Postrouting Optimization of the Working Clock Frequency of Single-Flux-Quantum Circuits

Ting-Ru Lin, Bo Zhang, and Massoud Pedram

***Abstract*—With the emergence of large-scale single-flux-quantum (SFQ) circuits, it is desirable to offload heavy routing tasks to automated electronic design automation (EDA) tools. The unique characteristic of SFQ circuits is that large clock distribution networks are generally required to pass the clock signal to clocked cells subject to limited fanout drive capability of the intervening clock buffers. This scenario results in a huge challenge to control the clock skew and minimize the signal path delays in an SFQ circuit during the routing step. Values of the clock skew and path delays determine not only the maximum working frequencies but also whether or not there are hold time violations. However, it is not necessary to develop a timing-driven routing tool specifically for SFQ circuits from scratch because the number of the paths that set the maximum clock frequencies or give rise to hold time violations is generally very small. Thus, we present a novel postrouting optimization framework that augments standard maze routing tools by adding the capability of reducing path lengths to maximize the working frequency of the chip and meandering paths to avoid hold time violations. A framework is developed in which machine learning is applied to analyze wire distributions and a maze routing algorithm is used to reroute targeted paths. Based on the MIT-LL SFQ5ee process technology, we demonstrate that our framework can improve the minimum working frequency by 7% on average over the state-of-the-art EDA routing tool for a suite of 14 SFQ circuits while fixing all hold time violations.**

***Index Terms*—Electronic design automation (EDA), machine learning, routing, single flux quantum (SFQ), superconducting integrated circuits.**

I. Introduction

RESEARCH on superconductive electronics is flourishing because of government and commercial interest in the technology [1]–[3]. The goal is to build ultra energy-efficient and high-speed computational systems using superconductive single-flux-quantum (SFQ) technology and its variants [4]. Active elements of SFQ designs are Josephson junctions (JJs), which propagate SFQ pulses through a logic cell in the order of $\sim$1 ps and dissipate only $\sim 10^{-19}$ J or lower per JJ switching action [5], [6]. Rapid SFQ (RSFQ) designs are known for their high working frequencies but require a bias current for JJs, which contributes to large static power dissipation. Energy-efficient RSFQ (ERSFQ) [7]–[10] and efficient SFQ [11], [12] have been proposed to eliminate the static power dissipation of RSFQ designs through the implementations of alternative biasing schemes. As a result, the static power dissipation of an ERSFQ cell is almost zero. The dynamic energy dissipation of a static minimum-size inverter in Global Foundries 14-nm FinFET process operating at 0.7-V supply and driving a fanout load of four is approximately 1 fJ (1×10^{-15}), whereas the energy dissipation of an RSFQ inverter is only 2 aJ (2×10^{-18}), which is 500x lower. In the limit of CMOS scaling, this energy efficiency advantage still stays at 200X or higher [7].

SFQ fabrication technologies have evolved for over two decades and the number of the JJs in an SFQ die has increased from 1000 to more than 800 000 [13], [14]. The advancement of SFQ fabrication technologies has thus enabled large-scale SFQ circuits. However, it was not until recently that powerful and RSFQ-specific electronic design automation (EDA) tools were developed to facilitate the design of large-scale SFQ circuits [15]. It is still under study to enhance these EDA tools with specialized timing optimization strategies [13], [16] because, unlike CMOS cells, most SFQ cells, including combinational and synchronous cells, are clocked cells with a clock input. Routing tools determine the exact values of data signal path delays and clock skew. Advanced routing tools are expected to optimize the routing process for critical paths that limit the maximum working frequency based on data signal path delays and clock skew. Another challenge for advanced routing tools is that of resolving hold time violations, which are prominent sources of timing failure in RSFQ circuits.

We present a postrouting optimization framework that reduces the delay of the critical paths of large-scale SFQ circuits while controlling the clock skew. Furthermore, hold time violations are resolved by meandering routing paths. The whole optimization framework consists of machine learning, critical path optimization, and path rectification. Machine learning observes local wire deployments and conducts a full-chip wire distribution analysis. Equipped with the distribution knowledge, critical paths are identified and then rerouted with alternative short wires during critical path optimization. Finally, path rectification refines clock skew for frequency maximization and builds detour paths for hold time violations. Given 14 SFQ circuits routed by a state-of-the-art routing tool, the proposed router not only improved the maximum working frequency by 7% on average but also resolved all hold time violations in 180 s for all circuits.

Manuscript received December 18, 2019; revised May 25, 2020; accepted May 29, 2020. Date of publication June 29, 2020; date of current version July 25, 2020. The work was supported by the Office of the Director of National Intelligence, Intelligence Advanced Research Projects Activity, via the U.S. Army Research Office under Grant W911NF-17-1-0120. This article was recommended by Associate Editor I. V. Vernik. *(Corresponding author: Ting-Ru Lin.)*

The authors are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90007 USA (e-mail: tingruli@usc.edu; zhan254@usc.edu; pedram@usc.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASC.2020.3005584

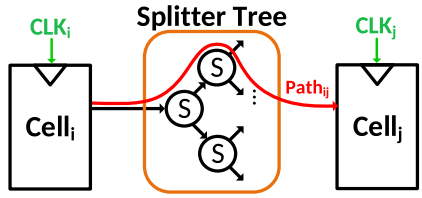

Fig. 1. Cell$_i$, Cell$_j$, and Cell$_k$ are SFQ logic cells with a clock input, such as AND, OR, or DFF. S stands for a splitter.

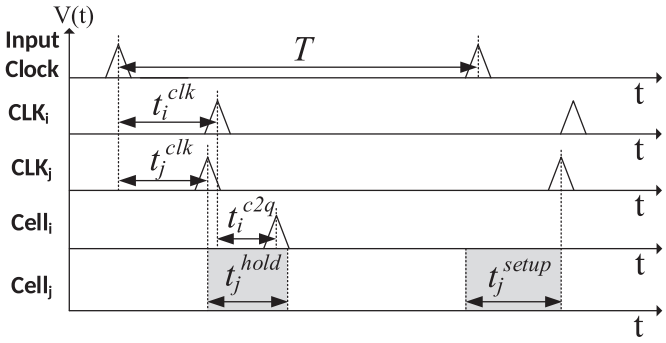

Fig. 2. Timing diagram of two connected clocked cells, denoted by Cell$_i$ and Cell$_j$.

The remainder of this article is organized as follows. Section II provides background on SFQ timing concepts. Section III describes our motivation based on the progress of routing tools for SFQ circuits. Section IV elaborates our proposal for postrouting optimization. Section V specifies the standard SFQ cell library for framework evaluation. Section VI provides our experiment results and discussions. Finally, Section VII concludes this article.

II. Background

Timing behaviors of SFQ circuits are principally governed by the same rules and constraints as CMOS circuits [17]. Successful data exchanges between a pair of connected clocked SFQ cells must satisfy two conditions: a setup time condition and a hold time condition. Figs. 1 and 2 are used to explain and derive the conditions. In Fig. 1, a pair of clocked cells are connected by a data signal path and clock signals are forwarded from a global clock input to these connected cell. Fig. 2 illustrates a timing diagram to represent the SFQ pulses in Fig. 1 in the time domain. *Note that an interconnection between two cells in an SFQ circuit can consist of multiple Josephson transmission lines (JTLs)/passive transmission lines (PTLs) and splitters for fanout requirements.* Since the splitters in SFQ circuits are added for fanout requirements, they are clockless cells by default.

Referring to Fig. 2, a clock period T denotes the time interval between two consecutive clock pulses and the interval is measured between the peak of two pulses. We define clock skew as the time difference of receiving the clock signal between two cells. The clock skew $\text{skew}_{i,j}$ between a clocked cell i and a clocked cell j in Fig. 1 is expressed as

$$\text{skew}_{i,j} = t_j^{\text{clk}} - t_i^{\text{clk}} \tag{1}$$

where t_i^{clk} (t_j^{clk}) is the arrival time of the clock signal at the clock input of a cell i (j) from the primary clock input. A data signal path delay denotes the pulse propagation time from the first clocked cell of a path to the last clocked cell and it includes the clock-to-q delay of the first cell and the cell delays of the splitters along the path. We term the first cell as the launch cell and the last cell as the capture cell. In Fig. 1, the data signal path delay $\Delta_{i,j}$ of forwarding a signal from a cell i to a cell j through $N_{i,j}$ splitters is given by

$$\Delta_{i,j} = t_i^{c2q} + N_{i,j} \times t_{\text{splitter}} + \sum_{k=1}^{N_{i,j}+1} t_k^{\text{wire}} \tag{2}$$

where t_i^{c2q} is the clock-to-q delay of a cell i; t_{splitter} is the cell delay of a clockless splitter; and t_k^{wire} is the wire propagation delay of the kth JTL/PTL of the path.

With (1) and (2), we can specify setup and hold timing conditions [17], [18]

$$\text{Setup:} \quad -\text{skew}_{i,j} + \Delta_{i,j} + t_j^{\text{setup}} \le T \tag{3}$$

$$\text{Hold:} \quad \text{hold}_j \le -\text{skew}_{i,j} + \Delta_{i,j} \tag{4}$$

where t_j^{setup} and hold_j are the setup time and the hold time of a cell j, respectively. These two conditions apply to any data signal paths with cells i and j as the launch cell and the capture cell of each path. The first condition ensures no signal loss by capture cells during the circuit operation, whereas the second condition ensures signal integrity of output pulses from launch cells. Referring to Fig. 2, the SFQ pulse from Cell$_i$ in the fourth row should arrive ahead the second gray time interval of Cell$_j$ in the fifth row. As we can see, the characteristics of clocked cells and clockless cells in SFQ circuits revise the details of both conditions, which encourages the development of new timing optimization strategies rather than the direct utilization of conventional CMOS strategies.

The maximum working frequency of an SFQ circuit is the inverse of the minimum clock period T_{min}, which is given by

$$T_{\text{min}} = \max_{i,j} \quad -\text{skew}_{i,j} + \Delta_{i,j} + t_j^{\text{setup}}. \tag{5}$$

This equation indicates that the maximum working frequency is limited by the paths with the worst setup time condition. These paths are known as critical paths, including critical data signal paths related to data signal path delays and launch clock signal paths related to clock skew. A net refers to a direct connection between two SFQ cells and there can be multiple nets included in a path due to clockless splitters, as shown in Fig. 1. A physical interconnection of a net in an SFQ circuit is built with a physical wire.

