QuCTS—Single-Flux Quantum Clock Tree Synthesis

Rassul Bairamkulov, *Graduate Student Member, IEEE*, Tahereh Jabbari, *Graduate Student Member, IEEE*, and Eby G. Friedman, *Fellow, IEEE*

***Abstract*—Superconductive rapid single-flux quantum (RSFQ) is an emerging cryogenic technology, promising a significant boost in performance and ultralow power consumption. The operating frequency achieved by RSFQ digital integrated circuits is several orders of magnitude greater than traditional CMOS circuits. The fundamental difference of RSFQ circuits, however, renders traditional clocking techniques appropriate for CMOS unsuitable for RSFQ technology. Most RSFQ logic gates, such as AND and OR, are sequential in nature. The number of pipeline stages is therefore significantly greater in RSFQ as compared to CMOS, complicating the clock distribution network design process. This issue is further exacerbated with the need for splitters to achieve a fanout greater than one and the need for transmission lines rather than ordinary metallic wires as in CMOS. In this work, QuCTS—single-flux Quantum (SFQ) Clock Tree Synthesis—is presented. QuCTS utilizes a two-stage framework for synthesizing clock networks. In the clock skew scheduling stage, the clock signal arrival time of each gate is chosen to maximize the robustness of the circuit to timing variations. In the clock tree synthesis stage, the layout of the clock distribution network is generated based on a novel delay equilibration technique. QuCTS is the first clock tree synthesis tool for RSFQ circuits utilizing useful clock skew. The synthesized network satisfies the clock arrival time requirements while minimizing the associated overhead, such as the interconnect length and number of delay elements. The tool is validated on a set of benchmark circuits. In a prototypical case study, a clock tree is generated for the AMD2901 with 1049 clock sinks in 53 min while satisfying the clock arrival time.**

***Index Terms*—**

I. Introduction

RAPID single-flux quantum (RSFQ) technology offers a range of advantages as compared to CMOS. Several orders of magnitude greater operating frequency and three orders of magnitude lower power are among the most prominent advantages of RSFQ. Substantial progress has been made in the field of superconductive electronics in the past decades. SFQ manufacturing technology is capable of accommodating over 6000 Josephson junctions (JJ) per mm^2 [1]. An 8-bit superconductive microprocessor operating at a frequency of 80 GHz has been successfully fabricated [2]. Ongoing advancements in electronic design automation for RSFQ circuits are expected to enable the large scale integration of superconductive digital systems [3], [4].

Beyond the necessity for the cryogenic operation below approximately 4K and the relatively low density on-chip integration as compared to CMOS, the design of a robust on-chip clock distribution network remains a significant challenge in RSFQ systems [5]. The fundamental properties of RSFQ technology are described in the seminal work of Likharev and Semenov [6]. Unlike traditional CMOS, where the information is represented with a high or low DC voltage level, short quantized voltage pulses transfer information in RSFQ. A logical high or low is represented, respectively, by the presence or absence of a single-flux quantum (SFQ) pulse within a certain time interval. Most logic gates in RSFQ are therefore sequential, such as AND and OR gates that are combinatorial in CMOS. This structure drastically increases the pipeline depth as compared to CMOS, complicating the clock network design process. The complexity of the clock distribution network is further exacerbated by the interconnect structures in RSFQ systems [7]. Unlike CMOS, where the gates can be connected with a simple wire, RSFQ interconnect is either a passive transmission line (PTL) requiring a driver, receiver, and impedance matching [8], or an active Josephson transmission line (JTL) requiring bias current for each JJ. Finally, most RSFQ gates have a fanout of one. A splitter gate is used to generate two (or more) SFQ pulses from an input signal [7], [9].

Different approaches to clocking RSFQ circuits have been reported in the literature. Clockless self-timed systems have been proposed [10]–[13]. An effective operating frequency of 20 gigahertz has been demonstrated while eliminating the overhead of the clock distribution network. Self-timed circuits, however, remain vulnerable to timing violations, exhibit unpredictable performance due to sensitivity to logic delays, and use handshaking circuitry that requires significant area [14].

Hierarchical chains of homogeneous clover-leaves clocking $(HC)^2LC$ are described in [15]. The primary advantage of this structure is robustness since the clock period of the system adapts to the slowest hierarchical chain. Another advantage is the elimination of race condition hazards due to forced counterclocking [5], [15]. The primary drawback is reduced clock speed since the worst case path determines the clock period of the entire system. Another drawback of this method

Manuscript received 12 May 2021; revised 31 July 2021; accepted 26 September 2021. Date of publication 26 October 2021; date of current version 20 September 2022. This work was supported in part by the National Science Foundation under Grant CCF-1716091; in part by the Intelligence Advanced Research Projects Activity under Grant W911NF-17-9-0001; and in part by the Qualcomm and Synopsys. This article was recommended by Associate Editor R. Wille. *(Corresponding author: Rassul Bairamkulov.)*

The authors are with the Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627 USA (e-mail: rbairamk@ur.rochester.edu; friedman@ece.rochester.edu).

Digital Object Identifier 10.1109/TCAD.2021.3123141

is the underutilization of clock skew as an additional degree of design freedom. Counterclocking increases the setup time constraints which limit the minimum clock period [5].

A minimum skew clock tree synthesis algorithm for SFQ circuits is proposed in [16]. The algorithm incorporates the CMOS-based deferred merge embedding (DME) algorithm [17] to generate a zero skew clock tree. Due to the nonnegligible dimensions of the splitters, the clock tree generated by DME typically violates RSFQ design rules. A legalization step is therefore proposed [16] to correct the layout at the cost of introducing small skew into the clock tree. Minimizing the clock skew, however, results in a suboptimal clock frequency [18] and does not guarantee correct functionality [19]. Furthermore, nonzero clock skew in data paths can improve the performance and robustness of the synchronous system [20]. With clock skew scheduling, extra delay in the fast data paths is exploited to decrease the effective delay of the critical paths, thereby increasing the maximum attainable operating frequency [18], [21]–[24].

While clock skew optimization may provide a significant gain in performance and robustness, it is often overlooked in existing RSFQ clocking approaches. To bridge this gap, QuCTS, a SFQ clock tree synthesis algorithm, is introduced. In the clock skew scheduling stage, the arrival time of each clocked gate is based on the algorithm from [18] and [25]. The limitations of RSFQ technology, such as the use of splitters and limited fanout, prevent the direct application of CMOS techniques for clock tree synthesis. A novel method for generating a clock tree topology for RSFQ systems based on hierarchical clustering is proposed here. Unlike CMOS, the RSFQ clock tree is complicated by many splitters and delay elements whose placement is restricted to a set of vacant gate cells. A primary contribution of this article is the delay equilibration algorithm for RSFQ clock tree layout synthesis. By judiciously placing the splitters and delay elements and adjusting the interconnect, the clock arrival times determined from the clock skew scheduling stage are satisfied.

This article is organized as follows. In Section II, the clock skew scheduling algorithm is presented. The binary clock tree synthesis process is described in Section III, followed by the delay equilibration process presented in Section IV. The performance of the algorithm is evaluated in the case study and benchmark circuits presented in Section V, followed by the conclusion in Section VI.

II. Clock Skew Scheduling

Clock skew scheduling is a powerful technique to maximize the speed and robustness of a synchronous system [18], [22], [23]. Despite the potential benefits of useful clock skew, it is often viewed as a parasitic effect requiring minimization [26], [27]. In addition, achieving zero clock skew is quite difficult due to process and environmental variations as well as electromagnetic interference that permeate not only CMOS but also RSFQ circuits [5], [15], [28].

The first stage of QuCTS, presented in this section, mitigates this issue by adapting clock skew scheduling within the RSFQ circuit design process. QuCTS operates in four stages. The sequential circuit topology, described in Verilog, is initially converted into a sequential graph. The minimum clock period is determined by evaluating the expected delay and delay uncertainty of each data path. The permissible range (PR) of each data path is a function of the clock skew in sequentially adjacent registers [19], [20], [24]. The clock skew schedule is generated using a quadratic programming algorithm that maximizes the robustness of the circuit to parameter variations [18], [25]. The clock skew schedule is converted into a schedule of clock arrival times that is passed to the clock tree synthesis algorithm.

A. Sequential Graph

The first step in the clock skew scheduling process is conversion of the circuit topology into a directed sequential graph $G = (V, E, d^{\min} : E \to \mathbb{R}, D^{\max} : E \to \mathbb{R})$, where V is the set of nodes, E is the set of edges, and $d^{\min}$ and $D^{\max}$ are, respectively, the minimum and maximum delay of an edge in E. A typical sequential circuit consists of inputs, outputs, clocked gates, nonclocked gates, and interconnects. For brevity, the clocked and nonclocked gates are referred to as, respectively, registers and gates. Each edge $(i, j) \in E$ represents a combinational data path $p_{i,j}$ from a source to target register. The range of delays $d_{i,j} = [d_{i,j}^{\min}, D_{i,j}^{\max}]$ of an edge (i, j) within a graph is the sum of the delays along a data path

$$d_{i,j} = \sum_{k \in p_{i,j}} \left(d_k^{\text{gate}} + d_k^{\text{int}} \right) \tag{1}$$

where d_k^{int} denotes the range of delay of the interconnect between gate k and the next gate, and d_k^{gate} denotes the range of the input-to-output delay of gate k or a clock-to-output delay of register k. The gate and register delays are supplied externally as input data.

The inputs and outputs of a sequential circuit are often described in Verilog as floating signal nets. This structure is not supported in a graph, where the edges require both source and target nodes. Furthermore, it is often desired that the clock skew between the input and output nodes of a circuit is 0 [18]. A dummy I/O node is therefore added to the sequential graph, as illustrated in Fig. 1. The I/O node is the tail (source) of each input edge and the head (target) of each output edge. The dummy node is treated as a standard node during the clock skew scheduling process. Since a node cannot have a nonzero clock skew with itself, zero clock skew is ensured among the circuit inputs and outputs.

