Algebraic and Boolean Optimization Methods for AQFP Superconducting Circuits

Eleonora Testa
EPFL, Lausanne, Switzerland

Siang-Yun Lee
EPFL, Lausanne, Switzerland

Heinz Riener
EPFL, Lausanne, Switzerland

Giovanni De Micheli
EPFL, Lausanne, Switzerland

ABSTRACT

Adiabatic quantum-flux-parametron (AQFP) circuits are a family of *superconducting electronic* (SCE) circuits that have recently gained growing interest due to their low-energy consumption, and may serve as alternative technology to overcome the down-scaling limitations of CMOS. AQFP logic design differs from classic digital design because logic cells are natively abstracted by the majority function, require data and clocking in specific timing windows, and have fan-out limitations. We describe here a novel majority-based logic synthesis flow addressing AQFP technology. In particular, we present both algebraic and Boolean methods over *majority-inverter graphs* (MIGs) aiming at optimizing size and depth of logic circuits. The technology limitations and constraints of the AQFP technology (e.g., path balancing and maximum fanout) are considered during optimization. The experimental results show that our flow reduces both size and depth of MIGs, while meeting the constraint of the AQFP technology. Further, we show an improvement for both area and delay when the MIGs are mapped into the AQFP technology.

KEYWORDS

AQFP, superconducting electronics, majority logic, logic synthesis

1 INTRODUCTION

The growing interest in *superconducting electronics* (SCE) is related to the search for a scalable computing technology that can match and extend the current performances of CMOS at lower energy cost. The CMOS technology, the main workhorse in electronic systems, is showing increasingly higher fabrication costs and challenges in downscaling transistor dimensions, with marginal improvements in energy consumption at smaller technology nodes. Meanwhile, the SCE technology will be relevant to the design of fast computing systems in the twenties, to address challenging complex computational problems as, for example, those related to artificial intelligence, security, weather prediction and environmental modeling as well as bio-med-discovery and drug design. Superconducting electronics leverages computation at few degrees Kelvin (typically 4 K) where resistive effects can be neglected. IBM pioneered this technology in the 70s with the use of the *Josephson junctions* (JJs) [3]. Japanese supercomputer companies followed the lead and continued even after IBM abandoned the JJ technology route in the 80s. At that time, most companies indeed predicted and effected the downscaling of CMOS technology, thereby benefitting from the constant power density (as modeled by Dennard [9]) and thus riding Moore's law graphs until today. Note that SCE differs significantly from *quantum computing* (QC, [4]). SCE design is based on superconductive inductive loops and JJs. Logic design follows the principles (but differs in the realization) of classic Boolean logic. QC operates at 10 mK for noise reasons, and leverages superposition and entanglement provided by the quantum properties of the material. Thus, QC design is based on a different logic abstraction and on logic gates with different properties.

Various families of SCE circuits have been investigated over the years. Likharev [18] proposed *rapid single flux quantum* (RSFQ) circuits, where logic values (TRUE, FALSE) are represented by the presence or absence of single flux quantum pulses. Junctions are DC biased and when a pulse is applied to the junction, the small associated current pulse can be sufficient to drive the current level over its threshold and to generate a pulse that can be propagated through the circuit. A specific feature of RSFQ circuits is that logic gates are clocked, and that the overall circuit is pipelined [23]. The RSFQ technology evolved in many directions, e.g., *energy-efficient SFQ* (eSFQ, [22]), *reciprocal quantum logic* (RQL, [13]) and *low-voltage RSFQ* (LV-RSFQ, [30]). *Dynamic rapid single flux quantum* (DSFQ) technology is a recent asynchronous realization leveraging majority logic in the design [17]. In the last decade, research work has addressed technologies that target specifically low-energy consumption. This can be achieved by using AC power (i.e., alternating current supply), as in *adiabatic quantum-flux-parametron* (AQFP) technology. The parametron is a resonant circuit with a nonlinear reactive element [11]. In general, signal propagation in AQFP circuits requires overlapping clock signals from neighboring phases [8]. In AQFP, inductor loop pairs are used to store logic information in terms of flux quanta depending on the direction of an input current and to the magnetic coupling to other inductors. A corresponding output current represents the output of a logic gate.

A major difficulty in switching from standard CMOS to superconducting electronics is the ability of realizing complex designs, because of the limited availability of *electronic design automation* (EDA) tools. In the last two years, much progress has been achieved in EDA for superconducting electronics. Recent research on design tools have addressed the two main families of SCE, namely RSFQ [10, 14, 25] and AQFP [5, 8, 32] technologies. The common salient feature of these technologies is that the logic evaluation at each gate is triggered by a clock and/or bias. Thus, the clock

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ASPDAC '21, January 18–21, 2021, Tokyo, Japan
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7999-1/21/01.
https://doi.org/10.1145/3394885.3431606

pulses and the inputs have to be present at the logic gates in specific timeframes for the computation to realize the desired Boolean function. In early realizations, logic design was performed manually, by using padding techniques that can add delays so that logic gates receive inputs and clocks appropriately. Unfortunately, hand tuning is an error-prone operation and is not scalable. The goal of realizing large-scale SCE has prompted some recent research. A large collaborative project (in progress) is described in [10], dealing with various aspects of SCE design as discussed in [24]. Researchers at USC [14, 25] address the path balancing problem that is achieved by inserting delay registers (DFF) in combinational paths so that logic depth of each gate's input is the same. The total number of registers is later optimized by retiming. This contribution considers also to use AND and OR gates with multiple (2-5) inputs and complex cells, such as A+BC. Moreover it considers the efficient design of splitters for fanout size larger than 2. Pasandi [26] describes a technology-mapping algorithm (dubbed PBMap) based on dynamic programming that decomposes a logic function into path-balanced multi-stage logic.

