Fanout-Bounded Logic Synthesis for Emerging Technologies

Dewmini Sudara Marakkalage, *Graduate Student Member IEEE,* and Giovanni De Micheli, *Life Fellow IEEE*

***Abstract*—In logic circuits, the number of fanouts a gate can drive is limited, and such limits are tighter in emerging technologies such as superconducting electronic circuits. Moreover, some such technologies, e.g., *adiabatic quantum-flux-parametron* (AQFP), pose additional constraints such as the need for balanced input-to-output paths to ensure proper signal propagation. In this work, targeting emerging technologies, we study the problem of re-synthesizing a logic network with bounded-fanout gates while minimizing area for a given depth. Namely, we 1) formulate the *fanout-bounded synthesis* (FBS) problem for a fixed target logic depth as an *integer linear program* (ILP), 2) propose a scalable top-down approach to construct a feasible solution to the ILP, and 3) extend both the exact and the heuristic approaches to the setting of path-balanced networks. Using our ILP, we obtain the global optimum solutions for a number of benchmarks that serve as ground truth for evaluating heuristic algorithms in both general and path-balanced FBS. Our heuristic algorithm for general FBS achieves an 11.82% better area than the state of the art with matching or better delays while attaining the optimum/near-optimum area for several considered benchmarks. For the path-balanced setting, our heuristic approach achieves 8.76% better delay on average with an average area improvement of 0.5% when using AQFP as the exemplar technology, while achieving more than 17% better delays on several benchmarks.**

***Index Terms*—Fanout-bounded synthesis, Integer linear program, Emerging technologies, AQFP.**

I. Introduction

In digital electronics, the ability to have multiple fanouts per gate allows for compact implementations of complex logic functions. However, increasing the number of fanouts of a gate can negatively impact delay performance, and the maximum number of fanouts a gate can support is typically limited. Therefore, it is important to develop synthesis algorithms that effectively utilize fanouts.

In conventional CMOS technology, fanout optimization has been well-studied, both as a means of improving the critical path delay [1]–[5], and as a method of optimizing special high-fanout nets such as clock and reset signals [6]. However, the techniques developed for CMOS technology are not generally transferable to emerging technologies such as superconducting electronics (e.g., AQFP [7], RQL [8], RSFQ [9]), field-coupled nano-computing technologies (e.g., QCA [10]), and spintronics [11], which generally have tight, explicit fanout bounds and/or significantly different timing models (e.g., clocked gates). Thus the allowed circuit transformations in such technologies can be fundamentally different.

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

D. S. Marakkalage and G. De Micheli are with the Integrated Systems Laboratory, Swiss Federal Institute of Technology Lausanne, 1015 Lausanne, Switzerland.

For instance, in CMOS technology, the delay increase caused by a high number of fanouts can be mitigated by techniques such as transistor sizing. However, this option is not available for post-CMOS technologies. Instead, when designing for emerging technologies that have globally imposed, hard fanout limits, fanout-bounding is achieved through a combination of gate duplications and buffer insertions. This procedure tends to consume a significant portion of resources as compared to CMOS, so it is typically considered relatively early in the synthesis process, e.g., in the logic synthesis stage.

Motivated by the aforementioned differences from CMOS, we first consider the following general *fanout-bounded synthesis* (FBS) problem in the unit-delay model: Given an input logic network and the fanout bounds and area costs of different gate types/buffers, re-synthesize the logic network by means of gate duplications and buffer insertions such that each gate meets its respective fanout bound while the total area is minimized. Note that the unit-delay model encompasses many emerging technologies that have clocked gates (e.g., AQFP, QCA). Zhang and Jiang [12] recently studied this general FBS problem (in the same unit-delay model) and presented an algorithm composed of several heuristics, where the main idea was to duplicate gates if doing so locally reduces the number of buffers (see Section III for more details).

In this work[1], we revisit the FBS problem by taking a rigorous approach: namely, we present the first known *integer linear programming* (ILP) formulation of this problem for a fixed target delay and use it to obtain optimum area FBS solutions for a number of EPFL [14] benchmarks and benchmarks of [15]. Our ILP uses the number of copies and buffers associated with different gates and levels as variables, and has constraints to ensure that there are sufficiently many gate copies and buffers to support all fanouts subject to fanout bounds. As we see in Section IV, this formulation is versatile and can be extended, for example, to facilitate different types of gates and buffers as well as different fanout constraints for primary inputs.

We then present a scalable top-down synthesis algorithm for the general FBS problem based on a heuristic different than that of [12], where we give preference to adding buffers over duplicating gates. Specifically, the main idea of the new approach is to duplicate gates *only if* the critical path delay would be increased otherwise. As we explain in Section III, our heuristic exploits several improvement opportunities we

[1]This manuscript is an extension to a previous work by the authors on general FBS [13]. This work provides more detailed explanations of our techniques and extends both the exact and heuristic approaches to consider additional design constraints such as path balancing.

identified in the algorithm of [12]. We also present an additional optimization step on top of the proposed top-down approach which can be used as a high-effort optimization step to obtain even better results. Our basic top-down heuristic achieves a 10.9% better area as compared to the state of the art [12] while the top-down approach with the additional optimization step allows an 11.82% improvement on average. Notably, the critical path delays of the resulting output networks of our top-down approaches are less than or equal to those obtained by the state of the art because they retain the same logic depth as the original fanout-unbounded network[2].

Next, we consider the FBS with the additional requirement of *path-balancing*, which is a crucial constraint of several emerging technologies such as AQFP and QCA. In these technologies, gates can only drive at most one fanout, and special branching cells called splitters are required to support multiple fanouts. By considering the splitters as buffers with a fanout capacity of at least two, synthesizing for such technologies can be considered a special case of FBS. However, what makes synthesizing for these technologies more challenging is the constraints on the arrival times of fanins of a gate. For example, in AQFP technology, the signal propagation between gates is facilitated by a multi-phase clocking scheme, which requires all fanins of a gate to be clocked in the same phase (see Section II for details). One way to ensure this same-phase-fanins constraint is to require that all fanins of a gate be at the same logic level by adding extra buffers as necessary, which is referred to as path-balancing in the literature.

For FBS in the path-balanced case, we first adapt the aforementioned general-case-ILP to account for the path-balancing constraints considering both scenarios where gate duplications are enabled and disabled. (The latter setting has been studied as the AQFP splitter/buffer insertion problem in a series of research work [15]–[18] as we describe in Section III.)

Then, as with the general FBS problem, we present a scalable heuristic algorithm for path-balanced FBS focusing on AQFP technology. This algorithm starts with a top-down approach resembling our heuristic for the general setting (with some differences to avoid excessively duplicating gates) to determine initial gate/buffer counts, and then follows on with additional optimizations to mitigate the overhead of path-balancing buffers. Remarkably, as compared to the optimum delays in the setting *without* gate duplications, our heuristic *with* gate duplications achieves 8.76% better delays on average together with a 0.5% average area improvement.

In the rest of the paper, we first summarize some concepts useful to better understand our work including the logic network structures we use, timing and node equivalence concepts, and a brief introduction to AQFP technology (Section II). Then we discuss some prior work on general FBS as well as splitter-buffer insertion for AQFP technology (Section III). Next, in Section IV, we describe our ILP formulation for the general FBS, and in Section V, we present our scalable top-down algorithm and related further optimizations. Following that, in Section VI, we extend our approaches from Section IV and Section V to facilitate the path-balancing constraints. Finally, in Section VII, we present our experimental results, and in Section VIII, we conclude with a brief discussion on the results and possible future directions.

[2]Note that, to have a fair comparison with [12], we assume the primary inputs have unbounded fanout capacity.

II. Background

In this section, we first introduce two representations of logic that we use in our algorithms, namely and-inverter graphs (AIGs) and majority-inverter graphs (MIGs). Next, we describe the notion of static timing analysis for the unit delay model and the concept of node equivalence. Finally, we briefly introduce the AQFP technology on which we demonstrate our FBS approaches for the path-balanced setting.

A. And-Inverter Graphs / Majority-Inverter Graphs

The *and-inverter graph* (AIG) is a *directed acyclic graph* (DAG) representation of logic where nodes represent either primary inputs or 2-input AND gates which respectively have in-degree zero or two. AIGs have two possible types of directed edges, representing non-inverted or inverted fanins. The AIG is a universal logic representation, meaning that an AIG can represent an arbitrary logic function, and is supported by numerous logic synthesis tools and libraries such as ABC [19] and mockturtle [20], owing to its simplicity and wider compatibility with many logic synthesis algorithms. At the same time, structural hashing can be easily implemented in AIGs, enabling efficient collapsing of logically equivalent nodes. We use AIG as the preferred logic representation in Section IV and Section V.

The *majority-inverter graph* (MIG) is defined similarly to the AIG; the only differences are that the internal nodes represent 3-input majority gates and have in-degree three. The 3-input majority gate outputs 1 if and only if at least two of the inputs are 1. When one input is tied to constant 0 or 1, the majority gate acts as a 2-input AND gate or a 2-input OR gate. Thus, MIG is also a universal logic representation.

The use of majority gates in logic synthesis has been studied extensively in the past [21]–[23]. Recently Amarù et al. [24], [25] proposed MIG as a new paradigm for logic synthesis. Due to the majority gate being the natural gate in several superconducting technologies (as we see in Section II-D), it is preferable to consider a majority-gate-based logic representation when synthesizing for such technologies. Consequently, we use MIG as the preferred logic representation in Section VI.

B. Static Timing Analysis

In this work, we use the unit-delay model which assumes that a signal incurs a unit delay when it passes through a gate. The arrival time of a node n, denoted by t_n^{arr} is defined as follows: If n is a primary input, $t_n^{\text{arr}} = 0$. Otherwise $t_n^{\text{arr}} = 1 + \max_{m \in \text{FI}(n)} t_m^{\text{arr}}$, where $\text{FI}(n)$ denotes the set of fanin nodes of n. Note that the arrival time of a node is equal to the maximum length of a path from the node to any primary input. Hence, we sometimes use the term *level* to refer to the arrival time. The overall circuit delay (depth of the circuit) is defined as the maximum arrival time of any primary output.

For a given target delay D, the required time t_n^{req} of a node n is defined as follows: If n has no fanout nodes which are internal to the logic network (i.e., all fanouts are primary outputs), $t_n^{\text{req}} = D$. Otherwise, $t_n^{\text{req}} = \min_{m \in \text{FO}(n)} t_m^{\text{req}} - 1$, where $\text{FO}(n)$ denote the set of fanout nodes of node n.

