An Integrated Cell Placement and Interconnect Synthesis Tool for Large SFQ Logic Circuits

Soheil Nazar Shahsavani, Ting-Ru Lin, Alireza Shafaei, Student Member, IEEE,
Coenrad J. Fourie, Member, IEEE, and Massoud Pedram, Fellow, IEEE

Abstract—This paper presents a row-based design methodology covering cell placement, clock tree synthesis, and routing steps for large SFQ circuits. The proposed placement tool initiates by running a conventional placer, which places fixed-height cells on logic rows. Next, in each row, cells with the same logic level are grouped together, and cell groups are then sorted based on their logic level. Furthermore, instead of a costly H-tree network, a clock row is dedicated above each logic row, which uses a combination of splitters/JTLs and, if cells are not sorted based on their logic level order, PTLs to distribute the clock pulse to all cells in the corresponding row. To evaluate the effectiveness of the proposed approach, place-and-route results for a 32-bit Kogge-Stone adder using a state-of-the-art conventional placement and the proposed SFQ-specific placement are reported and compared. The half-perimeter wirelength of the SFQ-specific placement tool is 32% smaller than that of the conventional counterpart. As a result, while signal/clock routing with only two metal layers fails to complete within the specified timeout for conventional placement, a routable solution can be found for the SFQ-specific placement. Moreover, the minimum chip area is achieved with four metal layers using the SFQ-specific placement, whereas the conventional placer needs seven metal layers to achieve the same chip area.

Index Terms—Row-based design automation, single-flux quantum (SFQ), placement, routing, clock tree synthesis.

I. INTRODUCTION

Demand for high performance and energy efficient computing has been driving the development of the semiconductor technology for decades [1]. Until recently, conventional computing technology based on CMOS devices and standard metal interconnects has been able to increase computing performance and energy efficiency fast enough to keep up with this increasing demand. Unfortunately, with increasing challenges to physical scaling of CMOS devices and the conclusive end of Moore’s law in sight, there is a significant need to find new technologies and design methodologies that would allow continuation of performance and energy efficiency scaling to well beyond the end-of-scaling CMOS nodes (e.g., an all-around 5nm gate-length transistor).

Superconducting computing based on the Josephson effect has the potential to be one such solution. This is because Josephson junctions (JJs) switch quickly (~1ps) and dissipate little switching energy (~$10^{-19} J$) [2] at low temperatures. In particular, rapid single flux quantum (RSFQ) technology was introduced in the 1980s, which uses quantized voltage pulses in digital data generation, reproduction, amplification, memorization, and processing [3]. Further, it has been demonstrated that RSFQ circuits are functional at operating frequency of up to 770GHz [4]. Recent developments introduce various approaches, such as new single flux quantum (SFQ) logic families, including dual-rail RSFQ [5], self-clocked complementary logic (SCCL) [6], reciprocal quantum logic (RQL) [7], re-design of the current biasing network for RSFQ [8][9][10], and application of low supply voltage for RSFQ circuits [11]. These techniques significantly reduce the power and energy consumption of SFQ logic circuit realizations [12].

Although extraordinary characteristics such as high frequency and low energy dissipation have been observed, many problems, including architectures, design automation methodologies and tools, and device fabrication require solutions in order for the SFQ logic to become a realistic option for realizing large-scale, high-performance, and energy-efficient computing systems of the future [13]. The increase in integration density of modern superconducting circuit processes allows the design of increasingly complex circuits. The layout of such large circuits requires automated placement and routing tools.

A comprehensive study on the status and capabilities of software design tools for superconductive electronics (SCE) was published in 1999 [14], and some of the main shortcomings identified then were lack of uniform tool used, even within institutions, and lack of standardized data formats. However, design challenges of SFQ logic and tool development did not receive much attention in the decade thereafter, and a follow-up review published in 2013 [15] concluded that the status of SCE software tools was little better than in 1999.

In this paper, a new design methodology for large SFQ circuits is proposed. An earlier SFQ design methodology presented in [16] places cells on a rectangular grid using the min-cut placement algorithm. In this approach, each cell is surrounded by a reserved space for clock channels within which an H-tree synchronous clock is routed in order to construct a zero-skew clock scheme. Data signals are routed over the gates using passive transmission lines (PTLs). Such a design methodology suffers from the following two issues: (i) the large size of the H-tree intensifies the jitter problem, and (ii) a considerable portion of the chip area should be devoted to the H-tree network. As a result, we opted not to utilize the H-tree network for this paper.

