JSPlace: A Shape-Controllable and Length-Matching Placement for Rapid Single-Flux-Quantum Circuits

Rongliang Fu[1†], Minglei Zhou[2,3†], Huilong Jiang[3], Da Wang[3], Fei Wang[4], Huawei Cao[3], Junying Huang[3*]
[1]*Department of Computer Science and Engineering, The Chinese University of Hong Kong*, Hong Kong, China
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences*, Beijing, China
[3]*SKLP, Institute of Computing Technology, CAS*, Beijing, China
[4]*School of Integrated Circuits, Shandong University*, Jinan, China

Abstract

Superconducting Rapid Single Flux Quantum (RSFQ) logic is a promising candidate for high-performance computing due to its ultra-high speed and ultra-low power. Since most RSFQ logic gates require synchronization with a single clock pulse, RSFQ circuits have a clock-driven gate-level pipelined architecture in nature. However, the gate-level pipelined architecture leads to layouts with a high width-to-height ratio, especially in large circuits, because the layout width increases with the number of logic stages while the height is determined by the tallest column. To address this, we propose JSPlace, a shape-controllable and length-matching placement algorithm for RSFQ circuits. Our algorithm folds the multi-stage pipelined placement into three segments and merges columns via mixed-integer linear programming (MILP) to achieve the target width-to-height ratio. Subsequently, dynamic programming determines the vertical positions of nodes within each column, followed by connectivity-driven repositioning to minimize total vertical wirelength. Experimental results on ISCAS85 and EPFL benchmarks demonstrate that JSPlace achieves effective control of layout width-to-height ratio while significantly outperforming the state-of-the-art in layout area and wirelength.

1 Introduction

The superconductor-based rapid single-flux-quantum (RSFQ) technology [18] has emerged as an attractive candidate for future high-performance computing systems, owing to its ultrafast operational speed and ultralow power consumption. In RSFQ circuits, Josephson junctions (JJs) encode digital information via single flux quantum pulses [4]. Such circuits can achieve operating frequencies in the tens of gigahertz range [19] [2], while exhibiting switching energy as low as 1×10^{-19} J/bit [14, 21], which is two to three orders of magnitude lower than that of CMOS technology. These excellent properties make RSFQ circuits promising for high-performance microprocessors [8], energy-efficient qubit controllers [16], and power-efficient neural network accelerators [15].

Despite recent advancements, the unique physical characteristics of RSFQ circuits preclude the direct application of conventional CMOS-oriented electronic design automation (EDA) tools for physical design. In RSFQ circuits, most logic gates are driven by a clock pulse, enabling a gate-level pipelined architecture that clarifies timing constraints among gates and simplifies routing [5]. However, this architecture also introduces constraints for placement: gates must be arranged column-wise according to their logic stages and placed in appropriate positions [6, 10]. The height of a gate-level pipelined layout is determined by the tallest column, while its width increases with the number of logic stages. To satisfy strict timing constraints, numerous delay flip-flops (DFFs) must be inserted, adding clock and data routing overhead and substantially increasing inter-column routing widths. Increasing logic stages in large-scale RSFQ circuits accumulate inter-column routing width, producing elongated layouts with high width-to-height ratios. For example, a 128-bit adder with 255 logic stages exhibits a layout ratio approaching 13:1 [9]. Such an imbalanced aspect ratio poses a major challenge to the manufacturing yield of RSFQ circuits, as practical chip integration requires precise control over layout geometry.

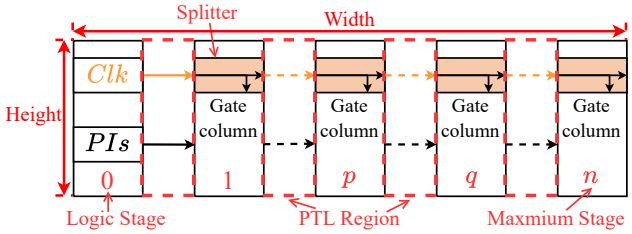

Figure 1. The n-stage pipelined placement of an RSFQ circuit, where n is the maximum logic stage of the RSFQ circuit.

Previous RSFQ placement studies focus on translating timing constraints into length-matching constraints and reducing routing width by minimizing wirelength. Kito *et al.*[17] proposed a simulated annealing (SA)-based algorithm that neglects the length matching of clock signals and exhibits high computational complexity for large-scale circuits. Yan *et al.*[22] proposed a fixed-order placement algorithm to minimize the routing area in pipelined RSFQ circuits. By fixing the relative positions of logic gates, the algorithm reduces the feasible solution space and achieves a faster runtime than the SA-based algorithm. However, it also does not consider timing alignment in RSFQ circuits. Chen *et al.*[6] proposed a clock-aware length-matching placement method that integrates clock tree construction into placement and considers delay matching between data and clock signals, significantly reducing the vertical wirelength required to meet timing constraints. However, all existing RSFQ placement methods only focus on minimizing wirelength under length-matching constraints to reduce the area of the circuit layout, without providing explicit control over the width-to-height ratio. Therefore, new

† Equal contribution. * Corresponding author: huangjunying@ict.ac.cn.

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804054

RSFQ placement methods must be designed to effectively control the width-to-height ratio of the circuit layout while minimizing wirelength under timing constraints.

This paper proposes a shape-controllable and length-matching placement algorithm that satisfies timing constraints while minimizing total wirelength and enabling explicit control over the layout width-to-height ratio. Overall, our contributions are as follows:

- To our knowledge, we propose JSPlace, the first shape-controllable and length-matching placement algorithm for RSFQ circuits.
- We fold the multi-stage pipelined placement into three segments based on the target width-to-height ratio to form a folded placement preserving input and output directions.
- We propose an MILP-based column merging algorithm that merges columns from different segments to balance column heights and minimize total vertical wirelength.
- We propose vertical placement refinement and connectivity-driven repositioning methods to determine the optimal intra-column position of each gate.
- On the ISCAS85 and EPFL benchmarks, JSPlace achieves effective control of the width-to-height ratio while reducing the total vertical wirelength by 21.36% and 46.43%, and layout area by 15.37% and 45.65%, respectively, compared to state-of-the-art.

