A Dynamic Accuracy-Refinement Approach to Timing-Driven Technology Mapping

Sz-Cheng Huang and Jie-Hong R. Jiang
Graduate Institute of Electronics Engineering/Department of Electrical Engineering
National Taiwan University, Taipei 10617, Taiwan
{r95943156@ntu.edu.tw; jhjiang@cc.ee.ntu.edu.tw}

***Abstract*— Technology mapping aims at searching an optimal implementation for a Boolean netlist using gates from a technology library. Compared with its $\mathcal{NP}$-complete area minimization counterpart, DAG mapping for delay minimization is considered much sophisticated because matching choices must be made without knowing actual arrival times and output loads. Traditional approaches to this problem involve too many approximate simplifications, and are far from accurate. In contrast, this paper tackles this problem directly under load-dependent DAG mapping. The enabling techniques for accurate optimization include on-the-fly load-estimation refinement, breadth-first backward covering for load consolidation, and use of a piecewise linear model for accurate timing calculation. Experimental results show that, compared with the state-of-the-art mapper, our method averagely reduces circuit delay by 39%, with 11% increase in area, for large benchmark circuits.**

I. Introduction

Technology mapping transforms a technology-independent Boolean network into a circuit composed of primitive gates chosen from a technology library. Before technology mapping, a Boolean network is often restructured to simplify its Boolean expressions according to various design constraints, such as area, timing, power, reliability, etc. Technology mapping then continues to further optimize the circuit with respect to a target technology node.

Technology mapping plays a pivotal role in the electronic design automation (EDA) flow. It bridges technology-independent logic synthesis and technology-dependent physical optimization, and is the key to resolving the infamous design closure problem, see, e.g., [1], [2], [3], [4], [5], [6] for some recent advances along this line. Accordingly, effective timing-driven technology mapping may alleviate the design closure problem and help the convergence of iterations between logic synthesis and physical design.

The first algorithmic approach to technology mapping is DAGON [7]. Based on dynamic programming, Keutzer formulated min-area technology mapping as a tree covering problem. Rudell went on and showed that min-delay technology mapping can be solved in linear time for tree mapping under a load-independent timing model [8]. He also extended the algorithm to take into account load effects for delay by a binning technique. Touati *et al.* [9] later proposed a more efficient approach using piecewise linear (PWL) functions to implicitly consider any load values. They also indicated tree covering alone tends to find a suboptimal solution because most circuits are directed acyclic graphs (DAGs), rather than trees. To handle the multiple-fanout problem, heuristic algorithms were proposed to reduce the delay incurred by distributing a signal to multiple destinations. These fanout optimization techniques were also used during tree covering to estimate delay at multiple-fanout nodes, which appear only for tree roots.

In contrast to tree mapping, based on FLOWMAP [10] Kukimoto *et al.* showed that min-delay DAG mapping can be done in polynomial time under the constant delay model [11]. On the other hand, Stok *et al.* took advantage of the load-independence property of the gain-based timing model and proposed the WAVEFRONT algorithm [12]. It is not until these efforts that a DAG could be mapped directly without being partitioned into a set of trees. Although impressive progress has been made in timing-driven technology mapping, to the best of our knowledge the problem has not been tackled directly for DAGs under load-dependent timing models. Even though load estimation by counting fanout numbers was proposed before in [13] and [14], the estimation is still not accurate enough. Furthermore, other difficulties, to be discussed, in load-dependent DAG mapping were not fully addressed.

In this paper, we reexamine the problem of timing-driven technology mapping for DAGs, and propose solutions to overcome difficulties risen due to the use of load-dependent timing models. Compared with prior work, our method provides a more accurate optimization via the following improvements:

- the use of PWL functions to the entire DAG, in contrast to tree roots in [9],
- the use of a more accurate adaptive estimation of fanout loads, and
- the use of backward covering in a breadth-first order for early load consolidation.

Thereby, our algorithm can perform technology mapping on DAGs directly under load-dependent timing models with more accurate timing optimization than prior methods. We believe our techniques can also be extended to estimate other load-dependent metrics, such as power.

This paper is organized as follows. Preliminaries and the problem definition are given in Section II. Our solutions to the identified difficulties are detailed in Section III; the computational complexity is analyzed in Section IV. Section V shows experimental results. Finally we conclude this paper and outline future work in Section VI.

