for exascale supercomputing. With the increasing need for big data and supercomputing, the hundreds of megawatts of power needed by current exa-scale computing platforms is a growing concern [23]. Rapid SFQ technology was introduced back in the late 1908s [18], with a theoretical potential of

Xi Li and Soheil Nazar Shahsavani contributed equally to this research.

This work was supported by the Office of the Director of National Intelligence, Intelligence Advanced Research Projects Activity, via the U.S. Army Research Office Grant W911NF-17-1-0120.

Authors' addresses: X. Li, S. N. Shahsavani, X. Zhou, M. Pedram, and P. A. Beerel, University of Southern California, Ming Hsieh Department of Electrical and Computer Engineering, Los Angeles, CA, 90089-2560; emails: {xli497, nazarsha, zhouxuan, pedram, pabeerel}@usc.edu.

1084-4309/2021/06-ART47 $15.00
https://doi.org/10.1145/3460289

meeting high-performance needs with three orders of magnitude lower power compared to state-of-the-art semiconductor technologies [20].

Nevertheless, the potential of SFQ is yet to be achieved for complex designs such as microprocessors for a variety of reasons. Most notably, the lack of an established reliable **computer-aided design (CAD)** tool flow has been a long-term obstacle for SFQ technology to scale [5, 11]. More specifically, SFQ circuits often have ultra high clock frequencies and significant time uncertainty [3, 31], which makes the design of **clock distribution networks (CDN)** and timing closure extremely challenging.

In particular, managing clock skew in high-frequency SFQ CDNs is extremely important [10], because, in addition to the storage gates, all logic gates are clocked and all fanouts are implemented with splitters, which increases the size and insertion delay of the clock network significantly. Moreover, due to variations in the fabrication process, and biasing or temperature variations, timing uncertainties in the clock trees are significant. Therefore, developing methodologies for generation of high-quality clock trees and robust timing uncertainty-aware timing closure algorithms is paramount.

A recent work [25] proposed a minimum-skew CDN topology generation algorithm using a fully-balanced tree structure that accounts for timing criticalities in the datapath and the total wirelength of the clock tree, and minimizes the total negative slack in the presence of timing uncertainties. Even when deploying such CDN algorithms, EDA flows need to utilize efficient techniques to *close timing*, i.e., fix setup and hold timing issues. In particular, potential hold-time violations are typically mitigated by adding clockless buffers into problematic short paths in the data path. The high clock frequency and gate-level pipelining of SFQ circuits makes this task particularly challenging. Moreover, as in CMOS, these buffers should account for expected timing variations and should be validated either through someform of static timing analysis [35] or classic Monte Carlo simulation [28].

To the best of our knowledge, the proposed method is the first work to propose a physical design methodology for hold buffer insertion in SFQ circuits. Our approach is variation-aware and reduces area and performance overheads by applying **common-path pessimism removal (CPPR)** to remove the pessimism associated with the common clock paths to pairs of sequentially adjacent gates [8, 34]. Furthermore, we employ an incremental placement methodology to place the added buffers and minimize the perturbation to the original placement solution. The efficacy of the proposed method is evaluated using ISCAS'85 benchmark circuits [2]. Compared with utilizing fixed constant margins for all timing paths, as suggested in Reference [28], the proposed method significantly reduces the number of inserted hold buffers with competitive timing yield. The key contributions of this article can be summarized as follows:

(1) The development of the first variation-aware hold time fixing approach for SFQ circuits. The approach considers both local and global timing uncertainties and effectively uses common path pessimism removal to reduce the number of inserted hold buffers on each timing path.
(2) The placement of hold buffers is implemented by the qPlace tool [24] that is enhanced with an incremental placement strategy that generates high-quality solutions.
(3) The approach is evaluated using dynamic timing analysis with a grid-based placement-aware variation model [28] on multiple ISCAS'85 benchmark circuits. The functionalities of circuits are verified via Monte Carlo co-simulation with behavioral netlists. Comparing our variation-aware approach to a fixed-margin baseline, the average number of hold buffers can be reduced by 8.4% with 6.2% increase in timing yield and 21.9% with a 1.7% increase in timing yield. It allows a tradeoff between timing yield and layout area by tuning algorithmic parameters.

Compared with related prior work in CMOS, there are two key differences in our approach. First, state-of-the-art CMOS CPPR algorithms use complex pre-processing steps to transform the problem of finding the **least-common-ancestor (LCA)** of sequentially adjacent gates into a **range minimum query (RMQ)** problem [8, 13, 14]. In SFQ circuits, however, the clock tree must be implemented with splitter circuitry that is often limited to binary forks. Moreover, since every logic gate is clocked, the clock tree is usually far deeper than CMOS and is typically balanced to minimize clock skew. Its balanced nature enables our LCA detection algorithm to be significantly simpler and more efficient to solve. Second, the combinational logic between sequentially adjacent gates is limited to splitters that route datapath signals to multiple clocked sinks. It is sufficient to insert hold buffers at the input port of any clocked gate with a hold violation. While in CMOS, due to the exponential number of timing paths and interaction between paths [27], this approach may lead to setup violations on other paths and thus more complex insertion algorithms are required [12, 16, 22, 32].

The rest of this article is organized as follows. Background and prior work are summarized in Section 2. Section 3 presents the proposed approach for hold time fixing and incremental placement. Section 4 details the experiments flow including the grid-based variation model and dynamic simulation methodology. Simulation results are presented in Section 5 followed by conclusions in Section 6.

2 BACKGROUND

2.1 Definitions and Notation

In this section, we introduce some definitions and notations used throughout the article.

2.1.1 Combinational and Sequential Cells in SFQ Logic. In CMOS logic, combinational logic refers to digital circuits that determine their outputs as Boolean functions of their inputs. However, in SFQ circuit, gates such as AND, OR, XOR, and NOT are clocked [6]. They process pulses on their inputs by changing states of their internal current loops and produce their output pulses in response to an input clock pulse and thus are also sequential.

Moreover, unlike CMOS, splitters are needed to actively copy input pulses to multiple outputs. In particular, splitters are needed to distribute the clock and implement the fanout of logic gates. In addition, buffers that simply copy their single input to their single output are also sometimes needed. Both of these gates need not be clocked and thus are considered as combinational SFQ gates.

2.1.2 Data Path Delay. In conjunction with interconnect delay, combinational gates make up the datapath delay between sequential SFQ elements. For example, Figure 1 illustrates two sequential SFQ elements G_1 and G_2, while $S_1 - S_5$ are clock splitters that distribute the clock signal from a clock source to each sequential element in the circuit. The data path delay of the path between G_1 and G_2 refers to the summation of clock-to-Q delay of G_1, the delay of the splitters, and interconnect delays between the gates.

2.1.3 Insertion Delay. The insertion delay refers to the total delay from the clock source to clock input of a sequential element [15]. This is also known as the arrival time of the clock signal at the clock pin of each sequential gate. For G_1 in Figure 1, it refers to the delay from clock source (Clk Src) through splitters S_1, S_3, and S_4 as well as the delay of interconnects connecting the clock source to G_1.