2 Preliminary

2.1 RSFQ Logic

RSFQ logic is a representative ultra-fast and low-power superconducting circuit technology that employs the single flux quantum (SFQ) as the information carrier. In RSFQ circuits, logical values '1' and '0' are encoded by the presence or absence of single SFQ pulses, which are generated and processed using JJs and propagated through passive transmission lines (PTLs). Unlike conventional CMOS circuits, RSFQ logic gates exhibit limited pulse-driving capability, as each gate can drive only one subsequent gate, thereby necessitating unclocked splitter trees to achieve multiple fanouts. Most RSFQ logic gates require a clock pulse to transfer the stored data pulses to adjacent gates. This synchronization naturally enables a gate-level pipelined architecture for RSFQ circuits.

Based on the gate-level pipelined architecture, RSFQ circuits are divided into multiple logic stages. The logic stage of a gate is defined as the maximum number of clocked gates from any primary input (PI) to that gate. To ensure correct operation of RSFQ logic gates, all paths from any PI to a specific logic gate must traverse an equal number of clocked gates, a condition known as the path balancing constraint. Achieving path balancing requires the insertion of numerous DFFs, which introduces additional power and routing overhead.

After DFF insertion, all logic gates sharing the same logic stage are grouped into one column, as shown in Fig. 1. These columns are arranged from left to right according to increasing logic stages, forming an n-stage pipelined placement, where n denotes the maximum logic stage of the RSFQ circuit. The spacing between adjacent columns defines n PTL routing regions, causing the layout width to grow with the number of logic stages. As the number of logic stages increases, the cumulative routing width of these PTL regions leads to an excessively large width-to-height ratio of the n-stage pipelined placement, posing a major challenge to achieving shape-controllable placement in RSFQ physical design.

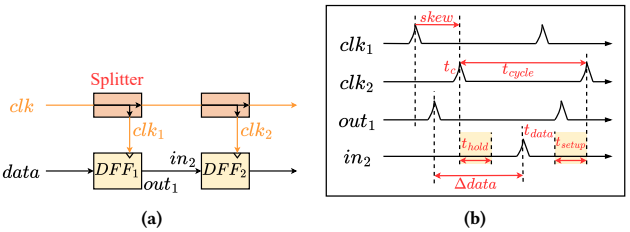

Figure 2. Timing constraints in concurrent-flow clocking-based RSFQ circuits. (a) An RSFQ sub-circuit. (b) Timing of the gates in (a).

2.2 Timing Constraints

To achieve the highest possible clock frequency, RSFQ circuits employ a clocking scheme that is fundamentally different from that of CMOS circuits. Concurrent-flow clocking [11] is a widely used clocking scheme in RSFQ design, where clock and data propagate in the same direction. In this scheme, the data pulse arrives shortly after the clock pulse and is logically processed during the subsequent clock cycle to generate an output. As shown in Fig. 2, concurrent-flow clocking enables clock (clk_2) and data pulses (in_2) to travel in the same direction and exploits clock skew ($skew$) to compensate for data transmission delay ($\Delta data$), thereby hiding propagation delays. To ensure correct circuit operation, the arrival times of clock and data pulses must satisfy the following timing constraint:

$$t_c + t_{hold} < t_{data} < t_c + t_{cycle} - t_{setup}, \tag{1}$$

where t_c and t_{data} represent the arrival times of the clock and data pulses at a logic gate, respectively. t_{cycle} represents the cycle of the clock pulse, while t_{setup} and t_{hold} correspond to the setup and hold times required for correct pulse capture.

In RSFQ circuits, clock pulses must be delivered precisely to nearly all logic gates, significantly increasing the complexity of the clock network. Previous studies [17, 23] have proposed various tree-based clock distribution methods to reduce routing and area overhead. This paper employs Yan's clock tree structure [23], where the primary clock pulse is divided by splitter trees into multiple pulses and propagated to the next column, as shown in Fig. 1. Each clocked logic gate forms a composite gate with a splitter that receives a clock pulse while simultaneously propagating it to the adjacent column.

3 Layout Shape and Problem Formulation

3.1 Terminology

For an n-stage RSFQ logic netlist comprising logic gates, PIs, and primary outputs (POs), the placement can be structured as an n-stage pipelined architecture represented by a directed graph $N(V, E)$, where n denotes the maximum logic stage. The node set is defined as $V = G \cup I \cup O$, where G denotes the set of logic gates, and I and O represent the PIs and POs, respectively. The logic stage of a node $v \in V$ is defined as $L(v) = \max_{u \in DI(v)} L(u) + 1$, where $DI(v)$ is the set of nodes that serve as data inputs to node v. Since all PIs arrive within the same clock cycle, $L(v) = 0, \forall v \in I$. Nodes with the same logic stage are placed within the same column. Thus, the n-stage pipelined placement comprises $n + 1$ columns $C_0, C_1, ..., C_n$. For each column i, $C_i = \{v | L(v) = i, v \in V\}$. $y(v)$ denotes the vertical position of a node within its column, and $h(v)$ represents the physical height.

