Depth-optimal Buffer and Splitter Insertion and Optimization in QFP Circuits

lessandro Tempia Calvino, Giovanni De Micheli
Integrated Systems Laboratory, EPFL, Lausanne, Switzerland

***bstract*—The diabatic Quantum-Flux Parametron (QFP) is an energy-efficient superconducting logic family. QFP technology requires buffer and splitting elements (B/S) to be inserted to satisfy path-balancing and fanout-branching constraints. B/S insertion policies and optimization strategies have been recently proposed to minimize the number of buffers and splitters needed in an QFP circuit. In this work, we study the B/S insertion and optimization methods. In particular, the paper proposes: i) two initial B/S insertion policies based on L P and S P scheduling that guarantee global depth optimality; ii) a new approach for B/S optimization based on minimum register retiming; iii) a B/S optimization flow based on (i), (ii), and existing work. We show that our approach reduces the number of B/S up to 2 % while guaranteeing optimal depth and providing a $55\times$ speed-up compared to the state-of-the-art.**

I. Introduction

In recent years, superconducting electronics (SCE) gained increasing interest proposing high-speed and power-efficient solutions. Superconducting circuits are based on *Josephson junctions* (JJs) and operate at a few degrees Kelvin (typically 4K) where resistive effects are negligible. The switching speed of Josephson junctions supports the realization of circuits clocked at several tens of Gigahertz and a considerably lower power consumption compared to CMOS. The potential of SCE is well supported by academic and industrial projects which address the electronic design automation (ED) challenges of digital SCE design [1]–[3].

The *adiabatic quantum-flux parametron* (QFP) is a superconducting logic family that targets low-energy consumption. In this technology, adiabatic switching operations drastically reduce the dynamic and static power consumption compared to other superconducting logic families [4]. QFP circuits operate at frequencies up 10 Gigahertz with a power dissipation of two orders of magnitude lower compared to CMOS when accounting also for the cryo-cooling energy [3].

In QFP circuits, due to a different encoding of the information compared to CMOS, each logic gate needs an alternating excitation current that periodically releases the computation. The excitation current is delivered as a clock [5]. Thus, data at each gate must be present at specific time frames for correct functionality. This may require the insertion of clocked *buffers* such that all data paths at each gate's fanin have the same length. This design constraint is called *path-balancing*. Logic gates also have limited driving capabilities. Branching elements called *splitters* are necessary for multiple fanouts. Splitters need a clock to operate and typically support up to 4 fanouts. This second design constraint is called *fanout-branching*.

The path-balancing and fanout-branching requirements complicate the design process and significantly affect area and delay. In some applications, buffers and splitters (B/S) arrived to occupy half of the total area even after optimization [6]–[10]. Hence, developing ED tools able to minimize the number of buffers and splitters is of primary importance. Existing work considered QFP constraints during logic optimization to reduce imbalances and high-fanouts by modifying the logic [8], [9], [11]. Other previous work developed techniques to insert and minimize the number of buffers and splitters needed in an QFP circuit after logic synthesis [7], [10], [12].

In this paper, we tackle the B/S insertion problem for area and delay minimization given a logic network. Similarly to [10], we formulate the B/S insertion problem as a *scheduling* problem. First, we present two linear-time algorithms to insert B/S elements such that the resulting QFP circuit is delay-optimal. Minimizing the delay is beneficial for the area since it reduces the number of buffers needed for path-balancing. None of the previous state-of-the-art approaches guarantee global delay optimality [7], [10], [12]. Then, we present a novel B/S optimization method based on minimum-register retiming [13] to reduce the number of buffers and splitters in an QFP circuit. In previous work, minimum-register retiming has been applied to the *rapid single-flux quantum* (RSFQ) superconducting family to reduce the number of buffers [14]. However, in RSFQ, splitters are not clocked and are not involved in path-balancing. Consequently, the problem is much harder in QFP logic and retiming is used to drive a heuristic algorithm. Finally, we propose an QFP B/S insertion and optimization flow based on depth-optimal B/S insertion, retiming, and the chunk movement algorithm presented in [10].

In the experiments, we evaluate our optimization approach based on retiming showing that it reduces the number of buffers and splitters of previously optimized circuits from [10] up to 17%. Then, we show that our approach reduces the number of buffers and splitters up to 20% compared to the state-of-the-art algorithm [12] while providing a $55\times$ speed-up. Finally, we show that our approach scales up to big benchmarks.

II. B ckground

. diabatic Quantum-Flux Parametron

The *adiabatic quantum-flux parametron* (QFP) is an energy-efficient superconducting technology. The main elements of QFP circuits are the buffer cell and the branch cell. The buffer cell is realized using *Josephson junctions* (JJs) and superconductive inductors to form a two-junction SQUID [4]. The basic functional block is the *majority-of-3* gate (M J3) that can be realized using three buffers and a 3-to-1 branch cell at their outputs. By modifying a buffer cell of the M J3 to produce a constant *zero* or *one* output, the M J3 implements the ND2 or the OR2 cell. Inverters can be implemented in a buffer without additional cost by using a negative mutual inductance instead of a positive one [15]. Since the QFP logic is inherently majority-based, the functionality of an QFP circuit can be represented by a *majority-inverter graph* (MIG) [16].