A critical path in a network is an input-to-output path of nodes where each node n on the path satisfies $t_n^{\text{req}} = t_n^{\text{arr}}$. We say a node is critical if it lies on at least one critical path.

C. Node Equivalence

In general, we say two nodes m and n in a logic network are equivalent if their outputs are equal under all possible value combinations of primary inputs. If the input graph contains two or more equivalent nodes, their fanouts can be re-distributed among themselves at the discretion of a synthesis algorithm without altering the overall output of the circuit. However, for a network with many primary inputs, the computation needed to identify all sets of equivalent nodes can be prohibitively expensive. Thus, a more practical approach is to find equivalent nodes by considering a node's function with respect to a small cut, i.e., a set of nodes that separates the considered node from primary inputs. An example of this type of weaker equivalence checking is *structural hashing* which was originally used in IBM CAD tools [26]; For AIGs, a widely used structural hashing technique is to identify each gate with a signature consisting of the gate's fanins and flags denoting which fanins are inverted.

In this work, we do not explicitly check for equivalent nodes; instead, we allow the AIG data structure to internally use structural hashing to collapse any equivalent nodes. For the output logic network, our algorithms may explicitly duplicate some gates, hence we disable structural hashing for the output.

D. AQFP Logic Circuits

Adiabatic quantum-flux-parametron (AQFP) is a superconducting electronics technology with very low power consumption due to adiabatic operations. In AQFP, logic gates are constructed using superconductive inductors and *Josephson Junctions* (JJs) which are based on the *Josephson effect* [27]. The number of JJs in an AQFP circuit is commonly used as a proxy for the area cost.

For AQFP, Takeuchi et al. [28] proposed a simple cell library 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. The different fanin inverted versions of a majority-3 gate are constructed by substituting a subset of buffer cells with inverter cells [28]. Analogously, 2-input AND and OR gates are constructed by substituting a buffer cell with a constant 0 or 1 cell. Each of the three primitive cells, buffer, inverter, and constant, consists of two JJs, and hence a splitter also uses 2 JJs. All gates—majority-3, AND-2, and OR-2—as well as all their input-inverted versions use 6 JJs each.

In AQFP logic, the majority-3 gate is the elementary gate as other gates AND and OR are derived from it. Moreover, these derived gates all have the same area as the original majority-3 gate. As such, Cai et al. [29] proposed that majority-gate-based logic synthesis is more suitable when optimizing logic networks for the AQFP technology.

The output signals of AQFP gates are rather weak and unable to drive more than one fanouts. Instead, when driving multiple fanouts, splitters (or a tree of splitters) must be used to boost the output signal. Depending on the implementation details, a splitter's branching capacity can vary (usually three or four [28], [30]), and in our logic synthesis experiments for AQFP, we assume it is four.

As with many superconducting technologies, AQFP gates are clocked. The logic values are propagated between consecutive gates when their active periods overlap. This overlap is achieved by ensuring that, for each gate n, all fanins of n are clocked by the same phase and n itself is clocked by the next available phase (e.g., for a 4-phase clocking scheme, if fanins of n are activated by a clock in some phase ϕ, then n is activated by a clock in phase $\phi + \pi/4$.) To achieve this kind of overlap throughout the network, the usual practice is

1) to ensure all fanins of a gate are in the same logic level,
2) map consecutive logic levels to consecutive rows of gates/buffers in the physical circuit, and
3) activate consecutive rows of gates by clock signals in consecutive phases.

We remark that, in general, it is not mandatory to have all fanins exactly in the same logic level, but it is sufficient to have them in the same logic level modulo the number of clock phases. Even this requirement can be eliminated by using a more elaborate clocking scheme where non-consecutive clock phases can also overlap [31]. To keep things simple and allow comparisons with recent work on AQFP logic synthesis, in this work, we work in the former setting.

Depending on the design of registers and the clocking mechanism used, there can be different requirements on whether splitters are needed for primary inputs, whether path balancing is needed for primary inputs, and if path balancing is needed for primary outputs [32]. In our proposed FBS approaches for the path-balanced setting, we assume that splitters are needed for primary inputs (which is a notable difference from the general FBS setting where we assume primary inputs have unbounded fanout capacity to be consistent with [12]) and that path-balancing is needed for primary inputs and primary outputs (i.e., all primary outputs are at the same level).

To illustrate synthesis for AQFP under fanout and path-balancing constraints, consider the example logic network on the left of Fig. 1 and two of its fanout-bounded, path-balanced versions in the middle and on the right. The one in the middle does not have any duplicated gates while the one on the right has one gate duplication. In this example, duplicating gates benefits both the area and the delay; the delay is reduced by one logic level and the area is reduced by two JJs.

III. Related Work

In this section, we first discuss some notable work related to FBS and briefly explain how our approach differs from the existing methods. Then, we also discuss some work

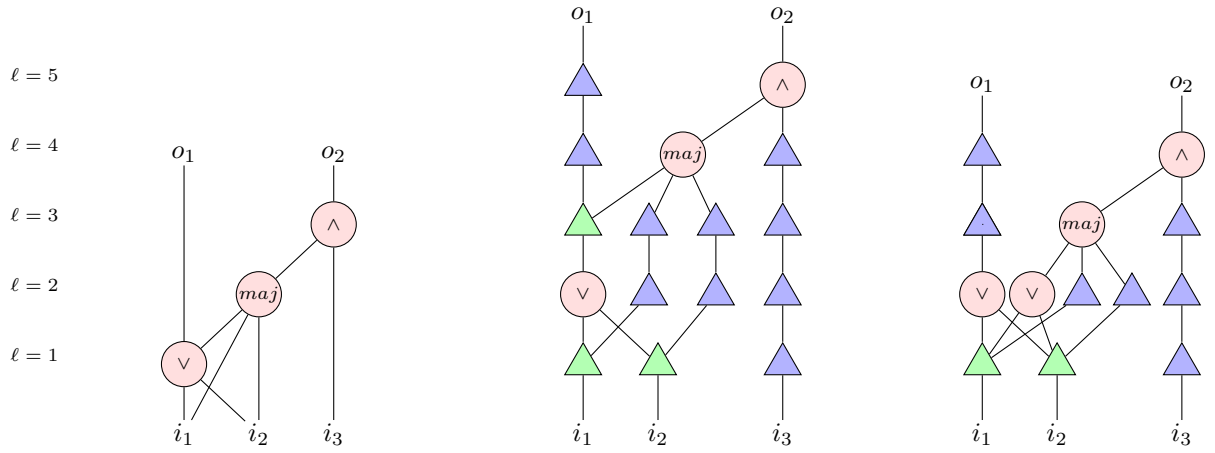

Fig. 1. Example logic network (left) and two of its possible fanout-bounded, path-balanced versions targeting AQFP technology assuming a fanout capacity of 1 for gates and 3 for splitters. (Buffers and splitters are shown by triangles.) The version in the middle does not use any gate duplication whereas the version on the right allows gate duplication resulting in a reduction in both the overall number of logic levels as well as the total area.

related to the AQFP splitter/buffer insertion problem which can be viewed as a special case of FBS with path-balancing constraints.

A. General Fanout-Bounded Synthesis

An early theoretical work on general FBS using gate duplications and buffers by Hoover et al. [33] presented an algorithm that limits the number of fanouts of each gate by any given constant $c \geq 2$ at the expense of a constant factor increase in both the total number of gates and the depth[3]. Their algorithm assumes the natural setting that the input consists of bounded-*fanin* gates.

A vital ingredient of their work that is pertinent to FBS in general is the minimum-size minimum-height buffer tree construction. Namely, given the levels of fanouts of a gate, construct a tree consisting of the gate and a set of buffers such that 1) the gate is at the root, 2) the total number of buffers is minimized, and 3), the height of the tree is minimized. In the case where the gates and buffers have the same fanout bound $t \geq 2$, Golumbic [34] showed how to construct such a tree using a slightly modified Huffman-coding-like algorithm [35].

Recently, Zhang and Jiang [12] studied the problem of general FBS in the unit delay model and proposed an algorithm consisting of several heuristic optimizations. The main idea of their work is to duplicate gates if that results in a buffer reduction in the local neighborhood without significantly affecting the critical path delay. To this end, they proposed a recursive evaluation procedure to determine the number of duplicates for each gate. After the duplicate count for each gate has been determined, for each node in the reverse topological order, their algorithm constructs "skewed" buffer trees using an algorithm similar to [35]. Finally, for each set of equivalent nodes, their buffer trees are considered together and the load is re-distributed. This step does not alter the levels of the nodes but may remove some redundant equivalent nodes.

[3]The depth increase allows for an additive $O(\log_c(\#\text{ primary outputs}))$-term, which is unavoidable under constant-factor size increase considering a network with a single gate that feeds to a large number of primary outputs.

After further analyzing the algorithm of Zhang and Jiang, we identify the following optimization opportunities:

1) The computed numbers of gate duplicates in the recursive evaluation step do not guarantee that the fanout-bounded version achieves the same minimum possible logic depth as the original, fanout-unbounded network. (Note that the original depth is always achievable using gate duplicates under the assumption that the number of fanouts for a primary input is unbounded.)
2) The priority-queue-based method used in [12] for skewed buffer tree construction, although achieves the best possible size for the buffer tree, is *not* guaranteed to achieve the best possible level for the root node *unless* the fanout bound is two. However, for fanout bounds ≥ 3, it is always possible to obtain the best size for the buffer tree as well as the optimal level for the root node using the method proposed by Golumbic [34].
3) In [12], it is not stated how the fanouts are initially assigned to the duplicated copies prior to the skewed buffer tree construction or how their initial levels are determined. For instance, if all copies of a gate are naively placed at the same level when it is possible to place some copies at higher levels, the critical path delay can be adversely affected. However, it is difficult for an algorithm to make such decisions *unless* it already knows the levels of the fanouts.
4) The buffer forest re-balancing step does not guarantee that we get the minimum possible duplicate count (even locally for a considered set of equivalent nodes). This is because the re-balancing step is run only *after* fixing the levels of the duplicated nodes.

In our scalable algorithm for general FBS, we capitalize on all these optimization opportunities. Specifically, by reconstructing the network in the reverse topological order, our algorithm has the full knowledge of the levels of fanouts of a gate, before the gate itself is synthesized. In Section V, we describe in detail how our top-down approach enables exploiting each aforementioned opportunity.