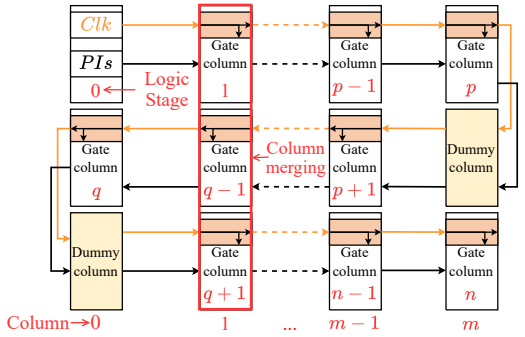

Figure 3. The m-column folded placement of an RSFQ circuit.

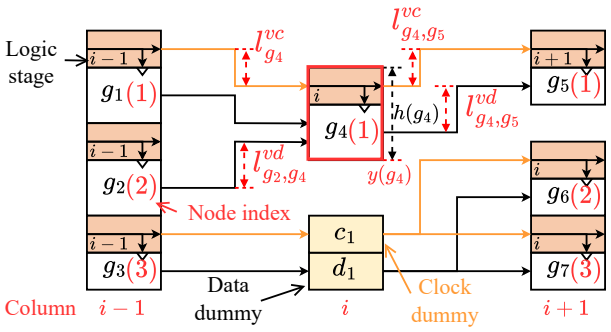

Figure 4. Partial schematic of the folded placement. Node g_4 with logic stage i has height $h(g_4)$ and vertical position $y(g_4)$. Its data inputs originate from g_1 and g_2, and its clock input is provided by g_1, *i.e.*, $DI(g_4) = \{g_1, g_2\}$ and $CI(g_4) = \{g_1\}$. Both its data and clock outputs flow to g_5 *i.e.*, $DO(g_4) = CO(g_4) = \{g_5\}$.

3.2 Problem Formulation

The final area and width-to-height ratio of an RSFQ circuit are determined by the height and width of its layout. The height H_p of the layout is set by the tallest gate column in an n-stage pipelined placement, while the total routing width of the PTL regions between adjacent columns determines both the final layout area and its width-to-height ratio. In an n-stage pipelined placement, the total routing width of the PTL regions increases with the number of logic stages. As the circuit scale expands, cumulative routing width leads to an elongated layout with a high width-to-height ratio.

Therefore, the primary objective of shape-controllable RSFQ placement is to achieve a target width-to-height ratio under timing constraints while minimizing the layout area. To achieve the target aspect ratio r_t while preserving the input and output directions of RSFQ circuits, the n-stage pipelined placement can be folded at columns p and q into three segments: $S_1 = \{C_i | 0 \le i \le p\}$, $S_2 = \{C_i | p+1 \le i \le q\}$, and $S_3 = \{C_i | q+1 \le i \le n\}$, where the selection of p and q is detailed in Section 4.2. As shown in Fig. 3, merging logic gates from these segments into new columns $C'_0, C'_1, ..., C'_m$ produces an m-column folded placement with fewer columns but higher height, where m is the maximum number of columns among S_1, S_2, and S_3. For segments with fewer than m columns, data and clock dummies are introduced to represent cross-column connections. This folding reduces the number of PTL routing regions while increasing column height, thereby reducing total routing width and mitigating excessive width-to-height ratios.

After obtaining the m-column folded placement, wirelength minimization under timing constraints is required to ensure proper circuit operation and reduce layout area. Because the delay of a PTL routing cell scales linearly with its length, the timing constraint (Equation (1)) can be transformed into a length-matching constraint:

$$l_c + l_{hold} < l_{data} < l_c + l_{cycle} - l_{setup}, \tag{2}$$

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_{cycle}, and l_{setup} denote the PTL lengths that match the delays of t_{hold}, t_{cycle}, and t_{setup}, respectively. Therefore, data and clock connections require detoured routing to precisely adjust the arrival times of clock and data pulses for timing alignment. Since the connections between adjacent columns have a uniform horizontal length, the minimization of wirelength focuses on minimizing the total vertical wirelength (*TVWL*):

$$TVWL = \sum_{v \in V} \left[l^{vc}_v + l^{mc}_v + \sum_{u \in DI(v)} \left(l^{vd}_{u,v} + l^{md}_{u,v} \right) \right], \tag{3}$$

where l^{vc}_v and l^{mc}_v are the vertical minimum length and vertical matching length of the clock connection between node v and its clock source, and $l^{vd}_{u,v}$ and $l^{md}_{u,v}$ are the vertical minimum length and vertical matching length of the data connection between u and v. The vertical minimum length is the vertical Manhattan distance between logic gates, and the vertical matching length is the detour length required to meet the length-matching constraint.

Consequently, the shape-controllable and length-matching RSFQ placement problem is formalized as folding the n-stage pipelined placement into three segments S_1,S_2, and S_3 according to r_t to obtain an m-column folded placement. In the m-column folded placement, columns from different segments are merged, and each logic gate is assigned a legal vertical position to minimize *TVWL* under length-matching constraints, thereby reducing layout area and achieving the target width-to-height ratio.

4 JSPlace

To solve the above RSFQ placement problem, we propose JSPlace, a shape-controllable and length-matching RSFQ placement algorithm that achieves the target aspect ratio while minimizing total vertical wirelength. The algorithm comprises five main steps: 1) initial placement; 2) MILP-based column merging; 3) clock distribution generation; 4) vertical placement refinement; and 5) connection-driven repositioning. Details of each step are given in subsections.

4.1 Initial Placement

During the initial placement phase, each node v is assigned to a column according to its logic stage $L(v)$ and is given a random vertical position $y(v)$, forming an n-stage pipelined placement (Fig. 1). The n-stage pipelined placement contains n PTL regions $RR_1, RR_2, \ldots, RR_n$. This paper assumes that PTL regions are routed under length-matching constraints in a two-layer Manhattan routing model [7]. Thus, the estimated PTL region width w_i of each PTL region RR_i is calculated as $w_i = \max\left(w^L_i, \frac{VL_i}{2H_p}\right)$, where w^L_i is the number of routing tracks

obtained by the left-edge algorithm [13], and VL_i represents the total vertical wirelength of RR_i.

