A Tight Consistent Delay Model for Black Boxes

Robert Brayton Niklas Een Alan Mishchenko

Department of EECS, University of California, Berkeley

brayton@eecs.berkeley.edu niklas@een.se alanmi@eecs.berkeley.edu

Abstract

A Boolean network with unknown components can be modeled as an AIG with black boxes (bb-AIG). The delay of an AIG is measured in terms of its maximum level after it is **balanced**. *We propose a delay model for a black box which is* **consistent** *in that if a bb-AIG is refined by replacing an internal black box with another bb-AIG, its delay cannot decrease. It is also* **tight** *in that there exists an AIG which can replace a black box resulting in a delay equal to the estimated one. We prove that the proposed black box delay estimation is consistent and tight.*

1 Introduction

In logic synthesis, there are many results which improve the delay of a block of logic by taking into account the arrival times at its inputs. Thus, the delay of a block is not a fixed number; it depends on the set of input arrival times.

An example is *k*-input LUT optimization. The delay of a LUT is given as a set of delays from each input pin to the output pin. An easy local logic synthesis operation is to assign the latest arriving signal to the fastest pin, the second latest to the second fastest, etc. Then the logic bits in the LUT are adjusted appropriately.

Balancing is another logic synthesis operation used to speed up a circuit. Given a combinational logic function $f(x)$ represented as an AIG, its delay is often approximated by its level, i.e. the maximum number of nodes found on a path from any input to the output. Given arrival times at the primary inputs, balancing is done using the following algorithm implemented in ABC [1] as command *balance*:

balance(*g* = AIG, *a* = input arrival times){

- Create multi-input AND nodes η^m ($m > 2$)from AIG nodes; the η^m are derived by expanding towards the fanins of the two-input nodes until the expansion hits: (a) a primary input, (b) a complemented edge, (c) a node with multiple fanouts.
- During a topological computation of arrival times, starting with the given primary input arrival times, decompose η_m nodes, when encountered, into a well-balanced tree of two-input nodes as follows:
  - The two inputs, with the earliest arrival times, say a_1 and a_2, are combined into a two-input AND node, η_2, with arrival time $\max(a_1, a_2) + 1$.
  - The chosen inputs are replaced by the output of η_2, creating an AND node η^{m-1}.
  - This is repeated until there is only one node left.
  - The derived binary AND tree, η_2^m, contains *m*-1 nodes and *m* inputs. This tree is used to replace the η^m.
- The arrival time of this η^m is set to be the arrival time of the output of η_2^m.
- When the arrival times of all primary outputs have been computed, find the maximum arrival time, *D*.
- return *D*.

}

We denote the arrival times derived by applying the algorithm *balance*[1] as $\beta_g(a)$ (β stands for balance).

A similar operation can be done where groups of nodes are merged into a commutative function of its inputs, e.g. an XOR function. This algorithm is implemented in ABC as command *balance –x*.

A more powerful synthesis speedup transformation is SOP balancing [5] where a cone of nodes computing $f(x)$ is replaced by an SOP for *f*, which is then balanced with respect to the arrival times at *x* using *balance*.

In all these situations, a logic function does not have a fixed delay *d*; it has a delay dependent on 1) its arrival times and 2) the ability to re-synthesize $f(x)$ to minimize its output arrival time.

In general, we would like a method to analyze the output arrival time of a Boolean network, which

1) accounts for the ability to re-synthesize nodes on-the-fly in order to minimize its output arrival time.
2) is applicable to networks with unknown components.

As with a LUT, it is not enough to know the actual input-to-output delays. A more meaningful concept is that a delay is a *function* which, given arrival times at the inputs, gives an arrival time at the output. In this paper, we use the term delay to mean delay in this wider sense.

[1] This algorithm will be modified later to apply to AIGs with black boxes using the black box delay model we propose.