B. Path-Balanced Fanout-Bounded Synthesis

As for the path-balanced setting, there is a line of work on satisfying fanout and path-balancing constraints for the AQFP technology (e.g., [16]–[18]), but these works mainly consider doing so without gate duplications. In literature, this problem is often referred to as the AQFP splitter/buffer insertion problem, and it is a special case of the path-balanced FBS.

In early work on AQFP splitter/buffer insertion, the main idea was to optimize individual fanout nets using different approaches such as dynamic programming and local retiming-like methods for pushing buffers from fanins to fanouts. The work of Lee et al. [16] took a rigorous approach where they presented an exact formulation of the problem as a *satisfiability modulo theory* (SMT) problem using the theory of integer linear arithmetic. Namely, they use the logic depth of each gate as an SMT variable and, for each fanout net, they consider constraints that must be satisfied by any valid splitter/buffer insertion. In contrast, our proposed method uses an ILP to encode the problem and uses the number of gate copies/buffers of each fanout net in each level as variables, which supports gate duplications.

The work in [16] also presented a more elaborate retiming algorithm where an initial splitter/buffer inserted network is further optimized by identifying collections of tightly-connected gates (chunks) where buffers can be pushed forward (from inputs to outputs) or vice-versa to reduce the buffer count. This retiming technique was later used in [17] for area recovery in delay optimal AQFP synthesis. More recently, Fu et al. [18] presented a dynamic programming approach to globally optimize splitters and buffers in AQFP synthesis and an ILP-based solution to approximate the optimum solution.

As a final remark, we emphasize the lack of gate duplications in the existing work on splitter/buffer insertion. However, duplicating gates is an important option that warrants increased attention because it can reduce both the area and the delay as we see in the example of Fig. 1.

IV. Globally Optimum General Fanout-Bounded Synthesis

In this section, we present our ILP formulation of FBS in the unit-delay model. Given an input logic network, a predefined target logic depth D, the gate and buffer costs (e.g., area), and their respective fanout bounds, the proposed ILP finds the minimum cost logic network that meets all fanout bounds, has logic depth at most D, and is functionally equivalent to the input logic network.

We remark that we do not aim to make any logic restructuring; instead, our ILP determines how to duplicate gates and add buffers to the input logic network. For instance, consider the logic network shown on the left of Fig. 2 where the primary inputs $(i_1, \ldots, i_4)$ are shown on the bottom and the primary outputs $(o_1, \ldots, o_5)$ are at the top. If we assume gates and buffers both have fanout capacity 2, then one possible solution to the FBS problem is the network shown on the right, where we have two gates duplications (n_1 and n_3) and added two buffers (shown in blue triangles.)

To derive the ILP, we start with the following notation: Let I be the set of all primary inputs of the input network, let G be the set of all gates, and let $N = I \cup G$ be the set of all nodes. For example, in the example network shown in Fig. 2, $I = \{i_1, \ldots, i_4\}, G = \{n_1, n_2, \ldots, n_7\}$ and $N = \{i_1, \ldots, i_4, n_1, \ldots, n_7\}$. For a node $n \in N$, let $\mathrm{FO}(n)$ be the collection of fanout nodes of n. Let k_n be the number of primary outputs directly connected to node n. Thus, for example, for the network in Fig. 2, we have $\mathrm{FO}(n_1) = \{n_3, n_4\}$ and $\mathrm{FO}(n_3) = \{n_4, n_5, n_6\}$, and $k_{n_2} = k_{n_4} = k_{n_5} = k_{n_6} = k_{n_7} = 1$.

Let c_{gate} be the cost (area) of a gate (we assume the network is homogeneous, but our ILP can easily be generalized to support different types of gates), let c_{buff} be the cost of a buffer, let f_{gate} be the fanout capacity of a gate, and let f_{buff} be the fanout capacity of a buffer.

For example, the setting studied in [12] for FBS assumed gates and buffers each have fanout capacity 2 and considered the optimization of the total node count. For this case, we thus have $f_{\mathrm{gate}} = f_{\mathrm{buff}} = 2$ and $c_{\mathrm{gate}} = c_{\mathrm{buff}} = 1$.

Let $n \in N$ be a node in the original graph. We say a node m in a fanout-bounded circuit is n-equivalent if one of the following holds:

1) n is a primary input and m is the corresponding primary input in the fanout-bounded version.
2) n is a gate with fanins n_1, n_2 and m is a gate with fanins m_1, m_2 such that m_1 is n_1-equivalent and m_2 is n_2-equivalent.
3) m is a buffer such that its fanin m_1 is n-equivalent.

Note that by the third criterion, any buffer in a buffer tree rooted at an n-equivalent gate is also n-equivalent. According to this definition, in the example fanout-bounded network (assuming $f_{\mathrm{gate}} = f_{\mathrm{buff}} = 2$) shown on the right of Fig. 2, there are two n_1-equivalent gates and two n_2-equivalent gates. Moreover, the two buffers represented as blue triangles in level 2 are n_2-equivalent.

1) Variables: We use two kinds of integer variables. For each node $n \in N$ and for each level $\ell \in \{1, \ldots, D\}$, we introduce variables $g_{n,\ell}$ to denote the number of gate copies in level ℓ in the fanout-bounded circuit that are n-equivalent. Similarly, we introduce variables $b_{n,\ell}$ to denote the number of buffers in level ℓ in the fanout-bounded circuit that are n-equivalent. For example, for the logic network shown in Fig. 2, the introduced variables take the following values: $g_{n_1,1} = 2, g_{n_2,1} = 1, g_{n_3,2} = 2, g_{n_4,3} = 1, g_{n_5,3} = 1, g_{n_6,3} = 1, g_{n_7,3} = 1, b_{n_2,2} = 2$ and $g_{n,\ell} = 0$ for all unspecified variables $g_{n_q,\ell}$ with $q \leq 7$ and $\ell \leq 3$.

2) Constraints: Next, we introduce constraints to ensure that the values of variables indeed correspond to a valid fanout-bounded logic network that is equivalent to the input network. To this end, we first have that $g_{n,0} = 0$ and $b_{n,0} = 0$ for all $n \in N$ since there cannot be any gates or buffers in the same level as the primary inputs. (In fact, these variables are redundant and we can write the ILP without them, but having these variables with the above constraint makes it easier to specify the remaining constraints in a concise manner.) Next, consider a fixed level $\ell \in \{1, \ldots, D\}$ and a fixed gate $n \in G$.

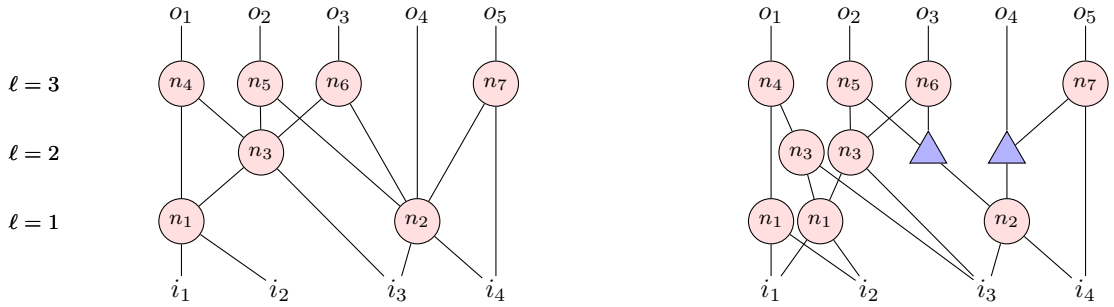

Fig. 2. Example logic network (left) and a possible fanout-bounded version assuming a fanout limit of 2 (right).

We denote by $\mathrm{avl}(n, \ell)$, which stands for "availability of n-equivalent signals by level ℓ," the total fanout capacity of all n-equivalent gates/buffers that are placed in levels strictly less than ℓ. Note that

$$\mathrm{avl}(n, \ell) = \sum_{\ell'=0}^{\ell-1} (f_{\mathrm{buff}} \cdot b_{n,\ell'} + f_{\mathrm{gate}} \cdot g_{n,\ell'}),$$

which is a linear function of the ILP variables. We denote by $\mathrm{req}(n, \ell)$, which stands for the "requirement of n-equivalent signals by level ℓ," the total fanout requirement of n-equivalent gates/buffers by all gates and buffers in level ℓ or below. Note that each copy of a fanout of an n-equivalent gate increases the fanout requirement by one, and each n-equivalent buffer also increases the fanout requirement by one. Namely, we can write

$$\mathrm{req}(n, \ell) = \sum_{\ell'=1}^{\ell} \left(b_{n,\ell'} + \sum_{m \in \mathrm{FO}(n)} g_{m,\ell'} \right),$$

which is again a linear function of the ILP variables.

Now, observe that, in any variable assignment that corresponds to a valid fanout-bounded network with depth D, it must hold that $\mathrm{avl}(n, \ell) \geq \mathrm{req}(n, \ell)$ for all $n \in G$ and $\ell \in 1, \ldots, D$. To see this, consider any valid depth-D fanout-bounded version of the input network, and let $g_{n,\ell}, b_{n,\ell}$ be the corresponding ILP variable values. Fix any gate $n \in G$ and let $\ell = 1$. Note that for any gate $m \in \mathrm{FO}(n)$, $g_{m,1}$ must be 0. Otherwise, there must be a copy of n at level 0, which is a contradiction as n is not a primary input. Similarly, there cannot be any n-equivalent buffer at level 1 either. Thus it must hold that $\mathrm{avl}(n, 1) = 0 \geq 0 = \mathrm{req}(n, 1)$. Now, suppose that $\mathrm{avl}(n, \ell) \geq \mathrm{req}(n, \ell)$ must hold for any valid depth-D fanout-bounded version. We inductively show that $\mathrm{avl}(n, \ell + 1) \geq \mathrm{req}(n, \ell + 1)$ must also hold. Observe that the total number of connections between n-equivalent gates/buffers and their fanouts that must cross the boundary between level ℓ and $\ell+1$ is at least $\sum_{m \in \mathrm{FO}(n)} g_{m,\ell+1} + b_{n,\ell+1}$. The total remaining capacity of n-equivalent gates/buffers that are at levels below ℓ is $\mathrm{avl}(n, \ell) - \mathrm{req}(n, \ell)$. Thus the additional capacity needed to support all crossing connections must be provided by n-equivalent gates/buffer that are at level ℓ. Namely, we must have

$$f_{\mathrm{gate}} \cdot g_{n,\ell} + f_{\mathrm{buff}} \cdot b_{n,\ell} \geq \sum_{m \in \mathrm{FO}(n)} g_{m,\ell+1} + b_{n,\ell+1} - (\mathrm{avl}(n, \ell) - \mathrm{req}(n, \ell)),$$

which yields

$$\mathrm{avl}(n, \ell) + f_{\mathrm{gate}} \cdot g_{n,\ell} + f_{\mathrm{buff}} \cdot b_{n,\ell} \geq \mathrm{req}(n, \ell) + \sum_{m \in \mathrm{FO}(n)} g_{m,\ell+1} + b_{n,\ell+1},$$

or equivalently, $\mathrm{avl}(n, \ell+1) \geq \mathrm{req}(n, \ell+1)$ after re-arranging.