4.2 MILP-based Column Merging

To achieve the target width-to-height ratio r_t while preserving the input-output direction of the circuit, the initial placement can be folded at columns p and q into three segments S_1, S_2 and S_3 with PTL region widths $W_1 = \sum_{i=1}^{p} w_i$, $W_2 = \sum_{i=p+1}^{q} w_i$, and $W_3 = \sum_{i=q+1}^{n} w_i$, respectively. The three-segment folding enables a compact U-shaped layout that maintains left-to-right signal flow: two segments cannot preserve both input and output directions, while more than three segments would increase routing complexity without significant benefits. The width-to-height ratio of the folded placement can be estimated as $r = \frac{\max(W_1,W_2,W_3)}{3H_p}$, where H_p denotes the maximum column height in the n-stage pipelined placement. Thus, the optimal $p, q = \arg\min |r - r_t|$. By merging columns from S_1, S_2 and S_3, the n-stage pipelined placement can be folded into m-column folded placement. For segments containing fewer than m columns, gates with the same logic stage can be assigned to different columns, enhancing the flexibility of placement. In the m-column folded placement, nodes are reassigned to merged columns, which determine both the height of the folded placement and the vertical minimum lengths of data connections.

To balance column heights and minimize vertical wirelength, we propose an MILP-based method for column-by-column reassignment of logic gates. For each column i, let V_i^{cand} denote the set of candidate nodes that can be assigned to column i. For $v \in V_i^{cand}$, a binary variable $x(v)$ is introduced to indicate whether v is assigned to column i. If $x(v) = 1$, node v is assigned to the current column; otherwise, it is included in the candidate set for the next column, and a dummy for data propagation is inserted in the current column. For the initial column 0, $V_0^{cand} = I \cup V_q$. For $i \geq 1$, V_i^{cand} is the set of nodes connected to nodes assigned in column $i-1$. As shown in Fig. 4, once column $i-1$ is fixed , it follows that $V_i^{cand} = \{g_4, g_6, g_7\}$. Since g_6 is not assigned to column i, a data dummy (d_1) is inserted to represent the data connection of g_6 traversing column i, with dummy heights h_d equal to the physical size of the PTL. Therefore, the height of column i can be expressed as

$$H_i = \sum_{v \in V_i^{cand}} h(v) \cdot x(v) + h_d \cdot (1 - x(v)). \tag{4}$$

To obtain an optimal reassignment, we formulate the following mathematical model for column i:

$$\min \quad |H_i - H_{avg}| + \sum_{v \in V_i^{cand}} \sum_{u \in DI(v)} |y(u) - y(v)|, \tag{5}$$

$$s.t. \quad 0 \leq y(v) \leq 3H_p, \forall v \in V_i^{cand} \tag{6}$$

$$y(u) + h(u) - y(v) \leq M \cdot (1 - b_{u,v}), \forall u, v \in V_i^{cand} \tag{7}$$

$$y(v) + h(v) - y(u) \leq M \cdot b_{u,v}, \forall u, v \in V_i^{cand}, \tag{8}$$

where the objective function is to balance column heights and minimize the vertical wirelength of data connections. H_{avg} denotes the average column height, calculated as $\frac{\sum h(v)}{m}, v \in V$. M denotes a sufficiently large constant set to $3H_p$ to ensure constraint validity,

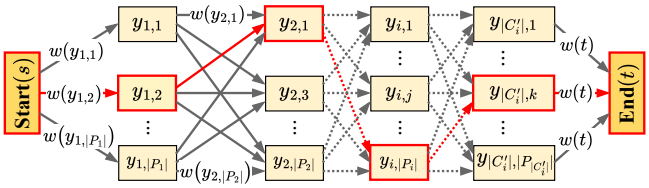

Figure 5. Vertical placement refinement of the i-th column, where the red line highlights the optimal placement solution.

and $b_{u,v}$ is a binary variable indicating the relative vertical order of nodes u and v, where $b_{u,v} = 1$ indicates that node u is placed above node v. Equation (7) and Equation (8) ensure that any two nodes assigned to the same column do not vertically overlap. The absolute value terms in Equation (5) are linearized using standard auxiliary variable transformations to convert the problem into a standard MILP formulation. After solving the problem with an MILP solver, the column and vertical positions of the logic gates in the folded placement are determined.

4.3 Clock Distribution Generation

Following the column merging, we propose a heuristic method for clock distribution minimizing the potential vertical matching length required for timing alignment. The method aims to place the clock input of each node as close as possible to the vertical center of its data inputs, thereby reducing the vertical distance that clock signals must traverse. For a node v with $L(v) = i$, all nodes u satisfying $L(u) = i - 1$ are sorted by their vertical positions $y(u)$ and assigned sequential indices. The clock pulse source for v is defined as the node corresponding to the average of the indices of its data inputs $u \in DI(v)$. As shown in Fig. 4, node g_4 receives data inputs from g_1 and g_2, which have indices 1 and 2, respectively. Thus, the clock pulse of g_4 is sourced from node g_1 with index $\lfloor (1+2)/2 \rfloor = 1$. For node g_6 with a single data input g_3, the clock pulse is sourced from node g_3 with index $\lfloor 3/1 \rfloor = 3$. The PTL segment crossing column i is represented by a clock dummy c_1 for clock propagation.