B. Minimum Clock Period

In the zero clock skew approach, the minimum clock period is determined by the delay of the critical paths. In a nonzero clock skew system, however, finding the minimum clock period requires a significantly more sophisticated process. The minimum clock period is determined by the cycles and reconvergent paths within the sequential graph [18], as shown in, respectively, Fig. 2(a) and (b).

An example of a sequential circuit containing cycle p_{ii} with n nodes is shown in Fig. 2(a). To ensure the correct operation

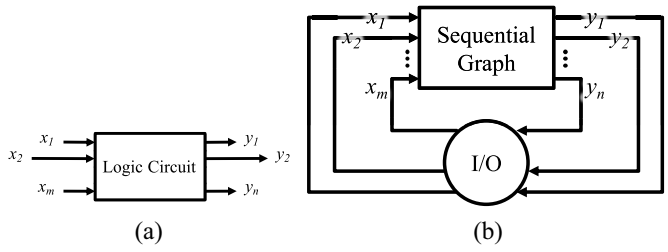


Fig. 1. Processing of inputs and outputs of a logic circuit in a sequential graph. (a) Initial system with inputs $x_1, \ldots, x_m$ and outputs $y_1, \ldots, y_n$. Note that the input and output edges (signal nets in Verilog) typically have one floating terminal. (b) Sequential graph representation of the input and output edges in QuCTS. The floating terminals of the input and output edges are connected to a dummy I/O node. This node acts as a tail (source) of all input edges and a head (target) of all output edges. The I/O node eliminates any clock skew between the circuit terminals.

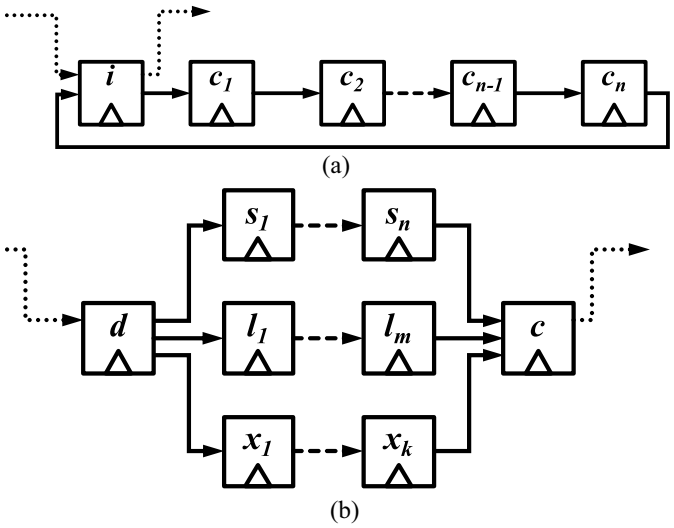


Fig. 2. Constraints of the minimum clock period within a sequential circuit. (a) Cycle path with n registers starting with node i. The dotted arrows represent the connection to external circuitry. (b) Reconvergent path between registers d and c.

of the circuit including this cycle, the clock period cannot be smaller than

$$T^i = \frac{1}{n} \sum_{j \in p_{ii}} \left(D^{\max}_{j,j+1} + \delta^s_{j+1} \right) \tag{2}$$

where δ^s_{j+1} is the setup time of the gate following gate j. The clock skew within the cycle is fixed at zero, since, as previously mentioned, a register cannot have a nonzero clock skew with itself [24]. Equation (2) therefore requires the average propagation delay of a data path within a cycle to not be greater than the clock period. Finding the minimum clock period requires determining the cycles within sequential graph G. The computational complexity of finding all cycles within a graph is $O((|V| + |E|)(n_c + 1))$, where n_c is the number of cycles within a graph.

The reconvergent paths are distinct sequential paths that begin at the same divergent register d and end at the same convergent register c. The optimization of these reconvergent paths includes delay insertion, i.e., intentionally adding delay to specific data paths to align the arrival time of the signals, thereby reducing the minimum clock period. Consider the example illustrated in Fig. 2(b). The short path $(s_1, \ldots, s_n)$ with n nodes has the smallest propagation delay, and the long path $(l_1, \ldots, l_m)$ with m nodes has the largest propagation delay. The minimum clock period T^{dc} due to the reconvergent paths between nodes d and c is

$$T^{dc} = \frac{D_l - D_s + \delta^s_c + \delta^h_c}{|m - n + 1|} \tag{3}$$

where D_l and D_s are, respectively, the maximum propagation delay of p_l and minimum propagation delay of p_s, and δ^s_c and δ^h_c are, respectively, the setup and hold time of convergent register c. While delay padding may reduce the minimum clock period, this method requires finding all reconvergent paths within graph G. The complexity of finding a single simple path in a directed graph is $O(|V| + |E|)$ [29]. The number of simple paths can however be prohibitively large, up to $|V|!$ in a fully connected graph. Depending upon the complexity, an integrated circuit may contain hundreds of thousands of nodes, leading to an exorbitant number of simple paths. Delay insertion is therefore not practical for large circuits. An alternative approach, adapted from [30], is utilized in the algorithm presented here. The minimum clock period is determined by the delay uncertainty of the edges

$$T^{\min}_{i,j} = \max_{(i,j) \in E} \left(D^{\max}_{i,j} - d^{\min}_{i,j} + \delta^s_j + \delta^h_i \right) \tag{4}$$

where δ^h_i is the hold time of register i.

The minimum clock period of the overall system is

$$T_{\min} = \max \left(\max_{(i,j) \in E} \left(D^{\max}_{i,j} - d^{\min}_{i,j} + \delta^s_j + \delta^h_i \right), \max_{i \in V} \left(T^i \right) \right). \tag{5}$$

This minimum clock period determines the target clock period in the clock skew scheduling process, as described in Section II-C. Note that although setting the clock period to $T_{\min}$ maximizes the performance of the system, a higher clock period can be chosen to improve other metrics, such as robustness to parameter variations [19], [20].

C. Clock Skew Optimization

Once the minimum clock period is determined, clock skew optimization is performed in two steps. The PR [19], [20], [31] of the clock skew for each path is used to form an objective function. The basis cycles are determined within the graph to form a constraint function. The clock skew schedule is optimized for robustness to parameter variations.

The PR is the range of clock skew between sequentially adjacent registers i and j that satisfy the setup and hold constraints of a circuit [24], [31], defined as

$$\mathrm{PR}_{i,j} = \left[-d^{\min}_{i,j} + \delta^h_i, T_{CP} - D^{\max}_{i,j} - \delta^s_j \right] \tag{6}$$

where T_{CP} is the target clock period. In vector form, the upper and lower bound of the PR for every combinational data path is expressed as vectors $\mathbf{s}_{\min}, \mathbf{s}_{\max} \in \mathbb{R}^{|E|}$. To maximize the robustness of the system, the clock skew of each data path is

maintained at the center $\mathbf{s}^*$ of the PR,

$$\mathbf{s}^* = \frac{1}{2}(\mathbf{s}_{\min} + \mathbf{s}_{\max}). \tag{7}$$

Clock skew deviations arising from parameter variations are therefore less likely to cause a setup or hold time violation. Note however that due to timing constraints, such as cycles, maintaining the clock skew at the center of the PR is often not possible [20]. The scheduling process therefore sets each target clock skew as close to the center of the PR while satisfying the local timing constraints.

Let $\mathbf{s} \in \mathbb{R}^{|E|}$ be the vector of clock skews for each local data path. The clock skew scheduling optimization problem is expressed as

$$\textbf{Minimize:}_{\mathbf{s}} \quad \left\|\mathbf{s} - \mathbf{s}^*\right\|^2 \tag{8}$$

$$\textbf{subject to} \quad s_i^{\min} \le s^i \le s_{\max}^i \forall i \in \mathbb{N}, i \le |E| \tag{9}$$

$$B\mathbf{s} = \mathbf{0} \tag{10}$$

where $s_i^{\min}$, s^i, and $s_{\max}^i$ are the ith element of, respectively, $\mathbf{s}_{\min}$, $\mathbf{s}$, and $\mathbf{s}_{\max}$; $\mathbf{0} \in \mathbb{R}^{|E|}$ is the zero vector; and $B \in \mathbb{R}^{(|E|-|V|+1)\times|E|}$ is the circuit connectivity matrix of graph G [18]. With (8), the clock skew of each data path is placed as close to the center of the PR as possible [20], [24]. Expression (9) requires the clock skew of each data path to be within the PR. Expression (10) requires the clock skew within a cycle to be zero. Each row b_i in B represents an independent cycle in G. The entry $b_{i,j}$ is equal to 1 or -1 if the edge, respectively, follows or opposes the direction of the cycle, and 0 if the edge does not belong to the cycle. An efficient solution of this problem can be achieved with quadratic programming (QP) in $O(|V|^3)$ time [25].

Once the final clock skew schedule is generated, a schedule of clock arrival times is produced. An arbitrary node x is marked as a reference node with a clock arrival time of 0. The clock arrival time at each register is determined using the fundamental equation of clock skew [24]

$$s_{i,f} = \tau_i - \tau_f \tag{11}$$

where τ_i and τ_f are, respectively, the clock arrival time at the initial and final register of a local data path. The arrival time τ_p of the register p preceding register x is

$$\tau_p = s_{p,x} + \tau_x \tag{12}$$

where $s_{p,x}$ is the clock skew of the edge (p, x) determined from the optimization process. Similarly, the arrival time τ_s of the successor s of register x is

$$\tau_s = \tau_x - s_{x,s} \tag{13}$$

where $s_{p,x}$ is the clock skew of the edge (x, s). The process is repeated until the arrival time at each register is determined. The resulting schedule of arrival times is passed to the clock tree synthesis algorithm, as described in Section III.