Finally, we ensure that we have enough capacity remaining in n-equivalent gates/buffers to support the respective primary outputs (if any). Namely, for all n, it must hold that

$$\mathrm{avl}(n, D + 1) - \mathrm{req}(n, D) \geq k_n.$$

The same can be achieved by *viewing* all fanouts connected to a gate n as n-equivalent buffers placed at level $D + 1$, and simply adding the constraint $\mathrm{avl}(n, D + 1) \geq \mathrm{req}(n, D + 1)$.

We thus get the following ILP formulation for FBS under a predetermined depth bound D, where the objective function is to minimize the total area.

Minimize $\sum_{n \in G} \sum_{\ell=1}^{D} (c_{\mathrm{gate}} \cdot g_{n,\ell} + c_{\mathrm{buff}} \cdot b_{n,\ell})$,

Subject to

$$\begin{aligned}
\mathrm{avl}(n, \ell) - \mathrm{req}(n, \ell) \geq 0 \quad & \forall\, n \in N, 1 \leq \ell \leq D, \\
\mathrm{avl}(n, D + 1) - \mathrm{req}(n, D) \geq k_n \quad & \forall\, n \in N, \\
g_{n,0} = 0 \quad & n \in G, \\
b_{n,0} = 0 \quad & n \in N, \\
g_{n,\ell}, b_{n,\ell} \in \mathbb{Z} \quad & \forall\, n \in N, 1 \leq \ell \leq D.
\end{aligned}$$

Let OPT be the optimum area of a fanout-bounded version of the input network with maximum depth D. Since any such valid network corresponds to a feasible solution for the ILP, it is clear that the value of ILP is at most OPT. We now give an algorithm (Algorithm 1) to transform any feasible ILP solution to a fanout-bounded network of maximum depth D, which is equivalent to the original network, thus showing that our ILP in fact finds the optimum area.

The algorithm first sorts all variables $g_{n,\ell}, b_{n,\ell}$ in the increasing order of ℓ. Then, considering the variable values in

Algorithm 1: Algorithm for constructing a fanout-bounded network using a feasible solution to the ILP.

input : Input network ntk, parameters $f_{\text{gate}}, f_{\text{buff}}$, and a feasible ILP solution $g_{n,\ell}, b_{n,\ell}$ for $n \in N$ and $0 \leq \ell \leq D$.
output: A fanout-bounded version of ntk.

1 Let newsig be a map from nodes in ntk to a queue of pairs (new node, remaining capacity)
2 **for** *all* $p \in$ *primary inputs of* ntk **do**
3 newsig[p].$push$((newntk.create_pi(), ∞))
4 Let $data$ be an empty list.
5 **for** *all nonzero* $g_{n,\ell}$ **do** Add $(\ell, n,$ "$gate$") to $data$
6 **for** *all nonzero* $b_{n,\ell}$ **do** Add $(\ell, n,$ "$buff$") to $data$
7 Sort $data$ in the ascending order of levels.
8 **for** *all* $(\ell, m, t) \in data$ *in the ascending order of levels* **do**
9 **if** t = *"gate"* **then**
10 Look up fanins of m in newsig.
11 newgate $\leftarrow$ Create a new gate by choosing the first available equivalent fanins in newsig.
12 Decrement remaining capacity for used fanin nodes and remove them from the queue if remaining capacity reach zero.
13 newsig[m].$push$((newgate, f_{gate}))
14 **else**
15 newbuff $\leftarrow$ Create a new buffer by choosing the first available equivalent node in newsig[m].
16 Decrement remaining capacity for the used fanin.
17 Pop from newsig[m] if remaining capacity is zero.
18 newsig[m].$push$((newbuff, f_{buff}))
19 **return** the constructed network.

that order, construct the $g_{n,\ell}$ gate copies or $b_{n,\ell}$ buffers in a new network. To facilitate this construction, for each $n \in N$, the algorithm maintains a queue of currently constructed n-equivalent gates/buffers together with their remaining fanout capacities. Each time it uses such a gate/buffer, it decrements the count; once the count reaches zero, the corresponding gate/buffer instance is removed from the queue. Since the algorithms construct gates/buffers in a level-by-level fashion using a feasible variable assignment, we can see that the algorithm always has sufficient equivalent signals in the corresponding queues when executing Line 11 and Line 15.

V. Top-Down Heuristic Approach for General Fanout-Bounded Synthesis Problem

In this section, we first present our scalable top-down heuristic algorithm that greedily finds a feasible solution to the derived ILP. We then propose an additional optimization step that we can integrate with the top-down approach that allows further area reductions in certain cases.

Although solving the ILP introduced in Section IV gives the optimum solution, solving it optimally for large networks which we often encounter in practice is a prohibitively expensive computation, and hence not a viable approach in many practical settings. On the other hand, the top-down approach we propose in this section is scalable to very large networks as it runs in $O(S \log S)$ time where S is the size of the input network (i.e., the number of wires in the network). Although this approach is not optimum in general, we note that it achieves optimum or near-optimum areas for several considered benchmarks in our experiments.

In the proposed approach, we consider the gates $n \in G$ in the reverse topological order, and for each n in this order, determine values for variables $g_{n,\ell}$ and $b_{n,\ell}$ such that the constraints $\text{avl}(n, \ell) - \text{req}(n, \ell) \geq 0$ and $\text{avl}(n, D+1) - \text{req}(n, D) \geq k_n$ are satisfied. Since we consider the nodes in the reverse topological order, when we consider a node n, we already know the levels of all fanouts of n-equivalent gates/buffers except for those fanouts that arise due to fanins of n-equivalent buffers. We call those fanouts *external fanouts* of n-equivalent gates/buffers.

When determining the values for $g_{n,\ell}$ and $b_{n,\ell}$, we prefer minimizing the number of gate duplicates by utilizing buffers as much as possible to support the fanout requirement. This decision is motivated by the following facts: First, duplicating a gate will increase the fanout requirement of other nodes: For example, suppose that n's fanins are m_1 and m_2. Then, duplicating a n-equivalent gate increases the fanout load of m_1 and m_2-equivalent gates/buffers. This is in contrast to adding a buffer which only increases the fanout load by one. Secondly, it is natural to assume that the area of a buffer is not more than that of a gate, and the fanout capacity of a buffer is usually more than that of a gate. Thus, in terms of area, replacing a gate copy with a buffer is always beneficial.

However, we cannot completely eliminate gate duplication because the addition of buffers can increase the number of logic levels (i.e., the critical path length). Recall that t_n^{arr} is the minimum level node n can be at even if we assume unbounded fanout capacities. Thus, for any $\ell < t_n^{\text{arr}}$, setting $g_{n,\ell}$ to a non-zero value makes the solution infeasible. Similarly, for any $\ell \leq t_n^{\text{arr}}$ (note the inclusion of equality), setting $b_{n,\ell}$ to a non-zero value also makes the solution infeasible.

For given levels of external fanouts of n-equivalent gates/buffers and the minimum possible level (i.e., t_n^{arr}) for an n-equivalent gate, we use Algorithm 2 to determine the values of $g_{n,\ell}$ and $b_{n,\ell}$ variables by considering each node in the reverse topological order. We then use Algorithm 1 to construct the corresponding fanout-bounded logic network.

We remark that our top-down approach is fundamentally different from the work of Zhang and Jiang [12]. In [12], a set of n-equivalent gates and their corresponding levels are already determined when the buffer-forest re-balancing algorithm is run in order to reduce the number of gate duplicates. This can lead to some redundant gate copies that remain in the network even after re-balancing is performed. In contrast, our algorithm uses Algorithm 2 to decide the set of n-equivalent gates that we absolutely need *along with their levels*, thus redundant gate copies are never created. Moreover, in the "skewed buffer tree construction" and "buffer-forest re-balancing" algorithms of [12], there can be situations where it does not construct the best buffer tree/forest when $f_{\text{gate}}, f_{\text{buff}} > 2$ and $c_{\text{gate}} > c_{\text{buff}}$. To see this, suppose that $f_{\text{gate}} = f_{\text{buff}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$ and consider the fanout net shown in Fig. 3 (a). The algorithm of [12] may either decide to duplicate node n and produce the forest shown in Fig. 3 (b) which has a cost of $2 \cdot c_{\text{gate}}$ or it may construct the skewed buffer tree shown in Fig. 3 (c) where the node n is placed at level 4. However, the buffer

Algorithm 2: Algorithm for determining $g_{n,\ell}$ and $b_{n,\ell}$ values for a node $n \in N$, given ℓ_n^{min} and the levels of all external fanouts of n-equivalent gates/buffers.

input : Input network ntk, parameters $f_{\text{gate}}, f_{\text{buff}}$, a node n, t_n^{arr}, and a list $folev_n$ of levels of n's fanouts.
output: Values of $g_{n,\ell}, b_{n,\ell}$ variables for $\ell = 1, \ldots, D$.

```
Set g_{n,ℓ}, b_{n,ℓ} = 0 for all ℓ
for t = 1 to length(folev_n) do
    Let rem ← length(folev_n) − t · f_gate
    if rem ≤ 0 then
        for i = 1 to length(folev_n) in steps of f_gate do
            Increment g_{n, folev_n[i]−1}.
            return variable values
    s ← rem mod (f_buff − 1)
    if s > 0 then
        Add f_buff − s many copies of ∞ to folev_n (i.e.,
          dummy fanouts with unbounded required time).
    Use the skewed buffer tree construction from [12] until
      we have t buffer trees.
    if the root levels of all buffer trees are at least t_n^arr then
        Set g_{n,ℓ} and b_{n,ℓ} according to the construction.
        return variable values
```

tree shown in Fig. 3 (d) is better than both the options; it has a lower area than the one in Fig. 3 (b) and gives a better placement for node n than the one in Fig. 3 (c). In contrast to [12], our algorithm always constructs the optimum buffer forest for given levels of external fanouts and t_n^{arr}. Namely, for $r = 1, 2, \ldots$, we consider r copies for the root gate, employ a modified version of the algorithm of Golumbic [34] to derive r buffer trees, and find the minimum value of r such that roots of all trees meet the arrival time requirement.