We consider *black-box AIGS* (bb-AIG) and propose a delay model for black boxes (bb-delay). Such a model should provide a good estimation of delay, based on the information available, such as the number of input pins and the set of arrival times *a* on these pins. We propose a delay model, which is tight and consistent.

A bb-delay is *consistent* if when a bb-AIG is *refined* by replacing an internal black box by a bb-AIG, the delay of the resulting bb-AIG can never be less that the original.

A bb-delay is *tight* if for any set of arrival times *a* at the inputs, there exists an AIG, which can replace a black box and the delay of the composition does not change.

A possible choice for a bb-delay might be $\max(a)+\log_2(n)$ where *a* is the set of arrival times at the *n* input pins of the black box. This is neither tight nor consistent because one can refine the box so that its delay is $\max(a)+1$, which also implies it is not consistent. One might consider $\max(a)+1$ as the delay, but this is not tight.

2 Black box delay and delay estimation

An AIG is a DAG where each node is a two-input AND node with an optional inversion at the inputs. The delay of the AIG is defined as the maximum level of the DAG.

A black box AIG (bb-AIG) is a DAG containing AIG nodes as well as "black-box" nodes. A black box node represents an unknown logic function with possibly more that two inputs. An important assumption is that the *black box depends on all of its inputs*.

The proposed *delay* of a black box is[2]:

$$d_{bb}(a)=\left\lceil \log_2(\sum_{i=1}^{n}2^{a_i})\right\rceil .$$

We will show that this is the delay of an *n*-input AND function balanced with respect to *a*. We use this to modify the balance algorithm to be applicable to bb-AIGs as follows: In the balance algorithm, when a black box is encountered, its output arrival time is computed using function d_{bb}. (In general, *balance* can use any black-box delay model whenever a black box is encountered.)

In the rest of this paper, we use (a) d_{bb} as the delay of a black box, (b) *balance* to refer to the balance algorithm modified to deal with black boxes, and (c) $\beta_g(a)$ to refer to the result of applying *balance* to a bb-AIG *g*.

Theorem 1 proves that, for a bb-AIG *g* with *n* inputs,

$$d_{bb}(a)\le\beta_g(a),$$

which implies consistency. It also proves that

$$\exists_g\forall_a, d_{bb}(a)=\beta_g(a),$$

which implies tightness.

3 Lower bound

Notation:

- G^n = the set of all bb-AIGs with *n* leaf nodes
- $a=\{a_1,a_2,\ldots,a_n\}$ arrival time profile (a set of non-negative integers)
- η^m = an *m*-input AND node
- η_2^m = the set of all *m*-input binary AND trees
- $\beta_g(a)$ = $balance(g,a)$ for $g\in G^n$.

Theorem 1. Let $g\in G^n$ and $a=\{a_1,a_2,\ldots,a_n\}$ be the set of arrival times of the leaf nodes. Then

$$d_{bb}(a)\le\beta_g(a).$$

This bound is tight, i.e. $\exists_g\forall_a, d_{bb}(a)=\beta_g(a)$.

The lower bound $d_{bb}(a)$ is equal to what we would get if we treated *g* as a single bb-node with *n* inputs. This is *black box delay* of *g*, whereas $\beta_g(a)$ is the balanced delay of $g\in G^n$. The theorem shows that the use of $\left\lceil \log_2(\sum_{i=1}^{m}2^{x_i})\right\rceil$ at a bb-node in the *balance* algorithm is a consistent lower bound in the sense that whenever a bb-node in *g* is replaced by a bb-AIG (with the same number of inputs), $\beta_g(a)$ can never decrease. Roughly, $\beta_g(a)$ is monotone increasing as a function of the refinement of *g*.

Proof: We first define two transforms and establish some Lemmas.