III. Clock Tree Synthesis

Once the clock arrival time of each logic gate is determined, the objective is to generate a clock network that satisfies these arrival times. A single external clock source is assumed in QuCTS. A tree structure distributes the clock signal from a single source to multiple sinks [32]. Due to the limited fanout of RSFQ gates, splitters are required to distribute the clock signal to many gates within a circuit. Standard splitters provide a fanout of two. Nonstandard splitters with a higher fanout exist, although the bias margins are significantly degraded as compared to standard splitters [7], [9]. A binary clock tree is therefore produced by QuCTS.

To distribute the clock signal to N gates, $N-1$ splitters are required, forming a directed binary tree

$$T = (V_T, E_T) \tag{14}$$

$$V_T = V_{\text{SPL}} \cup V_{\text{sink}} \tag{15}$$

where V_{sink} is the set of clock sinks (logic gates), and V_{SPL} is the set of splitters. The leaf nodes within T (i.e., nodes with zero fanout) correspond to the clock sinks. Other nodes correspond to splitters and have a fanout of two. The root node corresponds to the hierarchically topmost splitter, as shown in Fig. 3. The clock signal initially arrives at the root node within the clock tree and passes to the splitters corresponding to child nodes 0 and 1. At each successive node of tree T, the clock signal is split into multiple signals that eventually arrive at each sink within a cluster. The arrival time of the clock signal is the delay from the clock signal source (root node) to the clock sink. This delay is comprised of splitter delays, interconnect delay, and any intentional delay. By varying these components, the arrival time of the signal can be controlled to satisfy the timing requirements of each clock sink. The objective of the clock tree synthesis process in RSFQ is to produce a binary clock tree that delivers the clock signal at a precise time with the minimum interconnect and junction area.

The first step in the clock tree synthesis process is to produce a binary tree. A common approach in the binary tree synthesis is clustering [33], as illustrated in Fig. 3. Each gate is represented as a point in a 2-D or 3-D space. The location of each gate is represented by an X and Y coordinate, and the weighted clock signal wT serves as a third dimension. The importance of the clock arrival time is controlled by the weight parameter w. If w is large, distant gates exhibiting similar arrival times are grouped within the same cluster. Fewer delay elements are therefore needed in this case since the difference in arrival time of the gates within a cluster is generally small. In contrast, with the lower weight, the gates are grouped by physical proximity, disregarding any difference in arrival times. Shorter interconnects are required to connect the gates, since the distance between the target nodes is smaller. More delay elements are however required to accommodate the difference in arrival time of the clock signals.

To evaluate the effect of the clustering algorithm on the clock tree topology, several clustering algorithms are considered, including K-means [34], BIRCH [35], and agglomerative clustering [36]. Several layout patterns are used to evaluate the quality and speed of generating a clock tree. K-means clustering and BIRCH performed best, generating similar, more balanced trees with fewer levels, as compared to agglomerative clustering. The balanced trees exhibit a smaller variation in

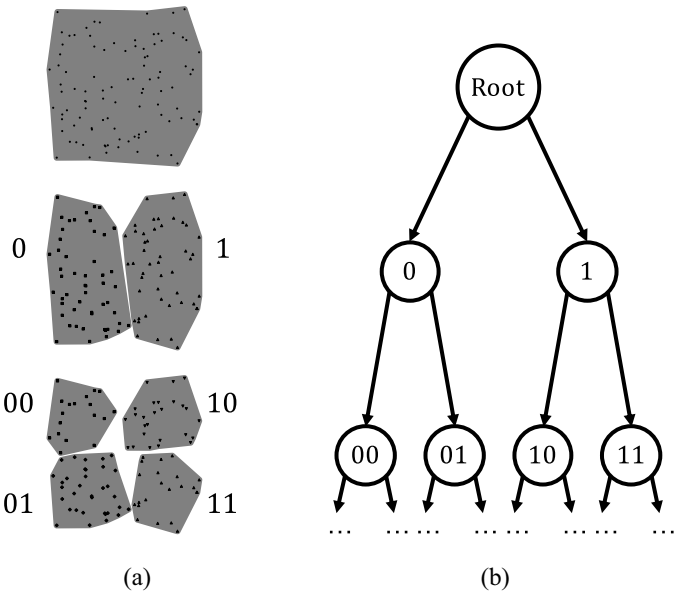


Fig. 3. Binary clock tree generation based on clustering. (a) Hierarchical clustering of the gates based on location. All of the gates are initially placed within a single top-level cluster c (top row). The set of gates are decomposed into two clusters, c_0 and c_1 (middle row), which, in turn, are further divided into smaller clusters (bottom row), until the clusters contain only a single clock sink. (b) Binary clock tree T with each node representing a splitter. The top-level cluster corresponds to the root splitter s_r. Gates within c_0 and c_1 receive the clock signal from the two branches of the root splitter s_r. s_0 and s_1 are added to the binary clock tree as successors of the root splitter s_r to distribute the SFQ clock pulse from s_r to the corresponding clusters. s_{00} and s_{01} (s_{10} and s_{11}) splitters therefore become the successors of splitter s_0 (s_1) to distribute the clock pulse to, respectively, c_{00} and c_{01} (c_{10} and c_{11}). Similarly, each successive clustering step adds two new successor splitters to the corresponding preceding node, producing a binary clock tree.

delay from the clock source to the gates and therefore require fewer delay elements and less wire snaking. BIRCH exhibited the smallest runtime, requiring approximately half the time for generating a clock tree as compared to K-Means. The result of BIRCH was however highly sensitive to the threshold parameter. Conversely, K-means is slower but consistently provides a more balanced tree. Since the clock tree topology requires negligible time as compared to the overall runtime of QuCTS, a more robust K-means algorithm is incorporated into QuCTS.

A binary clock tree is a directed tree, where each node corresponds to a splitter. The topmost (root) splitter $s_r \in T$ receives a clock pulse from an external clock source. The SFQ pulse at each clock sink is delivered through the parent splitter. After the first clustering step, the gates are decomposed into two groups, c_0 and c_1. The two SFQ output pulses of s_r are delivered to clusters c_0 and c_1 via corresponding splitters, respectively, s_0 and s_1. The SFQ pulse at each clock sink within c_0 (c_1) is delivered through splitter s_0 (s_1), as shown in Fig. 3. Each cluster is iteratively decomposed into a pair of subclusters until the size of the cluster is a single gate. A splitter is assigned to each nonsingular cluster, hierarchically distributing the clock signal to the clock sinks.

IV. Delay Equilibration

The binary tree generation process described in the previous section is a guideline for establishing the hierarchy of the gates. The actual connections are determined by a delay equilibration algorithm illustrated in Algorithm 1. To explain this process, consider the two gates shown in Fig. 4(a). Connecting a splitter to these gates via the shortest path is not suitable since a precise arrival time needs to be satisfied. The delay from the splitter to both gates determines the arrival time of the splitter. Delay equilibration is therefore required to satisfy the arrival time at each gate. A splitter can be placed closer to the gate with an earlier arrival time, thereby delivering the SFQ clock pulse earlier [see Fig. 4(b)]. Practically, however, splitter placement is not arbitrary but limited by physical layout constraints. In addition, if the difference in arrival time is large, the splitter placement may be insufficient to balance the arrival time of the clock signals.

In CMOS, a variety of techniques is available to adjust the wire delay, including wire snaking, wire sizing, dummy wire insertion, and active delay elements [37]–[39]. In RSFQ, PTLs require impedance matching, complicating the wire sizing and dummy wire insertion process. The wire snaking technique, illustrated in Fig. 4(c), is suitable for RSFQ, albeit requiring significant area for a modest increase in delay. A significantly larger delay with a relatively small area can be achieved with active delay elements. A JTL can be used as a delay element by controlling the bias current of the JJs [7]. JTLs, however, require dedicated space within the device layer. JTLs are therefore more suitable for providing large delays while PTL-based wire snaking can be used to tune the path delay.

Delay equilibration of a pair of gates requires the precise location of each gate. Since only the position of the clock sinks is initially known, the algorithm generates the clock tree layout in a reverse breadth-first search order. The gates are processed in pairs, starting from the farthermost leaves (sinks) of the tree.

