Optimizing Adiabatic Quantum-Flux-Parametron (AQFP) Circuits using an Exact Database

Dewmini Sudara Marakkalage
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
dewmini.marakkalage@epfl.ch

Heinz Riener
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
heinz.riener@epfl.ch

Giovanni De Micheli
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

***Abstract*—Adiabatic Quantum-Flux-Parametron (AQFP) is a family of superconducting electronic (SCE) circuits exhibiting high energy efficiency. In AQFP technology, logic gates require splitters to drive multiple fanouts and both the logic gates and the splitters are clocked, requiring path balancing using buffers to ensure all fanins of a gate arrive simultaneously. In this work, we propose a new synthesis approach comprising of two stages: In the first stage, a database of optimum small AQFP circuit structures is generated. This is a one-time, network-independent operation. In the second stage, the input network is first mapped to a LUT network and then the LUTs are replaced with the locally optimum (area or delay) AQFP structures from the generated database in the topological order. Our proposed method simultaneously optimizes the resources used by 1) gates that compute logic functions and 2) buffers/splitters. Hence, it captures additional optimization opportunities that are not explored in the state-of-the-art methods where buffer-splitter optimizations are done after the logic optimizations. Our method, when using a delay-oriented (area-oriented) strategy, achieves over a 40% (35%) decrease in delay in the critical path (the number of levels) and a 19% (21%) decrease in area (the number of Josephson Junctions) as compared to existing work.**

***Index Terms*—AQFP, emerging technologies, majority gates, exact synthesis, logic synthesis**

I. Introduction

Superconducting electronic (SCE) circuits are getting increasingly popular in the electronics industry due to their low energy consumption and high-speed operation. The growing interest in SCE is further fuelled by the escalating challenges and higher costs of transistor downscaling in traditional CMOS technologies. The potential of SCE to revolutionize the electronics industry is widely recognized as evidenced by the growing involvement of the EDA industry in developing tools and synthesis flows for SCE, supported by government-funded programs such as IARPA's SuperTools program [1].

SCE circuits are based on superconductive inductors and Josephson Junctions (JJs) [2]. There are several families of SCE circuits. The examples of emerging technologies in SCE include Rapid Single Flux Quantum (RSFQ) [3], Energy-efficient SFQ (eSFQ) [4], Reciprocal Quantum Logic (RQL) [5], Dynamic Single Flux Quantum (DSFQ) [6], and Adiabatic Quantum-Flux-Parametron (AQFP) [7]. Some technologies, like RQL and AQFP, operate in adiabatic mode and achieve superior energy efficiency using AC-biased junctions. The AQFP technology enables two orders of magnitude less energy consumption than conventional semiconductor technologies even after cooling energy is taken into consideration [8]. This work focuses on the AQFP technology.

This research was supported by the Swiss National Science Foundation grant Supercool 200021_1920981 and the EPFL Open Science Fund.

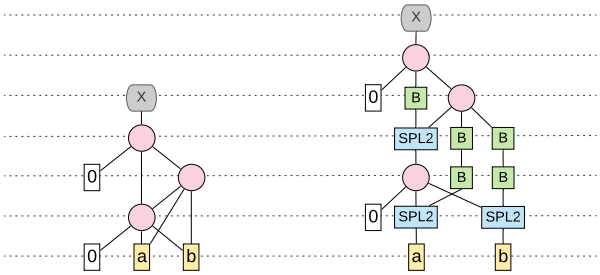

Fig. 1. An example logic network with unit-delay gates (left) and its splitter-inserted, path-balanced version (right).

The AQFP technology provides efficient implementations of majority-3 and majority-5 gates that offer more complex logic functionalities at a comparably low resource usage (see Section II), leading to more compact circuitry. On the other hand, AQFP logic gates cannot directly drive more than one fanout, necessitating clocked splitters to support multiple fanouts. Moreover, the gates are also clocked, requiring each gate's fanins to arrive at the same time via logic paths that are balanced with (clocked) buffers. Figure 1 shows a simple logic network with unit-delay gates (left) and its splitter-inserted, path-balanced version (right). In all figures, green squares labeled "B" denote buffers while blue rectangles labeled "SPL*k*" denote splitters of branching factor *k*.

