Automated Timing Model Generation

Ajay J. Daga, Loa Mize, Subramanyam Sripada, Chris Wolff, Qiuyang Wu
Synopsys, Inc.
2025 NW Cornelius Pass Rd., Hillsboro, OR 97124
503-547-6900
{ajdaga, loa, ssubram, wolff, qwu}@synopsys.com

ABSTRACT
The automated generation of timing models from gate-level netlists facilitates IP reuse and dramatically improves chip-level STA runtime in a hierarchical design flow. In this paper we discuss two different approaches to model generation, the design flows they lend themselves to and results from the application of these model generation solutions to large customer designs.

Categories and Subject Descriptors
J.6: ComputerApplication.CAD.

General Terms
Design, Performance, Algorithm, Verification

Keywords
Static Timing Analysis, Model Generation. EDA.

1. INTRODUCTION
System-on-a-chip (SoC) design is performed in an inherently hierarchical manner. Intellectual property (IP) is integrated on-chip with new blocks that are often designed by geographically dispersed teams. Timing models are important in such a design flow because they are a compact means of exchanging the interface timing information for blocks. The automatic generation of timing models from the gate-level netlist for a design is key to sustaining a hierarchical SoC design flow for two reasons:

1) To improve capacity and runtime of static timing analysis (STA) in a hierarchical design flow.

   Timing models are compact and accurate representations of the timing characteristics of a block. The use of timing models in place of the full gate-level netlist for a block is key to improving performance and capacity associated with chip-level STA. This is particularly the case for complex SoC designs whose gate counts exceed 5M gates.

2) To disseminate interface timing information for intellectual property (IP) blocks while hiding implementation details.

   From an IP reuse standpoint, timing models help protect IP by limiting the visibility that end users have to the implementation details of IP blocks, while still preserving information required to verify IP integration on a SoC.

DAC'02, June, 2002, New Orleans, LA

Automatic model generation is possible in PrimeTime [1] using two approaches: extracted timing models (ETMs) and interface logic models (ILMs). ETMs abstract the interface behavior of a design and replace the original netlist with a library cell that contains timing arcs between pins. These setup, hold and delay arcs are a function of input transition time and output load - for this reason the model is context independent. ILMs discard the netlist, constraints and back-annotation associated with internal register-to-register paths on a design. Rather than abstract, ILMs preserve only, but in entirety, the portion of a design that impacts and is impacted by the external world. Like ETMs, ILMs are also context independent.

For design styles where the transitive fanout of input ports and the transitive fanin to output ports are registered, ILMs offer significant runtime and capacity improvements without sacrificing accuracy. For this reason, ILMs are well suited for hierarchical timing signoff. ETMs offer significant performance improvements over the original netlist, but because circuit timing is abstracted there is some loss of accuracy. For this reason, ETMs are well suited during the early stages of a design flow, where fast exploration of design alternatives is important. Also, because they hide all implementation details, ETMs lend themselves to IP reuse scenarios. Finally, because the generated model is a library cell containing timing arcs in industry-wide standards such as lib [2] and Stamp [3], ETMs are portable and easy to use across EDA tools.

In the following sections we discuss the ILM and ETM model generation approaches in further detail. We then discuss issues related to the validation of a model against the original gate-level netlist. Finally, we present results on the application of ILMs and ETMs to ten large customer designs.

Prior work in the area of automated timing model generation has focussed on the automatic derivation of block clocking requirements [4] and on the characterization of interface delays without traversing false paths using user-specified mode information [5]. For mainstream IP reuse and hierarchical STA flows, our experience indicates that customers already have a good understanding of the clocking requirements for a block and so the automatic derivation of this information is not crucial. The main contribution of this paper is its comprehensive study, using large customer designs (some exceeding 700K gates), of model generation runtime, model accuracy and model performance for two fundamentally different model generation approaches.

2. INTERFACE LOGIC MODELS
ILMs embody a structural approach to model generation, where the original gate-level netlist is replaced by another gate-level netlist that contains only the interface logic of the original netlist. Interface logic contains all circuitry leading from I/O ports to

edge-triggered registers called interface registers. The clock tree leading to interface registers is preserved in an ILM. Logic that is only contained in register-to-register paths on a block is not in an ILM.