Single-step expansion. Given $g\in G^n$ and arrival times *a* for its inputs, a *single-step expansion* $y\in G^n$ is obtained by replacing some input s_i of *g* where $a_i>0$, by an η^2 with inputs s_{i_1} and s_{i_2}, whose arrival times are $a_{i_1}=a_{i_2}=a_i-1\ge 0$. The set of arrival times, a', for *y* is obtained by replacing a_i in *a* by the pair (a_i-1,a_i-1).

Complete expansion. Given $g\in G^n$, and arrival times *a* for its inputs, a *complete expansion* is obtained by iterating the single-step expansion for each signal s_i with a positive a_i until each input signal have arrival time 0.

Lemma 1: After a single-step expansion, $\beta_g(a)=\beta_y(a')$ where $y\in G^{n+1}$ is the circuit resulting from the single-step expansion of $g\in G^n$.

Proof: *g* and *y* differ only in that s_i is replaced by an η^2 with $\beta_{\eta^2}(a_{i_1},a_{i_2})=a_i$. Therefore, $\beta_g(a)=\beta_y(a')$.

Lemma 2: After single-step expansion, *g* and *y* have the same value for the computation $\sum_{k\in\text{inputs}}2^{a_k}$.

[2] As an interesting aside, we discuss some properties of the formula in the Appendix.

Proof: For $g \in G^n$, the computation is $\sum_{k=1}^{n} 2^{a_k} = 2^{a_{i_1}} + 2^{a_{i_2}} + (\sum_{k=1}^{n} 2^{a_k} - 2^{a_i})$. The right hand side is the computation for $y \in G^{n+1}$.

Lemma 3: After complete expansion, $\beta_g(a) = \beta_y(0)$, where *y* is the complete expansion of *g*.
Proof: Follows from Lemma 1, by applying it as many times as the number of single-step expansions performed.

Lemma 4: The complete expansion of $g \in G^n$ results in *y*, such that $y \in G^S$ where $S = \sum_{k=1}^{n} 2^{a_k}$.
Proof: Follows from Lemma 2, by applying it as many times as the number of single-step expansions performed. Each single step exxpansion adds one more input. This results in each input with a_i arrival time being replaced by a balanced AND tree with 2^{a_i} inputs.

Lemma 5: There exists $x \in G^n$, such that

$$\forall_a, [\beta_x(a) = \left\lceil \log_2(\sum_{i=1}^{n} 2^{a_i}) \right\rceil].$$

Proof: Let *a* be a the set of arrival times and let $N \in \eta_2^n$. Let $N_B \in \eta_2^n$ be the tree which results from applying *balance* to *N*. Perform the complete expansion of N_B with respect to *a* to obtain $Y_B \in \eta_2^S$ where $S = \sum_k 2^{a_k}$ by Lemma 4. By Lemma 3, $\beta_{N_B}(a) = \beta_{Y_B}(0)$. We claim that Y_B is already balanced and as such, $\beta_{Y_B}(0) = \lceil \log_2(S) \rceil$. Thus $x = N_B$ proves

$$\exists_{x \in G^n}, \beta_x(a) = \left\lceil \log_2(\sum_{i=1}^{n} 2^{a_i}) \right\rceil.$$

Since *a* was arbitrary, this proves the result for all *a*.
QED.

Lemma 6: $\forall_{g \in G^n}, \beta_g(a) \geq \left\lceil \log_2(\sum_{i=1}^{n} 2^{a_i}) \right\rceil$

Proof: By contradiction. Assume $x \in G^n$ has $\beta_x(a) < \left\lceil \log_2(\sum_{i=1}^{n} 2^{a_i}) \right\rceil$. The complete expansion of *x* yields $y \in G^S$. Since $\beta_x(a) = \beta_y(0) < \left\lceil \log_2(\sum_{i=1}^{n} 2^{a_i}) \right\rceil$ by Lemmas 3 and 4, we have $y \in G^S$ with $\beta_y(0) < \lceil \log_2(S) \rceil$. If we replace each 2-input node of *y* by η^2 and each black box by $n \in \eta_2^m$, we get an *S*-input AND function *h*. Each replacement can never increase $\beta_y(0)$, so $\beta_h(0) \leq \beta_y(0)$. During the *balance* algorithm, *h* into a balanced AND tree, also with *S* inputs. This is a contradiction because any balanced AND tree *B* has delay $\beta_B(0) = \lceil \log_2(S) \rceil$. **QED**

