Timing Model Extraction of Hierarchical Blocks By Graph Reduction

Cho W. Moon
Cadence Design Systems
15015 Avenue of Science
San Diego, CA 92128, USA
001-858-613-2714
cmoon@cadence.com

Harish Kriplani
Cadence Design Systems
555 River Oaks Parkway
San Jose, CA 95134, USA
001-408-944-7822
kriplani@cadence.com

Krishna P. Belkhale
Cadence Design Systems
555 River Oaks Parkway
San Jose, CA 95134, USA
001-408-895-3268
belkhale@cadence.com

ABSTRACT

Timing model extractor builds a timing model of a digital circuit for use with a static timing analyzer. This paper proposes a novel method of generating a gray box timing model from gate-level netlist by reducing a timing graph. Previous methods of generating timing models sacrificed accuracy and/or did not scale well with design size. The proposed method is simple, yet it provides model accuracy including arbitrary levels of latch time borrowing, correct support for self-loop timing checks and capability to support timing constraints that span multiple blocks. Also, cpu and memory resources required to generate the model scale well with size of the circuit. We were able to extract a model for a 456K gate block using under 2 minutes of cpu time and 464 MB of memory on a Sun Fire 880 machine. The generated model can provide a capacity improvement in timing verification by more than two orders of magnitude.

1. INTRODUCTION

Timing extraction or block characterization refers to the process of creating a timing model of a digital circuit for use with a static timing analyzer. Timing extraction plays an important role in hierarchical top-down flow and bottom-up IP authoring flow by reducing the complexity of timing verification and by providing a level of abstraction which hides the implementation details of IP blocks.

Three most desired features in timing extraction are accuracy, efficiency, and usability. The model must preserve the timing behavior of the original circuit and produce accurate results including correct transparent latch behaviors and timing violations. The model also needs to be efficient in terms of the resources needed to generate the model and in terms of the final model size. Finally, the model must be easy to use with existing static timing analyzers. This includes easy model instantiation, easy-to-follow timing reports and capability to allow original constraints to remain valid after a block is replaced by the model.

In this paper, we present a simple method to generate a timing model by reducing the timing graph. The method can generate a model that is accurate within a specified tolerance and is very efficient. Also, our formulation makes it easy to preserve and apply the original timing constraints. We view the timing constraints as part of the model, and generate a set of new timing constraints that can be applied automatically as part of the model extraction process. The support for timing constraints is very critical for top-down hierarchical flows.

The outline of the paper is as follows. Section 2 describes the related work in this area. Section 3 describes the graph reduction algorithm in detail and provides a simple example of graph reduction in action. Model size reduction by using anchor points is described in section 4, followed by some experimental results in section 5. Section 6 concludes with a summary and future work.

DAC 2002 , June 10-14, 2002, New Orleans, Louisiana., USA.

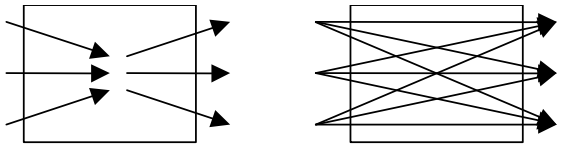
2. RELATED WORK

The timing model can be classified into two types: black box model and gray box model. Black box models have no internal visibility into the block: all the timing information relates to the pins at the boundary of the block. Gray box models, on the other hand, have internal pins that can have delay arcs and check arcs associated with them.

2.1 Black Box Models

One of the well known black box model generators comes from Pearl[1]. Users supply a set of input slew values and output load values and the tool performs path tracing to determine all port-to-port path delays and relevant timing checks. Although black box models have been used widely for many years, they suffer from the following drawbacks:

1. The model size can be larger than the original timing graph size. Figure 1 shows that the removal of internal pin can lead to more delay arcs (from 6 to 9) and a larger model.