2.1 ILM Generation

ILM generation requires identifying the interface logic on a design and then writing out the netlist, constraints and back-annotations for interface logic in an appropriate format.

The following is identified as belonging to interface logic:

1) All cells contained in timing paths leading from input ports to either an edge-triggered register or output ports at which these paths terminate. If a transparent latch is encountered then it is treated as combinational logic and path tracing continues through the latch until an edge-triggered register is encountered.
2) All cells contained in timing paths leading to output ports from either edge-triggered registers or input ports at which these paths originate. Transparent latches are handled as described previously.
3) The clock tree that drives interface registers is a part of interface logic. This includes any registers in the clock tree. Clock-gating circuitry that is driven by external ports is a part of interface logic, while clock-gating circuitry that is driven by registers on the block is not a part of interface logic.

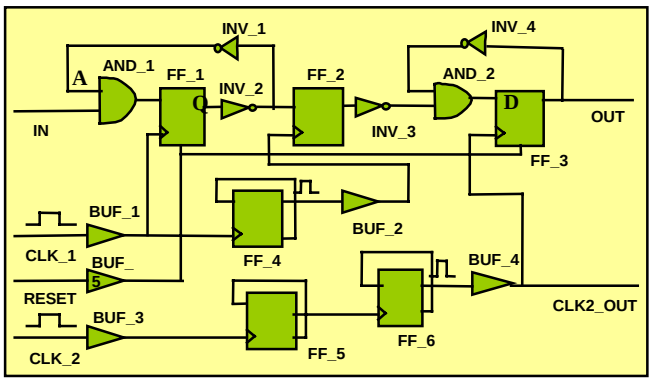

Figure1: Example gate-level netlist.

Figure 1 shows an example gate-level netlist for a block (ports and pins on which clocks are defined are marked ⎍) and Figure 2 its resulting interface logic. Note how cells `INV_*`, `FF_2`, and `AND_2` are not a part of the ILM. These cells are only instantiated in register-to-register-paths. Note how the cell `AND_1` is a part of the ILM though it does have an input pin, `A`, that belongs to a register-to-register path. The input pin `A` on `AND_1` is unconnected in the ILM. Similarly, the output pin `Q` on `FF_1` and the input pin `D` on `FF_3` are also unconnected in the ILM. `FF_1` and `FF_3` are interface registers. Notice how the clock tree for these interface registers is a part of the ILM. Registers `FF_5` and `FF_6` that belong to this clock tree are also a part of the ILM, but notice that the feedback paths on these registers are not a part of the ILM. Also notice how cells `FF_4` and `BUF_2` that belong to the clock tree leading to an internal register are not a part of the ILM.

When identifying interface logic it is necessary to ignore the fanout of input ports connected to global chip-level signals that feed internal registers on a block. Examples of such ports are clocks, reset and scan_enable. In Figure 1, the fanout of ports `CLK_1`, `CLK_2` and `RESET` are ignored because otherwise all registers on a design will be instantiated in the ILM. The clock tree leading to interface registers is identified as belonging to interface logic by processing the fanin of interface register clock pins.

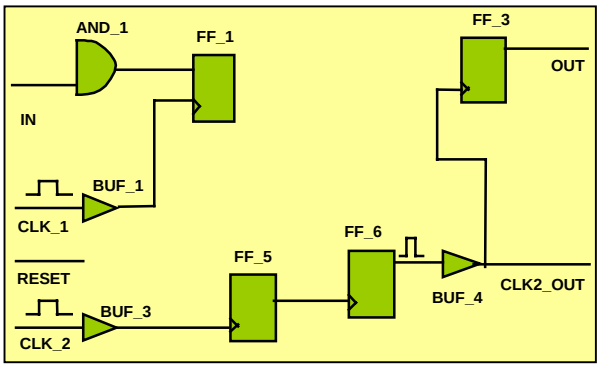

Figure2: ILM for example gate-level netlist.