Lemma 6 proves the bound and Lemma 5 proves the tightness. **QED.**

4 Applications

One application of the delay bound is in the estimation of delay for a hierarchy of logic blocks. The top level is a pure DAG with a mixture of black boxes and perhaps some logic instantiated as AIGs. Each black box might contain another hierarchy when its internal structure is exposed. In fact there might be several internal structures to choose from. We can do a delay trace on the top level using the black-box delay of each black box. The arrival times we obtain are strict lower bounds. This provides an estimate of the location of the critical paths.

Suppose for each black-box, there are several internal DAGs that reflect different ways of implementing the black-box. For example, we might have a node identified as an adder, and a list of possible implementations - carry look-ahead, serial-bit, etc. These structures are generic and do not take into account how they might be re-synthesized to account for arrival times.

The DAG delay computed using balancing for bb-AIGs represents the fastest possible implementation it can have. We can only slow it down by exposing internal black boxes. The problem we want to address is that each box can have a number of internal structures, from zero (implementation does not exist) to a very large number (circuit allows for many restructuring opportunities). So a hierarchy represents an inordinate number of delay-optimization possibilities.

Discussion topics:

a. How do we arrive at an implementation that is a good starting point for synthesis where we can meet our required times with as little area as possible?
b. Can we use the black box delay here in a useful way?
c. Do we need a slow estimator similar to how the black box delay is a fast estimator?
d. One idea is just a factor of 2 over the lower bound, which occurs for an XOR tree. Although this is not a bound, it is maybe a pretty good estimate.
e. If we had both fast and slow estimators, can we use some kind of interval arithmetic to get arrival time intervals? This might be useful to eliminate parts of the hierarchy that can never enter into critical path analysis.
f. Is this useful? Or should we just flatten the entire hierarchy and delay trace on this? The problem with flattening is that each BB can have several structures. Maybe these estimators can trim down the number of possibilities we would have to try.
g. Another extension is to apply all this to cyclic graphs using the ideas of sequential arrival times, where one does also an implicit retiming.

h. Extension to multi-output boxes. For hierarchies with multi-output boxes, we need information for each output which inputs it depends on. this is necessary when you think about an adder, a wide multi-plexer, or a multiplier. Input dependencies differ widely for the outputs.

Another potential application is in the circuit restructuring package, working on multi-input AND and XOR nodes. The formula introduced in Section 2 allows for computing the arrival times of each multi-input node without decomposing it into two-input gates. This way, the delay computation is accurate while the decompositions of multi-input nodes are performed to achieve different synthesis goals, such as to increase logic sharing.

A similar approach can be used in a technology mapper whose subject graphs contains multi-input AND/XOR nodes. The delay of each such node can be computed in terms of K-input LUTs by a formula similar to that given in Section 2. This way, delay computation is accurate while the node is left undecomposed to prevent structural bias.

5 Experimental results

An experiment was performed to compare the speed of computing arrival times at the output of multi-input AND/OR nodes using (1) delay-optimal two-input node decomposition and (2) the proposed analytical method.

The proposed arrival time computation is integrated into the code of SOP balancing [5], which is part of the priority-cut-based mapper *if* [4] in ABC [1][2].

In this application, structural cuts of the circuit are enumerated. For each cut, the Boolean function of the cut root, in terms of its leaves, is computed and its SOP is derived. The arrival time of the cut is found by performing an optimum tree-balancing of each product of the SOP, followed by balancing of the sum. Instead of actually performing the balanced two-input node tree, the arrival time is computed using the formula introduced in Section 2.

