Fanout-Bounded Logic Synthesis for Emerging Technologies - Top-Down pproach

Dewmini Sudara Marakkalage, Giovanni De Micheli

Integrated Systems Laboratory, EPFL, Lausanne, Switzerland

***bstract*—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. In this work, we study the problem of re-synthesizing a given logic network through buffer insertions and gate duplications such that 1) the logic depth meets a predefined bound, 2) each node meets given fanout constraints, and 3) the total area is minimized. We first formulate this problem as an integer linear program (ILP) and present exact solutions for small logic networks. For large networks, we show how to construct a feasible solution for the ILP efficiently using a top-down approach; Namely, for the fanout net of each node in the reverse topological order, we construct a collection of buffer trees where each tree is rooted at a different copy of the node. To minimize gate duplication, we minimize the number of buffer trees such that the critical path length is not increased. Noting that minimizing the number of trees for the fanout net of a node can increase the costs of fanout nets of 's fanins, we strengthen the local optimization by considering different numbers of buffer trees for each node and looking ahead to one logic level below to identify which choice is best. When using the minimum depth achievable with unbounded fanouts as the final depth bound, the proposed approach achieves 11.82% better area as compared to the known best prior results on EPFL benchmarks.**

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

I. Introduction

In digital electronics, the ability to have multiple fanouts per gate enables compact implementations of complex logic functions. However, increasing the number of fanouts of a gate can adversely affect delay performance, and the maximum number of fanouts a gate can support is usually bounded. Thus it is important to develop synthesis algorithms to effectively utilize fanouts.

In the conventional CMOS technology, fanout optimization has been well-studied, both as a method to improve 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 is not generally transferable to many emerging technologies such as superconducting electronics (e.g., QFP [7], RQL [8], RSFQ [9]) and spintronics [10], which generally have tight, explicit fanout bounds and/or significantly different timing models (clocked gates, for example). Thus the allowed transformations in such technologies can be fundamentally different. For example, in CMOS, the delay increase due to a high number of fanouts can be somewhat countered with techniques such as transistor sizing, which is not an option for post CMOS technologies. Instead, when designing for emerging technologies, fanout bounded synthesis is usually considered early in the design process (e.g., in the logic synthesis stage) using a combination of gate-duplications and buffer insertions. Notably, in superconducting electronic technologies, splitters are needed to drive multiple fanouts. However, we can model splitters as buffers with fanout capacity at least two, thus encompassing such scenarios under generic fanout bounded synthesis considered in this work.

In this work, we consider the following fanout bounded synthesis problem: Given an input logic network, a bound D on the number of logic levels, 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 1) the total number of logic levels is at most D, 2) each node in the synthesized network meets the respective fanout bounds, and 3) the total area is minimized.

n early theoretical work on fanout bounded synthesis using gate-duplications and buffers by Hoover et al. [11] presented an algorithm that limits the fanouts 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. Recently, Zhang and Jiang [12] revisited the problem of fanout bounded synthesis, specifically targetting emerging technologies. Their work combines several heuristics to obtain a non-trivial algorithm for fanout bounded synthesis in the unit delay model. (Unit delay model is an apt timing model for technologies such as adiabatic quantum-flux-parametron (QFP) which have clocked gates [13].)

To elaborate, the algorithm of [12] first computes the amount of duplicates needed for each gate using a recursive evaluation procedure; the number of duplicates for a gate is incremented if it results in an overall buffer reduction without significantly affecting the delay. Next, for each node in the reverse topological order, "skewed" buffer trees are constructed using an algorithm similar to that of [14]. Finally, for nodes that are equivalent, 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 unnecessary nodes from the collection of duplicates.

However, we identify several opportunities for improving this approach:

1) The computed gate copy-counts does not guarantee that the fanout bounded version achieves the same minimum possible delay as the original, fanout unbounded network.