The path balancing and splitter requirements of the AQFP technology pose additional challenges in logic synthesis because buffers and splitters also significantly affect the area and delay of a circuit. Hence, tailored-for-CMOS synthesis tools are ineffective for AQFP optimizations. As such, there have been several attempts to optimize the splitter insertion and path balancing of AQFP circuits [9]–[11]. However, existing work suffers from one or more of the following weaknesses: i) lack of consideration for interdependent logic paths [10],

ii) the bias towards using balanced splitter trees [9], and iii) the lack of support for more complex logic gates such as majority-5 [9]–[11]. In this work, we show how to mitigate these shortcomings using exact synthesis on small blocks of logic in a given logic network.

We propose a two-stage approach for optimizing logic circuits for the AQFP technology. First, we generate a database of minimum area (the number of JJs for example) AQFP circuits for all single-output, 4-input functions and for a set of different input arrival-time patterns. This stage is network-independent, and is performed only once. The second stage consists of rewriting logic blocks of a larger network in the topological order using locally optimum structures from the database. This stage runs in time linearly in the number of nodes in the input network. Our approach differs from the similar-looking method proposed by Amaru et al. [12] for exact delay synthesis as 1) their database consists of tree structures whereas we consider DAG structures, and 2) our rewriting algorithm is different due to its consideration of multiple fanout nets.

We evaluate our synthesis algorithm on the same subset of MCNC benchmarks considered by Testa et al. [9] using two different strategies, delay-oriented and area-oriented. The former (latter) strategy improves the critical path delay by over 40% (35%) while reducing the area by over 19% (21%) on average compared to the optimized results of Testa et al. [9]. Note that we use the total number of Josephson Junctions as the measure of area and the number of gate levels (including buffers and splitters) as the delay measure.

The paper is organized as follows: In Section II, we describe the relevant background and our motivation for this work. In Section III, we describe our database generation method and introduce the algorithm for synthesizing AQFP circuits. In Section IV, we present our experimental results, and in Section V, we conclude with a brief discussion.

II. Background and Motivation

This section discusses the background and our motivation.

A. AQFP logic circuits

Logic gates in the AQFP technology are mainly constructed using superconductive inductors and JJs [7]. Takeuchi et al. [13] proposed a simple cell library for AQFP technology based on four primitive cells—buffer, inverter, constant, and branch—where a gate is created using an array of primitive cells together with a branch while a splitter is constructed using a buffer and a branch. The majority-3 gate consists of three buffer cells together with a branch, and different fanin inverted versions of a majority-3 gate are constructed by substituting a subset of buffer cells with inverter cells [13]. The 2-input AND and OR gates are constructed by substituting a buffer cell with a constant cell. Each of the three primitive cells, buffer, inverter, and constant consists of two JJs, and hence a splitter also uses 2 JJs while all gates—majority-3, AND-2, and OR-2 as well as all their input-inverted versions—use 6 JJs each. Additionally, the majority-5 gate and its input-inverted versions can be implemented with 10 JJs each. As a majority-3 gate uses the same resources as an AND-2 or an OR-2 gate, Cai et al. [14] proposed that majority logic synthesis is more suitable for optimizing AQFP circuits.

AQFP logic gates cannot directly drive multiple fanouts. Instead, splitters must be used in this case. The gates and the splitters are clocked, and hence, buffers must be inserted to make the circuit pipelined. Depending on the clocking schemes and the design of registers, there can be different requirements on whether splitters are needed for primary inputs, whether path balancing is needed for primary inputs, and how the path balancing is done for primary outputs [15], [16].

B. Majority-inverter graph (MIG)

The k-input majority gate outputs 1 if and only if more than $k/2$ of the inputs are 1. A majority-inverter graph (MIG) is a directed acyclic graph (DAG) where each internal node represents a majority gate, and each directed edge is either a regular edge or a complemented edge indicating the absence or presence of an inverter at the respective fanin [17], [18].

C. NPN equivalence

Two functions f and g are NPN equivalent if f can be obtained from g using a combination of input negations, input permutations, and output negation [19]. For example, although there are $2^{2^4} = 65536$ different 4-input functions, there are only 222 different 4-input NPN classes.