2.1.4 Clock Skew. Two sequential elements connected by data path splitters and interconnect are called sequentially adjacent gates. In Figure 1, G_1 and G_2 are called launching and capturing

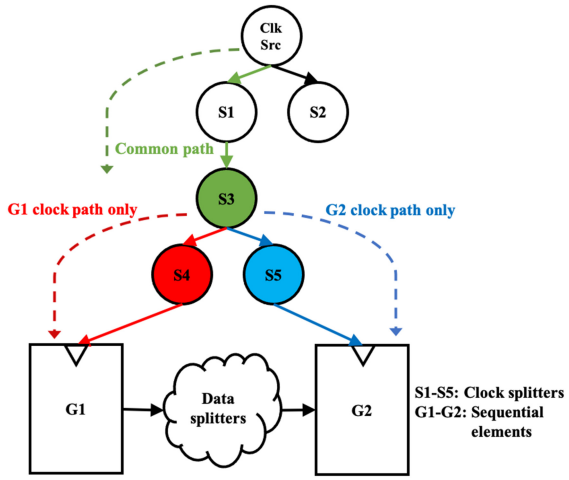

Fig. 1. Example of a timing path in a SFQ circuit.

flops, respectively. The difference between the clock arrival time at the launching and capturing flip-flops is defined as the clock skew between the pair of gates. For the pair of flip-flops illustrated in Figure 1, the clock skew is the difference between the delay from Clk Src through splitters S_1, S_3, and S_4 as well as the delay of interconnects connecting the clock source to G_1, on one hand, and the delay from clock source through splitters S_1, S_3, and S_5 as well as the delay of interconnects connecting the clock source to G_2, on the other hand.

2.1.5 Hold Time. Each sequential element requires some time to reliably capture input data at its data pin. Hold time is defined as the amount of time after the arrival of the clock pulse during which input data must remain stable and it imposes a timing constraint on every pair of sequentially adjacent gates G_1 and G_2. Let $T_{skew}(G_1, G_2)$ denote the clock skew and $T_{DP}^{min}(G_1, G_2)$ denote the minimum data path delay between G_1 and G_2. Let T_{hold}^{max} represent the maximum hold time required for G_2. Then, hold slack is defined as the difference between when the data can change and how long it should remain stable, i.e.,

$$Slack_{hold}(G_1, G_2) = T_{skew}(G_1, G_2) + T_{DP}^{min}(G_1, G_2) - T_{hold}^{max}. \tag{1}$$

A negative hold slack leads to a hold time violation. As shown in Equation (1), hold slack is not a function of the clock cycle time. Therefore, it cannot be fixed by adjusting clock frequency and even a single hold time violation may lead to circuit malfunction.

Unlike CMOS technology, where multiple hold time refinement techniques such as gate down-sizing, switching between low-threshold and high-threshold gates, and wire sizing are commonly used, such options are not available in the current SFQ technology and existing cell libraries. Hence, hold buffer insertion is one of the most applicable approaches for SFQ circuits.

2.1.6 Hold Time Margin. Hold time margin (i.e., safety margin) is the extra delay added to data paths to avoid hold violations accounting for variations in the timing parameters in Equation (1). Providing this "cushion" to the design can help make it more robust to timing variations and possible hold time violations. Hold margin is satisfied when

$$Slack_{hold}(G_1, G_2) \geq T_{hold}^{margin}(G_1, G_2). \tag{2}$$

2.1.7 Clock Tree Topology. In this article, we define the **clock tree topology (CTT)** as a directed binary tree T connecting clock source to all the clock sinks. The root of T denotes the clock splitter S_0 connected to the clock source while the nodes $\{S_{n+1}, S_{n+2}, \ldots, S_{n+k}\}$ represent the leaf nodes (clock sinks), i.e., sequential elements in a circuit. A clock topology generation algorithm creates nodes $\{S_1, \ldots, S_n\}$ representing the clock splitters, adds them to the clock tree, and connects the clock source to all the sink nodes. We assume the leaf nodes are at level 0, while the root is at the highest level of the tree. In Figure 1, the CTT refers to the binary tree including Clk Src, S_1 through S_5, and the edges between clock splitters S_4, S_5, and sink nodes G_1, G_2, respectively.

2.1.8 Common Clock Path. A **clock path (CP)** is a path in T from root node through tree nodes to each leaf node (clock sink). For two leaf nodes in T, the **common clock path (CCP)** refers to the portion of the clock tree that is common between the CPs to each gate, while the **non-common clock path (NCP)** represents the non-common portion of the CPs. In Figure 1, the common clock path includes nodes S_1 and S_3 and the edges connecting Clk Src to S_3.

2.2 Timing Uncertainty-aware Clock Topology Generation

The topology of a clock network is one of the key factors that determine the yield of a logic circuit and its robustness against variations in timing. One of the key goals in achieving high yield is to maximize the CCP to pairs of flip-flops that are on timing critical paths; that is to ensure the variations in the arrival time of the clock signal at launching and capturing flops does not lead to negative slacks and timing violations. Consequently, when generating a CTT, accounting for the criticality of data paths and timing yield are important factors. Not surprisingly, the importance of concurrent clock-and-data optimization is also emphasized in commercial CMOS tools [4].

The authors of References [25] present a low-cost, timing uncertainty-aware balanced CTT generation algorithm for SFQ circuits. This algorithm considers the criticality of the data paths in terms of timing slacks as well as the total wirelength of the clock tree and generates a (height-) balanced binary clock tree using a bottom-up approach and an **integer linear programming (ILP)** formulation. The results show significant improvement in terms of timing metrics (i.e., total and worst negative slack) over the baseline algorithm that is essentially a wirelength driven approach, i.e., **method of means and medians (MMM)** [17, 25].

In this work, we utilize this method to generate high-quality balanced clock topologies that improve the timing yield and minimize the need for hold-fixing buffers and associated area overhead. With current gate counts in the SFQ technology, we believe this solution is practical. However, as the scale of SFQ circuits grows, alternative optimization strategies may be necessary.

Table 1 lists the results of applying the proposed ILP-based topology generation algorithm for several benchmarks from the ISCAS'85 benchmark suite [2] and compares the results with that of the MMM algorithm in terms of the number of required hold buffers after placement and clock tree synthesis. To account for worst-case conditions, it is assumed that due to timing variations, the delay of all the gates on the launch (capture) CP decrease (increase) by $(3 * \sigma)$, where σ denotes the standard deviation of gate delay values. It is also assumed that the delay of all gates on each datapath is minimized. and the delay of hold buffers is set to be 5.5 ps. As shown in Table 1, using the proposed approach in Reference [25], the total number of hold buffers required to fix the hold time violations in various benchmarks is reduced by an average of 6% over 7 benchmarks, by as much as 14.8%, showing the importance of minimizing worst-case negative slack. Note, however, that the work in Reference [25] did not consider the physical placement of the hold buffers nor the potential benefit of CPPR.