For transparent latch based designs, it is necessary to specify the maximum number of levels of latch borrowing in I/O paths. By default transparent latches are viewed much like combinational logic cells and fanout or fanin traversal continues through a latch until an edge-triggered flip-flop is encountered. When the number of levels of latch borrowing is specified, fanout or fanin traversal stops at a latch whose level is greater than the specified borrowing level. Specifying one level of latch borrowing, for example, results in the first two latches encountered in an I/O path being a part of the ILM. The first latch can borrow, while the second functions as an edge-triggered register that does not borrow.

When delay calculation is performed using the ILM it is necessary to maintain all pins connected to interface logic nets. This ensures that the pin and wire capacitance for interface logic nets is accurate. The inclusion of all pins on interface logic nets for the ILM in Figure 2 results in the addition of pins `FF_4/CP` and `INV_4/A`.

In SDF flows where cell and net delays are annotated to an ILM it is not necessary to maintain accurate pin capacitance information for a net. Also, if clock tree synthesis has not been performed then non-interface-logic pins are not maintained in the ILM for ideal clock nets.

3. EXTRACTED TIMING MODELS

Extracted timing models differ from ILMs in that the interface logic for a block is replaced by context-independent timing relationships between pins on a library cell. In this section, we discuss the different types of ETMs and issues related to their automatic generation.

3.1 Extracted Timing Model Types

There are two types of extracted models, library cell and wrapper with core cell, illustrated in Figure 3. The library cell ETM replaces an entire design with a single library cell. The extracted library cell contains timing arcs between external pins. Internal pins are introduced only when there are clocks defined on internal pins of the design. The library cell is generated in industry standard formats such as lib and Stamp. In addition to the extracted library cell a constraint file is generated that defines exceptions (false paths and multi-cycle paths) that need to be

applied to the library cell when it is instantiated at a higher level of abstraction.

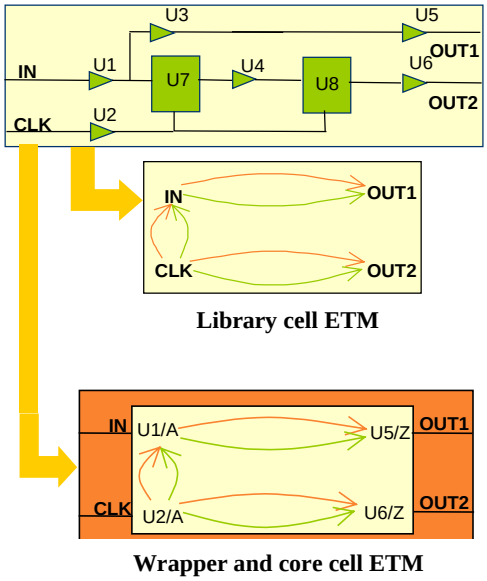

Figure3: Two types of ETMs.

With wrapper and core cell, the extracted model preserves boundary nets and parasitics on boundary nets. This model is more accurate than the library cell because it allows boundary parasitics on blocks to be stitched together with chip-level parasitics when the model is instantiated at a higher level of abstraction. This, in turn, allows inter-block net delays to be calculated with high accuracy by taking into account the complete parasitics for a chip-level net. The wrapper and core cell correspond to a view of the original design in terms of a wrapper design that instantiates a core block, as shown in Figure 3. The extracted core cell is a pin-to-pin replacement of the core block and is instantiated in the extracted wrapper design. Parasitics information is written out in industry standard SPEF [6] format. A constraint file that maintains exceptions that need to be applied to the extracted model is written out just as for a library cell model.

A library cell model is preferred in IP reuse scenarios and for import into third-party simulation and place & route tools. This is primarily because the library cell model is a port-to-port replacement for the original design and is written out in industry standard formats. The wrapper and core cell model is useful for import into tools that can handle the additional level of hierarchy introduced by the wrapper design. The benefit of being able to handle such a model is the additional accuracy it provides.