Algorithm 1: Given a Set of Gates Within the Circuit U, Two Gates $A \in U$ and $B \in U$, Target Clock Arrival Times τ_A and τ_B, Wire Pitch h, Set of Vacant Cells P, Set of Delays Realizable by a Delay Element D, and Set of Routing Obstacles O, Place a Splitter and, Optionally, Delay Elements to Deliver the Clock Signal to A and B at, Respectively, τ_A and τ_B

procedure DELAY_EQUILIBRATION;
Input: A, B, τ_A, τ_B, h, P, O
$P_{AB} \leftarrow$ CLOSEST_CELLS$\{P, A, B\}$;
$V_p \leftarrow \{A, B\} \cup P_{AB}$;
$E_p \leftarrow V_p^2 \setminus \{A, B\}$;
$w(a, b) \leftarrow |x_a - x_b| + |y_a - y_b|$;
$G_p \leftarrow (V_p, E_p, w : E_p \rightarrow \mathbb{R})$;
$q_A, q_B, g_k, d_A^*, d_B^* \leftarrow$ PROXY_PATH$(G_p, A, B, \tau_A, \tau_B, D)$;
$wire_A, wire_B, \Delta t, O \leftarrow$ HANAN$(q_A, q_B, g_k, \tau_A, \tau_B, h, O)$;
if $\Delta t > \varepsilon$ **then**
  $wire_A \leftarrow$ AURA_SNAKING$(wire_A, \Delta t, O, U)$;
else if $\Delta t < -\varepsilon$ **then**
  $wire_B \leftarrow$ AURA_SNAKING$(wire_B, -\Delta t, O, U)$;
$\tau_{SPL} \leftarrow \frac{\tau_A - d(wire_A) + \tau_B - d(wire_B)}{2} - d_{SPL}$;
return $wire_A$, $wire_B$, g_k, τ_{SPL}

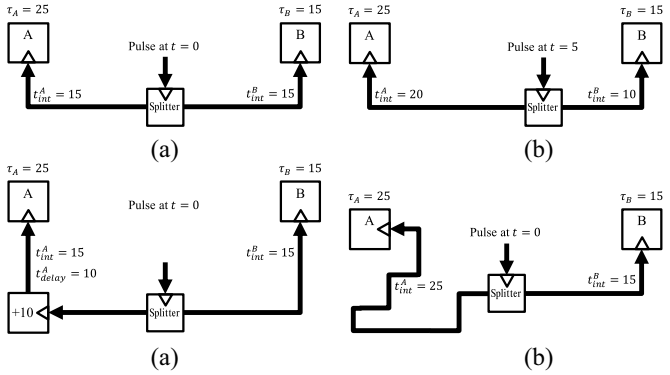


Fig. 4. Example of delay equilibration process. Two gates A and B require a clock pulse to arrive at, respectively, 25 and 15 time units. The clock signal initially arrives at the splitter, where two SFQ pulses for each gate are generated. (a) Example of an invalid topology. While the delay requirement of B is satisfied, A receives the SFQ pulse too early, producing a timing violation. (b) Strategic placement of the splitter closer to B reduces the delay from the splitter to B and increases the delay from the splitter to A. (c) Wire connecting the splitter to A is intentionally lengthened to increase the delay. (d) Delay element is placed between the splitter and A, thereby increasing the delay of the path.

The embedding of the clock tree into the layout is accomplished in three steps. In the coarse embedding step presented in Section IV-A, the location of the splitter, JTL delay elements, and initial PTL routing for every pair of nodes in a binary tree is determined. The local portion of the layout is converted into a proxy graph where the potential location of the splitters and JTLs is determined. The graph is evaluated to determine the location and delay of the splitters and JTLs, satisfying the arrival time of the clock signal with minimum interconnect, as described in Section IV-B. Based on the location of the splitters, JTLs, and blockages, the layout is converted into a Hanan grid [40]. The approximate PTL layout is determined using a shortest path algorithm, such as the A-star algorithm [41]. Precise routing of the interconnect is determined during the fine routing stage, as described in Section IV-C. The delay of the wires is finely adjusted with wire snaking to satisfy the precise requirement of the clock arrival times.

A. Coarse Routing

The coarse routing process for a pair of nodes A and B commences with identifying the cell location for the splitters and JTLs. The layout regions available for the JTLs and splitters are provided to QuCTS as a user input. Based on the cell dimensions and spacing information, these layout regions are converted into a set of points P describing a potential position of a cell (see Fig. 5).

The coarse routing procedure is outlined in Algorithm 2. Delay equilibrium can be achieved with wire snaking or delay insertion [38], [39]. Large delays with wire snaking however require prohibitively large area and increase the likelihood of routing congestion. Delay elements, in contrast, typically produce large delays, rendering these delay elements unsuitable if the delay difference is small. N cells located close to the line connecting nodes A and B form a subset of cells $P_{AB} \subset P$ suitable for routing. These gate cells combined with gates A and B

Algorithm 2: Given a Proxy Graph G_p, Two Nodes A and B, and Set of Delays D Available at Each Delay Element, Determine the Splitter g_k and Paths q_A and q_B Connecting a Splitter to, Respectively, A and B While Satisfying the Difference in Arrival Times Between Nodes

procedure PROXY_PATH;
Input: G_p, A, B, τ_A, τ_B, D
$L^* \leftarrow \infty$;
for *each path* $\{A, g_1, \ldots, g_m, B\}$ *from* A *to* B **do**
  $L = v(w_{A,1} + \cdots + w_{m,B})$;
  for $k \in \{1, 2, m-1, m\}$ **do**
    $q_A \leftarrow (A, g_1, \ldots, g_k)$;
    $q_B \leftarrow (g_k, g_{k+1}, \ldots, g_m, B)$;
    $W_{A,k} = w_{A,1} + \ldots + w_{k-1,k}$;
    $W_{k,B} = w_{k,k+1} + \ldots + w_{m,B}$;
    for *each combination* $\{d_1, \ldots, d_{k-1}\} \in \binom{D}{k-1}$ **do**
      for *each combination* $\{d_{k+1}, \ldots, d_m\} \in \binom{D}{m-k}$ **do**
        $S_{A,k} \leftarrow \sum_{i=1}^{k-1} d_i$;
        $S_{k,B} \leftarrow \sum_{i=k+1}^{m} d_i$;
        $\Delta t \leftarrow \tau_A - \tau_B - W_{A,k} + W_{k,B} - S_{A,k} + S_{k,B}$;
        $\Delta L \leftarrow v\Delta t$;
        $L_{total} \leftarrow L + \Delta L$;
        if $L_{total} < L^*$ **then**
          $L^* \leftarrow L_{total}$;
          $q_A^* \leftarrow q_A$;
          $q_B^* \leftarrow q_B$;
          $d_A^* \leftarrow \{d_1, \ldots, d_{k-1}\}$;
          $d_B^* \leftarrow \{d_{k+1}, \ldots, d_m\}$;
        end
      end
    end
  end
end
return q_A^*, q_B^*, g_k, d_A^*, d_B^*

form the node set of proxy graph vertices $V_p = \{A, B\} \cup P_{AB}$. Each pair of nodes in V_p except $\{A, B\}$ is connected with an edge. The weight of each edge is the Manhattan distance between the terminals. The edge weights therefore represent the length of the shortest rectilinear PTL connecting two points within a layout. For a proxy graph with $N+2$ nodes (two gates and N gate cells), a total of $(1/2)(N+2)(N+1)$ edge weights is determined. The resulting undirected proxy graph is

$$\begin{aligned} G_p &= \left(V_p, E_p, w : E_p \to \mathbb{R}\right) \\ V_p &= \{A, B\} \cup P_{AB} \\ E_p &= \left\{\{a, b\} \in V_p^2 | a \neq b \wedge \{a, b\} \neq \{A, B\}\right\} \\ w(a, b) &= |x_a - x_b| + |y_a - y_b| \end{aligned} \tag{16}$$

where x_a and y_a are, respectively, x and y coordinates of node a.

Four crucial assumptions are made when producing proxy graph G_p.

