A Placement Algorithm for Superconducting Logic Circuits Based on Cell Grouping and Super-Cell Placement

Soheil Nazar Shahsavani, Alireza Shafaei, and Massoud Pedram
University of Southern California
{nazarsha, shafaeib, pedram}@usc.edu

Abstract—**This paper presents a novel clustering based placement algorithm for single flux quantum (SFQ) family of superconductive electronic circuits. In these circuits nearly all cells receive a clock signal and a placement algorithm that ignores the clock routing cost will not produce high quality solutions. To address this issue, proposed approach simultaneously minimizes the total wirelength of the signal nets and area overhead of the clock routing. Furthermore, construction of a perfect H-tree in SFQ logic circuits is not viable solution due to the resulting very high routing overhead and the in-feasibility of building exact zero-skew clock routing trees. Instead a hybrid clock tree must be used whereby higher levels of the clock tree (i.e., those closer to the clock source) are based on H-tree construction whereas lower levels of the clock tree follow a linear (i.e., chain-like) structure. The proposed approach is able to reduce the overall half-perimeter wirelength by 15% and area by 8% compared with state-of-the-art techniques.**

I. Introduction

Development of the semiconductor technology has been driven by the demand for higher performance and energy efficient computing for decades. Conventional CMOS-based technology has encountered serious challenges such as high temperature and power consumption [1]. Hence, there is a significant demand to search for new innovations that would permit continuation of performance and energy efficiency scaling to well beyond the end-of-scaling CMOS nodes.

Superconducting electronics (SCE), including *rapid single flux quantum* (RSFQ) logic family, appears to be one of the most promising technologies to replace CMOS devices, primarily due to fast switching and low energy consumption. This is as a result of special attributes of *Josephson junctions* (JJs), basic circuit elements in SFQ logic, such as fast switching ($\sim$ 1 ps) and low switching energy per bit ($\sim 10^{-19}$ J) at low temperatures [2]. In particular, RSFQ technology uses quantized voltage pulses in digital data generation, reproduction, amplification, memorization, and processing [3]. Furthermore, it has been demonstrated that RSFQ circuits are functional at operating frequencies of up to 770 GHz [4].

In spite of extraordinary characteristics of SFQ logic, architectures, design automation methodologies, 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 [2].

Placement problem has been studied for more than 50 years, and it has played an important role in the overall design flow of integrated circuits [5]. Reducing the total wirelength which can be modeled by *half-perimeter wirelength* (HPWL) along with routability and performance are among the main objectives.

The focus of this paper is to propose a new placement methodology for large SFQ circuits, with the primary goal of reducing the total wirelength and area dedicated to clock network. The proposed placement algorithms starts by grouping cells of the same logic level into a set of super-cells, building a modified netlist capturing dependencies among these super-cells, and then using force-directed placement to find a global placement of these cells. A detailed placement follows in which cells within each cell group (super-cell) are re-ordered with the goal of minimizing a cost function comprising of the signal wirelength and the total chip area. The key contributions of this paper are as follows.

- We present a novel clock-tree aware placement algorithm to reduce the total wirelength and chip area, compared with state-of-the-art approaches. Results show an average improvement of 15% and 8% in total HPWL and area.
- The proposed algorithm is capable of generating various placement solutions with different trade-offs between the total wirelength and chip area.

The remainder of the paper is organized as follows. Prior work are discussed in section II. Proposed clock-tree aware placement methodology is presented in Section III. Simulation results and comparison of different placement approaches are reported in Section IV. The paper is concluded in Section V.

II. Prior Work

Various methods for placement and clock tree synthesis of RSFQ circuits have been proposed. In [7], initially a zero-skew H-tree clock network is constructed. Cells are then placed on predefined rectangular grids located at the leaf nodes of the H-tree clock network using the min-cut placement algorithm. This design methodology suffers from the large size of the H-tree clock network which results in a considerable portion of the chip area occupied by clock tree structure. Authors in [8] propose the *HL-tree* clock network to reduce the total chip area. *HL-tree* clock network adopts an *H-tree* to propagate the clock signal globally, to some of the nodes (namely clock sink nodes), and a *linear tree* (L-tree) to distribute the global clock signal to nearby cells locally. This work uses a global placement algorithm to create an initial placement of the cells. Next, cells of the same logic level in each row are grouped together and generated cell-groups are placed in the same row. Although this approach reduces the chip area compared with the initial placement, post placement cell-grouping algorithm leads to a considerable increase in total wirelength, which in turn increases the critical path delay and degrades the performance [8]. This is due to the fact that cell-grouping algorithm only considers same level cells within a row and not all the same level cells in the circuit. Hence, in a *HL-k* placement, a large number of cell-groups with fewer than k cells are generated which increase the clock routing cost.