D. Exact synthesis

Exact synthesis is the process of synthesizing circuits to meet exact specifications. For example, it can be used to synthesize the minimum area circuit structures for a given Boolean function while meeting a given set of constraints on the input arrival times. Prior work on exact synthesis includes [12], [20]–[22] and they use different approaches such as decomposition-based, SAT-based, and explicit enumeration-based methods. Since exact methods are computation-heavy, it is impractical to use them for large networks. Instead, they are typically used to construct databases of optimum circuit structures for logic functions with a small number of variables. The database construction is done once for all. Building blocks from the database are then used by separate rewriting algorithms [23] to synthesize circuits.

E. Motivation for our work

The existing work on AQFP synthesis considers logic optimization and buffer-splitter insertion as two independent problems. We identify three shortcomings of existing work:

a) Lack of consideration for interdependent logic paths: Figure 2 shows a logic network on the left, and two possible splitter tree choices for the fanout net of node u assuming we have 1-to-2 splitters. (For simplicity, we disregard the splitter/buffer requirement of all unspecified fanins.) The *logic path interdependencies* make the choice on the right a much better option than the choice in the middle. To elaborate, consider the three logic paths from node u to node v. The

first arrangement in Figure 2 (middle) increases the length of the path from u to v that goes through the rightmost fanin of v, thus increasing the lengths of the remaining paths as well due to the balanced path requirement. The existing algorithms are susceptible to making suboptimal choices in such situations as they reason based solely on the local view in the vicinity of u and hence consider both splitter trees as equally good options.

b) Bias for balanced-splitter trees: Prior approaches naïvely use balanced splitter trees for multiple-fanout nets when deciding the levels of those fanouts [9]. However, always using balanced splitter trees incurs additional buffer costs if the fanouts of a node have to be placed at uneven depths due to other constraints. In such cases, an unbalanced splitter tree can be a better match as shown in Figure 3.

For the given logic network (left), the unbalanced splitter tree (right) costs fewer buffers compared to a balanced splitter tree (middle two trees) assuming 1-to-2 splitters. The second network naïvely uses a balanced splitter tree and adds buffers on top of it whereas the third network optimizes the buffer count by pushing one of the splitters up in the hierarchy. Nevertheless, the network on the right with the unbalanced splitter tree has a better resource usage.

c) Lack of support for more complex gates: None of the prior works support the generation of optimized netlists with majority-k gates for $k > 3$ although the AQFP technology can support efficient implementations of such gates [24]. For example, consider the logic function of majority-5 itself. Using only AND-2, OR-2, and majority-3 gates, computing this function needs at least four gates which costs at least 24 JJs whereas using a single majority-5 gate uses only 10 JJs.

In the AQFP technology, the area overhead of buffers and splitters is quite significant. In [9], the buffers and splitters in optimized circuits amount to 39% of the total number of JJs. Improved path-balancing algorithms can result in significant area and delay reductions, but finding the optimal arrangement of buffers and splitters is a non-trivial task. However, such optimal arrangements can be computed reasonably fast for small logic networks using an enumeration algorithm, and this eliminates the aforementioned drawbacks when synthesizing such networks. We exploit this fact to mitigate those shortcomings when optimizing larger networks. To this end, we *precompute* a database of minimum area AQFP circuits (i.e., considering

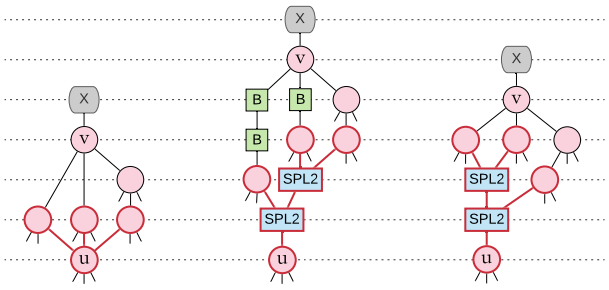

Fig. 2. A part of a logic network with unit-delay gates (left), and its path-balanced versions (middle and right) using two choices of locally optimum splitter trees with 1-to-2 splitters.

