# JPnR: A Length-Matching Placement and Routing Framework for Single-Flux-Quantum Circuits

Rongliang Fu, Minglei Zhou, Siyan Chen, Xinda Chen, Junying Huang, Xiaochun Ye, Zhimin Zhang, and Tsung-Yi Ho *Fellow, IEEE*

***Abstract*—Superconducting rapid single-flux-quantum (RSFQ) logic is a promising candidate for advancing future computing technologies due to its low-energy consumption and high-frequency capabilities. However, precise timing alignment is crucial for its physical design, posing significant challenges in length-matching placement and routing. This paper introduces JPnR, a physical design framework tailored for RSFQ circuits, featuring a clock-aware length-matching placer and a length-matching multi-terminal router. The placer simultaneously considers both clock distribution and timing constraints, distributing clock pulses heuristically and transforming the placement problem into a single-source shortest-path problem. This allows it to minimize vertical wirelength using dynamic programming and iteratively optimize placement via a barycenter-like reordering method. The router tackles challenges related to splitter placement and length-matching multi-terminal routing using a two-layer planar Manhattan routing model. Initial routing assigns tracks based on the left-edge algorithm to minimize routing width while employing the dogleg algorithm to resolve cycles in the vertical constraint graph. Length-matching is achieved via a splitter tree-based hierarchical approach with maximum-flow-based detour insertion. Finally, a PTL region expansion strategy is employed for unsatisfied connections. Experimental results on RSFQ benchmarks demonstrate the effectiveness and efficiency of JPnR.**

***Index Terms*—Superconducting electronics, single-flux-quantum, placement, routing, length-matching, clock-aware**

## I. Introduction

RAPID single flux-quantum (RSFQ) circuits [1] represent a promising category of superconducting integrated circuits, leveraging Josephson junctions (JJs) to achieve high-performance computing capabilities. JJs are superconducting active devices based on the Josephson effect, characterized by exceptional switching speeds ($\sim$1 ps per switch) and ultra-low switching energy ($<10^{-19}$ J per switch) [2]. The RSFQ T flip-flop has been demonstrated to operate at frequencies up to 770 GHz at a temperature of 4.2 K [3]. The potential of RSFQ technology to revolutionize high-speed computing has driven significant advancements in RSFQ processors and accelerators. With improvements in RSFQ fabrication, researchers have successfully designed critical processor components, including arithmetic logic units (ALUs) [4], register files [5], and branch predictors [6], as well as microprocessors that operate at frequencies exceeding 30 GHz [7]. Moreover, RSFQ technology has been applied to hardware accelerators capable of operating at tens of gigahertz frequencies [8], [9].

Despite these advancements, the design of complex RSFQ circuits remains challenging due to the immaturity of design automation tools. The lack of robust, industrial-strength design tools for RSFQ circuits complicates physical design, often necessitating extensive timing adjustments to achieve closure. Unlike CMOS technology, the operating frequency of RSFQ circuits depends critically on the difference between the arrival times of data and clock pulses [10]. A significant discrepancy in the arrival times of data and clock pulses at RSFQ logic gates can lead to a substantial decrease in operating frequency. Therefore, ensuring that the arrival times of data and clock pulses are closely matched is essential for maximizing the frequency of RSFQ circuits.

To develop an efficient physical design tool for RSFQ logic, several specific challenges must be addressed. A primary challenge involves ensuring the timing alignment of clock and data pulses for all clock-driven logic gates, as any timing violation can result in functional errors. Additionally, constructing the clock distribution network during the physical design process presents challenges, as different networks can significantly impact the lengths of clock and data connections, thereby affecting the overall circuit area. Furthermore, the non-standardized layouts of RSFQ library cells, which vary in width and height, complicate the consideration of realistic cell sizes and delays in automated physical design. Accurate modeling of these diverse cells is necessary for effective automation. Splitter placement also poses a significant challenge in RSFQ physical design.

This paper introduces JPnR, a physical design framework for RSFQ circuits that addresses these challenges. We propose two key methodologies: (1) a clock-aware length-matching placement algorithm that optimizes the timing alignment of clock and data pulses, and (2) a splitter-aware multi-terminal hierarchical length-matching routing algorithm that minimizes routing width. We model the RSFQ timing alignment limitation as a length-matching problem, accounting for both clock and data connection lengths during placement and routing. In

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust and was supported in part by the Research Grants Council of Hong Kong SAR (Grant No. CUHK14207523), in part by the National Natural Science Foundation of China (Grants No. 62302477), in part by State Key Lab of Processors, Institute of Computing Technology, CAS (Grant No. CLQ202402), and in part by Shandong Province Natural Science Foundation, China (Grant No. ZR2024MF073).

Rongliang Fu and Tsung-Yi Ho are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China. E-mail: {rlfu, tyho}@cse.cuhk.edu.hk.

Minglei Zhou, Junying Huang, Xiaochun Ye, and Zhimin Zhang are with the SKLP, Institute of Computing Technology, CAS, Beijing 100190, China. E-mail: {zhouminglei24s, huangjunying, yexiaochun, zzm}@ict.ac.cn.

Siyan Chen and Xinda Chen are with ShanghaiTech University, Shanghai 201210, China. E-mail: {chensy5, chenxd1}@shanghaitech.edu.cn.

Corresponding author: huangjunying@ict.ac.cn.

JPnR, clock distribution is generated during placement using a heuristic strategy within a multi-stage pipelined architecture. We use actual cell sizes and delay parameters from the RSFQ library to enhance practicality. Additionally, we address splitter placement during routing by considering the actual size and delay of splitters. Our evaluation demonstrates that the proposed placement and routing methods significantly outperform state-of-the-art methods.

Overall, the key contributions are as follows:

- We propose JPnR, an end-to-end framework for RSFQ physical design that integrates placement, clock tree generation, and routing.
- We propose a clock-aware length-matching placement algorithm that considers both clock and data connections to minimize total vertical wirelength, thereby alleviating timing alignment challenges during routing.
- We propose a multi-terminal hierarchical routing algorithm that jointly performs length-matching routing and splitter placement for RSFQ circuits.
- We introduce a track assignment-based initial routing algorithm that minimizes initial routing width while facilitating efficient track assignment for cyclic vertical constraint graph (VCG) structures.
- Experimental results show that the proposed placement algorithm reduces vertical wirelength by 43.91% and 43.23% on ISCAS85 [11] and EPFL [12] benchmarks, compared to the state-of-the-art methods. The proposed routing algorithm achieves substantial routing width reductions, averaging 38.20%, 38.29%, 21.52%, and 7.38% on random testcases, surpassing Kito's [13], Kou's [14], and Yan's [15], [16] methods.
- We use initial routing results to refine placement for more accurate wirelength estimation, improving overall quality. On ISCAS85 [11] and EPFL [12] benchmarks with the ColdFlux RSFQ logic cell library [17], JPnR achieves reductions of 4.56% in total wire length and 3.69% in circuit area.

## II. Background

### A. Rapid Single-Flux-Quantum Circuits

Superconducting RSFQ logic [1] is a representative ultra-fast and low-power circuit technology using superconducting devices called Josephson junctions, which consist of a thin insulator sandwiched between superconductors. RSFQ circuits utilize the existence of a single flux quantum (SFQ) in the JJ as an information carrier, similar to the voltage level in conventional CMOS technology. The presence or absence of an SFQ pulse indicates the logical value of '1' or '0'.

*1) Fan-out limitation*

Unlike conventional CMOS technology, an RSFQ logic gate can only drive one other gate due to its limited pulse drive capability. Therefore, multi-fan-out nets in RSFQ circuits necessitate the construction of splitter trees to distribute pulses from the source to multiple sinks. This requires a specialized JJ-based gate called the splitter (SPL). Consequently, numerous splitters appear in multi-fan-out nets, posing a significant challenge in RSFQ circuit routing.

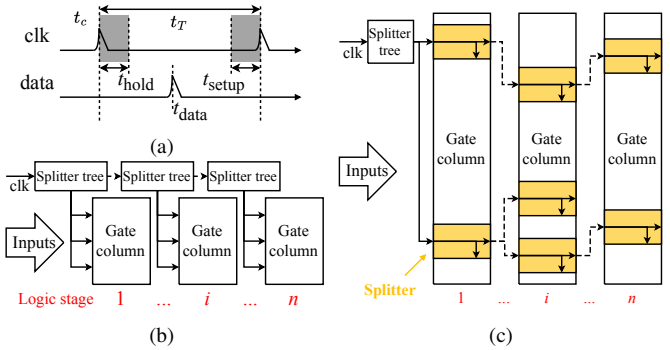



Fig. 1 (a) RSFQ logic timing constraints. (b) and (c) are the tree-based clock distributions for RSFQ circuits by Kito *et al.* [22] and Yan [23], respectively.

*2) Gate-level Pipelining*

Most RSFQ logic gates require a clock pulse to transfer the stored SFQ to adjacent gates, enabling natural gate-level pipelining. Consequently, to ensure proper logic operations of an RSFQ logic gate, all its fan-in gates should have the same logic stage, known as the path balancing property. Notably, the logic stage of a gate is defined as the maximum number of clocked gates from any primary input (PI) of the circuit to this gate [18]–[20]. Given this inherent gate-level pipelining characteristic, RSFQ circuits naturally adopt a pipelined layout based on the logic stage of each logic gate. In this pipelined RSFQ layout structure, the logic gates are organized in columns according to their logic stages. Logic gates with the same logic stage are placed within the same column. These columns are arranged in ascending order of the logic stages from left to right, establishing a multi-stage pipelined layout.

*3) RSFQ Interconnects*

RSFQ circuits utilize two primary interconnect types: Josephson transmission lines (JTLs) and passive transmission lines (PTLs). JTLs, comprising a series of JJs [1], are active components capable of introducing intentional delays but suffer from significant power consumption due to JJ switching. Conversely, PTLs are lossless superconducting waveguides, typically implemented as microstrip lines [21], enabling RSFQ pulses to propagate with minimal power consumption at approximately the speed of light. We employ PTLs for gate-to-gate connections and fine delay adjustments. Notably, since PTL delay scales linearly with length, timing adjustments can be effectively translated into length-matching constraints during physical design.