III. Proposed Method

Different placement algorithms ranging from force-directed to min-cut placements tend to produce high quality solutions (in terms of the minimization of the total wirelength) with very different cell placements. This degeneracy of placement solutions implies that one can optimize another objective function (e.g., the routing cost of the clock tree) while only minimally affecting the primary objective function (e.g., the total wirelength).

Major differences of SFQ compared with CMOS technology may be summarized as follows. 1) Size of the basic gates are huge (i.e., area of an AND gate in SFQ is 600 times larger than that of even a 45-nm CMOS technology [8][9]). 2) Current technology only handles a limited number of metal layers for routing (currently only 3 metal layers are available for both clock and signal routing [10]). 3) While in CMOS-based designs typically near 15% of the cells need clock signal [11], in SFQ all the cells (except for splitters) receive a clock signal, which leads to a large clock network. 4) In SFQ designs, the input netlist should path-balanced[1]. This increases the number of cells in the design, as DFF gates should be inserted for balancing all the paths. 5) Building a perfect H-tree is nearly impossible. Large number of clock splitters in the H-tree network increase the clock skew. Process variations result in increased clock jitter. Furthermore, the layout rules are more constrained in SFQ technology than CMOS which lead to asymmetries in clock network [3]. Above reasons lead to higher design complexity, large area especially in clock network, and significant performance degradation due to longer critical path delay.

This paper presents a clock-tree aware algorithm to produce high quality placement solution which minimizes the total wirelength while reduces the clock network area. Proposed approach utilizes the HL-tree clocking scheme [8]. An example of HL-tree clock network is shown in Fig. 1. The proposed placement algorithm, recognizing the need for this hybrid clock tree realization, generates a cell placement that intrinsically matches the structure of the clock tree. It does so by efficiently grouping cells of the same logic level into cell-groups (super-cells), building a modified netlist capturing dependencies among these cell-groups, which in turn minimizes the total number of clock sink nodes. Note that the reason that only same level cells are included in each super-cell is that if there is a connection between cells in a L-tree network, stage delay is added to clock cycle delay and degrades the overall performance of the circuit.

A. Design Flow

Our placement algorithm consists of three main phases, as shown in Fig. 2. Phase 1 (*Initial Cell Placement*) starts by placing cells using a force-directed global placement algorithm.

[1]In a path-balanced netlist, all paths from any primary input to any primary output have the same logical depth.

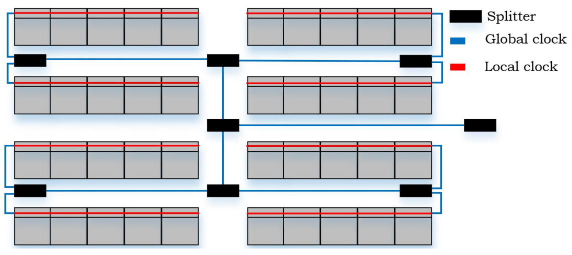

Fig. 1. A sample HL-tree clock network. Black boxes indicate clock splitters. Blue and red lines denote the global and local clock signals.

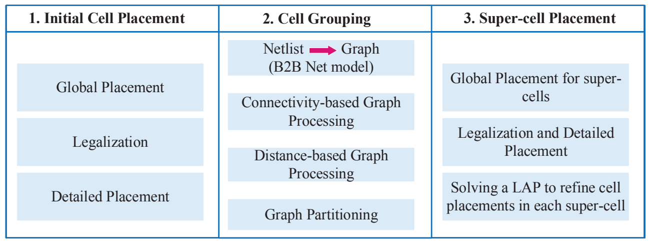
1. Initial Cell Placement
2. Cell Grouping
3. Super-cell Placement
Global Placement
Legalization
Detailed Placement
Netlist → Graph (B2B Net model)
Connectivity-based Graph Processing
Distance-based Graph Processing
Graph Partitioning
Global Placement for super-cells
Legalization and Detailed Placement
Solving a LAP to refine cell placements in each super-cell

Fig. 2. Overall view of the proposed placement algorithm. LAP and B2B stand for linear assignment problem [12] and bound to bound net model [13], respectively.