QFP gates need an C excitation current to operate. This signal also serves as a clock used to synchronize the computation at the output of the gates. Consequently, data at the inputs of a gate must be available in the same clock cycle to perform the correct operation. Hence, fast signals need to be delayed by inserting clocked buffers as delay elements. This problem is called *path-balancing*. QFP circuits also have limited driving capabilities. Branching elements composed of a buffer cell and a branch cell are necessary whenever a gate needs to drive more than one output. These elements are called *splitters*. This constraint is called *fanout-branching*. Typically, splitters have a maximum driving capacity of 3 or 4 and are clocked cells. Hence, splitters affect path-balancing.

Different technology assumptions have been proposed to approach the path-balancing and fanout-branching problems [10]. In particular, these assumptions consider the cases in which *primary inputs* (PIs) and *primary outputs* (POs) need to be branched and balanced or not. In this paper, we assume that PIs and POs need to be balanced and branched. Nevertheless, our work is easily adaptable to support all the other assumptions.

The area and delay in an QFP technology are commonly evaluated in terms of Josephson junctions (JJs). The area cost of a buffer or splitter cell is 2 JJs, while the area cost of a M J3, ND, and OR gates is 6 JJs. Each cell has a JJ depth of one. Hence, we describe the delay of an QFP circuit in terms of JJ depth.

B. Notation

Boolean network N is modeled as a direct acyclic graph defined by the pair $V, E)$ where V represents the set of nodes and E represents the set of directed edges. In this paper, we use Boolean network and circuit interchangeably.

For any node v, the *fanins* of v, denoted as $FI\ v)$, is a set of nodes driving node v, i.e., nodes that have an outgoing edge towards v. Similarly, the *fanouts* of v, denoted as $FO\ v)$, is a set of nodes which are driven by node v, i.e., nodes that have an incoming edge from v. The set of primary inputs (PIs) I is a subset of nodes without fanin in the network. The set of primary outputs (POs) O is a subset of nodes without fanout in the network. The set of logic gates G is a subset of nodes from a predefined gate library. Each node in V is either in I, O, or G. In this paper, each gate in an QFP-compatible network is either an ND2, OR2, or M J3 with optional input negations.

mapped network M representing an QFP circuit extends a Boolean network with a gate element *splitter* and a gate element *buffer*. splitter is a gate with a fanin size of 1 and fanout size of s_b where s_b is the splitting capacity. buffer is a special case of splitter with a fanout size of 1. *splitter tree* of a node v, denoted by $ST\ v)$, is a set of splitters and buffers reachable from the fanout of v such that every path from v to a B/S element $s \in ST\ v)$ traverses only buffers and splitters. splitter tree is said to be *irredundant* if each B/S element has at least one fanout and there is not a pair (s_1, s_2) such that their incoming edges are connected to the same node and they both have fanout size smaller than s_b [10].

schedule of a network is a function $d : V \rightarrow\ _0$ that assigns each node in the network to a non-negative integer level. The depth of a network $d\ N)$ is defined as $d\ N) = \max_{o\ O} d\ o)$. schedule of the network is *valid* if and only if a mapping function $f :\ N, d) \rightarrow M$ exists such that buffers and splitters can be inserted respecting the path-balancing and fanout-branching constraints.

C. Minimum Register Retiming

Minimum register retiming is the problem of relocating the registers in a circuit in order to minimize their number while preserving the functionality [13]. The repositioning is captured by the integer-valued *retiming lag* function $r\ v) : V \rightarrow \mathbb{Z}$ that describes the number of registers moved backward over node v, from its fanout to the fanin. Given a circuit where $w\ e)$ is the initial number of registers on an edge e, the minimum register retiming problem can be formulated as a linear problem as follows:

$$\min \sum_{\forall e = u\ v)\ E} \quad r\ v) \quad r\ u) \qquad s.t. \tag{1}$$

$$r\ u) \quad r\ v) \leq w\ e) \qquad \forall e =\ u, v) \tag{2}$$

This linear problem is dual to the minimum-cost flow problem [13] for which many algorithms exist. For instance, using the Goldberg and Tarjan algorithm [17] the problem is solvable in $O\ |V||E| \log\ |V|) \log\ |V|^2/|E|)$ time.

III. Depth-Optim l Buffer nd Splitter Insertion

In this section, we present a depth-optimal B/S insertion policy formulating the QFP mapping problem as a scheduling problem similarly to [10]. Then, we propose two depth-optimal initial scheduling assignment using the *as-late-as-possible* (L P) and *as-soon-as-possible* (S P) scheduling. These two schedules can be used as a starting point to carry the B/S size optimization.

The depth-optimal B/S insertion problem for a Boolean network N consists of finding splitter tree configurations such that the resulting circuit depth is minimal. Finding an optimal-depth QFP circuit is not only beneficial for the delay but also for the area. The area of an QFP circuit is related to its depth because of path-balancing. Intuitively, longer critical