Original timing graph w/ 6 arcs Black box model w/ 9 arcs

Figure 1: Black box models can be larger than original netlist

2. Only limited latch behavior can be modeled. The model can capture the latch time borrowing behavior of the original netlist for some given clock waveforms. If the clock waveforms change after the model is extracted, the model becomes invalid.
3. Support for assertions is limited, even for those assertions that are fully contained in the block. Only assertions that originate from and terminate at boundary ports can be fully supported. For example, multi cycle paths that do not originate from or terminate at boundary ports cannot be supported. Also, assertions that span multiple blocks cannot be supported. For example, consider a multi cycle path that

originates from block A and terminates in the middle of block B. The black box model for block B cannot support this multi cycle path assertion.

Clock characterization work of [2] addressed some of the drawbacks of the black box model by capturing register/latch to register/latch constraints in a clock feasibility graph. The resulting model consists of a black box for port-to-port path delays plus timing checks and a feasibility region in terms of clock pulse widths and clock edge separation. A separate checker external to the standard timing analysis tool is needed to verify if clock waveforms lie within the feasibility region. This extension allows for limited context insensitivity to clock waveforms in modeling latch behavior. Latch model is limited because increasing the level of transparency incurs some runtime penalty. Allowing arbitrary levels of transparency becomes prohibitively expensive. Also, since the basic underlying model is a black box, there is no guarantee that the model size is smaller than the original timing graph size. The work did not address the support for assertions.

There is a recent work [3] that uses symbolic simulation to generate black box models. This work is targeted mainly for transistor-level circuits where identifying registers or latches can be difficult. The model generation begins from a set of valid input arrival ranges. Simulation is used to grow the given ranges as much as possible in all input directions. The final input arrival ranges are mapped to setup/hold checks relative to some clock signals. This method does not generate port-to-port path delays and does not consider the dependency on input slews or output loads nor support for assertions.

2.2 Gray Box Models

There is not much published work that can deal with arbitrary levels of transparency in latches using gray box model. Latches can be transformed to registers or combinational gates but such transformations lead to models that are too conservative and do not allow for time borrowing. There is a latch path compression work [4] that collapses latch paths instead of individual latches. The extent of compression is controlled by specifying the desired level of latch transparency. Although it has been successfully used for many latch-based designs, the method cannot guarantee a reduction in model size. In some cases, the model size actually increases after latch path compression[4]. The method does not scale well with the number of latch paths or with the level of latch transparency.

3. TIMING GRAPH REDUCTION

We propose a novel method for generating a timing model by reducing the original timing graph. Timing graph is reduced by iteratively removing pins and merging arcs until no further change is possible. This is similar to the manipulation of timing graph in the context of static circuit optimization [5] but has different objectives and cost functions. The main advantages of this approach are

- it is simple
- it guarantees a reduction in model size
- it allows arbitrary levels of latch time borrowing and scales well with design size
- it allows the original assertions to remain valid after a block is replaced by the model

Latch time borrowing and support for the application of original assertions are accomplished by retaining some internal pins. Latch behavior is preserved by retaining all latch input pins and the corresponding latch output pins. Since all the latches are essentially retained, this allows for arbitrary levels of latch time borrowing as in the original netlist. We found that the number of these internal latch pins is not a limiting factor in model generation time or in model size, even for very large industrial latch-based designs. Also, all internal pins associated with assertions are retained. This enables the assertions to remain valid after a block is replaced by the extracted model in the original context, including assertions that are not completely contained in the block.

The outline of this section is as follows. Timing graph will be defined first. Then models for combinational as well as sequential circuits will be described. The last section discusses the reduction algorithm and gives an example of graph reduction on a simple example.

3.1 Timing Graph

Timing graph G is a three-tuple $G = (P, D, C)$, where P is a set of pins, D a set of delay arcs and C a set of check arcs. Delay arcs and check arcs originate from and terminate at pins: $D \subseteq (P \times P)$, $C \subseteq (P \times P)$. Associated with each delay (check) arc is a transition matrix which defines valid transitions (rising or falling) between the source (signal) pin and the sink (reference) pin. Each check arc has a type such as setup, hold, recovery, removal, etc.