2) Secondly, the algorithm proposed in [12] for skewed buffer tree construction, which uses a priority queue similarly to the well known Huffman coding algorithm [15], is guaranteed to achieve the best possible level for the root node of the tree *only* when fanout bound is 2. To achieve the optimal level for the root node, the same algorithm can be used with a different initialization of the priority queue as shown by Golumbic [16].
3) It is not stated how the fanout nodes are assigned to the duplicated copies before the skewed buffer trees are constructed. Moreover, in the process for evaluating whether to duplicate a gate or not, when checking if there is an impact on the delay, it is not specified at which levels the copies of a gate are placed. We note that there are situations where the effect on the overall delay is mitigated if the copies can be placed at different levels, as compared to placing all copies at the same level. But such decisions on levels of the copies of a gate can be more effectively made if we know the levels of the fanouts beforehand.
4) 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 *after* fixing the levels of the duplicated nodes.

In this work, we mitigate the above shortcomings by taking a more rigorous approach. Namely, we first formulate the minimum area fanout bounded synthesis problem for a given circuit delay D as an integer linear program (ILP) and solve it for small logic networks with relatively few logic levels to find the optimum area.. We then present a top-down approach to find a feasible (though not necessarily optimal) solution to the ILP together with an algorithm to construct a fanout-bounded logic network from any feasible solution to this ILP. To elaborate, for each gate n in the input network and for each possible level $1 \leq \ell \leq D$ in the output network, we use the number of duplicates of gate n that are in level ℓ and the number of buffers associated with gate n that are in level ℓ as our ILP variables. We then add constraints relating the total number of available and required fanouts by each logic level, which must be satisfied by any valid fanout bounded circuit. Our top-down approach can be viewed as considering nodes in the input network in the reverse topological order and constructing a buffer forest for each node without increasing the overall circuit delay. The number of trees in the constructed buffer forest for a given node n determines the number of copies of n in the output network.

The proposed top-down approach achieves $\sim$10.9% better area in comparison to [12] on the same EPFL benchmarks while a slightly more improved version of our algorithm yields $\sim$11.8% better area. We remark that for all benchmarks, our approach achieves matching or better delays as compared to [12] since we never increase the critical path delay. Our approach is versatile, and can be used on any graph representation of logic. In this work, we use and-inverter graphs (IGs) as the preferred logic representation in order to perform a fair comparison with [12].

The rest of the paper is organised as follows: In Section II, we summarize some concepts useful to better understand our work. In Section III, we discuss the ILP formulation in detail and also present our top-down algorithm for fanout bounded synthesis. Next, in Section IV, we present our experimental results, and finally, in Section V, we conclude with a brief discussion on the results and possible future directions.

II. B CKGROUND

In this section, we give background on and-inverter graphs (IGs), static timing analysis with the unit delay model, and node equivalence.

. *nd-Inverter Graphs*

The and-inverter graph (IG) is a directed acyclic graph (D G) representation of logic where nodes represent either primary inputs (which have in-degree zero) or 2-input ND gates (which have in-degree 2). IGs have two possible types of directed edges, representing non-inverted or inverted fanins. The IG is a universal representation, meaning that an IG can represent an arbitrary logic function, and is supported by many logic synthesis tools and libraries such as BC [17] and mockturtle [18] owing to its simplicity and wider compatibility with many logic synthesis algorithms. t the same time, IGs support efficient structural hashing which enables efficient collapsing of logically equivalent nodes.

B. Static Timing nalysis

In this work, we use the unit delay model which assumes that a signal incurs a 1-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, also called the depth of the circuit, is defined as the maximum arrival time of any primary output.

For a given target delay D, the required time of a node n, which we denote by t_n^{req} 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.

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, it can be computationally very expensive to identify all nodes that are equivalent to a given node. 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. n example of this type of weaker equivalence checking is structural hashing; For IGs, 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 assume the IG data-structure internally uses structural hashing to collapse any equivalent nodes. However, for the output circuit, the algorithm may need some explicitly duplicated gates, thus we disable structural hashing for the output network.

III. Method

In this section, we first present our ILP formulation of fanout bounded synthesis. We then present a top-down heuristic algorithm to greedily find a feasible solution to the derived ILP.

. *Fanout-Bounded Synthesis ILP Formulation*

We formulate the fanout bounded synthesis with a predefined depth bound D as an ILP. Namely, in our ILP, we aim to minimize the total number of gates and buffers subject to the constraint that all input-to-output path lengths are bounded by D while all gates and buffers meet the given fanout bounds. 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 original network.