In phase 2 (*Cell Grouping*) circuit is transformed to a graph $G(V, E)$, in which V denotes the set of all vertices (cells), and E represents set of all hyperedges (nets). In case of multi-pin nets, we use bound to bound net model [13], as it captures the HPWL objective function better than the clique and star net models [13]. Additionally, (for each logic level i) a sub-graph of the original graph which only contains cells of that logic level (namely G_i) is created. Initially, there are no connections between cells of same logic level. Furthermore, nodes of each sub-graph G_i are processed in two steps, namely connectivity-based and distance-based graph processing (cf., Fig. 2 Phase 2). The goal of these two steps is to add connections between cells of the same logic level (nodes of graph G_i) to facilitate grouping of same level cells. Subsequently, each sub-graph G_i is partitioned, and cell-groups consisting of strongly connected cells of same logic level are formed. In the final phase (*Super-cell Placement*), cell-groups (super-cells) are placed on the placement grid and a detailed placement algorithm reorders cells inside each super-cell to the minimize the total HPWL (cf., Fig. 2 Phase 3). Details of the proposed design flow are described in the following subsections.

B. Connectivity-based Graph Processing

Once the initial placement of the cells is produced by the first phase of the proposed approach, netlist is transformed to a graph and sub-graph G_is are formed. In this step, weights are added to edges between the nodes of sub-graph G_i solely based on the connectivity of nodes to their adjacent level neighbors (as initially there are no connections between nodes of G_i, a zero-weight edge is added between each pair of nodes). The intuition for this step is that if two nodes of logic level i have lots of common neighbors, it is desirable for the global placement to place them close to each other. As a result,

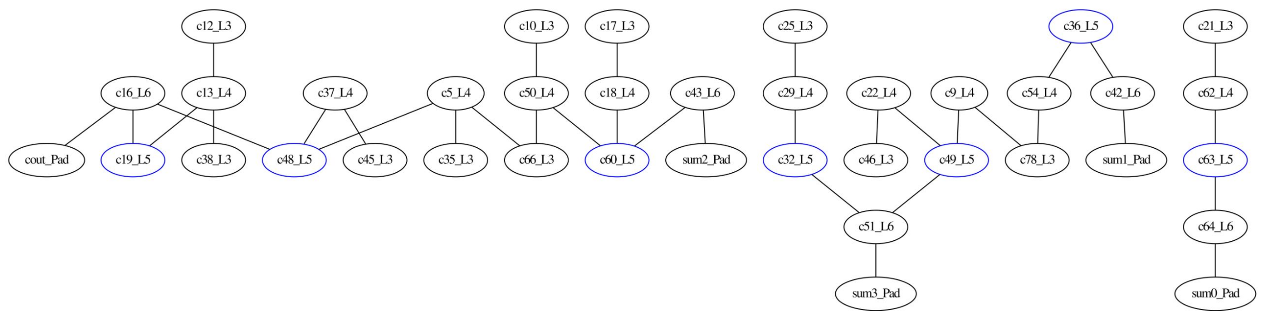

Fig. 3. Graph of nodes of level 3-6 for a 4-bit Kogge-Stone adder. Nodes of level 5 are shown in blue. Logic level of each node is added to its name.

to reduce total HPWL, we may group them together. This step runs for each of the sub-graphs G_i independently. Therefore it is performed L times, where L is the total number of logic levels. Details are provided below.

A pre-processing step finds the neighbor nodes of all cells in the design using Algorithm 1. This algorithm gets the base node and *maxSearchLevel* (a parameter which determines the search scope and is same for all the nodes) as input and returns a two dimensional vector, namely *neighbors*, consisting of the neighbors which have a level within *maxSearchLevel* (mSL) of the base node. Algorithm 1 starts by initiating an empty queue. Initially, the base node is pushed to the queue. In each iteration, front element of the queue is chosen and added to the two dimensional vector, *neighbors*, with the difference of its logic level and that of the base node as the index of first vector. Next, children (parents) of the front element in the queue which have a level within mSL of the base node are added to the queue. Finally, once all neighbors of the base node within its mSL are added to the *neighbors* vector, algorithm terminates. For instance, assume the base node is of level i, and mSL is equal to 2. The algorithm returns all nodes with level more than i-3 (parents) and less than i+3 (children) of the base node, which have a direct connection toward the base node. Direct connection from a child (parent) to its base node means that it can only pass through nodes of logic level lower (higher) than its own logic level, only from output to input (input to output).