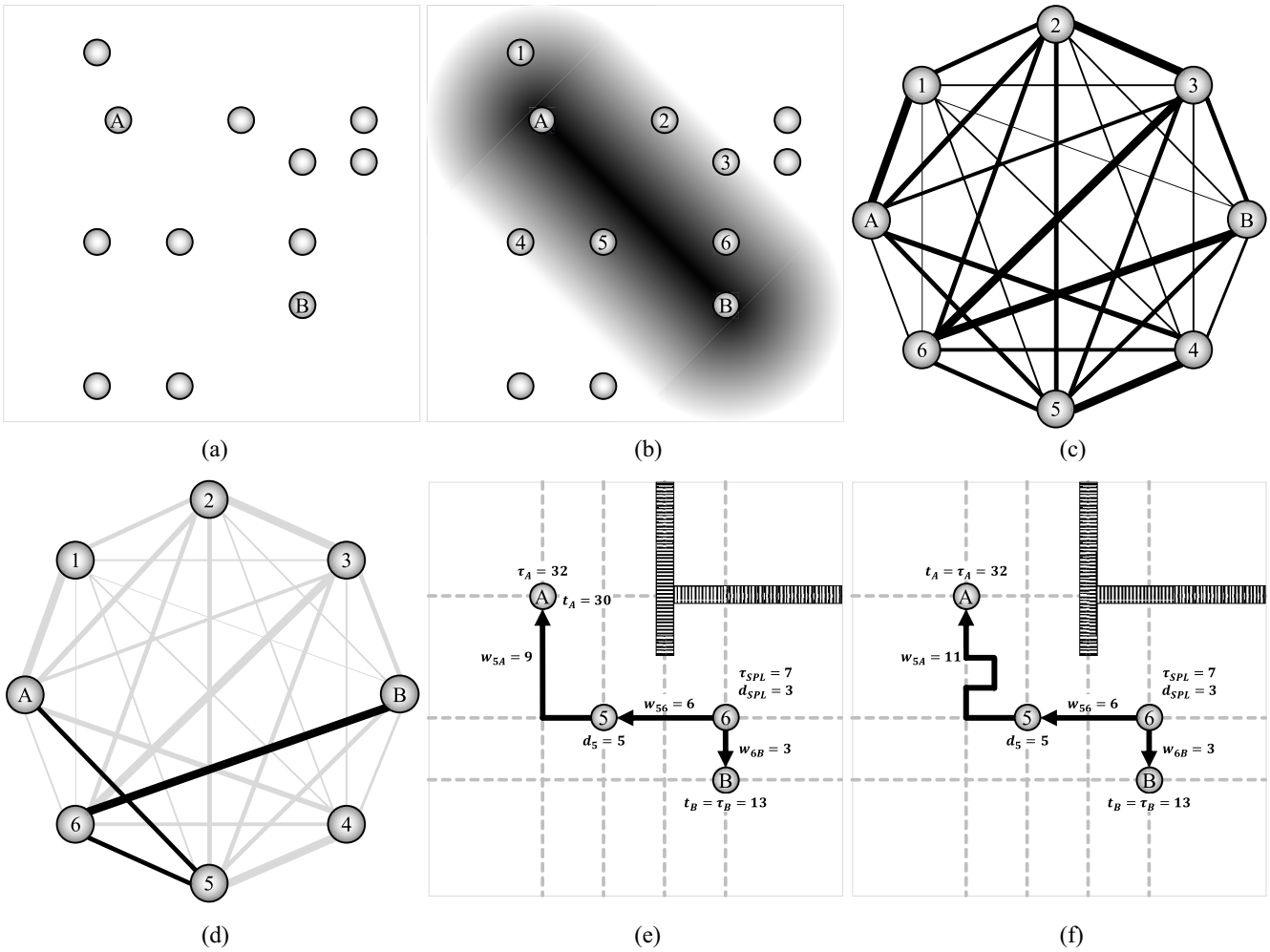


Fig. 5. Example of delay equilibration between gates A and B. τ_i is the arrival time of gate i. (a) Initial layout. The empty circles represent vacant gate cells. (b) Discovery of gate cells in proximity of the line connecting the two gates. The darker areas are closer to the line and are included in the proxy graph. (c) Proxy graph containing six discovered gate cells and gates A and B. The thickness of the edges represents the closeness of the two nodes within a layout. (d) Candidate proxy path $A - g_5 - g_6 - B$ is discovered in a proxy graph. (e) Candidate proxy path transferred to the layout. $w_{i,j}$ is the delay of the path between nodes i and j, and d_i is the delay of the element at cell i. The splitter is therefore placed at node 6. The delay from the splitter to A relative to τ_A is smaller than the delay from the splitter to B relative to τ_B. The arrival time of the splitter is therefore based on the arrival time of B. Additional delay is required along the path to node A. (f) Using wire snaking, additional delay is introduced along the path from the splitter to A. The arrival time is satisfied for both A and B.

1) Each gate is equipped with a PTL transmitter and receiver [42]. Including the PTL driver and receiver within each gate reduces the complexity of the routing process and enables a linear relationship between the length and delay of an interconnect [42].
2) The placement of splitters and delay elements is limited to certain areas of the layout. This assumption is consistent with a typical RSFQ IC layout, where the placement of the cells is limited to narrow regions, such as the cell rows [43]–[45]. Only those nodes within the dedicated regions have a connection to the vacant gate cells. Other nodes are not connected to the device layer, preventing placement of the devices within prohibited zones. QuCTS can however handle arbitrary cell placement regions.
3) The size of the splitters and delay elements is assumed to be similar [46] and cells do not overlap. These assumptions simplify the placement of the splitters and delay elements, accelerating the clock tree synthesis process.
4) The orientation and pin configuration of the cells are assumed flexible, allowing the splitters and JTL elements to be arbitrarily oriented to satisfy routing needs.

Note that edge $\{A, B\}$ is explicitly excluded from the proxy graph since this proxy path does not include a necessary gate cell for a splitter. The paths within a proxy graph model the connections in the layout. In this article, these paths are referred to as proxy paths. A shorter path corresponds to a PTL connection with a smaller interconnect length. To determine the shortest proxy paths, the k-shortest path algorithm, described in [47], is used. This algorithm finds all loopless paths from the source to the target in increasing edge weight. With this algorithm, the proxy paths requiring the least interconnect resources are identified.

B. Analysis of Proxy Path Delay

If the proxy path contains more than one gate cell, the splitter placement is determined by the delay analysis described in this section. For example, consider path $A-g_5-g_6-B$ shown in Fig. 5(d). Placing a splitter at g_5 requires the SFQ clock pulse to arrive at the splitter at

$$\tau_{\text{SPL}|g_5} = \tau_A - w_{A,5} - d_{\text{SPL}} \tag{17}$$

where d_{SPL} is the splitter delay. The resulting clock arrival time at node B is

$$t_B = \tau_{\text{SPL}|g_5} + d_{\text{SPL}} + w_{5,6} + d_6 + w_{6,B} \gg \tau_B \tag{18}$$

where d_i is the delay of the element placed at node g_i, and $w_{i,j}$ for brevity is equivalent to $w(g_i, g_j)$. The resulting arrival time is significantly later than the required arrival time. Correcting this discrepancy with wire snaking requires a significant area. If the splitter is instead placed at cell g_6, the SFQ clock pulse arrives at

$$\tau_{\text{SPL}|g_6} = \tau_A - w_{A,5} - d_5 - w_{5,6} - d_{\text{SPL}} \tag{19}$$

yielding a clock arrival time at B

$$t_B = \tau_{\text{SPL}|g_6} + d_{\text{SPL}} + w_{6,B} \approx \tau_B. \tag{20}$$

The discrepancy in arrival time is minimized and can be corrected with less area overhead using wire snaking.

To generalize this algorithm, consider path $A - g_1 - \cdots - g_m - B$ with one splitter and $m-1$ delay elements. Placing a splitter at cell g_k produces two paths

$$q_A(g_k) = (A, g_1, \ldots, g_{k-1}, \text{SPL}) \tag{21}$$

$$q_B(g_k) = (SPL, g_{k+1}, \ldots, g_m, B). \tag{22}$$

The delay of each path is the sum of the splitter delay d_{SPL}, interconnect delay, and intentional delay,

$$d(q_A(g_k)) = W_{A,k} + S_{A,k} + d_{\text{SPL}} \tag{23}$$

$$d(q_B(g_k)) = W_{k,B} + S_{k,B} + d_{\text{SPL}} \tag{24}$$

where

$$W_{A,k} = w_{A,1} + \cdots + w_{k-1,k} \tag{25}$$

$$W_{k,B} = w_{k,k+1} + \cdots + w_{m,B} \tag{26}$$

$$S_{A,k} = \sum_{i=1}^{k-1} d_i \tag{27}$$

$$S_{k,B} = \sum_{i=k+1}^{m} d_i. \tag{28}$$

Note that $d(g_k)$ is replaced with d_{SPL}.

To satisfy the arrival time at gate A, the SFQ clock pulse is required to arrive at the splitter at time

$$\tau_{\text{SPL}|g_k} = \tau_A - W_{A,k} - S_{A,k} - d_{\text{SPL}}. \tag{29}$$

The resulting arrival time at gate B is

$$t_B = \tau_A - W_{A,k} - S_{A,k} + W_{k,B} + S_{k,B}. \tag{30}$$

If the required arrival time at B is τ_B, the resulting mismatch in the clock arrival time is

$$\Delta t = \tau_A - \tau_B - W_{A,k} + W_{k,B} - S_{A,k} + S_{k,B}. \tag{31}$$

To minimize this mismatch, the splitter placement and delay of the delay elements are adjusted to minimize $|\Delta t|$. Ideally, $\Delta t = 0$, yielding

$$\tau_A - \tau_B = W_{A,k} + S_{A,k} - W_{k,B} - S_{k,B}. \tag{32}$$

Practically, however, a tolerance level $|\Delta t| < \varepsilon$ is set by the user that allows the proxy paths to be reasonably close to the target arrival time.

The intentional delay can be varied by choosing different delays from the set of possible delays, $D = \{d_1, d_2, \ldots, d_n | d_i < d_j | 1 < i < j < n\}$. The number of delay elements on each side of the splitter is, respectively, $k-1$ and $m-k$. The total number of possible splitter locations is m, yielding a total number of delay combinations

$$N = \sum_{k=1}^{n} \binom{k+n-2}{k-1} \binom{m-k+n-1}{m-k}. \tag{33}$$

To reduce the number of iterations, note that the gate with an earlier arrival time typically does not require a delay element. By varying the delay of the elements along the paths, the target arrival time can be achieved. In addition, a splitter is placed closer to the gate with a later arrival time, creating an unnecessary delay imbalance, requiring greater area. By restricting the splitter placement to $k \leq 2$, i.e., no more than two nodes from the node with a later arrival time, the total number of combinations is reduced to

$$N = \binom{m+n-2}{m-1} + n\binom{m+n-3}{m-2}. \tag{34}$$

For $m = 10$ and $n = 5$, (34) yields 3190 delay element combinations, as opposed to 48 620 by (33).

Many proxy paths are generated for further processing. Those proxy paths exhibiting a delay imbalance within a tolerance level are sorted by the number of delay elements and total interconnect length. The path tuning algorithm processes the least expensive paths first, yielding a significant savings in area.

C. Fine Routing

During the fine routing stage, the proxy path selected in the previous section is converted into a layout. To determine a feasible placement for the interconnect, the routing is based on a Hanan grid, widely used in VLSI routing [48], and illustrated by Algorithm 3. Hanan grid $H(S)$ is the set of points produced by drawing horizontal and vertical lines through each point in S. In QuCTS, the set of points for the Hanan grid consists of clocked gates, splitters, and JTL delay elements from the proxy graph, as well as bounds on the blockages, as illustrated in Fig. 5(e). A graph $G_{H(S)}$ is based on points within $H(S)$. Two nodes in $G_{H(S)}$ are connected if the corresponding points are adjacent along any of the lines within the Hanan grid $H(S)$ and no blockage exists between the nodes. The weight of an edge is related to the propagation delay of the clock signal along the straight interconnect segment connecting the terminals of the edge.