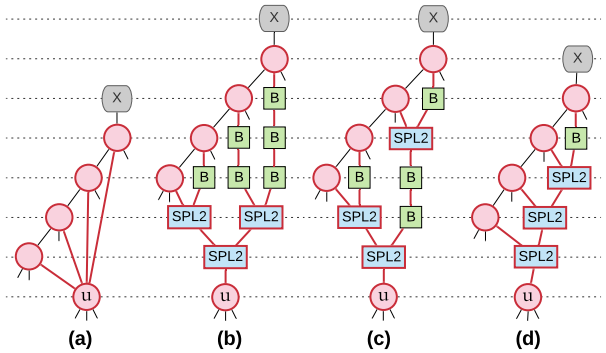

Fig. 3. (a) A logic network with unit-delay gates, (b) its path balanced version using a balanced splitter tree, (c) an optimized version of (a), and (d) optimum path balancing with an unbalanced splitter tree.

the buffers and splitters as well) for all logic functions with a small number of variables under different input arrival time patterns. We then use the database to efficiently rewrite logic blocks of larger networks in topological order using the locally optimum structures from the database.

III. AQFP Resynthesis Approach

In this section, we describe our database generation method and the synthesis algorithm.

A. Generation of the database

The exact synthesis database contains MIGs that give the minimum area (after splitter-insertion and path-balancing) for each 4-input NPN class[1] under different input arrival-time patterns. We identify the input arrival-time pattern of a single-output MIG structure by the depths of its inputs with respect to the output.

The database is generated in three steps which are summarized below. Note that we use the term *DAG* to mean the underlying directed acyclic graph structure of an MIG *without* considering inverters.

First, we systematically enumerate all single-output DAGs with two, three, four, and five leaf nodes (up to four inputs and a constant) and seven[2] gates. To avoid duplicates, we generate DAGs starting with single-level structures and then extending them in a level-by-level fashion by connecting new gates to the previously generated structures. Next, for each generated DAG, we compute the area of realizing those DAGs as a proper splitter-buffer inserted AQFP circuit. If we fix the levels of the nodes, the optimal splitter-buffer insertion for any fanout net can be efficiently computed using a dynamic programming approach using the recursive algorithm in Algorithm 1 together with caching. We then use a recursive backtracking algorithm to enumerate all possible level assignments for nodes and compute the minimum splitter-buffer area for each such level assignment using the aforementioned dynamic program. To limit the search space of the backtracking algorithm, we use

[1]We store MIGs that compute the function with the lexicographically smallest truth-table in each NPN class.

[2]It is known that all 4-input functions can be synthesized with MIGs of at most seven majority-3 gates [25].

Algorithm 1: Computing minimum cost for a given multiset of relative fanout levels S_{lev}, buffer cost c_b, splitter cost c_s, and splitter branching factor f_s.

function `cost` ($S_{\text{lev}} = \{\ell_1, \ldots, \ell_k\}$, c_b, c_s, f_s) **:**
  if $0 \geq \min_{\ell \in S_{\text{lev}}} \ell$ **then return** ∞
  if $|S_{\text{lev}}| = 1$ **then return** $c_b(\ell_1 - 1)$
  $c_{\text{best}} \leftarrow \infty$
  for *all* $T \subseteq S_{\text{lev}}$ *such that* $2 \leq |T| \leq f_s$ **do**
    $c_T \leftarrow c_s$, $\ell_{\min} \leftarrow \min T$
    for $t \in T$ **do** $c_T \leftarrow c_T + c_b(t - \ell_{\min})$
    $c_T \leftarrow c_T$ + `cost` ($\{\ell_{\min} - 1\} \cup S_{\text{lev}} \setminus T$, c_b, c_s, f_s)
    $c_{\text{best}} \leftarrow \min(c_{\text{best}}, c_T)$
  return c_{best}

a gradually increasing bound on the maximum number of levels; this bound is increased until a feasible buffer-splitter arrangement is found. In our algorithm, we use the number of JJs (proportional to the number of primitive cells) to measure the area, but our method can support more general measures such as the physical cell area. Finally, we enumerate the 4-input NPN classes computable by each DAG structure (considering fanin inverters) and store the best-area MIG (i.e., the DAG together with a fanin inverter configuration) for each 4-input NPN class and for different input arrival-time patterns that were discovered during the backtracking search.

B. Synthesis Algorithm

We now describe our algorithm for synthesizing a large logic network as an AQFP circuit using the generated database. The algorithm outputs an MIG (which we call the AQFP circuit) together with an assignment of levels to each gate.

The algorithm, outlined in Algorithm 2, first maps the input network to a 4-LUT circuit using ABC's [26] LUT mapping [27]. Then it considers each LUT in the topological order and replaces it with the locally optimum circuit structure selected from the database. To select the best structure, the algorithm does the following: Consider a LUT n with fanins $n_1, \ldots, n_4$. Since the algorithm operates in the topological order, all of n's fanins are already synthesized as AQFP gates and their respective levels are already computed.