II. Preliminaries and Problem Definition

A ***Boolean network*** is a graph $G(V,E)$ representation of Boolean equations, where a ***node*** or ***vertex*** in V is associated with a single-output completely specified Boolean function, and a directed edge $(u,v) \in E$, connecting from node u to node v, signifies the function of node v depends on the output of node u. Some nodes of V are identified as ***primary inputs*** (PIs) and some as ***primary outputs*** (POs), where PIs have only outgoing edges and POs have only incoming edges. The ***fanins*** of node v are the nodes $\{u \mid (u,v) \in E\}$; the ***fanouts*** of node u are the nodes $\{v \mid (u,v) \in E\}$. A node u is a ***transitive fanin*** of node v (likewise a node v is a ***transitive fanout*** of node u) if there is a path from u to v in G.

A Boolean network can be converted into a ***subject graph*** by decomposing the network into a circuit of primitive gates,

often two-input NAND gates and inverters. A technology ***library*** is a collection of pre-designed logic cells, where each cell function is represented as a circuit of primitive gates, called a ***pattern graph***. A library provides detail characteristics of its logic cells. Each ***pin*** of a pattern graph is associated with rise/fall time, input load, maximum allowable output load, etc., for circuit analysis. A ***match*** m at a vertex v of a subject graph S is an instance of a pattern graph that is functionally equivalent to some subgraph of S rooted at v. We associate such a subgraph with a ***cut*** c, which is a subset of the vertices in v's transitive fanin cone such that every path from PIs to v passes through at least one vertex in the cut. An input to cut c and to match m is a fanin of some vertex in c. Moreover, the match set of v, denoted M_v, contains all the matches of v. The set of matches functionally equivalent to the associated subgraph for cut c is denoted as M_c. Note that $M_c \subseteq M_v$ if c is a cut of v.

For timing analysis, a simple, yet effective, load-dependent timing model is the linear model, where the delay $d_p^g(l)$ of gate (or pattern graph) g with respect to its input pin p under output capacitive load l is given by

$$d_p^g(l) = b_p^g + l \times f_p^g, \tag{1}$$

where b_p^g is the intrinsic gate delay, and f_p^g is the induced delay per unit load in driving l. Excluding the second term on the right-hand side of 1, it reduces to the constant delay model. These two delay models are popularly used in logic synthesis tools for delay calculation, e.g., in SIS [15]. These two models are selected from load-dependent and load-independent classes for our discussion.

We define the technology mapping problem formally as follows. Given a subject DAG S, a library L, a load-dependent timing model, timing-driven technology mapping for S (without partitioning S into trees) aims at covering each vertex of S with at least one pattern graph from L such that

- the inputs to a match are the outputs of some other matches, and
- the circuit delay evaluated by the given timing model is minimized.

III. Bottlenecks and Solutions

This section constitutes the heart of this paper. We describe the state-of-the-art mapper followed by the solutions to the difficulties risen from the migration of the timing model.

A. Overview of the State-of-the-Art Mapper

Many mapping algorithms were developed under load-independent timing models for optimality, under the premise that load is unimportant, so was the state-of-the-art *ABC* mapper. Algorithm 1 shows the main ideas of the *ABC* mapper with some advanced techniques excluded, such as supergates and phase assignment, and will be used to illustrate our method. This algorithm is based on the work of [11] and is divided into two phases as usual. In the beginning, a Boolean network is transformed into a subject graph S. Then vertices of S are collected into the queue Q_v in a topological order by performing a depth-first search on S in line 4. Lines 5-15 look for a best match with minimum arrival time for every vertex of S. Since the vertices are matched in a topological order, the arrival times at the inputs would have been computed when a vertex v is under consideration. Subsequently the arrival time, AT_m^v, at vertex v covered by match m can be determined in line 11 by taking the maximum among the summations of every input arrival time and its corresponding pin-to-pin delay of m. The arrival time at v is determined after all matches of v are considered and the best match is remembered. After the matching phase is done, the function *Covering* covers, in a reverse topological order, an unvisited node with the best match kept in the matching phase, and then proceeds to cover the inputs of the match. The process repeats until all PIs are visited, thereby deriving a mapped network.

Algorithm 1: Tech_Map(NTK, LIB, MOD)

```
   inputs: NTK is a Boolean network to be mapped;
           LIB is a technology library;
           MOD is a constant pin-to-pin delay model.
   output: A solution with minimum delay.
 1 begin
 2     S = (V_S, E_S) ← ToSubjectGraph(NTK);
 3     // The matching phase.
 4     Q_V ← DFS(S);
 5     while Q_V ≠ ∅ do
 6         v ← POP(Q_V);
 7         AT^v ← ∞;
 8         forall c is a cut of v do
 9             M_c ← Match(c, LIB);
10             forall m ∈ M_c do
11                 AT_m^v ← ComputeArrival(m, MOD);
12                 if AT_m^v < AT^v then
13                     Set the best match of c to m;
14                     Set the best cut of v to c;
15                     AT^v ← AT_m^v;
16     // The covering phase.
17     forall n is a fanin of a PO of S do
18         Covering(n);
19 end
```