A. Improved Top-Down Approach with Over-Duplication

Recall that in our vanilla top-down approach, for each fanout net, we find the smallest buffer forest that does not increase the overall critical path length. The intuition behind settling for the smallest buffer forest is to minimize duplication of gates, and hence avoid unnecessarily increasing the load on the fanins of those gates.

One potential drawback of this frugal approach is the following: Consider a scenario where we may have the option of placing two copies of a node n at level $t_n^{\text{arr}}+1$. However, we may end up placing a single copy of n at level t_n^{arr} instead, thus forcing more duplications for n's fanin nodes as *their* fanout nets do not have enough slack to add buffers. To illustrate this point, assuming that $f_{\text{gate}} = f_{\text{buff}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$, consider the time our algorithm processes the fanout net of node n in the setting shown on top in Fig. 4 where the levels are already decided for all nodes except n, n_1 and n_2. Since the naive top-down approach greedily tries to minimize the number of duplicates for n, it will be placed at level 2 (no duplication) with one buffer at level 3 as shown in the middle of Fig. 4. This forces both n_1 and n_2 to be duplicated (unless the critical path length is to be increased) which results in an overall cost of $5 \cdot c_{\text{gate}} + c_{\text{buff}}$ for the fanout nets of n, n_1, and n_2. However, if we allow the locally suboptimal choice of duplicating n, it is possible to place two copies of n in level 3. This allows more room for fanout nets of n_1 and n_2 to have buffers, resulting in the outcome shown at the bottom of Fig. 4 with an overall cost of $4 \cdot c_{\text{gate}} + 2 \cdot c_{\text{buff}}$ (which is strictly a better cost when $c_{\text{gate}} > c_{\text{buff}}$). As such, allowing more duplicates than absolutely necessary (i.e., *over-duplication*) can be good if that provides more room for the fanins to have buffers and prevents them from being duplicated.

In an improved version of our top-down approach, we incorporate this idea of over-duplication as follows: For the fanout net of a considered node n, instead of stopping the algorithm at minimum possible number of trees t, we continue increasing t and construct the corresponding buffer forests. For each such buffer forest, we consider the overall area incurred by the fanout net of the considered node *and* the fanout nets of its fanin nodes, *assuming that we do not use over-duplication for those fanin nodes*. Then for node n, we choose the buffer forest that gives the minimum overall area computed in the above step.

There are two issues with this approach: First, due to the top-down implementation, when considering node n, all levels of its fanouts (including their potential copies) are known. However, for a fanin m of n, there can be some fanouts that are yet to be considered by the algorithm, and hence their final levels are not known. Secondly, suppose that a node m has k fanouts. For each of those fanouts, the cost of the fanout net of m will be re-evaluated multiple times. I.e., the fanout net of m is evaluated at least k-times. Since each evaluation also takes time at least linear in k, the total work involved in evaluating a node's fanout net can be very expensive for high-fanout nodes.

To circumvent the first issue, we propose to use a proxy level for the so-far unconsidered nodes; namely, we use their maximum possible level (i.e., the required time) as the proxy

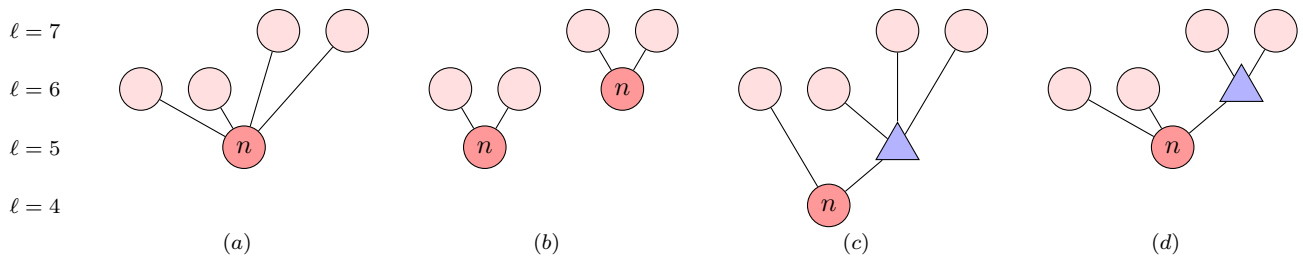

Fig. 3. A fanout net for a node n with levels of fanouts already decided (a), two possible outcomes for the fanout net of n if the algorithm of [12] is used (b and c), and the optimum buffer tree for n (d) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$.

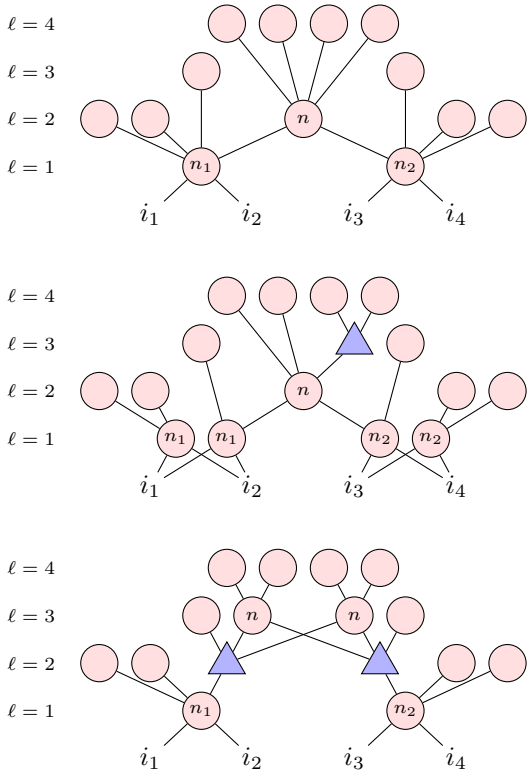

Fig. 4. An intermediate step of fanout-bounded synthesis with levels decided for all nodes except n, n_1, n_2 (top), the synthesized fanout nets by the algorithm described in naive top-down approach (middle), and the synthesized fanout nets if over-duplication allowed (bottom) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$.

level. To mitigate the effects of the second issue, we set a constant bound F_{max} (e.g., 10) and ignore nodes with more than F_{max} fanouts when computing the overall area impact.

VI. Path-Balanced Fanout-bounded Synthesis

In this section, we focus on FBS with the additional requirement of path-balancing.

A. ILP Formulation for the Global Optimum

Recall that the path-balancing constraint states that all input-to-output paths are of the same length. Equivalently, for a gate in level ℓ, all of its fanins must be in level $\ell - 1$. Thus for a gate or primary input n in the input network and for a level ℓ in the output network, it must hold the following: The total available fanout capacity of all n-equivalent nodes in level $\ell-1$ must be at least the total required number of n-equivalent signals by nodes in level ℓ. We can easily incorporate this constraint into the ILP of Section IV by simply redefining $\text{avl}(n, \ell)$ and $\text{req}(n, \ell)$ as

$$\text{avl}(n, \ell) = f_{\text{buff}} \cdot b_{n,\ell} + f_{\text{gate}} \cdot g_{n,\ell},$$

and

$$\text{req}(n, \ell) = b_{n,\ell} + \sum_{m \in \text{FO}(n)} g_{m,\ell}.$$

As discussed in Section II, the FBS with path-balancing constraints is a generalization of splitter/buffer insertion for AQFP technology, and AQFP technology can have different assumptions on the need for buffers/splitters on primary inputs and primary outputs. In particular, the requirement that *all* input-to-output paths must be of the same length falls under the assumption that both primary inputs and primary outputs need path-balancing.

However, our ILP is versatile as it can be adapted to different AQFP-technology-specific assumptions. For example, we can remove the path-balancing requirement on primary inputs by retaining the definitions of $\text{avl}(n, \ell)$ and $\text{req}(n, \ell)$ from Section IV for nodes $n \in I$, i.e., the primary inputs. Similarly, we can remove the path-balancing requirement on primary outputs by retaining those definitions only in the constraint $\text{avl}(n, D + 1) - \text{req}(n, D) \geq k_n$. Moreover, if we need to also enforce that primary inputs need splitters to support multiple fanouts, we can add constraints dictating $g_{n,0} = 1$ and $g_{n,\ell} = 0$ for all $n \in I$ and $\ell > 0$. (In the ILP for the general fanout-bounded setting with no fanout limit on primary inputs, we simply omitted these constraints. This allows the ILP solver to place as many copies of primary inputs anywhere in the network, which is effectively equal to assuming unbounded fanout capacity.)

In addition to supporting the different AQFP-specific assumptions, we can also change the ILP to match the original splitter/buffer insertion problem where duplicating gates is not an option. To this end, we simply have to introduce a new constraint that $\sum_{1 \leq \ell \leq D} g_{n,\ell} = 1$ for all gates $g \in G$.

B. Scalable Heuristic Approach

In the path-balanced setting, we need buffers not only to support multiple fanouts, but also to ensure that all input-to-output paths are of the same length. If we naively use the same top-down approach from the general FBS for the path-balanced setting, it can unnecessarily increase the area due to path-balancing buffers. To see this, suppose that we have a gate n whose arrival time is 1, but its only fanout is determined to be in level 3 by our top-down algorithm. In this case, the algorithm prefers to keep n in level 2 (as opposed to 1) because the main idea of the algorithm from Section V was to keep gates in the highest level possible to give sufficient room for its fanins to have buffers. Now suppose that n's fanins have no other fanouts, in which case, we will need two path-balancing buffers at n's fanins. However, if we placed n in level 1 instead, we could only use one path-balancing buffer at n's output. In general, the situation can be much worse: for example, we could have a block of logic that has k_1 outputs and k_2 inputs in place of n. If $k_1 < k_2$, moving the whole logic block down by 1 level can save $k_2 - k_1$ buffers. On the other hand, if $k_2 < k_1$, then the algorithm's choice to keep the logic block in the highest possible level is meaningful.