Using the computed level information, the algorithm first computes the arrival times of the fanins of n. To do this, it uses the number of fanouts of each of those fanins, and assumes that each of those fanin nodes use balanced splitter trees at their output to support their respective fanouts. For example, suppose that n'_1 is the synthesized AQFP node for LUT n_1. If n'_1 is at level ℓ and it has 4 fanouts, if we have splitters of branching factor 2, we need two levels of splitters. This means that the arrival time n'_1 with respect to n is $\ell + 2$.

Next, the algorithm finds the function h of LUT n with respect to its inputs $n_1, \ldots, n_4$, its NPN class f, and the input permutation σ that describes how to permute the inputs $n_1, \ldots, n_4$ in order to compute h from f. The algorithm then permutes the arrival times of the fanins of n according to σ.

Algorithm 2: Algorithm to synthesize a given logic network as an AQFP circuit.

$\text{ntk}_{\text{aqfp}} \leftarrow$ Empty AQFP circuit.
$\text{ntk}_{\text{lut}} \leftarrow$ `ABC_LUT_MAP` (ntk_{mig}) .
$m_{\text{lev}} \leftarrow$ Empty map from ntk_{aqfp} nodes to integers.
$m_{\text{sig}} \leftarrow$ Empty map from ntk_{lut} nodes to ntk_{aqfp} signals.
foreach *primary input* p *of* ntk_{lut} **do**
  Create new primary input p' in ntk_{aqfp}.
  $m_{\text{lev}}[p'] \leftarrow 0$, $m_{\text{sig}}[p] \leftarrow p'$.
foreach *node* $n \in \text{ntk}_{\text{lut}}$ *in topological order* **do**
  $n_i \leftarrow$ The i-th fanin of n for $i = 1, ..., 4$.
  $\ell_i \leftarrow$ Arrival time of $m_{\text{sig}}[n_i]$ for $i = 1, ..., 4$.
  $h \leftarrow$ Node function of n.
  $f \leftarrow$ NPN class of h.
  $\sigma \leftarrow$ Input permutation to get h from f.
  $(\ell'_1, \ldots, \ell'_4) \leftarrow \sigma(\ell_1, \ldots, \ell_4)$.
  $g \leftarrow$ Best DAG from DB for f and $(\ell'_1, \ldots, \ell'_4)$.
  $g \leftarrow$ Input permuted g according to inverse of σ.
  $g \leftarrow$ Fanin inverted g such that it computes h.
  Create g in ntk_{aqfp} with inputs $(m_{\text{sig}}[n_1], \ldots, m_{\text{sig}}[n_4])$ and let n' be the root node.
  Update $m_{\text{lev}}[n']$ and $m_{\text{sig}}[n]$.

Next, the algorithm iterates over the database entries with different input arrival-time patterns for NPN class f, and finds the locally optimum entry for the considered arrival-time pattern. The local optimality is defined either as the entry that gives the minimum area or the entry that gives the minimum delay for n. Depending on how the local optimality is defined, we thus have two strategies: area-oriented and delay-oriented. In either case, after selecting the locally optimum structure from the database, the algorithm transforms it such that it computes the function h (instead of the NPN class f). The transformed structure is then constructed in the target AQFP network and the levels of the newly created gates are updated.

Our database consists of a constant number of entries for each NPN class. A look-up by NPN class is done in constant time using a hash-table. Computing an NPN transformation on a 4-input node function also takes constant time. Moreover, as the size of each DAG structure in the database is upper bounded by a constant, transforming a DAG structure back and reconstructing it in the target network also takes constant time. Since ABC's LUT mapping algorithm also runs in linear time due to its dynamic programming approach, the overall running time of our synthesis algorithm is linear in the number of nodes in the input network.

IV. Experimental Results

In this section, we present the experimental results obtained from our AQFP synthesis algorithm and compare them with the results of the AQFP synthesis flow presented in [9]. We compare with [9] as other flows [10], [11] are not open source and hence their results cannot be reproduced. We consider the same subset of 18 MCNC benchmarks [28] used in that work.

