Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach

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

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

***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. In this work, we study the problem of resynthesizing a logic network with bounded-fanout gates while minimizing area. We 1) formulate this problem for a fixed target logic depth as an integer linear program (ILP) and present exact solutions for small logic networks, and 2) propose a top-down approach to construct a feasible solution to the ILP which yields an efficient algorithm for fanout bounded synthesis. When using the minimum depth achievable with unbounded fanouts as the target logic depth, our top-down approach achieves 11.82% better area as compared to the state-of-the-art with matching or better delays.**

***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 deteriorate delay performance, and a gate can only support a bounded number of fanouts. 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 are not generally transferable to many emerging technologies such as superconducting electronics (e.g., AQFP [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 circuit transformations in such technologies can be fundamentally different due to lack of techniques analogous to transistor sizing, and hence fanout bounded synthesis is considered early in the design process for emerging technologies. Notably, in superconducting electronic technologies, splitters are needed to drive multiple fanouts. However, we can model splitters as buffers with a fanout capacity of at least two, thus encompassing such scenarios under generic fanout bounded synthesis considered in this work.

We consider the problem of resynthesizing a logic network to meet given fanout bounds by means of gate duplication and buffer insertions such that the total area is minimized, which we refer to as fanout bounded synthesis (FBS). An early theoretical work on FBS by Hoover et al. [11] presented an algorithm that achieves only a constant factor increase in both the total number of gates and the depth. Recently, Zhang and Jiang [12] revisited the problem of FBS targetting emerging technologies and used several heuristics to obtain a non-trivial algorithm for FBS in the unit delay model. (Unit delay model is an apt timing model for technologies with clocked gates such as adiabatic quantum-flux parametron (AQFP) [13].)

To elaborate, the algorithm of [12] first computes the number of duplicates for each gate using a recursive evaluation procedure to check if duplicating reduces buffers 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 redistributed. This step does not alter the levels of the nodes but may remove some unnecessary nodes from the collection of duplicates. However, the algorithm of [12] does not always achieve the same minimum possible delay as the original, fanout unbounded network. Moreover, the skewed buffer tree construction yields the locally optimal depth for the buffer tree only when the fanout bound is 2, but we notice that it can be generalized to higher fanout limits by initializing the priority queue as shown by Golumbic [15]. Furthermore, the algorithm does not specify how duplicated copies are assigned to the original fanouts; if not done properly, this can yield sub-optimal results. Finally, buffer-forest rebalancing does not always give the locally-optimal structure as it is performed only after fixing the levels of duplicated nodes.

In this work, we mitigate the above shortcomings by taking a rigorous approach. We first formulate the FBS problem for a target delay D as an integer linear program (ILP) and solve it for small logic networks 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. A simple version of the proposed top-down approach achieves $\sim$10.9% better area in comparison to [12] on the same EPFL benchmarks while an 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] as our algorithms

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

always achieve the same delay as the original fanout unbounded network. In this work, we use and-inverter graphs (AIGs) as the preferred logic representation in order to perform a fair comparison with [12], and we use the same assumption that primary inputs have no fanout bounds. However, our approach is generalizable to other graph representations and can be extended to support fanout bounds on primary inputs as well.

Organization of the paper: Section II summarizes concepts useful to better understand our work and Section III discusses the ILP formulation and presents our top-down FBS algorithm. Section IV presents experimental results, and Section V concludes the paper with a brief discussion.

II. Background

This section gives background on and-inverter graphs (AIGs), the unit delay model, and node equivalence.

A. And-Inverter Graphs

The and-inverter graph (AIG) is a directed acyclic graph (DAG) representation of logic where nodes represent primary inputs or 2-input AND gates, and edges can be inverted or regular indicating the presence or absence of inverters. The AIG is a universal representation (i.e., AIGs can represent arbitrary logic functions), and is supported by numerous logic synthesis tools and libraries such as ABC [16].