To derive the ILP, we start with the following notation: Let P be the set of all primary inputs of the input network, let G be the set of all gates, and let $N = P \cup G$ be the set of all nodes. For example, in the example network shown in Figure 1, $P = \{p_1, \quad, p_4\}, G = \{n_1, n_2, \quad, n_7\}$ and $N = \{p_1, \quad, p_4, n_1, \quad, n_7\}$.[1] 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 Figure 1, we have $\mathrm{FO}(p_3) = \{n_2, n_3\}$ and $\mathrm{FO}(n_1) = \{n_3, n_4\}$, and $k_{n_2} = k_{n_4} = k_{n_5} = k_{n_6} = k_{n_7} = 1$.

Let c_{gate} be the 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_{bu} be the area of a buffer, let f_{gate} be the fanout capacity of a gate, and let f_{bu} be the fanout capacity of a buffer. For example, consider the QFP technology. In this technology, whenever a gate has more than one fanouts, splitters must be used, and a splitter can be modelled as a buffer that support multiple fanouts (usually three or four [19], [20]). In QFP technology, the area is measured in terms of the number of Josephson Junctions (JJs), and a gate (nd, Or, or Majority-3) needs 6 JJs while a splitter needs only 2 JJs [19]. Thus we can set $c_{\mathrm{gate}} = 6$, $c_{\mathrm{bu}} = 2$, $f_{\mathrm{gate}} = 1$ and $f_{\mathrm{bu}} = 4$ for this case. We remark that, similar to other superconducting technologies, QFP circuits also needs all input-to-output paths to be of same length, which is achieved by inserting buffers of fanout one on unbalanced paths. This step can either be performed separately after fanout bounded synthesis or it can be integrated into the algorithm described in this work.

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 circuit.
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.

[1]Note that, in the logic networks depicted in this work, the primary inputs are at the bottom and the primary outputs are at the top.

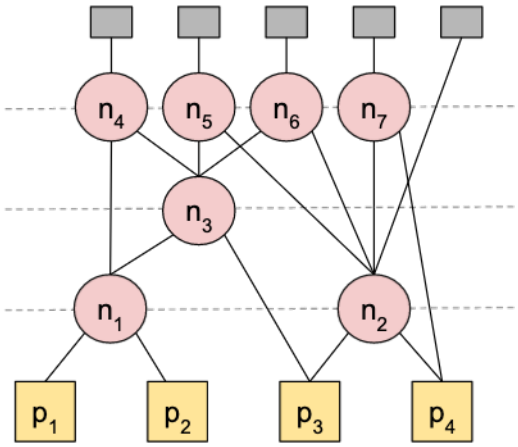

Fig. 1: Example logic network with node labels.

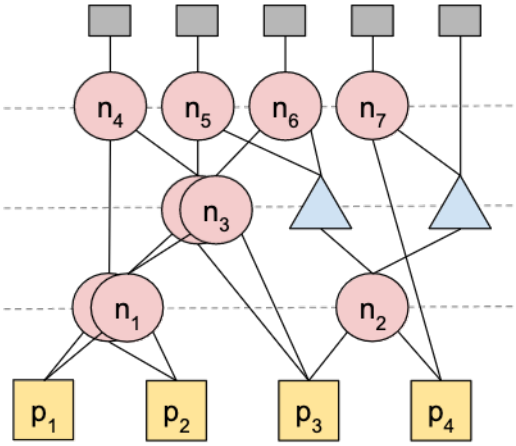

Fig. 2: possible fanout bounded version of the logic network shown in Figure 1.

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. ccording this definition, in the example network shown in Figure 2 (which shows a fanout bounded version of the sample network of Figure 1 where $f_{\text{gate}} = f_{\text{bu}} = 2$), there are two n_1-equivalent gates and two n_2-equivalent gates (represented by overlapping circles). Moreover, the two buffers represented as blue triangles in level 2 are n_2-equivalent.