TABLE I

RESULTS FOR THE EXPERIMENT WHERE THE PROPOSED AQFP SYNTHESIS ALGORITHM IS APPLIED FOR 10 ITERATIONS UNDER THE ASSUMPTION THAT NO SPLITTER-BUFFER INSERTION IS NEEDED FOR PRIMARY INPUTS BUT PRIMARY OUTPUTS NEED PATH-BALANCING.

	Reference		**All iterations use DB1**								**Last iteration uses DB2**							
			Area-Oriented				**Delay-Oriented**				**Area-Oriented**				**Delay-Oriented**			
Benchmark	Delay (Levels)	Area (#JJs)	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %
5xp1	8	824	8	716	0.00	13.11	8	742	0.00	9.95	8	674	0.00	18.20	8	730	0.00	11.41
c1908	53	5242	39	4512	26.42	13.93	36	5204	32.08	0.72	36	4082	32.08	22.13	32	4498	39.62	14.19
c432	50	2198	35	2178	30.00	0.91	36	2944	28.00	-33.94	32	1994	36.00	9.28	35	2696	30.00	-22.66
c5315	49	18932	33	14976	32.65	20.90	30	16312	38.78	13.84	31	14410	36.73	23.89	29	14850	40.82	21.56
c880	36	4520	24	3406	33.33	24.65	21	3678	41.67	18.63	22	3200	38.89	29.20	20	3402	44.44	24.73
chkn	28	4022	18	3312	35.71	17.65	14	3398	50.00	15.51	15	2900	46.43	27.90	13	2988	53.57	25.71
count	18	1426	13	1184	27.78	16.97	11	1346	38.89	5.61	13	1126	27.78	21.04	11	1326	38.89	7.01
dist	17	4208	13	3802	23.53	9.65	11	3990	35.29	5.18	12	3502	29.41	16.78	10	3480	41.18	17.30
in5	20	4312	15	3522	25.00	18.32	13	3754	35.00	12.94	12	3042	40.00	29.45	12	3116	40.00	27.74
in6	17	3472	12	2978	29.41	14.23	10	2952	41.18	14.98	10	2572	41.18	25.92	8	2552	52.94	26.50
k2	29	18294	19	16380	34.48	10.46	18	16306	37.93	10.87	16	14326	44.83	21.69	16	14372	44.83	21.44
m3	13	3118	12	2964	7.69	4.94	10	3016	23.08	3.27	10	2654	23.08	14.88	9	2680	30.77	14.05
max512	19	5536	14	5018	26.32	9.36	13	5334	31.58	3.65	13	4610	31.58	16.73	12	4636	36.84	16.26
misex3	29	14996	18	11922	37.93	20.50	15	12598	48.28	15.99	17	10580	41.38	29.45	14	10584	51.72	29.42
mlp4	19	3622	13	3222	31.58	11.04	11	3326	42.11	8.17	11	2938	42.11	18.88	10	2998	47.37	17.23
prom2	22	28774	16	26300	27.27	8.60	14	27302	36.36	5.12	14	24374	36.36	15.29	13	24586	40.91	14.55
sqr6	11	1102	9	962	18.18	12.70	8	978	27.27	11.25	8	896	27.27	18.69	7	902	36.36	18.15
x1dn	15	1296	11	1126	26.67	13.12	10	1148	33.33	11.42	10	988	33.33	23.77	10	1010	33.33	22.07
Total	453	125894	322	108480	**28.92**	**13.83**	289	114328	**36.20**	**9.19**	290	98868	**35.98**	**21.47**	269	101406	**40.62**	**19.45**

Similarly, we also use the same AQFP cell library discussed in Section II and use the number of JJs in the synthesized network as the area measure and the number of levels on the critical path as the delay measure. We construct two exact synthesis databases assuming we have 1-to-4 splitters. The difference is the set of DAG structures considered during the construction. Let $D(n, n_3, n_5, \ell)$ denote the set of DAG structure that has at most n gates in total, at most n_3 3-input gates, at most n_5 5-input gates, and at most ℓ levels. To generate the first database, DB1, we consider all DAGS in $D(7, 7, 0, 7)$. To generate the second database, DB2, we consider all dags in $D(7, 7, 0, 7) \cup D(4, 3, 3, 4) \cup D(5, 3, 3, 3)$. To generate the databases, we used a cluster with 48 cores of Intel Xenon E5-2680 v3 CPUs running at 2.5GHz, and 256GB main memory, and each of the three steps was executed using 48 parallel threads. The generation of all DAGs for DB1 consumed $\sim$20 minutes, and the output consists of 440 million DAGs (including different versions obtained by designating one leaf node as the constant node). The cost computation took $\sim$1.5 hours whereas enumerating the computable NPN classes and constructing the final database took $\sim$30 hours. Extending DB1 to DB2 using the DAGs with the given constraints took $\sim$25 hours in total. After removing redundant input depths patterns, DB1 and DB2 consist of only 5744 and 4317 DAGs respectively over all 222 4-input NPN classes.

