Balanced Factorization and Rewriting Algorithms for Synthesizing Single Flux Quantum Logic Circuits

Ghasem Pasandi

Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, California, USA
pasandi@usc.edu

Massoud Pedram

Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, California, USA
pedram@usc.edu

ABSTRACT

Single Flux Quantum (SFQ) logic with switching energy of $100zJ^1$ and switching delay of $1ps$ is a promising post-CMOS candidate. Logic synthesis of these magnetic-pulse-based circuits is a very important step in their design flow with a big impact on the total area, power consumption, and critical path delay. SFQ circuits has some properties different from CMOS which should be taken into consideration in the design and implementation flow of these circuits. One of these properties is requirement of path balancing in the standard SFQ circuit design. Standard CMOS-based rewriting and factorization algorithms fail to preserve the balancing property of SFQ circuits. Therefore, they end up generating circuits with huge path balancing overheads. Our proposed balanced factorization and rewriting algorithms are designed specifically to solve this problem. Experimental results show that a combination of balanced factorization and rewriting algorithms reduces the path balancing overhead by an average of 63% for 15 benchmark circuits, and area by up to 23% compared to state-of-the-art logic synthesis tools.

CCS CONCEPTS

• Computer systems organization; • Hardware → Logic synthesis; Combinational synthesis;

KEYWORDS

AIG, Factorization, Logic Synthesis, Refactoring, Rewriting, RSFQ, SFQ, Single Flux Quantum, SDE, Superconducting Digital Electronics, Technology Mapping

ACM Reference Format:

Ghasem Pasandi and Massoud Pedram. 2019. Balanced Factorization and Rewriting Algorithms for Synthesizing Single Flux Quantum Logic Circuits. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19)*, May 9–11, 2019, Washington, D.C., USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3299874.3317967>

1 INTRODUCTION

Shrinking CMOS devices has provided a decades-long advances in computing. However, by increasing challenges to scaling of these devices and by slowing the Moore's law down (end of Moore's law),

¹1z is equal to 10^{-21} .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '19, May 9–11, 2019, Washington, D.C., USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6252-8/19/05...\$15.00

<https://doi.org/10.1145/3299874.3317967>

new device, circuit, architectural, and system level solutions are needed to keep up with ever increasing demand for energy efficient and high speed circuits and systems [1]. Superconducting Digital Electronics (SDE), especially the Single Flux Quantum (SFQ) family, is proven to be a good candidate to achieve energy efficient and high performance systems [2], hence, a potential replacement for CMOS technology. SFQ devices are made of Josephson junctions (JJs), which are superconducting devices working based on the Josephson effect [3]. The switching delay of these devices is as low as $1ps$ with consuming $100zJ$ energy per switching [4]. This switching energy is even less than what is predicted by International Technology Roadmap for Semiconductors (ITRS) for 2020 [1], which again shows the prominence of these devices as a replacement for CMOS technology.

The first family of the SFQ logic is called Resistive (later on Rapid) Single Flux Quantum (RSFQ) logic, which was developed in the 1980s and uses resistors (JJs in the latest version) to regulate the biasing current to JJs [5]. RSFQ gates can operate as fast as 370GHz at $T = 4.2K$ [4]. In [6], the maximum speed of 770GHz is also reported for an RSFQ T-Flip-Flop (TFF).

While superiority of SFQ circuits in achieving super fast and very low-power circuits has been proven, there is no optimum full design automation suite for these devices. On the other hand, due to some key differences between SFQ and CMOS logic, CMOS Computer Aided Design (CAD) tools cannot be directly used for SFQ circuits. Therefore, it is critical to develop appropriate CAD tools including synthesis, place-and-route, timing and power analysis, and verification tools to fully automate the design and test process of SFQ circuits.

In this paper, we present balanced factorization and rewriting algorithms, which are designed to synthesize SFQ circuits efficiently. Unlike the standard factorization and rewriting algorithms, our proposed algorithms preserve balance of the given circuit during the logic synthesis. As a result, a huge reduction in path balancing overheads of these circuits is achieved, which is translated into saving in the total area. Please note that the proposed algorithms would be of interest for any technology that requires path balancing or adheres to a clocked propagation of data (gate-level pipelined, see Section 2.1.1) through out a circuit.