Taking such scenarios into account, for the path-balanced setting, we start with a top-down approach similar to Section V to determine initial gate/buffer counts in different levels (i.e., values for variables $g_{n,\ell}$ and $b_{n,\ell}$), but we then perform an additional optimization to modify these gate/buffer counts to further reduce the area. To this end, we first identify (gate, level)-pairs that may correspond to potential path-balancing

buffers. If all fanins of a gate are path-balancing buffers, we can push the buffers towards the output of the gate. In general, this can be done on blocks of logic whose inputs all correspond to path-balancing buffers.

This kind of retiming techniques have already been considered in the past [16], [17], but they work on existing AQFP netlists. Our proposed method is more general and works on gate/buffer counts in each level, *before* constructing the netlist, and hence it is able to capture more retiming opportunities. To illustrate, consider the part of a netlist shown in Fig. 5 (b), where we assume that splitter fanout capacity is 2 for the sake of simplicity. The existing retiming techniques can optimize this by moving node *a* one level down to obtain the configuration in Fig. 5 (c), saving one buffer in the process. However, these algorithms fail to identify an optimization opportunity in for the configuration in Fig. 5 (a) because one of the fanins of node *a* is not a path-balancing buffer but a splitter. Our approach, instead works on gate/buffer counts in each level and hence is able to identify the optimization opportunity in both scenarios. Namely, for each fanin x of node *a*, we check if we have a potential path-balancing buffer by checking

1) if we have x-equivalent buffers in the lower level,
2) and if we can isolate one path-balancing buffer (a buffer with fanout one) from those.

To check the first condition for a fanin x of node *a* in level ℓ, we check if $b_{x,\ell-1} > 0$. For the second condition, we check whether the remaining x-equivalent nodes in level $\ell - 1$ can still satisfy the requirement of remaining fanouts of x in level ℓ *after dedicating a single x-equivalent buffer to supply node* a*'s fanin*; namely, we check if $\mathrm{avl}(x, \ell - 1) - f_{\mathrm{buff}} \geq \mathrm{req}(x, \ell) - 1$. After optimizing the gate/buffer counts with this improved retiming step, we construct an AQFP netlist using Algorithm 1. Then we also run the state-of-the-art retiming from [17] on the constructed circuit to further optimize buffer counts.

In addition to the retiming, our initial top-down heuristic has some minor differences with respect to Section V. Namely, when computing the arrival times for signals, if the originating gate of a signal has more than one fanout, we assume a delay of 2 (instead of 1) accounting for an additional splitter at the output of that gate. This is an AQFP-specific setting: in AQFP, the gates can only support one fanout, and if we always assume a delay of 1 for a gate, the top-down algorithm can end up excessively duplicating gates to meet this delay bound. However, if a gate is in the critical path, has only two fanouts, and if its fanins will have splitters added at their outputs (i.e., they have multiple fanouts), then it is likely that we may be able to duplicate the gate with only a small additional cost. So for such gates, we take the delay to be 1 when computing the arrival time.

VII. Experimental Results

In this section, we present the experimental results obtained from our ILP formulations and heuristic FBS algorithms for both the general and path-balanced settings. All our experiments were run on a MacBook Pro M1 with 10 cores of CPU, 16 cores of GPU, and 32 GB of RAM.

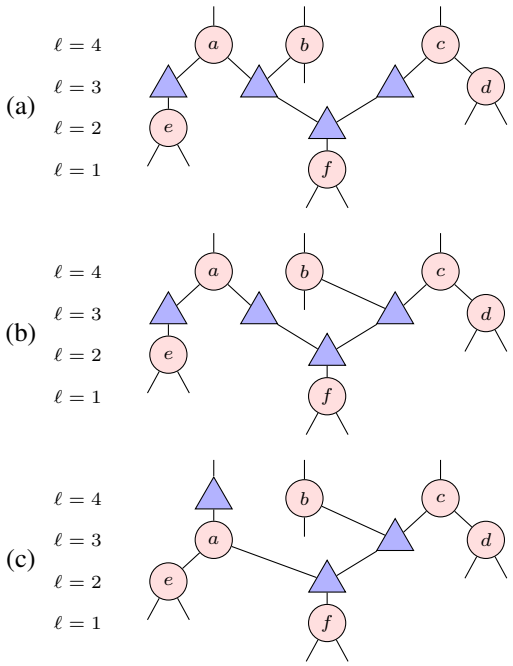

Fig. 5. Two possibilities for a part of an AQFP netlist and their retimed version.

Note that in all our experiments for the general FBS setting, the benchmarks are preprocessed with a single round of *resyn2* command in ABC [19], to do a fair comparison with prior work [12]. No such preprocessing was done in experiments for the path-balanced FBS setting.

A. Global Optimum for General Fanout-Bounded Synthesis

First, for a set of small benchmarks, we use the ILP to find the global optimum solutions; Namely, using the minimum possible circuit delay as the delay bound, we write the ILP introduced in Section IV, and then solve it using the Gurobi optimizer [36]. In the ILP formulation, we use the same setting as [12] where we have fanout capacity 2 and unit-area for both AND gates and buffers.

The results are shown in Table I where the first 8 benchmarks are from the EPFL logic synthesis benchmarks suite [14] and the rest of the benchmarks are a subset of those used in [15].

B. Heuristics for General Fanout-Bounded Synthesis

Next, we evaluate our top-down FBS approaches on the benchmarks of [15] and on EPFL benchmarks [14].

For benchmarks of [15], we present the results in Table II. As we see, our initial *top-down approach* already achieves the optimum on several benchmarks. Our *top-down approach with over-duplication* performs even better and achieves results that are optimum or closer to optimum on some additional benchmarks. We recall that both our approaches do not increase the number of logic levels of the input network (computed with no restrictions on the fanout capacity of gates).

For EPFL benchmarks, we present the results in Table III together with the results of [12] for a comparison. We remark

TABLE I
THE GLOBAL OPTIMUMS FOR GENERAL FANOUT-BOUNDED SYNTHESIS.

Benchmark	Input network		Output network				
	And gates	Levels	And gates	Buffers	Total gates (Area)	Levels	Time(s)
adder	1019	255	1021	126	1147	255	8538.26
bar	3141	12	3901	0	3901	12	242.97
cavlc	662	16	733	13	746	16	12.14
ctrl	108	8	123	3	126	8	0.24
dec	304	3	768	0	768	3	0.21
i2c	1162	15	1255	113	1368	15	59.27
int2float	214	15	224	7	231	15	2.76
router	177	19	180	5	185	19	1.52
adder1	7	4	7	0	7	4	0.01
adder8	77	17	78	7	85	17	0.37
mult8	439	35	447	13	460	35	1129.73
counter16	49	13	55	4	59	13	0.10
counter32	125	19	139	11	150	19	2.55
counter64	285	25	311	28	339	25	11.71
counter128	613	31	650	76	726	31	67.21
c17	6	3	6	0	6	3	0.03
c432	121	26	136	6	142	26	1.92
c499	387	18	410	42	452	18	8.15
c880	306	27	322	28	350	27	16.84
c1355	388	17	412	44	456	17	3.92
c1908	286	21	318	32	350	21	6.31
c2670	169	9	178	9	187	9	0.33
c3540	789	32	905	127	1032	32	3521.49
c5315	1294	26	1403	118	1521	26	553.95
c7552	1385	33	1562	192	1754	33	1335.10
sorter32	480	15	512	0	512	15	4.31
sorter48	984	25	984	64	1048	25	68.47

TABLE II
RESULTS OF THE TOP-DOWN FANOUT-BOUNDED SYNTHESIS ON BENCHMARKS OF [15].

Benchmark	Top-down approach			Top-down approach with over-duplication		
	And gates	Buffers	Total gates (Area)	And gates	Buffers	Total gates (Area)
adder1	7	0	7	7	0	7
adder8	77	8	85	77	8	85
mult8	441	19	460	441	19	460
counter16	52	7	59	52	7	59
counter32	130	20	150	130	20	150
counter64	298	41	339	298	41	339
counter128	638	88	726	638	88	726
c17	6	0	6	6	0	6
c432	132	13	145	134	9	143
c499	409	44	453	410	42	452
c880	306	47	353	306	47	353
c1355	412	44	456	414	42	456
c1908	314	44	358	314	44	358
c2670	172	18	190	172	18	190
c3540	819	256	1075	825	237	1062
c5315	1311	288	1599	1378	153	1531
c6288	1903	7	1910	1903	7	1910
c7552	1393	420	1813	1422	364	1786
sorter32	512	0	512	512	0	512
sorter48	984	64	1048	984	64	1048
alu32	1512	434	1946	1513	432	1945

that the measure of quality of results (QoR) used in [12] is slightly different, and if we were to use their QoR measure on our results, our approach would score even higher. Namely, the QoR measure used in [12] is $size(G)/size(G')+depth(G)/depth(G')$ where G is the original input network and G' is the fanout-bounded version produced by the algorithm. In our approach, the depths of G and G' are always equal, whereas in [12], $depth(G) \leq depth(G')$ with strict inequality for some benchmarks (e.g., see the results for benchmark "sqrt").

In our top-down approach without over-duplication, the average improvement over all standard EPFL benchmarks is 10.93%. However, for the benchmark "bar", our algorithm's result is 12.2% worse. Remarkably, combining the top-down algorithm with the over-duplication step from Section V-A achieves the same results as [12] for that benchmark, while increasing the average improvement over all EPFL benchmarks to 11.82%. Notably, our method results in fanout-bounded circuits that are much closer to the optimum results on several benchmarks (e.g., adder, cavlc, int2float, and router).

C. Global Optimum Splitter/Buffer Insertion for AQFP

In this section, we present the results of our ILP-based global optimization algorithm for the FBS in the path-balanced setting targeting the AQFP technology. To this end, we set $f_{\text{buff}} = 4$ and $f_{\text{gate}} = 1$ to capture the fanout constraints commonly used in prior work on the AQFP technology. We use the number of JJs as the area cost, and hence we set $c_{\text{gate}} = 6$ and $c_{\text{buff}} = 2$. Recall that, according to the ILP formulation, the *global optimum* means the minimum area for a fixed depth.

In our experiments, we consider two scenarios: one without gate duplications and one with gate duplications. To the best of our knowledge, *no* prior work on AQFP splitter/buffer insertion considers gate duplications.