Algorithm 2: *Covering*(n) – a depth-first manner.

```
   inputs: n is the node to reference.
   output: A set of matches for the solution.
 1 begin
 2     if NotVisited(n) then
 3         Mark n as visited;
 4         c ← the best cut of n;
 5         Add the best match of c to M;
 6         forall i ∈ Inputs(c) do
 7             if i is not a PI then
 8                 Covering(i);
 9     return M
10 end
```

Based on dynamic programming, Algorithm 1 always produces a solution optimal under a load-independent timing model. However, if loading effects are considered, the optimality is no longer guaranteed. Below we detail the bottlenecks for timing-driven technology mapping under load-dependent timing models, and propose solutions to eliminate them.

B. Difficulties and Solutions

1) Preparing optimal matching candidates: **Question.** The matching phase in Algorithm 1 determines a best match with minimum arrival time for each node. The decision, however, cannot be made unless the output pattern graphs are known *a priori* since the delay of a match depends on its output capacitive load under a load-dependent timing model. Unfortunately, load values are unavailable in advance because the network is traversed in a topological order

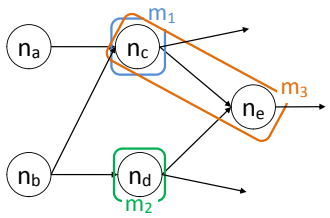

Fig. 1. Illustration of uncertainty of fanin delay.

from PIs to POs. Accordingly the first question is: *How to determine the best match at a node under unknown output load?*

Answer. Our solution is to defer decision making. Since different load values result in different best matches and real output loads are unknown, more than one optimal matching candidates must be kept for each vertex to take care of all possible load values. A straightforward idea is to store the best match for each possible load value at every vertex. However this approach is redundant and unpractical because candidates for best matches often repeat and there may be too many load values to consider. In [9] a better approach was proposed to take advantage of the linear timing model and was used in tree covering considering load effects. We extend this idea to DAG mapping as follows.

We assume the input arrival times are known, even if this is not true, but will be handled later. Under this assumption, the arrival time at a vertex v covered by a match m associated with a cut c can be computed by

$$\begin{aligned} AT_m^v &= \max_{\forall i \in Inputs(m)} \left\{ AT^i + d_{p_i}^m(l) \right\} \\ &= \max_{\forall i \in Inputs(m)} \left\{ (AT^i + b_{p_i}^m) + l \times f_{p_i}^m \right\}, \end{aligned} \quad (2)$$

where input i connects to m through pin p_i. As both $AT^i + b_{p_i}^m$ and $f_{p_i}^m$ are constants, Eq. (2) states that not only the pin-to-pin gate delay is PWL, but also is the arrival time at node v. Accordingly, best matching candidates can be computed by and represented in $\min_{\forall m \in M_v} \{AT_m^v\}$, which is again PWL. Every linear segment of $\min_{\forall m \in M_v} \{AT_m^v\}$ corresponds to a candidate; all load values under a linear segment share the same candidate. Thereby an optimal match can be computed easily by looking it up when a load value is given.

2) Estimating unknown input arrival time: **Question.** In addition to the unknown loading problem at gate outputs, problems occur at gate inputs, too. Since the arrival time at a node depends on its inputs, apparently it cannot be determined unless the input arrival times are certain. Unfortunately, they cannot be certain because of the unknown output load. In the literature, such a problem is addressed for subject graphs with multiple-fanout nodes. Partitioning a subject graph into trees make each node drive at most one output pattern graph in the final mapping. To tackle DAG mapping directly, the at-most-one-output property no longer holds for every node. As illustrated in Fig. 1, both arrival times at node n_c covered by match m_1 and at node n_d covered by match m_2 depend on the arrival time at the common input n_b; neither of them can be computed without knowing the cover of the other node. This "deadlock" results in the uncertainty of the arrival time at the input n_b. Even for the single-fanout node n_a, the arrival time at n_a is still unknown because a match at this node may drive m_1, m_3, or both in the final mapped network. So the second question is: *How to estimate input arrival time in DAG mapping?*

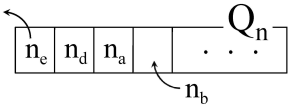

Fig. 2. The manipulation of the covering queue.

Answer. Our solution is to predict under partial information. Computing arrival times in the matching phase requires output loads to be known, which is not possible until the covering phase. Also the covering phase needs to know the delay information produced in the matching phase. The cyclic dependency of this chicken-and-egg problem must be broken somehow.