2 PRELIMINARIES

2.1 Background on SFQ

SFQ logic uses a single quantum of magnetic flux ($\Phi_0 = h/2e = 2.07mV \times ps$) to represent binary information. In this representation, the presence of a pulse is considered as "logic-1", while the absence of a pulse means "logic-0". Operation of SFQ logic is based on over-damped Josephson junctions, and hence, it does not experience the problem of hysteric I-V characteristics (IVCs), which degrades the

Figure 1. Schematic of an SFQ buffer, its IVC and the pulse shape representation of data in SFQ (waveforms are adopted from [7]).

operation speed of “1” to “0” switching [7]. Figure 1 shows the IVC and the pulse shape representation of data in an SFQ buffer.

Here is a couple of gate/circuit level properties of SFQ circuits. For more details please see [5, 8–10]:

2.1.1 Gate-level pipeline. In SFQ logic, most of gates (except for splitters, confluence buffers, IO cells, and TFFs) receive a clock signal. There are three main methods for clock distribution in SFQ circuits: (i) *clock-follow-data* where a clock signal arrives at a gate when its inputs have already been arrived and processed by the gate, (ii) *counter-flow clocking* where the clock flows in the opposite direction of the data, and (iii) *concurrent-flow clocking* in which the clock and data flow in the same direction.

2.1.2 Path Balancing. In standard SFQ circuit design, to guarantee the correct functionality, length of any path in terms of the gate count from any Primary Input (PI) to any Primary Output (PO) of the circuit should be the same. Otherwise, some path balancing D-Flip-Flops (DFFs) should be inserted into shorter paths to balance the whole circuit. If an SFQ circuit is not path balanced, there will be at least one gate with an early (late) input. This input will be consumed in a wrong clock cycle and causes generation of a potentially wrong output, which can propagate to the final outputs of the circuit.

2.2 Background on Logic Synthesis

Logic synthesis is divided into *technology-independent* and *technology-dependent (technology mapping)* phases. In the first phase, several transformations are performed to reduce the number of literals in the final form of a given Boolean expression. In the second phase, which is the final stage of the logic synthesis, suitable gates from a given library are assigned to nodes of the network in order to satisfy some constraints or to minimize some cost functions. Before technology mapping, the Boolean network is transformed into And Inverter Graphs (AIGs). This step is called *technology decomposition*, and such a network is called the *subject graph*.

A *k-feasible cone* at node v of a network $N = (V, E)$, denoted by C_v , is defined as a sub-graph containing v and its predecessors satisfying two conditions: (i) number of inputs of this sub-graph should be fewer than or equal to k , (ii) all paths connecting v to a node in C_v lies entirely in C_v . A cut $C = (X, X')$ with source s and sink t in a given network N is defined as a partition of the set V into

two sets X and $X' = V - X$ such that $s \in X$, and $t \in X'$. $C = (X, X')$ is a trivial cut, if set X has only one member (source s). The *node cut-size* of a cut $C = (X, X')$ denoted by $n(X, X')$ or $n(C)$ is defined as the number of boundary nodes in X' which are adjacent to some nodes in X . These boundary nodes are called the *leaf nodes* of the cut. A cut $C = (X, X')$ is called *k-feasible* if its node cut-size is at most k (i.e. $n(C) \leq k$). A *k-feasible cut* of a node v is defined as a valid *k-feasible* cut, in which node v is the source node of the cut and the sink node is a PI. A *fanin (fanout) cone* of a node v in a network $N = (V, E)$ is defined as the set of nodes in V that can be reached through the fanin (fanout) edges of v .

A *literal* is a variable or its negation, e.g. x, x' . A *cube* is AND (conjunction) of a set of literals. An *algebraic expression* $F = \{C_i\}$, which consists of a set of cubes, is an expression in which no cube contains another one, i.e., $C_i \subseteq C_j, i \neq j$. An expression that is not algebraic is called *Boolean*. The *support* of an expression F is the set of variables that F explicitly depends on. The *product of two cubes* C_i and C_j is a cube defined by the following:

$$C_i C_j = \begin{cases} \emptyset & \text{if } \exists x \in C_i \cup C_j \text{ and } x' \in C_i \cup C_j \\ C_i \cup C_j & \text{otherwise} \end{cases}$$

The *product of two Sum-Of-Product (SOP) expressions* F and G is a SOP expression denoted by FG , defined by $FG = \{C_i C_j \mid C_i \in F \text{ and } C_j \in G \text{ and } C_i C_j \neq \emptyset\}$. The product FG is an *algebraic product* if F and G are algebraic expressions and have disjoint variable support. The *sum of two SOP expressions* F and G denoted by $F + G$ is a set defined by $F + G = \{C_i \mid C_i \in F \text{ or } C_i \in G\}$. The sum $F + G$ is an *algebraic sum* if F and G are algebraic expressions and no cube in F contains a cube in G and vice versa. A *factored form* is defined recursively as follows: a factored form is either a product or sum where a product (sum) is either a single literal or product (sum) of factored forms. Also, a factored form can be defined as a parenthesized algebraic expression. For example, $ab+c(a+b)$, a , acd' are factored forms, but $b(c+d)'$ is not a factored form.