A timing graph representation of a simple D-type register is shown in Fig. 2. Delay arcs are represented as solid arrows and check arcs as dotted arrows. The transition matrix for CLK $\rightarrow$ Q delay arc looks like

CLK	Q	Transition
Rising	Rising	True
Rising	Falling	True
Falling	Rising	False
Falling	Falling	False

Delay (check) values are associated with delay (check) arcs. Delay or check values can be linear functions, lookup tables or delay equations.

3.2 Combinational Models

For the sake of exposition, timing models can be divided into two parts:

- Combinational part deals with interaction among delay arcs and are captured in the combinational model.

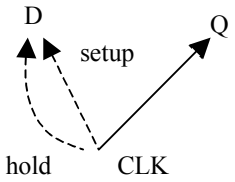


Figure 2: Timing graph for a D-type register

- Sequential part deals with interaction between delay arcs and check arcs. Such interactions are captured in the sequential model.

The two main operations in the combinational model that reduce the timing graph are

- s-merge (serial merge) and
- p-merge (parallel merge).

s-merge takes two delay arcs (*d1* and *d2*) in series and creates a new arc (*d3*) from the source of the first arc (*d1*) to the sink of the second arc (*d2*) to represent the "sum" of *d1* and *d2*. For example, if arc *d1* has a constant delay of 1 and *d2* a delay of 2, then *d3* will have a delay of 3. p-merge takes two arcs (*d1* and *d2*) in parallel (sharing both source and sink pins) and updates *d1* to represent the "worst" of *d1* and *d2*. For example, if *d1* has a constant delay of 3 and *d2* a delay of 4, *d1* will be updated to have a worst delay of 4.

3.2.1 s-merge (serial merge)

s-merge is a fundamental reduction operation that allows removal of pins from a timing graph. Whenever s-merge is performed, arc delays need to be computed. For example, consider two arcs *arc1* and *arc2* which are in series having the following delay values:

arc1		
Input Slew	Output Slew	Arc Delay
is1	os1	d1

arc2		
Input Slew	Output Slew	Arc Delay
os1	os2	d2

The new arc *arc3* resulting from s-merge between *arc1* and *arc2* will have the following delay:

arc3		
Input Slew	Output Slew	Arc Delay
is1	os2	d1 + d2

Note that we use "lazy" slew computation so that output slew and arc delay are computed only for new input slew values. Also, the load-dependent delay computation is confined to the last arc that drives an output port.

Accuracy in s-merge operation is controlled by the number of input slews/output loads that are retained for delay arcs that originate from input ports or terminate at output ports. For table-based libraries, the break points in the table provide the initial range of input slews and output loads. For linear or equation-based libraries, such breakpoints can be characterized. There is no loss in accuracy in s-merge as long as delay calculation is performed using the same input slews and output loads from the initial breakpoints. From the initial set of input slews and output loads, some slew/load values can be removed if such removal does not lead to errors exceeding some desired level. For example, consider a set of three input slews {*s1*, *s2*, *s3*} for a delay arc originating from an input port. Slew value *s2* can be removed from the set if the error between the delay specified for *s2* and the delay computed by interpolating between *s1* and *s3* is within the desired tolerance. Initial errors in slews tend to decrease after several s-merge operations in CMOS designs.

3.2.2 p-merge (parallel merge)

p-merge leads to a substantial reduction in model extraction time as it reduces the number of arcs which need to be processed in the next stage of computation. Given two parallel delay arcs, p-merge chooses the "worst" of the two arcs and allows the other arc to be discarded. Consider two parallel arcs, *arc1* and *arc2,* with the same timing characteristics as in section 3.2.1 except that both have the same input slews *is1* (assume $d1 < d2$).

If a late path is of interest, p-merge picks output slew and arc delay from *arc2* to update *arc1*. The resulting delay for *arc1* becomes

arc1		
Input Slew	Output Slew	Arc Delay
is1	os2	d2

Conversely, if an early path is of interest, *arc1* will remain unchanged.

Note that p-merge only preserves the "worst" (latest and earliest) port-to-port delays.