The hold time condition specified in (4) prevents signal race errors during circuit operation. A signal race error happens when the pulse fired from the launch cell undermines the pulse from the capture cell within one clock period. Referring to Fig. 2, a signal race error arises when the arrival time of the SFQ pulse from Cell$_i$ in the fourth row is within the first gray time interval of Cell$_j$ in the fifth row. To quantify the hold time violations, we

define hold slack as

$$\text{Slack}_{i,j}^{\text{hold}} = -\text{skew}_{i,j} + \Delta_{i,j} - \text{hold}_j. \quad (6)$$

If hold slack is no less than zero, the race error will not happen. Otherwise, the signal integrity of this path may be severely compromised during the operation. Timing margins can be added to the right-hand side of both (5) and (6) if more factors, such as process variations [19], are considered.

III. Motivation

The high working frequency of SFQ circuits is one reason for the large interest in SFQ circuits as the building blocks of superconductive supercomputers. The working frequency, however, has been decreasing with the scaling of SFQ circuits because of the need to sacrifice clock speed in order to cope with manufacturing process-induced sources of variability [19]. Even worse, the propagation delays along long wires that may be created by routing tools can significantly reduce the peak working frequencies (note that SFQ cell delays are typically around 10 ps (≤ 14 ps in our cell library), which is the same delay as sending an SFQ pulse on a PTL of length 1 mm (assuming an SFQ pulse propagation speed of 100 μm/ps).

To the best of our knowledge, the development of many SFQ routing tools [20], [21] follow that of CMOS routing tools, which search and place wires for nets one at a time based on complex cost functions. In [21], a router based on A* algorithm sequentially routes SFQ circuits with hundreds of nets, whereas qGDR proposed in [20] completes routing large-scale circuit with thousands of nets sequentially through a maze routing algorithm. The drawback of both routing tools is the weak guarantee that valuable routing resources are reserved for critical timing nets. Indeed, if the routing resource allocation is not done carefully, noncritical nets may use up all the shortest routing paths, leaving critical nets with scarce routing resources. In [22], the routing tool realizes a consistent delay of all paths in the same stage by exploiting integer programming solvers. However, the feasible solution for a consistent delay for each stage is unlikely to result in the minimum delay for all stages because again no priority is given to timing critical paths and nets. Moreover, the scalability problem occurs when we apply the routing tool proposed in [22] to large-scale SFQ circuits. As a result, none of the aforementioned routing tools can utilize the available SFQ timing information in order to maximize the chip working frequencies and resolve any hold time violations.

To improve the routing results generated by state-of-the-art SFQ routing tools, we present in this article a postrouting optimization framework, which increases the maximum operating frequencies and resolves all hold time violations in large-scale SFQ circuits. Our framework explores alternative short wires for both critical data signal paths and launch clock signal paths in order to improve the working frequency of a target chip. Next, the postrouting optimization framework targets paths with hold time violations and generates wire meanderings by using a maze routing algorithm. Our main contributions may be summarized as follows.

1) Machine learning that efficiently evaluates arbitrary wire distributions of diverse SFQ circuits.
2) Routing algorithms that account for wire distribution while routing critical paths.
3) Detour path creations that increase path delays to resolve hold time violations.
4) Results of improvements over extensive SFQ routing results generated by state-of-the-art routing tools.

Notice that our postrouting optimization must satisfy all design rules since it is the last stage of the physical design process.

IV. Postrouting Optimization

We present a postrouting optimization framework consisting of three steps: machine learning, critical path optimization, and path rectification. The proposed framework is depicted in Fig. 3. We will provide details about each step in the next section. The advantage of our proposed postrouting framework is to efficiently improve the working frequency of an SFQ circuit while maintaining most of the already deployed routing wires.

The objective of the machine learning step is to analyze the distributions of the irregular routing wires in routing regions. Given the analysis results, a subsequent ripup-and-reroute process chooses nets that have a low likelihood of dramatically changing existing wire distributions to be ripped up. We thus overcome the shortcoming of sequential routing procedures, which are typically strongly net ordering-dependent.

The critical path optimization step aims at reducing the delay of critical data signal paths of a given SFQ circuit in order to minimize the operating clock period. In this step, we identify critical data signal and clock signal paths by using a graph search algorithms and apply a ripup-and-reroute process to the identified paths. To achieve the best ripup-and-reroute result, we utilize an intelligent maze routing tool, which can reallocate alternative short wires to the critical nets while evaluating the risk of causing a ripup-and-reroute cycle based on the machine learning results. The run time of the whole framework is therefore kept rather low.

There are two goals for the path rectification. The first goal is to refine the launch clock signal paths for clock skew adjustment and the other is to minimize the number of the hold timing violations. To start with, clock nets in the clock distribution topology are targeted to appropriately set the clock skew of the critical data signal paths through the ripup-and-reroute process. To cope with the hold timing violations of the SFQ circuit, data signal paths with hold time violations are identified and are replaced with detour wires with large delays.

Next, we provide details about optimization procedures used in each of the aforesaid steps. Our explanations focus on two-pin nets but our ideas can be extend to multipin nets through focusing on the longest interconnection of each multipin net. To simplify the explanations, we assume there are only two available metal layers for routing, which applies to general SFQ technologies.

A. Machine Learning

A significant reallocation of routing resources is likely to occur when rerouting timing critical nets in a design. The change in allocated routing resources should, however, be kept to a

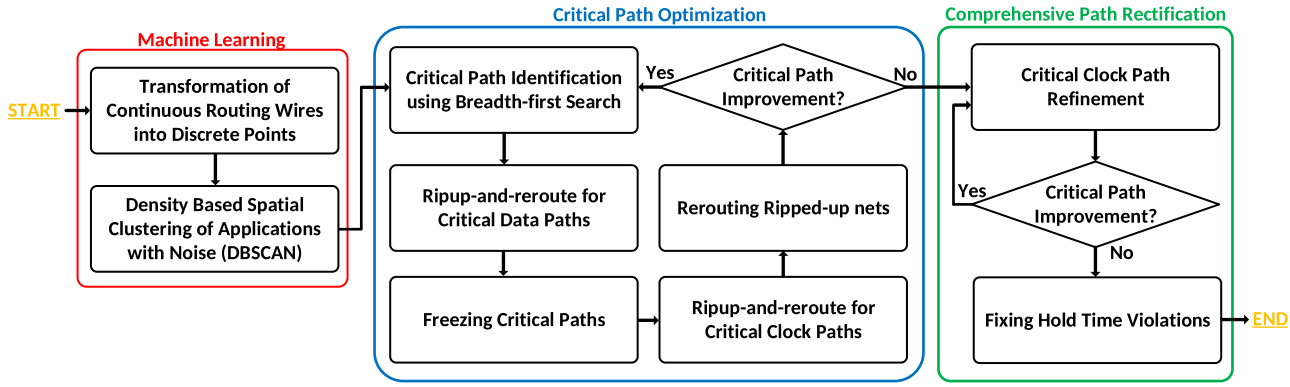

Fig. 3. Workflows of the postrouting optimization.

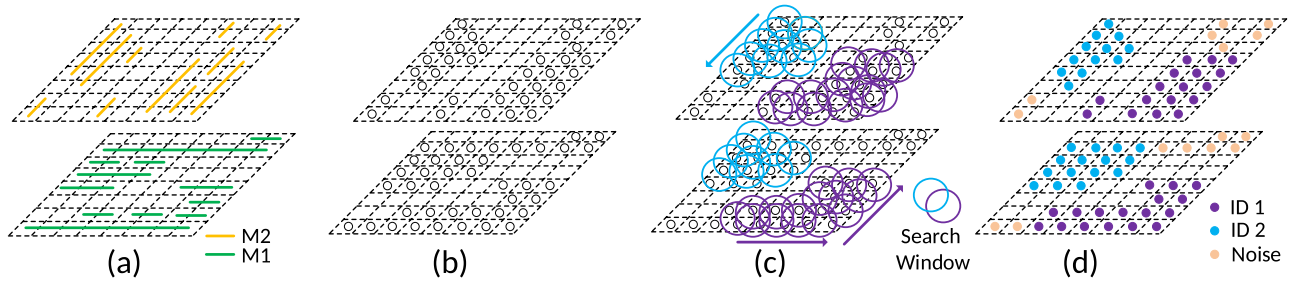

Fig. 4. DBSCAN algorithm in the machine learning stage. (a) Routing area partition. (b) Routing wire transformation. (c) Density-based search. (d) Grouping result.

minimum because it may create convergence issues and undo optimizations that had done previously, e.g., to minimize the total wire lengths or the total via count [20]. To evaluate the degree of potential resource reallocation, we utilize a well-known machine learning method, called *density-based spatial clustering of applications with noise* (DBSCAN) to analyze the wire distributions [3]. DBSCAN is an unsupervised learning method, which groups distributed points within a space based on the local point densities, as shown in Fig. 4. More precisely, given a set of points in some space, DBSCAN groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (those points in which nearest neighbors are far away).

1) Transformation of Continuous Routing Wires: The input of DBSCAN is discrete set of data points. Therefore, we transform routing wires of an SFQ circuit (which comprise contiguous wire segments) into a set of discrete points as follows. To begin with, the whole routing area is cut into bins, as shown in Fig 4(a). The width and the length of a bin are set to the wire pitch size in the x- and y-direction, respectively. Next, contiguous wire segments going over these bins are transformed into a set of discrete points by assigning an unlabeled point to the center of the bin, which is under the wire segment. White empty circles in Fig. 4(b) denote these points.

2) Density-Based Spatial Clustering of Applications With Noise: The discrete point representation empowers DBSCAN to perform wire distribution analysis based on local point densities without tracing irregular continuous wires. Distributed unlabeled points are grouped by DBSCAN. There are two required parameters for grouping points: radius of a search window (R) and a grouping threshold (*MinPts*).

Algorithm 1 describes details of DBSCAN [23]. There are four main steps that are repeated until all points are labeled, which are as follows.