Once this pre-processing step is completed for all nodes, each sub-graph G_i is processed as described in Algorithm 2. At each step a pair of nodes of level i, namely u and v are processed. Initially, their common neighbors is found by intersecting their *neighbors* vectors. If there is a common neighbor of level p, weight equal to $\frac{\alpha}{|p-i|}$ is added to the edge between them. α represents a normalization factor, directly proportional to mSL.

Fig. 3 shows nodes of level 3-6 for a 4-bit Kogge-Stone adder circuit with 6 logic levels. Assume Algorithm 2 processes nodes of level 5 (shown in blue in Fig. 3) with mSL of 2 and α value of 2. Consequently, edge weights in graph G_5 are modified as follows. Nodes *c60* and *c48* have a common neighbor of level 3, namely *c66*. Consequently, weight of the edge between them increases by 1. Same edge weight is added between nodes *c36* and *c49*. Additionally, *c51* is a common incident neighbor for nodes *c49* and *c32*, which results in an edge weight of 2 between these two nodes. Edge weight between nodes *c19* and *c48* is also increased by 2. Finally, since *c63* does not have any common neighbors with the other nodes (it is one of the DFFs inserted for path-balancing), it remains unconnected. This step is performed for all the sub-graph $G_i s$. After this step, the sub-graphs are passed to the next step (*Distance-based Graph Processing*).

Algorithm 1 Level Order Traversal

```
int maxSearchLevel, Node* base
vector<vector<Node*>> neighbors
queue<Node*> Q = $\phi$
Q.push(base)
while (!Q.empty()) do
    Node* front = Q.front()
    Q.pop()
    levelDiff = abs(level(front) - level(base))
    if (levelDiff > maxSearchLevel) continue
    neighbors[levelDiff].push_back(front)
    vector< Node* > children = findChildren(front)
    vector< Node* > parents = findParents(front)
    foreach node in {children, parents}
        if (abs (level(node) - level(base)) $\leq$ maxSearchLevel)
            Q.push(node)
    end foreach
end while
return neighbors
```

Algorithm 2 Connectivity-based Graph Processing

```
vector<Node*> sameLevelNodes = $G_i$(V)
foreach u in sameLevelNodes
    foreach v in sameLevelNodes
        commonNeighbors = intersect(neighbors(u), neighbors(v))
        foreach neighbor in commonNeighbors
            levelDiff = abs(level(u) - level(neighbor))
            extraEdgeWeight = $\frac{\alpha}{levelDiff}$
            addEdgeWeight(u, v, extraEdgeWeight)
        end foreach
    end foreach
end foreach
```

C. Distance-based Graph Processing

In this step, initial cell placement information (cf. Fig. 2 Phase 1) will be used to add edge weights between nodes of sub-graph G_i as a function of their relative distance. Edge weight between each pair of same level cells (u, v) is calculated using following formula:

$$W_x(u,v) = \beta \cdot (1 - \frac{|X_u - X_v| - \Delta X_{i_{min}}}{\Delta X_{i_{max}} - \Delta X_{i_{min}}}) \qquad (1)$$

where $\Delta X_{i_{max}}$ and $\Delta X_{i_{min}}$ represent maximum and minimum distance of cells of logic level i, X_u and X_v denote the x coordinates of u and v, and β is a normalization factor. Edge weight in y direction is calculated in a similar manner. Final edge weight between nodes u and v is calculated as follows:

$$W(u,v) = \lfloor \sqrt{W_x(u,v)^2 + W_y(u,v)^2} \rfloor \qquad (2)$$

where $W_x(u,v)$ and $W_y(u,v)$ denote the edge weights corresponding to x and y directions. The intuition for this step is that if two nodes of logic level i are placed near each other in the initial placement, it is desirable to place them close to each other in the final solution. As a consequence of this phase, none of the nodes of G_i will be unconnected. Hence, cell-grouping will be more deterministic rather than random in the case there are unconnected nodes which can be placed in any of the cell-groups.