We perform two experiments with the two databases: In the first experiment, we first synthesize the initial MIG as an AQFP circuit using our proposed algorithm with DB1. Then using the underlying MIG in the synthesized AQFP circuit as the input, the same algorithm was repeatedly applied for a total of 10 iterations, and considered the best result obtained among all iterations. In the second experiment, we additionally consider entries from DB2 as replacement candidates.

The two experiments were done using both the area-oriented and delay-oriented strategies for selecting an appropriate DAG from the database. When using the area-oriented strategy, we select the circuit with the minimum area over the 10 iterations as the best result. Similarly, when using the delay-oriented strategy, we select the circuit with the minimum critical-path length as the best result.

We first perform all experiments under the same assumptions used by Testa et al. [9] that no splitters or buffers are needed on primary inputs but all primary outputs have to be path-balanced using buffers. The results are shown in Table I together with the improvements as compared to the results of Testa et al. [9] (shown in column *reference*). As seen from Table I, the repeated application of our proposed algorithm reduces the delay by 40.62% and decreases the area by 19.45% when the delay-oriented strategy was used, while achieving a 35.98% reduction in delay and a 21.47% decrease in area when the area-oriented strategy was used. It is evident that having majority-5 gates in the database allows the algorithm to achieve up to 7% delay improvements with further area reductions as compared to the case where only majority-3 gates were allowed in the database. Note that, in the output circuits of our algorithm, the percentage of path balancing resources, i.e., the percentage of JJs in splitters and buffers compared to the total number of JJs, is $\sim$25% on average whereas that quantity is over $\sim$39% in [9].

In our results, we also observed that, when majority-5 gates are allowed, 28% (33%) of the logic resources are used by the majority-5 gates in the area-oriented (delay-oriented) strategy, implying that majority-5-like functions often occur as parts of larger logic networks. Such functions include 5-input functions that are in the same NPN-class as majority-5, as well as their versions where one input is repeated. For example, $w(xy + yz + xz) + xyz = \langle w, w, x, y, z \rangle$ is a four input function synthesizable with a single majority-5 gate.

We also performed the same experiments under the assumption that splitters are needed for primary inputs to support multiple fanouts and primary outputs have to be balanced (but, as before, we assumed that path balancing is not needed for primary inputs). Even with these relaxed assumptions, our algorithm achieves better area and delay as compared to [9], which did not use splitters on primary inputs. We remark that our flow supports other assumptions on the need of buffers and splitters on primary inputs and outputs, but due to space constraints we are unable to present the results.

V. Conclusion

We propose a two-stage AQFP synthesis approach: a one-time generation of an exact synthesis database and an efficient algorithm for rewriting small logic blocks with locally optimum structures from the database. Our algorithm performs simultaneous optimizations of logic and path balancing resources that capture more optimization opportunities as compared to prior work in the field and achieves much improved circuits in terms of both delay and area (more than a 40% delay improvement and a 19% area improvement). To the best of our knowledge, our approach is also the first AQFP synthesis approach that can produce AQFP circuits with majority-5 gates. Our results demonstrate that having majority-5 gates can help significantly improve resource usage and reduce delay.

Our database generation method is not restricted to using the number of JJs as the area cost. Instead, it can work with more general cost functions such as the cell area and consider multiple types of splitters with varying branching factors and area. Our database can be further improved by considering more DAG structures at the expense of the one-time computational cost of generating the database.

Our synthesis algorithm depends on an external LUT mapping algorithm. Thus improvements to the LUT mapping stage or skipping LUT mapping altogether by directly integrating the database with a technology mapper may yield better results.

