LIBRARY CHARACTERIZATION AND STATIC TIMING ANALYSIS OF ASYNCHRONOUS CIRCUITS

by

Mallika Prakash

A Thesis Presented to the
FACULTY OF THE USC VITERBI SCHOOL OF ENGINEERING
UNIVERISTY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
MASTER OF SCIENCE
(COMPUTER ENGINEERING)

December 2007

Acknowledgements

I thank my advisor, Prof. Peter Beerel who has been the driving force behind this thesis for being an excellent teacher, mentor and friend, and his wife Janet and baby girl Kira for providing a welcoming and friendly environment to work in. I thank the entire Asynchronous CAD/VLSI Group at USC, Georgios Dimou, Pankaj Golani, Arash Saifhashemi, Roger Su, Gokul Govindu, Amit Bandlish and Prasad Joshi for their support and companionship during the long hours that went into this work and for very useful brainstorming Friday group meetings. Special mention goes to Pankaj, Amit, Prasad and Rahul Rithe (from Indian Institute of Technology, Kharagpur) for their help with characterizing SSTFB cells and debugging the characterization flow and Gokul and Amit for writing and maintaining the TCL script to convert HSpice measure files to the liberty format.

I also thank Fulcrum Microsystems for the use of PCHB and MLD libraries, Andrew Lines especially for his critical feedback on the STA flow, Professor Ken Stevens of University of Utah for his help in formally verifying the PCHB relative timing constraints, Giorgios Dimou for providing the PCHB and MLD test cases for the static timing analysis flow, Marly Roncken at Intel for her technical advice, Silistix UK Ltd. for giving me the opportunity to develop a static timing analysis flow for their asynchronous design methodology as a summer 2007 intern, and my manager and mentor David Jackson and John Bainbridge at Silistix for guiding me towards a successful internship.

This work was in part supported by an SRC Grant CADTS-1425 and by Viterbi School of Engineering Teaching Assistantships. I thank the entire EE-Systems staff, Diane Demetras, Rosine Sarafian, Mandy Sheedy and Annie Yu for entertaining endless questions with patience.

I am grateful to my thesis committee members Prof. Massoud Pedram and Prof. Melvin Breuer for their valuable feedback and my professors at Illinois Institute of Technology, Prof. James Stine, Prof. Donald Ucci and Prof. Jaffar Saniie for cultivating my passion for research and building my technical foundation.

My parents Sunil and Anju and my siblings Soumya and Suryanshu have always supported me and believed in me and I thank them for their unconditional love, faith and advice that can not be matched. My fiancé Gokul has been my greatest inspiration to excel in whatever I do and his unreserved love, trust and confidence in me are unsurpassed.

Table of Contents

List of Figures

List of Tables

Abstract

For main stream acceptance of asynchronous circuits, a mature EDA tool flow is necessary that leverages off commercially available libraries and tools for synchronous circuits. Many asynchronous templates however rely on specialized and complex circuits that are not present in commercial libraries. For such templates, designers either technology map these cells to existing libraries at the cost of area and performance or rely on full-custom design and extensive SPICE simulation to verity timing correctness and performance. This thesis addresses both of these issues by developing both library characterization and static timing analysis flows for non-standard asynchronous circuit templates that together support back-annotated power and timing simulations as well as static timing and performance verification. We first create a fully-automated characterization flow for the static single-track full-buffers template. We then develop a fully-automated static-timing flow for timing and performance verification using the gold-standard commercial tool, Synopsys PrimeTime. The proposed flow is successfully demonstrated on three different asynchronous design styles.

Chapter 1

1. Introduction

Asynchronous design has begun to demonstrate its advantages in the commercial marketplace. There is a growing need for high performance low power circuits in areas such as on-chip interconnect and globally asynchronous locally synchronous (GALS) systems. Many asynchronous circuits fulfill these needs but are not easily accepted by designers due to a lack of support by commercial CAD tools. Asynchronous logic cells not found in standard-cell libraries, such as dual-rail domino logic and dynamic C-elements with staticizers, must be characterized for both timing and power so as to not limit the designers to full-custom design and time consuming SPICE simulations to verify timing and correctness. Library characterization for such non-standard cells enables back annotated Verilog simulations, timing driven place and route and is a pre-cursor to timing sign-off with commercial static analysis tools.