We propose a look-ahead strategy, using the *DynamicExpectedLoad* as defined below, such that input arrival times can be estimated and determined independently.

DEFINITION 1. *A **potential fanout** of a vertex v in a subject graph is a match m' at another vertex v' in v's transitive fanout cone such that $v \in Inputs(m')$.*

For example, in Fig. 1, m_3 and m_1 are potential fanouts of node n_a.

DEFINITION 2. *A **potential load** of v is a load incurred by one of its potential fanouts.*

Let v be a vertex whose arrival time is to be determined and let match $m_{\widehat{v}} \in M_{\widehat{v}}$ be a potential fanout of v. The arrival time at v required by the vertex $\widehat{v}$ in its transitive fanout cone can be calculated by

$$\begin{aligned} DynamicExpectedLoad(v, m_{\widehat{v}}) = PotentialLoad(v, m_{\widehat{v}}) + \\ \sum_{\forall v' \in V'} \sum_{\forall m_{v'} \in M_{v'}} PotentialLoad(v, m_{v'}) \times \frac{1}{|M_{v'}|} \end{aligned} \quad (3)$$

where V' is the collection of all the nodes in v's transitive fanout cone except $\widehat{v}$. By substituting this value for l in Eq. (2), the arrival time at v is obtained.

This strategy has several features: First, given a match $m_{\widehat{v}}$ of $\widehat{v}$, Eq. (3) assumes that $m_{\widehat{v}}$ happens to be the match of $\widehat{v}$ selected in the final covering. Hence this assumption excludes other matches of $\widehat{v}$ from being selected in the final covering and from contributing to the potential load of v. On the other hand, since the matches of $v' \in V'$ can possibly contribute to v's potential load, their averaged potential loads are taken into account. Thereby, Eq. (3) dynamically computes potential loads with respect to different $\widehat{v}$'s, and even different $m_{\widehat{v}}$'s. Second, the computation explores in advance the unmapped part of the subject graph in v's transitive fanout cone. Moreover, only matches that may possibly contribute to v's output load are considered. It disregards the matches that are not potential fanouts. Third, although theoretically there can be too many potential fanouts to consider for a node v, especially for v close to PIs or having many fanouts, the cut generator, which identifies cuts for every node in the subject graph, tends to make our strategy practical since a cut is often limited to some small input size or depth. Note that only one forward traversal on the subject graph is needed in this strategy. Whenever a cut is identified and matched, the potential loads are remembered for the inputs of the match.

As will be seen in Table I, empirical results suggest our estimation technique is superior to prior approaches [13] [14].

3) Covering subject graph: **Question.** The remaining problem is about the covering phase. Note that Algorithm 2 finds an optimal cover in a depth-first manner from POs to PIs. By depth-first search, when a node is visited, probably not all of its output matches to drive are clear yet. By line 2, if a node has been covered as a root of a pattern

graph, it will never be covered as a root again. So upon visiting a vertex, a match rooted at it must be decided even under partial information only. Although this procedure is okay for a load-independent timing model, it may result in a suboptimal solution under a load-dependent model. The third question is: *How to rectify the partial load information under depth-first based covering?*

Answer. Our solution is instead to use breadth-first covering as shown in Algorithm 3.

Algorithm 3: *Covering2*(S) – a breadth-first manner.

```
   inputs: S := (V_S, E_S) is the subject graph.
   output: A set of matches for the solution.
1  begin
2      GetReverseTopologicalOrderAttribute(S);
3      Q_n ← ∅;
4      forall n is a fanin of a PO of S do
5          InsertByReverseTopologicalOrder(Q_n, n);
6      while Q_n ≠ ∅ do
7          n ← POP(Q_n);
8          c ← the best cut of n;
9          Add the best match of c to M;
10         forall i ∈ Inputs(c) do
11             if i is not a PI then
12                 InsertByReverseTopologicalOrder(Q_n, i);
13     return M
14 end
```

Algorithm *Covering2* starts to find a solution by first computing the reverse topological-order attribute of each node of S in line 2 by traversing S from POs to PIs. Lines 4-5 then collect all the fanins of POs. Q_n here is a queue that stores all the nodes waiting to be covered. Each activation of *InsertByReverseTopologicalOrder* inserts an entry, if not in Q_n, into Q_n by comparing the attribute of the entry with those of the elements already in Q_n. Every element in Q_n always takes priority over elements inserted afterwards. For example, let nodes n_d and n_e in Fig. 1 connect to POs and the best match at n_e be m_3. Then line 5th will collect these two nodes into Q_n; the content of Q_n is shown in Fig. 2. If now n_e is popped out to be mapped, by line 12 n_a and n_b then will be inserted into the queue behind n_d.