To alleviate the high cost of the clock network in large SFQ circuits and enable automated placement and routing of such circuits, a row-based design methodology is presented in this paper. More specifically, we present a new design methodology based on fixed-height, variable-width logic cell layouts and
propose a row-based placement strategy that improves packing efficiency and automation capabilities of layout tools. With our method, clock pulses are routed using splitters/JTLs and if needed PTLs in dedicated channels on top of cell rows to allow localized clock-follow-data or counter-flow-clocking implementations [17]. Our routing-aware interconnect synthesis technique uses direct pin connection for abutted cells and PTL wiring for long interconnects, while different physical instances of the same logic cell are instantiated according to interconnect requirements. Finally, we place and route a 32-bit Kogge-Stone [18] adder using the methods presented here, and discuss the effectiveness thereof.

The rest of the paper is organized as follows. Our SFQ design methodology including placement, clock tree synthesis, and signal routing approaches are discussed in Section II. Simulation results of a 32-bit Kogge-Stone adder are presented in Section III. Finally, the paper is concluded in Section IV.

II. PROPOSED SFQ DESIGN METHODOLOGY

Our proposed design methodology is comprised of three different phases: (i) cell placement, (ii) clock tree synthesis, and (iii) routing. Details of each phase are explained in this section. An overview of the place-and-route algorithm is also depicted in Fig. 1.

A. Cell Placement

The output of logic synthesis, which is a gate-level netlist, is input to the placement tool. In addition, the placement tool needs dimensions and pin locations of each cell (gate). The placement tool then assigns cells to positions on the chip such that no two cells overlap with each other and a cost function (e.g., chip area, total wire length, critical path delay) is minimized. An important consideration in the placement problem is that the placement solution should be routable. Accordingly, to avoid multiple costly iterations between placement and routing steps, routing-aware placement algorithms are of significant interest.

To simplify the design automation of the placement step, row-based placement techniques are widely used in the VLSI design community and semiconductor industry. These techniques place (fixed-height, but variable width) cells in rows on the chip. Furthermore, power interconnects run horizontally through the top and bottom of cells. Therefore, when cells are placed adjacent to each other, power interconnects form two continuous parallel tracks in each row. Clock signal is distributed through an H-tree network to all cells. Input and output pins of cells are available at the top and/or bottom sides of the cell, and are connected by interconnects routed in the routing channels between adjacent rows. Connections from one row to another are done either through the surrounding "ring" or by using feed-through cells. A cell that complies with these features will be referred to as a standard cell.

To efficiently place netlists with millions of cells, the following three-step placement algorithm is typically used. (i) **Global placement:** the non-overlapping cell constraint is ignored in this step and approximate locations of cells are obtained by placing cells in global bins. The main focus of the global placer is to optimize the cost function by iterating between the solution of some mathematical program (e.g., a constrained quadratic or linear programming problem) and a cell spreading (or bi-partitioning) step. (ii) **Legalization:** the output of the global placement must be legalized to remove any cell overlaps. (iii) **Detailed placement:** legalization is further refined by using local adjustments such as cell movement or swapping to reduce wire length. The placement legality must be preserved during the detailed placement. Legalization and detailed placement are sometimes grouped as one step.

Modern global placement algorithms are based on analytical methods in which a mathematical analysis is adopted to optimize the cost function. For instance, SimPL [19] solves large and sparse systems of linear equations (formulated using force-directed placement) by Conjugate Gradient method [20]. More specifically, the force-directed method reduces the placement problem to that of solving a set of simultaneous linear equations to determine equilibrium (i.e., zero-force) locations for cells based on Hooke’s law analogy. Next, cell spreading is performed by inserting additional forces that pull cells away from dense regions toward carefully placed anchors (pseudo-fixed pins). On the other hand, min-cut placement and simulated annealing-based placement are often used for performing the legalization and detailed placement steps.

Assuming that a zero-skew clock scheme (i.e., the clock tick simultaneously arrives at all registers) is available, which can be done by using the H-tree clock, one may directly use the output of the CMOS placement tool for placing SFQ circuits. This approach has been adopted in [16], which uses the min-cut placement algorithm for deriving cell positions. Although the results can be improved by using state-of-the-art placement tools, such as SimPL [19], the main issue is the implementation of the H-tree clock in SFQ circuits. Since almost all SFQ cells require a clock signal, a huge H-tree clock is needed especially in large circuits. The large size of the H-tree along with the high-speed requirements of SFQ circuits intensifies jitter problem to unmanageable levels. Additionally, a considerable portion of the chip area would be devoted to the H-tree routing, which in turn decreases the chip density (i.e., portion of the chip used for logic cells). To alleviate the high cost of the clock network in SFQ circuits, a new methodology with various modifications to the corresponding CMOS flow is proposed in this paper.