D. Partitioning

Once each sub-graph (G_i) is processed through connectivity-based and distance-based steps, a partitioning algorithm is used to partition it to generate cell-groups. As a result, cells with higher edge weight end up in the same part and are grouped to form the super-cells. Assuming the total number of cells of level i to be m and the the group sizes to be k, we seek to obtain $p = \lceil m/k \rceil$ cell-groups. For this purpose, a p-way partitioning should be performed to generate p parts of size k. After the partitioning a super-cell is created for each part and all nodes in that part are added to that super-cell. Once partitioning all sub-graphs (G_i) is completed, and super-cells are created, connections among nodes in the original graph are transformed to connections among super-cells in the reduced graph.

It should be noted that as the number of cells in each group (k) increases, the overall chip area decreases. The reason is that more cells of same logic level are grouped (abutted) and total number of clock sink nodes decreases. On the other hand, using larger values of k increases the delay of linear clock propagation inside each group [8]. Therefore, increasing group size creates a trade-off between total chip area and maximum clock frequency. However, in current technology, critical path delay is much higher than that of linear clock distribution. Consequently, increasing number of cells in each group leads to smaller chip area with negligible performance penalty.

E. Super-cell Placement

Once the *Cell Grouping* phase (cf. Fig. 2) is completed and super-cells are generated, original netlist is transformed into a smaller netlist in terms of total cell count. If the total number of cells in original netlist is N, and group size is set to be k, the new netlist contains approximately N/k cells.

In this phase, namely *Super-cell Placement* (cf. Fig. 2), reduced netlist is placed using a global placement algorithm, followed by legalization and detailed placement. Once the position of all super-cells is determined, original netlist is retrieved, and position of original cells is updated based on the position of their corresponding super-cell. They are initially placed in the center of each super-cell, on top of each other. Subsequently, a *linear assignment problem* (LAP) [12] is solved to find the optimal placement of each cell inside its corresponding cell-group similar to [8].

IV. Simulation Results

Our implementation was written in C++. We used SimPL [6] placement algorithm for performing global placement and HMETIS package for p-way partitioning [14]. The effectiveness of our approach was evaluated using 8 different SFQ benchmarks obtained from [15] and [16], with group sizes of $k = 2$ and $k = 4$. *HL-tree* clocking scheme was used to synthesize the clock network [8]. Once the proposed placement algorithm terminates, first cell of each cell-group is marked as the clock sink. Next, BST/DME algorithm [17] is used to create an *H-tree* clock network to propagate clock signal to all the sink nodes. Splitters on the top portion of each cell within a cell-group are used to create *L-tree* clock network. Finally, total HPWL was calculated using the GSRC Bookshelf Evaluator [18]. It should be noted that total HPWL has been reported as sum of data and clock signal nets. The approach in [8] was implemented and used as the baseline solution.

Table I compares the results of our method to the baseline solution. Proposed approach is able to improve the total HPWL and total area by 15% and 8%, respectively. This is primarily due to the *Cell Grouping* phase (cf. Fig. 2). Cell-grouping in baseline solution is only limited to same level cells within each row, while proposed solution considers all the cells of same logic level globally, irrespective of the row at which they are placed during global placement. Consequently, fewer number of clock sink nodes are created which in turn reduces the total area and total HPWL simultaneously. Proposed approach reduces the total number of clock sink nodes approximately to N/k in the case of group size of k (cf. Table I). However, baseline solution generates lots of cell-groups with fewer cells than k in each cell-group. Consequently, total area of the proposed approach is always smaller than that of [8]. Moreover, Table I shows that proposed methodology reduces the overall area significantly, using the *HL-tree* clocking scheme, when compared with a global placement accompanied by *H-tree* clock network as proposed in [8]. Comparison of total chip area for different benchmarks, and different group sizes (k), shows that total area can be reduced significantly, 27.8% on average, using the proposed clock-tree aware placement approach, which in turn reduces the critical path delay and increases the performance.

V. Conclusion

Special attributes of *rapid single flux quantum* (RSFQ) technology such as fast switching and low energy consumption,

TABLE I. Comparison of total number of clock sink nodes, percentage of HPWL and area improvement for 8 different SFQ logic circuits using different placement strategies. KS stands for Kogge-Stone adder and INT-DIV stands for integer divider.