B. Unit Delay Model

In this work, we use the unit delay model where each gate incurs a unit delay. The arrival time (or level) of a node n, denoted by t_n^{arr}, is defined to be 0 if n is a primary input and $1 + \max_{m \in \text{FI}(n)} t_m^{\text{arr}}$ otherwise where $\text{FI}(n)$ is the fanin nodes of n. The overall circuit delay (i.e., circuit depth) is the maximum arrival time of any primary output. For a given target delay D, the required time of a node n denote by t_n^{req} is defined to be D if all fanouts of n are primary outputs and $\min_{m \in \text{FO}(n)} t_m^{\text{req}} - 1$ otherwise, where $\text{FO}(n)$ is the fanout nodes of n. A critical path is an input-to-output path of nodes where each node n on the path satisfies $t_n^{\text{req}} = t_n^{\text{arr}}$.

C. Node Equivalence

We say two nodes in a logic network are equivalent if their outputs are the same under all possible values of primary inputs. If two or more nodes are equivalent, their fanouts can be redistributed among themselves without altering the overall functionality of the network. As identifying all such node equivalences is computationally expensive, the equivalence is often considered with respect to a small cut. An example of this is *structural hashing* which was originally used in IBM CAD tools [17]; For AIGs, a widely used structural hashing technique is to identify gates with signatures consisting of their fanins (including flags denoting the presence of inverters).

In this work, we use an AIG data structure that internally uses structural hashing to avoid multiple equivalent nodes in the input network. However, for the output circuit, the algorithm may need some explicitly duplicated gates, thus we disable structural hashing for the output network.

III. Fanout-Bounded Synthesis

In this section, we first present our ILP formulation of FBS and then present our top-down algorithm.

A. Optimum Fanout-Bounded Synthesis ILP Formulation

We formulate the minimum area FBS problem for a predefined depth bound D as an ILP. We remark that our ILP does not consider logic restructuring; instead, it determines how to duplicate gates and add buffers to the original network in the best possible way to meet the fanout bounds.

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. 1, $I = \{i_1, \ldots, i_4\}, G = \{n_1, \ldots, n_7\}$ and $N = \{i_1, \ldots, i_4, n_1, \ldots, n_7\}$. For a node $n \in N$, let $\text{FO}(n)$ be the 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. 1, we have $\text{FO}(n_1) = \{n_3, n_4\}$ and $\text{FO}(n_3) = \{n_4, n_5, n_6\}$, and $k_{n_2} = k_{n_4} = k_{n_5} = k_{n_6} = k_{n_7} = 1$. Let c_{g} and c_{b} be the area of a gate and a buffer and let f_{g} and f_{b} be the fanout capacity of a gate and a buffer.

Let $n \in N$ be a node in the input logic network. We say a node m in a fanout bounded version 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 has fanins n_1, n_2 and m has fanins m_1, m_2 such that m_1 is n_1-equivalent and m_2 is n_2-equivalent, or
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 network shown on the right of Fig. 1, there are two n_1-equivalent gates and two n_3-equivalent gates (represented by overlapping circles). The two buffers represented as triangles in level 2 are also n_2-equivalent.

1) Variables: We use two kinds of integer variables. For each node $n \in G$ and for each level $\ell \in \{0, \ldots, D\}$, the variable $g_{n,\ell}$ denotes the number of n-equivalent gate copies in level ℓ in the fanout bounded version. Similarly, the variable $b_{n,\ell}$ denotes the number of n-equivalent buffers in level ℓ in the fanout bounded version. For example, for the logic network shown on the left of Fig. 1, 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 $\ell \leq 3$.

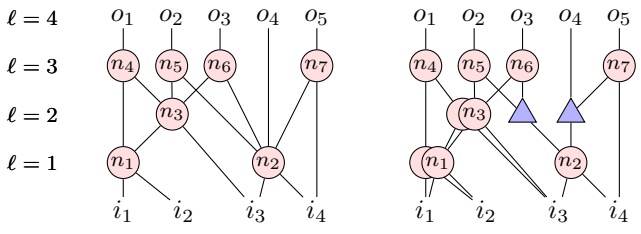

Fig. 1: Example logic network (left) and a possible fanout bounded version assuming a fanout limit of 2 (right).