As stated above, our placer utilizes a row-based placement methodology. That is, the chip is partitioned into several rows for placing standard cells. Each SFQ standard cell in our library is composed of two parts (cf. Fig. 2): (i) **A logic**
design part (logic part for short), which implements a Boolean function such as AND, OR, INV, etc. (ii) A built-in clock distribution part (clock part for short), which is placed above the logic part. The clock part contains a splitter which provides the clock signal to the corresponding logic part, and also passes the clock pulse to the successor cell in the logic level order. If the logic part does not need a clock signal (e.g., splitter or merger cells), the clock part implements a JTL to pass the clock pulse to the next cell. If cells in a row are sorted based on their logic level, then the built-in clock part can distribute the clock signal to all cells in that row; otherwise, PTL tracks are needed to complete the clock routing.

In our SFQ standard cell library, logic parts are of the same height. Similarly, all clock parts have the same height, which is smaller than that of the logic part. Accordingly, our SFQ standard cells will have the same height. Additionally, for logic and clock parts, different templates based on the type of input and output pins exist (cf. Fig. 3 and Fig. 5). Proper templates are selected after the placement solution is found.

In the SFQ placement problem, we assume that the input netlist is path-balanced\(^1\) and all fan-outs are implemented with splitters. The placement tool then initiates by running a global placement followed by detailed placement and legalization, which together generate a legal solution with the minimum total wire length. Our proposed clock tree network can work with this initial placement solution, but results in a large number of PTL tracks and vias needed for clock routing (more details will follow in the next sub-section). Accordingly, in our proposed placement approach, this initial placement solution is refined in order to minimize the clock tree cost. For this purpose, the logic level of each cell in the netlist is needed, which can be generated by the logic synthesis tool. The logic level of a cell captures the stage of the clock signal that is received by that cell. More precisely, if the longest path from any primary input of the circuit to a SFQ logic cell in a circuit has a logical depth (in terms of node count) of \(k\), then the clock stage associated with that cell is \(k\), and the cell will be given level \(k\).

After the initial placement solution is generated by a conventional VLSI placer, all cells with the same logic level in each row are grouped together. Cell groups in each row are subsequently sorted in increasing order of their levels. As a result, no PTL track for clock routing within each row is needed. This is because cells in each row are now placed based on the same order that they should receive the clock. This solution results in a simple clock tree which in turn decreases the total wirelength of the circuit.

B. Clock tree synthesis

Clock tree synthesis is a process that makes sure that the clock signal is properly distributed to all sequential elements in a circuit. Clock routing is performed before signal routing to avoid competition for resources occupied by signal nets. In SFQ circuits, a DFF is included in each logic cell (except for splitters and mergers). As a result, the clock tree network is significantly larger in SFQ circuits compared with that of CMOS circuits. Accordingly, the clock tree synthesis must be given a very high priority in SFQ designs.

Our adopted row-based design methodology not only simplifies the placement algorithm but also results in a well-structured clock tree network. Standard cells are placed in fixed-height rows. We also assume that all cells with the same logic level in each row are placed consecutively on the row to form a cell group. More precisely, cell group \(i\) refers to group of cells all of which are of logic level \(i\). If cell groups in a row are sorted in the ascending order of their levels, as it happens in our proposed SFQ-specific placer, then the built-in clock part of standard cells is sufficient to distribute the clock signal to all logic cells in the corresponding row.

When cell groups are not sorted in a row (e.g., in a conventional placement solution), we use PTLs to transfer the clock signal from the output of cell group \(i\) to the input of cell group \(i + 1\) if cell group \(i + 1\) is not placed immediately after cell group \(i\). For this purpose, one or more clock routing channels are reserved on top of each row. Furthermore, we assume that the clock signal is provided from the left side of the chip, and a combination of PTLs and splitters (splitters with PTL receiver and transmitter) distribute the clock signal to cell group 1 in each row. If cell group 1 is missing from a row, then the entering clock (after a calculated predetermined delay) is routed directly to the cell group with the smallest level in that row (e.g., to cell group 3 if cell groups 1 and 2 do not exist in the row).