3 RELATED WORK

Brayton [11] presented a few methods for obtaining different factored forms of a given logic expression. These methods range from fast purely algebraic methods to Boolean ones, which are slower but capable of providing more saving in the total literal count. Iman and Pedram [12] addressed the problem of reducing power consumption by extending algebraic procedures for node extraction and factorization targeting minimization of a cost function that measures the power cost of Boolean expressions. Roy *et. al* [13] introduced a unique cost function based on decomposed factored forms representation of a given Boolean expression to guide clustering and factorization methodologies for minimizing the power consumption.

Mishchenko *et. al* [14] presented the Directed Acyclic Graph (DAG)-aware AIG rewriting algorithm, by extending the DAG-aware circuit compression method in [15]. In DAG-aware AIG rewriting, 4-feasible cuts of nodes are computed and function of each cut together with its NPN-class² is determined using a hash-table lookup. According to [16], for 4-input expressions, there are

²Two expressions F and G are NPN-equivalent or belong to the same NPN-class, if F can be achieved from G by Negation, Permutation of its inputs or Negation of its output.

222 NPN-classes, among which around 100 are being used more frequently in most of the well-known benchmark circuits [14], which can be pre-computed and stored.

Soeken and Thomsen [17] showed that it is possible to use standard expression rewriting rules to derive fairly complex formulas which is beneficial in algebraic operations used in reversible logic circuits. Haaswijk *et. al* [18] showed that the use of exact synthesis for logic rewriting can be improved by a better sub-network selection strategy, avoiding useless enumerations, and employing XOR majority graphs. In this paper, we present balanced factorization and rewriting, which are the first structural factorization and rewriting approaches to the best of our knowledge.

4 BALANCED FACTORIZATION AND REWRITING

4.1 Balanced Factorization

In the standard method of computing algebraic factored forms, given $F = G_1 G_2 + R$, a factorization value is calculated as follows:

$$\begin{aligned} fact_val &= lits(F) - (lits(G_1) + lits(G_2) + lits(R)) \\ &= (|G_1| - 1) \times lits(G_2) + (|G_2| - 1) \times lits(G_1) \end{aligned} \quad (1)$$

in which, it is assumed that G_1 , G_2 , and R are algebraic expressions, $lits(F)$ returns literal count of F , and $|G_1|$ is the number of cubes in the SOP form of G_1 . This value represents the number of literals that is saved by performing the corresponding factorization. The literal saving is translated into the node count reduction in the subject graph and it results in total area saving for CMOS circuits. Therefore, if a factored form gives the highest factorization value among all possible factored forms, it has a good chance to minimize the total area in CMOS circuits. However, in SFQ circuits, due to requirement of path balancing, the said standard method for selecting the best factored forms will not necessarily result in a circuit with the least area. It even can increase the total area compared with the case of not applying any factorization algorithms. This is because the standard factorization algorithm does not preserve the balance of the network being generated during the factorization process.

To solve the above problem, we propose a balance preserving factorization algorithm, and consider the cost of path balancing when calculating the factorization value. More precisely, we first generate a factoring tree for each factored form of a given expression. Factoring tree is a labeled tree in which each node is labeled with either $\times$ or $+$ expressing a conjunctive, and disjunctive operations, respectively. Also, in a factoring tree, each leaf node is a literal. Next, we compute level³ of each node in the said factoring tree. Then, we compute an imbalance factor (denoted by Λ) for this factoring tree. To define Λ , first we need to define the imbalance factor of a node in a factoring tree. If the maximum level of immediate fanin nodes of the node v in a factoring tree is L_{max} , the imbalance factor of this node is calculated as follows:

$$\lambda_v = \sum_{v_i \in fanins(v)} (L_{max} - level(v_i)) \quad (2)$$

where $fanins(v)$ is the set of immediate fanin nodes of node v , and $level(v_i)$ is the level of one of these fanins. The imbalance factors of leaf nodes are 0. The imbalance factor of a tree $t = (V_t, E_t)$ denoted by Λ_t is calculated by summing up the imbalance factors of nodes

³Level of a node v in a network $N = (V, E)$ which is modeled by a DAG, is defined as the length of the longest path in terms of the node count from any leaf node to v .