Table 1. Number of Hold Buffers with Greedy Topology Generation (MMM) vs. ILP Topology Generation

	3σ		
Design	**# buffers MMM [17]**	**# buffers ILP [25]**	**Saving (%)**
c432	1188	1012	14.8
c499	447	426	4.7
c880	828	757	8.6
c1355	421	431	−2.4
c1908	805	739	8.2
c3540	1490	1416	5.0
c5315	3249	3055	6.0
AVG.	**1204**	**1119**	**6**

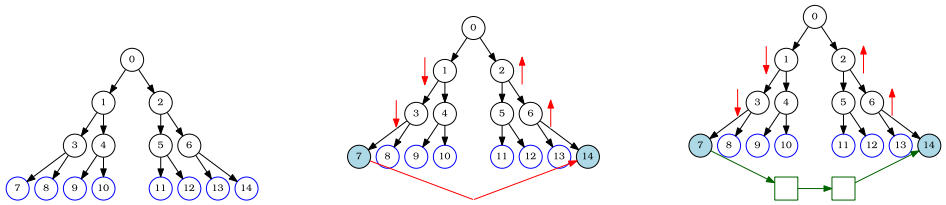

Fig. 2. A balanced binary tree with 8 sinks. (a) At nominal condition, the max clock skew is zero. (b) If all the splitter delays on path $S_0 \rightarrow S_7$ decrease to the min propagation delay and all the splitter delays on the $S_0 \rightarrow S_{14}$ increase to the max propagation delay, then a hold time violation may occur on data path $S_7 \rightarrow S_{14}$. (c) Two buffers are added to the path between $S_7 \rightarrow S_{14}$ to increase its hold slack.

3 THE PROPOSED METHOD

3.1 Overview

In this section, we formally define the problem, provide a motivating example, and illustrate the proposed methodology in detail.

3.2 Motivating Example

Consider the clock tree topology depicted in Figure 2 where a clock signal is propagated to eight sink nodes using a balanced binary tree. Assume, the propagation delay of internal nodes (i.e., splitters) as well as hold buffers to be 5 ps with a ±20% variation on the delay due to timing variations. Consider a data path connecting node 7 to node 14. Considering the worst-case scenario in terms of hold slack, due to timing variations the delay of all clock splitter nodes on the launch CP decrease to 4 ps and all the splitter delays on the capture CP increase to 6 ps. As a result, the hold slack may be reduced by 6×1 ps $= 6$ ps. However, such an estimation is overly pessimistic as node 0 is on the common CP to nodes 7 and 14. Therefore, the worst-case incurred hold slack cannot be less than −4 ps. Consequently, accounting for CPPR during the timing closure can reduce the number of required hold buffers, connecting node 7 to 14, from 3 to 2, assuming each hold buffer adds a delay of 2 ps. This reduces the area overhead by 33% while achieving the same timing yield.

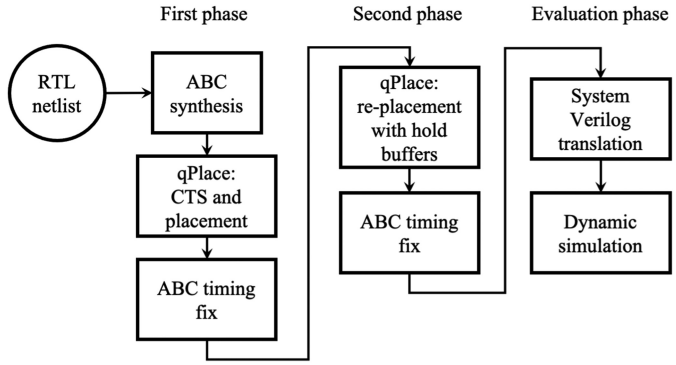

Fig. 3. Overview flow of the proposed algorithm.

3.3 Problem Formulation

The input to this problem is a netlist graph (G) comprising of the logic cells (i.e., nodes), their connections (i.e., edges), the location of each node on the layout area, descriptions of the clock network, including the clock tree topology (T) and the location of the splitter nodes ($S_0 \ldots S_n$), and a variability model that defines the variations of gate delays under the presence of process, biasing, or temperature variations. The objective function is to minimize the total number of required hold buffers such that timing of the circuit with respect to hold constraints is satisfied or a target timing yield is reached. Therefore, the total negative hold slack in the presence of timing variations should be less than a predefined threshold value. In other words, we aim to satisfy the constraints in terms of worst negative hold slack while minimizing the number of inserted hold buffers, i.e., the area overhead of timing closure.

3.4 Proposed Methodology—An Overview

The overall flow of our proposed method is depicted in Figure 3. The first phase follows the algorithms outlined in Reference [24] and Reference [28]. The Berkeley open source logic synthesis tool, ABC [26], is used to synthesis the netlist. The qPlace tool, as part of the qPALACE CAD tool suite for SFQ circuit [24, 25], is employed to place the logic gates, construct a minimum-skew clock network, and place the clock splitters [24, 25] to minimize the nominal clock skew, reducing the total negative slack associated with all sequentially adjacent gates. ABC then parses and translates the generated netlist and clock tree information, extracts delays associated with the logic gates, splitters, and interconnect using a linear delay model [25], and performs timing analysis for all setup and hold constraints. Using our proposed variation-aware hold fixing algorithm, ABC then inserts hold buffers, i.e., **Josephson transmission line (JTL)** cells, between sequentially adjacent pairs in the netlist as needed.

As shown in Figure 3, we then enter the second phase of the flow in which the updated circuit, including newly inserted JTLs, is physically placed by the qPlace tool, either using our proposed incremental placement technique or by re-placing the entire circuit. The IBM CPLEX v12.10 package [1] is used for solving **mixed integer linear programming (MILP)** problems for clock tree placement and legalization. Since this step may modify the location of logic gates, the clock tree is re-synthesized to minimize the nominal clock skew and generate a high-quality clock network, albeit we force it to use the same clock tree topology as in phase one. This is motivated by the variation-aware hold fixing algorithm that takes advantage of the topology of the clock tree and common clock paths to minimize the area overhead of timing fixes. Although clock skews and

hold slacks in the nominal condition may be modified, the timing variations will not lead to additional timing violations, as the worst-case variations as a function of the clock topology remains constant.

In addition, to both mitigate routing congestion caused by the relative few number of routing layers and to facilitate a final pass of hold buffer insertion, during this second placement phase we reserve empty space next to each logic cell and existing hold buffers. After this second placement, we parse the re-placed circuit into ABC and run a final round of timing analysis. If any additional hold buffers are required, then they are inserted into the reserved spaces without the need to move other gates. We note that our ABC timing fix algorithm and System Verilog translation with interconnect delay model has been added to the existing qPALACE tool suite [24, 25]. In particular, the qPlace placement tool [24] has been extended to include an option for our incremental placement algorithm.

The next subsections detail our novel hold time fixing and incremental placement techniques. The description of the third (evaluation) phase of the flow is presented in Section 4.