### B. Timing Constraints

RSFQ circuits utilize concurrent-flow clocking as a representative clock scheme [10], where the flow directions of the clock and data pulses are the same, and the clock arrives before the data. For the circuit to function correctly, the RSFQ data pulses must not only arrive within the same clock cycle but also maintain a specific delay in relation to the clock pulse. These requirements form the timing constraints of RSFQ circuits. The timing constraints associated with concurrent-flow clocking are illustrated in Fig. 1(a). To ensure the correct operation of the logic gate, the arrival time of its clock and data pulses must satisfy:

$$t_c + t_{hold} < t_{data} < t_c + t_T - t_{setup}, \quad (1)$$

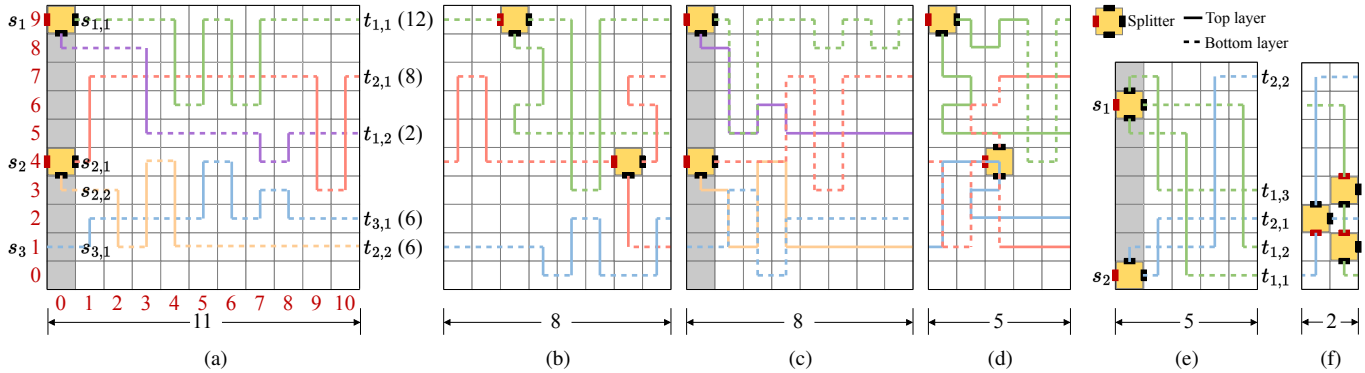



Fig. 2 Solutions of a length-matching routing example using (a) Kito's routing algorithm [13], (b) Kou's routing algorithm [14], (c) Yan's routing algorithm [15], and (d) ours, where the red numbers indicate the row and column indices. (e) and (f) show a comparison between splitter pre-allocation and splitter reallocation. The leftmost column, highlighted in gray, is the pre-allocated column for splitters, while the white grids are the PTL routing region.

which $t_c$ and $t_{data}$ refer to the arrival times of the clock and data pulses, respectively. $t_{setup}$, $t_{hold}$, and $t_T$ are timing parameters determined by the specific characteristics of the gate. $t_{setup}$ is the minimum time that the data must remain stable before the next clock pulse to ensure correct processing, while $t_{hold}$ ensures that the data arriving after this time does not affect the operation results of the gate in the current clock cycle. $t_T$ represents the clock cycle. If the arrival times of the data and clock pulses do not meet these timing constraints, the logic gate may produce incorrect results.

### C. Timing Alignment using Length-Matching

The timing constraints for each logic gate in the design of the RSFQ circuit can be expressed using Equation (1). Given the essential characteristic of PTL, where transmission delay is proportional to its length, the timing constraints can be transformed into a length-matching problem:

$$l^c + l^{hold} < l^{data} < l^c + l^T - l^{setup}, \tag{2}$$

where $l^c$ and $l^{data}$ represent the PTL lengths of the clock connection and data connection, respectively, with corresponding delays of $t_c$ and $t_{data}$. Additionally, $l^{hold}$, $l^T$, and $l^{setup}$ denote the PTL lengths that match the delays of $t_{hold}$, $t_T$, and $t_{setup}$, respectively. To ensure the circuit maintains a stable operating state, even with variations caused by process deviations, we expect the data pulses to arrive at the midpoint of the time interval, which means:

$$l^{data} = \frac{1}{2}\left[(l^c + l^{hold}) + (l^c + l^T - l^{setup})\right] = l^c + \Delta l, \tag{3}$$

where $\Delta l = \frac{l^T - l^{setup} + l^{hold}}{2}$ is the additional PTL length that the data connection requires compared to the clock connection.

In RSFQ circuits, the wiring is divided into two components: basic wires and detours. Basic wires link the corresponding pins with the shortest path possible while avoiding overlap with other connections. The detours are introduced to adjust the timing of the RSFQ circuit. Their values are estimated during placement and serve as input for routing.

The detour process utilizes available routing areas to make detours. However, when space is insufficient, an expanded routing region is required to facilitate timing adjustments. As detours increase, more routing space is needed, leading to a wider routing region. To optimize layout compactness, we adopt the zigzag wiring strategy from [13], [22], which restricts detours to the vertical direction between adjacent gate columns, thereby enhancing routing efficiency.

## III. Related Work

Superconducting circuits exhibit unique characteristics that present challenges across multiple domains, including device and material, fabrication processes, circuit and architecture design, system modeling, and electronic design automation (EDA) [24], [25]. Due to these distinctive properties and constraints, conventional EDA tools developed for CMOS circuits cannot be directly applied to superconducting circuits. Extensive research has therefore produced various automated methods for synthesis [18], [20], [26]–[29], verification [19], [30], timing analysis [31], and physical design [32], [33] of superconducting circuits. This paper concentrates on the physical design of superconducting RSFQ circuits, providing a review of existing work in three critical areas: clock distribution, placement, and routing.

### A. Clock Distribution

In RSFQ circuits, due to the strict timing constraints and the presence of numerous splitters for multiple fan-outs, effective clock distribution is essential in the physical design of RSFQ circuits. Previous studies have explored various tree-based clock distribution structures in RSFQ circuits. Kito *et al.* [22] developed a multi-level distribution tree to achieve clock distribution. As shown in Fig. 1(b), the input clock pulse first enters a splitter tree. Subsequently, this splitter tree splits this pulse into multiple clock pulses, where one clock pulse propagates to the next splitter tree while others propagate to the logic gates within the current gate column. However, since the clock inputs of all gates within one gate column originate from the same splitter tree, connections between the outputs of this tree and the clock pins of these gates require lengthy wires, increasing routing costs. To reduce the complexity of clock distribution, Yan [23] proposed a more efficient method for constructing clock trees, as shown in Fig. 1(c). Similar to Kito's method, the clock pulse is initially fed into a splitter tree, which delivers clock pulses to the gates within the first gate column. For the subsequent gate columns, each logic gate within the $i^{\text{th}}$ column is linked to a splitter, allowing the clock pulse to propagate to the $(i+1)^{\text{th}}$ column. This design effectively integrates clock splitter trees into logic gate

placement, significantly reducing routing costs and minimizing signal transmission delays. This paper adopts this clock distribution strategy and proposes a heuristic method to generate the clock distribution efficiently. Researchers have also drawn upon design practices from semiconductor circuits, proposing an H-tree based clock network to minimize clock skew in conjunction with a row-based cell placement method [34], where all RSFQ cells are of equal height. Alternatively, novel timing schemes to trade off maximum achievable throughput with lower overhead, such as the dual clocking method [35] and multi-phased clocks [36] have also been explored.

### B. RSFQ Placement

Several studies [22], [37]–[39] have been proposed for RSFQ placement. Kito *et al.* [22] proposed a simulated annealing (SA)-based algorithm, which is computationally intensive for large circuits and does not consider timing alignment. Yan [37] proposed a fixed-order placement method that outperforms the SA-based method in speed. However, this method fixes the relative positions of logic gates, thereby restricting the exploration of potential solution spaces, while it only focuses on the delay matching between data pulses and ignores clock pulses, which complicates timing alignment during routing. Similarly, Kitamura *et al.* [38] proposed a length-matching-aware placement method to minimize extended wirelength. However, this method also primarily focused on matching the lengths of the data nets, ignoring the timing constraints between the data and clock nets. Therefore, this paper first integrates the construction of the clock distribution into the placement and then minimizes the routing width while meeting the timing constraints.

### C. RSFQ Routing

Numerous studies [13]–[16], [40] have been proposed for the routing of RSFQ circuits. Kito *et al.* [13] employed a two-layer horizontal/vertical routing model and developed an integer linear programming-based method to minimize the routing width. In this routing model, as shown in Fig. 2(a), one layer is used for horizontal wires, and the other is used for vertical wires, significantly limiting the routing efficiency. Similarly, qGDR [41] adopts this routing model to employ channel-based routing methodologies for RSFQ circuits inspired by CMOS design. However, it does not apply superconducting concurrent clocking, resulting in lower operating frequencies. Yan [15] addressed this limitation by proposing a more flexible two-layer planar Manhattan routing model, which allows both horizontal and vertical wire assignments in the top and bottom layers, as shown in Fig. 2(c). This flexibility improves the utilization of limited routing resources in two routing layers. Additionally, Yan [16] proposed a via-minimization-oriented routing algorithm to mitigate the adverse effects of impedance mismatch in routing. However, these methods in [13], [15], [16] pre-place splitters into gate columns, which may increase the wirelength and decrease routability during length-matching routing. To integrate splitter placement with routing, Cheng *et al.* [40] and Kou *et al.* [14] achieved co-optimizing splitter placement and routing, as shown in Fig. 2(b). However, their methods are limited to the two-layer horizontal/vertical routing model, which restricts flexibility in resource utilization. These studies collectively highlight the complexity of RSFQ routing due to the extensive clock tree distribution and limited routing resources.

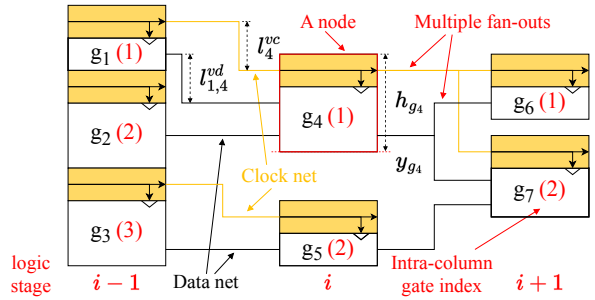



Fig. 3 Partial schematic of the circuit placement, where each node consists of one clock splitter (yellow box) and one logic gate. $g_4$ has a height of $h_{g_4}$ and is located at $y_{g_4}$. Its data inputs come from $g_1$ and $g_2$, and its output flows to $g_6$ and $g_7$, *i.e.*, $\text{FI}(g_4) = \{g_1, g_2\}$ and $\text{FO}(g_4) = \{g_6, g_7\}$. The vertical minimum length of its data connection from $g_1$ is $l_{1,4}^{vd}$. The vertical minimum length of its clock connection is $l_4^{vc}$.

## IV. Problem Formulation

### A. Terminology

An RSFQ circuit can be represented as a directed acyclic graph $G(V, E)$. The node set $V$ consists of three disjoint subsets: $I$, denoting PIs; $O$, denoting primary outputs (POs); and $C$, denoting logic gates. For a node $v \in V$, let $E_i(v)$ and $E_o(v)$ represent the sets of its input and output edges, respectively. The fan-in and fan-out nodes of $v$ are denoted by $\text{FI}(v)$ and $\text{FO}(v)$, respectively. The logic stage of $v$ is defined as $L(v) = \max_{u \in \text{FI}(v)} L(u) + 1$. By definition, PIs have no fan-in nodes (*i.e.*, $\text{FI}(i) = \emptyset$ for all $i \in I$), and POs have no fan-out nodes (*i.e.*, $\text{FO}(o) = \emptyset$ for all $o \in O$). The edge set $E$ represents the interconnections (nets), and consists of two subsets: $E_u$, representing 2-pin nets, and $E_m$, representing multi-pin nets (nets with more than two pins). For an edge $e \in E$, let $e_s$ denote its source node and $e_t$ the set of its sink nodes. Then, $e \in E_u$ if and only if $|e_t| = 1$, and $e \in E_m$ if and only if $|e_t| > 1$. After inserting splitters for multi-fan-out nets, an extended graph $G'(V', E')$ is constructed, where $V' = V \cup S$, with $S$ denoting the sets of inserted splitters, respectively.