4.4 Vertical Placement Refinement

After generating the clock distribution, the next objective is to optimize the vertical position of each node to minimize *TVWL* while satisfying timing constraints. However, for the m-column folded placement, considering only the relative ordering of nodes within each column C_i' entails $\prod_{i}^{m} |C_i'|!$ possible orderings, incurring substantial computational overhead, particularly for large circuits. To address this challenge, we observe that the wirelength determined by the vertical position of node v can be expressed as

$$W_v = \begin{cases} l_v^{vc} + l_v^{mc} + \sum_{u \in CO(v)} (l_{v,u}^{vc} + l_{v,u}^{mc}) + \sum_{u \in E_d(v)} (l_{u,v}^{vd} + l_{u,v}^{md}), & v \in G, \\ l_v^{vc} + l_v^{mc} + \sum_{u \in CO(v)} (l_{v,u}^{vc} + l_{v,u}^{mc}), & v \in D^c, \\ \sum_{u \in E_d(v)} (l_{u,v}^{vd} + l_{u,v}^{md}), & v \in D^d. \end{cases} \tag{9}$$

where $CO(v)$ denotes the set of nodes that serve as clock outputs of v. E_d denotes the set of nodes that serve as data inputs or outputs of v. D^c and D^d denote the sets of clock dummies and data dummies, respectively. According to Equation (9), the impact of node v on the

Algorithm 1: JSPlace algorithm.

Input: A path-balanced netlist $N(V, E)$, target width-to-height ratio r_t, iteration R.
Output: A corresponding legal placement.
1 Assign gates to n gate columns based on their stages.
2 Randomly initialize the positions of nodes within each column.
3 Fold the n-stage pipelined placement at column p and q.
4 Form m-column folded placement by merging columns based on MILP.
for $r \in [1, R]$ **do**
5 Generate the clock distribution.
6 **for** *column* $i \in [0, m]$ **do**
7 Vertical placement refinement for column i.
8 Repositioning the nodes within each column.

TVWL depends only on its own vertical position and the vertical positions of its fanin and fanout nodes, with no dependence on other nodes in the same column.

Therefore, we propose a dynamic programming (DP)-based algorithm that optimizes the vertical positions of nodes column by column. The algorithm determines the vertical positions of nodes in column i under the assumption that node positions in all other columns remain fixed. To efficiently compute optimal placements within column i, it preserves the relative ordering of nodes in column i and introduces a hyperparameter α to bound their allowable vertical positions. The value of α controls the trade-off between solution quality and computational efficiency: a larger α provides more flexibility but increases the DP state space. In our experiments, $\alpha = 50$ provides a good balance. Specifically, the nodes in column i are sorted in ascending order of their current vertical positions and denoted as $C_i' = \left\{v_1, v_2, \ldots, v_k, \ldots, v_{|C_i'|}\right\}$. For each node $v_k \in C_i'$, its vertical position $y(v_k)$ restricted to $[y(v_k) - \alpha, y(v_k) + \alpha]$. Considering the non-overlap constraints, the set of feasible candidate positions for node v_k in column i is defined as:

$$P_k = \{y_{k,c} \mid y_{k,c} \in [y(v_k) - \alpha, y(v_k) + \alpha] \cap [\sum_{j=1}^{k-1} h(v_j), H_f - \sum_{j=k}^{|C_i'|} h(v_j)]\}, \quad (10)$$

where $c = 1, 2, \ldots, |P_k|$. H_f denotes the maximum column height in m-column folded placement. Based on the set of candidate positions P_k for each node v_k in column i, the DP-based algorithm constructs a weighted directed graph $\mathcal{N}_i$ for column i.

As shown in Fig. 5, the graph $\mathcal{N}_i$ includes vertex sets $\{s, t\} \cup P_1 \cup \cdots \cup P_{|C_i'|}$, where source s connects to all vertices in P_1, and all vertices in $P_{|C_i'|}$ connect to sink t. The weights of edges connecting to t are set to zero. Edges exist from any vertex $y_{k,c} \in P_k$ to $y_{k+1,j} \in P_{k+1}$ only if $y_{k,c} + h(v_k) \leq y_{k+1,j}$, ensuring no vertical overlap between nodes. Each outgoing edge from $y_{k,c}$ is assigned a weight $w(y_{k,c})$ representing the cost associated with placing node v_k at position $y_{k,c}$. The edge weight $w(y_{k,c})$ is defined as $w(y_{k,c}) = W_{v_k}(y_{k,c})$, where $W_{v_k}(y_{k,c})$ corresponds to W_{v_k} when the position of v_k is $y_{k,c}$. The optimal vertical positions for all nodes $v \in C_i'$ correspond to the path from s to t with the minimum sum of edge weights. To find this path, we propose the following DP algorithm:

$$dp[k][y_{k,c}] = \begin{cases} \min\limits_{y < y_{k,c} - h(v_{k-1})} \left(dp[k-1][y] + w(y_{k,c}), dp[k][y_{k,c}]\right), & k > 1 \\ w(y_{k,c}), & k = 1 \end{cases}. \quad (11)$$

Table 1. Placement results for priority.

Method	r_t	H(mm)	W(mm)	A(mm^2)	Ratio	TVWL(mm)	Time(s)
JPlace[6]	/	9.24	103.95	960.50	11.25	106192.01	265.77
Ours	2	19.46	39.17	762.25	2.01	69789.15	522.01
	1.33	21.9	37.61	823.66	1.72	61896.57	233.07
	1	22.24	41.74	928.30	1.88	82062.86	534.45
	0.75	21.09	39.20	826.73	1.86	73518.04	653.00
	0.5	22.24	43.53	968.11	1.96	95324.70	529.37

For $k = 1$, $dp[1][y_{k,c}]$ is initialized to $w(y_{k,c})$. For $k > 1$, $dp[k][y_{k,c}]$ represents the minimum cumulative weight for placing nodes v_1 to v_k with v_k at $y_{k,c}$. The minimum value in the array $dp[|C_i'|]$ identifies the path with the least cumulative weight, and the vertices on this path correspond to the optimal vertical positions for each node. By backtracking through the dp array, the algorithm determines the optimal vertical positions for nodes in column i.