Our work addresses specifically the AQFP technology, that has been investigated mainly in Japan. Also in AQFP, gate inputs have to be present in specific timing windows, due to the inherently pipelined nature of the technology. It was shown that AQFP design can be made tractable by using simple libraries, where the basic constituent is a buffer and where the parallel connection of three buffers yields a majority-3 gate [29]. Moreover, buffers can be simply altered to provide a permanent FALSE or TRUE value, thus enabling the realization of AND and OR gates respectively. A particular feature of AQFP buffers (and their combination) is the ability of inverting an input (by reversing an inductor). Thus, we can consider as logic primitives the family of majority gates with complemented or uncomplemented inputs. As a result, from this standpoint, any AQFP circuit can be abstracted by a *majority-inverter graph* (MIG, [2]). Moreover, it has been suggested that wider (say 5 or 7 input) majority gates can be used in the future. Cai [8] proposed majority logic synthesis for realizing AQFP networks. Nevertheless, the efficient realization of AQFP networks requires also solving the buffering problem, i.e., handling multiple fanouts. Splitters and splitter trees cells of various type have been used. We consider in the sequel the model based on the circuits realized at Yokohama university, where splitters consist of a buffer cell followed by a branch circuit [32]. The former is clocked, and thus its synchronous delay has to be accounted for. The latter is a combination of inductors, and it can have a fanout of 4 (even though larger fanout design have been reported) [34]. Cai [7] designed algorithms and tools to do splitter design, thus providing an automated way to handle the fanout problem in AQFP. Ayala [5] recently described an entire flow for AQFP.

The contribution of this paper is to apply both algebraic and Boolean transformations to optimize AQFP combinational circuits using majority logic. We leverage a large body of work in terms of algorithms [2, 27], and software tools developed primarily for CMOS and other technologies [6, 28]. Our algorithms go from algebraic methods for depth optimization, to Boolean resubstitution and refactoring for size improvement. In particular, we leverage the MIG model as it captures well AQFP networks, and we study algorithms that yield path-balanced circuits while respecting correct

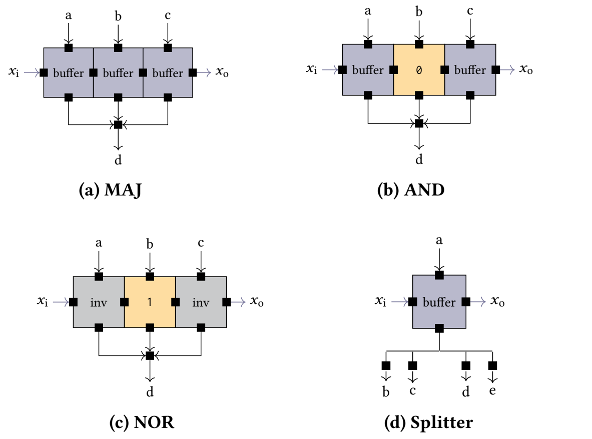


Figure 1: AQFP library: (a) MAJ cell, obtained using 3 buffers with an output stage; (b) AND gate, achieved by replacing the central buffer by one providing the 0 value; (c) NOR gate, (b) a 4-way splitter (similar pictures in [29] [8])

.

fanout buffering. Our results demonstrate an improvement in both size and depth of MIGs, while respecting the fanout limit of the nodes. We also present results of MIGs when mapped into AQFP cells. For this step, buffers and splitters insertion is considered in order to build functionally correct circuits. Our algorithms result into smaller number of buffers and splitters (15.7% on average), improved area and delay up to 22.6% and 38.5%, respectively. The work presented here focuses on algorithms and tools for SCE logic synthesis to explore capabilities and limitations of logic transformations. Broader SCE design flows [5, 15] can benefit from our techniques in their logic synthesis component. It is important to remark that some of the techniques presented here can be ported to RFSQ circuits. Nevertheless, the need of clocked inverters in RFSQ changes the delay computation and the balancing algorithms. Thus, this work focuses on AQFP technology only.

This paper is organized as follows. Section 2 presents the details on the AQFP library, used to map circuits into the AQFP technology, and the MIGs, involved as data structure for the proposed algorithms. The complete flow (depth optimization, resubstitution, and refactoring) is shown in Section 3, while Section 4 presents the experimental results. Finally, Section 5 concludes the paper.

2 BACKGROUND

We detail here (i) the *adiabatic quantum-flux parametron* (AQFP) technology library, and (ii) *majority-inverter graphs* (MIGs). The AQFP library is used in Section 4.2 to map the networks into AQFP circuits, while MIGs are used as data structure for the algorithms.