3.3 Sequential Models

Sequential models involve delay arcs and check arcs. All check arcs can be classified into two main groups. One group is called the "setup" group where the data signal is expected to arrive before the reference or clock signal. Examples of "setup" group are setup, recovery, skew, clock separation, etc. The other group is called the "hold" group where the reference signal is expected to arrive before the data signal. Examples of "hold" group are hold and removal. To preserve latch behavior, all latch input pins and latch output pins are retained but all registers are removed. The operations associated with the removal of sequential elements are similar to those for combinational models. To correctly model the interaction between delay arcs and check arcs, two different s-merge operations are used: forward s-merge and backward s-merge. Sequential p-merge is essentially the same as combinational p-merge. The only difference is that only check arcs of the same type can be merged.

3.3.1 Forward s-merge

Forward s-merge operation is used when register/latch clock pins are removed. When a delay arc and a check arc meet at a clock pin, a new check arc is created to represent a new check value between the old data signal (signal end of the check arc) and the new clock signal (source of the delay arc). Let d denote the delay value on the delay arc and c the check value on the check arc. For setup (hold) group, the new check value becomes $c - d$ ($c + d$).

3.3.2 Backward s-merge

Backward s-merge operation is used when register input pins are removed. When a delay arc and a check arc meet at a data pin, a new check arc is created to represent a new check value between the source of the delay arc and the old clock signal (reference end of the check arc). For setup (hold) group, the new check value becomes $c + d$ ($c - d$).

3.3.3 Self-loop Check Arcs

There are some special check arcs where both the signal end and the reference end point to the same pin. Such timing checks include, but are not limited to, minimum pulse width (MPW) and minimum period (MP) checks on clocks. By using forward and

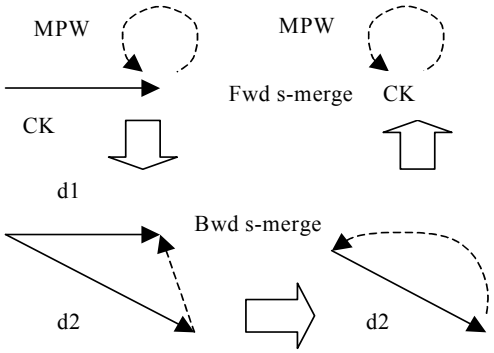


Figure 3: Modeling self-loop check arcs

backward s-merge operations, such self-loop check arcs can be modeled with correct clock path delays (including asymmetrical rise/fall) and slew propagation.

Self-loop check arcs may be "expanded" as a check arc with no self-loop by conceptually duplicating the incoming delay arc. For example, consider the timing graph fragment for some clock path with a MPW check arc in Fig. 3. The delay arc is duplicated to produce two arcs *d1* and *d2* and now check arc is no longer a self-loop arc. First, backward s-merge is performed on delay arc *d1* and the check arc. Then, forward s-merge is performed on delay arc *d2* and the check arc from the previous step. This leads to another self-loop check arc at port CK that correctly reflects the clock path delay and the slew propagation. Note that no delay arcs or pins need to be duplicated here. The duplication in Fig. 3 is for the sake of exposition only.

3.4 Graph Reduction Algorithm

We start with a timing graph that is acyclic. The timer that builds the timing graph is responsible for breaking combinational cycles. The graph is reduced one pin at a time by visiting each internal pin in breadth first search (BFS) order. The reduction is repeated until no further changes are possible. Fig. 4 shows the core loop of the algorithm. `removePin()` is the main routine which

```
reduceGraph(graph){
  changed = 1;
  while (changed) {
    changed = 0;
    for each pin of graph in BFS order {
      if (removePin(pin)) {
        changed = 1;
  } } }
  postProcess(graph);
}
```