In Table IV, we present our optimum results on the same benchmarks used by [15] for the case with no gate duplicates and compare them with the results of four prior work [15]–[18] in the same setting. In this experiment, we use the minimum achievable delay without duplicating gates as the target depth bound. In the table, the optimum area for the target depth is shown in blue. The term "opt" in the last columns means that the ILP solver was able to find the optimum solution. On the other hand, the term "tle" (time-limit-exceeded) means that the solver failed to find the optimum solution within a given time limit of 300 seconds, so the presented results for "tle" rows are based on a tentative feasible solution found by the solver. Note that having the global optimum results in this setting allows for an objective evaluation of other heuristic algorithms.

In Table V, we present the optimum results obtained considering *different target logic depths* on the same benchmarks for the setting with gate duplicates, which can be used to evaluate future algorithms in this setting. To obtain these results, we start with the minimum delay achievable without gate duplications as the target delay, and proceed with gradually decreasing the target delay. In the table, for each benchmark, the minimum observed is shown in blue where the ties are broken using the overall delay. Note that these results serve as a proof-of-concept that allowing gates duplications can help improve both the area and delay in AQFP synthesis.

TABLE III
RESULTS OF THE TOP-DOWN FANOUT-BOUNDED SYNTHESIS ALGORITHM ON EPFL BENCHMARKS.

	Input network		**Output of [12]**		**Output (top-down approach)**					**Output (top-down approach with over-duplication)**				
Benchmark	And gates	Levels	Total gates (Area)	Levels	And gates	Buffers	Total gates (Area)	Area Impr.%	Time (s)	And gates	Buffers	Total gates (Area)	Area Impr.%	Time (s)
adder	1019	255	1273	255	1020	128	1148	9.82	0.00	1020	128	1148	9.82	0.08
arbiter	11839	87	22911	87	11839	10176	22015	3.91	0.01	11839	10176	22015	3.91	0.04
bar	3141	12	3901	12	3425	952	4377	-12.20	0.00	3901	0	3901	0.00	0.01
cavlc	662	16	840	16	663	128	791	5.83	0.00	677	100	777	7.50	0.00
ctrl	108	8	147	8	108	26	134	8.84	0.00	114	14	128	12.93	0.00
dec	304	3	768	3	768	0	768	0.00	0.00	768	0	768	0.00	0.00
div	40772	4361	79413	4365	41087	12126	53213	32.99	0.04	41131	12038	53169	33.05	1.72
hyp	211330	24794	332744	24817	211458	45199	256657	22.87	0.20	212237	43641	255878	23.10	41.01
i2c	1162	15	1530	15	1162	264	1426	6.80	0.00	1171	247	1418	7.32	0.01
int2float	214	15	251	15	214	23	237	5.58	0.00	216	19	235	6.37	0.00
log2	29370	376	56617	376	29893	15018	44911	20.68	0.03	29857	15045	44902	20.69	1.08
max	2834	204	4157	206	3094	997	4091	1.59	0.00	3096	993	4089	1.64	0.09
mem_ctrl	45614	110	63788	110	45662	15326	60988	4.39	0.04	46140	14642	60782	4.71	2.18
multiplier	24556	262	31930	262	24567	7011	31578	1.10	0.02	24618	6909	31527	1.26	0.90
priority	676	203	795	203	676	59	735	7.55	0.00	676	59	735	7.55	0.05
router	177	19	222	19	177	8	185	16.67	0.00	177	8	185	16.67	0.00
sin	5039	177	10329	178	5415	2747	8162	20.98	0.01	5431	2677	8108	21.50	0.13
sqrt	19437	4968	32141	5449	20152	9432	29584	7.96	0.02	20152	9432	29584	7.96	0.65
square	16623	248	27556	248	16625	1533	18158	34.11	0.01	16720	1343	18063	34.45	1.44
voter	9756	57	13158	58	9810	1185	10995	16.44	0.01	9810	1185	10995	16.44	0.06
sixteen	11976864	99	24461292	99	11976864	9510308	21487172	12.16	23.61	12084231	9443891	21528122	11.99	527.31
twenty	15317374	86	31481612	86	15317374	12493285	27810659	11.66	29.84	15460597	12411371	27871968	11.47	520.78
twentythree	17168790	94	35358029	94	17168790	14056097	31224887	11.69	32.97	17316727	13968865	31285592	11.52	655.45
Average								10.93					11.82	

D. Heuristic Splitter/Buffer Insertion for AQFP

Finally, we run our scalable heuristic algorithm for path-balanced FBS on the same benchmarks used by [15] and compare our results with the latest scalable algorithm for AQFP splitter/buffer insertion [17] in Table VI. For all benchmarks, our approach achieves the same or significantly better delays as compared to the optimum delay achieved by the method in [17]. For some benchmarks with significant delay improvements, there is a considerable area overhead which is likely caused by duplicated gates. However, some other benchmarks with higher delay improvements show considerable area improvements as well, which can be attributed to the decrease in path-balancing buffers. The average delay improvement of our approach is 8.76% while the average area improvement is 0.5%. Notably, our heuristic algorithm achieves more than 17% delay improvements on several benchmarks.

TABLE IV
RESULTS OF AQFP SPLITTER/BUFFER INSERTION WITHOUT GATE DUPLICATION.

	Input network		**ICCAD'21 [15]**			**DAC'22 [16]**			**ASP-DAC'23 [17]**			**ASP-DAC'23 [18]**			**Global optimum**			
Benchmark	Area	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	opt/tle
adder1	7	4	16	74	8	16	74	8	16	74	8	-	-	-	16	74	8	opt
adder8	77	17	371	1204	33	371	1204	33	372	1206	33	-	-	-	371	1204	33	opt
mult8	439	35	1833	6300	70	1869	6372	71	1688	6010	70	1681	5996	70	1724	6082	70	tle
counter16	29	9	82	338	17	65	304	17	65	304	17	66	306	17	65	304	17	opt
counter32	82	13	189	912	23	155	802	23	154	800	23	156	804	23	154	800	23	opt
counter64	195	17	419	2134	30	352	1874	30	347	1864	30	351	1872	30	347	1864	30	opt
counter128	428	22	895	4652	38	760	4088	38	747	4062	38	755	4078	38	747	4062	38	opt
c17	6	3	12	60	5	12	60	5	12	60	5	-	-	-	12	60	5	opt
c432	121	26	837	2406	37	874	2474	38	839	2404	37	829	2384	37	829	2384	37	opt
c499	387	18	1251	4858	30	1275	4872	31	1173	4668	29	1173	4668	29	1173	4668	29	opt
c880	306	27	1723	5296	40	1703	5242	41	1511	4858	40	1536	4908	40	-	-	-	tle
c1355	389	18	1216	4784	29	1290	4914	31	1184	4702	29	1186	4706	29	1178	4690	29	opt
c1908	289	21	1505	4810	35	1298	4330	35	1236	4206	34	1253	4240	34	1232	4198	34	opt
c2670	368	21	2055	7392	27	2132	6472	30	1932	6072	28	1869	5954	28	1794	5796	28	opt
c3540	794	32	2395	9610	53	2266	9296	55	1972	8708	52	1963	8690	52	1926	8516	52	tle
c5315	1302	26	6447	20854	41	6026	19864	42	5646	19104	40	5505	18942	40	6260	20332	42	tle
c6288	1870	89	9297	29814	179	9893	31006	180	9009	29238	179	8832	28884	179	-	-	-	tle
c7552	1394	33	8342	25140	59	8759	25882	66	7505	23374	56	6768	21908	58	-	-	-	tle
sorter32	480	15	480	3840	30	480	3840	30	480	3840	30	-	-	-	480	3840	30	opt
sorter48	880	20	880	7040	35	880	7040	35	880	7040	35	-	-	-	880	7040	35	opt
alu32	1513	100	17178	43574	170	14655	38388	171	13837	36752	169	13976	37030	169	-	-	-	tle

TABLE V
RESULTS OF AQFP SPLITTER/BUFFER INSERTION WITH GATE DUPLICATION.

Benchmark	Gates	Buffers	#JJ	Levels	Time (s)	opt/tle
adder1	7	16	74	8	0.04	opt
adder1	8	13	74	7	0.03	opt
adder1	9	10	74	6	0.03	opt
adder8	82	352	1196	33	305.20	tle
adder8	81	347	1180	32	304.82	tle
adder8	81	337	1160	31	304.61	tle
adder8	81	328	1142	30	304.28	tle
adder8	81	320	1126	29	304.01	tle
counter16	29	65	304	17	21.47	opt
counter16	31	57	300	16	17.46	opt
counter16	32	53	298	15	6.48	opt
counter16	34	47	298	14	1.28	opt
counter16	36	45	306	13	0.95	opt
counter32	82	154	800	23	303.32	tle
counter32	84	149	802	22	303.12	tle
counter32	88	133	794	21	302.83	tle
counter32	92	121	794	20	302.58	tle
counter32	99	116	826	19	302.34	tle
counter64	195	347	1864	30	328.99	tle
counter64	198	335	1858	29	327.17	tle
counter64	203	323	1864	28	325.26	tle
counter64	209	299	1852	27	323.59	tle
counter64	225	284	1918	26	321.81	tle
counter128	431	742	4070	38	515.67	tle
counter128	439	730	4094	37	505.04	tle
counter128	440	706	4052	36	494.14	tle
counter128	455	662	4054	35	483.08	tle
counter128	458	651	4050	34	474.59	tle
c17	6	12	60	5	0.04	opt
c17	7	8	58	4	0.02	opt
c432	122	822	2376	37	316.69	tle
c432	122	793	2318	36	315.87	tle
c432	123	784	2306	35	314.97	tle
c432	130	775	2330	34	314.12	tle
c432	135	757	2324	33	313.41	tle
c499	396	1123	4622	29	386.41	tle
c499	398	1086	4560	28	380.61	tle
c499	400	1051	4502	27	375.08	tle
c499	403	1007	4432	26	369.49	tle
c499	444	962	4588	25	364.10	tle
c2670	370	1825	5870	28	398.12	tle
c2670	372	1793	5818	27	391.00	tle
c2670	378	1749	5766	26	384.23	tle
c2670	382	1699	5690	25	378.11	tle
sorter32	566	481	4358	30	435.07	tle
sorter32	534	450	4104	29	426.82	tle
sorter32	553	417	4152	28	420.74	tle
sorter32	576	384	4224	27	410.68	tle
sorter32	608	352	4352	26	402.67	tle
sorter48	896	896	7168	35	913.68	tle
sorter48	896	848	7072	34	875.92	tle
sorter48	944	816	7296	33	841.85	tle
sorter48	960	756	7272	32	807.86	tle
sorter48	1008	704	7456	31	778.18	tle

TABLE VI
RESULTS OF SCALABLE HEURISTIC APPROACH FOR AQFP.