3.2 Extracted Timing Arcs

The objective of model extraction is to establish context-independent timing relationships between input and output ports on a design. These timing relationships are represented by timing arcs between pins on a library cell. Extracted timing arcs can be classified into three distinct types: combinational delay arcs, sequential delay arcs, constraint arcs.

3.2.1 Combinational Delay Arcs

Combinational paths from an input port to an output port result in the extraction of either two or four combinational delay arcs. These arcs capture the minimum and maximum timing delays for inverting and non-inverting paths from an input port to an output port. Each extracted combinational delay arc has associated with it timing sense information that indicates whether the arc is inverting or non-inverting and whether it is representative of minimum or maximum delay. Minimum and maximum delay arcs represent lower and upper bounds on circuit delay for a single operating condition, and so are indicative of delay variations resulting from circuit topology rather than from process variations. For the design in Figure 3, a pair of non-inverting combinational delay arcs are extracted between IN and OUT1 (there are no inverting paths between IN and OUT1).

3.2.2 Sequential Delay Arcs

Paths to an output port that start at interface registers that are similarly clocked, i.e. by the same clock and by the same clock edge, result in the extraction of a pair of minimum and maximum sequential delay arcs. A set of similarly clocked interface registers is referred to as a clock group. Extracted sequential arcs are relative to a clock and a specific edge of a clock. The clock edge that a sequential delay arc is relative to is a function of whether interface registers in a clock group are rising or falling edge triggered and whether there is inversion in the path leading from the clock definition point to register clock pins. For the design in Figure 3, given that U8 is rising-edge triggered and there is no inversion in the clock tree from CLK to U8/CLK, a pair of sequential delay arcs are extracted from the rising edge of CLK to OUT2. Extracting sequential delay arcs requires summing clock and data path delays for each interface register in a clock group. While minimum and maximum delay arcs are computed for paths from each interface register in a clock group to an output port, only a single worst-case minimum and maximum delay arc is extracted for a output port, clock group pair.

3.2.3 Constraint Arcs

Paths from an input port to the data pins of a clock group result in the extraction of a setup and a hold constraint arc. Similar to sequential delay arcs, these arcs are relative to a specific edge of a clock port. For the design in Figure 3, given that U7 is rising-edge triggered and that there is no inversion in the clock tree leading from CLK to U7/CLK, setup and hold arcs are extracted for IN relative to the rising edge of CLK. Setup and hold arcs are extracted in compliance with the following equations:

1) *Setup = Max data path - Min clock path + Register setup*
2) *Hold = Max clock path - Min data path + Register hold*

As with sequential delay arcs, though setup and hold times are computed for paths from an input port to each interface register in a clock group, only a single worst-case setup and hold arc is extracted for a input port, clock group pair.

3.3 Extracted Timing Tables

Associated with each timing arc are tables that define timing values as a function of input transition time and output load. By extracting timing tables, rather than scalar timing values, the generated model is context independent and applicable for varying transition times and output loads. There could be different critical paths for different transition times and the tables are merged appropriately to handle this.

3.3.1 Delay Tables

Associated with each extracted delay arc (combinational and sequential) are a pair of two-dimensional delay tables and a pair of two-dimensional transition tables. Rise and fall delay tables

define timing delays for rising and falling transitions at an output port as a function of input transition time and output load. Rise and fall transition tables define rising and falling transition times at an output port as a function of input transition time and output load. Both the numbers of points in extracted delay and transition tables and the input transition and output load range over which delay is characterized is controlled by a user.

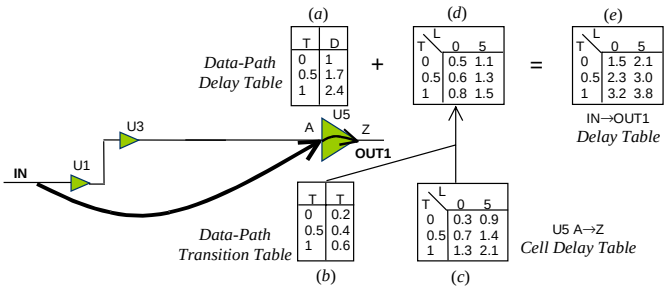

Figure 4: Extracting delay tables.