Optimizing path balancing resources in AQFP circuits is a non-trivial problem, and effective path balancing can heavily reduce resource usage in AQFP circuits. The superior performance of our approach makes it attractive as a state-of-the-art AQFP synthesis flow. We believe that expanding on the insights of this work will yield even better results.

References

[1] S. R. Whiteley and J. Kawa, "Progress toward VLSI-capable EDA tools for superconductive digital electronics," in *2019 IEEE International Superconductive Electronics Conference (ISEC)*, 2019, pp. 1–3.

[2] D. S. Holmes, A. M. Kadin, and M. W. Johnson, "Superconducting computing in large-scale hybrid systems," *Computer*, vol. 48, no. 12, pp. 34–42, 2015.

[3] K. K. Likharev and V. K. Semenov, "RSFQ logic/memory family: a new josephson-junction technology for sub-terahertz-clock-frequency digital systems," *IEEE Transactions on Applied Superconductivity*, vol. 1, no. 1, pp. 3–28, 1991.

[4] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 760–769, 2011.

[5] Q. P. Herr, A. Y. Herr, O. T. Oberg, and A. G. Ioannidis, "Ultra-low-power superconductor logic," *Journal of applied physics*, vol. 109, no. 10, p. 103903, 2011.

[6] G. Krylov and E. G. Friedman, "Asynchronous dynamic single-flux quantum majority gates," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 5, pp. 1–7, 2020.

[7] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An adiabatic quantum flux parametron as an ultra-low-power logic device," *Superconductor Science and Technology*, vol. 26, no. 3, p. 035010, 2013.

[8] O. Chen, R. Cai, Y. Wang, F. Ke, T. Yamae, R. Saito, N. Takeuchi, and N. Yoshikawa, "Adiabatic quantum-flux-parametron: Towards building extremely energy-efficient circuits and systems," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[9] E. Testa, S.-Y. Lee, H. Riener, and G. De Micheli, "Algebraic and Boolean optimization methods for AQFP superconducting circuits," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21, New York, NY, USA, 2021, p. 779–785.

[10] R. Cai, O. Chen, A. Ren, N. Liu, N. Yoshikawa, and Y. Wang, "A buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 429–436.

[11] C. L. Ayala, R. Saito, T. Tanaka, O. Chen, N. Takeuchi, Y. He, and N. Yoshikawa, "A semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors," *Superconductor Science and Technology*, vol. 33, no. 5, p. 054006, 2020.

[12] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P.-E. Gaillardon, J. Olson, R. Brayton, and G. De Micheli, "Enabling exact delay synthesis," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 352–359.

[13] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, "Adiabatic quantum-flux-parametron cell library adopting minimalist design," *Journal of Applied Physics*, vol. 117, no. 17, p. 173912, 2015.

[14] R. Cai, O. Chen, A. Ren, N. Liu, C. Ding, N. Yoshikawa, and Y. Wang, "A majority logic synthesis framework for adiabatic quantum-flux-parametron superconducting circuits," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 189–194.

[15] R. Saito, C. L. Ayala, O. Chen, T. Tanaka, T. Tamura, and N. Yoshikawa, "Logic synthesis of sequential logic circuits for adiabatic quantum-flux-parametron logic," *IEEE Transactions on Applied Superconductivity*, vol. 31, no. 5, pp. 1–5, 2021.

[16] S.-Y. Lee, H. Riener, and G. De Micheli, "Irredundant Buffer and Splitter Insertion and Scheduling-Based Optimization for AQFP Circuits," in *[Proceedings of the 30th International Workshop on Logic & Synthesis (IWLS 2021)]*, no. CONF, 2021.

[17] S. B. Akers, "Synthesis of combinational logic using three-input majority gates," in *3rd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1962)*. IEEE, 1962, pp. 149–158.

[18] L. Amarú, P. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.

[19] S. Muroga, *Threshold Logic and its Applications*. New York: Wiley-Interscience, 1971.

[20] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 227–238, 1962.

[21] D. E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison Wesley, 2011.

[22] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "SAT-based exact synthesis: Encodings, topology families, and parallelism," *IEEE Trans. on CAD of Integrated Circuits and Systems*, pp. 1–1, 2019.

[23] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 532–535.

[24] C. Ayala and N. Yoshikawa, personal communication.

[25] M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1842–1855, 2017.

[26] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.

[27] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 354–361.

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