lgorithm 1: L P node scheduling

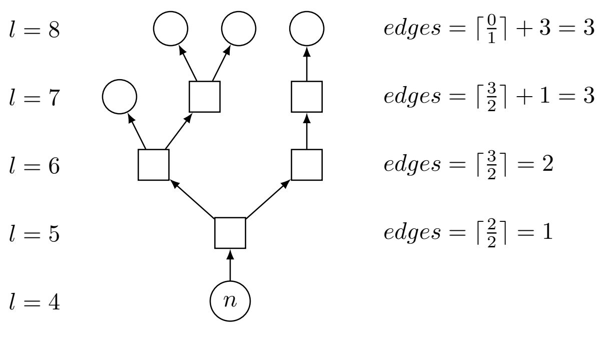
```
1  **Input** : node $n$, partial scheduling $d$
2  **Output**: level of node $n$ in the schedule
3  $L \leftarrow \{ \ v, d\ v)) \mid v \in FO\ n) \};$
4  $l_{last} \leftarrow \max\limits_{v\ l) \in L} l;$
5  $edges \leftarrow 0;$
6  **foreach** $v,\ l) \in L$ *in descending order of* $l$ **do**
7      $splitters \leftarrow \lceil \frac{edges}{s_b^{l_l\ st\ \ l}} \rceil;$
8      $edges \leftarrow splitters + 1;$
9      $l_{last} \leftarrow l;$
10 **end**
11 $l_{last} \leftarrow l_{last} \quad 1;$
12 **while** $edges \neq 1$ **do**
13     $edges \leftarrow \lceil \frac{edges}{s_b} \rceil;$
14     $l_{last} \leftarrow l_{last} \quad 1;$
15 **end**
16 **return** $l_{last}$;
```

paths (considering B/S elements) would potentially require to insert more buffers due to longer paths to balance.

Claim 1. *Given a Boolean network, the optimal depth of an QFP circuit and a valid scheduling assignment that satisfies the requirements of path-balancing and fanout-branching can be found using an as-late-as-possible (L P) scheduling.*

To verify this claim, we first define our B/S insertion policy for a single node n given a partial schedule d. Lee et al. [10] presented an irredundant algorithm for B/S insertion on a single node given the relative depths of the fanout. Their approach applied to a pre-scheduled network finds an QFP circuit in linear time. In this paper, we modify that algorithm to find the minimum-height irredundant splitter tree of a node given a level assignment of its fanout. The algorithm is shown in lgorithm 1.

lgorithm 1 fits a minimum height splitter tree for a node n given the level assignment of its fanout in a partial schedule d and returns a level assignment for node n. First, the pairs composed of nodes and scheduled levels in the fanout of n are saved in L (line 3). Then, the highest level in L is saved in l_{last} (line 4). In the main loop (lines 6 to 10), for each node-level pair in L in descending order, *edges* is updated. Variable $edges$ tracks the number of nodes including B/S elements needed to



$l = 8$
$edges = \lceil \frac{0}{1} \rceil + 3 = 3$
$l = 7$
$edges = \lceil \frac{3}{2} \rceil + 1 = 3$
$l = 6$
$edges = \lceil \frac{3}{2} \rceil = 2$
$l = 5$
$edges = \lceil \frac{2}{2} \rceil = 1$
$l = 4$
n

Fig. 1: Example to illustrate lgorithm 1

be connected at level l (line 8). Every time l_{last} is decreased, the number of buffers and splitters needed to connect to higher levels nodes is computed (line 7). Once every fanout of n has been processed, the algorithm finds the highest level where n can be scheduled (lines 12 to 15). This level corresponds to the first position where *edges* is equal to one. Figure 1 illustrates an example for a node n with four fanout nodes to better understand lgorithm 1. Figure 1 also shows the generated minimum-height splitter tree. Logic gates are represented by circles while B/S elements by squares. The splitting capacity is $s_b = 2$. The variable *edges* keeps track of the number of nodes and B/S elements for each level. Gate n is inserted as soon as $edges = 1$. It follows that:

Lemma 2. *lgorithm 1 finds an irredundant minimum-height splitter tree for a node given a partial schedule.*

Proof. lgorithm 1 inserts the minimum number of B/S elements per level. Hence, it also generates a minimum-height splitter tree.

Since algorithm 1 finds a level assignment that satisfies the path-balancing and fanout-branching constraints for a node based on its fanout, it can be used as a scheduling function in an L P schedule. n L P scheduling algorithm schedules all the POs of a network to a bound and applies a scheduling function for each node in reverse topological order.

Proposition 3. *Let be a sufficiently large bound to obtain a legal L P schedule. Let* $l_{min} = \min_{i \quad I} d\ i)$ *be the lowest level of a scheduled node in the network. Then, the depth of the QFP circuit is* $\quad l_{min}$.

Proof. By definition, the L P scheduling bound is relaxed such that $\geq d\ N)$. Since there are not scheduled nodes from level zero to $l_{min} \quad 1$, we could reduce by l_{min} to removed unassigned levels.

From Proposition 3, it follows that the scheduling objective for minimizing the depth is to maximize the level of PIs in the L P schedule since $d\ N) = \quad l_{min}$ where is constant.

Lemma 4. *The global optimal-depth buffer and splitter insertion problem is solvable using dynamic programming by inserting minimum-height splitter trees in reverse topological order.*

Proof. n L P scheduling algorithm assigns all the POs to the bound that is the maximum possible level. Given a node n and its already scheduled fanout, a minimum-height splitter tree can be inserted using lgorithm 1. Thus, n is scheduled to its maximum level. By induction, the algorithm maximizes the level of each node and thus the resulting QFP circuit is depth-optimal.