Algorithm 1: Balanced Factorization

Input: A Boolean network $N = (V, E)$
Output: Optimized network $N_{OPT} = (V_{OPT}, E_{OPT})$
 // pre-processing:
 1 Convert the given network into AIG;
 2 Sort nodes in V to be in topological order;
 3 Compute k-feasible cuts of each node;
 4 **for each node v in V do**
 5 **for each cut C_j in k-feasible cuts of v do**
 6 Extract function of v based on inputs of C_j ;
 7 Generate factoring trees and compute their $blnc_overhead_val$ (Eq (4));
 8 Select a factored form for node v with the least value for $blnc_overhead_val$;
 // Generate a new network using the best factored forms for each node:
 9 $N_{OPT} = Generate_Ntk(N)$;
 10 **return** N_{OPT} ;

of this tree:

$$\Lambda_t = \sum_{v \in V_t} \lambda_v \quad (3)$$

Finally, to consider the effect of both node (or literal) count and imbalance factor, a new factorization value is defined and computed as follows:

$$blnc_overhead_val = |V_t| + \Lambda_t \quad (4)$$

where $|V_t|$ is the total node count of the factoring tree t . Please note that unlike standard factorization method, in which a factored form with maximum value for $fact_val$ is selected, in balanced factorization, we select a factored form which minimizes $blnc_overhead_val$. This is because a factored form with smaller value for $blnc_overhead_val$ consumes fewer number of nodes and has smaller path balancing overhead (smaller value for imbalance factor).

Example 1: Suppose $F = age + agf + bge + bgf + cdge + cdgf$. One possible factoring tree for this expression is shown in Figure 2. Total node count of this tree is equal to six and its imbalance factor is $\Lambda = 0 + 0 + 1 + 1 + 2 + 1 = 5$. Therefore, the balanced factorization value of this factored form is equal to 11.

Figure 2. A factoring tree for the expression given in the example 1.

Algorithm 1 shows pseudo code of balanced factorization. As seen, after converting the given network into AIG representation (line 1), nodes are sorted to be in a topological order (a node will be visited only when all of its predecessors have been visited) in line 2, and k -feasible cuts ($k=4$) for each node is computed (line 3). Then, a factored form which gives the least balanced factorization value is selected for each node (lines 4–8), and finally, the optimized network is generated in line 9.

4.2 Balanced Rewriting

Balanced rewriting is a step in the technology-independent combinational logic synthesis flow of SFQ circuits which uses fast local transformation of AIG nodes. Input to this algorithm is an AIG representation of the given network. In the balanced rewriting algorithm similar to the DAG-aware AIG rewriting in [14], the given AIG is traversed in a topological order starting from leaf nodes of the graph. For each node v in this graph, 4-feasible cuts are computed as in [19]. For each computed cut of node v , all pre-computed 4-variable Boolean expressions that can implement the function of this node is tried. Different from the DAG-aware AIG rewriting algorithm presented in [14], in balanced rewriting algorithm, in addition to extracting the node count of the new rewritten expression, we calculate an imbalance factor for the corresponding sub-graph (similar to Section 4.1). To calculate the saving that is achieved from the rewritten version of the Boolean function of node v , we keep track of the sum of the current number of nodes and imbalance factors in the current sub-graph (m_1) and the new value for this sum after substituting a candidate sub-graph (m_2). If $m_2 < m_1$, there is a positive gain and the new rewritten version will be accepted. After visiting all nodes of the network and trying the rewritten version of their Boolean function, the final optimized AIG is constructed by tracing back the best solutions for nodes connected to POs (reverse topological ordering traversal).

5 EXPERIMENTAL RESULTS

The balanced rewriting and factorization algorithms are implemented inside an open source logic synthesis and verification tool called ABC [20]. Two sets of experimental results are extracted. The first set is node count, imbalance factor and sum of them before performing technology mapping, and the second set is total area and logical depth after finishing the technology mapping. The baseline for comparing our technology-independent optimizations is ABC's built-in optimization scripts including `resyn`, `resyn2`, `resyn2a`. For mapping, the standard cut-based technology mapping command (`map`) of ABC is employed, and an SFQ library of gates as in [21], consisting of `and2`, `or2`, `xor2`, `DFF`, `splitter`, and `inverter` gates are used, and several ISCAS [22], EPFL [23], MCNC [24], and arithmetic benchmark circuits are considered.