Benchmark	Gates	Buffers	#JJ	Levels	Time (s)	Area Impr. %	Delay Impr.%
adder1	9	10	74	6	0	0.00	25.00
adder8	84	285	1074	26	0.01	10.95	21.21
mult8	469	1543	5900	58	0.06	1.83	17.14
counter16	29	65	304	17	0	0.00	0.00
counter32	82	154	800	23	0.01	0.00	0.00
counter64	195	347	1864	30	0.01	0.00	0.00
counter128	428	747	4062	38	0.03	0.00	0.00
c17	6	12	60	5	0	0.00	0.00
c432	139	821	2476	36	0.02	-3.00	2.70
c499	391	1131	4608	28	0.04	1.29	3.45
c880	306	1545	4926	40	0.08	-1.40	0.00
c1355	392	1151	4654	28	0.03	1.02	3.45
c1908	293	1176	4110	32	0.05	2.28	5.88
c2670	371	2172	6570	27	0.09	-8.20	3.57
c3540	797	1930	8642	50	0.15	0.76	3.85
c5315	1302	5710	19232	40	0.23	-0.67	0.00
c6288	1908	8468	28384	163	0.31	2.92	8.94
c7552	1400	9163	26726	54	0.61	-14.34	3.57
sorter32	704	256	4736	23	0.01	-23.33	23.33
sorter48	912	816	7104	33	0.04	-0.91	5.71
alu32	1543	11459	32176	139	0.65	12.45	17.75
Weighted average compared to [17]						0.51	8.76

VIII. CONCLUSION

In this work, we took a rigorous approach for the FBS of circuits in the unit-delay model. To this end, we formulated the problem of FBS for fixed target delay as an ILP and obtained the global optimum solutions for a number of benchmarks. We then showed how to find a feasible solution to the ILP using a scalable top-down approach while mitigating some shortcomings of earlier work. As compared to the known best results for this problem, our algorithm produced an 11.82% improved area while achieving matching or better delays.

As we see in Section VII, the over-duplication heuristic with a local cost function improved the area reduction. It will be interesting to find a more elaborate but efficiently computable cost function for evaluating heuristic choices such as the one we introduced in Section V-A. We also believe that a deeper analysis of the benchmark "bar" might hint at what kind of real-world circuit patterns benefit more from such heuristics.

We extended both our optimum and heuristic approaches to the setting with path-balancing constraints and demonstrated their effectiveness considering the splitter/buffer insertion problem in the AQFP technology. Our globally optimum results considering different target depths for the setting with gate duplications show that there exists a large gap in existing AQFP splitter/buffer insertion techniques. Remarkably, our scalable heuristic algorithm for this setting was able to exploit many optimization opportunities by considering the duplication of gates on critical paths. In particular, several benchmarks showed over 17% delay improvements under our method including two benchmarks (adder8, alu32) that also showed over 10% area improvements. However, comparing the results of our heuristic with the globally optimum solutions, it is clear that there are many opportunities for further improvements.

Considering these promising findings, we envision that FBS will have a bigger role to play in logic synthesis for emerging technologies with unconventional design constraints, and we hope that our work will motivate more research in this direction that would ultimately lead to better heuristics. The globally optimum solutions presented in this work can serve as the ground truth for evaluating such heuristics.

REFERENCES

[1] R. Murgai, "On the Global Fanout Optimization Problem," in *Int'l Conf. on Computer-Aided Design*, 1999, p. 511–515.

[2] A. Srivastava, R. Kastner, and M. Sarrafzadeh, "Timing driven gate duplication: complexity issues and algorithms," in *Int'l Conf. on Computer-Aided Design*, 2000, pp. 447–450.

[3] D. Baneres, J. Cortadella, and M. Kishinevsky, "Layout-Aware Gate Duplication and Buffer Insertion," in *Design, Automation & Test in Europe Conf. & Exhibit.*, 2007, pp. 1–6.

[4] Z. Li, D. A. Papa, C. J. Alpert, S. Hu, W. Shi, C. Sze, and Y. Zhou, "Ultra-Fast Interconnect Driven Cell Cloning for Minimizing Critical Path Delay," in *Proc. 19th Int'l Symposium on Physical Design*, New York, NY, USA, 2010, p. 75–82.

[5] D. A. Papa, I. L. Markov, D. A. Papa, and I. L. Markov, "Physically-driven logic restructuring," *Multi-Objective Optimization in Physical Synthesis of Integrated Circuits*, pp. 83–103, 2013.

[6] J.-L. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *Int'l Conf. on Computer-Aided Design*, 2005, pp. 575–581.

[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, 2013.

[8] A. L. Braun and D. C. Harms, "RQL majority gates, and gates, and or gates," Sep. 25 2018, US Patent 10,084,454.

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

[10] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum Cellular Automata," *Nanotechnology*, vol. 4, no. 1, p. 49, 1993.

[11] V. Calayir, D. E. Nikonov, S. Manipatruni, and I. A. Young, "Static and Clocked Spintronic Circuit Design and Simulation With Performance Analysis Relative to CMOS," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, pp. 393–406, 2014.

[12] H.-T. Zhang and J.-H. R. Jiang, "SFO: A Scalable Approach to Fanout-Bounded Logic Synthesis for Emerging Technologies," in *Design Automation Conference*, 2020, pp. 1–6.

[13] D. S. Marakkalage and G. De Micheli, "Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach," in *Design, Automation & Test in Europe Conf. & Exhibit.*, 2023, pp. 1–6.

[14] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," in *Proc. IWLS*, 2015.

[15] C.-Y. Huang, Y.-C. Chang, M.-J. Tsai, and T.-Y. Ho, "An Optimal Algorithm for Splitter and Buffer Insertion in Adiabatic Quantum-Flux-Parametron Circuits," in *Int'l Conf. on Computer-Aided Design*, 2021, p. 1–8.

[16] S.-Y. Lee, H. Riener, and G. De Micheli, "Beyond Local Optimality of Buffer and Splitter Insertion for AQFP Circuits," in *Design Automation Conference*, 2022, p. 445–450.

[17] A. T. Calvino and G. De Micheli, "Depth-Optimal Buffer and Splitter Insertion and Optimization in AQFP Circuits," in *Asia and South Pacific Design Automation Conference*, 2023, p. 152–158.

[18] R. Fu, M. Wang, Y. Kan, N. Yoshikawa, T.-Y. Ho, and O. Chen, "A Global Optimization Algorithm for Buffer and Splitter Insertion in Adiabatic Quantum-Flux-Parametron Circuits," in *Asia and South Pacific Design Automation Conference*, 2023, p. 769–774.

[19] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*, 2010, pp. 24–40.

[20] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, and D. S. Marakkalage, "The EPFL logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

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

[22] H. S. Miller and R. O. Winder, "Majority-Logic Synthesis by Geometric Methods," *IRE Trans. on Electronic Computers*, vol. EC-11, no. 1, pp. 89–90, 1962.

[23] S. Amarel, G. Cooke, and R. O. Winder, "Majority Gate Networks," *IEEE Trans. on Electronic Computers*, vol. EC-13, no. 1, pp. 4–13, 1964.

[24] 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.

[25] L. Amarù, P. Gaillardon, and G. De Micheli, "Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization," in *Design Automation Conference*, 2014, pp. 1–6.

[26] G. L. Smith, R. J. Bahnsen, and H. Halliwell, "Boolean comparison of hardware and flowcharts," *IBM Journal of Research and Development*, vol. 26, no. 1, pp. 106–116, 1982.

[27] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, "Energy-efficient superconducting computing—Power budgets and requirements," *IEEE Trans. on Applied Superconductivity*, vol. 23, no. 3, 2013.

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

[29] 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 *ACM Great Lakes Symposium on VLSI*, 2019, pp. 189–194.

[30] 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 *Int'l Conf. on Computer Design*, 2019, pp. 429–436.

[31] R. Saito, C. L. Ayala, and N. Yoshikawa, "Buffer Reduction Via N-Phase Clocking in Adiabatic Quantum-Flux-Parametron Benchmark Circuits," *IEEE Trans. on Applied Superconductivity*, vol. 31, no. 6, pp. 1–8, 2021.

[32] 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 Trans. on Applied Superconductivity*, vol. 31, no. 5, pp. 1–5, 2021.

[33] H. J. Hoover, M. M. Klawe, and N. J. Pippenger, "Bounding fan-out in logical networks," *Journal of the ACM (JACM)*, vol. 31, no. 1, pp. 13–18, 1984.

[34] M. Golumbic, "Combinatorial Merging," *IEEE Trans. on Computers*, vol. 25, no. 11, pp. 1164–1167, 1976.

[35] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[36] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022. [Online]. Available: https://www.gurobi.com

Dewmini Sudara Marakkalage is a Ph.D. student at the Integrated Systems Laboratory, EPFL, Lausanne, Switzerland. She received a B.Sc. in Engineering from the Department of Electronic and Telecommunication Engineering, University of Moratuwa, Sri Lanka, in 2016 and a M.Sc. in Computer Science from the School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland, in 2020. Her research interests include logic synthesis and design automation for emerging technologies.

Giovanni De Micheli is a research scientist in electronics and computer science. He is credited for the invention of the Network on Chip design automation paradigm and for the creation of algorithms and design tools for Electronic Design Automation (EDA). He is a Professor and Director of the Integrated Systems Laboratory at EPFL Lausanne, Switzerland.

Prof. De Micheli is a Fellow of ACM, AAAS, and IEEE, a member of the Academia Europaea, and an International Honorary member of the American Academy of Arts and Sciences. His current research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies. He is the author of Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994, co-author and/or co-editor of ten other books, and of over 900 technical publications. His citation h-index is above 100 according to Google Scholar. He is a member of the Scientific Advisory Board of IMEC (Leuven, Belgium) and STMicroelectronics.

Prof. De Micheli is the recipient of the 2022 ESDA-IEEE/CEDA Phil Kaufman Award, the 2019 ACM/SIGDA Pioneering Achievement Award, and several other awards.