### B. Problem Formulation

RSFQ physical design involves the physical placement and routing of RSFQ circuits, with the primary goal of minimizing circuit area and wirelength while satisfying strict timing constraints. In RSFQ circuits, precise timing alignment is crucial for correct RSFQ operations and is achieved by imposing length-matching constraints on each PTL connection. Since detours only occur in the vertical direction, length-matching is applied exclusively to vertical wire segments. Therefore, the primary objective of pipelined placement with length-matching constraints is to minimize the total vertical wirelength (*TVWL*), which is the sum of the vertical minimum length and the vertical matching length, formulated as:

$$TVWL = \sum_{v \in V} \left[ l_v^{vc} + l_v^{mc} + \sum_{u \in \text{FI}(v)} \left( l_{u,v}^{vd} + l_{u,v}^{md} \right) \right], \quad (4)$$

where $l_v^{vc}$ and $l_v^{mc}$ are the vertical minimum length and vertical matching length of node $v$'s clock connection, and $l_{u,v}^{vd}$ and $l_{u,v}^{md}$

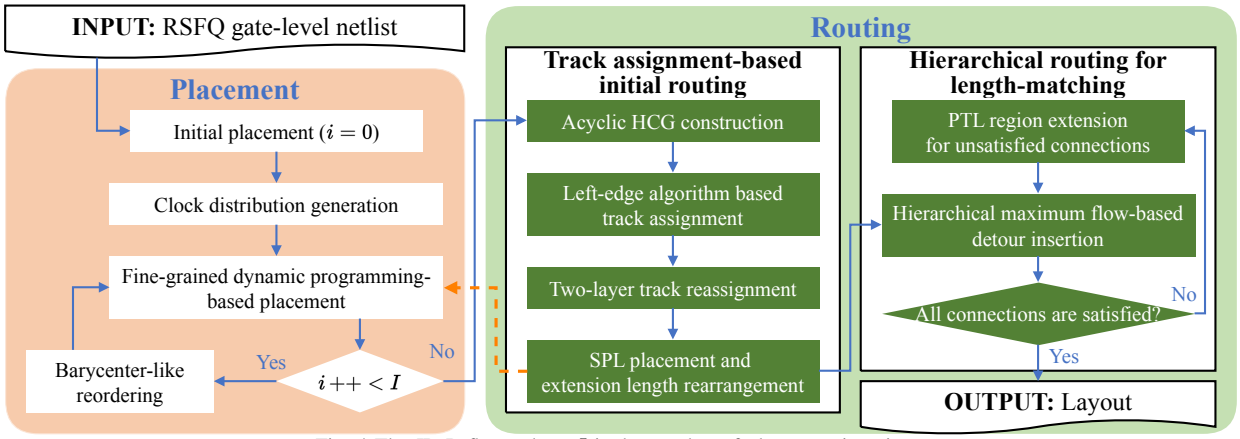



Fig. 4 The JPnR flow, where $I$ is the number of placement iterations.

are the vertical minimum length and vertical matching length of the data connection between $u$ and $v$. The vertical minimum length is the overall vertical Manhattan distance between logic gates, while the vertical matching length signifies the total length of the detours required to meet timing constraints. Fig. 3 shows a partial schematic of the circuit placement, with illustrations of the vertical minimum length for the clock and data connections. To calculate the minimum vertical matching length of data and clock connections, let $l_v^{vd} = \max_{u \in \mathrm{FI}(v)} l_{u,v}^{vd}$, which indicates the time when all data pulses arrive at $v$. When $l_v^{vc} + \Delta l < l_v^{vd}$, node $v$'s clock connection requires detours, which means $l_v^{mc} = l_v^{vd} - l_v^{vc} - \Delta l$ and $l_{u,v}^{md} = l_v^{vd} - l_{u,v}^{vd}$. Otherwise, the data connections between $u \in \mathrm{FI}(v)$ and $v$ require detours, which means $l_{u,v}^{md} = l_v^{vc} + \Delta l - l_{u,v}^{vd}, u \in \mathrm{FI}(v)$ and $l_v^{mc} = 0$.

After completing the placement, the location of each node can be obtained, allowing the calculation of the vertical minimum length and vertical matching length for all connections. Since detours only occur in the vertical direction, the vertical matching length of the connection is its extension length required during routing. Subsequently, the routing width between adjacent gate columns needs to be determined. As shown in Fig. 2, the routing region between two adjacent gate columns is modeled as a rectangular grid, where each grid represents a PTL routing region with unit height and width. The set of nets to be routed is defined as $\{net_1, net_2, \cdots, net_m\}$, where each net $net_i$ connects a source $s_i$ to a set of $n_i$ sinks $\{t_{i,1}, t_{i,2}, \cdots, t_{i,n_i}\}$, represented as $net_i : s_i \rightarrow \{t_{i,1}, t_{i,2}, \cdots, t_{i,n_i}\}$. A connection between the source $s_i$ and a sink $t_{i,j}$ (for $j \in [1, n_i]$) is denoted by $s_i \rightarrow (t_{i,j}, e_{i,j}), j \in [1, n_i]$, where $e_{i,j}$ denotes its required extension length. For example, in Fig. 2(a), the net $s_1 \rightarrow \{t_{1,1}, t_{1,2}\}$ has two connections that require extension lengths of 12 and 2, respectively.

Hence, the length-matching routing problem seeks to minimize this routing width ensuring all connections are successfully routed and satisfy their length-matching constraints. Specifically, the total routing length $d_{i,j}$ for a connection $s_i \rightarrow (t_{i,j}, e_{i,j})$ must adhere to the following equation:

$$d_{i,j} = |s_i - t_{i,j}| + e_{i,j} + w, \tag{5}$$

where $s_i$ and $t_{i,j}$ also denote the vertical positions of the corresponding terminals, and $w$ represents the routing width of the PTL region. Since detours only occur in the vertical direction, extension lengths are always even numbers [13]. In Fig. 2(a), the connection $s_1 \rightarrow (t_{1,1}, e_{1,1})$ (green line) requires a total length of $d_{1,1} = 23$. Given $w = 11$ and $|s_1 - t_{1,1}| = 0$, an extension length of $e_{1,1} = 12$ is necessary to satisfy Equation (5).

Therefore, the physical design problem for RSFQ circuits can be formulated as follows.

- Input:
  1) A path-balanced RSFQ circuit $G(V, E)$.
  2) An RSFQ cell library including the physical and timing parameters of logic gates, splitters, and PTLs.
- Output:
  1) Splitter trees for multi-fan-out nets.
  2) The clock distribution of the circuit.
  3) A legal horizontal and vertical location $(x_v, y_v)$ for $\forall v \in V$.
  4) A legal routing path for $\forall e \in E$.
- Constraints:
  1) Timing constraints defined by Equation (3).
  2) Overlap constraints: For two nodes $u$ and $v$ in the same column, $y_u + h_u \leq y_v$ if $y_u \leq y_v$.
  3) For every node $v \in V$, $y_v \geq 0$ and $y_v + h_v \leq H$, where $H$ is the maximum height among all columns.
  4) The $e_{i,j}$ for $s_i \rightarrow t_{i,j}$ must be rounded to even.
- Goal:
  Our goal is to minimize the circuit area and wirelength while satisfying the timing constraints.

## V. JPnR

To address the physical design challenges in RSFQ circuits, we propose a length-matching placement and routing framework named JPnR, which consists of two key components: (1) a clock-aware length-matching placement method, and (2) a length-matching multi-terminal routing method. An overview of the proposed framework is displayed in Fig. 4. During the RSFQ circuit placement phase, we consider clock tree construction and timing constraints and iteratively optimize the placement result to minimize wirelength. Subsequently,

the routing phase implements the placement of splitters and performs length-matched multi-terminal routing, conforming to a two-layer planar Manhattan routing model.

### A. Placement Method

To address the RSFQ placement problem, we propose a length-matching RSFQ placement algorithm, depicted in the yellow part of Fig. 4. The algorithm comprises four key steps: 1) initial placement, 2) clock distribution generation, 3) fine-grained dynamic programming (DP)-based placement, and 4) barycenter-like reordering. In the initial placement step, logic gates are grouped into columns based on their logic stages, and their initial positions are assigned randomly. A heuristic method is then proposed to generate the clock distribution. Subsequently, a DP-based method is proposed to minimize *TVWL* while assigning the positions of all gates within each column. Finally, a barycenter-like reordering method is presented to adjust gate positions to explore further optimization. Details of each step are elaborated in subsequent subsections.

*1) Initial placement*

For convenience for subsequent clock distribution generation, as shown in Fig. 3, the nodes in the gate column are modeled as composite gates, each consisting of a logic gate and a splitter. The splitter is integrated into each composite gate to facilitate clock pulse propagation to the next column.

Given the gate-level pipeline nature of RSFQ circuits, all nodes are organized into columns in terms of their logic stages, as illustrated in Fig. 3. PIs are assigned to the first column (stage 0), and POs are assigned to the last column (stage $l_{\text{max}}$). Consequently, $L(i) = 0$ for all $i \in I$ and $L(o) = l_{\text{max}}$ for all $o \in O$, and the circuit depth is $d = l_{\text{max}} - 1$. The $i^{\text{th}}$ gate column contains the nodes with logic stage $i$, *i.e.*, $V_i = \{v | L(v) = i, v \in V\}$. Within each gate column, the vertical location of node $v$ is denoted by $y_v$, representing its lower boundary, and its height is indicated by $h_v$. After dividing all nodes into $l_{\text{max}}+1$ gate columns, all nodes within each gate column are shuffled and placed side by side, thereby completing the initial placement.

*2) Clock distribution generation*

After determining the initial placement, a heuristic method is proposed to generate the clock distribution, aiming to minimize the potential extension length required for routing. The strategy involves positioning each node as close as possible to the vertical center of its data inputs. Specifically, within each gate column, nodes are first assigned indices in a top-down order based on their vertical positions. The data inputs of each node are subsequently tracked column by column, and their indices are used to determine the source of the clock pulse. For a node $v$ within the $i^{\text{th}}$ column, the indices of all its data inputs $\text{FI}(v)$ are recorded, *i.e.*, $\{i(u) \mid u \in \text{FI}(v)\}$. The source of the clock pulse for $v$ is then determined as the node within the $(i-1)^{\text{th}}$ column whose index corresponds to the arithmetic mean of these indices, *i.e.*,

$$i(v) = \lfloor \frac{\sum_{u \in \text{FI}(v)} i(u)}{|\text{FI}(v)|} \rfloor. \tag{6}$$

For example, in Fig. 3, node $g_4$ has data inputs from nodes $g_1$ and $g_2$. The indices of $g_1$ and $g_2$ are 1 and 2, respectively. Thus, the clock pulse for $g_4$ is sourced from the node with an index of $\lfloor (1+2)/2 \rfloor = 1$, which is $g_1$. For node $g_5$, which has a single data input $g_3$, its clock pulse is sourced from the node at index $\lfloor 3/1 \rfloor = 3$, corresponding to $g_3$.

*3) Fine-grained DP-based Placement*

Following the generation of the clock distribution, the subsequent objective is to identify the optimal placement of each node in order to minimize *TVWL*. However, this task poses a considerable challenge. When considering only the order of node arrangement, for nodes located within column $V_i$, there exists at least $|V_i|!$ possible arrangements. In a circuit with $n$ logic stages, this leads to a total of $\prod_{i=1}^{n} |V_i|!$ possible arrangements, which becomes computationally intractable for large-scale circuits.