Algorithm 3: Given Splitter Location g_k, Proxy Paths q_A and q_B, Target Arrival Times τ_A and τ_B, Wire Pitch h, and Set of Obstacles O, Embed Paths q_A and q_B Within the Layout

procedure HANAN;
Input: q_A, q_B, g_k, τ_A, τ_B, h, O
$S \leftarrow q_A \cup q_B \cup \{g_k\} \cup$ BOUNDARIES(O);
$H(S) \leftarrow$ HANAN_GRID(S);
for *each line segment* $l \in H(S)$ **do**
  if l *intersects an obstacle in* O **then**
    remove l from $H(S)$;
  end
end
try:
  $segment \subseteq E \leftarrow$ SHORTEST_PATH($G_{H(S)}, u, v$);
$success \leftarrow$ `False`;
while $success \neq$ `True` **do**
  try:
    $G_{H(S)} = (V, E) \leftarrow$ CONVERT_TO_GRAPH($H(S)$);
    $wire_A \leftarrow \varnothing$;
    for *each pair of nodes* $\{u, v\} \in q_A$ **do**
      $segment \subseteq E \leftarrow$ SHORTEST_PATH($G_{H(S)}, u, v$);
      $E \leftarrow E \setminus segment$;
      $wire_A \leftarrow wire_A \cup segment$;
    end
    $wire_B \leftarrow \varnothing$;
    for *each pair of nodes* $\{u, v\} \in q_B$ **do**
      $segment \subseteq E \leftarrow$ SHORTEST_PATH($G_{H(S)}, u, v$);
      $E \leftarrow E \setminus segment$;
      $wire_B \leftarrow wire_B \cup segment$;
    end
    $t_A \leftarrow v\,length(wire_A)$;
    $t_B \leftarrow v\,length(wire_B)$;
    $\Delta t \leftarrow (\tau_A - \tau_B) - (t_A - t_B)$;
    $O \leftarrow O \cup wire_A \cup wire_B$;
    $success \leftarrow$ `True`;
  catch *No path between u and v*:
    $S \leftarrow$ DETAILED_HANAN(S, h, O)
  end
end
return $wire_A$, $wire_B$, Δt

Algorithm 4: Given Set of Points S, Wire Pitch h, and Set of Obstacles O, Produce a More Complete Set of Points

procedure DETAILED_HANAN;
Input: S, h, O
$S^* \leftarrow \varnothing$ **for** *each point* $(x_0, y_0) \in S$ **do**
  for *each point* $s \in \{(x_0 \pm h, y_0), (x_0, y_0 \pm h)\}$ **do**
    if p *does not intersect an obstacle in* O **then**
      $S^* \leftarrow S^* \cup s$;
    end
  end
end
return $S \cup S^*$

During the routing process, previously routed wires may disconnect graph $G_{H(S)}$, isolating the target cells. To avoid this situation, the Hanan grid refinement process is applied, as shown in Algorithm 4. Four additional points, at a distance of wire pitch h, are added near each point within S, producing an extended set S^*. A Hanan grid graph $G_{H(S^*)}$ with superior connectivity is produced and the routing process is repeated.

The delay of the path generated in a Hanan grid graph is typically different from the estimate based on a proxy path.

Algorithm 5: Given Interconnect Layout *Wire*, Set of Obstacles O, and Set of Gates U, Extend the Interconnect to Increase the Delay by Δt

procedure AURA_SNAKING;
Input: *wire*, Δt
$Q \leftarrow \{\}$;
for *each segment* $\in$ *wire* **do**
  for *each point* $q \in segment$ *spaced by* d **do**
    if *segment is vertical* **then**
      $q_1 \leftarrow (x_p, y_p - d)$;
      $q_2 \leftarrow (x_p, y_p + d)$;
    else
      $q_1 \leftarrow (x_p - d, y_p)$;
      $q_2 \leftarrow (x_p + d, y_p)$;
    end
    if $q_1 \notin O$ **then**
      $Q[segment] \leftarrow Q[segment] \cup \{q_1\}$;
      $p[q_1] \leftarrow \sum_{p \in U} \frac{1}{||\vec{pq_1}||_s}$;
    end
    if $q_2 \notin O$ **then**
      $Q[segment] \leftarrow Q[segment] \cup \{q_2\}$;
      $p[q_2] \leftarrow \sum_{p \in U} \frac{1}{||\vec{pq_2}||_s}$;
    end
  end
end
while $\Delta t > \varepsilon$ **do**
  $d^* \leftarrow -\infty$;
  for *each segment* $\in$ *wire* **do**
    for *each pair of adjacent points* $q_1, q_2 \in Q[segment]$ **do**
      $d \leftarrow p[q_1] + p[q_2]$;
      if $d < d^*$ **then**
        $d^*, q_1^*, q_2^*, segment^* \leftarrow d, q_1, q_2, segment$;
      end
    end
  end
  extend $segment^*$ with q_1 and q_2; update *wire*;
  update Q;
  $\Delta t \leftarrow \Delta t - \frac{2d}{v}$;
end
return $wire_A$, $wire_B$, Δt

To adjust the delay and satisfy the arrival time requirements, the wire length is increased using wire snaking. A snaking method—aura snaking—is proposed here (see Algorithm 5) to increase the wire length, as illustrated in Fig. 6. The set of points Q within distance d from the interconnect segment separated by distance s is initially identified [see Fig. 6(b)]. Set Q is referred to as an aura of the interconnect segment. The proximity metric of point $q \in Q$ to the other gates is defined as

$$p_q \equiv \sum_{p \in U} \frac{1}{\|\vec{pq}\|_s} \tag{35}$$

where U is the set of clock sinks, $\vec{pq}$ is the vector connecting points p and q, and $\|v_{pq}\|_s$ is the s-norm of $\vec{pq}$. A point located closer to the other gates has a greater proximity metric and can create congestion. An adjacent pair of aura points with the smallest proximity metric is therefore chosen for snaking to minimize the likelihood of congestion. The aura points are evaluated for intersections with blockages, ensuring the feasibility of the wire snaking process. Once the aura points

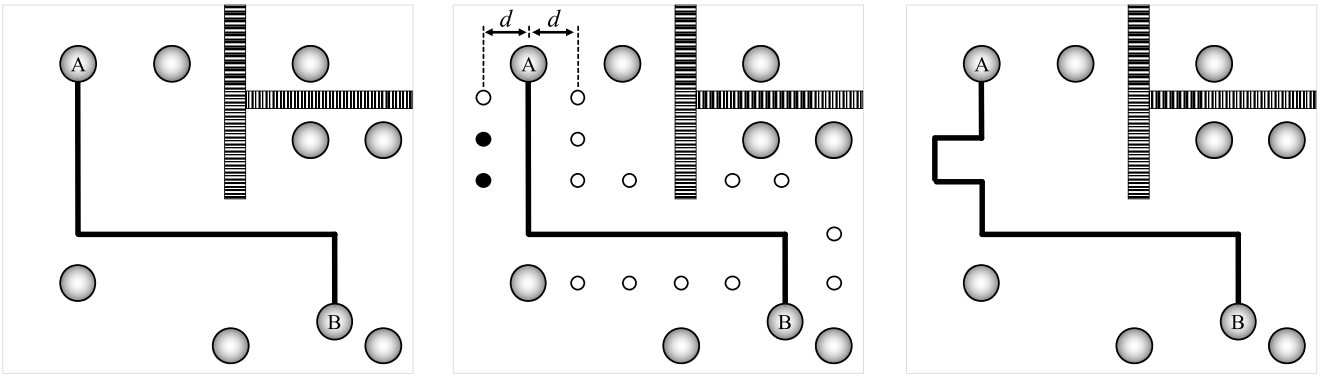


Fig. 6. Single iteration of the aura snaking process. (a) Initial wire segment surrounded by vacant cells and blockages. (b) Aura points generated within distance d from the wire. Two points near node A (filled) are selected for snaking. Note that the aura point is not generated within the blockage. (c) Final extended segment.

are selected, the wire segment adjacent to the aura points is replaced with a snaking segment, as depicted in Fig. 6(c). The interconnect is therefore extended by $2d$, increasing the wire delay by

$$\delta t = \frac{2d}{v} \tag{36}$$

where v is the speed of the RSFQ pulse propagation within a PTL.

To ensure that the aura points can be generated, spacing s should be smaller than the length l of the interconnect. The spacing however cannot be smaller than the wire pitch h due to manufacturing constraints. In the case studies, as described in Section V, the spacing is equal to the wire pitch. The minimum length of any interconnect segment is two times greater than the wire pitch, ensuring that at least three aura points can be generated. A sufficient number of aura points cannot be generated if the wire is completely surrounded by blockages. In this case, an alternative proxy path is embedded, avoiding the congested area.

Once a valid route for a pair of nodes is determined, several operations are necessary before the next pair can be processed. The splitter, delay elements, and interconnect are placed within the layout. The corresponding points in P_{AB} are removed from set P, preventing the placement of additional gates into these locations. Interconnect is added to the blockages to ensure no intersection with any subsequent wires exists. The process described in this section is repeated for each pair of nodes within the circuit, thereby determining the position of the $N-1$ splitters with N clock sinks.

V. Case Study

QuCTS is verified with the Verilog model of the AMD2901 CPU and the corresponding layout. 1050 clocked gates are distributed within a 225 mm^2 IC. The maximum and minimum delay of each gate is known. The circuit topology is represented as a Verilog netlist. The PTL driver and receiver are embedded within each gate and splitter. The dimension of each gate is 40 μm $\times$ 40 μm. Two layers of interconnect are dedicated to the clock distribution network. The vertical interconnects are placed in layer M2, and the horizontal interconnects are placed in layer M3. The gates are located in layer M5 and connected to layer M3 with vias. The interconnect pitch is 20 μm. The propagation speed of the RSFQ pulse in layers M2 and M3 is 6.25 μm/ps. The vertical connections between layers are established by the vias and produce negligible delay.