Experiments are performed on 10 industrial designs, using IntelCore 2Quad Q9450, 2.66GHz. Only one core is used.

The results are shown in Table 1. The following notation is used in the table:

- Column 1 (Name): the design name.
- Column 2 (AND): the number of AND-gates in the structurally hashed AIG.
- Column 3-4 (LevelBefore, LevelAfter): the number of levels in the AIG before (after) SOP balancing.
- Column 5 (TimeAll): total runtime of SOP balancing implemented as shown in [5] (command *if -g -K 6*).
- Column 6-7 (TimeOld, TimeNew): the cumulative runtime of the arrival time computation using a) the old two-input node decomposition and b) the new proposed method based on the formula introduced in this paper.

The table shows that the delay computation runs 2x faster using the proposed method, compared to the naïve one. In the context of SOP balancing, the cumulative runtime savings are about 7%, because the total runtime of SOP balancing consists of:

- various tasks performed by the mapper (~40%)
- cut computation (~15%)
- truth table computation (~15%)
- ISOP computation (~15%)
- arrival time computation (~15%).

6 References

[1] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[2] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool", *Proc. CAV'10*, LNCS 6174, pp. 24-40.

[3] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *Proc. DAC '06*, pp. 532-536.

[4] A. Mishchenko, S. Cho, S. Chatterjee, R. Brayton, "Combinational and sequential mapping with priority cuts", *Proc. ICCAD '07*, pp. 354-361.

[5] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets, "Delay optimization using SOP balancing", *Proc. ICCAD'11*, pp. 375 - 382.

7 Appendix - Properties of $\log_2(\sum_{i=1}^{n} 2^{a_i})$.

Consider the following *delay mean*:

$$D(x) = \log_2(\sum_{i=1}^{n} 2^{x_i}) - \log_2 n$$

It is a mean in that is obeys most of the properties of means (like arithemetic, geometric and harmonic):

a. *value preserving* $D(x, x, ..., x) = x$

b. *order invariant*
$D(x_1, ..., x_i, ..., x_j, ..., x_n) = D(x_1, ..., x_j, ..., x_i, ..., x_n)$

c. *averaging* $\min(x) \le D(x) \le \max(x)$

d. *homogeneous* under addition $D(b + x) = b + D(x)$ (unlike other means which are homogeneous under multiplication, $M(bx) = bMh(x)$).

This delay mean can be used as an approximate delay of a node that can partially account for re-synthesis using arrival times. Thus, given a node with n inputs and arrival times $a = \{a_1, a_2, \ldots, a_n\}$, its output arrival time will be given by $D(a) + \log_2 n$. $D(a)$ averages the input arrival time spread and $\log_2 n$ can be inerpreted as the node delay. For a 2-input node, we might traditionally use $\max(a_1, a_2) + 1$ for the output arrival time, in contrast to the arrival time proposed in this paper,

$$\log_2(2^{a_1} + 2^{a_2}) = D(a) + 1 \le \max(a) + 1,$$

This difference can account for unequal arrival times at the two inputs. For 2-input nodes, this difference may not be important.

Table 1: Experimental evaluation of delay computation using the proposed formula.

Name	AND	LevelBefore	LevelAfter	TimeAll, s	TimeEvalOld	TimeEvalNew
designA	144,231	155	83	77.29	10.87	6.21
designB	81,935	33	13	13.86	2.06	0.96
designC	142,043	98	61	108.36	16.06	8.39
designD	188,019	53	28	121.18	16.55	8.50
designE	207,453	80	33	102.17	14.16	7.56
designF	414,600	80	33	203.26	27.30	15.60
designG	134,275	67	31	55.88	7.49	3.86
designH	177,021	99	60	93.45	13.17	7.57
designI	106,519	23	12	19.20	2.26	1.53
designJ	47,779	52	26	12.44	1.59	1.11
Geo mean		1.000	0.494	1.000	0.136	0.076