Consider extraction of the rise delay table for the non-inverting arc from IN to OUT1 in Figure 4 (assume that boundary net delays are zero). A two-dimensional cell delay table (*c*) for U5 defines the rise delay at output pin Z relative to pin A. Model extraction establishes the rise-to-rise (*rr*) transition table (*b*) from IN to U5/A. This table captures transition times at U5/A as a function of transition times on IN. The transition table at U5/A is used to index into the cell rise delay table for U5 to yield an intermediate table (*d*) that describes the delay contributions through U5 as a function of the load on OUT1 and the transition time on IN. This intermediate table is summed with the delay table (*a*) from IN to U5/A that captures the delay at U5/A as a function of transition times on IN. This yields the two-dimensional rise delay table (*e*) for the arc from IN to OUT1.

3.3.2 Constraint Tables

Associated with each extracted constraint arc is a pair of constraint tables. Rise and fall constraint tables define setup or hold times, depending on the type of arc, for rising and falling transitions at an input port. These tables are a function of input transition time and clock transition time.

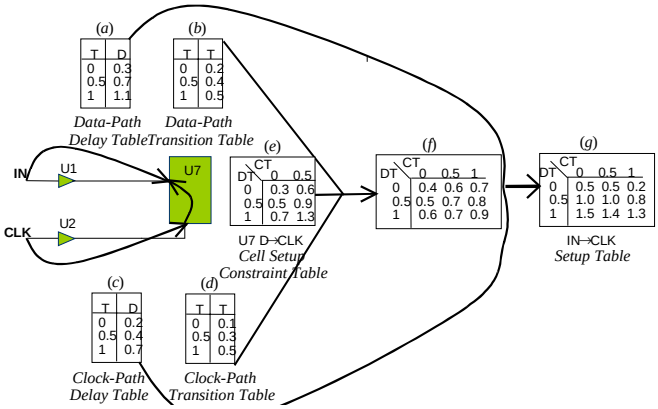

Figure 5. Extracting constraint tables

Consider the computation of the fall constraint table for the setup arc between IN and the rising edge of CLK shown in Figure 5. This computation first requires extracting the transition tables associated with the maximum fall-to-fall (*ff*) path from IN to U7/D (*b*) and the minimum *rr* clock-path from CLK to U7/CLK (*d*). These two tables are used to index into the two-dimensional fall constraint table (*e*) for the setup arc from D to CLK in the library cell for U7. The resulting intermediate table (*f*) defines setup times on register U7 as a function of transition times on IN and CLK. Next, this intermediate table is combined with the delay tables for the maximum *ff* path from IN to U7/D (*a*) and the minimum *rr* path from CLK to U7/CLK (*c*). This is done in accordance with equations in section 3.2.3 to yield the desired fall constraint table for the setup arc from IN to the rising edge of CLK.

3.4 Exception Handling

Exceptions such as false paths and multi-cycle paths (MCPs) are handled during model extraction. Exceptions that only refer to the clocks on a design are simply written out to the generated constraint file and must be reapplied to the extracted model when it is instantiated in a design. False paths that refer to pins or ports on a design are accounted for during model extraction.

Multi-cycle exceptions that are found to apply to all paths leading to or from a port are written out to the constraint file and must be reapplied to the extracted model. The MCP written to the constraint file only refers to pins and clocks on the extracted model and not to internal design pins. For example if an MCP were defined through U1/Z in Figure 6, because it applies to all paths from IN, it would be written out to the constraint file and apply to all paths through port IN that are clocked by CLK.

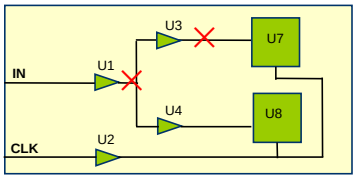

Figure 6: MCP handling during model extraction.