Variables

We use two kinds of integer variables. For each node $n \in N$ and for each level $\ell \in \{1, \quad, D\}$, 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 Figure 2, the introduced variables take the following values: $g_{n\ 1} = 2, g_{n_2\,1} = 1, g_{n_3\,2} = 2, g_{n_4\,3} = 1, g_{n_5\,3} = 1, g_{n_6\,3} = 1, g_{n_7\,3} = 1, b_{n_2\,2} = 2$ and $g_{n\,\ell} = 0$ for all unspecified variables $g_{n\,\ell}$ where $n = 1, \quad, 7$ and $\ell = 0, \quad, 3$.

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 $L \in \{1, \quad, D\}$ and a fixed gate $n \in N$. We denote by $\text{avl}(n, L)$, 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

$$\text{avl}(n, L) = \sum_{\ell=0}^{L\ \ 1} (f_{\text{bu}} \quad \cdot b_{n\,\ell} + f_{\text{gate}} \cdot g_{n\,\ell}),$$

which is a linear function of the ILP variables. We denote by $\text{req}(n, L)$, 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

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

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 $\text{avl}(n, L) \geq \text{req}(n, L)$ for all $n \in G$ and $L \in 1, \quad, D$. To see this, consider any valid depth-D fanout bounded version of the input network. Let $g_{n\,L}, b_{n\,L}$ be the corresponding ILP variable values. Fix any gate $n \in G$ and let $L = 1$. Note that for any gate $m \in \text{FO}(n)$, $g_{m\,1}$ 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 $\text{avl}(n, 1) = 0 \geq 0 = \text{req}(n, 1)$. Now, suppose that $\text{avl}(n, L) \geq \text{req}(n, L)$ must hold for any valid depth-D fanout bounded version. We inductively show that $\text{avl}(n, L+1) \geq \text{req}(n, L+1)$ 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 $L+1$ is at least $\sum_{m \in \text{FO}(n)} g_{n\,L+1} + b_{n\,L+1}$. The total remaining capacity of n-equivalent gates/buffers that are at levels below L is $\text{avl}(n, L) \quad \text{req}(n, L)$. 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

$$f_{\text{gate}} \cdot g_{n\,L} + f_{\text{bu}} \quad \cdot b_{n\,L} \geq \sum_{m \in \text{FO}(n)} g_{n\,L+1} + b_{n\,L+1} \quad (\text{avl}(n, L) \quad \text{req}(n, L)),$$

which yields

$$\text{avl}(n, L) + f_{\text{gate}} \cdot g_{n\,L} + f_{\text{bu}} \quad \cdot b_{n\,L} \geq \text{req}(n, L) + \sum_{m \in \text{FO}(n)} g_{n\,L+1} + b_{n\,L+1},$$

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

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

Minimize $\sum_{n \in G} \sum_{\ell=1}^{D} (c_{\text{gate}} \cdot g_{n\,\ell} + c_{\text{bu}} \quad \cdot b_{n\,\ell})$,
Subject to

$$\begin{aligned}
\text{avl}(n, L) \quad \text{req}(n, L) &\geq 0 && \forall\, n \in N, 1 \leq L \leq D, \\
\text{avl}(n, D+1) \quad \text{req}(n, D) &\geq k_n && \forall\, n \in N, \\
g_{n\,0} &= 0 && n \in G, \\
b_{n\,0} &= 0 && n \in N, \\
g_{n\,L}, b_{n\,L} &\in && \forall\, n \in N, 1 \leq L \leq D
\end{aligned}$$

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

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

input : Input network ntk, parameters $f_{\text{gate}}, f_{\text{bu}}$, and a feasible ILP solution $g_{n\ L}, b_{n\ L}$ for $n \in N$ and $0 \leq L \leq D$.
output: fanout bounded network that is logically equivalent to ntk.