1) Select any unlabeled point as the center point of a search window of radius R and count the number of points enclosed within the search window (including the initial unlabeled point).
2) If the aforesaid count is equal to or higher than *MinPts*, label the selected point with a cluster ID (*ID*) and group all enclosed points within the search window as a *seed set*. Otherwise, label the selected point as *NOISE* and go back to Step 1.
3) Count the enclosed points within the search window centered at every point in the seed set. If there are at least *MinPts* such enclosed points, then the chosen point will be labeled with the same cluster ID as in Step 2 and other enclosed points without a label are added to the seed set of Step 2.
4) Repeat Step 3 until all points in the seed set are chosen [this loop is illustrated in Fig. 4(c)]. Next, increment *ID* by 1 and go back to Step 1 if any points are unlabeled.

Fig. 4(d) illustrates a grouping result in which all points are labeled with $ID = 1$, $ID = 2$, or *NOISE*. The grouping result can be used to identify congested (layout) regions in the layout because regions occupied by points with the same *ID* tend to experience higher routing resource pressure than other regions. The time complexity of DBSCAN is $O(N * \log(N))$, where N is the number of points in the complete layout space.

Algorithm 1: DBSCAN.

Input: Points, R, $MinPts$
$ID = 1$
for p in Points **do**
  if p is unlabeled **then**
    $enclosedPts =$ RangeQuery(Points, p, R)
    if $|enclosedPts| < MinPts$ **then**
      label p as *NOISE*
      continue
    end if
    label p with ID
    $seeds = enclosedPts \backslash \{p\}$ **do**
    for p_s in $seeds$
      $enclosedPts =$ RangeQuery(Points, p_s, R)
      if $|enclosedPts| \geq MinPts$ **then**
        label p_s with ID
        for p_n in $enclosedPts$ **do**
          if p_n is unlabeled **then**
            $seeds = seeds \bigcup \{p_n\}$
            label p_n with ID
          else if p_n is *NOISE* **then**
            label p_n with ID
          end if
        end for
      end if
    end for
    $ID = ID + 1$
  end if
end for

B. Critical Path Optimization

The objective of critical path optimization is to reduce the wire lengths of critical nets without violating any design rules. The wire length reduction is expected to be realized without having to rip up many noncritical nets. This section describes the proposed critical path optimization following Fig. 3.

1) Critical Path Identification Considering Data and Clock Propagation Delays: Breadth-first search (BFS) is a graph search algorithm that explores outward from a source node in all possible directions, adding nodes one layer at a time (a node is visited only if all of its input nodes have been visited) [24]. In this article, the nodes refer to the primary inputs, clocked cells, clockless cells, and primary outputs of a given SFQ circuit. The BFS produces a BFS tree with a source node as the root to connect other nodes reachable from the source node. The source node in a BFS tree is a primary input and the leaf nodes should be primary outputs unless there is a cell with no output connection. Since there are multiple primary inputs in the SFQ circuits, we create a dummy source node that connects directly to the primary inputs of the SFQ circuit via direct edges. Similarly, we create a dummy sink node that receives direct inputs from all primary outputs of the SFQ circuit. In this way, we run BFS from the dummy source and continue exploring the circuit graph until the dummy sink is reached. Next, by examining the resulting BFS tree, we identify all data signal paths—note that in SFQ circuits, a data signal path is one that goes from a primary input to a clocked logic cell, or from a clocked logic cell to another clocked logic cell, or from a clocked logic cell to a primary output without going through any intervening clocked logic cells. We call the set of all such paths, the *data signal path set*. Based on the found data signal paths as well as SFQ cell and PTL delay information, we can easily calculate the maximum propagation delay for a target data signal path comprising a clock cell driver and a clocked cell receiver (possibly going through multiple clockless cells and PTL connections).

We can similarly do a BFS on the clock distribution tree, where the clock input to the SFQ circuit acts as the source node and all clocked cells in the circuit act as the clock sinks. Again we create a dummy clock sink node that receives direct inputs from all clocked cells in the circuit. The BFS starts from the clock source and continues until the dummy clock sink is visited. Note that this stopping criterion is based on the assumption that the clock signal is totally absorbed by a sink clocked cell [6]. To our knowledge, this assumption is true for standard SFQ designs. Note that a clock signal path refers to the complete clock propagation path starting from the source node (the primary clock input) to the leaf node (the reached clocked cell), possibly going through splitters and buffers as well as PTL connections. Based on the found clock signal paths as well as SFQ cell and PTL delay information, we calculate the clock skew for a target data signal path comprising a clock cell driver and a clocked cell receiver. Now then, from (5), we find the critical data and clock signal path combination that set the minimum clock period of the SFQ circuit. The following procedures of the critical path optimization is to reduce the path delay of the critical data and clock signal paths by a ripup-and-reroute process.

2) Ripup-and-Reroute for Critical Data Signal Paths: The primary target of the critical path optimization is the critical data signal path in an SFQ circuit because, in general, the magnitude of the clock skew of two launch clock signal paths is much smaller than that of the path delay of the critical data signal path. As discussed in previous sections, the critical data signal path can be formed either by a very long wire or multiple wires with clockless splitters in between. The number of the critical data nets of the critical data signal path is one in the first case and is more than one in the latter case.

The objective of the ripup-and-reroute process is to efficiently reduce the wire lengths of the critical data nets in a large-scale SFQ circuit without ripping up many noncritical nets. Notice that we rip up and reroute the critical data net one by one instead of ripping up all critical data nets at the same time so as to avoid significant layout changes. In the rerouting process, a maze routing algorithm is utilized to find an alternative short wire for each ripped-up critical net [20]. There are three main steps in the maze routing algorithm: bounding box formation, wave propagation, and trace back. We explain these steps on a 7 by 7 bin array, as depicted in Fig. 5, in which the maze routing algorithm creates a wire connection for a net with two pins labeled by b.

To begin with, a search space is specified by forming a minimal rectangular bounding box that fully encompasses the

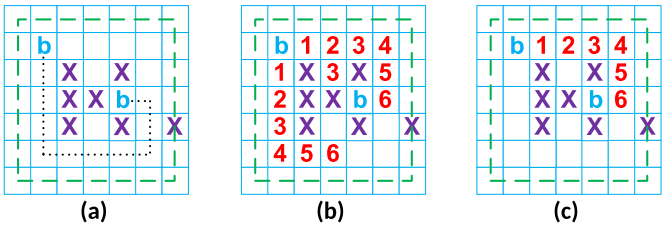

Fig. 5. Maze routing algorithm. (a) Bounding box formation. The dotted line represents the previous routing wire. b: source vertex. X: blockage. (b) Fanout wave propagation. (c) Trace back.

previous routing wire of a target (data) net, as shown in Fig. 5(a). This bounding box guarantees that an alternative wire for this net is either the original routing wire with the same wire length or preferably a shorter wire. Next, the maze routing algorithm starts searching for a solution with a minimum routing cost. Routing costs of creating a wire are calculated from the bin where the source pin of the net is located to other reachable bins using a wave (frontier) propagation method. Note that the routing cost calculation is skipped for bins, which are occupied by blockages.

The cost of a wire that reaches a bin b_j^{l1} in the $l1$th layer starting from the source bin of net n_i is

$$C(b_j^{l1}, n_i) = \min_{b_k^{l2}} \quad f(b_j^{l1}, b_k^{l2}) + g(b_j^{l1}, n_i) + C(b_k^{l2}, n_i) \quad (7)$$

where b_k^{l2} is the adjacent bin of b_j^{l1}. Values in the grid cells in Fig. 5(b) represent the routing cost. The value of $f(b_j^{l1}, b_k^{l2})$ depends on three parameters [20]: *SegCost*, *JogCost*, and *ViaCost*. The initial value of $f(b_j^{l1}, b_k^{l2})$ is 0. *SegCost* is added to $f(b_j^{l1}, b_k^{l2})$ if $l1 = l2$ and b_j^{l1} is aligned with b_k^{l2} following the preferred direction of the $l1$th routing layer. *JogCost* is added to $f(b_j^{l1}, b_k^{l2})$ if $l1 = l2$ and b_j^{l1} is not aligned with b_k^{l2} following the preferred direction of the $l1$th routing layer. *ViaCost* is added to $f(b_j^{l1}, b_k^{l2})$ if $l1$ is not equal to $l2$. $g(b_j^{l1}, n_i)$ is zero if b_j^{l1} is not occupied by a routing wire. Otherwise, $g(b_j^{l1}, n_i)$ is given by

$$g(b_j^{l1}, n_i) = c_1 * \text{IDCount}(b_j^{l1}) + c_2 * \frac{\text{WL}(b_j^{l1})}{\text{MaxWL}} + c_3 * \frac{1}{1 + \exp(1 - \text{PreWL}(n_i)/\text{MaxWL})} \quad (8)$$

where c_1, c_2, and c_3 are weight coefficients. $\text{IDCount}(b_j^{l1})$ returns the number of bins that are labeled with the same cluster ID as b_j^{l1} and the return is zero if b_j^{l1} is labeled with *NOISE*. For example, if b_j^{l1} is occupied by a blue point marked with ID2 in Fig. 4(d), $\text{IDCount}(b_j^{l1})$ returns 27 for cost calculation. $\text{WL}(b_j^{l1})$ is the length of the wire that passes b_j^{l1} and MaxWL is the maximal wire length of all nets in Manhattan distance. $\text{PreWL}(n_i)$ is the previous physical wire length of the given net n_i before the rip-up.

We provide more details about (8). The first term in the equation helps our router predict the number of bins that may be affected in the worst case if the net in b_j^{l1} is ripped up because $\text{IDCount}(b_j^{l1})$ returns the number of bins sharing the same cluster ID as b_j^{l1}. Specifically, rerouting nets using wires that pass through congested regions is challenging because the routing algorithm must create alternative wires for these nets using scarce routing resources. The price of allowing such a rerouting is the potentially significant change of the routing layout compared to the initial routing layout. To minimize the probability of a massive routing resource reallocation, we make $\text{IDCount}(*)$ the dominant factor in (8). $\text{WL}(b_j^{l1})$ in (8) is divided by MaxWL for normalization because $\text{WL}(b_j^{l1})$ of long wires is generally larger than that of short wires. If we do not divide $\text{WL}(b_j^{l1})$ by MaxWL, the value of $g(b_j^{l1}, n_i)$ for the bin occupied by a long wire could be much larger than that for the bin occupied by a short wire. The third term in (8) adds a constant cost of utilizing an occupied bin for rerouting n_i. The constant cost for rerouting a large net is bigger than that of rerouting a small net because we want to limit the bin count for net rerouting. The cost is not (linearly) proportional to the previous physical wire length of the net n_i because we want to play down the previous routing result.