3.5 Variation Aware Common Path Pessimism Removal

CPPR is the removal of unnecessary pessimism of timing analysis on launching clock path and capturing clock path by accounting for the common portion of the launch and capture CPs [8, 34].

Particularly, the timing uncertainty of the propagation delay of these elements in the CCP affects the launching and capturing clock path delays similarly. Thus we propose to remove this unnecessary timing uncertainty from our timing analysis to more accurately capture the worst-case timing scenario.

To account for the remaining timing variations, we add hold margins to each timing path and thereby make the design robust to hold time violations. Assume G_1 and G_2 are the launching and the capturing logic gates, respectively. The hold slack from G_1 to G_2 is formulated with timing constraints as Equation (2). Instead of applying a constant hold margin to all hold slacks as proposed in Reference [28], we propose a variation-aware strategy to determine the hold margin required for each timing path and apply CPPR to eliminate unnecessary pessimism.

For all gates in the circuit, we assume the gate delay follows a Gaussian distribution with same standard variation value σ. In the worst-case scenario, a hold short path can be formulated as follows: the delay of clock splitters in the launching clock path bias to a low value, while the delay of clock splitters in the capturing clock path bias to a high value. At the same time, the delay of gates in data path bias to a low value. We consider the 3σ rule [9], which states that 99.7% of the possible data are within three standard variation deviations from the mean and assume a worst-case delay change of 3σ for each gate.

Consider the worst-case scenario in Equation (2), the variation-aware hold margin can be set as the the maximum reduction of $Slack_{hold}(G_1, G_2)$ caused by gate delay variations,

$$T_{hold}^{margin}(G_1, G_2) = 3\sigma * (D_{NCP}(G_1, G_2) + D_{DP}(G1, G2)), \tag{3}$$

where $D_{NCP}(G_1, G_2)$ represents the sum of the delay of the clock splitters that are not common to LP and CP, while $D_{DP}(G1, G2)$ refer to the sum of delay of the gates along the data path from G_1 to G_2.

$D_{NCP}(G_1, G_2)$ is determined by finding the lowest common ancestor in the balanced binary clock tree. We record the routes from the clock source to clock sinks when building the clock tree. For any two sequentially adjacent gates, the algorithm traverses both launching and capturing clock

paths level by level, starting at the leaves (level 0), until reaching a common node, the LCA.[1] The number of splitters in the NCP is simply the number of nodes met during the traversal, which is equal to twice the level of the LCA minus 1. Thus, we can compute $D_{NCP}(G_1, G_2)$ in Equation (3) as:

$$D_{NCP}(G_1, G_2) = 2 * (level(LCA(G1, G2)) - 1) * D_{sp}, \tag{4}$$

where D_{sp} denotes the nominal splitter delay.

For each timing path, the hold time constraint is checked and in case of a negative slack, hold buffers are added before the corresponding input pin of G_2 until the constraint is met. Consider as an example the circuit in Figure 1 and the hold time for the path G_1 to G_2. The clock path of G_1 consists of the following clock segments: the clock source, splitters S_1, S_3, S_4. Similarly, the clock path of G_2 comprises of clock segments: clock source, splitters S_1, S_3, S_5. The CCP contains S_1 and S_3, while NCP contains S_4 and S_5. $level(LCA(G1, G2))$ is 2 in this case. The hold time margin for G_2 is thus

$$T_{hold}^{margin}(G_1, G_2) = 3\sigma * (2 * D_{sp} + D_{G_1} + N_{data\_sp} * D_{sp}), \tag{5}$$

where D_{G_1} denotes the nominal clock-to-Q delay of G_1, and N_{data_sp} denotes the number of splitters in data path from G_1 to G_2. $T_{hold}^{margin}(G_1, G_2)$ can be used to guide hold buffer insertion for the timing path from G_1 to G_2.

Assuming there are N clock sinks in the whole circuit, the height H of clock tree, a fully-balanced binary tree, is $\lfloor log_2 N \rfloor + 1$. For CCP, the time complexity to find LCA is $O(H)$. We denote the total number of pairs of sequentially adjacent gates as E. Consequently, the time complexity of finding LCA of all sequentially adjacent gates is $O(E * \log_2 N)$. The fact that the clock tree is a fully-balanced binary tree simplifies the LCA detection algorithm. In particular, for traditional CMOS circuits, state-of-the-art LCA detection algorithms typically require a reduction to an instance of RMQ problem via an Euler walk of the clock tree and the storage of several extra tables [13, 14, 19].

3.6 Placement Methodology

After adding the hold buffers to the netlist, the location of logic cells and inserted hold buffers is determined. Note that since the timing closure algorithm is based on the original clock topology and inserted hold buffers are not sequential elements, after placement and legalization of the hold buffers, the same clock topology can be utilized. However, to minimize timing metrics such as maximum clock skew and the negative timing slacks, the location of gates are properly adjusted. Similarly to Reference [24], we adopt the *deferred merge embedding* algorithm for calculating the location of the tapping points of the clock network on the layout area and employ an integer linear programming algorithm to map the clock splitters to routing channels and remove the overlaps among the clock splitters and logic gates, such that the maximum clock skew is minimized. Finally, once a legal high-quality solution in terms of placement and timing metrics is generated, another iteration of timing fixes adjusts the timing slacks.

Accordingly, the location of the placed logic and clock splitters are then modified to accommodate the placement of the hold buffers. The total number of inserted hold buffers is a function of the degree of timing uncertainties and can be a large fraction of the size of the original netlist. Therefore, to optimize the quality of results in terms of the total wirelength and timing metrics, we propose two placement strategies: (i) Incremental Placement and (ii) Placement from Scratch.

3.6.1 Incremental Placement. The incremental placement methodology helps preserving the original placement solution by constraining the placement of logic cells and data splitters to the

[1]Note that the common node must be at the same level in both the launch and capture paths because the clock tree is balanced.

same rows as they are initially placed. When the number of inserted hold buffers is relatively small, an incremental placement approach helps minimizing the displacement of original netlist elements, eliminates the need for extensive modifications to the clock network, and facilitates converging to a high-quality solution without the need for multiple iterations of placement and clock synthesis. Alternatively, when the number of inserted hold buffers is large, executing the placement algorithm from scratch may yield better overall placement metrics such as the total wirelength, layout area, and routing congestion. As the number of required hold buffers depends on the structure of circuit and the degree of variations, this article considers both.

In our incremental placement flow, the logic cells and data splitters remain in their initially placed logic row, i.e., their y coordinates are preserved. However, their x coordinates are adjusted to accommodate the placement of hold buffers.