Let $newsig$ be a map from nodes in ntk to a queue of pairs (new node, remaining capacity)
for *all* $p \in ntk\ PIs$ **do**
  $newsig[p] \leftarrow$ newntk.create_pi()
Let $data$ be an empty list.
for *all nonzero* $g_{n\ L}$ **do** dd $(L, n,$ "$gate$ $)$ to $data$
for *all nonzero* $b_{n\ L}$ **do** dd $(L, n,$ "$buff$ $)$ to $data$
Sort $data$ in the ascending order of levels.
for *all* $(\ell, m, t) \in data$ *in the ascending order of levels* **do**
  if t = *"gate"* **then**
    Look up fanins of m in $newsig$.
    newgate $\leftarrow$ Create a new gate by choosing the first available node from the corresponding queues in $newsig$ as fanin.
    Decrement the remaining capacity for used fanin nodes and remove them from the queue if the remaining capacity reach zero.
    $newsig[m]\ push((newgate, f_{\text{gate}}))$
  else
    Look up m in $newsig$.
    newbuff $\leftarrow$ Create a new buffer by choosing the first available node from the corresponding queue as the fanin.
    Decrement the remaining capacity for the used fanin.
    $newsig[m]\ push((newbuff, f_{\text{bu}}\ )$
return the constructed network.

lgorithm 2: lgorithm for determining $g_{n\ L}$ and $b_{n\ L}$ 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 $f_{\text{gate}}, f_{\text{bu}}$, a node n, t_n^{arr}, and a list $folev_n$ of levels of n's fanouts.
output: Values of $g_{n\ L}, b_{n\ L}$ variables for $L = 1,\quad , D$.

Set $g_{n\ L}, b_{n\ L} = 0$ for all L
for t = *1 to* $length(folev_n)$ **do**
  Let $rem \leftarrow length(folev_n)\quad t \cdot f_{\text{gate}}$
  if *rem* ≤ 0 **then**
    for i = *1 to* $length(folev_n)$ *in steps of* f_{gate} **do**
      Increment $g_{n\ folev\ [i]\ \ 1}$.
      return variable values
  LastBufLoad $\leftarrow rem\ \ \text{mod}\ (f_{\text{bu}}\quad 1)$
  if *LastbufLoad* > 0 **then**
    dd $f_{\text{bu}}\quad LastBufLoad$ 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 forests.
    if *ll roots of the trees in the forest are at least* t_n^{arr} **then**
      Determine $g_{n\ L}$ and $b_{n\ L}$ using the constructed buffer forest
      return variable values

algorithm 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 optimal 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 that order, construct the $g_{n\ \ell}$ gate copies or $b_{n\ \ell}$ buffers in a new network. To facilitate this constructions, 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 constructs gates/buffers in a level-by-level fashion using a feasible variable assignment, we can see that the corresponding queues of n-equivalent gates/buffers do not prematurely get empty due to the availability-requirement constraints of an ILP. The pseudo-code of this algorithm is presented in lgorithm 1.

Remark: Recall that, in technologies such as QFP, in addition to the fanout constraints, there is an additional requirement that for each gate, its inputs must arrive at the same time. In our ILP formulation, this is easy to ensure; we simply re-define the quantities $\text{avl}(n, L)$ and $\text{req}(n, L)$ to be, respectively, the available number of fanouts by n-equivalent gates and buffers in exactly level $L\quad 1$ and the required number n-equivalent fanouts for gates and buffers in exactly level L.

B. Top-Down Fanout-Bounded Synthesis

lthough solving the ILP introduced in Section III- gives the optimal solution, solving it optimally for large networks which we often encounter in practice is a prohibitively expensive computation, and hence not a viable approach. Motivated by this, we now show how to find a feasible, but not necessarily the optimal solution to the ILP using a *top-down* approach.

Namely, we consider the gates $n \in G$ in the reverse topological order, and for each n in this order, determine values for variables $g_{n\ L}$ and $b_{n\ L}$ such that the constraints $\text{avl}(n, L)\quad \text{req}(n, L) \geq 0$ and $\text{avl}(n, D{+}1)\quad \text{req}(n, D) \geq k_n$ are satisfied. Note that by considering 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 $g_{n\,L}$ and $b_{n\,L}$, we prefer minimizing the number of gate copies, and utilize 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 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 a 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 lgorithm 2 to determine the values of $g_{n\,L}$ and $b_{n\,L}$ variables. Considering each node in the reverse topological order, we thus use lgorithm 2 to determine values of $g_{n\,\ell}$ and $b_{n\,\ell}$ for all gates $n \in G$, and then use lgorithm 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, by re-wiring the buffers which may or may not render some n-equivalent gates redundant. In contrast, our algorithm uses lgorithm 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 $f_{\text{gate}}, f_{\text{bu}} > 2$ and $c_{\text{gate}} > c_{\text{bu}}$. In contrast, 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 slightly modified version of the algorithm of Golumbic [16] to derive r buffer trees, and find the minimum value of r such that roots of all trees meet the arrival time requirement.