2.1 Adiabatic Quantum-Flux-Parametron

An AQFP circuit consists of an interconnection of primitive gates and a related supply and clocking scheme. The achievement of a design in AQFP is complex and requires several tasks [5]. We consider here logic synthesis of combinational networks without memory elements. We require balancing of all reconverging paths to satisfy the supply/clocking requirements. We assume the use of a semicustom library [29] consisting of majority-3 (MAJ) gates with (possibly inverted) inputs, buffers and splitters, which consist of buffers and 1-to-4 branch circuits. These cells are depicted in Fig. 1. We assume that a majority-3 gate can be constructed by combining three buffers in parallel [29] and that 2-input AND and OR gates can be achieved as degenerate majority-3 gates (i.e., by modifying a buffer). Note that in Fig. 1(a), the output d is the majority of the three inputs a, b, and c. Majority-3 gates and buffers have unit delay. Thus, splitters driving up to 4 fanout have unit delay as well, and splitter trees driving up to 16 fanouts have delay of 2. The splitter is depicted in Fig. 1(d), where input a is split into the 4 signals b, c, d, and e. We avoid computational nodes with fanout size larger than 16, as deep splitter trees influence the circuit latency adversely. We assume also that the area cost of the cells is dominated by the number of JJs, which is equal to 2 for each buffer and splitter, and thus 6 for a majority gate. Note that input inverters to a cell have zero cost, as they are achieved by an internal interchange of the terminals of an inductor. For details on AFPQ libraries, see [29, 32].

2.2 Majority-Inverter Graphs

In this work, we make use of MIGs [2] as the data structure for our optimization flow and algorithms. MIGs are defined as *directed acyclic graphs* (DAGs) where each internal node has in-degree three and represents a majority-3 gate, and each directed edge is either complemented or regular indicating the presence/absence of an inverter. The central function in MIGs is thus the *majority-of-three-inputs* (MAJ) function. The majority function of three Boolean variables x, y, and z, denoted here as $\langle xyz \rangle$, evaluates to true if and only if at least two of the three inputs are true. AND and OR logic operations are included in the MAJ, i.e., $\langle x0y \rangle = x \wedge y$ and $\langle x1y \rangle = x \vee y$, where $\wedge$ and $\vee$ are the logic AND and OR, respectively. MIGs are thus universal representation forms and can efficiently represent any Boolean function thanks to the expressiveness of the majority operator. An example of MIG is depicted in Fig. 2 (a); it has depth (number of levels) equal to 4 and size (number of nodes) of 6.

In order to manipulate MIGs and reach advantageous MIG representations, a dedicated Boolean algebra and transformation rules were described in [2]. We report here the transformation rules that are used in the rest of the paper:

$$\textbf{Associativity}-\langle xu\langle yuz\rangle\rangle = \langle zu\langle yux\rangle\rangle \quad (1)$$

$$\textbf{Distributivity}\ -\langle xy\langle uvz\rangle\rangle = \langle\langle xyu\rangle\langle xyv\rangle z\rangle \quad (2)$$

$$\textbf{Complementary Assoc.}-\langle xu\langle y\bar{u}z\rangle\rangle = \langle xu\langle yxz\rangle\rangle \quad (3)$$

$$\textbf{Relevance}\ -\langle xyz\rangle = \langle x_{y/\bar{z}}yz\rangle \quad (4)$$

where the $\bar{}$ symbol represents the complementation (inversion) of the signal, and $x_{y/\bar{z}}$ is obtained by replacing all occurrences of y with $\bar{z}$ in x. A strong property of MIGs and their algebraic framework is that by using a sequence of transformations, it is possible to traverse the entire MIG representation space [2]. In other words, given any two equivalent MIG representations, it is possible to transform one into the other by using axioms from the Boolean algebra. This result is of paramount interest to logic synthesis because it guarantees that the best MIG, for a given target metric, can be found by a sequence of transformations, even though such transformations may be hard to find. We refer the reader to paper [2] for an in-depth discussion on MIG optimization.

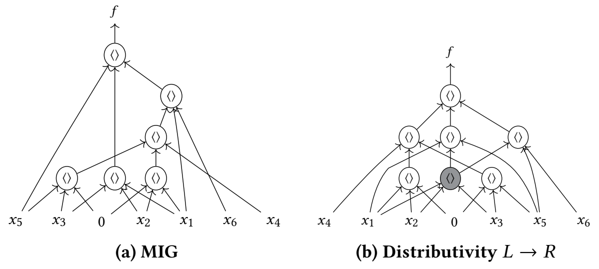


Figure 2: Example (a) of MIG when distributivity is applied. If splitters are taken into account, the depth of (b) is equal to 4 as one node (gray) has fanout size of 2.

3 LOGIC SYNTHESIS FLOW

In this section, we describe our logic synthesis flow specifically implemented to address the AQFP technology. In particular, since the AQFP technology is abstracted as majority operation, the flow works over MIGs. Furthermore, the flow consists of algebraic and Boolean methods, interleaved to address both size and depth optimization of MIGs. Most importantly, all the optimization steps are implemented in order to take into account the fanout size increase of each node. The fanout size is defined as the number of outgoing edges from each node, and as a matter of fact, a fanout size increase may result in higher depth, as more splitters may be needed. The rationale of our approach is the following. Since the logic gates have the same area cost, we leverage logic optimizations that maximize the use of majority gates. Next, as shallower networks tend to use fewer buffers, we privilege depth reduction of the MIG. Finally, as splitter insertion induces delays, we perform path equalization.