4.5 Connection-Driven Repositioning

Although vertical placement refinement improves computational efficiency by fixing the relative order of nodes within each column, it restricts the solution space. To achieve a higher-quality placement, we propose a connection-driven repositioning method. As indicated by Equation (9), the contribution of a node to *TVWL* is determined by the vertical positions of its data fanin and fanout nodes. Specifically, the larger the vertical minimum length and the vertical matching length between node v and a connected node u, the greater the influence of $y(u)$ on the contribution of v to *TVWL*. Thus, we reposition node v based on the vertical minimum lengths and vertical matching lengths of its data connections. The new vertical position of node v after repositioning can be expressed as:

$$y(v) = \frac{\sum\limits_{u \in E_d(v)} (l_{u,v}^{vd} + l_{u,v}^{md}) \cdot y(u)}{\sum\limits_{u \in E_d(v)} (l_{u,v}^{vd} + l_{u,v}^{md})} \quad (12)$$

After repositioning, the relative ordering of nodes within each column is also changed, enabling a further iteration of vertical placement refinement that transcends the fixed-order constraint and explores higher-quality placements.

4.6 Time Complexity Analysis

Algorithm 1 outlines the complete flow of our shape-controllable and length-matching RSFQ placement algorithm, where R denotes the total number of iterations. In the initial placement phase, logic gates are assigned to columns according to their logic stages (line 1). They are given random vertical positions (line 2), forming an n-stage pipelined placement in $O(V)$ time. Next, an MILP-based column merging step folds the pipelined placement into the m-column folded placement (line 4), incurring a runtime of T_{MILP}. The clock distribution (line 5) is then generated with a complexity of $O(|V|\ log|V|)$. Vertical placement refinement (line 7) employs dynamic programming within each column, requiring $O(\alpha^2|V|)$ time. Finally, nodes are repositioned based on the vertical wirelength of their data connections (line 8), updating their relative ordering within each column. Thus, the per-iteration complexity is $O(\alpha^2|V| + |V|log|V|)$. Since $\alpha = 50$ is a small constant in our experimental setup, the overall runtime of JSPlace is $T_{MILP} + O(R \cdot |V|log|V|)$.

Table 2. Comparison between our placement algorithm and the baselines with $r_t = 1.33$.

Circuit	Folded JPlace[6]						Folded SA[17]						Ours					
	H(mm)	W(mm)	A(mm^2)	Ratio	TVWL(mm)	Time(s)	H(mm)	W(mm)	A(mm^2)	Ratio	TVWL(mm)	Time(s)	H(mm)	W(mm)	A(mm^2)	Ratio	TVWL(mm)	Time(s)
c432	5.39	10.44	56.27	1.94	4680.89	10.202	6.28	12.66	79.50	2.02	5495.96	27.086	6.29	7.95	50.01	1.26	3887.06	14.880
c880	9.38	12.5	117.25	1.33	6648.39	12.132	6.95	17.17	119.33	2.47	8900.75	40.679	8.98	9.12	81.90	1.02	4735.76	26.996
c1355	10.89	6.92	75.36	0.64	4392.48	11.696	7.58	9.19	69.66	1.21	4575.55	15.220	8.61	5.94	51.14	0.69	3302.69	16.880
c1908	7.65	10.62	81.24	1.39	5104.93	8.903	6.87	15.72	108.00	2.29	7179.07	32.106	9.27	8.03	74.44	0.87	4286.43	23.528
c3540	21.39	19.89	425.45	0.93	23841.76	28.010	21.39	27.46	587.37	1.28	32577.25	231.413	23.01	14.9	342.85	0.65	16681.84	36.102
c5315	41.98	31.33	1315.23	0.75	83016.93	88.636	34.46	64.09	2208.54	1.86	117734.21	674.344	44.82	27.73	1242.86	0.62	60953.09	703.617
c6288	19.01	22.8	433.43	1.20	50681.41	18.242	15.68	32.02	502.07	2.04	50288.98	131.038	17.55	21.58	378.73	1.23	41427.47	53.780
c7552	37.44	48.16	1803.11	1.29	65480.96	128.521	38.5	91.33	3516.21	2.37	147977.04	730.788	37.4	29.76	1113.02	0.80	62257.93	186.279
int2float	5.88	5.54	32.58	0.94	2363.59	3.260	5.88	6.44	37.87	1.10	2484.9	15.600	6.65	4.66	30.99	0.70	1795.21	5.492
sin	48.97	62.27	3049.36	1.27	275088.99	393.855	55.18	140.87	7773.21	2.55	796191.08	4135.644	51.88	65.55	3400.73	1.26	302681.92	532.632
priority	22.97	48.19	1106.92	2.10	119583.64	266.728	20.91	135.44	2832.05	6.48	161545.98	713.909	21.9	37.61	823.66	1.72	61896.57	233.069
adder	49.7	70.76	3516.77	1.42	246476.61	1005.598	42.07	281.48	11841.86	6.69	667605.38	3213.573	45.63	56.85	2594.07	1.25	180908.81	696.271
max	84.05	167.04	14039.71	1.99	1036629.30	1892.337	98.81	642.55	63490.37	6.50	3512722.13	16061.986	86.18	128.36	11062.06	1.49	492620.67	1471.111
multiplier	131.92	327.36	43185.33	2.48	6162456.35	1386.161	80.82	815.98	65947.50	10.10	8690331.82	14094.452	114.04	412.16	47002.73	3.61	6693944.69	5029.944
Ave. ratio	1.00	1.22	1.22	/	1.34	0.69	0.91	2.48	2.32	/	2.33	3.67	1.00	1.00	1.00	/	1.00	1.00

5 Experimental Results

5.1 Experimental Setup

JSPlace is implemented in Python and evaluated on an Intel(R) Xeon(R) Gold 5218R CPU with 128 GB of memory. The gate-level netlists are first synthesized using the ABC tool [3] and then DFFs are inserted [24] to achieve path balancing. Gate dimensions and timing parameters are derived from the ColdFlux RSFQ standard cell library [20]. All gates are modeled as composite gates with a uniform layout width of 90 µm in placement. The MILP problems are solved column-by-column using Gurobi 12.0 and a time limit of 600 seconds per column.