The covering procedure maintains two properties: First, only nodes that will appear in the final mapped network are put in Q_n. Second, whenever an element is popped out from Q_n, its output load has been consolidated. The former ensures the correctness of the final solution; the latter ensures the readiness of complete load information during covering.

Practical experience suggests that breadth-first covering is superior to depth-first one. Experiments show that, without using breadth-first covering, circuit delay can increase 13.24% on average and up to about 2 times.

With our solutions to the difficulties, timing-driven DAG mapping under a load-dependent timing model can be solved with more accurate timing information, thus achieving better optimization.

IV. Complexity Analysis

In our actual implementation, there are three main traversals on the subject graph. The first (forward) traversal collects all possible matches at each node. For a match, its induced output loads at its fanin nodes are recorded for later delay calculation. The computation is of complexity $O(npk)$, where n is the number of nodes in the subject graph, p the number of pattern graphs in the library, and k the maximal number of inputs among pattern graphs.[1] The second (forward) traversal computes arrival times using the potential loads obtained in the first traversal, and filters out non-optimal matches at each node. The arrival time at an input of a match is computed by looking up delay values in its corresponding PWL table. With binary search, there are at most $O(\log p)$ look-ups [9]. Then the arrival time, a PWL function, at the output of a match is computed by taking the piecewise minimum among its pin-to-pin delay functions added with their corresponding input arrival times. For each node, only optimal matches are kept. The complexity is bounded by $O(nkp \log p)$. The third (backward) traversal in the covering phase is of complexity $O(n)$. Hence the overall complexity is $O(nkp \log p) = O(n)$ because p and k are constants.

V. Experimental Evaluation and Analysis

We implemented our method within ABC [16]. Experiments were conducted on a Linux machine with Xeon 3.2GHz CPU and 5Gb RAM. Large circuits from ISCAS89, IWLS93, and ITC99 benchmark suits are collected for evaluation. Target library MCNC.GENLIB [15] is used. Boolean matching was applied to identify matches. All mapped networks were verified by equivalence checking.

Table I compares our method with the state-of-the-art mapper in ABC. Columns 1-2 list the circuits and their sizes in terms of AIG nodes. The mapping results of ABC for delay minimization (the flow of Algorithm 1) are shown in Columns 3-4. With further area recovery in ABC [17], where matches for timing non-critical nodes are replaced by slower but smaller gates, the results are shown in Columns 5-6. Our results are listed in Columns 7-9.

As can be seen in the bottom three rows of Table I which show average ratios that our method, compared with *ABC-Delay*, averagely reduces circuit delay by 39%, under 11% increase in area. Circuit `b14` is the only exception, where our method yields 1% increase in delay.[2] In fact, some of the delay improvements are substantial. Taking circuit `s35932` for example, we achieve 91% delay improvement subject to only 2% area increase shown in Table I. Such substantial improvements can be explained in Table II, where the fanout numbers of nodes were profiled for the mapped circuits of `s35932` and `s38584`. For `s35932`, there are three high-fanout nodes produced by *ABC-Delay* whereas our mapper produced gates with no more than three fanouts. An analysis shows that the gate delay, 176.4, of the 585-fanout node dominates the circuit delay, 178. Such high-fanout nodes are unfavorable. A similar scenario happens to `s38584`. As discussed in Section III.B.3, our method tends to avoid such problems and is better in general since load effects have been taken care of directly during the mapping phase.

In addition, to see how well our output-load estimation works, prior estimation techniques *FanAvg* and *SqrtFanAvg* [13], which estimate the output load of a node by multiplying the average load of the library with its fanout number and with the square root of its fanout number, respectively, were applied in our framework for comparison. The results are listed in Columns 10-13 of Table I. With our estimation technique, technology mapping yields better results, averagely by 29% and 30% decreasing in delay over *FanAvg* and *SqrtFanAvg*, respectively. *FanAvg* and *SqrtFanAvg* are likely to under-estimate output loads since the actual number of

[1] The standard complexity for Boolean matching is excluded.

[2] In our experiment, only 7 out of 99 circuits are not improved by our method, and most of them are small circuits not listed due to space limitation.

TABLE I
COMPARISON OF DIFFERENT MAPPING APPROACHES.