Figure 4: reduceGraph() code

performs all the merge operations. Pseudo code for `removePin()` is given in Fig. 5. `postProcess()` takes care of delay arcs or check arcs which originate from primary input pins (or preserved internal pins) and terminate at primary output pins (or preserved internal pins). For such arcs, delay computation routines need to be called explicitly as no s-merge or p-merge operations are done. A pin cannot be removed if it is associated with assertions or if it is needed as an anchor point. The preservation of internal pins will be discussed in section 4. Self-loop check arcs are processed as check arcs in Fig. 5.

```
removePin(pin){
 if (!canRemovePin(pin)) {
    retainPin(pin);
    return 0;
 }
 for each incoming delay arc d1 to pin {
    for each outgoing delay arc d2 from pin {
      d3 = s-merge(d1, d2);
      for each delay arc d parallel to d3 {
        p-merge(d3, d);
        delete d;
    } }
    for each check arc c1 having pin as sig {
      c3 = backward-s-merge(d1, c1);
      for each check arc c parallel to c3 {
        p-merge(c3, c);
        delete c;
    } }
    for each check arc c2 having pin as ref {
      c4 = forward-s-merge(d1, c2);
      for each check arc c parallel to c4 {
        p-merge(c4, c);
        delete c;
  } } }
  delete pin;
  return 1;
}
```

Figure 5: removePin() code

BFS traversal guarantees that

- all the incoming delay arcs to a pin have already been processed before the pin is removed, and
- the delay arcs are characterized with respect to a minimum number of input slew values.

However, BFS traversal does not guarantee that both the signal end and the reference end of all check arcs are processed before forward s-merge or backward s-merge operations. For example, consider the timing graph in Figure 6. When pin *c1* is removed, the range of slew values coming from the signal end (*d2*) of the check arc is not available. In this case, we characterize the check arc with respect to all possible slew values (usually fewer than 6 slew values are sufficient to characterize a check arc). Later when pin *d2* is processed, we re-characterize the check arc for

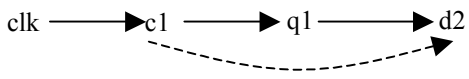


Figure 6: Some check arcs need re-characterization

appropriate slew values.

Fig. 7 gives an example of graph reduction on a circuit with two registers and one latch. The final model includes two internal pins to preserve the latch behavior. Note that the setup/hold checks on *reg1* are modeled as self-loop setup/hold check arcs on clock *C*.

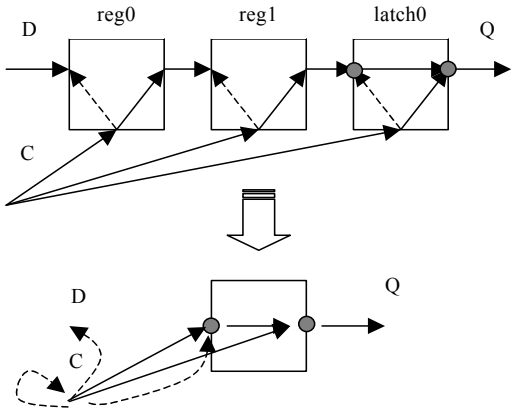


Figure 7: Example of model extraction on a design with two registers and a latch

4. RETAINING INTERNAL PINS

We retain internal pins in the timing graph for several reasons. One reason is to model latch behavior that is correct even in the presence of clock waveform changes. Another reason is to support all original assertions such as multi cycle paths, false paths, generated clocks, etc., including those that span multiple blocks. We retain all internal pins associated with assertions and re-write the original assertions in terms of the retained internal pins. The new assertions become a part of the model and, if the library format allows it, they may even be embedded within the timing model and applied automatically after the model is read. Internal pins retained for these reasons are called *preserved pins*. Another important reason for retaining internal pins is to avoid increasing the model size. Such pins are called *anchor points*. Essential to supporting both preserved pins and anchor points is the `retainPin()` operation whose pseudo code is given in Figure 8. Associated with each retained pin is a range of input slew values which are obtained from the incoming delay arcs.

```
retainPin(pin) {
  for each incoming delay arc d of pin {
    compute_delay (d);
  }
  combine all incoming input slews at pin;
  for each outgoing delay arc d from pin {
    compute_delay(d);
} }
```

Figure 8: retainPin() code

4.1 Anchor Points

Anchor points are existing internal pins which, if removed, may lead to an increase in model size. To estimate the model size, we use the number of delay arcs. We assume that all the delay arcs make equal contribution to the final model size. Given this assumption, the identification of anchor points is similar to the problem of finding a set of pruning pins in the context of static circuit optimization [5]. The objective here is to minimize the number of delay arcs, not the number of variables. Also, since we want to characterize the delay arcs with respect to a minimum number of input slew values, we do not want to remove internal pins at random without first processing their incoming delay arcs. This constrains the order in which anchor points are identified and retained.