Notice that similar to the definition of imbalance factor for a factoring tree in Section 4.1, we can define an imbalance factor for a DAG too. Also, to compare different technology-independent optimization algorithms, we assign a total cost to a subject graph. The total cost of a subject graph is defined as sum of its imbalance factor and total number of nodes in its AIG representation.

Interleaving balanced rewriting and factorization commands together with `balance` command of ABC⁴, provides higher reduction in total node count and imbalance factor of a given subject graph. Moreover, our experiments show that the order of applying

⁴This command performs algebraic AND-balancing [25].

Table 1. Different technology-independent optimization scripts.

script	Corresponding sequences of commands
ABC's <code>resyn</code>	<code>b; rw; rwz; b; rwz; b</code>
ABC's <code>resyn2</code>	<code>b; rw; rf; b; rw; rwz; b; rfz; rwz; b</code>
ABC's <code>resyn2a</code>	<code>b; rw; b; rw; rwz; b; rwz; b</code>
Our <code>blnc_syn1</code>	<code>brw; brf; b</code>
Our <code>blnc_syn2</code>	<code>brf; b; brw; brwz; b; brfz; brwz; b</code>

these optimization commands can make a huge difference for some benchmark circuits. In the following, we provide an example from [26] to demonstrate this.

Example 2: Suppose $F = abg + acg + adf + aef + afg + bd + be + cd + ce^5$. Applying standard rewriting, balance, and refactoring commands with this order will generate an optimized AIG as shown in Figure 3a with eight nodes, imbalance factor of four and total cost of 12. Applying ABC's `resyn2` script (see Table 1) will generate the subject tree shown in Figure 3b with nine AIG nodes, imbalance factor of three and total cost of 12. However, if balanced rewriting, balanced factorization, and balance command of ABC are applied with this order, the resulting subject tree will be as shown in Figure 3c which is perfectly balanced, has seven AIG nodes, imbalance factor of 0, and total cost of seven.

We introduce two sets of optimization scripts called `blnc_syn1` and `blnc_syn2`, and also considered three optimization scripts of ABC. Table 1 shows these scripts and their corresponding sequence of commands. In these scripts, `b` is an alias for `balance` command of ABC, `rw` is an alias for standard rewriting, and `rf` is an alias for standard refactoring. Also, `rwz` and `rfz` are aliases for standard rewriting and refactoring, respectively with accepting zero gains. In our scripts, `brf` stands for balanced factorization and `brw` stands for balanced rewriting. Also, `brwz` and `brfz` are `brw` and `brf`, respectively with accepting zero gains.

Figure 4 shows values of imbalance factors for graphs generated by each set of optimization scripts listed in Table 1 together with the case when no optimization is employed. On average for 15 benchmark circuits, our `blnc_syn1` and `blnc_syn2` optimization scripts reduce the imbalance factor by 1.06×, and 1.38×, respectively compared with the case of applying no optimizations, and 41%, and 63%, respectively compared with `resyn2` (the best among `resyn`, `resyn2`, `resyn2a`). Table 2 lists the graph cost function of different benchmark circuits generated by various optimization scripts. On average for 15 benchmark circuits, our `blnc_syn1` and `blnc_syn2` reduces this parameter by 68%, and 83%, respectively compared with the case of not applying any optimizations, and 19%, and 30%, respectively compared with `resyn2`.

Figure 5 shows the logical depth (critical path length) of different circuits optimized by employing different optimization scripts and mapped by ABC's cut-based technology mapper (plus splitter insertion and path balancing [8]). On average for 15 benchmark circuits, our `blnc_syn1` and `blnc_syn2` optimization scripts reduce the logical depth by 26% and 36%, respectively compared with the case of not applying any technology-independent optimizations. When it is compared with `resyn2`, the average improvements are 10% and 18%, respectively for `blnc_syn1` and `blnc_syn2`. Table 3 lists total area of these circuits. The total area includes area of gates, path

⁵This expression is mentioned in page 434 of [26] for comparing literal saving that different factorization algorithms can provide.