Both JSPlace and the baseline methods employ the same parameters $R = 100$ and $\alpha = 50$, which are empirically determined to provide a good balance between solution quality and computational efficiency. All methods terminate automatically after five consecutive iterations without improvement. The layout width is estimated as described in Section 4.1.

5.2 Benchmarks and Baselines

The experiments employ combinational benchmark circuits from ISCAS85 [12] and EPFL [1]. All comparisons are conducted under a target aspect ratio $r_t = 1.33$, which approximates common reticle aspect ratios in semiconductor manufacturing, unless otherwise specified. We compare JSPlace against two baselines. The primary baseline is JPlace [6], a state-of-the-art multi-stage pipelined placement algorithm for RSFQ circuits. For further evaluation, we also apply the same folding strategy described in Section 4.2 to partition the placement generated by JPlace into three segments, where the i-th column of each segment is directly merged with the i-th column of the other segments. Since JPlace was not designed for folding, this naive column-by-column merging serves as a baseline to isolate the contribution of our MILP-based optimization.

Additionally, we also compare our algorithm with an SA-based multi-stage pipelined placement method [17]. This SA method first pairs logic gates within the same column, and during each perturbation, simultaneously adjusts multiple gate pairs to enhance search efficiency and minimize *TVWL*. The SA parameters follow the original work [17]: 10 iterations per temperature step, a cooling rate of 0.95, a termination temperature threshold of 0.01, and an initial temperature of 100. The resulting multi-stage pipelined placement from SA is also processed using the same three-segment folding strategy to serve as a baseline for comparison.

5.3 Performance Results

TABLE 2 presents a comparison of placement quality between our placement algorithm and the folded baselines, where 'H' is the layout height, 'W' is the estimated layout width, 'A' is the layout area, and "Ratio" denotes the width-to-height ratio of the layout.

Compared to the folded JPlace placement, our algorithm exhibits a 26.37% higher average relative error with respect to the target width-to-height ratio but achieves 15.37% and 21.36% reductions in layout area and total vertical wirelength, respectively.

Compared to the folded SA-based placement, our algorithm reduces the average relative error to the target width-to-height ratio by 77.12%, while reducing layout area and total wirelength by 45.65% and 46.43%, respectively, along with a 54.34% decrease in runtime.

Results of Different Ratios. TABLE 1 compares placement quality between our algorithm and JPlace on the priority circuit (a medium-scale circuit with 17778 gates and 215 logic stages) under varying target width-to-height ratios. Our algorithm achieves significantly better control over the layout width-to-height ratio compared to JPlace. While JPlace produces a highly elongated layout with a ratio of 11.25, our algorithm achieves a ratio of 2.01 when $r_t = 2$, demonstrating excellent shape control. When the target ratio is further reduced, however, the realized ratio decreases only marginally. This limited reduction is likely due to the length-matching constraints of RSFQ circuits, which restrict the minimization of total vertical wirelength and thus limit the achievable reduction in layout width. For all tested r_t, our algorithm reduces the *TVWL* by an average of 27.94% and the layout area by 10.27% on the priority circuit.

6 Conclusion

This paper proposed the first shape-controllable and length-matching placement method for RSFQ circuits. We fold the multi-stage pipelined placement into three segments and merge columns via mixed-integer linear programming. Then, dynamic programming determines optimal vertical node positions to minimize wirelength. Connection-driven repositioning further refines placement by adjusting node positions based on data connection lengths. Experiments on ISCAS85 and EPFL benchmarks show our method effectively controls the layout aspect ratio while improving wirelength and layout area.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62302477); in part by the State Key Lab of Processors, Institute of Computing Technology, CAS (Grant No. CLQ202402); in part by the Shandong Province Natural Science Foundation, China (Grant No. ZR2024MF073); and in part by the Shandong Province Major Scientific and Technological Innovation Projects, China (Grant No. 2025S0101-01954).

References

[1] Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL Combinational Benchmark Suite. In *IEEE/ACM International Workshop on Logic Synthesis*.

[2] Yuki Ando, Ryo Sato, Masamitsu Tanaka, Kazuyoshi Takagi, Naofumi Takagi, and Akira Fujimaki. 2016. Design and Demonstration of an 8-bit Bit-Serial RSFQ Microprocessor: CORE e4. *IEEE Transactions on Applied Superconductivity* 26, 5 (2016), 1–5. https://doi.org/10.1109/TASC.2016.2565609

[3] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *International Conference on Computer-Aided Verification (CAV)*. Springer-Verlag, 24–40. https://doi.org/10.1007/978-3-642-14295-6_5

[4] Paul Bunyk, Konstantin Likharev, and Dmitry Zinoviev. 2001. RSFQ technology: Physics and devices. *International Journal of High Speed Electronics and Systems* 11 (03 2001). https://doi.org/10.1142/S012915640100085X

[5] P. Bunyk and P. Litskevitch. 1999. Case study in RSFQ design: fast pipelined parallel adder. *IEEE Transactions on Applied Superconductivity* 9, 2 (1999), 3714–3720. https://doi.org/10.1109/77.783835