When an MCP only applies to some, but not all, paths to or from a port then the impact of this exception is taken into account when extracting a timing arc. For example, in Figure 6, assume a MCP defined two-cycle clocking through U3/Z. Assume a clock period of 10ns, a data-path delay from IN to U7/D of 17ns and a data-path delay from IN to U8/D of 6ns. Also, assume that the clock-path delay from CLK to the register clock pins is zero and the setup times for both registers is also zero. In this case, a setup arc of 6ns is extracted from IN to CLK as a result of the path from IN to U8/D. A setup arc of 7 ns is extracted from IN to CLK as a result of the path from IN to U7/D after reducing the data-path delay to U7/D (17ns) by the clock cycle shift (10ns). As 7ns is more worst-case than 6ns, a single-cycle setup arc of 7ns is extracted between IN and CLK. So, when a MCP does not apply to all paths to or from a port, then clock periods are taken into account to generate a worst-case arc appropriate for single-cycle clocking of the port.

3.5 Transparent Latch Handling

When the interface logic for a design contains transparent latches, these need to be handled appropriately. Transparent latches allow propagation from the D pin to the Q pin (referred to as a borrowing latch) if the arrival time on the D pin is later than the open edge of its clock. Transparent latches do not permit propagation from the D pin to the Q pin (referred to as a non-borrowing latch) if the arrival time on the D pin occurs before the open edge of its clock. The open edge of a positive level sensitive latch is the rising edge of CLK.

Transparent latches are handled during model extraction by establishing, based on arrival times defined on input ports and specified clock periods, whether a path from an input port to a latch causes borrowing or not. If the path causes borrowing then path traversal continues through the latch until either a non-borrowing latch or edge-triggered flip-flop is encountered. If the path does not case cause borrowing then path tracing stops and a setup arc is extracted relative to the open-edge of the latch. The borrowing behavior of latches does not impact the extracted hold arc. Hold arcs are always extracted relative to the close edge of the first interface register (latch or flip-flop) encountered while tracing paths from input ports.

Latch borrowing can result in paths being traced through latches from an input port to an output port. In this case, a delay arc is extracted from the input port to output port. When tracing paths from interface registers to output ports, transparent latches are handled by tracing through borrowing latches and generating a sequential delay arc from a clock port to an output port. If a non-borrowing latch is encountered then no arc is extracted because this represents an internal register-to-register path.

Any clock cycle shift resulting from borrowing through several levels of latches is not accounted for in the extracted timing arcs and must be accounted for by applying MCPs to the model. Also, the model will match netlist timing as long as the borrowing behavior at the time of model generation continues to hold when the model is used. If input arrival times or clock periods change sufficiently from what was specified at the time of model generation then it is possible for latch borrowing behavior to also change. The model would then no longer be an accurate reflection of original design behavior. In this situation, a new model must be generated for the new input arrival times and clock periods.

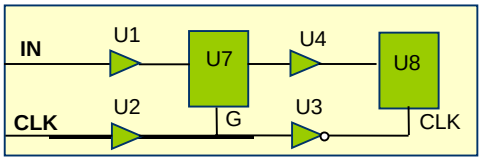

Figure 7: Latch handling during model extraction.

For an illustration of how transparent latches are handled during model extraction consider the design in Figure 7. If the positive-level sensitive latch U7 does not borrow, then a setup time relative to the rising edge of CLK is extracted. The setup time for this arc is influenced by the data-path delay from IN to U7/D and the clock-path delay from CLK to U7/G. If U7 does borrow, then path tracing continues through U7 to U8 (a rising-edge triggered flip-flop), and a setup arc is extracted relative to the falling edge of CLK. This setup arc is impacted by the data-path delay from IN, through U7/D, to U8/D, by the clock-path delay from CLK to U8/CLK and by the setup time between U8/D and U8/CLK.

4. MODEL VALIDATION

Once a model has been generated it is necessary to compare it to the original design and establish that discrepancies between the model and design are within acceptable bounds. This is referred to as model validation and is performed for a given user-specified context (input arrival times and transition times, output loads and clock periods). Model validation takes place by loading the design and model standalone, capturing relevant interface timing parameters for both, and then comparing these parameters. For completeness it is often desirable to validate a model and design at two different contexts; a "min" environment (small loads, fast transition times) and a "max" environment (large loads, slow transition times).