Finally, the wire connecting pins of n_i and having the minimum routing cost is identified by performing trace back from the terminal pin to the source pin in a backward propagation manner, as shown in Fig. 5(c). To control the number of the nets to be ripped up, the routing cost of creating a wire for n_i cannot be larger than an upper bound, which is given by

$$\text{Ubound}(n_i) = c_4 * \text{PreWL}(n_i) * \text{SegCost} \quad (9)$$

where c_4 is another weight coefficient. As an example if $\text{Ubound}(n_i)$ of the shown net in Fig. 5(c) is 6 or higher, we will accept the traced wire; otherwise, we will reject it. The accepted wire is committed to be a physical routing wire for n_i and the nets that overlap (short) this committed wire are ripped up and rerouted in a subsequent ripup-and-reroute process for noncritical nets.

3) Freezing Rerouted Critical Paths: Alternative wires for the rerouted critical nets should not be modified during the ripup-and-reroute process for noncritical nets. A straightforward strategy to hinder invalidating the wire length reduction results is to add a huge penalty to the routing cost for ripping up any critical nets. However, a huge penalty can still be an acceptable penalty when a large upper bound is set for exploring feasible wires in a large-scale SFQ circuit. Thus, we propose a rigorous control strategy. The wires of the rerouted critical data nets are temporarily frozen as blockages (we call the condition of the wires as frozen because it is not a permanent condition). The frozen wires can be thawed under some conditions. We first describe how the wires are frozen and then explain when the frozen wires can be thawed.

We realize freezing the wires of the rerouted critical nets by placing temporary blockages along these wires. Take Fig. 5(c) as an example. If the bins fond by trace back step are utilized to create a wiring solution for a critical data net, these bins are marked with "X" (denoting fixed blockages in our routing framework). As a result, the ripup-and-reroute process for noncritical nets cannot touch the wires of any rerouted critical data nets. A side benefit of this freeze strategy is that the proposed framework avoids potential convergence problems due to repetitive ripup-and-reroute processes.

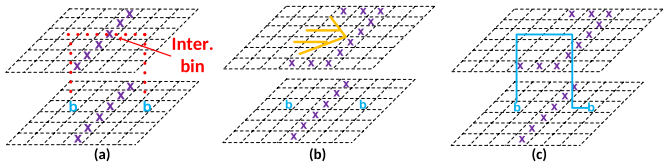

Fig. 6. Plowing operations. (a) An exploration space with barricades. Inter. bin: intersection bin. (b) Barricade plowing. (c) Routing wire connection.

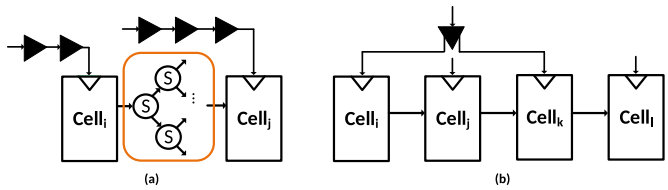

Fig. 7. Interconnections between clocked cells. (a) Interconnection between two clocked cells. The black triangles represent clockless cells for forwarding clock signals. (b) Three interconnections between pairs of clocked cells.

A lane of temporary blockages can be viewed as a barricade that, unfortunately, can hinder the routing algorithm from finding a wiring solution for a target net. Fig. 6(a) depicts the effective exploration space for a net with two bins but there are two parallel barricades in the middle of the space for both bottom and top routing layers. Given this scenario, the routing algorithm cannot create a wire connecting pins of the target net without going through the barricades. When this special condition (or other similar conditions) is observed, we thaw a small segment of the frozen wire and then move the thawed segment. Similar to prior work [25], we call the movement a plowing operation that sweeps the frozen wire segment while maintaining the wire connection. The special condition is not limited to the scenario in which the top barricade is just placed above the bottom one. Any parallel barricades that hinder the rerouting process are recognized as the special condition. We provide more details below.

The plowing operation is called when the exploration space for rerouting a net is partitioned by parallel barricades. To begin with, a L-shape wire for this net is deployed temporarily in the top routing layer and this L-shaped wire will intersect the bin(s) that are occupied by the frozen wire. The bin shared by both the L-shaped and frozen wire is called an *intersected bin*, as shown in Fig. 6(a). The plowing operation thaws a short wire segment centered at the intersected bin and then moves the thawed wire segment to the direction, which is orthogonal to the barricades. The wire segment in the top layer is considered first because the wire density of the top layer is usually less than that of the bottom layer [20]. The plowing operation is applied to the wire segment in the bottom layer when there is no free space to move the segment located in the top layer. Notice that the plowing operation will be called several times if there are multiple parallel barricades, which partition the whole exploration space into several parts.

4) Ripup-and-Reroute for Launch Clock Signal Paths: The proposed critical path optimization reduces the path delay of the critical data signal paths but it may also increase the clock skew ($\text{skew}_{i,j} = t_j^{\text{clk}} - t_i^{\text{clk}}$) of the launch clock signal paths. There are two possible ways to increase the clock skew: first, increase the arrival time of the clock signal at the input of the capture cell and second, decrease the arrival time of the clock signal at the input of the launch cell. If we are only allowed to control the wire length of SFQ circuits, the first method requires increasing the wire length of the critical clock nets, which is counter-intuitive and generally undesirable. Thus, the second method is pursued in this article. We locate the critical clock nets for wire length reduction by the assumption that the clock distribution topologies in large-scale SFQ circuits are built with clockless cells (e.g., splitters and buffers). Applicable clock distribution topologies can be found in [2], [3], [20], and [26]. The assumption can be relaxed for other topologies but details will not be addressed here.

Each clock signal travels from a primary clock input to a clocked cell after passing several clock nets and clockless cells. The complete travel path constitutes the clock signal path of the clocked cells. Fig. 7(a) illustrates an interconnection between two clocked cells and the clock signal paths of these two clocked cells. The black triangles represent the clockless cells along the clock signal path. In most SFQ clock distribution topologies, clock nets are shared between clock signal paths because the fanout of the clockless cells is typically more than 1. Thus, we only apply the ripup-and-reroute process to the clock net, which directly connects to the launch cell of the critical data signal path. Let the interconnection in Fig. 7(a) be the critical path, the targeted clock net is the net connecting the clock input of the left clocked cell. The applied ripup-and-reroute process is the same as the one used for the critical data nets. However, we apply a conservative optimization strategy here because minor wire length reduction in some clock nets can result in many hold time violations due to clock skew increase. Aggressive clock skew improvements for launch clock signal paths are attempted in the path rectification stage (see below) to minimize the likelihood of hold time violation growth.

5) Ripup-and-Reroute Process for Noncritical Nets: The objective of the ripup-and-reroute process for noncritical nets is to create wiring solutions for the nets whose wires were previously ripped up in order to enable the rerouting of critical nets. Again we use the maze routing algorithm to find feasible new wires for these nets. Unlike prior work [20], we do not use a comparison function to decide the routing order of the nets because our net count is small. The routing order of the nets is simply the order in which these nets were ripped up.

Since the wire length control is not necessary for noncritical nets, we do not construct an explicit bounding box to limit the exploration space. Note, however, that the exploration space is implicitly limited by the upper bound of the routing cost because wires with routing costs that are higher than the upper bound will not be accepted. Equation (7) is modified by replacing $g(b_j^{l1}, n_i)$ with $h(b_j^{l1}, n_i)$ in order to calculate the routing cost of a wiring solution for a noncritical net n_i and rewritten as

$$C(b_j^{l1}, n_i) = \min_{b_k^{l2}} \; f(b_j^{l1}, b_k^{l2}) + h(b_j^{l1}, n_i) + C(b_k^{l2}, n_i) \quad (10)$$

where $f(b_j^{l1}, b_k^{l2})$ remains the same as that in (7). $h(b_j^{l1}, n_i)$ is zero if b_j^{l1} is not used by a routing wire. Otherwise, $h(b_j^{l1}, n_i)$ is

$$h(b_j^{l1}, n_i) = c_1 * \text{IDCount}(b_j^{l1}) + c_2 * \frac{\text{WL}(b_j^{l1})}{\text{MaxWL}} + c_5 \quad (11)$$

where c_1 and c_2 are the same weight coefficients used in (8) and c_5 is a new coefficient. Given (10), the maze routing algorithm is run for ten passes to find a feasible wire for a net. The upper bound of the routing cost in the kth pass for routing a net n_i is

$$\text{Ubound}(n_i, k) = 2 * \text{Ubound}(n_i, k-1) \tag{12}$$

$$\begin{aligned}\text{Ubound}(n_i, 1) &= c_6 + c_7 * \text{ViaCost} \\ &+ c_8 * \min(\text{width}(n_i), \text{length}(n_i)) * \text{SegCost}\end{aligned} \tag{13}$$

where c_6, c_7, and c_8 are coefficients. $\text{width}(n_i)$ and $\text{length}(n_i)$ denote the Manhattan distance of two pins in the y-direction and x-direction, respectively.

The ripup-and-reroute process for noncritial nets is repeated until there are no unrouted nets. If the path length of the critical paths is improved, the ripup-and-reroute process for the critical data signal path will be executed again. Otherwise, the framework proceeds to the path rectification step.

C. Path Rectification

The objective of path rectification is to reduce the delay of the launch clock signal path and resolve any hold time violations. The *launch clock signal path*| denotes to the clock signal path that forward a clock signal to the launch cell of the critical path. All clock nets of the launch clock signal path are considered for wire length reduction to achieve aggressive clock skew improvements. We fix hold time violations at the end because we can reserve as many routing resources as possible for critical paths.