Our novel flow consists of the following steps – that will be presented in detail in the next sections:

- Create the initial MIG (Section 3.1);
- Depth optimization with algebraic rules of MIGs – limiting the fanout size increase (Section 3.2);
- Size optimization with Boolean resubstitution – considering fanout size limitations (Section 3.3);
- Refactoring – to further decrease the size (Section 3.4);
- Splitters insertion and path equalization with buffers (Section 4.2).

3.1 Create the Initial MIG

The step consists of creating an MIG from a HDL description of a combinational circuit. As we are addressing AQFP technology, the MIG requires to not have any node with fanout size larger than 16. To obtain the starting point MIG, we thus implemented (i) the method described in [12], followed by (ii) the method from [31], that limits the fanout of nodes to 16. Method (i) is a rewriting algorithm, that obtains a size-optimized MIG by implementing each k-LUT using a optimum-size MIGs retrieved from a database. To limit the fanout size of each node to a maximum of 16, we applied method

(ii) on top of the MIG obtained from (i). This method guarantees all nodes to have fanout size ≤ 16, but could result in size increase. For the benchmarks considered in Section 4, the largest fanout size in the MIG obtained from (i) is equal to 35 (for benchmarks *c432*). When limiting the fanout size with [31], the size increase is only equal to 4 (from 174 nodes to 178). On average, less than one node per circuit has fanout size > 16, resulting in a overall limited size increase for all benchmarks.

3.2 Depth Optimization

In this section we describe the second step of the optimization flow. Since we target the AQFP technology, this step is tuned to consider the fanout size (i.e., splitters) of the nodes. It is worth highlighting that for each fanout size larger than 1, a splitter needs to be inserted in the MIG for the mapping into AQFP. Splitters have delay equal to 1, thus they contribute to the final depth of the mapped-MIG.

In our flow, depth optimization is achieved by locally applying the algebraic rules from (1) (2) (3). We make use these rules to produce an efficient depth optimization heuristic for MIGs that applies local algebraic rules over the critical path to push up variables with late arrival times (i.e., in larger levels). Two changes w.r.t. to classical MIG depth-optimization need to be stressed:
(i) The depth of MIGs is affected by the splitters, i.e., fanout size larger than 1. Thus, the algorithm needs to work over an MIG that considers different depth according to the fanout size of nodes. For instance, a node with fanout size 1 has delay 1, a node with $2 \leq$ fanout size ≤ 4 has delay 2, and nodes with $5 \leq$ fanout size ≤ 16 have delay 3. Note that fanout size larger than 16 is not allowed.
(ii) The algebraic rules may not increment the fanout size of some nodes, as this would result in larger depth. As splitter trees of 1 and 2 levels are considered (up to 4 and 16 outputs respectively), a fanout size increase from 1 to 2, 4 to 5, and 16 to 17 could change the depth of the MIG, and thus, should be avoided. On the other hand, for instance, a fanout size increase from 3 to 4 would not change the depth of the MIG, but only fill the outputs of the already existing splitter. Consider as an example the distributivity rule from (2) applied from left to right ($L \rightarrow R$): The fanout size of node x is always increased by 1. Plus, if $\langle xyu \rangle$ or $\langle xyv \rangle$ are already present in the MIG, they would also have their fanout sizes increased, and so on. It means, if $fanout\_size(x)$ is equal to 1, 4 or 16 the distributivity rule is not applied, as this would increase the depth of the MIG. Consider as an example the MIG from Fig. 2. The distributivity is applied ($L \rightarrow R$) to go from (a) to (b). The depth of the MIG seems at first reduced from 4 to 3. Instead, since the fanout size of $\langle x_1x_2x_3 \rangle$ is increased from 1 to 2, the depth of the MIG in (b) would be 4, when splitters are considered. Similar considerations can be drawn for the associativity and complementary associativity rules.

Our complete flow is depicted in Alg. 1. First, the MIG is updated to consider the delay of the splitters; then, for all *primary outputs* (POs) on the critical path (i.e., that have their level equal to the depth of the MIG), the depth is reduced by applying the algebraic rules on all nodes in the *transitive fanin* (TFI) cone of the PO. We make sure that the fanout size of the largest-level child of node n is always equal to 1 – it is not needed anywhere else in the MIG. This is done not to have size increase. For instance, in the distributivity rule from (2), this would require the node $\langle uvz \rangle$ to have fanout size equal to 1. First, associativity and complementary associativity are attempted. If these are successful, the computation continues with the next node. Otherwise, distributivity is tried. Note that distributivity is tried as last option as it will always result in size increase (even when the fanout size of $children\_2$ is equal to 1). It is worth mentioning that associativity, complementary associativity, and distributivity from Alg. 1 are the modified versions of the rules that take into account the fanout size as previously described.

```
Input: MIG $M$, $allow\_area\_increase$
Output: Depth Optimized MIG $M$
1  $M \leftarrow count\_splitters\_delay(M)$;
2  $max\_depth \leftarrow depth(M)$;
3  foreach output po in M do
4      if $level(po) < max\_depth$ then continue;
5      foreach node n in topological order in TFI po do
6          $children\_2 \leftarrow$ child of $n$ with largest level;
7          if $fanout\_size(children\_2) > 1$ then continue;
8          if $try\_associativity(n)$ then continue;
9          if $try\_compl\_associativity(n)$ then continue;
10         if $allow\_area\_increase$ then
11             if $try\_distr(n)$ then continue;
12 network-cleanup-and-sweeping($M$);
```