A clock skew schedule is generated for a 154 ps clock period in less than one minute. The clock network layout is generated in 52.5 min and is shown in Fig. 7. 2290 gates, 1049 splitters, and 1241 delay elements are added to the layout. The total wire length is 1027 mm, occupying an area of 5.134 mm^2. 9862 vias are placed between layers M2 and M3, and 6676 vias are placed between layers M3 and M5. The maximum difference between the required and actual arrival times is 1.6 ps.

The proposed tool has also been applied to a suite of ISCAS'89 [49] and ITC'99 [50] benchmark circuits with high gate count. The cell placement for the benchmarks is generated with Synopsys IC design compiler [51]. The results are listed in Table I. Note that the number of delay elements is linearly correlated with the number of clocked gates. For all six benchmarks, an average of 1.3 delay elements per splitter is included within the clock tree. This trend is explained by the clustering method used in QuCTS. Since the clock arrival time is considered during the routing process, gates with a similar arrival time are grouped together, producing a small delay imbalance with fewer delay elements and less wire snaking. Despite the AMD2901 being composed of fewer gates than the S13207, the total wirelength is significantly larger. This trend is explained by the more compact placement of the cells in the S13207 as compared to the AMD2901.

Since QuCTS is the first SFQ-based clock tree synthesis tool supporting nonzero skew, a direct comparison is complicated by the heterogeneity of the benchmark circuits and different target metrics. In [16], for example, a minimum clock skew tree is generated that reduces the clock skew below 4.6 ps. The splitters are initially placed arbitrarily and later embedded into the routing channels. Since the propagation delay of multiple

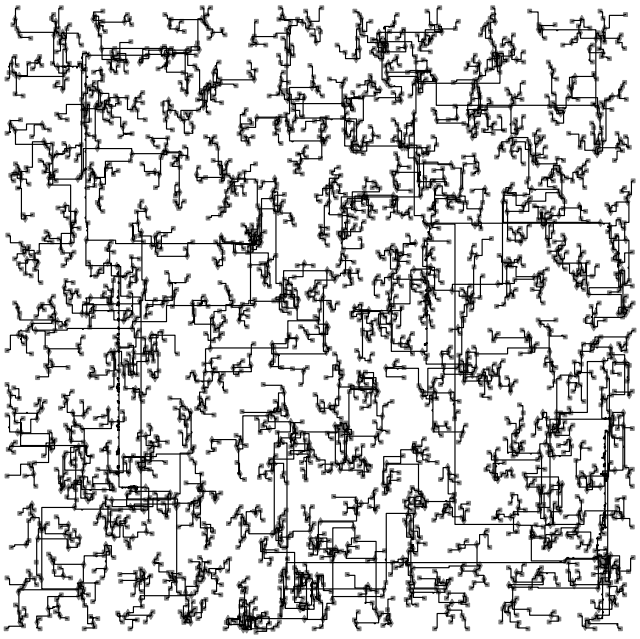


Fig. 7. Clock tree layout of AMD2901 synthesized with QuCTS.

JTLs is significantly greater than the delay of a PTL, and the minimum propagation delay is of interest in zero clock skew scheduling, no JTLs are used in [16]. The number of cells is, on average, doubled after clock tree synthesis, consistent with these case studies. No information on the algorithm runtime is provided [16]. The interconnect routing process is however transformed into two mixed integer programming problems, each completed in under 30 min. A timing uncertainty-aware clock tree is proposed in [52], where robustness to process variations is improved by maximizing the delay available for the setup and hold times. This process is analogous to clock skew scheduling, where the target skew is placed at the center of the PR [20]. The resulting total wirelength is 0.428 mm per clocked gate, consistent with the 0.419 mm per clocked gate exhibited by QuCTS.

The primary bottleneck of the algorithm in these case studies is the Hanan grid-based interconnect placement, requiring, on average, 78% of the total runtime, followed by proxy graph analysis (15%) and clock skew scheduling with 6%. Determining the clock tree topology requires negligible time in these case studies. Finding the intersection between the geometric objects is the most expensive operation, requiring more than two thirds of the time for path embedding. Based on these case studies, the path embedding process grows linearly with the number of clocked gates. This trend can be explained by Hanan grid interconnect placement which only depends on the number of gates, cells, and blockages in proximity of the target gates. Other parts of QuCTS, however, scale superlinearly. The primary bottleneck of clock skew scheduling is the optimization process. The typical complexity of the optimization algorithms ranges from $O(n^2)$ to $O(n^3)$ [53]. Although the clock skew scheduling process is not a bottleneck in these case studies, clock skew scheduling may dominate the runtime if the number of gates increases to hundreds of thousands to millions of gates. The current RSFQ technology, however, only supports several tens of thousands of junctions, making QuCTS applicable to modern RSFQ circuits and systems.

TABLE I
PERFORMANCE OF QUCTS APPLIED TO AMD2901, ITC99, AND ISCAS89 BENCHMARK CIRCUITS WITH HIGH GATE COUNT

Circuit	Clocked gates	Delay elements	Total wirelength (mm)	Runtime (minutes)
AMD2901	1,049	1,241	1,027	54
ISCAS89 S13207	1,636	2,405	272	77
ITC99 B14	6,365	5,762	905	227
ISCAS89 S38417	11,796	11,367	1,002	425
ISCAS89 S35932	14,914	15,814	6,656	394
ITC99 B18	45,710	71,090	31,736	2,485

VI. CONCLUSION

Advances in RSFQ electronics over the past decades have enabled the development of sophisticated superconductive systems. Design methodologies and related algorithms and techniques targeting the large-scale integration of RSFQ circuits are essential for managing the increasing complexity of these systems. Elevating the performance of large-scale superconductive systems requires a significant advancement in existing design capabilities, particularly the synchronous clock distribution network.

QuCTS—SFQ Clock Tree Synthesis—was described in this article. This tool is the first clock tree synthesis capability for RSFQ circuits that also utilizes useful clock skew. Using quadratic programming, the clock skew schedule is optimized for robustness to parameter variations and converted into a schedule of clock arrival times. A binary clock tree was generated by recursive clustering of the clock sinks based on the physical location and, optionally, the clock arrival times. Splitters and delay elements are placed within the layout, and the paths are tuned to satisfy the schedule of arrival times. The tool was validated using the AMD2901 4-bit microprocessor as well as ITC99 and ISCAS89 benchmark circuits. By exploring different topologies, QuCTS minimizes the number of delay elements and interconnect length. The clock arrival time schedule is precisely satisfied with wire snaking.

REFERENCES

[1] V. K. Semenov, Y. A. Polyakov, and S. K. Tolpygo, "New AC-powered SFQ digital circuits," *IEEE Trans. Appl. Supercond.*, vol. 25, no. 3, pp. 1–7, Jun. 2015.

[2] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki, "Design and demonstration of an 8-bit bit-serial RSFQ microprocessor: CORE e4," *IEEE Trans. Appl. Supercond.*, vol. 26, no. 5, pp. 1–5, Aug. 2016.

[3] K. Gaj, Q. P. Herr, V. Adler, A. Krasniewski, E. G. Friedman, and M. J Feldman, "Tools for the computer-aided design of multigigahertz superconducting digital circuits," *IEEE Trans. Appl. Supercond.*, vol. 9, no. 1, pp. 18–38, Mar. 1999.

[4] C. J. Fourie, "Digital superconducting electronics design tools—Status and roadmap," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 5, pp. 1–12, Aug. 2018.

[5] K. Gaj, E. G. Friedman, and M. J. Feldman, "Timing of multi-gigahertz rapid single flux quantum digital circuits," *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 16, no. 2, pp. 247–276, Jun. 1997.

[6] K. K. Likharev and V. K. Semenov, "RSFQ logic/memory family: A new josephson-junction technology for sub-terahertz-clock-frequency digital systems," *IEEE Trans. Appl. Supercond.*, vol. 1, no. 1, pp. 3–28, Mar. 1991.

[7] T. Jabbari, G. Krylov, S. Whiteley, E. Mlinar, J. Kawa, and E. G. Friedman, "Interconnect routing for large-scale RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 29, no. 5, Aug. 2019, Art. no. 1102805.

[8] T. Jabbari, G. Krylov, S. Whiteley, J. Kawa, and E. G. Friedman, "Repeater insertion in SFQ interconnect," *IEEE Trans. Appl. Supercond.*, vol. 30, no. 8, Dec. 2020, Art. no. 5400508.

[9] T. Jabbari, G. Krylov, J. Kawa, and E. G. Friedman, "Splitter trees in single flux quantum circuits," *IEEE Trans. Appl. Supercond.*, vol. 31, no. 5, pp. 1–6, Aug. 2021.

[10] Z. J. Deng, N. Yoshikawa, S. R. Whiteley, and T. Van Duzer, "Data-driven self-timed rsfq digital integrated circuit and system," *IEEE Trans. Appl. Supercond.*, vol. 7, no. 2, pp. 3634–3637, Jun. 1997.

[11] H. R. Gerber, C. J. Fourie, W. J. Perold, and L. C. Muller, "Design of an asynchronous microprocessor using RSFQ-AT," *IEEE Trans. Appl. Supercond.*, vol. 17, no. 2, pp. 490–493, Jun. 2007.

[12] T. V. Filippov *et al.*, "20 GHz operation of an asynchronous wave-pipelined RSFQ arithmetic-logic unit," *Phys. Procedia*, vol. 36, pp. 59–65, Jan. 2012.

[13] Y. Nobumori *et al.*, "Design and implementation of a fully asynchronous SFQ microprocessor: SCRAM2," *IEEE Trans. Appl. Supercond.*, vol. 17, no. 2, pp. 478–481, Jun. 2007.