To address this challenge, total contribution of node $v$ to *TVWL* can be defined as:

$$l_v^{sum} = l_v^{vc} + l_v^{mc} + \sum_{u \in \text{FI}(v)} (l_{u,v}^{vd} + l_{u,v}^{md}), \tag{7}$$

Hence, *TVWL* can be written as the sum of $l_v^{sum}$ for all nodes according to Equation (4). We observe that the contribution of nodes within the same column to *TVWL* is independent of each other. This means that the placement of one node does not affect the placement of others within the same column. This observation allows us to reformulate the placement problem as a shortest-path problem by introducing a fixed-order assumption. This way allows us to minimize the sum of $l_v^{sum}$ by identifying the path with the lowest cost in a graph.

To solve this problem efficiently, we propose a DP-based algorithm that optimizes node placement column by column. The algorithm assumes that the positions of nodes in other columns remain fixed, and then determines the vertical position of each node within the current column from bottom to top. For the $i^{\text{th}}$ column $V_i = \{v_1, v_2 \dots v_p \dots v_{|V_i|}\}$, we can set a vertical position range for node $v_p \in V_i$ of $[y_{v_p} - r, y_{v_p} + r]$, where $r$ is the moving radius. Given the fixed order assumption, each node must be placed in locations that leave enough space for other nodes within the range $[0, H]$. This constraint can be expressed as follows:

$$\forall v_p \in V_i, \sum_{j=1}^{p-1} h_{v_j} \le y_{v_p} \le H - \sum_{j=p}^{|V_i|} h_{v_j}. \tag{8}$$

The candidate locations $Y_p$ for node $v_p$ are then defined as:

$$Y_p = \{y_{p,m_p} \mid y_{p,m_p} \in [y_{v_p} - r, y_{v_p} + r] \cap [\sum_{j=1}^{p-1} h_{v_j}, H - \sum_{j=p}^{|V_i|} h_{v_j}]\}, \tag{9}$$

where $m_p = 1, 2, \dots, |Y_p|$, and $\cap$ means the intersection of the ranges. The assignment of $y_{p,m_p}$ to node $v_p$ can ensure sufficient space for other nodes in the same column to be placed, under the fixed order constraint. Below, we will illustrate the details of the DP algorithm for this problem.

The DP-based algorithm constructs a weighted directed graph $G_i$ for each column $V_i$, thereby effectively converting the RSFQ placement problem into a shortest-path problem. As illustrated in Fig. 5, the graph contains vertices $\{s, t\} \cap Y_1 \cap \dots \cap Y_n$, where $n = |V_i|$. The source vertex $s$ connects to each vertex in $Y_1$, while all vertices in $Y_n$ connect to the target vertex $t$. Furthermore, a vertex $y_{p,m_p}$ in $Y_p$ is connected to a vertex $y_{p+1,m_q}$ in $Y_{p+1}$ if and only if $y_{p,m_p} + h_p \le y_{p+1,m_q}$, thus maintaining the fixed order

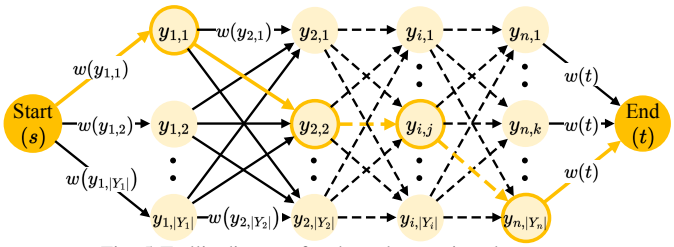


Fig. 5 Trellis diagram for the column-wise placement.

assumption. Each edge connected to $y_{p,m_p}$ has a weight $w(y_{p,m_p})$, which represents the cost of placing node $v_p$ at $y_{p,m_p}$. For columns with logic stages greater than 0, the edge weight is defined as $w(y_{p,m_p}) = l_p^{sum}(y_{p,m_p})$, where $l_p^{sum}(y_{p,m_p})$ corresponds to the total contribution of node $v_p$ to *TVWL* when placed as $y_{p,m_p}$. For the column with a logic stage of 0, which contains only PIs, the weight is calculated as $w(y_{p,m_p}) = \sum_{u \in \text{FO}(v)} l_{v,u}^{vd}$, representing the sum of all vertical minimum lengths of its outputs. The edges connected to $t$ have a weight of zero, *i.e.*, $w(t) = 0$. A path from $s$ to $t$ with the minimum cost, that is, the sum of edge weights, provides the optimal configuration for minimizing *TVWL* for column $V_i$. The vertices along this path indicate the optimal locations for each node $v_p$.

Based on the above analysis, the state transition model of our DP-based algorithm can be formulated as follows:

$$dp[p][y_{p,m_p}] = \begin{cases} \min\limits_{y < y_{p,m_p} - h_{p-1}} \left( dp[p-1][y] + w(y_{p,m_p}), dp[p][y_{p,m_p}] \right), & p > 1 \\ w(y_{p,m_p}), & p = 1 \end{cases}, \quad (10)$$

where $dp$ is an array storing the partial cost sums. For $p = 1$, $dp[p][y_{p,m_p}]$ is initialized with $w(y_{p,m_p})$. For $p > 1$, $dp[p][y_{p,m_p}]$ stores the minimum cost subsum for placing nodes $v_1, \cdots, v_p$ when $v_p$ is placed at $y_{p,m_p}$. The minimum *TVWL* of this column is given by the minimal value in $dp[n]$. By backtracking the $dp$ array, the algorithm identifies the placement associated with the minimum cost.

*4) Barycenter-like reordering*

In the DP-based placement approach, the relative order of nodes within each column remains fixed, which significantly restricts the solution space for minimizing *TVWL*. To overcome this limitation, we propose a reordering method that adjusts the node order across all columns based on the placement results obtained from the DP process. This reordering aims to achieve a placement configuration with reduced *TVWL* when the DP algorithm is reapplied. However, assessing the quality of reordering in terms of *TVWL* requires completing the reordering process and performing subsequent DP calculations, which can be computationally expensive. To mitigate this, we introduce a new target metric related to *TVWL* that can effectively guide the reordering process.

From Equations (4) and (7), we observe that *TVWL* depends on both the vertical minimum length and the vertical matching length, which are dictated by the relative positions of the nodes. This suggests that reordering nodes based on their positional influence and connection lengths could yield significant improvements. Therefore, we propose a barycenter-like reordering algorithm to rearrange nodes within each column. The updated location $y_v$ of a node $v$ is calculated as follows:

$$y_v = \frac{\sum_{u \in \text{FI}} w_{u,v} \cdot y_u}{\sum_{u \in \text{FI}} w_{u,v}}, \quad (11)$$

where $w_{u,v}$ is a weight associated with the connection from node $u$ to node $v$. A higher weight $w_{u,v}$ indicates that the position $y_u$ of node $u$ exerts a stronger influence on the updated position of node $v$, thereby encouraging node $v$ to move closer to $y_u$ in the subsequent placement iteration.

A straightforward approach to selecting the weight $w_{u,v}$ for $u \in \text{FI}(v)$ is to define it as the sum of the vertical minimum length and vertical matching length of the connection, *i.e.*, $w_{u,v} = l_{u,v}^{vd} + l_{u,v}^{md}$. Using this weight, the updated position of node $v$ is influenced by all its input connections, with the connection having the longest length exerting the most significant impact. This ensures that $v$ is placed closer to its most influential input, reducing the overall connection lengths.

However, this basic weighting strategy only accounts for a portion of the delay impact on a node. In practice, the delay affecting a node is determined by the propagation delay from a PI to the node's input pins, rather than solely the delay from its immediate predecessor. To capture this broader delay influence, we introduce a refined weighting strategy. For any $u \in \text{FI}(v)$, the weight $w_{u,v}$ is defined as the delay from a PI, passing through node $u$, and terminating at node $v$. This refined weight accounts for the delay of the entire critical path leading to $v$, rather than just the local connection length. Consequently, the input connection corresponding to the critical path exerts the greatest influence on the updated position of $v$.

This delay-aware weighting strategy encourages the placement of $v$ closer to the node $u$ that lies along the critical path, thereby reducing the delay contribution from the critical input. By minimizing the delay of the critical connection, other input connections to $v$ will typically require shorter vertical matching lengths to satisfy timing constraints. From a broader perspective, this approach reduces *TVWL* by balancing timing requirements across all columns, ensuring that critical paths are prioritized in the placement process.

## *B. Routing Method*

To address the length-matching routing problem, we propose a multi-terminal hierarchical length-matching routing method under the planar Manhattan routing model. This method comprises two key steps: (1) track-assignment-based initial routing and (2) hierarchical routing for length-matching, as illustrated in the green part of Fig. 4. Below, we provide a detailed explanation of each step.

*1) Track-assignment-based Initial Routing*

The first step establishes a preliminary routing path using a track assignment approach and reallocates tracks between the top and bottom layers to ensure a conflict-free layout. As this step does not yet consider length-matching constraints, it serves as a foundation for the subsequent hierarchical routing strategy. Furthermore, the extension lengths of multi-terminal connections are rearranged across the SPL tree to maximize PTL segment sharing.

**① Acyclic VCG construction**

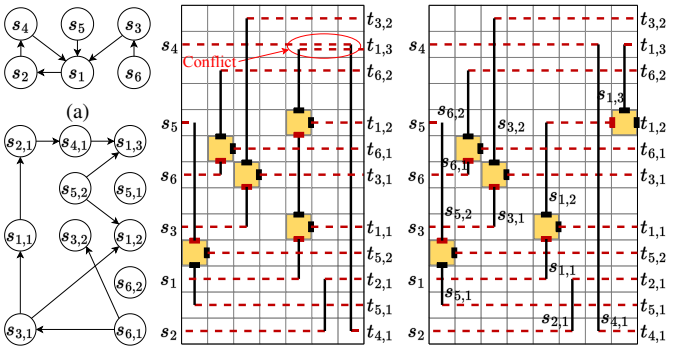



Fig. 6 Track assignment and conflict resolution using the dogleg algorithm in RSFQ circuit initial routing. (a) The VCG for (c). (b) The VCG for (d) after net splitting. (c) Track assignment (black line) and the routing overlap between $s_1$ and $s_4$. (d) Track reassignment based on the dogleg algorithm.

To facilitate routing within the PTL region, a VCG is constructed, where each vertex represents a specific net, labeled as its source label, and edges indicate constraints from left to right. Fig. 6 provides a routing example involving six nets, *i.e.*, $s_1 \rightarrow \{t_{1,1}, t_{1,2}, t_{1,3}\}$, $s_2 \rightarrow \{t_{2,1}\}$, $s_3 \rightarrow \{t_{3,1}, t_{3,2}\}$, $s_4 \rightarrow \{t_{4,1}\}$, $s_5 \rightarrow \{t_{5,1}\}$, and $s_6 \rightarrow \{t_{6,1}, t_{6,2}\}$. Fig. 6(a) depicts the initial VCG corresponding to Fig. 6(c). Notably, vertices $s_1$, $s_2$, and $s_4$ form a cycle. Existing routing approaches commonly employ the left-edge algorithm [42] to sort the vertical segments of all connections and assign them to a minimal number of tracks. However, this algorithm only applies to acyclic VCGs, as it cannot resolve the track assignment order for nodes within a cycle. Arbitrarily selecting a node (e.g. $s_1$) from the cycle for track assignment leads to routing overlap, such as the routing path for $s_1$ and $s_4$ (see Fig. 6(c)). To resolve this, we employ the dogleg algorithm [43] to eliminate cycles in the VCG by splitting the vertical segments of nets. Specifically, for a net $s_i \rightarrow \{t_{i,1}, t_{i,2}, \cdots, t_{i,n_i}\}$, its vertical segment is split into $n_i$ segments: $\{s_i \rightarrow t_{i,1}, t_{i,1} \rightarrow t_{i,2}, \cdots, t_{i,n_i-1} \rightarrow t_{i,n_i}\}$. Fig. 6(d) shows the result of net splitting for Fig. 6(c). For example, the net $s_1 \rightarrow \{t_{1,1}, t_{1,2}, t_{1,3}\}$ is divided into three segments: $\{s_{1,1} \rightarrow t_{1,1}, t_{1,1} \rightarrow t_{1,2}, t_{1,2} \rightarrow t_{1,3}\}$. Fig. 6(b) shows the corresponding acyclic VCG, where each node $s_{i,j}$ represents the $j^{\text{th}}$ vertical segment of the net from bottom to top.