Using Lemma 2, Proposition 3, and Lemma 4, we verified Claim 1.

lgorithm 2 shows an algorithm to construct the depth-optimal L P schedule. We assume that each PI and PO needs balancing and branching. The algorithm first computes a

lgorithm 2: Depth-optimal L P scheduling

```
1  Input  : network $N =\ I \cup G \cup O, E)$
2  Output: schedule $d$
3    $\leftarrow$ depth($N$) $\times\ 1 + \lceil \log\ FO_{max})/\log\ s_b)\rceil)$;
4  $d \leftarrow \emptyset$;
5  foreach  $o \in O$ do
6      $d\ o) \leftarrow\ $ ;
7  end
8  $l_{min} \leftarrow \infty$;
9  foreach $n \in I \cup G$ in reverse topological order do
10     $d\ n) \leftarrow$ schedule_node($n$, $d$);
11     $l_{min} \leftarrow$ min($l_{min}$, $d\ n)$);
12 end
13 foreach $i \in I$ do
14     $d\ i) \leftarrow 0$;
15 end
16 foreach $n \in G \cup O$ do
17     $d\ n) \leftarrow d\ n)\ \ l_{min}$;
18 end
19 return $d$;
```

higher bound   for the L P scheduling (line 3). This bound is computed assuming that every node in the critical path needs a splitter tree whose height is the one of a balanced tree of the highest fanout in the network. lternatively, any number sufficiently high can be used for this assignment. Then, every PO is scheduled to the bound (lines 5 to 7). The algorithm continues by scheduling in reverse topological order each node in the network using lgorithm 1 (line 10). The reverse topological ordering guarantees that each fanout has been already visited. Hence, an irredundant minimum-height splitter tree is inserted. Meanwhile, the minimum assigned level is saved (line 11). Finally, the PIs are scheduled to level zero (lines 13 to 15) and the rest of the nodes are lowered by the minimum assigned level l_{min} (lines 16 to 18).

The depth-optimal L P scheduling algorithm runs in linear time to the number of nodes in the network with a complexity of $O\ |V|FO_{max} \log\ FO_{max}))$ where FO_{max} is the maximum fanout size in the network. The L P scheduling in lgorithm 2 can be used as an initial scheduling assignment to generate an QFP circuit. The resulting QFP circuit is depth-optimal.

While we mainly focused on delay, another optimization objective is area. Minimizing area consists of maximizing the sharing of B/S elements. The initial B/S configuration could affect the area results even after optimization. Moreover, different technology assumptions [10] amplify this difference, e.g., an S P configuration is generally beneficial if POs do not need balancing. Hence, we present another depth-optimal alternative that may have a better size for some circuits or assumptions. This method is based on the S P scheduling algorithm. Both S P and L P can be used as a starting point to carry the buffer and splitter optimizations described in Section IV and V.

 n S P scheduling algorithm schedules each node in topological order. We use the previously defined L P schedule in our S P method to compute a mobility number for each node which is captured by $\mu : V \rightarrow\ _0$. The mobility is used by S P to lower the nodes towards the PIs. The algorithm works as follows. First, it initializes the mobility μ to infinite for each node. Then, it assigns PIs to level zero. Finally, it schedules each gate using lgorithm 3.

lgorithm 3: S P node scheduling

```
1  Input  : node $n$, mobility $\mu$, scheduling $d$
2  $d\ n) \leftarrow d\ n)\ \ \mu\ n)$;
3  $L \leftarrow \{\ v, d\ v)) \mid v \in FO\ n)\ \}$;
4  $T \leftarrow$ empty map;
5  foreach  $v \in FO\ n)$ do
6      $T[v] \leftarrow 0$;
7  end
8  $l_{last} \leftarrow \max_{v\ l) \in L} l$;
9  $edges \leftarrow 0$;
10 foreach  $v,\ l) \in L$ in descending order of $l$ do
11     $mobility \leftarrow 0$;
12     for  $j \leftarrow 1$ to $l_{last}\ \ l$ do
13         if  $edges = 1$ then
14             $mobility \leftarrow mobility + 1$;
15         end
16         $edges \leftarrow \lceil \frac{edges}{s_b} \rceil$;
17     end
18     foreach  $v\ ,\ l\ ) \in L$ such that $l\ \ > l$ do
19         $T[v\ ] \leftarrow T[v\ ] + mobility$;
20     end
21     $l_{last} \leftarrow l$;
22     $edges \leftarrow edges + 1$;
23 end
24 $mobility \leftarrow 0$;
25 for  $j \leftarrow d\ n)$ to $l_{last}\ \ 2$ do
26     if  $edges = 1$ then
27         $mobility \leftarrow mobility + 1$;
28     end
29     $edges \leftarrow \lceil \frac{edges}{s_b} \rceil$;
30 end
31 foreach $v \in FO\ n)$ do
32     $\mu[v] \leftarrow \min\ \mu[v], T[v] + mobility)$;
33 end
34 return;
```

 lgorithm 3 computes a lower bound on the mobility of each node in the fanout of a node n. First, node n is scheduled to a new position by decreasing the L P level by the mobility (line 2). Then, the fanout levels are saved in L (line 3). Next, a map T stores the temporary mobility of each fanout of n. Mobilities are initialized to zero (lines 4 to 7). The main loop (lines 10 to 23) is similar to the one in lgorithm 1. Line 7 in lgorithm 1 is unrolled to compute the local mobility (lines 12 to 17). The local mobility is the number of buffers needed to balance the splitter tree from level l_{last} to l. The local mobility is added to all the previously visited nodes in the loop (lines 18 to 20). The same reasoning applies from line 25 to 30. Finally, μ is updated using the mobility of each node in the current splitter tree of n. Nodes are scheduled according to the worst mobility to guarantee a valid schedule. The mobility computation is conservative but works well in practice.

In this section, we presented the L P and S P algorithms assuming that PIs and POs need to be balanced and branched. Nevertheless, these algorithms can be easily modified to support other assumptions without higher complexity costs.