1) Critical Clock Net Selection: Clock net optimization is a delicate optimization process because clock delay changes can result in clock skew increase for a pair of cells and clock skew decrease for another pair of cells, simultaneously. Take Fig. 7(b) as an example in which a cell i and a cell k share all clock nets except the clock nets that directly connect to their respective clock inputs. Let the interconnection between a cell i and a cell j be the *critical data signal path*. To improve the working frequency, we decrease the clock arrival time of a cell i by reducing the wire length of multiple nets along the launch clock signal path. At the same time, the clock skew between a cell k and a cell l also increases. However, if the clock skew increase is too large, the hold time condition between a cell k and a cell l may be violated. To avoid possible hold time violations, we estimate the risk of optimizing a clock net n_i by

$$\text{risk}(n_i) = \text{ChildNetCnt}(n_i) + c_9 * \frac{\text{CurWL}(n_i)}{\text{WLL}(n_i)} \tag{14}$$

where $\text{ChildNetCnt}(n_i)$ denotes the number of the child nets of n_i in the clock tree, $\text{CurWL}(n_i)$ is the current wire length of n_i, $\text{WLL}(n_i)$ is the wire length lower bound of n_i, and c_9 is another weight coefficient. We select the critical clock nets with the lowest risk and apply the ripup-and-reroute process for the selected nets.

2) Ripup-and-Reroute for Selected Clock Nets: The objective of the ripup-and-reroute process for selected clock nets remains the wire length reduction. Therefore, the same ripup-and-reroute process for critical nets is applied to the selected clock nets. The critical clock net selection and the ripup-and-reroute process are repeated a few times. The critical data and clock signal paths in an SFQ circuit are re-evaluated each time because the critical paths may change after the wire length reduction of the clock nets. The selected clock nets will not be reselected.

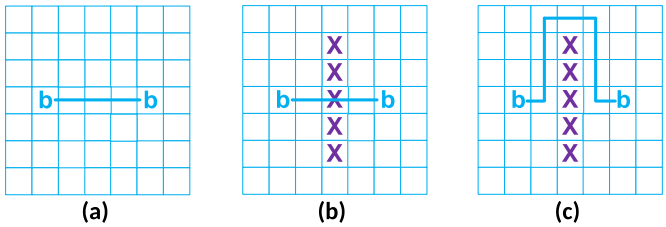

Fig. 8. Fixing hold time violations by temporary blockage placement. (a) A connection with a hold time violation. (b) Blockage placement. (c) A new connection without a hold time violation.

3) Fixing Hold Time Violations: There are three possible approaches to fix any hold time violations in an SFQ circuit: decreasing clock skew, inserting clockless cells, and increasing data signal path delays. The first approach must increase the path length of the clock signal path connecting to the launch cell or decrease the path length of the clock signal path connecting to the capture cell. This approach, however, has a high risk of losing control of the clock skew when the magnitude of the negative hold slack is very large. The cell insertion approach is usually implemented before or during the placement step. Moreover, timing adjustments using extra clockless cells, such as JTLs can cause margin reduction and even yield rate drop [22]. Therefore, we resort to data signal path delay increase even though it is counter-intuitive. The main reason is that with this method, we can fix significant number of hold time violations in the circuit while having a low risk of adversely affecting the clock skews and operation margins.

We present a novel strategy for resolving hold time violations given an SFQ circuit. Following are four steps in the strategy that we explain with the aid of Fig. 8.

1) Identifying a data signal path with negative hold slack and selecting the shortest wire of the identified data signal path for rerouting. Fig. 8(a) illustrates a wire as the selected wire.
2) Finding the midpoint of the selected wire and creating a temporary blockage lane orthogonal to the wire segment going through the mid point, as shown in Fig. 8(b). If the mid point is a turning point, the segment closer to the launch cell is chosen.
3) Ripping up the selected wire and performing a maze routing algorithm to rebuild a new wire for the original connection, as shown in Fig. 8(c). If the hold time violation is not fixed, we will repeat Steps 2 and 3 for up to ten times.
4) Removing the temporary blockages lanes created in Step 2 and repeating Step 1 until all data signal paths with negative hold slack are identified.

The length of the blockage lane created in Step 2 is the equivalent bin length of the hold slack minus one bin length. In Fig. 8, the hold slack is equivalent to 6 bin length, so a blockage lane of length 5 is created and placed in the space. The length of

TABLE I
STANDARD SFQ CELL LIBRARY

Cells	SplitterCLK	Splitter	NOT	DFF	AND	OR	XOR	NDRO
Height (μm)	40	120	120	120	120	120	120	120
Width (μm)	40	30	30	40	50	50	50	50
#Inputs (+Clk)	0 (+1)	1 (+1)	1 (+1)	1 (+1)	2 (+1)	2 (+1)	2 (+1)	1 (+1)
#Outpus (+Clk)	0 (+2)	2 (+1)	1 (+1)	1 (+1)	1 (+1)	1 (+1)	1 (+1)	1 (+1)
Clock-to-q Delay (ps)	5.7	5.7	13.0	6.8	8.7	6.0	6.3	10.0
Setup Time (ps)	-	-	3.9	1.1	0.0	2.6	4.8	10.0
Hold Time (ps)	-	-	6.1	4.0	4.7	3.1	4.8	10.0

a rebuilt wire is expected to be 6 if the rebuilt wire goes around the blockage lane. The maze routing algorithm performed in the third step follows (10) and (11) except that c_5 is increased by a factor of ten. The optimized routing result with a given SFQ circuit is generated in the design exchange format (DEF) after the path rectification step.

V. STANDARD SFQ CELL LIBRARY

Referring to the MIT-LL SFQ5ee process technology [13], we only have two (Nb) metal layers for signal routing, denoted as a bottom-layer M1 (connections on this layer are created using striplines that are sandwiched between grounded M0 and M2 layers) and a top-layer M3 (connections on this layer are created using striplines that are sandwiched between grounded M2 and M4 layers). Both routing layers, M1 and M3, are reserved for deploying PTLs. M5 is used for the biasing, whereas M6 is used to implement connections (and inductors) inside the cells.

We follow the design rules of the MIT-LL SFQ5ee process technology [27] to build our standard SFQ cell library and test our postrouting optimization framework. The standard SFQ cell library used in this article is also used in prior work [2], [20]. There are two parts in each SFQ cell: a logic design part realizing a Boolean function and a built-in clock distribution part splitting and/or passing clock pulses. There are eight types of the standard cells: SplitterCLK, Splitter, NOT, DFF, AND, OR, and XOR. SplitterCLK without the logic part is used for clock tree synthesis and the tree topology is the H-tree. The height of the clock part is 40 μm and that of logic part is 80 μm. Therefore, the height of all cells is 120 μm except that the height of SplitterCLK is 40 μm. The pins for the signal input and output of the logic part are in the M1 layer, whereas the pins for the signal input and output of the clock part are in the M3 layer. All cells are dc biased and each bias pillar is of size 2.5 μm $\times$ 2.5 μm. Details for timing parameters are listed in Table I.

Our postrouting optimization framework deploys PTLs with a propagation speed of 100 μm/ps for interconnections. We do not use JTLs for routing because JTLs require special routing when they cross in an orthogonal direction to one another [21], [28]. Moreover, long JTLs introduce a significant delay (JTL's tend to be ten times or so slower than PTLs), which goes against the objective of maximizing the working frequency of the SFW circuit. Notice that each PTL requires a PTL driver at the driving point and a PTL receiver at the receiving point. The PTL driver and receiver are embedded in each standard cell with matching characteristic impedances for the used PTL connections. The length of PTLs can reach 5 mm with negligible signal loss. More details can be found in [29]. The width and the pitch of PTLs are 5 and 10 μm, respectively. PTLs between two layers connect with each other through a via and the size of a via is 5 μm $\times$ 5 μm.

VI. EXPERIMENTAL RESULTS AND DISCUSSIONS

We have implemented the postrouting optimization framework in about 6000 lines in C language. The hardware environment for the experimental results is a Linux machine with the Intel(R) Xeon(R) CPU E7-8837 @2.67 GHz.

To evaluate the proposed optimization framework for two layer routing, we synthesize Kogge–Stone adders (KSAs), array multipliers (Muls), and an integer divider (IntDiv) using state-of-the-art SFQ EDA tools. These SFQ EDA tools covers the steps of logic synthesis [30], cell placement [2], clock tree generation [26], and net routing [20], [31]. Moreover, a number of the ISCAS c-series benchmarks with different net counts are generated by the same EDA tools. The synthesized SFQ circuits are path-balanced, and splitters are inserted during the logic synthesis step. We run the placement tool to place SFQ cells in rows with routing channels in between. An optimized H-tree with minimum clock skew is generated as the clock distribution topology of each SFQ circuit. All netlists of SFQ cells, including both data nets and clock nets, are routed using a general router, Qrouter, and an SFQ router, qGDR. Neither Qrouter nor qGDR is empowered with the relevant knowledge to optimize SFQ timing behaviors. Input file formats of the proposed framework are open standard library exchange format (LEF) and DEF files. LEF elaborates design rules and cell layouts, whereas DEF describes circuit netlists and corresponding circuit layouts.

Table II reports the routing results generated by Qrouter without and with the postrouting optimization for the test circuits. Qrouter with the postrouting optimization is marked as Qrouter*. *IdealTotalWL* denotes the ideal total wire length obtained by routing all nets with L-shaped wires regardless of any blockages or routing resource limitations. *ActualTotalWL* is the actual total wire length of PTLs. Values of *ActualTotalWL* in the seventh column are nearly the same as those in the sixth column, which suggests that the routing wires of most nets are not changed after the postrouting optimization. *ViaCnt* denotes the total number of vias used for routing. When we compare the values of *ViaCnt* between the eighth and the ninth columns, we observe a minor increase in *ViaCnt* for most circuits after the postrouting optimization. This increase is reasonable because extra vias are needed for building wires that must detour. For example, building the detour wire in Fig. 8(c) requires four vias because there are four turn points.