**② Left-edge algorithm based track assignment**

After establishing an acyclic VCG, the left-edge algorithm [42] is employed to minimize the initial routing width. Fig. 7(a) illustrates an acyclic VCG for the routing nets in Fig. 7(b). In Fig. 7(a), an edge between $s_2$ and $s_4$ indicates that the track for $s_2$ must be positioned to the left of $s_4$, as shown in Fig. 7(b). Tracks corresponding to VCG vertices without parent nodes are assigned from left to right. Once a net is assigned to a track, its corresponding vertex is removed from the VCG. Therefore, the number of used tracks can be set as the minimum width $w_{min}^T$ of the PTL region for the given nets. However, considering the routing length required for each net, the minimum width $w_{min}^L$ can be defined as:

$$w_{min}^L = \lfloor \frac{\sum_{i=1}^{m} (l_{net_i}^v + l_{net_i}^e)}{2 * h_{region}} \rfloor, \tag{12}$$

where $h_{region}$ is the height of the PTL region, $l_{net_i}^v$ and

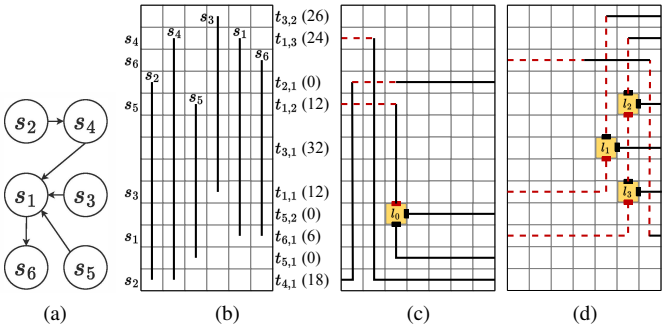



Fig. 7 (a) The VCG constructed for the nets in (b). (b) Source and sink pin positions of connections and the initial track assignment. Result of track reassignment on the (c) top and (d) bottom layer corresponding to (b).

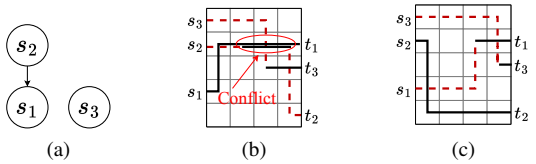



Fig. 8 (a) The VCG constructed for the nets in (b). (b) Horizontal conflict with track assignment model [15]. (c) Result of our two-layer track reassignment.

$l_{net_i}^e$ are the vertical length and extension length of $net_i$, respectively. Therefore, the initial routing width can be set to $\max(w_{min}^T, w_{min}^L)$. Notably, for convenience, the initial width in the figures in this paper is set to $w_{min}^T$.

**③ Two-layer track reassignment**

In the two-layer planar Manhattan routing model, tracks are subsequently reassigned between the top and bottom layers. In existing work [15], tracks in odd (even) columns are assigned from left to right (right to left) on the top (bottom) layer. However, this disrupts the ordered routing of tracks, leading to horizontal conflicts, as shown in Fig. 8(b). To address this, we propose a novel reassignment strategy: the first half of the tracks is allocated to the top layer, while the second half is assigned to the bottom layer, as shown in Fig. 8(c). The results of our track-assignment-based initial routing on both layers are illustrated in Figs. 7(c) to 7(d). Finally, vertical segments are connected to their respective source and sink pins.

**④ SPL placement and extension length rearrangement**

Effective SPL placement in the PTL routing region reduces routing width by optimizing connection distribution across hierarchical levels of SPL trees. Following the track assignment, SPLs are placed at branch points of multi-terminal interconnects. The required extension lengths are rearranged systematically through the SPL tree. To minimize routing width, we employ a greedy optimization strategy that prioritizes detour insertion in lower-level PTL segments first. This maximizes shared routing utilization by leveraging path overlaps across multiple levels.

Fig. 9(c) illustrates the SPL tree for the 4-pin net $s_1 \rightarrow \{t_{1,1}, t_{1,2}, t_{1,3}\}$. Internal nodes represent SPLs, and edges terminating at leaf nodes have initial extension lengths matching their source-to-sink path requirements. Other edges are assigned initial extension lengths based on structural constraints, where the maximum allowable length equals the minimum extension length of subsequent edges sharing the same node. For example, adding 12 units to edge $s_1 \rightarrow l_3$ in Fig. 9(c) adjusts the extension lengths for level 2 to 0 and 0, and for level 3 to 0 and 12. Fig. 9(a) and Fig. 9(b) depict PTL segments

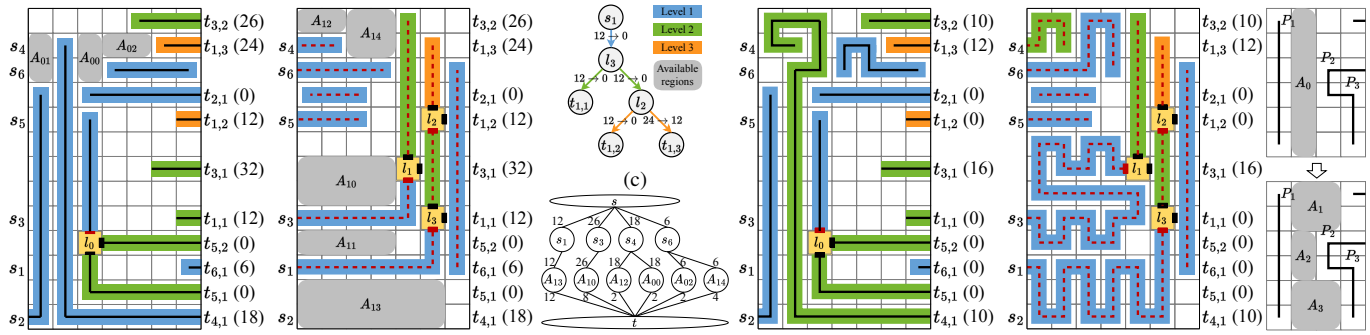



Fig. 9 PTL segments with tree levels and available regions during the first iteration at level 1 on (a) top and (b) bottom layers. (c) Splitter tree and extension lengths rearrangement of $s_1 \rightarrow \{t_{1,1}, t_{1,2}, t_{1,3}\}$. (d) The capacity-constrained flow graph during the first iteration at level 1. Maximum flow-based detour insertion for level 1 on (e) top and (f) bottom layers. (g) Improved method to search available regions.

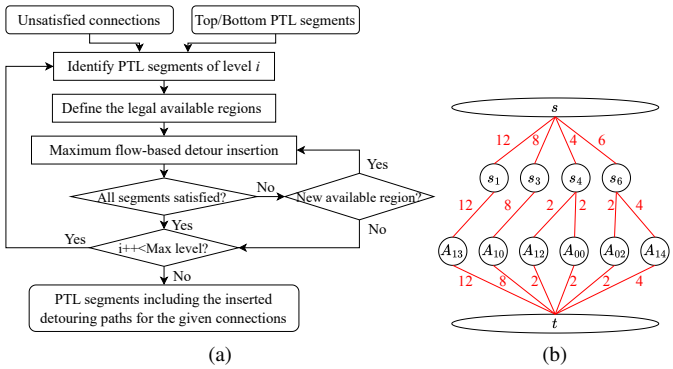



Fig. 10 (a) The flow of hierarchical maximum flow-based detour insertion. (b) Assignment of six legal available regions onto four connections, $s_1$, $s_3$, $s_4$ and $s_6$.

at three levels in the top and bottom layers, respectively. Levels 1, 2, and 3 are marked by blue, green, and orange, while gray blocks indicate available regions for detours for a single PTL segment. The height (width) of these regions must be even for vertical (horizontal) segments.

*2) Hierarchical Routing for Length-Matching*

To integrate detours for PTL connections while ensuring adherence to length-matching requirements derived from the initial routing results, we propose a hierarchical routing strategy. A maximum-flow-based model is applied at each PTL segment level to implement detour insertion, thereby satisfying the length-matching constraints. For connections that fail to meet their specified extension lengths, an iterative PTL region extension strategy is employed to resolve the issue.

**① Hierarchical maximum flow-based detour insertion**

The flow of hierarchical maximum flow-based detour insertion is displayed in Fig. 10(a). To maximize the utilization of unoccupied regions for detour routing, we first identify available regions. For horizontal PTL segments, available regions are identified through a top-to-bottom analysis (*e.g.*, regions $A_{13}$ and $A_{11}$ for segment $s_1 \rightarrow l_3$ in Fig. 9(b)). For vertical PTL segments, regions are identified from left to right (*e.g.*, regions $A_{01}$ and $A_{00}$ for segment $s_4 \rightarrow t_{4,1}$ in Fig. 9(a)). Segments requiring longer extension lengths are prioritized for region assignment to minimize routing width. Regions are not assigned to PTL segments that have already achieved length-matching, such as $s_5 \rightarrow l_0$ in Fig. 9(a). To optimize routing resources, we allow multiple available regions per PTL segment. This contrasts with the approach in [15], which restricts each segment to one region. For instance, in Fig. 9(g), our method enables $P_1$ to use regions $A_1$, $A_2$, and $A_3$, thereby improving the optimization of the maximum flow model. Moreover, mutual exclusion occurs when two regions in opposite directions are available for one PTL segment. Assigning both (e.g., $A_{01}$ and $A_{00}$ to $s_4 \rightarrow t_{4,1}$) makes detour insertion infeasible. Thus, the smaller region is marked as illegal for this iteration. If these two regions are equal in size, one is randomly marked as illegal. In Figs. 9(a) to 9(b), $A_{01}$ and $A_{11}$ are illegal available regions.

After identifying all legally available regions, a capacity-constrained flow graph can be constructed for unsatisfied connections. The maximum flow algorithm [44] is applied to determine the optimal PTL extension lengths for each region. The vertex set of the flow graph consists of a source, a sink, a set of unsatisfied connections, and a set of available regions. A directed edge connects the source to each unsatisfied net, with its capacity set to the required PTL extension length for the net. Each unsatisfied net also has a directed edge to its available regions, with the edge capacity set to the required PTL extension length for the net. All available regions have a directed edge to the sink, with the edge capacity set to the area of the corresponding region. For a detailed explanation of the graph's construction, refer to [15].