IV. Retiming-B sed Buffer nd Splitter Optimiz tion

In this section, we present an algorithm based on minimum register retiming to globally minimize the numbers of buffers and splitters in an QFP network. Previous work applied a retiming-like optimization to QFP logic [6], [7]. However, their approach does not perform retiming but moves buffers locally from the output of splitters to the input. This optimization is included in our depth-optimal scheduling algorithms since our scheduling approach creates irredundant splitter trees.

Buffers and splitters are used in QFP circuits to meet the circuit constraints of path-balancing and fanout-branching. Minimizing the number of B/S elements consists of maximizing the sharing of B/S elements. Without accounting for fanout-branching, e.g., assuming that buffers have an infinite driving capability, the minimum number of buffers is achievable in polynomial time using a minimum register retiming algorithm considering each buffer as a register. Retiming preserves the path-balancing constraint since each path traverses the same number of registers before and after retiming. Existing work applied this idea to *Rapid Single-Flux Quantum* (RSFQ) superconducting logic [14]. When considering fanout limitations, splitters cannot be relocated freely since their movement is conditional on preserving the fanout constraints. Hence, retiming can be only used as a heuristic for B/S optimization. Figure 2 shows a splitter tree where a gate is represented by a circle and a B/S element is represented by a rectangle. Let us suppose that the maximum splitting capacity is $s_b = 3$. In this example, splitter s_0 cannot be selected for retiming since its movement would increase the fanout of g_0 to 2 not satisfying the fanout constraint of a gate. Splitters s_1 and s_2 are only mutually selectable for retiming since the movement of both of them would increase the fanout of s_0 to 4 not satisfying the fanout constraint s_b. Note that the latter case may happen only in redundant splitter trees, as in Figure 2. Moreover, retiming optimization of the splitters s_1 and s_2 may depend on different fanout grouping such as $FO\ s_1) = \{f_0, f_2\}$, $FO\ s_2) = \{f_1, f_3\}$ instead of the current $FO\ s_1) = \{f_0, f_1\}$, $FO\ s_2) = \{f_2, f_3\}$. Some groupings may unlock a B/S minimization that is not achievable in others.

The B/S retiming algorithm is shown in lgorithm 4. The algorithm receives a valid QFP circuit, the technologies assumptions, and a direction as input. Since the retiming problem is solved as a maximum flow problem similarly to [18], the flow computation is separated in the forward and backward directions. The input *dir* defines which direction to execute first. Generally, forward is the preferred first direction if the circuit has an L P configuration since most of the registers would be placed closer to the PIs. Backward is instead a better first direction on an S P configuration. The

lgorithm 4: B/S retiming

```
1  Input :  QFP circuit $M$, technology assumptions $ps$, start
     retiming direction $dir$
2  while improvement do
3      select_retimeable_elements($M$, $ps$);
4      set up retiming direction according to $dir$;
5      maximum_flow($M$);
6      get_minimum_cut($M$);
7      move_retimed_elements($M$);
8  end
9  invert direction $dir$;
10 while improvement do
11     select_retimeable_elements($M$, $ps$);
12     set up retiming direction according to $dir$;
13     maximum_flow($M$);
14     get_minimum_cut($M$);
15     move_retimed_elements($M$);
16 end
17 reconstruct_splitter_trees($M$, $ps$);
18 return;
```

algorithm performs two optimization loops in both directions. loop starts by selecting the elements to be retimed (line 3 or 11). Those are the splitters and buffers that can be relocated without exceeding the driving capacity of the fanin gate or the B/S element. In the case of mutually exclusive selections, one is picked using a deterministic random function. Each selected B/S element is a source and a sink of a unitary flow. Next, the algorithm proceeds by selecting the retiming direction, computing the maximum flow, getting the minimum cut, and moving the selected elements to the new position if there is an improvement. Since retiming movements could create redundant splitter trees, the algorithm terminates by reconstructing each splitter tree to be irredundant (line 17). t this step, also possible buffer chains at PIs and POs are removed such that if each PI (PO) is connected to at least x buffers, those x buffers can be removed. This latter condition may only occur if an QFP circuit is not depth-optimal.

 n example of a forward retiming iteration is depicted in Figure 3. Figure 3a shows an initial configuration of the QFP circuit where circles represent gates, rectangles represent buffers and splitters, and arrows represent connections. In this example, the splitter capacity is $s_b = 3$. The algorithm selects the B/S elements in orange to perform retiming as they satisfy all the conditions. Figure 3b shows the configuration after retiming. Two new buffers are inserted (in green). The

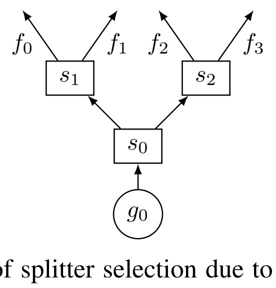
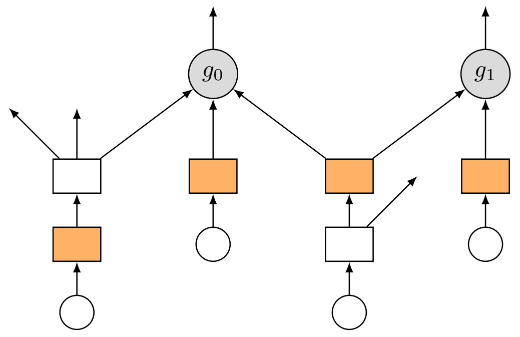
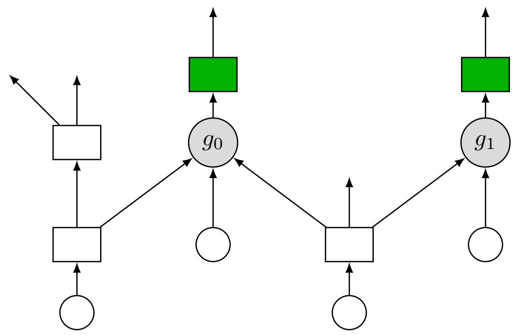


Fig. 2: Example of splitter selection due to fanout limitations.