The pseudo code for the incremental placement flow is shown in Algorithm 1. Lines 1–23 show the assignment of the hold buffers to logic rows. First, the ideal number of gates per row is determined by computing the total number of gates (including hold buffers, logic cells, data splitters and clock splitters) and dividing this number by the number of rows (cf. line 1 in Algorithm 1). The algorithm tries to assign hold buffers to logic rows to minimize the final width of the layout area thereby the total layout area after hold buffer insertion, using the number of cells assigned per row as a proxy for row width. This is achieved by setting a threshold value for the number of gates per row and trying to distribute the hold buffers among rows such that the final distribution of logic gates per row is below this threshold for all rows. Initially, this threshold value is set to the ideal number of gates per row and each row is marked as full or partially filled by comparing the existing gate count with this threshold (cf. line 7). We then assign each hold buffer to the nearest partially filled row to its fanout gate. In particular, if the row of the fanout gate is partially filled, we assign the hold buffer to the same row with its x coordinate shifted from its fanout gate by the width of one hold buffer (cf. lines 8–10). Otherwise, the hold buffer is assigned to the nearest partially filled row with x coordinate set to the same value as the fanout gate (cf. lines 12–15). There may be scenarios in which all close-by rows for a hold buffer are full which can increase the wire delay more than expected and force an increase in the clock cycle time. To avoid degrading the clock frequency when this situation is encountered, we instead increase the threshold by one buffer and accept a small increase in area (cf. lines 16–20). Once the assignment of the hold buffers to the logic rows are determined, each logic row is legalized to remove all cell overlaps using the algorithm presented in Reference [25] (cf. line 24 in Algorithm 1). In lines 25 and 26, the algorithm synthesizes and places the clock tree [24] and produces the final legal placement.

Because the logic cells are mapped to their original logic rows, the location of the sink nodes of the clock network are minimally modified. Therefore, the location of the tapping points of the clock network, i.e., the placement of the clock splitters, will be similar to those of the original clock network.

In summary, the incremental placement algorithm tries to fulfill three objectives: (i) preserve the layout area by distributing the hold buffers among partially filled rows, (ii) minimize the perturbation to the original placement solution to facilitate the clock tree synthesis and control the clock skew, and (iii) minimize the adverse effect of additional cell and interconnect delays, due to the insertion of hold buffers, on the maximum clock frequency of the circuit.

3.6.2 Placement from Scratch. For some circuits and variation settings, the number of inserted hold buffers per row can be comparable to the initial number of gates. Thus, the incremental placement algorithm is forced to make substantial modifications to the original placement solution resulting in a significant increase in the total wirelength. For these cases, it is beneficial to re-place the updated netlist without any row restrictions.

ALGORITHM 1: Incremental Placement

INPUT: A netlist, logic cell placement, a CTT, a list of inserted hold buffers
OUTPUT: A placed netlist with clock tree and hold buffers

```
1: Th = ⌈#nodes/#rows⌉                          ▷ the threshold (max) # of gates per row
2: rowMap[rowId] = no. gates in row rowId
3: for each path from u to v do
4:     currRowId = getRow(v)
5:     for each hold buffer j do
6:         currGateNum = rowMap[currRowId]
7:         filled = compare(Th, currGateNum)
8:         if filled is false then
9:             assign j to same row as v
10:            rowMap [currRowId] += 1
11:        else
12:            closestRowId ← closest unfilled row
13:            if clock frequency meets then
14:                assign j to closest unfilled row
15:                rowMap [closestRowId] += 1
16:            else                                  ▷ Increase threshold by 1
17:                Th += 1
18:                assign j to same row as v
19:                rowMap [currRowId] += 1
20:            end if
21:        end if
22:    end for
23: end for                                      ▷ complete row assignment of hold buffers
24: legalize and remove overlaps of hold buffers
25: add clock splitters from original CTT to netlist
26: place clock splitters, legalize, and remove overlaps
```

Once the placement and clock tree synthesis are completed, the timing closure flow adds a final round of hold buffer insertion to ensure timing requirements under nominal conditions are satisfied. Experimental results show that the total number of required hold buffers in this phase is much smaller than the size of the netlist and can be placed in the reserved empty spaces next to existing logic cells without requiring multiple iterations.

4 EVALUATION FLOW

We evaluated the timing yield of our final circuits by translating them to System Verilog netlists using Python scripts, similarly to Reference [29], and running dynamic **Monte Carlo (MC)** co-simulations in Cadence NCsim, checking all setup and hold constraints. This section first presents the variation model used and then details of the co-simulation Monte Carlo flow.

4.1 Variation Model

In this subsection, we summarize the variation model utilized to generate the random gate delays used in the MC simulations for evaluating timing yield [28].

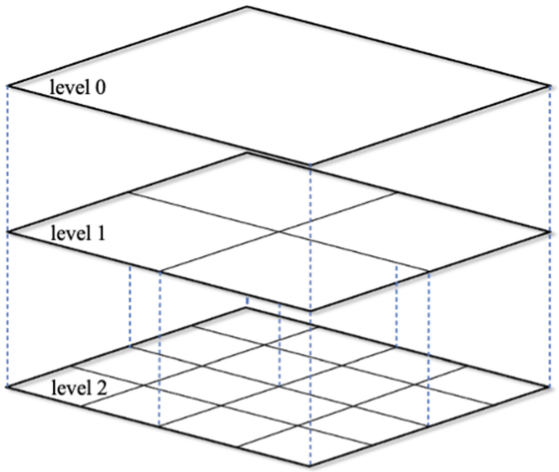

Fig. 4. Grid-based variation model with uniform grids.

To account for the spatial correlation of timing uncertainties, we employ a placement-aware variation model, utilizing a grid-based model similarly to References [28, 33] illustrated in Figure 4. In this model, the entire layout area is considered to be inside one grid bin (level 0) where we assume global variations affect all the gate delays. Subsequently, to account for the local variations on the chip, we divide the layout area level by level, each bin is subdivided into smaller bins. At each level, it is assumed that all process parameters and variations within the same bin have the same characteristics. The differences of bins are determined by the local variation of the level and different hierarchy levels are assumed to be statistically independent [28]. By adding up the time variations induced by all levels, the delay factor of each gate can be defined. The delay of all gates produce a dataset that follows a Gaussian distribution. We set the targeted standard variation of this Gaussian distribution dataset under process variations based on the process control monitor data for 350-nm fabrication process SFQ5ee developed by MIT Lincoln Laboratory [30]. Then the parameters for global and local variations can be set according to the targeted standard variation [7, 21, 28].

4.2 Dynamic Simulation

To run MC simulations, we define gate delay scaling factors for each gate in the circuit and write them as System Verilog macros. According to our variation model, we generate a new random delay factor set for each Monte Carlo simulation. In case of a plain simulation without random gate delays, the gate delay factors are set to one. This is used to verify the circuit in the nominal case before Monte Carlo simulations. A "golden" behavior netlist is co-simulated to validate the circuit functionality with the same random input vectors. A MC run is considered as a "pass" only when there are no setup or hold violations and no mismatches between the primary outputs of the model under simulation with those of the golden results.