The maximum flow algorithm is iteratively executed at each PTL segment level until all connections meet length-matching constraints or no available regions remain for current-level PTL segments. A new capacity-constrained flow graph is constructed in each iteration by identifying new available regions. For example, Figs. 9(a) to 9(b) show available regions during the first iteration at level 1, leading to the flow graph in Fig. 9(d). The maximum flow is then determined to be 30, as shown in Fig. 10(b). The flow on the edge connecting an available region to the sink represents the detour length within the available region. As shown in Figs. 9(e) to 9(f), the PTL segments at level 1 terminate after three iterations, result in a 16-unit extension being added to $s_3 \rightarrow l_1$. This reduces the required extension lengths for $s_3 \rightarrow t_{3,1}$ and $s_3 \rightarrow t_{3,2}$ by 16 units each. Once detours are inserted for a specific level, any remaining unsatisfied PTL segments are promoted to the next level. This process continues until detours are inserted for the

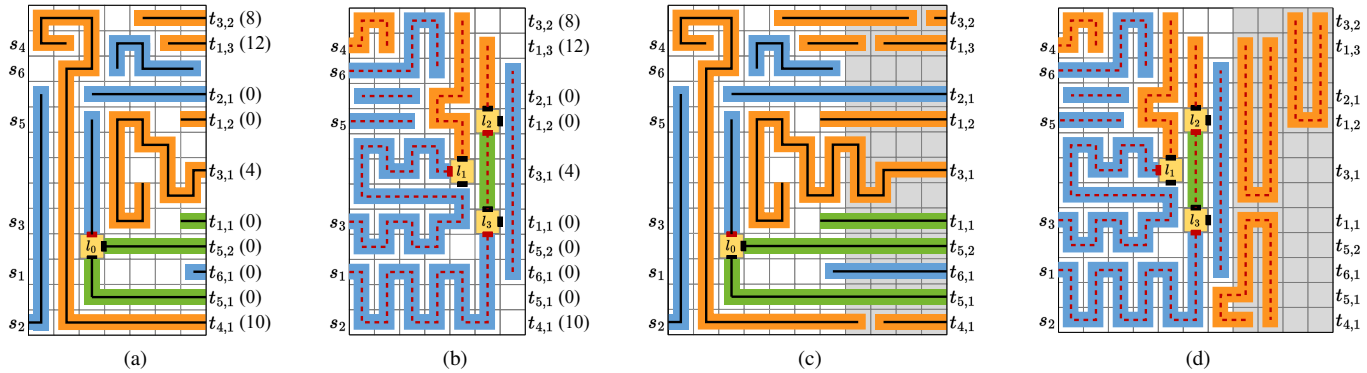



Fig. 11 Results of splitter tree-based hierarchical routing for length-matching on (a) top layer and (b) bottom layer for level 3. Results of the PTL region extension algorithm on (c) top and (d) bottom layer, with the routing width increased by 4.

highest-level PTL segments.

**② PTL region extension for unsatisfied connections**

After routing using the initial width, some connections may still fail to meet length-matching constraints. As shown in Figs. 11(a) to 11(b), the unsatisfied connections include $s_1 \rightarrow t_{1,3}$, $s_3 \rightarrow t_{3,1}$, $s_3 \rightarrow t_{3,2}$, and $s_4 \rightarrow t_{4,1}$. To address this, we propose a PTL region extension strategy. This method expands the PTL routing region by adding two columns at the right boundary and allocating two-unit horizontal segments on the top layer for satisfied connections. Unsatisfied connections are sorted in descending order of their remaining extension lengths. The longest connection is prioritized, with detours inserted in the bottom layer if feasible; otherwise, the pin is connected in the top layer. To determine the PTL insertion direction, we analyze the first two unsatisfied connections in the sorted set. The insertion direction is upward if the first connection has more available space above and satisfies the condition $row_1 + h < row_2$, where $h$ denotes the height of the insertion space, and $row_1$ and $row_2$ are the sink rows of the first and second connections, respectively. Conversely, downward insertion is chosen when more space is available below and the condition $row_1 - h > row_2$ is satisfied.

Next, it will iteratively perform available region search and maximum flow-based detour insertion until all space is utilized or all connections meet their required lengths. To reduce the time overhead, we use a parameter $I_m$ to control the number of iterations. When $I_m = 0$, it will iteratively run until no available regions can be found or all connections are satisfied. If $I_m > 0$, the iterations will terminate after $I_m$ runs or when all connections are satisfied, whichever occurs first. If any connections remain unsatisfied, the process iterates from the initial expansion step to ensure all connections are eventually satisfied. Figs. 11(c) to 11(d) show the final routing results for both layers with $I_m = 0$, demonstrating successful satisfaction of the length-matching constraint after two iterations of PTL region extension.

### C. Time Complexity Analysis

The time complexity of our placement algorithm is analyzed as follows. During the initial placement, each node is assigned a random location, which requires $O(|V|)$ time. During the clock distribution generation, each node records its index in its FO($v$) with an extra $O(|V|)$ space for sorting, and assigning the clock source can be achieved in $O(|V| \cdot \log |V| + |V|)$ time. Next, in the fine-grained DP-based placement, we introduce a hyperparameter $r$. According to Equation (10), the time complexity of this step is $O(|V| \cdot r^2)$. Notably, a large value of $r$ is unnecessary to achieve good performance. Then, the barycenter-like reordering phase updates node locations based on their inputs. Since each node has at most three inputs, updating the location of a single node has linear-time complexity. Additionally, the sorting process in this phase requires $O(|V| \cdot \log |V|)$ time. Overall, the time complexity for one iteration of the placement algorithm is $O(|V| + |V| \cdot r^2 + |V| \cdot \log |V|)$. Since $r$ is a small constant (set to 50 in our experiments), this simplifies to $O(|V| \cdot \log |V|)$. Considering that the algorithm runs for $I$ iterations (set to 100 in our experiments), the total time complexity of our placement algorithm is $O(I \cdot |V| \cdot \log |V|)$.

The time complexity of our routing algorithm is analyzed as follows. When routing between two adjacent gate columns, the construction of an acyclic VCG takes $O(n)$ time, where $n$ is the number of connections within the PTL region. Next, it takes $O(n \log n)$ time to perform track assignment for the vertical segment of these connections using the left-edge algorithm [42], followed by taking $O(n)$ time to assign these vertical segments to feasible tracks on two routing layers. Then, SPL placement and extension length rearrangement take $O(n)$ time. Therefore, the time complexity of the track-assignment-based initial routing is $O(n \cdot \log n)$. Subsequently, constructing the capacity-constrained flow graph for unsatisfied connections and available regions requires $O(n^2)$ time. Additionally, determining the optimal PTL extension lengths for each region using the maximum flow algorithm also takes $O(n^2)$ time. Thus, the time complexity of hierarchical maximum flow-based detour insertion is $O(n^2)$. When unsatisfied connections exist, PTL region extension requires $O(n \cdot \log n)$ time. This process iterates until all connections satisfy the length-matching constraints. Overall, the total time complexity for routing between two adjacent gate columns is $O(N \cdot n^2)$, where $N$ denotes the number of iterations.

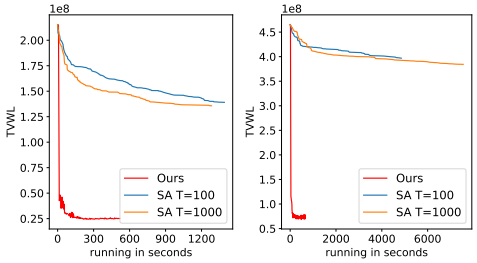
## VI. Experimental Results

### A. Experimental Setup

The JPnR framework was implemented with Python for placement and C++ for routing. Experiments were conducted on an

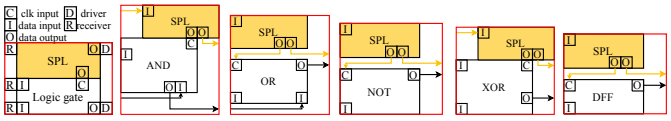
TABLE I Comparison between our placement method and the baseline.

| Circuit | S | G | SA | | | | Our placement | |
|---|---|---|---|---|---|---|---|---|
| | | | T=100 | | T=1000 | | | |
| | | | Time(s) | TVWL | Time(s) | TVWL | Time(s) | TVWL |
| c432 | 27 | 1015 | 6.328 | 538841 | 11.328 | 522028 | 6.156 | 436435 |
| c499 | 12 | 604 | 3.500 | 313421 | 7.203 | 315317 | 3.406 | 251154 |
| c880 | 28 | 1376 | 23.063 | 818409 | 16.421 | 808452 | 8.688 | 601397 |
| c1355 | 12 | 635 | 12.094 | 372912 | 6.593 | 361474 | 3.219 | 272560 |
| c1908 | 22 | 1149 | 16.578 | 716500 | 8.188 | 662313 | 6.981 | 423619 |
| c3540 | 33 | 2279 | 129.031 | 5314391 | 62.797 | 5212187 | 22.828 | 2147667 |
| c5315 | 29 | 5503 | 220.734 | 16526683 | 165.453 | 16012675 | 144.531 | 5843549 |
| c6288 | 76 | 4765 | 21.969 | 4456708 | 22.539 | 4139557 | 16.812 | 4404936 |
| c7552 | 48 | 7571 | 156.156 | 16680602 | 212.938 | 16389954 | 89.000 | 4677287 |
| int2float | 16 | 494 | 14.748 | 272699 | 21.118 | 272973 | 3.844 | 201678 |
| sin | 169 | 16516 | 1390.297 | 139084975 | 1283.406 | 135734764 | 520.593 | 25240196 |
| priority | 215 | 17778 | 1254.225 | 15780332 | 837.100 | 15136645 | 280.921 | 11050073 |
| adder | 255 | 49278 | 3105.877 | 69742105 | 6860.039 | 69716344 | 392.328 | 21588396 |
| multiplier | 257 | 77538 | 17606.856 | 1164910327 | 19539.981 | 1194509219 | 503.46875 | 521097358 |
| max | 207 | 83933 | 4850.391 | 396901851 | 7550.328 | 384329568 | 655.1875 | 77790331 |
| Ave. ratio | / | / | 5.49 | 2.38 | 6.36 | 2.32 | 1 | 1 |




(a) sin (b) max

Fig. 12 *TVWL* changes over time for (a) sin and (b) max circuits, comparing our placement method with SA-based methods at different initial temperatures.




(a) Template (b) AND (c) OR (d) NOT (e) XOR (f) DFF

Fig. 13 Composite gates using the ColdFlux library [17], with a width of 90 µm and built-in PTL drivers/receivers in both logic gates and SPL.

Intel(R) Xeon(R) Gold 5218R CPU with 128 GB of memory. For evaluation, we utilized circuits from the ISCAS85 [11] and EPFL benchmarks [12], which varied in logic stages and gates, with the largest circuit containing up to 207 logic stages and 83933 logic gates. These gates were composite gates, exhibiting a uniform layout width of 90 µm after composition (as shown in Fig. 13). Size and timing parameters for these composite gates were computed using data from the RSFQ cell library [17]. Unless specified otherwise, experimental results in this section were obtained using these composite gates.

### B. Placement Evaluation

To assess the proposed placement method, we conducted a comparative evaluation against the baseline, a modified version of Kito's SA-based method [22]. To ensure a fair comparison, we introduced two key modifications to Kito's original method. First, we updated its objective metric as *TVWL*. Second, we paired two gates within the same column and allowed modifications to multiple pairs in a single placement perturbation, thereby enhancing its efficiency. Regarding the remaining SA settings, we set the number of iterations to 10, the cooling ratio to 0.95, and the termination criterion to a temperature threshold of 0.01. Our placement method was executed with parameters $I = 100$ and $r = 50$, and automatically terminated if no improvement was observed over five consecutive iterations.