		ABC-Delay		ABC-Default		Ours			FanAvg[13]		SqrtFanAvg[13]		[14]		
circuit	#node	linear	area	linear	area	linear	nominal	area	linear	area	linear	area	linear	nominal	area
apex2	3191	169.3	9150.0	64.3	8006.0	82.8	41.9	11575.0	81.6	10028.0	80.9	9774.0	81.6	45.9	9148.0
b14	9821	133.1	17091.0	107.3	10531.0	134.6	89.2	18222.0	137.0	16274.0	135.3	16669.0	170.5	86.7	16310.0
b15	8437	189.2	23253.0	131.0	14948.0	172.3	102.7	24974.0	189.6	21881.0	190.6	22817.0	209.8	91.9	23034.0
b17	30874	254.6	71437.5	179.7	47149.5	181.0	114.9	78389.5	215.8	69491.5	204.9	72296.5	271.9	114.1	71800.5
b20	19704	193.3	34056.0	135.6	21372.0	148.1	97.2	36970.0	156.9	32727.0	173.0	32903.0	176.3	93.7	32833.0
b21	20049	190.3	34878.0	129.6	21758.0	149.6	98.7	37175.0	159.8	33359.0	153.8	33885.0	177.2	93.4	33435.0
b22	29184	224.5	53031.0	171.8	32485.0	155.7	103.6	56121.0	213.8	50641.0	161.8	51769.0	213.8	112.3	50420.0
C5315	3211	59.4	3875.5	57.0	2945.5	49.6	39.9	4520.5	61.4	4159.5	61.1	4162.5	61.4	39.2	4090.5
C6288	4832	154.7	8383.0	149.8	4666.0	127.5	102.8	8545.0	137.0	8835.0	128.9	8697.0	145.9	104.2	8935.0
C7552	3512	72.6	5171.0	78.3	3779.0	43.7	34.8	6028.0	74.0	5306.0	72.7	5577.0	99.4	40.4	5485.0
clma	25124	431.4	63622.0	327.4	38964.0	129.3	91.1	58907.0	395.9	59984.0	446.4	66379.0	395.9	162.4	52818.0
dsip	3429	279.5	6721.0	256.9	5141.0	104.5	59.7	6967.0	277.1	6744.0	277.4	6739.0	279.0	99.9	6510.0
misex3	3652	214.9	9637.0	191.8	8747.0	195.2	70.4	13962.0	219.0	12276.0	231.7	10148.0	219.0	87.2	9733.0
s13207	7951	160.7	6269.5	108.0	5070.5	67.4	49.0	6710.5	64.7	6144.5	88.1	6145.5	101.0	56.8	6167.5
s15850	9772	99.9	8384.0	98.5	6288.0	85.6	58.1	8890.0	134.2	7967.0	99.1	8295.0	134.2	59.5	8290.0
s35932	16065	178.0	22019.0	348.1	18181.0	15.4	13.5	22535.0	178.0	21443.0	178.0	21603.0	178.0	55.3	20771.0
s38417	22655	52.8	20850.0	56.3	16292.0	39.1	29.4	24283.0	57.4	21452.0	54.2	22027.0	58.2	29.4	21913.0
s38584	19253	443.8	26921.0	417.8	21441.0	71.7	47.2	30775.0	416.8	26838.0	420.4	27314.0	473.6	142.0	27156.0
s38584.1	18580	449.8	27479.0	413.2	21570.0	60.9	42.7	30526.0	101.5	28613.0	436.9	27585.0	180.7	141.9	27730.0
Avg_linear_ratio		1.00		0.89		0.61			0.89		0.90		0.98		
Avg_nominal_ratio							0.63							1.00	
Avg_area_ratio			1.00		0.72			1.11		1.01		0.90			0.99

output pattern graphs in a mapped network is often larger than the number of fanouts in the subject graph.

The last three columns of Table I show the results obtained using the fanout prediction method of [14], which was proposed under the nominal delay model. We evaluated the delay of the resultant mappings in terms of both linear and nominal delays. In comparison, the mapping with our prediction technique yields more than 37% delay reduction on average under both linear and nominal delay models.

Another interesting observation from Table I is that the area recovery of *ABC-Default* improves *ABC-Delay* not only in terms of area, but also in terms of delay. Supposedly circuit delay should remain the same, indeed under the constant delay model. Here however we analyzed timing in a more accurate load-dependent model. An investigation suggestes that the delay reduction is resulted from the replacements happening in non-critical regions. As a by-product, gates drive less loads after area recovery and behave faster.