5 SIMULATION RESULTS

We compare the proposed method with a baseline approach proposed in Reference [28] that applies a constant hold margin to each timing critical path in the circuit. The fixed margin approach does not consider clock topology and requires, in general, multiple iterations to obtain a design-specific

Table 2. Number of Hold Buffers (JTLs) and Timing Yield (%) for the Fixed Margin (7 ps), and the Proposed Variation-based 3σ and 2σ Hold Margin, with *Incremental Placement*

		fixed margin (7ps)			3σ					2σ				
Design	**Std Dev**	**Yield(%)**	**# JTLs plc**	**# JTLs replc**	**Yield(%)**	**# JTLs plc**	**# JTLs replc**	**Yield+(%)**	**# JTLs-(%)**	**Yield(%)**	**# JTLs plc**	**# JTL replc**	**Yield+(%)**	**# JTLs-(%)**
c432	0.0808	99.1	1508	1564	99.7	1012	1420	0.6	9.2	95.4	1012	1185	−3.7	24.2
c499	0.0808	95.9	564	584	99.7	426	531	4.0	9.1	99.3	426	470	3.5	19.5
c880	0.0808	90.4	1031	1151	99.8	757	1018	10.4	11.6	95.4	757	871	5.5	24.3
c1355	0.0809	99.1	577	591	100.0	431	525	0.9	11.2	97.8	431	470	−1.3	20.5
c1908	0.0808	97.5	1085	1150	99.7	739	985	2.3	14.3	98.3	739	830	0.8	27.8
c2670	0.0831	93.8	2831	3570	99.8	2513	3573	6.4	−0.1	96.1	2513	3196	2.5	10.5
c3540	0.0829	93.7	2055	2294	99.7	1416	2190	6.4	4.5	93.5	1416	1770	−0.2	22.8
c5315	0.0829	90.2	4199	5036	99.1	3055	4747	9.9	5.7	91.4	3055	3939	1.3	21.8
c6288	0.0829	92.5	4697	6039	99.2	3592	5404	7.2	10.5	93.0	3592	4685	0.5	22.4
c7552	0.0829	86.8	2601	2990	98.9	1794	2745	13.9	8.2	94.2	1794	2252	8.5	24.7
AVG.	**0.0819**	**93.9**	**2115**	**2497**	**99.6**	**1574**	**2313.8**	**6.2**	**8.4**	**95.4**	**1574**	**1967**	**1.7**	**21.9**

fixed margin that achieves a high timing yield while managing the overhead in terms of incurred area. In contrast, our proposed method adds more hold margins to paths where the lowest common ancestor of sequentially adjacent FFs is higher in the clock tree and thereby achieves superior results over all the benchmarks. We also compare the two proposed re-placement methods, i.e., placement from scratch and incremental placement.

We run Monte Carlo simulations on the placed ISCAS'85 benchmarks [2]. All experiments were run on two Intel Xeon E5-2450 v2 CPUs with 128 GB of RAM. We experimented with both 2σ and 3σ variations in hold time margin with random Gaussian distributed gate delays to analyze the tradeoff between area and timing yield. For the fixed margin approach, we evaluated hold margin values of 10 ps and 4 ps, before settling on a compromise of 7 ps, optimizing both buffer area overhead and timing yield. In all our experiments, the additional white-space around logic cells and hold buffers designed to reduce local routing congestion (i.e., ensuring a routable solution is produced) and inserting hold buffers in the final step of timing closure, is set to 8× the routing track (10 μm). To solve the MILP problem for clock tree synthesis, the CPLEX time limit is set to 60 minutes.

Table 2 lists the timing yield values and the number of inserted hold buffers before and after re-placement, for the baseline, 2σ, and 3σ approaches. In these experiments, the clock frequencies are set to the delay of the longest timing path with a small setup margin. In our simulation results, the average clock frequency of fixed and variation-based hold fixing approaches differ by an average of 5% across all the benchmarks.

Compared with the fixed margin approach, our 3σ approach produces netlists with an average reduction of 8.4% in terms of number of hold buffers and an increase in timing yield of 6.2%. The 2σ approach achieves an average of 21.9% saving in terms of number of hold buffers with a 1.7% higher timing yield. The primary source of improvement over the baseline approach comes from the application of CPPR to the clock tree topology.

Note that these results assume no common clock path between PIs/POs and the logic cells. If we estimate the CCP length of PI/PO paths with the average CCP length of the circuit, then the 3σ approach achieves an average of 10.2% saving in hold buffers over the fixed margin approach. Alternatively, if we simply exclude JTLs on PI/PO paths, then the 3σ approach saves an average of 14.1% hold buffers.

The results of the placement from scratch algorithm are shown in Table 3. The 3σ approach shows a 2.0% saving on the number of hold buffers with a 5.9% improvement on timing yield, when compared with the baseline approach. Additionally, the 2σ approach shows a 19.4% saving on the number of hold buffers with a 0.9% drop on yield.

Table 3. Number of Hold Buffers (JTLs) and Timing Yield (%) for the Fixed Margin (7 ps), and the Proposed Variation-based 3σ and 2σ Hold Margin, with *Placement from Scratch*

		fixed margin (7ps)			3σ					2σ				
Design	**Std Dev**	**Yield(%)**	**# JTLs plc**	**# JTLs replc**	**Yield(%)**	**# JTLs plc**	**# JTLs replc**	**Yield+(%)**	**# JTLs-(%)**	**Yield(%)**	**# JTLs plc**	**# JTL replc**	**Yield+(%)**	**# JTLs-(%)**
c432	0.0809	90.7	1508	1935	98.8	1012	1685	8.9	12.9	95.6	1012	1385	5.4	28.4
c499	0.0810	99.2	564	698	99.9	426	731	0.7	−4.7	95.9	426	601	−3.3	13.9
c880	0.0808	97.9	1031	1408	99.5	757	1394	1.6	1.0	92.6	757	1132	−5.4	19.6
c1355	0.0810	96.9	577	695	99.5	431	678	2.7	2.4	95.8	431	573	−1.1	17.6
c1908	0.0808	91.3	1085	1343	99.9	739	1274	9.4	5.1	92.5	739	1032	1.3	23.2
c2670	0.0819	95.7	2831	4170	99.5	2513	4155	4.0	0.4	95.1	2513	3615	−0.6	13.3
c3540	0.0829	94.3	2055	2922	99.3	1416	3105	5.3	−6.3	94.6	1416	2417	0.3	17.3
c5315	0.0828	90.8	4199	8507	97.8	3055	8621	7.7	−1.3	89.3	3055	7143	−1.7	16.0
c6288	0.0829	85.9	4697	7731	98.9	3592	7591	15.1	1.8	87.4	3592	6333	1.7	18.1
c7552	0.0829	94.7	2601	5115	98.3	1794	4658	3.8	8.9	89.1	1794	3742	−5.9	26.8
AVG.	**0.0818**	**93.7**	**2115**	**3452**	**99.1**	**1574**	**3389**	**5.9**	**2.0**	**92.8**	**1574**	**2797**	**−0.9**	**19.4**