Figure 3. Three AIGs obtained for Boolean expression mentioned in the example 2. Nodes are 2-input AND gates, and dashed lines are inverted edges. (a) generated by applying standard rewriting, balance, and refactoring commands of ABC [20], (b) generated by applying resyn2 optimization script of ABC, and, (c) a perfectly balanced tree generated by our blnc_syn1 optimization script (see Table 1).

Figure 4. Comparing imbalance factors of original graphs (no opt.) and those generated by different optimization scripts. For better exhibition purposes, the data for *priority*, *i10*, and *voter* circuits are scaled down by a factor of 10.

balancing DFFs, and splitters. On average for 15 benchmark circuits, blnc_syn1 reduces area by 21% compared with the case when no technology-independent optimizations are employed. Also, it has almost the same average area as resyn2. blnc_syn2 reduces area by 28% when compared to not using any technology-independent optimizations, and by 4% when compared to resyn2. blnc_syn2 reduces area for *x4* circuit by 23% when it is compared to resyn2.

Figure 6 shows the post place-and-route of the ISCAS c7552 benchmark circuit which is optimized using the blnc_syn2 script. The dimensions are $8440\mu m \times 8420\mu m$, which shows around 38% less chip area compared with the case of not applying blnc_syn2 (dimensions: $10090\mu m \times 9750\mu m$). Smaller chip has usually shorter critical interconnect, hence, it results in increasing the frequency of the local clock. For this reason, the chip generated by applying our blnc_syn2 enjoys increase in the local clock frequency from 13GHz to 14GHz.

6 CONCLUSION

In this paper, balanced factorization and rewriting algorithms are presented. Unlike standard rewriting and refactoring algorithms,

Figure 5. Comparing logical depth of the mapped circuits optimized by using different optimization scripts and the original one (no opt.). For better exhibition purposes, the data for *priority*, and *IntDiv8* circuits are scaled down by a factor of 10.

our technology-independent optimization algorithms preserve balance of the given graph while optimizing it. Experimental results show that our optimization scripts reduce imbalance factor by an average of 1.38 $\times$. Using an SFQ library of gates, our optimization scripts reduce total area and logical depth after technology mapping by an average of 28%, and 36%, respectively for 15 benchmark circuits.

ACKNOWLEDGEMENT

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 U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein. This project is also supported in part by a grant from the Software and Hardware Foundations program of the National Science Foundation.

The presented place-and-route results are generated by using software tools provided by S. N. Shahsavani and T. Lin from the University of Southern California.

Table 2. Comparing the graph cost of different benchmark circuits generated by using different optimization scripts.

circuit	no opt.	resyn	resyn2	resyn2a	blnc_syn1	blnc_syn2
c499	1884	1604	1604	1604	1389	1362
c7552	6879	4275	4438	4792	3638	3039
c5315	9275	5428	5176	5609	4665	4743
c3540	4569	3131	2958	3108	2642	2560
c2670	1785	1544	1503	1545	1257	1345
i2c	4567	3596	3608	3587	3334	2932
priority	43083	37486	25772	36652	18505	15371
voter	25536	21519	18565	20350	19104	15544
cavlc	2271	1893	1882	1891	1867	1659
int2float	825	580	590	580	573	507
KSA32	1477	1437	1340	1436	1180	1156
IntDiv8	5677	5296	5295	5296	4175	3854
i10	18851	8580	8415	8489	6937	6191
x4	1555	1082	1093	1082	683	661
apex6	2069	1817	1825	1819	1405	1307

Table 3. Comparing the area (mm^2) of different benchmark circuits generated by using different optimization scripts.

circuit	no opt.	resyn	resyn2	resyn2a	blnc_syn1	blnc_syn2
c499	5.37	4.91	4.91	4.91	4.85	4.59
c7552	41.11	35.95	36.76	39.23	34.55	34.92
c5315	45.73	41.4	38.54	40.15	35.35	39.26
c3540	22.11	21.52	20.99	20.78	22.06	21.82
c2670	28.12	23.93	25.8	24.28	25.22	25.39
i2c	35.91	31.72	30.7	31.72	32.2	29.72
priority	171.22	158.11	91.41	154.73	107.38	93.06
voter	258.93	168.15	138.24	169.78	162.65	132.45
cavlc	12.15	11.79	11.93	11.75	11.85	12.09
int2float	4.95	4.6	4.55	4.6	4.64	4.34
KSA32	9.16	8.38	9.32	8.42	9.61	10.17
IntDiv8	21.24	20.38	20.42	20.38	20.95	20.13
i10	118.71	82.6	80.97	80.57	78.12	72.27
x4	11.1	8.6	8.8	8.85	7.71	7.13
apex6	20.85	19.22	19.24	19.08	19.48	17.2