Algorithm 1: Algebraic depth optimization of MIG

3.3 Resubstitution

Resubstitution is a method usually adopted in modern logic synthesis flows [21] to reduce the size of logic networks. It expresses the function of a node n using other nodes (called *divisors*) already present in the logic network. For size optimization, a resubstitution is accepted if the new implementation has fewer nodes than the original one. In our scenario, we always refer to "MAJ-resubstitution", highlighting the type of operator added to the network. Resubstitution is also usually classified according to the number of operators that it adds to the logic network, e.g., 0-resubstitution, if it does not add any new operator. When resubstitution adds k new nodes, size improvement is achieved if $l > k$, where l is the number of nodes in the *maximum fanout free cone* (MFFC, [20]) of node n.

Our tool minimizes the number of MAJ gates in the logic network, but, at the same time, it specifically considers the fanout of nodes. As in the previous case, state-of-the-art resubstitution algorithms need to be re-investigated to take the fanout into account. Constant inputs (TRUE and FALSE) do not take part in the cost, thus const-resubstitution can always be adopted. In case of 0-resubstitution and 1-resubstitution (majority based), the increase in the fanout of the divisors needs to be limited to avoid adding new splitters. Similar considerations as the one from Section 3.2 are adopted. Alg. 2 presents the pseudocode. First, the procedure computes for nodes in topological order, a reconvergent-driven cut and the MFFC of n as in [27]. Divisors are collected by fixing a maximum number of divisors (*max_divs*) and the maximum level (*max_depth*). In particular, divisors with a level larger than the one of node n are not considered. The procedure attempts different types of resubstitution. *const-resubstitution* substitutes a node with a constant signal; while *0-resubstitution* tries to substitute the node n with another node in

```
Input: MIG $M$, $cut\_size$, $max\_divs$
Output: Size Optimized MIG $M$
$M \leftarrow count\_splitters\_delay(M)$;
$list \leftarrow topological\_sort\_network(M)$ ;
foreach node n in list do
    $max\_depth \leftarrow level(n) - 1$;
    $cut \leftarrow find\_reconvergent\_cut(n, cut\_size)$ ;
    $mffc \leftarrow computeMFFC(n)$;
    if $|mffc| > 0$ then
        $div \leftarrow divisors(list, n, max\_div, max\_depth)$;
        $truth\_tables(cut)$;
        if $try\_const$-resub(list, n, div) then continue;
        if $try\_0$-resub(list, n, div) then continue;
        if $try\_relevance$-resub(list, n, div) then continue;
        if $try\_1$-resub(list, n, div, mffc) then continue;
network-cleanup-and-sweeping($M$);
```

Algorithm 2: Resubstitution to reduce size of MIG

the network. *Relevance-resubstitution* substitutes a majority node n using the relevance rule (4). In particular, it substitutes node n with $\langle d_0 yz \rangle$, where d_0 is one divisor meeting the relevance rule requirement. For both 0-resubstitution and relevance-resubstitution, the fanout size of the divisor d_0 should not be increased by the resubstitution. Finally, *1-resubstitution* searches for divisors d_0, d_1, d_2 to replace n using one majority $\langle d_0 d_1 d_2 \rangle$. The size of the MFFC is used for the size gain of the resubstitution transformation. In this scenario, the splitters at the fanout of all the three divisors should not be changed.

3.4 Refactoring

Refactoring is a logic synthesis technique that resynthesizes subnetworks in a logic network without using existing nodes but instead rebuilding them from scratch. Different refactoring algorithms have been proposed [1, 19], depending on the data structure and optimization goal. Since our flow works over MIGs, we implemented the technique proposed in [1]. The method from [1] is an iterative technique that builds majority gates from the truth table of a Boolean function. It is important to highlight that (i) there is no guarantee of any kind for this solution to be exact (in terms of size and depth) and (ii) the scalability of this method is poor, i.e., functions of about 6 input should be used. Nevertheless, it has been demonstrated (in our experiments) to be an effective technique to reshape the network when local minima are encountered. The refactoring procedure has been implemented following the general implementation guidelines in [27]. The MFFC of each node in the network is evaluated (with a limit on the maximum number of primary inputs) and resynthesized using the method from [1]. If the new implementation of the MFFC is better, the new MFFC is substituted to the previous one, resulting in a new MIG network. In this scenario, the cost of each MFFC implementation considers the size, enriched with the fanouts information: Each node has the cost obtained by examining the node and its splitters. For instance, nodes with a fanout size of 1 have the lower cost, while nodes with fanout size between 5 and 16 have the highest cost, as they correspond to a larger depth. Furthermore, the refactoring is not applied if it results in MIGs with larger depth or overall size. Our modified Akers heuristic refactoring resulted for instance in an improvement of 7% on top of the size improvement obtained by the resubstitution technique, for the MCNC benchmark *sqr6*. This improvement came without increasing depth, or nodes with more than 16 fanouts nodes (i.e., fanout size limited to 16).