Table 4. Number of Hold Buffers (JTLs) for *Placement from Scratch* vs. *Incremental Placement* Approaches

		fixed margin (7ps)			3σ			2σ		
Design	**# logic gates +splitters**	**# JTLs scratch**	**# JTLs increm.**	**Saving (%)**	**# JTLs scratch**	**# JTLs increm.**	**Saving (%)**	**# JTLs scratch**	**# JTLs increm.**	**Saving (%)**
c432	3760	1935	1564	19.2	1685	1420	15.7	1385	1185	14.4
c499	1916	698	584	16.3	731	531	27.4	601	470	21.8
c880	3585	1408	1151	18.3	1394	1018	27.0	1132	871	23.1
c1355	1916	695	591	15.0	678	525	22.6	573	470	18.0
c1908	3596	1343	1150	14.4	1274	985	22.7	1032	830	19.6
c2670	6841	4170	3570	14.4	4201	3573	14.9	3615	3196	11.6
c3540	7831	2922	2294	21.5	3105	2190	29.5	2417	1770	26.8
c5315	15575	8507	5036	40.8	8621	4747	44.9	7143	3939	44.9
c6288	14169	7731	6039	21.9	7591	5404	28.8	6333	4685	26.0
c7552	9420	5115	2990	41.5	4658	2745	41.1	3742	2252	39.8
AVG.	**6861**	**3452**	**2497**	**22.3**	**3394**	**2314**	**27.5**	**2797**	**1967**	**24.6**

Finally, aggregating Tables 2, 3, and 4 provides a comparison between placement from scratch and incremental placement approaches. Under the assumed variations, incremental placement outperforms the placement from scratch approach, with savings on hold buffers with 22.3%, 27.5%, and 24.6% under fixed, 3σ and 2σ margin approaches, respectively. Additionally, incremental placement minimizes the total power consumption associated with DC bias resistors in each SFQ cell by reducing the number of inserted hold buffers, hence static power consumption, which is responsible for most of the circuit power dissipation in standard RSFQ logic [20]. The main advantage in terms of reducing the number of hold buffers originates from minimizing the perturbation to the original placement solution and layout area. As mentioned earlier, our flow utilizes the original timing-aware clock topology, which considers the criticality of the data paths in terms of timing slacks, after placement and legalization of the hold buffers. Therefore, less perturbations to the placement solution and the fixed clock topology lead to fewer changes in the location of clock splitters, less modifications to the clock arrival times, hence less effort on timing closure and hold fixing.

Table 5 presents the results including the minimum clock cycle and the layout area by applying two placement approaches under 3σ margin approach. Under the assumed variations, incremental placement has an average overhead of 12.4% and 0.2% in terms of minimum clock cycle time and layout area, respectively. By distributing the hold buffers such that the logic and hold buffers are somewhat uniformly distributed among all the rows, incremental placement minimizes the impact on layout area. The reason behind the degradation of clock cycle times is that incremental placement sometimes creates setup critical paths by adding to the wire delay of paths with inserted hold buffers, whereas placement from scratch approach, aimed at minimizing the total wirelength

Table 5. CCT and Area Comparison between *Placement from Scratch* vs. *Incremental Placement* (3σ)

		Placement from Scratch		Incremental Placement			
Design	**Std Dev**	**CCT (ps)**	**Area**	**CCT (ps)**	**CCT Overhead**	**Area**	**Area Overhead**
c432	0.0809	116.6	3.72E+07	139.6	19.7%	3.64E+07	-2.3%
c499	0.0810	115.8	1.80E+07	121.7	5.2%	1.82E+07	1.0%
c880	0.0808	118.9	3.19E+07	117.4	-1.2%	3.16E+07	-1.0%
c1355	0.0810	82.0	1.79E+07	92.5	12.8%	1.80E+07	0.1%
c1908	0.0808	88.6	3.20E+07	115.8	30.7%	3.15E+07	-1.6%
c2670	0.0819	249.3	7.17E+07	302.2	21.2%	7.24E+07	0.8%
c3540	0.0829	146.3	7.11E+07	157.0	7.3%	7.26E+07	2.0%
c5315	0.0828	299.5	1.44E+08	324.5	8.4%	1.44E+08	0.4%
c6288	0.0829	204.5	1.30E+08	248.9	21.7%	1.32E+08	1.6%
c7552	0.0829	306.3	9.81E+07	299.9	-2.1%	9.95E+07	1.5%
AVG.	**0.0818**	**172.8**	**6.52E+07**	**192.0**	**12.4%**	**6.57E+07**	**0.2%**

Table 6. Runtime in the Fixed Margin (7ps), and the Proposed Variation-based 3σ and 2σ Hold Margin, with *Incremental Placement*

	Runtime (s)		
Design	**fixed margin (7ps)**	**3σ**	**2σ**
c432	101	92	93
c499	57	56	55
c880	95	87	86
c1355	59	56	57
c1908	90	87	86
c2670	206	199	199
c3540	245	235	237
c5315	618	594	593
c6288	516	484	489
c7552	383	363	361
AVG.	**237**	**225**	**226**

without row restrictions, manages to minimize more of the wirelengths and hence reduces the long wires on some paths that improves the clock cycle time.

Note that if the timing uncertainties were significantly higher, more hold buffers would be needed and the perturbations to the original gate locations would be larger, increasing the overheads of incremental placement. For example, consider circuit c880 with variation increased from 0.080 to 0.134. Then, incremental placement overheads increase to 0.4% in area and 8% in minimum clock cycle time.

Finally, to quantify the scalability of our proposed hold time fixing algorithm, Table 6 summarizes the runtime of our timing closure flow beginning after clock tree synthesis and including incremental placement. Our variation-aware approach takes similar time as the fixed margin approach and even for benchmarks with a large gate count takes less than 10 minutes.

6 CONCLUSIONS

This article presents a variation-aware hold time fixing flow for SFQ circuits. We consider the worst-case scenario in terms of gate delay variations caused by timing uncertainties and apply the common path pessimism removal technique to the clock tree topology to reduce the number

of required hold buffers for closing the timing of a circuit. Our flow allows a tradeoff between timing yield and layout area of the circuit. Additionally, we present two placement techniques, incremental and from scratch, to efficiently place inserted hold buffers while reducing incurred overheads in terms of layout area and maximum clock frequency. The efficacy of the presented methodology is verified using Monte Carlo simulations on ISCAS'85 benchmark circuits and an SFQ5ee-based cell library.

ACKNOWLEDGMENTS

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Office of the Director of National Intelligence, Intelligence Advanced Research Projects Activity, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

REFERENCES

[1] 2019. IBM ILOG CPLEX 12.10. Retrieved from http://www.ilog.com/products/cplex/.

[2] D. Bryan. 1985. The ISCAS'85 benchmark circuits and netlist format. In *Proceedings of the Technischer Bericht, Microelectronics Center ofNorth Carolina (MCNC)*. https://davidkebo.com/documents/iscas85.pdf.

[3] P. Bunyk, K. Likharev, and D. Zinoviev. 2001. RSFQ technology: Physics and devices. *Int. J. High Speed Electron. Syst.* 11 (03 2001). https://doi.org/10.1142/S012915640100085X

[4] P. Cunningham, M. Swinnen, and Steev Wilcox. 2009. Clock concurrent optimization rethinking timing optimizationto target clocks and logic at the same time. Azuro Inc.