MaxViaCnt denotes the maximum number of vias used for routing any net in a test circuit. Similar to the prior work [20], values of *MaxViaCnt* in the tenth and the eleventh columns are rather large because there are only two routing layers. However, *MaxViaCnt* is prone to increase after the optimization for some circuits because a few of the nets are redone with many vias so as to reduce their wire length. However, a significant increase in *MaxViaCnt* is observed for some circuits. We explain the reason for this large increase in the context of the 16-b KSA but the

TABLE II
ROUTING RESULTS OF KSAS, ARRAY MULTIPLIERS, INTEGER DIVIDERS, AND C-SERIES CIRCUITS OF ISCAS BENCHMARKS BY STANDARD QROUTER AND QROUTER WITH POSTROUTING OPTIMIZATION (DENOTED BY QROUTER*)

Circuit	Spec.				ActualTotalWL (μm)		ViaCnt		MaxViaCnt		Frequency (GHz)		Running
	#Cells	#Nets	Area (mm^2)	IdealTotalWL (μm)	Qrouter	Qrouter* (Ratio)	Qrouter	Qrouter* (Ratio)	Qrouter	Qrouter* (Ratio)	Qrouter	Qrouter* (Ratio)	Time (s)
4-bit KSA	171	258	1.75	4.83e4	5.72e4	5.64e4 (0.99)	375	367 (0.99)	8	6 (0.75)	22.4	27.1 (1.22)	0.55
8-bit KSA	534	776	3.87	1.44e5	1.76e5	1.77e5 (1.00)	1185	1199 (1.01)	12	11 (0.91)	23.9	29.3 (1.23)	13.70
16-bit KSA	1215	1847	8.88	4.03e5	5.12e5	5.32e5 (1.03)	3776	3999 (1.05)	20	34 (1.70)	18.7	18.7 (1.00)	519.4
32-bit KSA	3753	5311	30.9	1.62e6	1.86e6	1.88e6 (1.01)	9500	9678 (1.01)	28	28 (1.00)	11.6	12.3 (1.06)	336.0
4-bit Mul	526	771	4.24	1.32e5	1.57e5	1.59e5 (1.01)	1000	1006 (1.00)	10	14 (1.40)	18.1	21.6 (1.19)	3.31
8-bit Mul	3458	4815	25.1	9.51e5	1.07e6	1.08e6 (1.00)	5414	5494 (1.01)	16	16 (1.00)	15.3	17.4 (1.14)	175.8
4-bit IntDiv	1092	1636	9.41	3.68e5	4.35e5	4.49e5 (1.03)	2578	2708 (1.05)	20	20 (1.00)	17.9	18.8 (1.05)	232.1
8-bit IntDiv	7363	10555	86.0	2.87e6	3.08e6	3.08e6 (1.00)	11416	11416 (1.00)	24	24 (1.00)	7.4	7.5 (1.01)	225.2
c432	2291	3500	25.0	9.70e5	1.09e6	1.10e6 (1.00)	5346	5446 (1.01)	16	20 (1.25)	11.4	11.7 (1.03)	44.65
c499	2091	3045	18.2	9.52e5	1.11e6	1.12e6 (1.00)	6357	6447 (1.01)	22	32 (1.45)	11.4	12.2 (1.07)	173.0
c880	3649	5129	33.8	1.59e6	1.79e6	1.80e6 (1.00)	8317	8491 (1.02)	26	20 (0.76)	10.2	11.9 (1.17)	479.0
c1355	2130	3124	21.5	1.12e6	1.29e6	1.29e6 (1.00)	6424	6470 (1.00)	28	28 (1.00)	11.3	11.8 (1.04)	75.19
c1908	3706	5255	30.3	1.45e6	1.65e6	1.68e6 (1.01)	8238	8408 (1.02)	22	22 (1.00)	10.4	10.8 (1.04)	208.4
Average Ratio	-	-	-	-	-	1.00	-	1.01	-	1.09	-	1.09	-

TABLE III
ROUTING RESULTS OF KSAS, ARRAY MULTIPLIERS, INTEGER DIVIDERS, AND C-SERIES CIRCUITS OF ISCAS BENCHMARKS BY STANDARD QGDR AND QGDR WITH POSTROUTING OPTIMIZATION (DENOTED BY QGDR*)

Circuit	Spec.				ActualTotalWL (μm)		ViaCnt		MaxViaCnt		Frequency (GHz)		Running
	#Cells	#Nets	Area (mm^2)	IdealTotalWL (μm)	qGDR	qGDR* (Ratio)	qGDR	qGDR* (Ratio)	qGDR	qGDR* (Ratio)	qGDR	qGDR* (Ratio)	Time (s)
4-bit KSA	171	258	1.75	4.83e4	5.55e4	5.52e4 (0.99)	353	353 (0.98)	6	6 (1.00)	24.9	26.5 (1.11)	0.14
8-bit KSA	534	776	3.87	1.44e5	1.71e5	1.71e5 (1.00)	1086	1076 (0.99)	14	14 (1.00)	24.3	24.6 (1.01)	0.56
16-bit KSA	1215	1847	8.88	4.03e5	4.72e5	4.76e5 (1.00)	3025	3067 (1.01)	16	16 (1.00)	19.7	22.1 (1.12)	5.97
32-bit KSA	3753	5311	30.9	1.62e6	1.80e6	1.81e6 (1.00)	7703	7785 (1.01)	26	26 (1.00)	10.5	11.9 (1.13)	119.1
4-bit Mul	526	771	4.24	1.32e5	1.49e5	1.49e5 (1.00)	893	887 (0.99)	8	9 (1.12)	21.2	22.2 (1.05)	0.52
8-bit Mul	3458	4815	25.1	9.51e5	1.04e6	1.04e6 (1.00)	4523	4555 (1.00)	18	18 (1.00)	15.7	16.2 (1.03)	26.95
4-bit IntDiv	1092	1636	9.41	3.68e5	4.15e5	4.18e5 (1.00)	2187	2211 (1.01)	19	19 (1.00)	15.7	17.9 (1.14)	12.49
8-bit IntDiv	7363	10555	86.0	2.87e6	3.05e6	3.05e6 (1.00)	9959	9959 (1.00)	24	24 (1.00)	6.8	7.1 (1.04)	31.32
c432	2291	3500	25.0	9.70e5	1.06e6	1.07e6 (1.00)	4560	4582 (1.00)	16	16 (1.00)	11.9	12.5 (1.05)	14.53
c499	2091	3045	18.2	9.52e5	1.07e6	1.07e6 (1.00)	5031	5069 (1.00)	24	24 (1.00)	11.7	12.6 (1.08)	84.78
c880	3649	5129	33.8	1.59e6	1.75e6	1.75e6 (1.00)	6969	7071 (1.01)	24	24 (1.00)	10.7	11.9 (1.11)	162.1
c1355	2130	3124	21.5	1.12e6	1.25e6	1.25e5 (1.00)	5575	5597 (1.00)	28	28 (1.00)	11.9	12.3 (1.03)	16.79
c1908	3706	5255	30.3	1.45e6	1.60e6	1.60e6 (1.00)	6783	6851 (1.00)	24	24 (1.00)	10.7	11.5 (1.07)	37.78
Average Ratio	-	-	-	-	-	1.00	-	1.00	-	1.00	-	1.07	-

explanation applies to other circuits (e.g., c499). Referring to the routing result of the 16-b KSA in the fourth row of Table II, the large increase in *MaxViaCnt* is caused by redoing a net with a large hold time violation. The initial hold slack of this net is -6.2 ps and the pins of this net are in congested regions of the chip. Consequently, 34 vias are used to create a detour wire within the congested regions to fix the hold time violation of this net.

Frequency in Table II is the maximum working frequency of an SFQ circuit and is the inverse of the minimum clock period calculated in (5). The thirteenth column in Table II shows ratio improvements in *Frequency* for the test circuits. The frequency improvements are 9% on average and the maximum working frequency can be boosted by more than 20% for small circuits (e.g., 4-b and 8-b KSAs). The improvements confirm the power of the postrouting optimization framework. The fourteenth column in Table II shows that the largest execution time is 519 s (encountered when optimizing the 16-b KSA circuit).

Table III reports the routing results generated by qGDR without and with the postrouting optimization given the testing SFQ circuits. qGDR with the postrouting optimization is identified as qGDR*. The values of *ActualTotalWL* in the seventh column are almost the same as those in the sixth column. There is no change in *MaxViaCnt* for all circuits except the 4-b Mul, which *MaxViaCnt* increases by 1 after the postrouting optimization. Furthermore, the increase in *ViaCnt* in Table III after the optimization is less than 1% of the initial value. If we compare values of *Frequency* in the twelfth columns with those in the thirteen columns, the maximum working frequency of all circuits is improved. The frequency improvements is 7% on average instead of 9% because the working frequency enabled by qGDR is on average 1.02X of that enabled by Qrouter. Therefore, the absolute magnitudes of frequency improvements achieved by the postrouting optimizer on the Qrouter and qGDR routing results are close to each other although the overheads of optimizing the qGDR results are lower as explained below. The fourteenth column in Table III shows that the whole optimization process can finish in 180 s for all circuits.

We achieve comparable frequency improvements with lower overheads for qGDR routing results because, in general, qGDR produces routing results with better wire distributions compared to that produced by Qrouter. We use the 16-b KSA as an example to illustrate our observations. The top and bottom two figures of Fig. 9 show the wire density graphs of the 16-b KSA routing

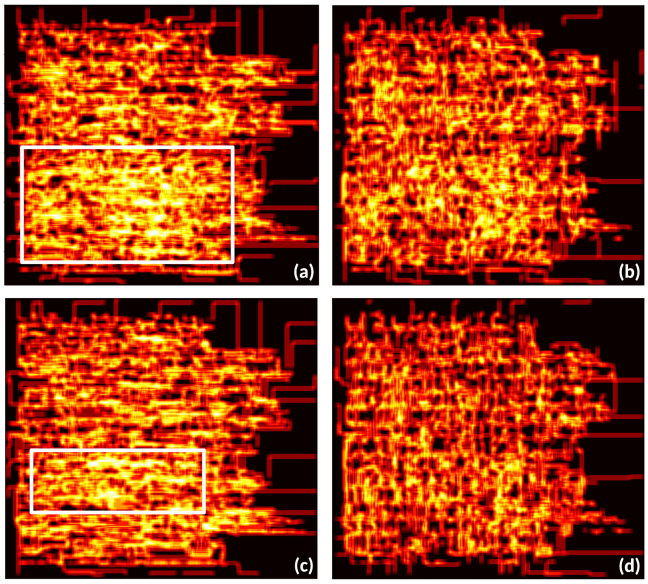

Fig. 9. Routing results of a 16-b KSA. (a) M1 wire density graph using Qrouter. (b) M3 wire density graph using Qrouter. (c) M1 wire density graph using qGDR. (d) M3 wire density graph using qGDR.