We define *gain* at an internal pin as

(#incoming delay arcs × #outgoing delay arcs) –

#incoming delay arcs – #outgoing delay arcs.

This represents an increase in the number of delay arcs if the pin is removed. For example, in Figure 1, the number of delay arcs is 6 before the removal of the internal pin. If the pin is removed, the number of delay arcs becomes 9. Thus, the gain is 3. Any pin that has a positive gain and is *observable* becomes a candidate for anchor points. A pin is *observable* if there exists a path from the pin to a primary output or to a preserved pin. Observability prevents anchor points from being formed in the transitive fanin cone of pins that will eventually be removed. For example, register input pins are not observable and are eventually removed. Anchor points with a gain value greater than or equal to some threshold are identified by `canRemovePin()` in Figure 5. Once anchor points are identified, they are valid for a particular pass. One pass constitutes a complete sweep of all internal pins. Anchor points must be re-identified in the subsequent passes. Table 1 presents the results of varying the gain threshold on a simple 32-bit combinational multiplier. The cpu time and

Table 1: Results of Varying Gain Parameter

Gain	cpu sec	mem MB	#anchor points	#delay arcs	Model size KB	#passes
1	2.7	10.6	359	3312	896	5
2	2.8	10.9	244	3295	1024	4
3	2.9	11.0	178	3411	1095	4
5	2.8	11.0	176	3472	1104	4
50	4.0	12.4	49	5210	2089	3
1000	5.9	14.6	0	3104	1543	2

memory represent the runtime and the memory needed to extract the model on a 750MHz Sun Fire 880 machine running Sun OS 5.8. With a very high value of gain threshold (like 1000), we can remove all anchor points. We provide an option to change the gain threshold but the default is set to 1 and the guarantee in model size reduction compared to the original timing graph only comes with a threshold value of 1. We found this greedy 1-gain heuristic to be very effective.

4.2 Limitation of Anchor Point Heuristic

Although the multi-pass greedy heuristic is fast and effective, this does not guarantee an optimal solution. As a result, the greedy method may produce a slightly larger model than an approach that has a more global view of the graph. Fig. 7 shows an example of such sub-optimality for the 1-gain greedy heuristic. The greedy heuristic fails to delete pins *D* and *E* because the removal of either pin leads to an increase in the number of delay arcs. However, if both pins are removed, the total number of delay arcs would go down. We have not found this to be a big limitation in more than 70 industrial designs that we have tested so far. Nevertheless, we are investigating ways to remedy this limitation.

Table 2: Timing Model Extraction Results

ckt	#gates	#regs& latches	#I/O ports	#int pins intial	#arcs initial	#int pins final	#arcs final	#passes	Model extraction time / mem (sec / MB)	Pre-extraction timing analysis time / mem (sec / MB)	Post-extraction timing analysis time / mem (sec / MB)	model size (MB)
ex1	108K	1855	133	116079	157660	215	1452	3	20.8 / 62.4	11.5 / 43.8	1.5 / < 1.0	1.3
ex2	129K	7	2322	117959	279820	12703	54803	3	41.3 / 126.1	17.5 / 51.0	6.1 / < 1.0	46.4
ex3	307K	7742	777	320697	514033	21	636	3	113.4/162.9	30.3/ 109.1	1.2 / < 1.0	1.0
ex4	456K	17949	180	467049	649922	204	545	4	109.4/463.6	56.1/ 235.1	1.3 / < 1.0	0.3