Model validation ensures that timing arc values are within user specified tolerances. This is done by comparing the absolute discrepancy between worst-case timing slacks for all unique input-to-clock, clock-to-output and input-to-output paths on the model and design. This can also be accomplished through an absolute or percentage comparison of the discrepancy between interface timing arc values. Percentage comparison of interface timing slacks is inappropriate because timing slack values are influenced by the specified context (input arrival times, clock periods, etc.). On the other hand, absolute discrepancies between interface timing slacks for the same context applied to model and design can be completely attributed to a discrepancy between model and design. An attractive aspect of comparing interface timing arc values is that such comparisons are independent of some aspects of the user-specified context such as input arrival times and clock periods.

Model validation also ensures that the capacitance values seen at input ports and the transition times at output ports match. This is necessary to ensure that in addition to correctly capturing interface delays and constraints the model presents its external environment with the same input load and output drive characteristics as the original design.

Finally, model validation ensures that design rules such as max transition time on input ports and max load for output ports are the same for both model and design. This ensures that the same design rule violations are reported for model and design.

Once the timing parameters for a model have been established as falling within user-specified tolerances of the original design the model is ready for use in place of the original design.

5. EXPERIMENTAL RESULTS

In this section, we present results from the application of ILMs and ETMs to ten customer designs. The objective was to study model generation runtimes, the performance improvement provided by models in comparison to the original design and the accuracy of the models in comparison to the original design. For all designs a wrapper and core cell ETM was generated. The calculation of delays and critical paths is referred to as “update timing”.

Table 1: Design information.

	1	2	3	4	5	6	7	8	9	10
# Cells	29277	26540	35254	30449	137557	43850	94925	263490	23441	55499
# Latches	240	0	12	3	3807	2225	0	4565	557	0
Back-annotation	RC	RC	RC	RC	RC	RC	-	SDF	-	-
Clock Period (ns)	8.3	8	16	36	26	6	5	6.9	6.66	6
Timing Relns. Verified	356	204	3489	3206	2006	3544	3084	1646	6114	5392

Information on the ten designs is shown in Table 1. Six of the designs had RC back-annotation, one had SDF and the remaining three designs had no back-annotation. Some of the designs had transparent latches in the interface logic while others did not. The number of cells on the designs varied from 20K to 260K - this corresponds to a gate count ranging from about 60K gates to 800K gates. The minimum clock periods on these designs ranged

from 5ns to 36ns, or about 25MHz to 200MHz. The number of interface timing relationships that were compared between model and design, as part of model validation, ranged from 200 to 6000.

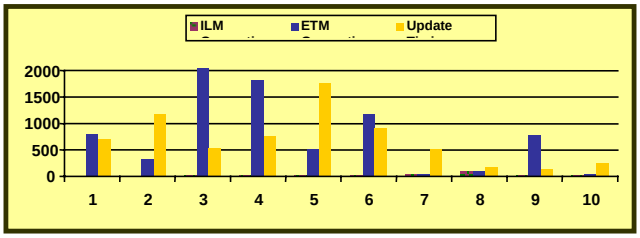

Chart 1: Model generation runtimes.

Chart 1 shows model generation runtimes (in seconds) for the ten designs. As a baseline, the time taken to update the timing of the original design is shown. As may be observed, ILM generation times are insignificant because of the simplicity of the task - simply discard register-to-register logic, constraints and back-annotation and write out what remains. ETM generation times are significant when a design has RC back-annotation because of the effort involved in characterizing circuit delay for multiple input transition times. In some cases ETM generation times were about 7x update timing times, while in others ETM generation was even faster than update timing. On average ETM generation takes about 3x longer than update timing.