REFERENCES

- [1] T. N. Theis and H. S. P. Wong. The end of moore's law: A new beginning for information technology. *Computing in Science Engineering*, 19(2):41–50, Mar 2017.
- [2] D Scott Holmes, Andrew L Ripple, and Marc A Manheimer. Energy-efficient superconducting computing-power budgets and requirements. *IEEE Transactions on Applied Superconductivity*, 23(3), 2013.
- [3] Brian David Josephson. Possible new effects in superconductive tunnelling. *Physics letters*, 1(7):251–253, 1962.
- [4] PI Bunyk, A Oliva, VK Semenov, M Bhushan, KK Likharev, JE Lukens, MB Ketchen, and WH Mallison. High-speed single-flux-quantum circuit using planarized niobium-trilayer josephson junction technology. *Applied physics letters*, 66(5):646–648, 1995.
- [5] KK Likharev and VK Semenov. Rsfq logic/memory family: A new josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity*, 50(1), 1991.
- [6] W Chen, AV Rylyakov, Vijay Patel, JE Lukens, and KK Likharev. Rapid single flux quantum t-flip flop operating up to 770 ghz. *IEEE Transactions on Applied Superconductivity*, 9(2):3212–3215, 1999.
- [7] Rudolf Gross, Achim Marx, and Frank Deppe. *Applied superconductivity: Josephson effect and superconducting electronics*. De Gruyter, 2015.
- [8] Ghasem Pasandi and Massoud Pedram. PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits. *IEEE Transactions on Applied Superconductivity*, 29(4):1–14, 2019.
- [9] Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram. SFQmap: A technology mapping tool for single flux quantum logic circuits. In *International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 27, 2018.
- [10] Naveen Katam, Alireza Shafaei, and Massoud Pedram. Design of multiple fanout clock distribution network for rapid single flux quantum technology. In *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 384–389. IEEE, 2017.
- [11] Robert K Brayton. Factoring logic functions. *IBM Journal of research and development*, 31(2):187–198, 1987.
- [12] Sasan Iman and Massoud Pedram. Logic extraction and factorization for low power. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pages 248–253. ACM, 1995.
- [13] Sumit Roy, Harm Arts, and Prithviraj Banerjee. Powershake: A low power driven clustering and factoring methodology for boolean expressions. In *Proceedings of the conference on Design, automation and test in Europe*, pages 967–968. IEEE Computer Society, 1998.
- [14] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd annual Design Automation Conference*, pages 532–535. ACM, 2006.
- [15] Per Jesse and Arne Boralv. DAG-aware circuit compression for formal verification. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 42–49. IEEE Computer Society, 2004.
- [16] Saburo Muroga. Logic design and switching theory. *John Wiley & Sons*, 1979.
- [17] Mathias Soeken and Michael Kirkedal Thomsen. White dots do matter: rewriting reversible logic circuits. In *International Conference on Reversible Computation*, pages 196–208. Springer, 2013.
- [18] Winston Haaswijk, Mathias Soeken, Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. A novel basis for logic rewriting. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 151–156. Ieee, 2017.
- [19] Jason Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):1–12, 1994.
- [20] Mishchenko et. al. ABC : A system for sequential synthesis and verification. *Berkeley Logic Synthesis and Verification Group*, 2018.
- [21] C. Fourie. Rsfq cell library, 2018.
- [22] Mark C Hansen, Hakan Yalcin, and John P Hayes. Unveiling the iscas-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.
- [23] Amarú et. al. The epfl combinational benchmark suite, 2017.
- [24] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.
- [25] Alan Mishchenko, Robert Brayton, Stephen Jang, and Victor Kravets. Delay optimization using sop balancing. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 375–382. IEEE, 2011.
- [26] Gary D Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms*. Springer Science & Business Media, 2006.

Figure 6. Post place-and-route of ISCAS c7552 benchmark circuit synthesized by applying our technology-independent optimization script, *blnc_syn2*. Dimensions are $8440\mu m \times 8420\mu m$. Dimensions of the chip without applying our optimization scripts are $10090\mu m \times 9750\mu m$.