C. Improving the Top-Down pproach by llowing Over-Duplication

In the previous section, we explained how to find the smallest buffer forest that does not increase circuit delay for a given fanout net. The intuition behind settling for the smallest buffer forest is to minimize gate duplication in order to avoid increasing the load of their fanins. One potential drawback of not duplicating more than what is absolutely necessary to meet the critical path delay constraint is the following: We end up placing some node n at level t_n^{arr} although we may have the option of placing two copies of n at level $t_n^{\text{arr}} + 1$ instead, thus forcing more duplication for n's fanin nodes as their fanout nets does not have enough slack to add buffers. s such, allowing more duplicates than absolutely necessary, which we call "over-duplication", can be still good if the increased load to fanins does not increase their buffer requirement, while increasing the room to add buffers in the fanout nets of those fanins. In contrast to our approach, in [12], the decision of duplication is made ignoring the final levels of subsequent gates, and hence it is not guaranteed to retain the minimum possible circuit delay.

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 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.

IV. Experiment l Results

In this section, we present the experimental results obtained from our ILP formulation and the top-down fanout bounded synthesis algorithm. We compare the result of our top-down approach with the results of [12].

T BLE I: Exact fanout-bounded synthesis results using ILP from Section III- solved with Gurobi optimizer [21].

Benchmark	Input network		Output network				
	nd gates	Levels	nd gates	Buffers	Total gates (rea)	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

First, for a set of small benchmarks, we use the ILP to find the exact solutions; Using the minimum possible circuit delay as the delay bound, we write the ILP introduced in Section III- , and solve it using Gurobi optimizer [21] using an academic product license. In the ILP formulation, we use the same setting as [12] where we have a fanout bound of 2 and unit-area ND gates and buffers. The experiment was run on a MacBook Pro M1 with 10 cores of CPU, 16 cores of GPU, and 32 GB of R M. The results are shown in Table I. The first 8 benchmarks are from the EPFL logic synthesis benchmarks suite [22] whereas the rest of the benchmarks are a subset of those used in [23].

Next, we present the results (in Table II) of our top-down algorithm on the full set of EPFL benchmarks, together with the results of [12] for a comparison. Similarly to [12], our algorithm was also run on top of the resulting circuits after one round of `r syn2` command in BC. We remark 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").

Note that, in our top-down approach (without over-duplication), the average improvement over all standard EPFL benchmarks is 10 93%. However, for benchmark "bar", our algorithm's result is 12 2% worse. Remarkably, combining the top-down algorithm with over-duplication step from Section III-C 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., on benchmarks adder, cavlc, int2float, and router).

s per the running time, both our top-down algorithms can be implemented to run in $O(n \log n)$ time where n is the size of the input network, and hence it scales well to large networks. The over-duplication version is only a constant factor slower (recall that we restrict over-duplication to nodes with a constant number of fanouts) than the naive top-down version due to re-computation of costs in the duplication step.

V. Conclusion

In this work, we take a rigorous approach for fanout bounded synthesis of circuits in the unit-delay model. To this end, we formulate the problem of fanout bounded synthesis for fixed target delay as an ILP, and we showed how to find a feasible solution to the ILP using a top-down approach while mitigating some shortcomings of earlier work. s compared to the known best results for this problem, our algorithm produces 11.82% improved area while achieve matching or better delays.

s we see from Section IV, the over-duplication heuristic with a local cost function improves 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 III-C. We also believe that a deeper analysis of benchmark "bar" might hint at what kind of real-world circuit patterns benefit more from such heuristics.

nother promising research direction is to investigate if the ILP can be relaxed to be a simple linear program on real variables, and then use the possibly fractional optimum solution to guide a fanout bounded synthesis algorithm. For example, the fractional values of $g_{n\ L}$ variables may be good approximations to the duplicate gate counts in the optimal integral solution. Or, the fractional values may be rounded to integral values (with some loss in the objective value) without violating the availability-requirement constraints of the ILP. If such a rounding is possible, it may also yield some provable guarantees for the quality of algorithm output. In any case, we hope that our ILP would serve as a theoretical basis for future research in fanout bounded synthesis for emerging technologies with hard fanout constraints.