Based on the aforesaid clock tree network, we introduce six templates for the clock part of standard cells as shown in Fig. 3. Four templates on top of Fig. 3 are pass-through cells that are used to distribute the clock signal to the next cell in the ascending order of the logic level. These templates contain either a splitter (if the corresponding logic part needs a clock signal) or a JTL (if the corresponding logic part is a splitter or a merger), and their difference is in the type (JTL or PTL) of the input/output pins. The other two templates, referred to as end-of-line templates, terminate the clock signal and differ in

---

1\(^1\)In a path-balanced netlist, all paths from any primary input to any primary output have the same logical depth [21]. Any netlist can be path-balanced by inserting D flip-flops (DFFs) to paths with a smaller depth.
In this paper, we do not use JTLs for signal routing, since (i) JTLs require JJs and hence occupy the active layer, which in turn complicates the placement tool, and (ii) JTLs are slower than PTLs especially for long-distance communications. Therefore, signal routing is done either by direct connection if source and destination pins are abutted or by PTLs otherwise. Accordingly, we will have different templates for the logic part of standard cells. As an example, Fig. 5 shows four possible templates for a 2-input gate.

Our routing tool will be built on top of the open-source Qrouter tool [22], which is developed based on the standard Lee maze router algorithm. Given LEF and DEF files, the routing area is partitioned into a two-dimensional grid of routing tracks. A wave propagation method connects the identified source and target nodes while avoiding obstructions during propagation and calculating the corresponding cost along multiple paths. The L-shaped or doglegged paths with the lowest cost are then traced back to the source node from the target and committed to memory. The grid positions occupied by routed paths become obstructions for future routes on other nets (or become additional source or target nodes for further routing of the same net).

There are two steps to complete the routing of a circuit. The first step seeks to find the routing solution for each net one at a time according to some net ordering, while keeping track of any failed nets in a list. In the second step, each net that had previously failed is routed again, this time allowing the route to create shorts with other routed nets. Subsequently, all such shorted nets are removed and added to the list of failed routes. This process continues until all nets have been routed. To avoid an infinite loop in this iterative rip-up-and-re-route process, the same sequence of rip-up and re-route for two nets is not allowed to happen more than once. In addition, four corners of our standard cells are reserved for bias pillars, and clock and signal PTLs cannot be routed through these reserved zones. We handle such cases by introducing “no-route zones” in the Qrouter tool.

D. Place-and-Route Exploration

The placement solution directly impacts the clock tree construction. Our clock tree generation method can handle any arbitrary placement of standard cells. For instance, we can directly use the output of a conventional placement tool (cf. Fig. 6 (a)), which tends to result in the minimum total wire length solution. However, this solution ignores the logic levels of cells and hence require many more PTL tracks and vias in the clock routing channel. These PTL tracks create blockage for the signal routing. As a result, more metal layers or larger chip area may be needed to complete the routing step.

As indicated in Fig. 6 (b), when cells are sorted, no PTL track outside the clock row is needed. Thus, the absence of PTL tracks in the clock row allows PTL resources to be used for the signal routing, which in turn simplifies the signal routing step. Therefore, using our SFQ-specific placement solution, it is easier to come up with a routable placement solution of the highest quality for SFQ circuits. Furthermore, when a SFQ pulse visits multiple vias (i.e., connections between different consecutive PTL layers), signal integrity may degrade to a level that circuit stops functioning. Using our proposed placement tool, the clock tree is constructed...
A standard cell with logic level $i$. Cell group $i$.

Fig. 6. Impact of (a) conventional and (b) SFQ-specific placements on the clock tree network. Conventional placement shown in (a) requires multiple PTL tracks to complete the clock routing, whereas our SFQ-specific placement routes the clock signal without PTLs.

mainly by splitters and JTLs, and almost all PTLs can be used for signal routing. Hence, compared with a conventional placement tool, SFQ pulses experience a smaller number of vias with our SFQ-specific placement tool.

III. EXPERIMENTAL RESULTS

To implement the global placement, we adopt a methodology similar to SimPL [19]. The main purpose of the global placement is to reduce the total interconnect length which could be approximated by half-perimeter wirelength (HPWL) denoted by:

$$HPWL = HPWL_x + HPWL_y$$

(1)

where

$$HPWL_x = \sum_{e \in E} |\text{max}(x_i)|_{i \in e} - |\text{min}(x_i)|_{i \in e}$$

(2)

$HPWL_y$ could be calculated in a similar manner. A set of cells with connections to each other represents a graph $G(V, E)$ with set of edges $E$, where $e$ denotes a hyperedge (a multi-pin net), and set of vertices $V$ denoting cells, with edge weights $w_{ij}$ representing the cost of each net. Consequently, the total wirelength, $\theta$, can be calculated as:

$$\theta = \sum_{i,j} w_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2]$$

(3)

which could be rewritten in each of the $x$ and $y$ coordinates as follows [19]:

$$\theta_x = \frac{1}{2} x^T A_x \bar{x} + B_x^T \bar{x} + \text{const.}$$

(4)

The Hessian matrix $A$ represents the connection between movable cells and $B$ denotes the connections between movable and fixed cells. Based on [19], (4) could be reduced to:

$$A_x \bar{x} = -B_x$$

(5)