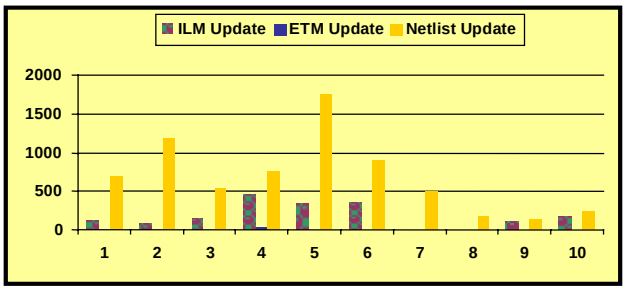

Chart 2. Performance improvements with ETMs and ILMs

Chart 2 compares update timing runtimes on the original design, ILM and ETM. Across the board, update timing on ETMs only took a few seconds, resulting in performance improvements that ranged from 20x to 1200x when compared to update timing on the original design. ILM performance improvements ranged from about 1.5x to 14x. On average, ETM performance improvements were about 250x relative to the original design, while ILM performance improvements were about 5x. This is not surprising given that ILMs preserve all interface logic while ETMs abstract interface logic into timing arcs. The large model generation times spent characterizing design timing has its payoff when the ETM is used in place of the original design.

Charts 3 and 4 measure ETM and ILM accuracy, respectively. These charts show the percentage of verified interface timing relationships that had discrepancies between netlist and model of less than 10ps and between 10 and 100ps. Across all designs ILMs tend to match the original design within 10ps; 98.4% of all interface timing relationships on an ILM matched the netlist within 10ps. This is not surprising given that ILMs preserve interface logic without modification.

On designs with SDF back-annotation, or no back-annotation, ETMs also had greater than 90% of interface timing relationships with a discrepancy of less than 10ps. For designs with RC back-annotation, however, ETMs tend to be within 100ps of the

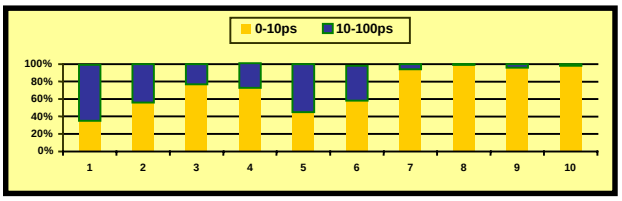

Chart 4: ETM accuracy.

original design. Overall, across all ten designs, 99.6% of interface timing slacks for ETMs were within 100ps of the original design. For clock speeds of 200MHz, 100ps error is 2% of the clock cycle and is insignificant. It is equally likely for model interface relationships to be optimistic or pessimistic when compared to the original design.

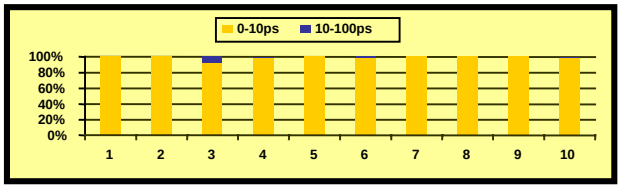

Chart 3: ILM accuracy.

6. SUMMARY

In this paper we have presented two approaches to automated timing model generation from gate-level netlists: interface logic models and extracted timing models. Interface logic models discard register-to-register paths on a design but preserve all interface logic without modification. Extracted timing models abstract interface logic into context-independent timing arcs.

Experimental results show that interface logic models provide 5x runtime improvement while preserving original design timing with no accuracy loss. This makes interface logic models well suited for hierarchical timing signoff. Extracted timing models provide greater than 100x runtime improvements with high accuracy. This makes extracted models well suited for IP reuse scenarios where it is important to hide circuit implementation details and early in the design flow where fast design exploration is important. Our experience indicates that different model generation solutions will continue to be appropriate for different design scenarios.

REFERENCES

[1] Synopsys, "PrimeTime: Synopsys Static Timing Solution", Technical White Paper, July 1999.

[2] Synopsys, "Library Compiler Reference Manual", Version 2001.08, August 2001.

[3] Synopsys, "PrimeTime Modeling User Guide", Version 2001.08, August 2001.

[4] S.V. Venkatesh et al., "Timing Abstraction of Intellectual Property Blocks", CICC'97, May 1997.

[5] H. Yalcin et al., "An Advanced Timing Characterization Method Using Mode Dependency", DAC'01, June 2001.

[6] IEEE Std 1481-1999, "IEEE Standard for Integrated Circuit Delay and Power Calculation System", June 1999.