As last step of our MIG flow, we consider splitters insertion followed by path balancing through buffers. This is an essential step as it guarantees the correct behavior of the AQFP circuits. This step will be described and detailed in Section 4.2 together with the results of AQFP-mapped circuits.

4 IMPLEMENTATION AND RESULTS

All our techniques are implemented together to create a logic synthesis flow addressing AQFP technology. In this section, we give details of our implementation and experimental results. First, we demonstrate an improvement of both size and depth of MIGs, which at the same time does not result in increased fanout size. Then, we map the MIGs into AQFP technology. For this second task, we add a last step to our flow, being the splitters and buffers insertion.

The first step (Section 3.1) is obtained by running the ABC [6][1] command `&if -a -K 4`, with k equal to 4, to obtain the 4-LUT mapping; while the database of optimum-size MIGs is implemented using exact synthesis [16]. Algebraic depth optimization, resubstitution, and refactoring are implemented using the EPFL logic synthesis library *mockturtle*[2], and all networks are tested for equivalence using ABC. Our flow run depth optimization, resubstitution, and refactoring until convergence of the results. For AQFP technology, the depth optimization is of primary importance, as this will consequently reduce the number of buffers. We thus allow area increase (set *allow_area_increase* to TRUE) in our Alg. 1. For resubstitution, the *max_divs* is set to 250 and the *cut_size* to 8. For refactoring, the size of the MFFC is set to a maximum 6 to limit the runtime complexity of the Akers algorithm.

4.1 Technology-Independent Results

In this section, we present results over MIG networks, by testing the proposed flow on 18 benchmarks from the MCNC benchmarks suite [33]. The results are listed in Table 1. The table reports the name of each benchmark, together with results for the original MIG and the optimized one. The starting point MIG (**Original MIG**) is the one obtained with the algorithms from Section 3.1, and has thus fanout size limited to maximum 16; while the **Optimized MIG** is the result of running depth optimization, resubstitution, and refactoring on top of the original MIG. For each MIG, Table 1 presents the size and the depth of the MIG. Note that both size and depth here do not take into account AQFP technology, which means splitters, fanouts, and buffers are not accounted for. The column **Max. F.** shows the maximum fanout size of the MIG. As previously mentioned, we first aim at reducing the depth of the MIG, as this correlates with the number of buffers to be inserted by the AQFP technology. Our flow reduces the depth of the MIG by 21.4% on average (column **D.Impr.%**), but carefully considers the number of splitters that would be required (as will be demonstrated

[1]Available at: *https://github.com/berkeley-abc/abc*
[2]Available at: *https://github.com/lsils/mockturtle*

Table 1: Results for size and depth optimization over MIG

	Original MIG			Optimized MIG				
Benchmark	Size	Depth	Max. F.	Size	S. Impr. %	Depth	D. Impr. %	Max. F.
5xp1	116	10	5	112	3.4	7	30.0	5
c1908	381	38	9	379	0.5	33	13.2	9
c432	178	44	16	176	1.1	30	31.8	16
c5315	1270	33	16	1257	1.0	29	12.1	15
c880	300	28	10	300	0.0	23	17.9	10
chkn	421	28	9	420	0.2	22	21.4	7
count	119	18	5	119	0.0	9	50.0	6
dist	535	16	7	514	3.9	13	18.8	8
in5	443	19	14	449	-1.4	15	21.1	15
in6	370	17	6	364	1.6	11	35.3	10
k2	1957	25	16	1934	1.2	23	8.0	16
m3	411	13	7	389	5.4	10	23.1	8
max512	713	17	9	672	5.8	14	17.6	11
misex3	1533	24	16	1515	1.2	23	4.2	16
mlp4	462	16	5	435	5.8	14	12.5	6
prom2	3484	22	16	3451	0.9	17	22.7	16
sqr6	138	13	3	126	8.7	9	30.8	4
x1dn	152	14	6	153	-0.7	12	14.3	7
Averages					2.2		21.4	

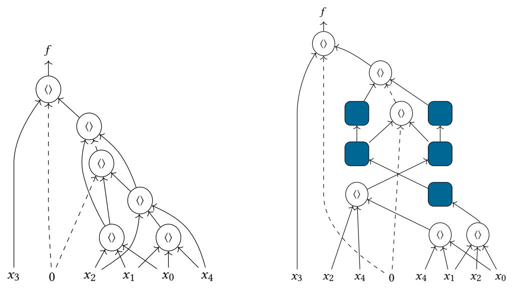
Table 2: Area and delay costs for AQFP cells.

	Area (# of JJs)	Delay (Levels of JJs)
3-input MAJ	6	1
2-input AND	6	1
2-input OR	6	1
1:4 Splitter	2	1
Buffer	2	1

for the mapped results in Section 4.2). It also optimizes the size of the benchmarks up to 8.7% (*sqr6* - column **S.Impr.%**). Consider as an example the *sqr6* benchmark: The depth is reduced by 30.8% (with a size improvement of 8.7%), but the maximum fanout size in the MIG is only increased from 3 to 4.