In addition to showing the solution quality evaluated by the linear delay model, we also studied the circuit performance based on the constant delay model. Fig. 3 compares the difference. The column chart has y-axis indices on the left-hand side indicating the delay of the solutions derived by the *ABC* mapper and by our method, estimated by the constant (denoted superscripted C) and the linear (superscripted L) delay models. The line chart with y-axis indices on the right-hand side shows a series of ratios. By comparing *ABC-Delay*^{C} and *Ours*^{C}, we see that the *ABC* mapper always outperforms our method under the constant delay model. But under the linear delay model, however, situation is reversed. This phenomenon suggests that the constant delay model is not reliable for delay prediction since load effects are ignored. The small values of *ABC-Delay*$^{C/L}$, 0.23 on average, indicates that the delay estimated by the constant delay model is not representative. A win of the *ABC* mapper under the constant delay model is often a loss under the linear delay model. On the other hand, the values of *Ours*$^{C/L}$, 0.46 on average, reveal that our method is less misled by the constant delay model than *ABC-Delay*. Furthermore, the similarity between the trends of the line for *ABC-Delay*$^{C/L}$/*Ours*$^{C/L}$ and the line for *Ours*^{L}/*ABC-Delay*^{L} indicates that

- the larger *Ours*$^{C/L}$ is than *ABC-Delay*$^{C/L}$, the more our method outperforms the *ABC* mapper;
- the *ABC* mapper does not outperform ours much under the constant delay model since a small value of *Ours*^{C}/*ABC-Delay*^{C} is implied[3];
- the high correlation of these two lines suggests potential room for circuit delay reduction and justifies our method.

TABLE II
COMPARISON OF FANOUT NUMBERS AFTER TECHNOLOGY MAPPING.

s35932			s38584		
	ABC-Delay	Ours		ABC-Delay	Ours
#fanout	#node		#fanout	#node	
001	6540	6505	001-033	11020	12212
002	736	1185	034-066	4	3
003	0	64	067-100	1	2
208	1	0	144	1	0
432	1	0	177	0	1
585	1	0	1467	1	0

Table III compares the run time between ABC-Delay and our method. Ten representative circuits were selected for comparison. The results unsurprisingly show that our method runs much slower than the *ABC* mapper. This slowdown is due to the more elaborate computation. Nevertheless, our run time is still reasonable even for the largest circuits.

[3]Note the *ABC* mapper guarantees optimality if the solution is evaluated by the constant delay model. Therefore our method also achieve nearly optimal results under the constant delay model.

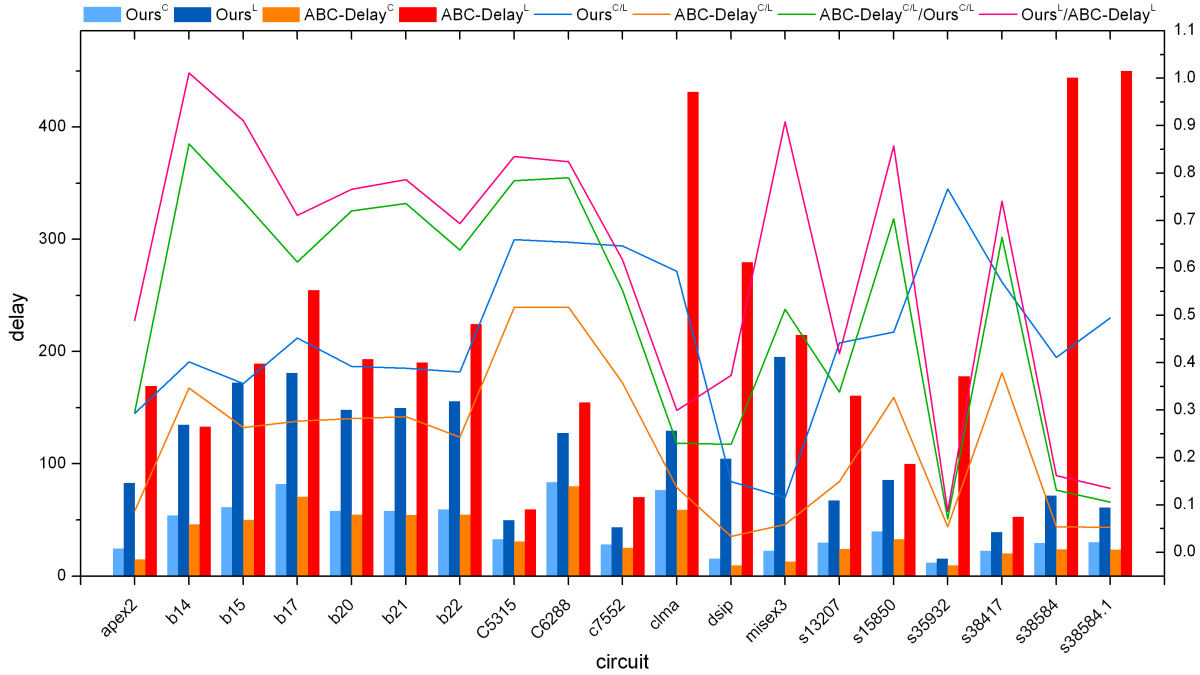

Fig. 3. CIRCUIT DELAY ESTIMATED BY LINEAR & CONSTANT MODELS.