2) Constraints: Next, we introduce constraints to ensure that the variables correspond to a valid fanout bounded version of the input network. To this end, consider a fixed level $L \in \{1, \ldots, D\}$ and a fixed gate $n \in G$. Let $\mathrm{a}(n, L)$ denote the total fanout capacity of all n-equivalent gates/buffers that are placed in levels strictly less than L, namely $\mathrm{a}(n, L) = \sum_{\ell=0}^{L-1}(f_\mathrm{b} \cdot b_{n,\ell} + f_\mathrm{g} \cdot g_{n,\ell})$. Let $\mathrm{r}(n, L)$ be 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 and each n-equivalent buffer increases the fanout requirement by one. Thus we have $\mathrm{r}(n, L) = \sum_{\ell=1}^{L}\left(b_{n,L} + \sum_{m \in \mathrm{FO}(n)} g_{m,L}\right)$.

Now, observe that in any variable assignment that corresponds to a valid fanout bounded version with depth D, it must hold that $\mathrm{a}(n, L) \geq \mathrm{r}(n, L)$ for all $n \in G$ and $L \in 1, \ldots, D$. One can easily verify this for $L = 1$; for any gate $n \in G$, its fanouts can never be in the same level, thus both $\mathrm{r}(n, 1)$ and $\mathrm{a}(n, 1)$ are zero. Suppose that $\mathrm{a}(n, L) \geq \mathrm{r}(n, L)$ holds for any valid depth-D fanout bounded version. We inductively show that $\mathrm{a}(n, L+1) \geq \mathrm{r}(n, L+1)$ must also hold. 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 $\sum_{m \in \mathrm{FO}(n)} g_{n,L+1} + b_{n,L+1}$. The total remaining capacity of n-equivalent gates/buffers that are at levels below L is $\mathrm{a}(n, L) - \mathrm{r}(n, L)$. Thus the additional capacity needed for the crossing connections must be provided by n-equivalent gates/buffer at level L. Namely, we must have $f_\mathrm{g} \cdot g_{n,L} + f_\mathrm{b} \cdot b_{n,L} \geq \sum_{m \in \mathrm{FO}(n)} g_{n,L+1} + b_{n,L+1} - (\mathrm{a}(n, L) - \mathrm{r}(n, L))$, which yields $\mathrm{a}(n, L+1) \geq \mathrm{r}(n, L+1)$ after rearranging. Next, there must be sufficient capacity remaining in n-equivalent gates/buffers to support the primary outputs (if any). Namely, for all n, it must hold that $\mathrm{a}(n, D+1) - \mathrm{r}(n, D) \geq k_n$.

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 $\sum_{n \in G} \sum_{\ell=1}^{D}(c_\mathrm{g} \cdot g_{n,\ell} + c_\mathrm{b} \cdot b_{n,\ell})$ subject to

$$\begin{aligned} \mathrm{a}(n, L) - \mathrm{r}(n, L) &\geq 0 && \forall\, n \in N, 1 \leq L \leq D, \\ \mathrm{a}(n, D+1) - \mathrm{r}(n, D) &\geq k_n && \forall\, n \in N, \\ g_{n,0} = 0,\ b_{n,0} &= 0 && n \in N, \\ g_{n,L}, b_{n,L} &\in \mathbb{Z} && \forall\, n \in N, 1 \leq L \leq D. \end{aligned}$$

Let OPT be the optimum area of a fanout bounded version of the input network with maximum depth D. As any valid network corresponds to a feasible ILP solution, the value of the ILP is at most OPT. We now give an algorithm (Algorithm 1) to construct the corresponding depth-D fanout bounded version of the input network from any feasible ILP solution, 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 do so, 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 node, it decrements the remaining capacity; once it reaches zero, the gate/buffer is removed from the queue. Since the algorithms constructs nodes in a level-by-level fashion using a feasible variable assignment that satisfies availability-requirement constraints, we can see that the algorithm always has sufficient equivalent signals in the corresponding queues when executing Line 11 and Line 15.