4.2 AQFP-Mapped Results

In this section, we present the results when the MIGs are mapped into AQFP circuits. For this purpose, we use the library cells presented in Fig. 1, and evaluate the area and delay in terms of JJs. Our library consists of MAJ, AND, and OR logic cells, plus splitters and buffers, while inverters are free of cost. The costs in terms of JJs and JJ levels of each cell are summarized in Table 2. In our mapping, splitters are inserted for all nodes with a fanout size larger than 1. In particular, splitters with one input that splits into 4 outputs are considered (compare to Fig. 1(b)). As we limit the maximum fanout size to 16, it follows that we need at most 5 splitters (in 2 levels) for each node. On the other hand, buffers are needed to balance the paths of the circuit. Note that, since splitters may affect the level of nodes and the depth, the buffers insertion is done after all splitters have been inserted. The buffers insertion works as follows: for each node n, buffers are inserted in each level between n and its highest fanout node. Moreover, for each PO, buffer insertion is done until its level matches the one of the critical path (i.e., depth) of the MIG. For a node with fanout size larger than 1, buffers in the same level can be shared; sharing of buffers is considered up to 4 fanouts at each level. Fig. 3 shows an example of MIG after



(a) MIG **(b) Buffers and splitters**

Figure 3: Buffers and splitters (blue) are inserted in the MIG from (a) - note that dashed lines represent inversions. In (b), splitters are inserted for each fanout size larger than 1, while buffers are needed to balance the paths.

buffers and splitters are inserted. Fig. 3(a) illustrates the MIG of the majority-of-five-inputs function, while Fig. 3(b) shows the MIG with two splitters for nodes with fanout size larger than 1, and buffers to balance the inputs of nodes. Note that when mapped into AQFP, this circuit corresponds to an area of 46 JJs and a delay of 6. In our model and experiments, we do not consider storage, I/O conversion to other type of logic and I/O drivers, because technology and circuit solutions are still under development. As a consequence of the lack of standard models in the literature, we do not attempt numerical comparisons of results to work of others.

The results are depicted in Table 3. As in the previous set of experiments, the starting point MIG (**Original MIG**) is the one obtained with the step from Section 3.1, while the **Optimized MIG** is the optimized one (i.e., when depth optimization, resubstitution, and refactoring are applied). For the mapped MIGs, we compare the number of buffers and splitters (column **B&S**), the area, and the delay. The last two are computed in terms of JJs and shown as columns **Area (# JJs)** and **Delay (JJs Levels)**. The number of buffers and splitters is reduced by 15.7% on average. For the *in5* and *x1dn* benchmarks, even though the MIGs size (number of nodes) from Table 1 was not improved, the mapped MIG results into improved area. Overall, the area of AQFP circuits is improved up to 22.6%, and the delay is improved by 20.9% on average.

5 CONCLUSION

We presented a novel logic synthesis flow addressing *adiabatic quantum-flux-parametron* (AQFP) superconducting circuits. We implemented a variety of algorithms ranging from algebraic depth optimization to Boolean resubstitution and refactoring. As AQFP technology efficiently implements the majority-of-three-inputs function, we leveraged the properties of *majority-inverter graphs* (MIGs) to abstract and optimize combinational circuits. Moreover, our algorithms specifically consider features of the AQFP technology such as splitters (which depend on the fanout size of each node) and buffers. Our results demonstrated both size and depth improvement of MIGs, within the limit of the constraints set by the AQFP circuits. In particular, we obtained an average improvement of 7.4% in area and 20.9% in delay when mapping into AQFP circuits. We believe

Table 3: Results for area, delay, and number of buffers & splitters for MIGs mapped into AQFP technology

	Original MIG			Optimized MIG					
Benchmark	B. & S.	Area (# JJs)	Delay (JJs Levels)	B. & S.	Impr. B. & S. %	Area (# JJs)	Area Impr. %	Delay (JJs Levels)	Delay Impr. %
5xp1	139	974	13	76	45.3	824	15.4	8	38.5
c1908	1683	5652	59	1481	12.0	5236	7.4	53	10.2
c432	780	2628	64	732	6.2	2520	4.1	50	21.9
c5315	6109	19838	53	5674	7.1	18890	4.8	49	7.5
c880	1518	4836	42	1354	10.8	4508	6.8	36	14.3
chkn	831	4188	33	751	9.6	4022	4.0	28	15.2
count	464	1642	27	356	23.3	1426	13.2	18	33.3
dist	680	4570	23	562	17.4	4208	7.9	17	26.1
in5	907	4472	26	821	9.5	4336	3.0	20	23.1
in6	694	3608	20	649	6.5	3482	3.5	17	15.0
k2	3646	19034	36	3347	8.2	18298	3.9	29	19.4
m3	516	3498	18	387	25.0	3108	11.1	13	27.8
max512	879	6036	23	752	14.4	5536	8.3	19	17.4
misex3	3108	15414	33	2946	5.2	14982	2.8	29	12.1
mlp4	559	3890	22	506	9.5	3622	6.9	19	13.6
prom2	4881	30666	27	4008	17.9	28722	6.3	21	22.2
sqr6	241	1310	16	129	46.5	1014	22.6	10	37.5
x1dn	206	1324	19	189	8.3	1296	2.1	15	21.1
Averages					15.7		7.4		20.9

that the MIG-based transformations, supported by a formal model, will be key for the optimization of AQFP and other SCE circuits.

ACKNOWLEDGMENTS

This research was supported by the SNF grant "Supercool: Design methods and tools for superconducting electronics" - 200021_1920981, and by the EPFL Open Science Fund. We also acknowledge the support from Synopsys Inc.