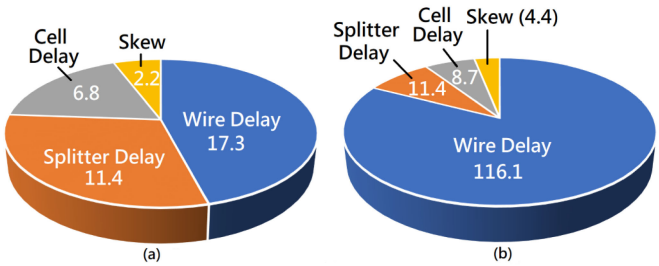

Fig. 10. Pie charts of the timing factors of a critical path. (a) 4-b KSA. (b) 8-b IntDiv. The unit of the number is picoseconds.

results generated by Qrouter and qGDR, respectively. If we compare the M1 wire densities in Fig. 9(a) and (c), we can see that the total size of high wire density regions in Fig. 9(a) is much larger than that in Fig. 9(c). These high wire density regions signify the congested regions of the layout. In addition, more routing resources in the M3 routing layer are consumed by Qrouter compared to qGDR, as shown in Fig. 9(b) and (d). The larger size of congested regions and the overutilization of routing resources are also observed in other routing results generated by Qrouter.

We further analyze the importance of each timing factor of a critical path. Specifically, we illustrate the relative scale of the timing factors using the smallest and largest SFQ circuits after optimizing the routing results generated by qGDR. The result is shown in Fig. 10. The cell delay includes the clock-to-q delay and the setup time of the critical path, whereas the wire delay is the summation of delays of all wire segments of the critical data signal path. Other timing factors are self-evident. The nonzero skew value in Fig. 10 is due to a minor path length difference after routing. Fig. 10(a) suggests that the critical timing factor for small circuits includes the splitter delay and the wire delay because there could be multiple splitters on a data signal path.

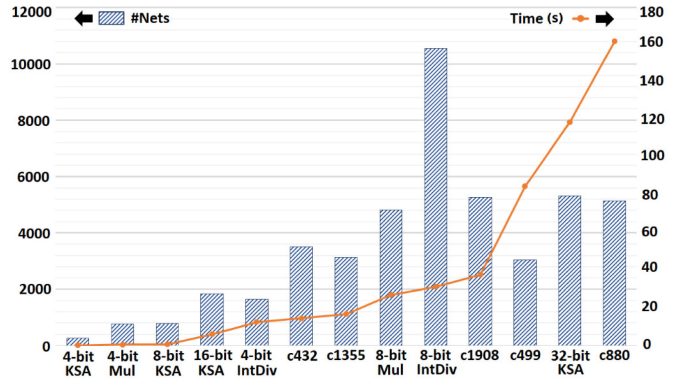

Fig. 11. Execution time of benchmark circuits with different numbers of nets.

The former factor becomes less important when the number of nets increases, as shown in Fig. 10(b). The large value of the wire delay confirms that the path length minimization for the critical path is crucial for realizing large-scale SFQ circuits with high performance.

Given the benchmark results generated by qGDR, we report the execution time of our postrouting optimization in Fig. 11. This figure shows that the execution time generally increases when the number of cells (or nets) increases. However, the execution time of some circuits can be less than expected (e.g., 8-b IntDiv) or more than expected (e.g., c499 and c880). We attribute this irregularity to the large variations of the M1 wire distribution and explain details using Fig. 12. We emphasize the M1 wire distribution because the routing resource demand of the M3 layer (used mainly for vertical wire segments) is far lower than the demand for the M1 layer (used mainly for horizontal wire segments). Fig. 12(a), which is the M1 wire density graph of the 8-b Mul, is used as a typical M1 wire density graph for comparisons. Fig. 12(b) is the M1 wire density graph of the 8-b IntDiv. Obviously, the area of high wire density regions is lower for the 8-b IntDiv compared to that of the 8-b Mul. As a result, fewer nets are ripped up in the 8-b IntDiv to build alternative short wires for critical data signal paths, and therefore, the execution time is lower. Similarly, comparing Figs. 12(a) and (c), we observe that the number of rerouted nets in Fig. 12(c) is much higher than that in Fig. 12(a), which explains the large execution time of c499. The execution time of c880 is also considerably increased due to the large area of high wire density regions, as shown in Fig. 12(d). These results point to the large effect of the area of high wire density regions on the execution time of the postrouting optimization.

Next, we report a full timing analysis of all routing results for the SFQ circuits in Table IV. The postrouting optimization reduces the number of hold time violations and increases the hold slack. In practice, hold time violations in some of Qrouter routing results could not be fixed, as shown in the sixth column of this table. Let us consider hold time violations of the 16-b KSA in the fifth row as an example. There are 32 hold time violations and the worst hold slack is -10.7 ps. If we want to fix the path with the worst hold slack, we need to increase the data signal path length of this path by 1070 μm. Adding wires with such a large length increase is undesirable and impractical

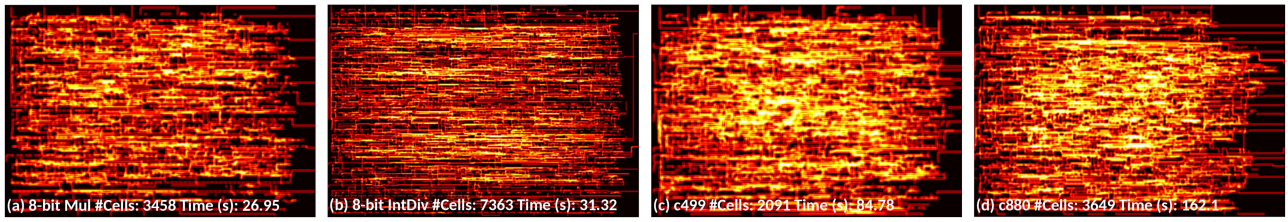

Fig. 12. M1 wire density graph after qGDR. (a) 8-b Mul. (b) 8-b IntDiv. (c) c499. (d) c880.

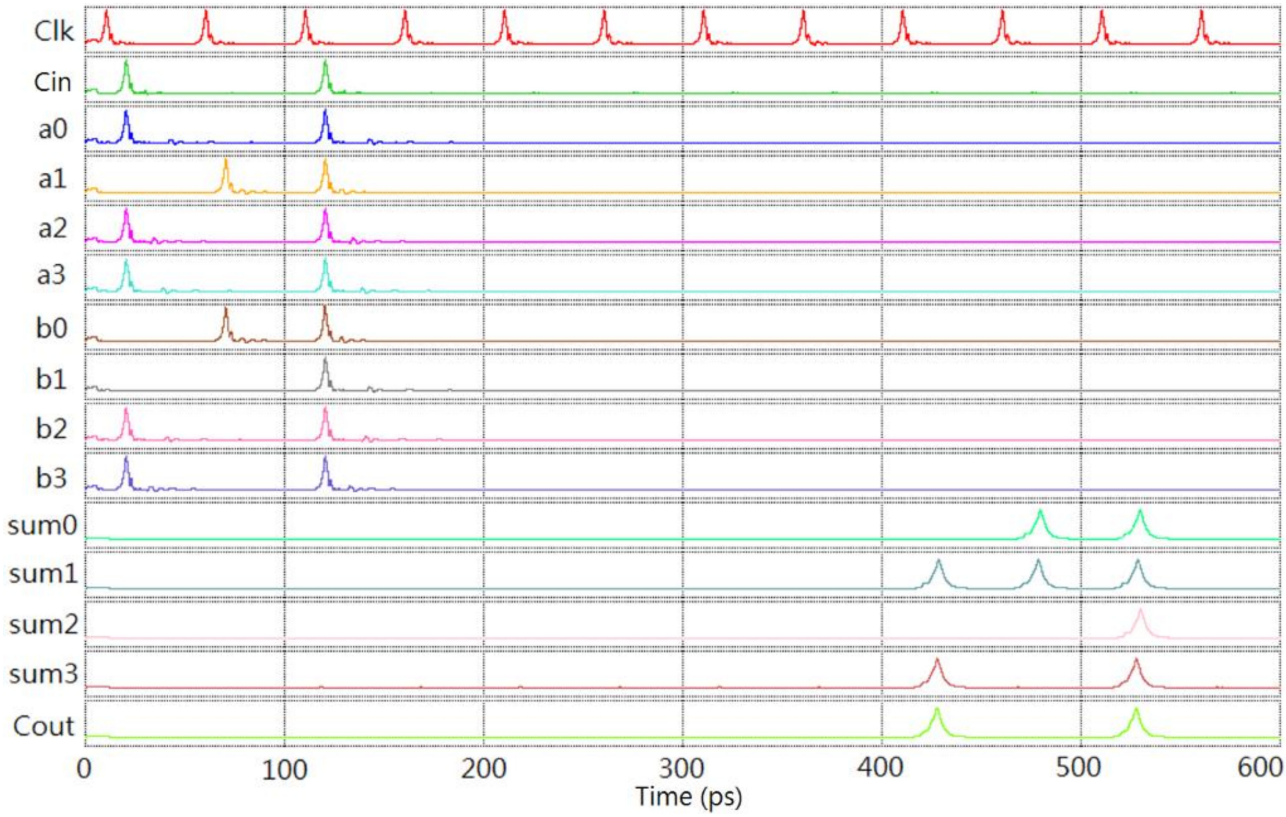

Fig. 13. Simulation result of the 4-b KSA by WRspice.