TABLE I presents a comparative analysis of placement quality and runtime between our placement method and the modified SA-based method, where "S" and "G" refer to the number of logic stages and gates for each circuit. The initial annealing temperatures for the SA-based method were set to 100 and 1000, respectively. Overall, our placement method demonstrated average reductions of 43.91% and 42.23% in *TVWL*, alongside improvements in runtime of 62.39% and 70.48%, respectively. Furthermore, Fig. 12 visualizes how the *TVWL* evolves during the execution of our placement method and the SA-based method for the sin and max circuits from the EPFL benchmark [12]. The results clearly demonstrate that our placement method achieves rapid convergence to superior solutions compared to the baseline.

### C. Routing Evaluation

We first evaluated our routing algorithm with $I_m = 0$ using randomly generated testcases from [13], which consist of 2-pin connections with pre-allocated SPLs and utilize two routing layers within the PTL region. To create multi-terminal connections, we converted paired adjacent 2-pin connections into 3-pin connections by randomly selecting one source. It is important to note that while JPnR can accommodate SPLs of any size, for a fair comparison with existing routing methods [13]–[16], we assumed in this subsection that splitters occupy one unit within the PTL routing region and are placed in an additional gate column, increasing the routing width by one unit.

TABLE II presents the results of our routing algorithm and the baselines, where "NC" represents the number of connections in a PTL routing region, "H" represents the height of a PTL routing region, "Ave" represents the average extension length of a PTL routing region, "Max" represents the maximum extension length of a PTL routing region, "RW" represents the routing width of a PTL routing region, and "Time" indicates the runtime in seconds for evaluating each case. The results demonstrate that our routing algorithm achieves a smaller routing width in a reasonable runtime. Overall, our proposed algorithm achieved average reductions of 38.20%, 38.29%, 21.52% and 7.38% in routing width compared with Kito's [13], Kou's [14], and Yan's [15], [16] routing algorithms, respectively. Our proposed algorithm outperforms JRouter [45] in all testcases from [13] except one, where the track assignment difference due to the dogleg algorithm [43] causes slightly worse performance.

We further evaluated our routing algorithm using the int2float circuit from the EPFL benchmark [12] to assess its effectiveness in handling VCG cycles. The circuit consists of 16 PTL routing regions (RR1 to RR16), each with a height of 668. The routing results for JRouter and our router are

TABLE II Experimental results of our routing method on the testcases from [13].

| NC | H | Ave | Max | Kito's [13] | | Kou's [14] | | Yan's [15] | | Yan's [16] | | JRouter [45] | | Our router | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | RW | Time(s)[1] | RW | Time(s)[1] | RW | Time(s)[2] | RW | Time(s)[2] | RW | Time(s)[3] | RW | Time(s)[3] |
| 15 | 200 | 43 | 105 | 15 | 0.73 | 15 | 0.23 | 12 | 0.13 | 11 | 0.11 | 9 | 0.02 | 9 | 0.03 |
| 15 | 200 | 44 | 122 | 15 | 1.10 | 17 | 0.10 | 12 | 0.14 | 11 | 0.12 | 9 | 0.04 | 9 | 0.04 |
| 15 | 30 | 8 | 26 | 17 | 2.76 | 17 | 0.03 | 14 | 0.11 | 11 | 0.10 | 10 | 0.02 | 10 | 0.02 |
| 15 | 30 | 18 | 79 | 23 | 8.47 | 23 | 0.07 | 16 | 0.13 | 13 | 0.11 | 11 | 0.02 | 11 | 0.03 |
| 25 | 50 | 12 | 46 | 23 | 4.17 | 24 | 0.14 | 20 | 0.46 | 17 | 0.39 | 17 | 0.03 | 17 | 0.05 |
| 25 | 50 | 16 | 51 | 25 | 108 | 24 | 0.12 | 20 | 0.51 | 18 | 0.41 | 18 | 0.04 | 18 | 0.04 |
| 40 | 80 | 16 | 49 | 29 | 348 | 29 | 0.35 | 24 | 2.17 | 20 | 1.95 | 17 | 0.04 | 17 | 0.07 |
| 40 | 80 | 19 | 56 | 31 | 1346 | 33 | 0.43 | 24 | 2.24 | 21 | 2.02 | 20 | 0.22 | 20 | 0.23 |
| 40 | 200 | 40 | 168 | 29 | 1345 | 30 | 0.59 | 23 | 0.79 | 19 | 0.63 | 19 | 0.05 | 19 | 0.09 |
| 40 | 200 | 42 | 182 | 31 | 1346 | 30 | 0.60 | 23 | 0.84 | 19 | 0.68 | 19 | 0.08 | 21 | 0.12 |
| 50 | 100 | 23 | 99 | 43 | 1346 | 40 | 2.11 | 33 | 4.25 | 27 | 2.73 | 27 | 0.16 | 25 | 0.18 |
| 50 | 100 | 25 | 111 | 44 | 1346 | 40 | 2.09 | 33 | 4.39 | 27 | 2.82 | 24 | 0.16 | 24 | 0.23 |
| Average routing width (RW) ratio | | | | 1.64 | | 1.64 | | 1.29 | | 1.09 | | 1.00 | | 1 | |

[1] Runtime on Intel(R) Xeon(R) 2.60 GHz CPU machine with 128 GB memory.
[2] Runtime on Intel Core i7-3770 3.40 GHz CPU machine with 32 GB memory.
[3] Runtime on Intel(R) Xeon(R) Gold 5218R 2.10 GHz CPU with 128 GB memory.

TABLE III Routing results for int2float.

| R | Nets | Ave | Max | JRouter [45] | Our router |
|---|---|---|---|---|---|
| RR1 | 11 | 233.89 | 579 | 26 | 26 |
| RR2 | 38 | 91.21 | 278 | 37 | 34 |
| RR3 | 70 | 68.56 | 309 | **N/A** | 26 |
| RR4 | 91 | 80.51 | 477 | 37 | 37 |
| RR5 | 108 | 95.68 | 568 | **N/A** | 35 |
| RR6 | 111 | 112.48 | 579 | 40 | 40 |
| RR7 | 95 | 108.22 | 735 | 37 | 37 |
| RR8 | 77 | 91.46 | 422 | 40 | 38 |
| RR9 | 58 | 83.55 | 772 | 24 | 24 |
| RR10 | 43 | 90.88 | 607 | 31 | 31 |
| RR11 | 35 | 60.78 | 668 | 20 | 16 |
| RR12 | 26 | 84.45 | 552 | 19 | 19 |
| RR13 | 21 | 55.13 | 326 | 13 | 13 |
| RR14 | 18 | 77.21 | 425 | 18 | 18 |
| RR15 | 16 | 87.11 | 615 | 11 | 11 |
| RR16 | 14 | 46.00 | 112 | 9 | 9 |

[*] **N/A**: Fails to handle cyclic VCGs, causing an infinite loop and no solution.

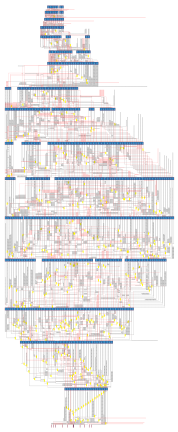


Fig. 14 Layout of int2float.

detailed in TABLE III, where "Nets" refers to the total number of nets, and "Ave" and "Max" represent the average and maximum extension lengths per routing region, respectively. Notably, JRouter failed to route in regions RR3 and RR5 due to VCG cycles, whereas our method successfully resolved these failures while reducing the average routing width by 2.36%. Fig. 14 shows the resulting layout of the int2float circuit, with an area of $13.89\,\mathrm{mm} \times 5.59\,\mathrm{mm}$ ($77.64\,\mathrm{mm}^2$).

### D. JPnR

Accurately estimating routed wirelength during placement is challenging because the SPL tree shape and SPL cell locations are determined only during routing. To address this, incorporating routing information into placement is crucial for accurate wirelength estimation. To mitigate runtime overhead, we use initial routing results to guide the placement optimization process (orange arrow in Fig. 4). We refer to this enhanced placement and routing flow as HQ-JPnR. Specifically, in HQ-JPnR, the placement cost is set to the total wirelength (*TWL*), including both vertical and horizontal wires, obtained after initial routing. $I_m$ is set to 1 in both HQ-JPnR and JPnR. The effectiveness of JPnR and HQ-JPnR for RSFQ physical design is demonstrated through end-to-end results summarized in TABLE IV. The table lists circuit layout width, height, total area, *TWL*, and runtime. On average, HQ-JPnR reduces *TWL* by 4.56% and circuit area by 3.69% compared to JPnR, highlighting the benefits of integrating routing information into placement. Notably, HQ-JPnR requires less runtime on most circuits than JPnR, possibly because the introduction of routing information during placement results in a routing-friendly placement, thereby reducing the iterations of PTL region extension for unsatisfied connections.

## VII. Conclusion

This paper presented JPnR, a length-matching physical design framework for RSFQ circuits. JPnR achieves precise timing alignment by considering both clock and data connections during placement and routing. The placement process employs a heuristic clock distribution method, fine-grained dynamic programming to minimize vertical wirelength, and a barycenter-like reordering for further wirelength reduction. The routing process begins with track assignment using the dogleg and left-edge algorithms to minimize routing width and avoid horizontal conflicts. It then applies a splitter tree-based hierarchical routing algorithm to insert detours for PTL connections and extends routing regions for unsatisfied connections. Our evaluation of JPnR on multiple circuits demonstrated its end-to-end effectiveness, showcasing significant improvements in RSFQ physical design under length-matching constraints.