(a) Initial configuration



(b) Configuration after retiming

Fig. 3: Example of forward retiming of buffers and splitters.

B/S elements have been reduced from 6 to 5.

V. Optimiz tion Flow

In this section, we present a flow for QFP mapping consisting of B/S insertion followed by B/S optimization. The optimization flow is shown in lgorithm 5.

The mapping algorithm takes a MIG as an input expressing the QFP circuit functionality. Then, the circuit is scheduled both in L P and S P using the algorithms in Section III. The circuit with fewer B/S elements is selected for generating the QFP circuit. This choice reduces the run time of the optimization flow since the starting QFP circuit would contain less B/S elements. The best choice for quality of results is instead to proceed with a portfolio approach consisting of carrying the optimization with both the L P and S P schedules and picking the best final result. Then, the algorithm starts the optimization phase. It first performs an initial B/S retiming. Next, an optimization loop applies one pass of the chunk movement algorithm in [10], retiming, and deterministic randomization. The latter function picks a different topological ordering to escape local minima and find different fanout groupings when constructing splitter trees. The loop is iterated until no further improvement or iteration limit is reached. Finally, the mapped and optimized QFP circuit is returned.

VI. Experiments

The optimization framework has been implemented in C++ 17 in the logic synthesis framework *Mockturtle*[1] [19]. The experiments have been conducted on an Intel i5 quad-core 2GHz on MacOS. ll the results were verified to fulfill the path-balancing and fanout-branching assumptions.

lgorithm 5: Splitters and buffers insertion method

```
Input  : MIG network $N_{mig}$, technology assumptions $ps$
Output:  QFP circuit $M$
$M \leftarrow$ empty network;
$d_{ALAP} \leftarrow$  L P($N_{mig}$, $ps$);
$d_{ASAP} \leftarrow$  S P($N_{mig}$, $ps$, $d_{ALAP}$);
if num_bs($d_{ALAP}$) $<$ num_bs($d_{ASAP}$) then
    $M \leftarrow$ dump_circuit($N_{mig}$, $d_{ALAP}$, $ps$);
    dir $\leftarrow$ forward;
else
    $M \leftarrow$ dump_circuit($N_{mig}$, $d_{ASAP}$, $ps$);
    dir $\leftarrow$ backward;
end
bs_retiming($M$, $ps$, $dir$);
while improvement do
    chunk_movement($M$, $ps$);
    bs_retiming($M$, $ps$, $dir$);
    det_randomize($M$);
end
return $M$;
```

. Evaluation of the Retiming lgorithm

In this experiment, we evaluate the retiming-based optimization algorithm. The benchmarks have been obtained from the MCNC benchmark suite [20]. To show the potential of our B/S retiming optimization over other approaches, we apply our method to already optimized QFP circuits. Those circuits have been generated using the scheduling-based optimization in [10], which is available in the library Mockturtle, until convergence. In this experiment, the splitting factor is $s_b = 3$. We assume that PIs and POs need to be balanced and branched.

Table I shows the results of our retiming-based optimization (lgorithm 4). The retiming-based algorithm reduces the number of B/S elements up to 17.1%. Since the approach in [10] does not guarantee depth-optimality, our retiming approach also reduces the depth up to 17.8%.

B. Comparison gainst the State of the rt

In this experiment, we compare our approach to the state-of-the-art method [12]. The benchmarks have been obtained from the authors of the paper[2]. We use the same settings by balancing and branching PIs and POs using a splitting capacity of $s_b = 4$.

Table II shows the results of the comparison. Our approach (lgorithm 5) performs significantly better in almost every benchmark reducing the number of B/S elements up to 20% with a 55$\times$ speed-up compared to the state-of-the-art approach. Our results improve even more using a portfolio approach[3]. The portfolio approach reduces the total number of B/S elements to 50002 while maintaining a 19$\times$ speed-up.

[1] vailable at: https://github.com/lsils/mockturtle

[2]The benchmarks contained redundant gates and inverter cells. Since the Mockturtle framework automatically applies structural hashing to the networks, the starting points of the experiments are slightly different.

[3]Results are not included in the table for space reasons.

T BLE I: Evaluation of the retiming-based optimization

Benchmarks	**Initial circuit**		**Scheduling-based opt. [10]**			**Retiming optimization**		
	Size	Depth	#B/S	#JJs	JJ Depth	#B/S	#JJs	JJ Depth
c1908	381	38	2820	7926	64	2591	7468	60
c432	174	44	2220	5484	68	2100	5244	64
c5315	1270	33	9457	26534	60	9247	26114	60
c880	300	28	2159	6118	42	2060	5920	42
chkn	421	28	1241	5008	38	1126	4778	34
count	119	18	766	2246	29	635	1984	24
dist	535	16	809	4828	28	770	4750	23
in5	443	19	1042	4742	30	972	4602	27
in6	370	17	884	3988	23	884	3988	23
k2	1955	25	4171	20072	43	4167	20064	43
m3	411	13	620	3706	22	620	3706	22
max512	713	17	1078	6434	28	1078	6434	28
misex3	1532	24	2879	14950	38	2879	14950	38
mlp4	462	16	653	4078	26	653	4078	26
prom2	3477	22	5300	31462	33	5300	31462	33
sqr6	138	13	246	1320	20	246	1320	20
x1dn	152	14	428	1768	22	378	1668	20
Total			36773	150664	614	35706	148530	587

The higher JJ depth of our method in benchmark *c2670* is due to the structural hashing pre-processing.