[6] Siyan Chen, Rongliang Fu, Junying Huang, Zhimin Zhang, Xiaochun Ye, Tsung-Yi Ho, and Dongrui Fan. 2024. JPlace: A Clock-Aware Length-Matching Placement for Rapid Single-Flux-Quantum Circuits. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 1–6. https://doi.org/10.23919/DATE58400.2024.10546887

[7] Xinda Chen, Rongliang Fu, Junying Huang, Huawei Cao, Zhimin Zhang, Xiaochun Ye, Tsung-Yi Ho, and Dongrui Fan. 2023. JRouter: A Multi-Terminal Hierarchical Length-Matching Router under Planar Manhattan Routing Model for RSFQ Circuits. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. 515–520. https://doi.org/10.1145/3583781.3590267

[8] Junhyuk Choi, Ilkwon Byun, Juwon Hong, Dongmoon Min, Junpyo Kim, Jungmin Cho, Hyeonseong Jeong, Masamitsu Tanaka, Koji Inoue, and Jangwoo Kim. 2024. SuperCore: An Ultra-Fast Superconducting Processor for Cryogenic Applications. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1532–1547. https://doi.org/10.1109/MICRO61859.2024.00112

[9] Rongliang Fu, Minglei Zhou, Siyan Chen, Xinda Chen, Junying Huang, Xiaochun Ye, Zhimin Zhang, and Tsung-Yi Ho. 2025. JPnR: A Length-Matching Placement and Routing Framework for Single-Flux-Quantum Circuits. *IEEE Trans. Comput.* (2025). https://doi.org/10.1109/TC.2025.3625822

[10] Rongliang Fu, Minglei Zhou, Huilong Jiang, Junying Huang, Xiaochun Ye, and Tsung-Yi Ho. 2025. J2Place: A Multiphase Clocking-Oriented Length-Matching Placement for Rapid Single-Flux-Quantum Circuits. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD66269.2025.11240637

[11] Kris Gaj, Eby G. Friedman, and Marc J. Feldman. 1997. Timing of multi-gigahertz rapid single flux quantum digital circuits. *Journal of VLSI signal processing systems for signal, image and video technology* 16, 2 (1997), 247–276. https://doi.org/10.1023/A:1007903527533

[12] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. 1999. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design & Test* 16, 3 (1999), 72–80. https://doi.org/10.1109/54.785838

[13] Akihiro Hashimoto and James Stevens. 1971. Wire routing by optimizing channel assignment within large apertures. In *ACM/IEEE Design Automation Conference (DAC)*. 155–169.

[14] D Scott Holmes, Andrew L Ripple, and Marc A Manheimer. 2013. Energy-efficient superconducting computing—Power budgets and requirements. *IEEE Transactions on Applied Superconductivity* 23, 3 (2013), 1701610–1701610.

[15] Koki Ishida, Ilkwon Byun, Ikki Nagaoka, Kosuke Fukumitsu, Masamitsu Tanaka, Satoshi Kawakami, Teruo Tanimoto, Takatsugu Ono, Jangwoo Kim, and Koji Inoue. 2020. SuperNPU: An Extremely Fast Neural Processing Unit Using Superconducting Logic Devices. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 58–72. https://doi.org/10.1109/MICRO50266.2020.00018

[16] Mohammad Reza Jokar, Richard Rines, Ghasem Pasandi, Haolin Cong, Adam Holmes, Yunong Shi, Massoud Pedram, and Frederic T Chong. 2022. DigiQ: A scalable digital controller for quantum computers using SFQ logic. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 400–414.

[17] Nobutaka Kito, Kazuyoshi Takagi, and Naofumi Takagi. 2018. A Fast Wire-Routing Method and an Automatic Layout Tool for RSFQ Digital Circuits Considering Wire-Length Matching. *IEEE Transactions on Applied Superconductivity* 28, 4 (2018), 1–5. https://doi.org/10.1109/TASC.2018.2793203

[18] K.K. Likharev and V.K. Semenov. 1991. RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity* 1, 1 (1991), 3–28. https://doi.org/10.1109/77.80745

[19] Ryo Sato, Yuki Hatanaka, Yuki Ando, Masamitsu Tanaka, Akira Fujimaki, Kazuyoshi Takagi, and Naofumi Takagi. 2017. High-Speed Operation of Random-Access-Memory-Embedded Microprocessor With Minimal Instruction Set Architecture Based on Rapid Single-Flux-Quantum Logic. *IEEE Transactions on Applied Superconductivity* 27, 4 (2017), 1–5. https://doi.org/10.1109/TASC.2016.2642049

[20] L. Schindler and T. Hall. [n. d.]. RSFQ cell library. https://github.com/sunmagnetics/RSFQlib. Version: 3.0, Release date: 21 March 2023.

[21] T Van Duzer, CW Turner, DG McDonald, and Alan F Clark. 1982. Principles of superconductive devices and circuits. *Physics Today* 35, 2 (1982), 80.

[22] Jin-Tai Yan. 2022. Fixed-Order Placement of Pipelined Architecture in Rapid Single-Flux-Quantum Circuits. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 30, 10 (2022), 1519–1532. https://doi.org/10.1109/TVLSI.2022.3182290

[23] Jin-Tai Yan. 2022. 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)* 41, 4 (2022), 1090–1102. https://doi.org/10.1109/TCAD.2021.3070834

[24] Minglei Zhou, Rongliang Fu, Ran Zhang, Xiaochun Ye, Tsung-Yi Ho, and Junying Huang. 2025. An Optimal DFF-Oriented Technology Legalization Algorithm for Rapid Single-Flux-Quantum Circuits. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*. 6 pages. https://doi.org/10.1145/3716368.3735163