TABLE IV
TIMING ANALYSIS OF KSAS, ARRAY MULTIPLIERS, INTEGER DIVIDERS, AND C-SERIES CIRCUITS OF ISCAS BENCHMARKS

Circuit	Spec.			#HoldViolations				WorstHoldSlack (ps)				Frequency (GHz)	
	#Cells	#Nets	Area (mm^2)	Qrouter	Qrouter*	qGDR	qGDR*	Qrouter	Qrouter*	qGDR	qGDR*	Qrouter*	qGDR* (Ratio)
4-bit KSA	171	258	1.75	1	0	0	0	-0.7	≥ 0	≥ 0	≥ 0	27.1	26.5 (0.98)
8-bit KSA	534	776	3.87	4	0	4	0	-4.3	≥ 0	-2.2	≥ 0	29.3	24.6 (0.84)
16-bit KSA	1215	1847	8.88	32	2	7	0	-10.7	-2.6	-2.1	≥ 0	18.7	22.1 (1.18)
32-bit KSA	3753	5311	30.9	24	0	16	0	-6.3	≥ 0	-6.8	≥ 0	12.3	11.9 (0.97)
4-bit Mul	526	771	4.24	4	0	1	0	-4.0	≥ 0	-0.1	≥ 0	21.6	17.9 (0.83)
8-bit Mul	3458	4815	25.1	22	0	8	0	-8.4	≥ 0	-4.3	≥ 0	17.4	17.9 (1.03)
4-bit IntDiv	1092	1636	9.41	17	1	8	0	-7.9	-2.6	-3.1	≥ 0	18.8	17.9 (0.95)
8-bit IntDiv	7363	10555	86.0	74	4	14	0	-12.7	-3.8	-3.2	≥ 0	7.5	7.1 (0.95)
c432	2291	3500	25.0	25	0	12	0	-7.6	≥ 0	-3.3	≥ 0	11.7	12.5 (1.07)
c499	2091	3045	18.2	12	0	1	0	-8.2	≥ 0	-1.0	≥ 0	12.2	12.6 (1.03)
c880	3649	5129	33.8	26	2	13	0	-8.1	-5.7	-6.1	≥ 0	11.9	11.9 (1.00)
c1355	2130	3124	21.5	12	1	4	0	-4.6	-3.6	-1.8	≥ 0	11.8	12.3 (1.04)
c1908	3706	5255	30.3	29	1	12	0	-1.3	≥ 0	-4.6	≥ 0	10.8	11.5 (1.06)
Average Ratio	-	-	-	-	-	-	-	-	-	-	-	-	0.99

in the postrouting step because the addition of such elongated wires will change the routing result significantly, causing design convergence problems. A better approach is to insert clockless cells (e.g., JTLs or splitters) on this path in order to increase the signal path delay [3], [18], and [22]. As explained before, a cell insertion changes the circuit netlist and necessitates cell replacement and rerouting, which is undesirable in the postrouting step (although it has been suggested, see for example [21].)

The seventh and the eleventh column in Table IV suggest that qGDR can effectively control both hold time violations and the worst hold slack of all SFQ circuits. The largest number of the hold time violations is 23 and the worst hold slack is -6.8. Given the qGDR routing results with low routing congestion, our postrouting optimization framework successfully creates alternative detour wires and, thus, resolves all hold time violations, as seen in the twelfth column. Moreover, the overhead of resolving the violations is rather small as we seen in Table III. Although removing all hold time violations cannot be guaranteed, the proposed framework can still effectively resolve most hold time violations and reduce the worst hold slack of SFQ routing results. The last two columns in Table IV show that we achieve comparable working frequencies for the two cases of the optimized Qrouter and the optimized qGDR.

After the postrouting optimization, we run a WRspice circuit simulator given the 4-b KSA to confirm the correct functional behaviors of all interconnected cells at a high frequency. This circuit performs the operation of A + B + Cin and the corresponding output is Sum and Cout. Considering the signal path delays and operating margins, we opt to test the circuit at 20-GHz frequency (which is 075% of the maximum clock frequency we can run this circuit at). Simulation results are reported in Fig. 13. We give three sets of inputs, which are as follows:

1) A=1101, B=1100, and Cin=1 (the last digit is the least significant bit);
2) A=0010, B=0001, and Cin=1;
3) A=1111, B=111, and Cin=1.

By observing the signal at the output, we obtain corresponding correct output sets: Sum=1010 and Cout=1; Sum=0010 and Cout=0; and Sum=1111 and Cout=1. The simulation result points to the feasibility of our framework for building high-performance SFQ circuits with correct functional behaviors post-place&route.

VII. Conclusion

We present a postrouting optimization framework for large SFQ circuits fabricated in the MIT-LL SFQ5ee process technology. Given only two PTL routing layers, the proposed framework not only enhances the maximum working frequency of a routed SFQ circuit but also resolves hold time violations. The framework comprises three stages: machine learning, critical path optimization, and path rectification. Machine learning efficiently identifies congested regions in a circuit layout through clustering algorithms. With the congestion analysis result, critical path optimization reduces the path length of the critical paths in the circuit by a ripup-and-reroute process for a working frequency improvement. Path rectification controls the clock skew of the critical paths by an aggressive ripup-and-reroute process and fixes the hold time violations by building detour paths. Given 13 routing results generated by the state-of-the-art SFQ router, our postrouting optimization framework improves the working frequency by 7% on average and resolves all hold time violations with negligible overheads. The execution time of all 13 routing result is no more than 3 min, including an 8-b integer divider with 10 555 nets.

Acknowledgment

The authors would like to thank N. Katam and G. Pasandi for helpful discussions and providing circuit netlists used in this article.

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 ODNI, IARPA, 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] J. A. Delport and C. J. Fourie, "A static timing analysis tool for superconducting digital circuit applications," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 5, Aug. 2018, Art. no. 1300705.
[2] S. N. Shahsavani, T.-R. 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, Jun. 2017, Art. no. 1302008.
[3] C. J. Fourie, "Digital superconducting electronics design tools—Status and roadmap," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 5, Aug. 2018, Art. no. 1300412.
[4] S. Nishijima *et al.*, "Superconductivity and the environment: A roadmap," *Supercond. Sci. Technol.*, vol. 26, Sep. 2013, Art. no. 113001.
[5] T. V. Duzer and C. W. Turner, *Principle of Superconducting Devices and Circuits*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1998.
[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] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 760–769, Jun. 2011.
[8] A. F. Kirichenko, I. V. Vernik, J. A. Vivalda, R. T. Hunt, and D. T. Yohannes, "ERSFQ 8-bit parallel adders as a process benchmark," *IEEE Trans. Appl. Supercond.*, vol. 25, no. 3, Jun. 2015, Art. no. 1300505.
[9] D. Kirichenko, S. Sarwana, and A. Kirichenko, "Zero static power dissipation biasing of RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 776–779, Jun. 2011.
[10] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-efficient superconducting computing—Power budgets and requirements," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, Jun. 2013, Art. no. 1701610.
[11] M. H. Volkmann, A. Sahu, C. J. Fourie, and O. A. Mukhanov, "Implementation of energy efficient single flux quantum digital circuits with sub-aJ/bit operation," *Supercond. Sci. Technol.*, vol. 26, no. 1, 2013, Art. no. 015002.
[12] M. H. Volkmann, I. V. Vernik, and O. A. Mukhanov, "Wave-pipelined eSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 25, no. 3, Jun. 2015, Art. no. 1301005.
[13] S. K. Tolpygo, "Superconductor digital electronics: Scalability and energy efficiency issues," *Low Temp. Phys.*, vol. 42, no. 5, pp. 361–379, May 2016.
[14] V. K. Semenov, Y. A. Polyakov, and S. K. Tolpygo, "AC-biased shift registers as fabrication process benchmark circuits and flux trapping diagnostic tool," *IEEE Trans. Appl. Supercond.*, vol. 27, no. 4, Jun. 2017, Art. no. 1301409.
[15] C. J. Fourie *et al.*, "ColdFlux superconducting EDA and TCAD tools project: Overview and progress," *IEEE Trans. Appl. Supercond.*, vol. 29, no. 5, Aug. 2019, Art. no. 1300407.

[16] B. Zhang and M. Pedram, "qSTA: A static timing analysis tool for superconducting single-flux-quantum circuits," *IEEE Trans. Appl. Supercond.*, vol. 30, no. 5, Aug. 2020, Art. no. 1700309.
[17] 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. 2/3, pp. 247–276, 1997.
[18] K. Takagi, Y. Ito, S. Takeshima, M. Tanaka, and N. Takagi, "Layout-driven skewed clock tree synthesis for superconducting SFQ circuits," *IEICE Trans. Electron.*, vol. E94-C, pp. 288–295, Mar. 2011.
[19] Y. Tukel, A. Bozbey, and C. A. Tunc, "Development of an optimization tool for RSFQ digital cell library using particle swarm," *IEEE Trans. Appl. Supercond.*, vol. 23, no. 3, Jun. 2013, Art. no. 1700805.
[20] T.-R. Lin, T. Edwards, and M. Pedram, "qGDR: A via minimization oriented routing tool for large-scale superconductive single-flux-quantum circuits," *IEEE Trans. Appl. Supercond.*, vol. 29, no. 7, Oct. 2019, Art. no. 1303412.
[21] M. Tanaka *et al.*, "Automated passive-transmission-line routing tool for single-flux-quantum circuits based on A* algorithm," *IEICE Trans. Electron.*, vol. E93.C, no. 4, pp. 435–439, Apr. 2010.
[22] N. Kito, K. Takagi, and N. Takagi, "Automatic wire-routing of SFQ digital circuits considering wire-length matching," *IEEE Trans. Appl. Supercond.*, vol. 26, no. 3, Apr. 2016, Art. no. 1300305.
[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data*, Aug. 1996, pp. 226–231.
[24] J. Kleinberg and E. Tardos, *Algorithm Design*. Reading, MA, USA: Addison-Wesley, 2005.
[25] W. S. Scott and J. K. Ousterhout, "Plowing: Interactive stretching and compaction in magic," in *Proc. Des. Autom. Conf.*, Jun. 1984, pp. 166–172.
[26] 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, Dec. 2019, Art. no. 1303513.
[27] S. K. Tolpygo, V. Bolkhovsky, T. J. Weir, L. M. Johnson, M. A. Gouker, and W. D. Oliver, "Fabrication process and properties of fully-planarized deep-submicro Nb/Al-AlO$_x$/Nb Josephson junctions for VLSI circuits," *IEEE Trans. Appl. Supercond.*, vol. 25, no. 3, Jun. 2015, Art. no. 1101312.
[28] 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.
[29] H. Suzuki, S. Nagasawa, K. Miyahara, and Y. Enomoto, "Characteristics of driver and receiver circuits with a passive transmission line in RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 10, no. 3, pp. 1637–1641, Sep. 2000.
[30] G Pasandi and M. Pedram, "A dynamic programming-based, path balancing technology mapping algorithm targeting area minimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2019, pp. 1–8.
[31] "Open circuit design," Aug. 2016. [Online]. Available: http://opencircuitdesign.com/qrouter/index.html