C. Results on the EPFL Benchmark suite

In this experiment, we applied our B/S insertion and optimization method (lgorithm 5) to the EPFL benchmark suite[4] [21] to demonstrate its scalability. The baseline has been obtained by mapping the benchmarks into MIGs using the graph mapper in [22] in the delay mode. In this experiment, PIs and POs are balanced and branched, and the splitting capacity $s_b = 4$. To decrease the run time, we limited the retiming iterations to 250, the size of the chunks to 100, and we executed the main optimization loop once. Moreover, we limited the execution run time budget to 300 seconds.

Table III shows the experimental results after depth-optimal B/S insertion and after carrying the optimization in lgorithm 5. The B/S insertion algorithm scales very well with limited run time for all the benchmarks. The B/S optimization reduces the number of splitters up to 37.83% and the total area up to 24.04%.

VII. Conclusion

In this work, we studied the buffer and splitter insertion problem in QFP circuits. While existing work focused only on area [7], [10], [12], in this paper we stated the importance of delay reduction to minimize the path-balancing costs and consequently benefit the area. We demonstrated that there exists a linear time algorithm based on L P scheduling to insert buffers and splitters such that the resulting QFP circuit is globally depth-optimal. In this regard, we proposed two depth-optimal B/S insertion policies based on L P and S P scheduling that run in linear time to the number of logic gates in the circuit. Next, we presented a novel algorithm to minimize buffers and splitters based on minimum register retiming. We showed that this method improves up to 17% previously optimized QFP circuits. Finally, we proposed a B/S insertion and optimization flow based on the algorithms in this paper and the chunk movement algorithm in [10]. Our approach reduces the number of B/S elements up to 20% while guaranteeing optimal depth and providing a speed-up of $55\times$ compared to the state-of-the-art method.

[4] vailable at: https://github.com/lsils/benchmarks

CKNOWLEDGMENTS

This research was supported by the SNF grant “Supercool: Design methods and tools for superconducting electronics”, 200021 1920981.

References

[1] S. R. Whiteley and J. Kawa, “Progress toward VLSI-capable ED tools for superconductive digital electronics,” in *2019 IEEE International Superconductive Electronics Conference (ISEC)*, pp. 1–3, 2019.
[2] G. Meuli, V. Possani, R. Singh, S.-Y. Lee, . T. Calvino, D. S. Marakkalage, P. Vuillod, L. maru, S. Chase, J. Kawa, and G. D. Micheli, “Majority-based design flow for QFP superconducting family,” *D TE*, p. 6, 2022.
[3] O. Chen, R. Cai, Y. Wang, F. Ke, T. Yamae, R. Saito, N. Takeuchi, and N. Yoshikawa, “ diabatic quantum-flux-parametron: Towards building extremely energy-efficient circuits and systems,” *Scientific Reports*, vol. 9, 2019.
[4] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, “ n adiabatic quantum flux parametron as an ultra-low-power logic device,” *Superconductor Science and Technology*, vol. 26, no. 3, 2013.
[5] N. Takeuchi, M. Nozoe, Y. He, and N. Yoshikawa, “Low-latency adiabatic superconductor logic using delay-line clocking,” * pplied Physics Letters*, vol. 115, no. 7, 2019.
[6] C. L. yala, R. Saito, T. Tanaka, O. Chen, N. Takeuchi, Y. He, and N. Yoshikawa, “ semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors,” *Superconductor Science and Technology*, vol. 33, no. 5, 2020.
[7] R. Cai, O. Chen, . Ren, N. Liu, N. Yoshikawa, and Y. Wang, “ buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits,” in *ICCD*, 2019.
[8] R. Cai, O. Chen, . Ren, N. Liu, C. Ding, N. Yoshikawa, and Y. Wang, “ majority logic synthesis framework for adiabatic quantum-flux-parametron superconducting circuits,” in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, CM, 2019.
[9] D. S. Marakkalage, H. Riener, and G. De Micheli, “Optimizing adiabatic quantum-flux-parametron (QFP) circuits using an exact database,” in *N NO RCH*, pp. 1–6, 2021.