CKNOWLEDGMENTS

This research was supported by the SNSF grant Supercool 200021_1920981.

T BLE II: Results of the top-down fanout bounded synthesis algorithm on EPFL benchmarks.

	Input network		**Output of [12]**		**Output (top-down)**					**Output (top-down with over-duplication)**				
Benchmark	nd gates	Levels	Total gates (rea)	Levels	nd gates	Buffers	Total gates (rea)	rea Impr.%	Time (s)	nd gates	Buffers	Total gates (rea)	rea 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
verage								10.93					11.82	

REFERENCES

[1] R. Murgai, "On the global fanout optimization problem," in *1999 IEEE/ CM International Conference on Computer- ided Design. Digest of Technical Papers (Cat. No.99CH37051)*, 1999, pp. 511–515.

[2] . Srivastava, R. Kastner, and M. Sarrafzadeh, "Timing driven gate duplication: complexity issues and algorithms," in *IEEE/ CM International Conference on Computer ided Design. ICC D - 2000. IEEE/ CM Digest of Technical Papers (Cat. No.00CH37140)*, 2000, pp. 447–450.

[3] D. Baneres, J. Cortadella, and M. Kishinevsky, "Layout-aware gate duplication and buffer insertion," in *2007 Design, utomation Test in Europe Conference Exhibition*, 2007, pp. 1–6.

[4] Z. Li, D. . Papa, C. J. lpert, S. Hu, W. Shi, C. Sze, and Y. Zhou, "Ultra-fast interconnect driven cell cloning for minimizing critical path delay," in *Proceedings of the 19th International Symposium on Physical Design*, ser. ISPD '10. New York, NY, US : ssociation for Computing Machinery, 2010, p. 75–82. [Online]. vailable: https://doi.org/10.1145/1735023.1735047

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

[6] J.-L. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *ICC D-2005. IEEE/ CM International Conference on Computer- ided Design, 2005.*, 2005, pp. 575–581.

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

[8] . 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 Transactions on pplied Superconductivity*, vol. 1, no. 1, pp. 3–28, 1991.

[10] V. Calayir, D. E. Nikonov, S. Manipatruni, and I. . Young, "Static and clocked spintronic circuit design and simulation with performance analysis relative to cmos," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, pp. 393–406, 2014.

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

[12] H.-T. Zhang and J.-H. R. Jiang, "Sfo: scalable approach to fanout-bounded logic synthesis for emerging technologies," in *2020 57th CM/IEEE Design utomation Conference (D C)*, 2020, pp. 1–6.

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

[14] . Mishchenko, R. Brayton, S. Jang, and V. Kravets, "Delay optimization using sop balancing," in *Proceedings of the International Conference on Computer- ided Design*, ser. ICC D '11. IEEE Press, 2011, p. 375–382.

[15] D. . Huffman, " method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[16] M. Golumbic, "Combinatorial merging," *IEEE Transactions on Computers*, vol. 25, no. 11, pp. 1164–1167, nov 1976.

[17] R. Brayton and . Mishchenko, " BC: n academic industrial-strength verification tool," in *Computer ided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.

[18] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, and G. De Micheli, "The EPFL logic synthesis libraries," Nov. 2019, arXiv:1805.05121v2.

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

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

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

[22] L. marù, P.-E. Gaillardon, and G. De Micheli, in *The EPFL Combinational Benchmark Suite*, 2015. [Online]. vailable: http://infoscience.epfl.ch/record/207551

[23] C.-Y. Huang, Y.-C. Chang, M.-J. Tsai, and T.-Y. Ho, " n optimal algorithm for splitter and buffer insertion in adiabatic quantum-flux-parametron circuits," in *2021 IEEE/ CM International Conference On Computer ided Design (ICC D)*. IEEE Press, 2021, p. 1–8. [Online]. vailable: https://doi.org/10.1109/ICC D51958.2021.9643456