[5] C. J. Fourie and M. H. Volkmann. 2013. Status of superconductor electronic circuit design software. *IEEE Trans. Appl. Supercond.* 23, 3 (2013), 1300205–1300205.

[6] Kris Gaj, Eby G. Friedman, and Marc J. Feldman. 1997. Timing of multi-gigahertz rapid single flux quantum digital circuits. *J. VLSI Sign. Process. Syst. Sign. Image Vid. Technol.* 16 (1997), 247–276.

[7] Kris Gaj, Eby G. Friedman, and Marc J. Feldman. 1997. Timing of Multi-Gigahertz rapid single flux quantum digital circuits. *Journal of Vlsi Signal Processing Systems for Signal, Image and Video Technology* 16, 2 (1997), 247–276. https://doi.org/10.1023/A:1007903527533

[8] V. Garg. 2014. Common path pessimism removal: An industry perspective: Special Session: Common Path Pessimism Removal. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 592–595.

[9] Erik W. Grafarend. 2006. *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*. Walter de Gruyter. 553 pages.

[10] K. Han, A. B. Kahng, and J. Li. 2020. Optimal generalized h-tree topology and buffering for high-performance and low-power clock distribution. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 39, 2 (2020), 478–491.

[11] D. S. Holmes, A. L. Ripple, and M. A. Manheimer. 2013. Energy-efficient superconducting computing power budgets and requirements. *IEEE Trans. Appl. Supercond.* 23, 3 (2013), 1701610–1701610.

[12] S. Huang, G. Jhuo, and W. Huang. 2010. Minimum buffer insertions for clock period minimization. In *Proceedings of the 2010 International Symposium on Computer, Communication, Control and Automation (3CA'10)*, Vol. 1. 426–429. https://doi.org/10.1109/3CA.2010.5533776

[13] T. Huang, P. Wu, and M. D. F. Wong. 2014. Fast path-based timing analysis for CPPR. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'14)*. 596–599. https://doi.org/10.1109/ICCAD.2014.7001413

[14] T. Huang, P. Wu, and M. D. F. Wong. 2014. UI-Timer: An ultra-fast clock network pessimism removal algorithm. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'14)*. 758–765. https://doi.org/10.1109/ICCAD.2014.7001436

[15] K. Hubert. 2008. *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press.

[16] Inhak Han, Daijoon Hyun, and Youngsoo Shin. 2016. Buffer insertion to remove hold violations at multiple process corners. In *Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC'16)*. 232–237. https://doi.org/10.1109/ASPDAC.2016.7428016

[17] Michael A. B. Jackson, Arvind Srinivasan, and E. S. Kuh. 1990. Clock routing for high performance ICs. In *Proceedings of the ACM/IEEE Design Automation Conference*. 573–579.

[18] K. K. Likharev and V. K. Semenov. 1991. RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Trans. Appl. Supercond.* 1, 1 (1991), 3–28.

[19] M. A. Bender and M. F. Colton. 2000. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, LNCS, Vol. 1776, 88–94. https://doi.org/10.1007/10719839_9

[20] O. A. Mukhanov. 2011. Energy-efficient single flux quantum technology. *IEEE Trans. Appl. Supercond.* 21, 3 (2011), 760–769.

[21] L. C. Müller, H. R. Gerber, and C. J. Fourie. 2008. Review and comparison of RSFQ asynchronous methodologies. *J. Phys. Conf. Ser.* 97 (Feb. 2008), 12109. https://iopscience.iop.org/article/10.1088/1742-6596/97/1/012109/pdf.

[22] Pei-Ci Wu, M. D. F. Wong, I. Nedelchev, S. Bhardwaj, and V. Parkhe. 2014. On timing closure: Buffer insertion for hold-violation removal. In *Proceedings of the 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC'14)*. 1–6. https://doi.org/10.1145/2593069.2593171

[23] Daniel A. Reed and Jack Dongarra. 2015. Exascale computing and big data. *Commun. ACM* 58, 7 (Jun. 2015), 56–68. https://doi.org/10.1145/2699414

[24] S. N. Shahsavani and M. Pedram. 2019. A minimum-skew clock tree synthesis algorithm for single flux quantum logic circuits. *IEEE Trans. Appl. Supercond.* 29, 8 (2019), 1–13.

[25] Soheil Nazar Shahsavani, Bo Zhang, and Massoud Pedram. 2020. A timing uncertainty-aware clock tree topology generation algorithm for single flux quantum circuits. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'20)*. 278–281.

[26] Michael Shell. 2020. ABC: A System for Sequential Synthesis and Verification. Retrieved from https://people.eecs.berkeley.edu/~alanmi/abc/.

[27] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. 1993. Minimum padding to satisfy short path constraints. In *Proceedings of the 1993 International Conference on Computer Aided Design (ICCAD'93)*. 156–161. https://doi.org/10.1109/ICCAD.1993.580048

[28] R. N. Tadros and P. A. Beerel. 2020. Optimizing $(HC)^2LC$, A robust clock distribution network for SFQ circuits. *IEEE Trans. Appl. Supercond.* 30, 1 (2020), 1–11.

[29] R. N. Tadros, A. Fayyazi, M. Pedram, and P. A. Beerel. 2020. Systemverilog modeling of SFQ and AQFP circuits. *IEEE Trans. Appl. Supercond.* 30, 2 (2020), 1–13.

[30] S. K. Tolpygo, V. Bolkhovsky, D. E. Oates, R. Rastogi, S. Zarr, A. L. Day, T. J. Weir, A. Wynn, and L. M. Johnson. 2018. Superconductor electronics fabrication process with MoNx kinetic inductors and self-shunted josephson junctions. *IEEE Trans. Appl. Supercond.* 28, 4 (2018), 1–12.

[31] I. V. Vernik, Q. P. Herr, K. Gaij, and M. J. Feldman. 1999. Experimental investigation of local timing parameter variations in RSFQ circuits. *IEEE Trans. Appl. Supercond.* 9, 2 (1999), 4341–4344.

[32] T. Xiao, H. Bagga, G. J. Chen, R. Cheung, and R. Pattipati. 2011. Path aware event scheduler in HoldAdvisor for fixing min timing violations. In *Proceedings of the IEEE 29th International Conference on Computer Design (ICCD'11)*. 71–77. https://doi.org/10.1109/ICCD.2011.6081378

[33] J. Xiong, V. Zolotov, and L. He. 2007. Robust extraction of spatial correlation. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 26, 4 (2007), 619–631.

[34] J. Zejda and P. Frain. 2002. General framework for removal of clock network pessimism. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 632–639.

[35] B. Zhang and M. Pedram. 2020. qSTA: A static timing analysis tool for superconducting single-flux-quantum circuits. *IEEE Trans. Appl. Supercond.* 30, 5 (2020), 1–9.

Received December 2020; revised March 2021; accepted April 2021