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

input : Input network ntk, parameters $f_\mathrm{g}, f_\mathrm{b}$, and a feasible ILP solution $g_{n,L}, b_{n,L}$ for $n \in N$ and $0 \leq L \leq D$.
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] ← newntk.create_pi()
4 Let $data$ be an empty list.
5 **for** *all nonzero* $g_{n,L}$ **do** Add $(L, n,$ "$gate$") to $data$
6 **for** *all nonzero* $b_{n,L}$ **do** Add $(L, n,$ "$buff$") 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 ← Create a new gate by choosing the first available equivalent fanins in newsig.
12 Decrement the remaining capacity for used fanin nodes and remove them from the queue if the remaining capacity reach zero.
13 newsig[m].$push$((newgate, f_g))
14 **else**
15 newbuff ← Create a new buffer by choosing the first available equivalent node in newsig[m].
16 Decrement the remaining capacity for the used fanin.
17 Pop from newsig[m] if remaining capacity is zero.
18 newsig[m].$push$((newbuff, f_b))
19 **return** the constructed network.

Remark: Recall that, in technologies such as AQFP, there is an additional requirement that the inputs of each gate must arrive at the same time. In our ILP formulation, this is easy to ensure; we simply re-define $\mathrm{a}(n, L)$ and $\mathrm{r}(n, L)$ to be, respectively, the available fanout capacity by n-equivalent nodes in level $L-1$ and the required fanout capacity of n-equivalent nodes due to nodes in level L.

B. Top-Down Fanout-Bounded Synthesis

Although solving the ILP introduced in Section III-A gives the optimal solution, solving it optimally for large networks can be prohibitively expensive in general. Thus we now show how to find a good (but not necessarily optimal), feasible 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 $g_{n,L}$ and $b_{n,L}$ such that $\mathrm{a}(n, L) - \mathrm{r}(n, L) \geq 0$ and $\mathrm{a}(n, D+1) - \mathrm{r}(n, D) \geq k_n$ 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.

Observe that duplicating a gate will increase the fanout requirement of two (or more if using higher-fanin gates) other

Algorithm 2: Algorithm 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 $f_{\mathrm{g}}, f_{\mathrm{b}}$, 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 $L = 1, \ldots, D$.

```
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_g
    if rem ≤ 0 then
        for i = 1 to length(folev_n) in steps of f_g do
            Increment g_{n,folev_n[i]−1}.
            return variable values
    s ← rem mod (f_b − 1)
    if s > 0 then
        Add f_b − 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
```

nodes. In contrast, adding a buffer only increases the fanout load by one. Moreover, 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, when determining the values for $g_{n,L}$ and $b_{n,L}$, our top-down approach prefers buffers over gate-duplication.

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

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

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

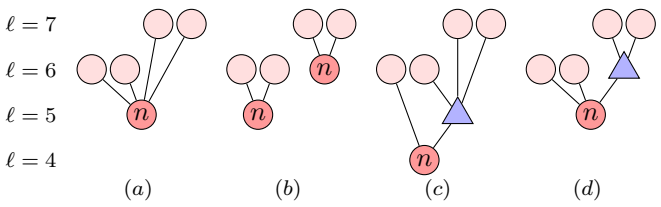

Fig. 2: A fanout net for a node n with levels of fanouts already decided (a), two possible outcomes for the fanout net of n if the algorithm of [12] is used (b and c), and the optimum buffer tree for n (d) when $f_{\mathrm{b}} = f_{\mathrm{g}} = 3$ and $c_{\mathrm{g}} > c_{\mathrm{b}}$.

when the buffer-forest rebalancing algorithm is run. This can lead to some redundant gate copies that remain in the network even after rebalancing is performed. In contrast, our algorithm only creates as many n-equivalent gates as we absolutely need (and decides their levels), thus redundant gate copies are never created. Moreover, in the "skewed buffer tree construction" and "buffer-forest rebalancing" algorithms of [12], there can be situations where it does not construct the best buffer tree/forest when $f_{\mathrm{g}}, f_{\mathrm{b}} > 2$ and $c_{\mathrm{g}} > c_{\mathrm{b}}$. To see this, suppose that $f_{\mathrm{g}} = f_{\mathrm{b}} = 3$ and $c_{\mathrm{g}} > c_{\mathrm{b}}$ and consider the fanout net of Fig. 2 (a). The algorithm of [12] may either duplicate n to produce the structure of Fig. 2 (b) with cost $2 \cdot c_{\mathrm{g}}$ or it may construct the skewed buffer tree of Fig. 2 (c) where n is placed at level 4. However, the buffer tree shown in Fig. 2 (d) is better than both the options; it has a lower area than the one in Fig. 2 (b) and gives a better placement for node n than the one in Fig. 2 (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 [15] to derive r buffer trees, and find the minimum value of r such that roots of all trees meet the arrival time requirement.