[14] Z. J. Deng, N. Yoshikawa, J. A. Tierno, A. R. Whiteley, and T. Van Duzer, "Asynchronous circuits and systems in superconducting RSFQ digital rechnology," in *Proc. IEEE Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 1998, pp. 274–285.

[15] R. N. Tadros and P. A. Beerel, "A robust and self-adaptive clocking technique for RSFQ circuits—The architecture," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.

[16] S. N. Shahsavani and M. Pedram, "A minimum-skew clock tree synthesis algorithm for single flux quantum logic circuits," *IEEE Trans. Appl. Supercond.*, vol. 29, no. 8, pp. 1–13, Dec. 2019.

[17] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and steiner routing," *ACM Trans. Design Autom. Electron. Syst.*, vol. 3, no. 3, pp. 341–388, Jul. 1998.

[18] I. S. Kourtev, B. Taskin, and E. G. Friedman, *Timing Optimization Through Clock Skew Scheduling*, vol. 166. New York, NY, USA: Springer, 2008.

[19] J. L. Neves and E. G. Friedman, "Buffered clock tree synthesis with non-zero clock skew scheduling for increased tolerance to process parameter variations," *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 16, no. 2, pp. 149–161, Jun. 1997.

[20] J. L. Neves and E. G. Friedman, "Optimal clock skew scheduling tolerant to process variations," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 1996, pp. 623–628.

[21] E. G. Friedman, "The application of localized clock distribution design to improving the performance of retimed sequential circuits," in *Proc. IEEE Asia-Pac. Conf. Circuits Syst.*, Dec. 1992, pp. 12–17.

[22] E. G. Friedman, "Performance limitations in synchronous digital systems," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 1989.

[23] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[24] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proc. IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.

[25] I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 1999, pp. 239–243.

[26] L. Xiao *et al.*, "Local clock skew minimization using blockage-aware mixed tree-mesh clock network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2010, pp. 458–462.

[27] Y.-S. Su, W.-K. Hon, C.-C. Yang, S.-C. Chang, and Y.-J. Chang, "Value assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design*, Nov. 2009, pp. 535–538.

[28] A. L. Pankratov and B. Spagnolo, "Suppression of timing errors in short overdamped josephson junctions," *Phys. Rev. Lett.*, vol. 93, no. 17, Oct. 2004, Art. no. 177001.

[29] R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms*, Pearson Educ., 2001.

[30] C. Lin and H. Zhou, "Clock skew scheduling with delay padding for prescribed skew domains," in *Proc. IEEE Asia South Pac. Design Autom. Conf.*, Jan. 2007, pp. 541–546.

[31] J. L. Neves and E. G. Friedman, "Design methodology for synthesizing clock distribution networks exploiting nonzero localized clock skew," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 286–291, Jun. 1996.

[32] J. L. Neves and E. G. Friedman, "Topological design of clock distribution networks based on non-zero clock skew specifications," in *Proc. IEEE Midwest Symp. Circuits Syst.*, vol. 1, Aug. 1993, pp. 468–471.

[33] K. Han, A. B. Kahng, and J. Li, "Optimal generalized H-tree topology and buffering for high-performance and low-power clock distribution," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 2, pp. 478–491, Feb. 2020.

[34] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[35] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," *ACM SIGMOD Rec.*, vol. 25, no. 2, pp. 103–114, Jun. 1996.

[36] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: Which algorithms implement Ward's criterion?" *J. Classif.*, vol. 31, no. 3, pp. 274–295, 2014.

[37] A. Balatsos, "Clock buffer IC with dynamic impedance matching and skew compensation," M.S. thesis, Dept. Electr. Comput. Eng., Univ. Toronto, Toronto, ON, USA, 1998.

[38] R. Chaturvedi and J. Hu, "Buffered clock tree for high quality IC design," in *Proc. IEEE Int. Symp. Signals Circuits Syst.*, Jul. 2004, pp. 381–386.

[39] J.-L. Tsai, T.-H. Chen, and C. C.-P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 565–572, Apr. 2004.

[40] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, Mar. 1966.

[41] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: The case for A-star," *Int. J. Geogr. Inf. Sci.*, vol. 23, no. 4, pp. 531–543, Jun. 2009.

[42] M. Kou, P.-Y. Cheng, J. Zeng, T.-Y. Ho, K. Takagi, and H. Yao, "Splitter-aware multi-terminal routing with length matching constraint for RSFQ circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2251–2264, Nov. 2021.

[43] N. Kito, K. Takagi, and N. Takagi, "A fast wire-routing method and an automatic layout tool for RSFQ digital circuits considering wire-length matching," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 4, pp. 1–5, Jun. 2018.

[44] S. N. Shahsavani, T. Lin, A. Shafaei, C. J. Fourie, and M. Pedram, "An integrated row-based cell placement and interconnect synthesis tool for large SFQ logic circuits," *IEEE Trans. Appl. Supercond.*, vol. 27, no. 4, pp. 1–8, Jun. 2017.

[45] C. J. Fourie, C. L. Ayala, L. Schindler, T. Tanaka, and N. Yoshikawa, "Design and characterization of track routing architecture for RSFQ and AQFP circuits in a multilayer process," *IEEE Trans. Appl. Supercond.*, vol. 30, no. 6, pp. 1–9, Sep. 2020.

[46] R. S. Bakolo and C. J. Fourie, "Development of a RSFQ cell library for the University of Stellenbosch," in *Proc. IEEE AFRICON*, Sep. 2011, pp. 1–5.

[47] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manage. Sci.*, vol. 17, no. 11, pp. 712–716, Jun. 1971.

[48] M. Zachariasen, "A catalog of Hanan grid problems," *Networks*, vol. 38, pp. 76–83, Sep. 2001.

[49] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*,, vol. 3, May 1989, pp. 1929–1934.

[50] S. Davidson, "ITC'99 benchmark circuits—Preliminary results," in *Proc. IEEE Int. Test Conf.*, 1999, p. 1125.

[51] "Design compiler and IC compiler physical guidance technology application note," Application Note Version G-2012.06, Synopsys, Mountain View, CA, USA, Jun. 2012.

[52] S. N. Shahsavani, B. Zhang, and M. Pedram, "A timing uncertainty-aware clock tree topology generation algorithm for single flux quantum circuits," in *Proc. Design Autom. Test Europe Conf. Exhibition*, 2020, pp. 278–281.

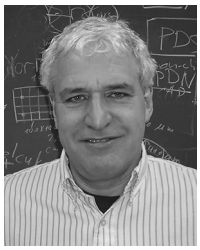
[53] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artif. Intell.*, vol. 206, pp. 79–111, Jan. 2014.

Rassul Bairamkulov (Graduate Student Member, IEEE) received the B.Eng. degree in electrical and electronic engineering from Nazarbayev University, Astana, Kazakhstan, in 2016, and the M.S. degree in electrical engineering from the University of Rochester, Rochester, NY, USA, in 2018, where he is currently pursuing the Ph.D. degree, under the supervision of Prof. E. G. Friedman.

He was an Intern with Qualcomm Technologies, Inc., San Diego, CA, USA, in 2018 and 2020. His current research interests include power delivery network design, electronic design automation, and optimization algorithms in very large scale integration.

Tahereh Jabbari (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2012 and 2015, respectively, and the M.Sc. degree in electrical engineering from the University of Rochester, Rochester, NY, USA, in 2019, where she is currently pursuing the Ph.D. degree in electrical engineering.

From 2015 to 2017, she was a Researcher of Superconducting Digital Electronics with the Superconductor Electronics Research Laboratory, Department of Electrical Engineering, Sharif University of Technology. In 2017, she joined the graduate program with the Department of Electrical and Computer Engineering, University of Rochester. Her research interests include superconducting digital electronics, electronic design automation, global signaling, synchronous and asynchronous clocking of very large scale integration SFQ circuits, security analysis of superconductive electronics, microwave behavior of superconductive striplines, noise coupling and flux trapping issues in superconductive circuits, AQFP interface circuits, SFS Josephson junctions, and inductorless superconductive circuits.



Eby G. Friedman (Fellow, IEEE) received the B.S. degree in electrical engineering from Lafayette College, Easton, PA, USA, in 1979, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Irvine, Irvine, CA, USA, in 1981 and 1989, respectively.

He was with Hughes Aircraft Company, Glendale, CA, USA, from 1979 to 1991, rising to a Manager of the Signal Processing Design and Test Department, where he was responsible for the design and test of high-performance digital and analog ICs. He has been with the Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY, USA, since 1991, where he is a Distinguished Professor and the Director of the High Performance VLSI/IC Design and Analysis Laboratory. He is also a Visiting Professor with the Technion–Israel Institute of Technology, Haifa, Israel. He has authored over 500 articles and book chapters and authored or edited 19 books in the fields of high-speed and low-power CMOS design techniques, 3-D design methodologies, high-speed interconnect, superconductive circuits, and the theory and application of synchronous clock and power distribution networks, and he holds 23 patents. His current research and teaching interests include high-performance synchronous digital and mixed-signal circuit design and analysis with application to high-speed portable processors, low-power wireless communications, and server farms.

Dr. Friedman was a recipient of the IEEE Circuits and Systems Mac Van Valkenburg Award, the IEEE Circuits and Systems Charles A. Desoer Technical Achievement Award, the University of Rochester Graduate Teaching Award, and the College of Engineering Teaching Excellence Award. He was the Editor-in-Chief and the Chair of the Steering Committee of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS and the *Microelectronics Journal*, a Regional Editor of the *Journal of Circuits, Systems and Computers*, an editorial board member of numerous journals, and a program and technical chair of several IEEE conferences. He is a Senior Fulbright Fellow, a National Sun Yat-sen University Honorary Chair Professor, and an Inaugural Member of the UC Irvine Engineering Hall of Fame.