REFERENCES

[1] Sheldon B. Akers. 1962. Synthesis of combinational logic using three-input majority gates. In *Annual Symp. on Switching Circuit Theory and Logical Design.* 149–158.

[2] Luca G. Amarù et al. 2016. Majority-inverter graph: A new paradigm for logic optimization. *IEEE Trans. on Comp. Aided-Design of Int. Circuits and Systems* 35, 5 (2016), 806–819.

[3] Wilhelm Anacker. 1980. Josephson computer technology: An IBM research project. *IBM Journal of research and development* 24, 2 (1980), 107–112.

[4] Frank Arute et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[5] Christopher L. Ayala et al. 2020. A semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors. *Superconductor Science and Technology* 33, 5 (2020), 054006.

[6] Robert K. Brayton et al. 2010. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification.* 24–40.

[7] Ruizhe Cai et al. 2019. A buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits. In *ICCD.* 429–436.

[8] Ruizhe Cai et al. 2019. A majority logic synthesis framework for adiabatic quantum-flux-parametron superconducting circuits. In *GLSVLSI.* 189–194.

[9] Robert H. Dennard et al. 2007. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Solid-State Circuits Society Newsletter* 12, 1 (2007), 38–50.

[10] Coenrad J. Fourie et al. 2019. ColdFlux superconducting EDA and TCAD tools project: Overview and progress. *IEEE Trans. Appl. Supercond.* 29, 5 (2019), 1–7.

[11] Eiichi Goto. 1959. The parametron, a digital computing element which utilizes parametric oscillation. *Proceedings of the IRE* 47, 8 (1959), 1304–1316.

[12] Winston Haaswijk et al. 2017. A novel basis for logic optimization. In *ASPDAC.*

[13] Quentin P. Herr et al. 2011. Ultra-low-power superconductor logic. *Journal of applied physics* 109, 10 (2011), 103903.

[14] Naveen K. Katam et al. 2018. Logic optimization, complex cell design, and retiming of single flux quantum circuits. *IEEE Trans. Appl. Supercond.* 28, 7 (2018), 1–9.

[15] Jamil Kawa. 2020. The challenges of automating the design flow of superconducting electronic circuits. *Presentation at IWLS* (2020).

[16] Arist Kojevnikov et al. 2009. Finding efficient circuits using SAT-solvers. In *Int'l Conf. on Theory and Applications of Satisfiability Testing.* 32–44.

[17] Gleb Krylov et al. 2020. Asynchronous dynamic single flux quantum majority gates. *IEEE Trans. Appl. Supercond.* (2020).

[18] Konstantin K. Likharev et al. 1991. RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Trans. Appl. Supercond.* 1, 1 (1991), 3–28.

[19] Alan Mishchenko et al. 2001. An algorithm for bi-decomposition of logic functions. In *DAC.* 103–108.

[20] Alan Mishchenko et al. 2006. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *DAC.* 532–535.

[21] Alan Mishchenko et al. 2006. Scalable logic synthesis using a simple circuit structure. In *IWLS.* 15–22.

[22] Oleg A. Mukhanov. 2011. Energy-efficient single flux quantum technology. *IEEE Trans. Appl. Supercond.* 21, 3 (2011), 760–769.

[23] Oleg A. Mukhanov et al. 1995. Design and operation of RSFQ circuits for digital signal processing. In *Supercond. Electron. Conf.* 27–30.

[24] Louis C. Muller. 2015. *RSFQ digital circuit design automation and optimisation.* Ph.D. Dissertation. Doctoral Thesis dissertation, Stellenbosch University.

[25] Ghasem Pasandi et al. 2018. SFQmap: A technology mapping tool for single flux quantum logic circuits. In *ISCAS.* 1–5.

[26] Ghasem Pasandi et al. 2019. A dynamic programming-based path balancing technology mapping algorithm targeting area minimization. In *ICCAD.*

[27] Heinz Riener et al. 2019. Scalable generic logic synthesis: One approach to rule them all. In *DAC.*

[28] Mathias Soeken et al. 2019. The EPFL logic synthesis libraries. arXiv:1805.05121v2.

[29] Naoki Takeuchi et al. 2015. Adiabatic quantum-flux-parametron cell library adopting minimalist design. *Journal of Applied Physics* 117, 17 (2015), 173912.

[30] Masamitsu Tanaka et al. 2013. Low-energy consumption RSFQ circuits driven by low voltages. *IEEE Trans. Appl. Supercond.* 23, 3 (2013), 1701104–1701104.

[31] Eleonora Testa et al. 2017. Inverter propagation and fan-out constraints for beyond-CMOS majority-based technologies. In *Annual Symp. on VLSI.* 164–169.

[32] Qiuyun Xu et al. 2017. Synthesis flow for cell-based adiabatic quantum-flux-parametron structural circuit generation with HDL back-end verification. *IEEE Trans. Appl. Supercond.* 27, 4 (2017), 1–5.

[33] Saeyang Yang. 1991. *Logic synthesis and optimization benchmarks user guide: version 3.0.* Microelectronics Center of North Carolina (MCNC).

[34] Nobuyuki Yoshikawa et al. 2017. Recent research developments of adiabatic quantum-flux-parametron circuitstechnology toward energy-efficient high-performance computing.