TABLE III
COMPARISON OF RUN TIME.

	ABC-Delay (sec.)			Ours (sec.)		
circuit	phase1	phases2	total	phase1	phases2	total
apex2	0.02	0.02	0.04	2.45	0.13	2.58
b14	0.03	0.03	0.06	1.83	0.12	1.95
b20	0.07	0.08	0.15	3.47	0.45	3.92
C6288	0.02	0.02	0.04	0.52	0.01	0.53
C7552	0.01	0.01	0.02	0.46	0.01	0.47
clma	0.1	0.12	0.22	34.78	0.93	35.71
dsip	0.01	0.01	0.02	1.12	0.08	1.2
misex3	0.03	0.03	0.06	5.76	0.21	5.97
s15850	0.01	0.01	0.02	0.71	0.05	0.76
s38417	0.04	0.05	0.09	2.23	0.32	2.55

VI. CONCLUSIONS AND FUTURE WORK

We proposed a dynamic accuracy-refinement approach to timing-driven technology mapping. In particular, applied are timing calculation with a load-dependent timing model, on-the-fly load estimation refinement, and load consolidation with breadth-first backward covering. Thereby we can perform DAG mapping directly under a load-dependent timing model. Experimental results showed that, compared with the state-of-the-art mapper, our method averagely reduces circuit delay by by 39%, with a modest 11% increase in area, for large benchmark circuits.

Future work includes identifying timing non-critical regions for area recovery, and extending our methods for different optimization objectives and constraints.

REFERENCES

[1] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 16, no. 8, pp. 813–834, Aug. 1997.

[2] A. Mishchenko, S. Chatterjee, J.-H. Jiang, and R. Brayton, "Integrating logic synthesis, technology mapping, and retiming," in *Proc. IWLS*, 2005, pp. 177–181.

[3] J. Y. Lin, A. Jagannathan, and J. Cong, "Placement-driven technology mapping for lut-based fpgas," in *Proc. FPGA*, Feb. 2003, pp. 121–126.

[4] D. Pandini, L. T. Pileggi, and A. J. Strojwas, "Global and local congestion optimization in technology mapping," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 22, no. 4, pp. 498–505, Apr. 2003.

[5] R. Shelar, S. Sapatnekar, P. Saxena, and X. Wang, "An efficient technology mapping algorithm targeting routing congestion under delay constraints," in *Proc. ISPD*, Apr. 2005, pp. 137–144.

[6] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "Wiremap: Fpga technology mapping for improved routability," in *Proc. FPGA*, Feb. 2008, pp. 121–126.

[7] K. Keutzer, "Dagon: Technology binding and local optimization by DAG matching," in *Proc. DAC*, Jun. 1987, pp. 341–347.

[8] R. L. Rudell, "Logic synthesis for VLSI design," Ph.D. dissertation, Univ. of California Berkeley, CA, May 1989.

[9] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in *Proc. 6th MIT Conf. on Advanced Research in VLSI*, Apr. 1990, pp. 79–97.

[10] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 13, no. 1, pp. 1–12, Jan. 1994.

[11] Y. Kukimoto, R. K. Bryton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Proc. DAC*, Jun. 1998, pp. 348–351.

[12] L. Stok, M. A. Iyer, and A. J. Sullivan, "Wavefront technology mapping," in *Proc. DATE*, Mar. 1999, pp. 531–536.

[13] D. Jongeneel and R. H. J. M. Otten, "Technology mapping for area and speed," *Integr. the VLSI Journal*, vol. 29, no. 1, pp. 45–66, 2000.

[14] J. Cong and Y. Ding, "On nominal delay minimization in LUT-based FPGA technology mapping," in *Proc. FPGA*, 1995, pp. 82–88.

[15] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Electron. Reas. Lab., Dept. Elect. Eng. and Comput. Sci., Univ. California Berkeley, CA, Tech. Rep., May 1992.

[16] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* http://www.eecs.berkeley.edu/~alanmi/abc/.

[17] A. Mishchenko, S. Chatterjee, R. Brayton, and M. Ciesielski, "An integrated technology mapping environment," in *Proc. IWLS*, 2005, pp. 383–390.