Benchmark	# Cells	# Levels	Group size (k)	# Clock sinks		% HPWL improv.	% Area improvement	
				baseline	proposed	prop. vs. base.	prop. vs. base.	HL-k vs. Htree
16-bit KSA	500	10	2	353	317	3.7	1.3	23.1
			4	225	161	1.0	11.6	37.0
iscas85/C499	704	14	2	643	443	3.0	2.1	21.3
			4	505	224	-2.5	7.2	36.4
32-bit KSA	1208	13	2	893	829	8.1	6.2	24.1
			4	523	422	6.4	5.7	35.1
iscas85/C880	1299	24	2	906	776	13.0	0.5	20.9
			4	613	394	3.6	7.0	31.2
iscas85/C1908	1310	26	2	923	781	6.2	8.6	26.2
			4	649	398	-2.9	11.0	34.8
iscas85/C432	1569	43	2	1101	867	8.5	13.2	25.4
			4	854	443	-2.5	14.6	36.4
8-bit INT-DIV	5431	89	2	3487	2871	45.8	4.8	26.9
			4	2475	1458	44.9	5.5	35.9
16-bit INT-DIV	15315	305	2	11936	8242	56.3	16.4	21.4
			4	10619	4196	52.4	23.6	33.7
Average(%)	-	-	-	-	-	**15.3**	**8.6**	**27.8**

makes it one of the best replacements for CMOS-based design. However, large cell sizes, limited number of metal layers in current technology [10] and the fact that all the cells in SFQ technology need a clock signal, require special algorithms to improve the performance and reduce the total area in large SFQ circuits, especially the area dedicated to clock network. In this paper, a novel cell-grouping based placement approach is introduced and effectiveness of the proposed method is compared with state-of-the-art techniques. Proposed approach is able to improve the total HPWL and total area by 15% and 8%, respectively.

ACKNOWLEDGMENT

The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

REFERENCES

[1] N. Z. Haron and S. Hamdioui, “Why is cmos scaling coming to an end?” in *International Design and Test Workshop*, Dec 2008.

[2] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, “Energy-efficient superconducting computing power budgets and requirements,” *IEEE Trans. on App. Supercond.*, June 2013.

[3] K. K. Likharev and V. K. Semenov, “RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock-frequency digital systems,” *IEEE Trans. on App. Supercond.*, Mar 1991.

[4] W. Chen *et al.*, “Rapid single flux quantum T-flip flop operating up to 770 GHz,” *IEEE Trans. on App. Supercond.*, Jun 1999.

[5] I. L. Markov, J. Hu, and M. C. Kim, “Progress and challenges in vlsi placement research,” *Proceedings of the IEEE*, Nov 2015.

[6] M.-C. Kim, D.-J. Lee, and I. L. Markov, “Simpl: An effective placement algorithm,” *TCAD*, Jan. 2012.

[7] Y. Kameda, S. Yorozu, and Y. Hashimoto, “A new design methodology for single-flux-quantum (sfq) logic circuits using passive-transmission-line (ptl) wiring,” *IEEE Trans. on App. Supercond.*, June 2007.

[8] S. N. Shahsavani *et al.* “An integrated row-based cell placement and interconnect synthesis tool for large sfq logic circuits,” *IEEE Trans. on App. Supercond.*, June 2017.

[9] “Webpage for freepdk45.” [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[10] S. K. Tolpygo *et al.*, “Fabrication process and properties of fully-planarized deep-submicron nb/al josephson junctions for vlsi circuits,” *IEEE Trans. on App. Supercond.*, June 2015.

[11] D.-J. Lee and I. L. Markov, “Obstacle-aware clock-tree shaping during placement,” *TCAD*, Feb. 2012.

[12] R. E. Burkard and E. Çela, *Linear Assignment Problems and Extensions*. 1999.

[13] P. Spindler, U. Schlichtmann, and F. M. Johannes, “Kraftwerk2 ;a fast force-directed quadratic placement approach using an accurate net model,” *TCAD*, Aug 2008.

[14] G. Karypis and V. Kumar, “Multilevel k-way hypergraph partitioning,” in *DAC*, 1999.

[15] N. Katam *et al.* Sport lab sfq logic circuit benchmark suite, technical report, USC, CA, USA, 2017. [Online]. Available: Available: http://sportlab.usc.edu/downloads/sfq_benchmarks/

[16] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the iscas-85 benchmarks: A case study in reverse engineering,” *IEEE Design and Test of Computers*, vol. 16, no. 3, Jul. 1999.

[17] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, “Bounded-skew clock and steiner routing,” *TODAES*, Jul. 1998.

[18] S. N. Adya and I. L. Markov, “Executable placement utilities, 2005.” [Online]. Available: http://vlsicad.eecs.umich.edu/BK/PlaceUtils