## References

[1] K. Likharev and V. Semenov, "RSFQ logic/memory family: a new josephson-junction technology for sub-terahertz-clock-frequency digital systems," *IEEE Transactions on Applied Superconductivity*, vol. 1, no. 1, pp. 3–28, 1991.
[2] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-efficient superconducting computing—power budgets and requirements," *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, pp. 1 701 610–1 701 610, 2013.
[3] W. Chen, A. Rylyakov, V. Patel, J. Lukens, and K. Likharev, "Rapid single flux quantum T-flip flop operating up to 770 GHz," *IEEE Transactions on Applied Superconductivity*, vol. 9, no. 2, pp. 3212–3215, 1999.
[4] A. Inamdar, S. S. Meher, B. Chonigman, A. Sahu, J. Ravi, and D. Gupta, "50 GHz operation of RSFQ arithmetic logic unit designed using the advanced design flow and the dual RSFQ/ERSFQ cell library," *IEEE Transactions on Applied Superconductivity*, vol. 33, no. 5, pp. 1–8, 2023.
[5] H. Zha, N. K. Katam, M. Pedram, and M. Annavaram, "HiPerRF: A dual-bit dense storage SFQ register file," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2022, pp. 415–428.
[6] H. Zha, S. Tannu, and M. Annavaram, "SuperBP: Design space exploration of perceptron-based branch predictors for superconducting CPUs,"

TABLE IV Experimental results of JRnR and HQ-JPnR on ISCAS85 and EPFL benchmarks.

| Circuit | JPnR | | | | | HQ-JPnR | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Width(μm) | Height(μm) | Area($\mu m^2$) | TWL | Time(s) | Width(μm) | Height(μm) | Area($\mu m^2$) | TWL | Time(s) |
| c432 | 22860 | 5020 | 114757200 | 5917850 | 288.85 | 21850 | 5030 | 109905500 | 5691130 | 199.92 |
| c499 | 11990 | 4720 | 56592800 | 3290780 | 160.40 | 11610 | 4830 | 56076300 | 3012250 | 80.88 |
| c880 | 26040 | 5890 | 153375600 | 7783860 | 181.97 | 25680 | 5910 | 151768800 | 7669590 | 199.43 |
| c1355 | 12360 | 5180 | 64024800 | 3462150 | 193.66 | 11930 | 5200 | 62036000 | 3382600 | 136.26 |
| c1908 | 22420 | 5750 | 128915000 | 6091080 | 156.99 | 20900 | 5800 | 121220000 | 5803260 | 196.26 |
| c3540 | 51440 | 20480 | 1053491200 | 29223460 | 479.05 | 45450 | 20480 | 930816000 | 27341960 | 750.06 |
| c5315 | 60890 | 29530 | 1798081700 | 77258730 | 10645.23 | 60380 | 29540 | 1783625200 | 75144370 | 10535.61 |
| c6288 | 95110 | 6400 | 608704000 | 47744810 | 19529.02 | 89850 | 6510 | 584923500 | 44413070 | 10345.80 |
| c7552 | 71330 | 23010 | 1641303300 | 60347080 | 1669.20 | 66810 | 23080 | 1541974800 | 59540140 | 2306.39 |
| int2float | 15640 | 5570 | 87114800 | 3010630 | 15.01 | 13570 | 5590 | 75856300 | 2719470 | 54.93 |
| sin | 275630 | 47790 | 13172357700 | 331657650 | 568031.17 | 253030 | 47790 | 12092303700 | 326857350 | 793334.00 |
| priority | 190950 | 9250 | 1766287500 | 133407820 | 40577.93 | 181640 | 9390 | 1705599600 | 128052820 | 16218.46 |
| adder | 214990 | 18190 | 3910668100 | 261324330 | 7154.57 | 230680 | 18090 | 4173001200 | 265749620 | 12714.64 |
| max | 310270 | 51880 | 16096807600 | 611530350 | 1115620.09 | 322330 | 51870 | 16719257100 | 537173150 | 1550504.83 |
| Ave. ratio | 1.05 | 0.99 | 1.04 | 1.05 | 1.11 | 1 | 1 | 1 | 1 | 1 |


in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 599–613.

[7] K. Ishida, M. Tanaka, I. Nagaoka, T. Ono, S. Kawakami, T. Tanimoto, A. Fujimaki, and K. Inoue, "32 GHz 6.5 mW gate-level-pipelined 4-bit processor using superconductor single-flux-quantum logic," in *Symposium on VLSI Circuits (VLSIC)*, 2020, pp. 1–2.

[8] K. Ishida, I. Byun, I. Nagaoka, K. Fukumitsu, M. Tanaka, S. Kawakami, T. Tanimoto, T. Ono, J. Kim, and K. Inoue, "SuperNPU: An extremely fast neural processing unit using superconducting logic devices," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 58–72.

[9] R. Fu, J. Huang, H. Wu, X. Ye, D. Fan, and T.-Y. Ho, "JBNN: A hardware design for binarized neural networks using single-flux-quantum circuits," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3203–3214, 2022.

[10] K. Gaj, E. G. Friedman, and M. J. Feldman, "Timing of multi-gigahertz rapid single flux quantum digital circuits," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 16, no. 2, pp. 247–276, 1997.

[11] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test*, vol. 16, no. 3, pp. 72–80, 1999.

[12] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IEEE/ACM International Workshop on Logic Synthesis (IWLS)*, 2015.

[13] N. Kito, K. Takagi, and N. Takagi, "Automatic wire-routing of SFQ digital circuits considering wire-length matching," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 3, pp. 1–5, 2016.

[14] M. Kou, P.-Y. Cheng, J. Zeng, T.-Y. Ho, K. Takagi, and H. Yao, "Splitter-aware multiterminal routing with length-matching constraint for RSFQ circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 11, pp. 2251–2264, 2021.

[15] J.-T. Yan, "Length-matching-constrained region routing in rapid single-flux-quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 5, pp. 945–956, 2020.

[16] J.-T. Yan, "Via-minimization-oriented region routing under length-matching constraints in rapid single-flux-quantum circuits," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 29, no. 6, pp. 1257–1270, 2021.

[17] L. Schindler and T. Hall, "RSFQ cell library," https://github.com/sunmagnetics/RSFQlib, version: 3.0, Release date: 21 March 2023.

[18] G. Pasandi and M. Pedram, "PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 4, pp. 1–14, 2019.

[19] R. Fu, J. Huang, and Z.-M. Zhang, "Equivalence checking for superconducting RSFQ logic circuits," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2021, pp. 51–56.

[20] M. Zhou, R. Fu, R. Zhang, X. Ye, T.-Y. Ho, and J. Huang, "An optimal DFF-oriented technology legalization algorithm for rapid single-flux-quantum circuits," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2025.

[21] T. Jabbari, G. Krylov, S. Whiteley, E. Mlinar, J. Kawa, and E. G. Friedman, "Interconnect routing for large-scale RSFQ circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–5, 2019.

[22] 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 Transactions on Applied Superconductivity*, vol. 28, no. 4, pp. 1–5, 2018.

[23] J.-T. Yan, "Tree-based clock distribution of multiple-stage pipelined architecture in rapid single-flux-quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 4, pp. 1090–1102, 2022.

[24] J. Huang, R. Fu, X. Ye, and D. Fan, "A survey on superconducting computing technology: Circuits, architectures and design tools," *CCF Transactions on High Performance Computing*, vol. 4, no. 1, 2022.

[25] S. Razmkhah, R. S. Aviles, M. Li, S. Gupta, P. A. Beerel, and M. Pedram, "Challenges and unexplored frontiers in electronic design automation for superconducting digital logic," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–6.

[26] G. Pasandi and M. Pedram, "qSeq: Full algorithmic and tool support for synthesizing sequential circuits in superconducting SFQ technology," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 133–138.

[27] R. Fu, O. Chen, N. Yoshikawa, and T.-Y. Ho, "Exact logic synthesis for reversible quantum-flux-parametron logic," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–9.

[28] R. Fu, M. Wang, Y. Kan, O. Chen, N. Yoshikawa, B. Yu, and T.-Y. Ho, "Buffer and splitter insertion for adiabatic quantum-flux-parametron circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 44, no. 3, pp. 975–988, 2025.

[29] R. Fu, R. Wille, N. Yoshikawa, and T.-Y. Ho, "Efficient cartesian genetic programming-based automatic synthesis framework for reversible quantum-flux-parametron logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–12, 2025.

[30] M. Munir, A. Gopikanna, A. Fayyazi, M. Pedram, and S. Nazarian, "qMC: a formal model checking verification framework for superconducting logic," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2021, pp. 259–264.

[31] B. Zhang, M. Li, and M. Pedram, "qSSTA: A statistical static timing analysis tool for superconducting single-flux-quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 7, pp. 1–12, 2020.

[32] C. J. Fourie, K. Jackman, M. M. Botha, S. Razmkhah, P. Febvre, C. L. Ayala, Q. Xu, N. Yoshikawa, E. Patrick, M. Law, Y. Wang, M. Annavaram, P. Beerel, S. Gupta, S. Nazarian, and M. Pedram, "ColdFlux superconducting EDA and TCAD tools project: Overview and progress," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–7, 2019.

[33] R. Fu, O. Chen, B. Yu, N. Yoshikawa, and T.-Y. Ho, "DLPlace: A delay-line clocking-based placement framework for AQFP circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–8.

[34] S. N. Shahsavani and M. Pedram, "A minimum-skew clock tree synthesis algorithm for single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 8, pp. 1–13, 2019.

[35] G. Pasandi and M. Pedram, "An efficient pipelined architecture for superconducting single flux quantum logic circuits utilizing dual clocks," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 2, pp. 1–12, 2019.

[36] X. Li, M. Pan, T. Liu, and P. A. Beerel, "Multi-phase clocking for multi-

threaded gate-level-pipelined superconductive logic," in *IEEE Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 62–67.
[37] J.-T. Yan, "Fixed-order placement of pipelined architecture in rapid single-flux-quantum circuits," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 30, no. 10, pp. 1519–1532, 2022.
[38] K. Kitamura, T. Kawaguchi, and N. Takagi, "Wire length-matching aware placement method for rapid single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 33, no. 5, pp. 1–5, 2023.
[39] S. Chen, R. Fu, J. Huang, Z. Zhang, X. Ye, T.-Y. Ho, and D. Fan, "JPlace: A clock-aware length-matching placement for rapid single-flux-quantum circuits," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–6.
[40] P.-Y. Cheng, K. Takagi, and T.-Y. Ho, "Multi-terminal routing with length-matching for rapid single flux quantum circuits," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.
[41] T.-R. Lin, T. Edwards, and M. Pedram, "qGDR: A via-minimization-oriented routing tool for large-scale superconductive single-flux-quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 7, pp. 1–12, 2019.
[42] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *ACM/IEEE Design Automation Conference (DAC)*, 1971, pp. 155–169.
[43] D. N. Deutsch, "A "Dogleg" channel router," in *ACM/IEEE Design Automation Conference (DAC)*, 1976, pp. 425–433.
[44] E. Dinic, "An algorithm for the solution of the max-flow problem with the polynomial estimation," *Doklady Akademii Nauk SSSR*, vol. 194, no. 4, pp. 1277–1280, 1970.
[45] X. Chen, R. Fu, J. Huang, H. Cao, Z. Zhang, X. Ye, T.-Y. Ho, and D. Fan, "JRouter: A multi-terminal hierarchical length-matching router under planar manhattan routing model for RSFQ circuits," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2023, pp. 515–520.

**Rongliang Fu** received his B.S. degree in software engineering from Northwestern Polytechnical University, Xi'an, China, in 2018, and his M.S. degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation and computer architecture.

**Minglei Zhou** received his B.S. degree in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2024. He is currently pursuing the M.S. degree at SKLP, Institute of Computing Technology, CAS, and the University of Chinese Academy of Sciences. His research interests include electronic design automation and computer architecture.

**Siyan Chen** received his M.S. degree in Computer Science from ShanghaiTech University, China. He is currently an engineer at UNIVISTA.

**Xinda Chen** received his M.S. degree in Computer Science from ShanghaiTech University, China. He is currently an engineer at UNIVISTA.

**Junying Huang** received her Ph.D. degree in microelectronics and solid-state electronics from the University of Chinese Academy of Sciences in 2016. She is currently an associate professor with the Department of High-Throughput Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include superconducting RSFQ logic, computer architecture, electronic design automation, and hardware security.

**Xiaochun Ye** received the Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences (CAS), in 2010. He is currently a Professor and Director of the High-Throughput Computer Research Center at the Institute of Computing Technology, CAS. His main research interest is computer architecture.

**Zhimin Zhang** received the Ph.D. degree in computer architecture from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), in 2003. He is currently a Professor at ICT, CAS. His main research interests include designs for many-core processors, system-on-chip (SoC), superconducting processors, and superconducting computing systems.

**Tsung-Yi Ho** (F'24) is a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.