C. Allowing Over-Duplication

The previous sections explained how to find the smallest buffer forest for a given fanout net without increasing the circuit delay, where we preferred buffers over gate duplication. Consider a scenario where we may have the option of placing two copies of a node n at level $t_n^{\mathrm{arr}} + 1$, but we end up placing a single copy of n at level t_n^{arr} as we locally minimize duplication. However, this can force duplication of more than one of n's fanin nodes as *their* fanout nets might not have enough room to add buffers. However, we might avoid all those duplicates if we had duplicated n instead.

Motivated by this, we now present an improved version of our top-down approach which we call *top-down with over-duplication*. Namely, for the fanout net of a given node n, instead of stopping the algorithm Algorithm 2 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.

TABLE I: Exact fanout-bounded synthesis results using ILP from Section III-A solved with Gurobi optimizer [18].

	Input network		Output network				
Benchmark	Gates	Levels	Gates	Buffers	Total	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

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. Experimental Results

In this section, we present the experimental results obtained from our ILP formulation and the top-down FBS algorithm and compare the results against those of [12].

First, for a set of small benchmarks, we use the ILP to find the exact solutions; Using the minimum possible circuit delay for the fanout-unbounded input network as the delay bound, we write the ILP introduced in Section III-A, and solve it using Gurobi optimizer [18] 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 AND 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 RAM. The results are shown in Table I. The first 8 benchmarks are from the EPFL logic synthesis benchmarks suite [19] whereas the rest of the benchmarks are a subset of those used in [20].

Next, we evaluate our algorithm on the benchmarks of [20]. As we show in Table II, our initial top-down approach already achieves the optimum on several benchmarks. With over-duplication allowed, our approach 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 number of fanouts).

TABLE II: Results of the top-down fanout bounded synthesis on benchmarks of [20].

	Naive top-down			With over-duplication		
Benchmark	Gates	Buffers	Total	Gates	Buffers	Total
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

We then present the results (Table III) 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 after one round of `resyn2` command in ABC. Note that the quality of results (QoR) measure 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., "sqrt").

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 the 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).

As 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 the recomputation of costs in the over-duplication step.

V. Conclusion

In this work, we formulate the problem of FBS for a fixed target delay as an ILP, and we show how to find a feasible solution to the ILP using a top-down approach. As compared to the

TABLE III: 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	And gates	Levels	Total gates	Levels	And gates	Buffers	Total gates	Impr.%	Time (s)	And gates	Buffers	Total gates	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	

state-of-the-art, our top-down algorithm with over-duplication achieves 11.82% improved area while achieving matching or better delays. Both our ILP and the top-down approach are very versatile and can be adapted to consider additional technology-specific constraints in emerging technologies such as the path balancing constraints of the AQFP technology.

We hope that our ILP would serve as a theoretical basis for future research in FBS for emerging technologies with hard fanout constraints. The exact results produced by the ILP serve as the ground truth for evaluating future FBS algorithms.

Our over-duplication heuristic with a local cost function helps in improving overall area reduction as compared to the simple top-down approach. We believe this motivates further research on whether there are other efficiently computable such heuristics that yield even better results.

REFERENCES

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

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

[3] 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 *Proceedings of the 19th International Symposium on Physical Design*, ser. ISPD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 75–82. [Online]. Available: https://doi.org/10.1145/1735023.1735047

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

[5] D. A. 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 *ICCAD-2005. IEEE/ACM International Conference 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, p. 035010, 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 Transactions on Applied Superconductivity*, vol. 1, no. 1, pp. 3–28, 1991.

[10] 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 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 ACM (JACM)*, vol. 31, pp. 13–18, 1984.

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

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

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

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

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

[17] 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.

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

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

[20] 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 *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE Press, 2021, p. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD51958.2021.9643456