T BLE II: Evaluation of our approach against the state of the art

Benchmark	Initial circuit		State of the art [12]				Our approach			
	Size	Depth	#B/S	#JJs	JJ Depth	Time (s)	#B/S	#JJs	JJ Depth	Time (s)
adder1	7	4	16	74	8	0.13	16	74	8	0.00
adder8	77	17	371	1204	33	0.16	372	1206	33	0.01
mult8	439	35	1833	6300	70	0.47	1688	6010	70	0.06
counter16	29	9	82	338	17	0.18	65	304	17	0.00
counter32	82	13	189	912	23	0.18	154	800	23	0.00
counter64	195	17	419	2134	30	0.23	347	1864	30	0.01
counter128	428	22	895	4652	38	0.56	747	4062	38	0.02
c17	6	3	12	60	5	0.15	12	60	5	0.00
c432	121	26	837	2406	37	0.34	839	2404	37	0.02
c499	387	18	1251	4858	30	0.38	1173	4668	29	0.02
c880	306	27	1723	5296	40	0.45	1511	4858	40	0.10
c1355	389	18	1216	4784	29	0.40	1184	4702	29	0.03
c1908	289	21	1505	4810	35	0.35	1236	4206	34	0.04
c2670	368	21	2055	7392	27	0.71	1932	6072	28	0.09
c3540	794	32	2395	9610	53	1.15	1972	8708	52	0.16
c5315	1302	26	6447	20854	41	4.00	5646	19104	40	0.45
c6288	1870	89	9297	29814	179	5.70	9009	29238	179	0.23
c7552	1394	33	8342	25140	59	85.27	7505	23374	56	1.01
sorter32	480	15	480	3840	30	0.35	480	3840	30	0.01
sorter48	880	20	880	7040	35	0.52	880	7040	35	0.02
alu32	1513	100	17178	43574	170	64.68	13837	36752	169	0.82
Total			57423	185092	989	166.36	50605	169346	982	3.08

T BLE III: Experimental results for QFP mapping on the EPFL benchmark suite

Benchmark	Baseline		Depth-optimal B/S insertion				B/S optimization				
	Size	Depth	#B/S	#JJs	JJ depth	Time (s)	#B/S	B/S (%)	#JJs	JJs (%)	Time (s)
adder	384	129	49788	101880	258	0.00	49535	0.51	101374	0.50	0.39
bar	3016	12	2001	22098	20	0.01	2001	0.00	22098	0.00	0.10
div	57300	2217	1881255	4106310	4371	0.87	-	-	-	-	300
hyp	136109	8762	9035578	18887810	17246	2.78	-	-	-	-	300
log2	24456	200	129547	405830	379	0.10	86705	33.07	320146	21.11	64.18
max	2413	150	69892	154262	160	0.01	68462	2.05	151402	1.85	1.38
multiplier	19710	133	102005	322270	264	0.08	63414	37.83	245088	23.95	43.50
sin	4303	110	18905	63628	188	0.01	14886	21.26	55590	12.63	4.12
sqrt	23238	3366	1791005	3721438	6628	0.49	1343705	24.97	2826838	24.04	284.10
square	12180	126	89516	252112	251	0.03	63630	28.92	200340	20.54	18.30
arbiter	7000	59	27566	97132	63	0.01	25721	6.69	93442	3.80	1.28
cavlc	667	11	663	5328	16	0.00	657	0.90	5316	0.23	0.02
ctrl	118	5	120	948	9	0.00	120	0.00	948	0.00	0.00
dec	304	3	184	2192	7	0.00	184	0.00	2192	0.00	0.01
i2c	1246	11	2742	12960	16	0.00	2670	2.63	12816	1.11	0.09
int2float	237	10	255	1932	13	0.00	246	3.53	1914	0.93	0.01
mem_ctrl	42758	73	216927	690402	114	0.27	215202	0.80	686952	0.50	10.55
priority	988	84	14715	35358	136	0.00	13109	10.91	32146	9.08	1.33
router	267	28	1600	4802	42	0.00	1290	19.38	4182	12.91	0.08
voter	7860	47	19263	85686	86	0.01	15736	18.31	78632	8.23	0.92

[10] S.-Y. Lee, H. Riener, and G. De Micheli, "Irredundant buffer and splitter insertion and scheduling-based optimization for aqfp circuits," in *in Proc. IWLS*, 2021.

[11] E. Testa, S.-Y. Lee, H. Riener, and G. De Micheli, " lgebraic and boolean optimization methods for QFP superconducting circuits," in *Proc. SP-D C*, 2021.

[12] C.-Y. Huang, Y.-C. Chang, M.-J. Tsai, and T.-Y. Ho, " n optimal algorithm for splitter and buffer insertion in adiabatic quantum-flux-parametron circuits," in *ICC D*, 2021.

[13] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," * lgorithmica*, vol. 6, p. 5–35, jun 1991.

[14] N. Katam, . Shafaei, and M. Pedram, "Design of complex rapid single-flux-quantum cells with application to logic synthesis," in *ISEC*, 2017.

[15] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, " diabatic quantum-flux-parametron cell library adopting minimalist design," *Journal of pplied Physics*, vol. 117, no. 17, 2015.

[16] L. marú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: new paradigm for logic optimization," *IEEE Trans. C D*, vol. 35, no. 5, 2016.

[17] . V. Goldberg, " n efficient implementation of a scaling minimum-cost flow algorithm," *Journal of lgorithms*, vol. 22, no. 1, 1997.

[18] . P. Hurst, . Mishchenko, and R. K. Brayton, "Fast minimum-register retiming via binary maximum-flow," in *Formal Methods in Computer ided Design (FMC D'07)*, 2007.

[19] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, and G. D. Micheli, "The EPFL logic synthesis libraries," *CoRR*, vol. abs/1805.05121, 2019.

[20] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0.* Microelectronics Center of North Carolina (MCNC), 1991.

[21] L. marù, P.-E. Gaillardon, and G. D. Micheli, "The EPFL combinational benchmark suite," in *Proc. IWLS*, 2015.

[22] . T. Calvino, H. Riener, S. Rai, . Kumar, and G. De Micheli, " versatile mapping approach for technology mapping and graph optimization," in * SP-D C*, 2022.