For the placement problem, an initial random placement of cells is generated. Then, based on the Bound2Bound (B2B) net model [23] and the initial location of fixed and movable cells, $A$ and $B$ matrices are calculated and (5) is solved using Conjugate Gradient method [19][20]. Since there are no constraints in this step and the main objective is to minimize the HPWL, most of the cells may end up sitting on top of each other, resulting in a densely populated area in the middle of the chip. In order to avoid such a placement, an update ratio $\alpha$ is set, such that the new location of the cells is calculated as:

$$x_i = \alpha \cdot x_i^{\text{new}} + (1 - \alpha) x_i^{\text{old}},$$

(6)

in which $x_i^{\text{new}}$ is the solution of (5).

Once the new locations are calculated, $A$ and $B$ matrices are updated and (5) is solved again. This step continues until the HPWL reduction decreases to less than a preset threshold value. During the initial steps, update ratio is relatively large but gradually decreases to avoid too much cell overlap in the chip. This step is usually completed in 5-13 iterations depending on the update ratio and number of cells in the design. Next, based on an algorithm similar to look-ahead legalization (LAL) [19], grids are formed throughout the chip and in each grid cell, based on the location of the fixed and movable cells, pseudo pins are located and matrices $A$ and $B$ are updated. Solving (5), generates the new location of the cells and based on the update ratio, cells are moved toward the lower density areas of each grid cell to remove overlaps. LAL algorithm is performed until the ratio of cell area to whitespace area in each grid cell is less than a preset threshold value (called the overfill ratio). Legalization and detailed placement are then performed to generate the solution with the minimum wirelength. Furthermore, based on the logic level of the cells, placement algorithm modifies the cell locations based on our proposed SFQ-specific placement method.

In this method, cells with the same level are grouped together and cell groups in each row are sorted by their logic level. Furthermore, a linear assignment program [24] is solved to reorder cells in each group to refine the placement and improve HPWL. This method makes the clock routing more efficient as clock could be propagated through cells, from each cell to its adjacent cell using only splitters/JTLs (no PTL is needed).

Clock nets are then connected from the global clock pin (denoted by GCLK) to all cells in the circuit. These new clock nets are also added to the netlist, and this updated netlist is passed to the router tool [22] to complete clock and signal routing. The router algorithm starts with the original placement and tries to connect all the nets. If there are no failed nets, the routing is finished. However, if some nets cannot be connected due to the lack of horizontal or vertical space, the vertical space between each two row ($Y_{Space}$) and the horizontal space between each two cell ($X_{Space}$) are increased. We use a binary search algorithm to find the minimum vertical and horizontal spaces for which there are no failed nets after routing is performed.

We have tested our placement and routing methodology on a 32-bit Kogge-Stone [18] adder, and generated final routing results for conventional and SFQ-specific placements. For the adopted adder, the total number of cells and data nets (signals) are 1669 and 2072, respectively, and the maximum logic level of a cell is 12. The total number of nets added for clock routing in the case of SFQ-specific placement and conventional
Fig. 7. Total HPWL of the chip, considering both data and clock nets (after adding clock nets).

Fig. 8. Total chip area (mm$^2$) for different placement methods and various number of metal layers.

Fig. 9. Part of the layout of the 32-bit Kogge-Stone adder after SFQ-specific placement and routing. GCLK pad represents the input clock pin.

IV. CONCLUSION

We presented a row-based design methodology, where fixed-height cells are placed in rows on the chip. To obtain a low cost and more practical clock network, a space is reserved above each row which uses a combination of splitters/JTLs and PTLs to distribute the clock signal to all cells in a row. If cells are not sorted in the ascending order of their logic levels, PTL wires are needed to complete the clock routing. Accordingly, after obtaining the output of a state-of-the-art conventional placer, we sort cells in each row in the ascending order of their logic levels to eliminate PTL wires within each row. Therefore, most of the PTL resources are available for signal routing which results in smaller number of PTL layers and/or smaller chip area to complete the signal routing by using our SFQ-specific placer compared with a conventional placement tool. Simulation results on a 32-bit Kogge-Stone adder also point to the effectiveness of our proposed design methodology. We also showed that by increasing the number of PTL layers in the process fabrication, chips with smaller area can be achieved. For instance, by increasing the PTL layers from two to three and using our SFQ-specific placement, we can reduce the chip area by 2.5×. As a future work, we will integrate new constraints that capture the logic levels into the mathematical formulation of the force-directed placement in order to come up with the minimum total HPWL while maintaining the ascending order of levels in each row.

REFERENCES