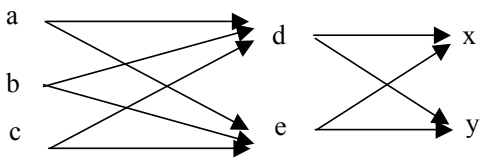


Figure 9: Greedy heuristic fails to remove d & e pins

5. EXPERIMENTAL RESULTS

Table 3: Results showing accuracy of models

ckt	Timing analysis on original netlist		Timing analysis on extracted model	
	Early slack	late slack	early slack	late slack
ex1	-11857.34	-11857.34	-11857.34	-11857.34
ex2	-3.74	-87.37	-3.76	-87.35
ex3	-1.85	-1.85	-1.85	-1.85
ex4	-7.47	-7.47	-7.47	-7.47

Tables 2 and 3 present results of graph reduction on four industry designs. Multi-pass 1-gain greedy heuristic was used on all of them. Reported cpu times and memory usage are on 750MHz Sun Fire 880 machine running Sun OS 5.8. The number of internal pins (final) includes all preserved pins and anchor points. The models were written out in Cadence Timing Library Format (TLF) version 4.4. The runtime includes characterization of clock networks using insertion delays.

The resources needed to generate a timing model are about 2-4x in terms of cpu time and 1.4-2.5x in terms of memory usage compared to performing static timing analysis. This is very efficient since static timing analysis is performed using only a single input slew value and a single output load value, whereas model extraction considers, on the average, about 4 input slew values and 4 output load values. Timing analysis was performed by examining 1000 worst paths in both late mode and early mode (with a maximum of 500 reported paths per endpoint). All the worst late paths and the worst early paths in the model match those in the original circuit within 1% (the specified tolerance value) as can be seen in Table 3.

The design *ex2* led to an out of memory error (> 4 GB) for a black box generator based on [1] and a model size of 5 GB for a gray box generator based on [4]. The design has only 7 latches and the logic is mostly combinational in nature. We identified several anchor points with gain values of 395 and 384. Removal of such anchor points leads to long runtime, large memory usage and large models. We produced a 1% accurate model in 41.3 sec that has a reasonable size (46.4 MB) for this design.

As can be seen from Table 2, extracted timing model leads to significant gains in capacity in terms of both runtime and memory usage. If netlists are replaced by timing models, we gain up to 43x in terms of runtime and up to 235x in terms of memory usage in timing verification.

6. CONCLUSIONS AND FUTURE WORK

We have presented a novel method of generating a timing model by graph reduction. This method is simple and is shown to be quite effective in increasing the capacity of timing verification. The generated model provides a plug-in replacement capability for blocks for static timing analysis and it is well suited for top-down hierarchical flows. It takes 1.4-4x more computational resources to generate an accurate model than to perform timing analysis. The method guarantees a reduction in model size compared to the original timing graph and lends nicely to a formulation where constraints that span multiple blocks can be supported. Also, the model allows for arbitrary levels of latch time borrowing.

As discussed in section 4.2, the greedy heuristic may not always produce an optimal solution. Future research will examine ways to further minimize the model size. Also, work is under way to provide support for bottom-up IP characterization flows and an option to generate black box models for applications that are not capable of reading gray box models.

7. REFERENCES

[1] Cherry, James J. Pearl: A CMOS Timing Analyzer, Design Automation Conference, (1988), 148-159.

[2] Venkatesh, S.V., Palermo, R., Mortazavi, M., Sakallah, K.A. Timing Abstraction of Intellectual Property Blocks, Custom Integrated Circuit Conference (1997), 99 – 102.

[3] McDonald C.B., Bryant, R.E. A Symbolic Simulation-Based Methodology for Generating Black-Box Timing Models of Custom Macrocells, ICCAD (2001), 501 – 506.

[4] Segal, Russell B. Extracting accurate and efficient timing models of latch-based designs, U.S. Patent 5,790,830 (August 4, 1998).

[5] Visweswariah, C., Conn, A.R. Formulation of Static Circuit Optimization with Reduced Size, Degeneracy and Redundancy by Timing Graph Manipulation, ICCAD (2000), 244 – 251.