In general, the challenges of characterizing asynchronous cells comes from their more general circuit structure that may include internal combinational loops, bi-directional pins, and mutual exclusion rules on dual-rail or 1-of-N inputs. These input constraints and general structures do not conform to the standard latch or flip-flop templates supported by commercial library characterization tools. Consequently, library characterization has been a limiting factor for otherwise promising asynchronous design styles. In particular, the library characterization challenges are epitomized in the proposed next generation STFB circuit family called static single-track full buffers (SSTFB) [5][6][7]. In [4] layout and abstract views of STFB were developed and used to design a tree adder test chip. This library was extended in [8] to include timing characterization which enabled back-annotated timing analysis, but extensions to SSTFB and verifying power consumption was left as future work. SSTFB was used in [6] to implement a turbo decoder but due to the absence of characterized delay information, the authors were limited to time consuming post-layout simulations in Nanosim. Using a characterized library, timing correctness and performance can be analyzed either through back-annotated simulations or static analysis.

Similarly, the timing constraints necessary for correctness in an asynchronous circuit can be much more complex than setup and hold constraints on flip-flops and latches. In this thesis we adopt the relative timing (RT) approach introduced in [16] for modeling timing constraints necessary for correct operation. Relative timing is premised on the observation that timing correctness typically depends on enforcing a relative ordering between signals that can and should be explicitly identified and verified. Coupled with absolute margins, such RT constraints should be adhered to during synthesis and P&R and then verified post-layout. Previous work in RT such as in [3][13][17][18] present various approaches for using the relative timing approach during synthesis and verification of asynchronous systems. However, the connection between relative timing and post-layout sign-off using static timing analysis has yet to be addressed.

The first half of this thesis addresses the characterization challenges by demonstrating an effective timing and power characterization flow for static STFB cells. It describes the decomposition of SSTFB cell behavior into a set of timing arcs that can be understood by commercial place and route and back annotation tools. It then describes a novel methodology and tool kit to automatically characterize the library and represent the information in the commercially supported Liberty file format. The second half of this thesis focuses on how both relative-timing constraints as well as performance (e.g., throughput) targets can be verified using standard STA tools for three different asynchronous design methodologies. The first is the well-used pre-charged-half-buffer template [10][12], the second is a novel template in which stages consist of multi-level domino-dual-rail logic and a full-buffer handshake controller and the third is a non template-based hierarchical design methodology. We develop fully-automated scripts that verify all relative timing assumptions (e.g., isochronic forks) as well as the gate-level performance of any circuit designed with these templates.

Related work includes the application of STA tools in desynchronization briefly described in [1][3]. The authors describe a fully-automated flow from synthesis to place and route in which an asynchronous design is produced from a synchronous Verilog netlist by replacing each flip-flop with two latches and the clock with handshaking control signals coupled with run-time-configurable matched delay lines. As part of their flow, STA is used to verify correct timing in the specific semi-decoupled four-phase controllers used but extensions to the template-based styles covered in this thesis are not addressed. In addition, the verification is not based on relative timing but rather on max-delay constraints with absolute metrics, which can lead to false negatives. Virtual clocks are added to the design to model the datapath as a conventional master-slave latch-based design with non-overlapping clocks. This guarantees that the latency of the combinational logic is constrained and that the associated delay-lines will be sufficiently long. In addition, the authors do not address how STA could verify the performance of the desynchronized circuit and instead rely on time-consuming back-annotated simulation.

We also note that STA has been integrated into the single-rail back-end design flow developed by Handshake Solutions, but to the best of our knowledge these details are not published and thus a comparison to our approach is not possible.

Chapter 2

2. Background

This chapter presents background material on delay and power characterization, followed by some example circuit families that are used in the characterization and static timing analysis flows.

2.1. Delay and Power Characterization

Both delay and power consumption of a library are characterized and stored in an industry format (Liberty) file. For delay, both pin to pin delays and the corresponding output slopes are typically characterized for identified timing arcs as a function of load and/or input slope. In general, this allows slews to propagate during delay and timing analysis and be used to characterize and analyze power consumption.

For power, both static and dynamic sources of power are characterized. Dynamic power is made up of internal power and switching power. The former is dissipated by the cell in the absence of a load capacitance and the latter is the component that is dissipated while charging/discharging a load capacitance. Dynamic power is measured per timing arc (as with delay). Static dissipation is due to leakage currents through `OFF' transistors and can be significant when the circuit is in the idle state (there is no switching activity). It has four principle sources: reverse-biased junction leakage current, gate induced drain leakage, gate direct-tunneling leakage and subthreshold (weak inversion) leakage. For 180nm, gate leakage is about 0.1% of total static power and subthreshold leakage dominates. The other two components are much smaller and thus generally ignored. With the above simplification, leakage power can be computed as the product of supply voltage and the sub-threshold leakage current. Unlike delay and dynamic power, leakage power is typically represented as a single value per cell.

2.2. Static Single Track Full Buffers

Static single track full buffer (SSTFB) is an asynchronous design template that uses dual rail domino pre-charged by an asynchronous control signal instead of a clock. It uses a 2-phase handshaking protocol where the sender pulls the channel high and the receiver pulls the channel low. Figure 2-1 shows the general structure of a static single track full buffer.

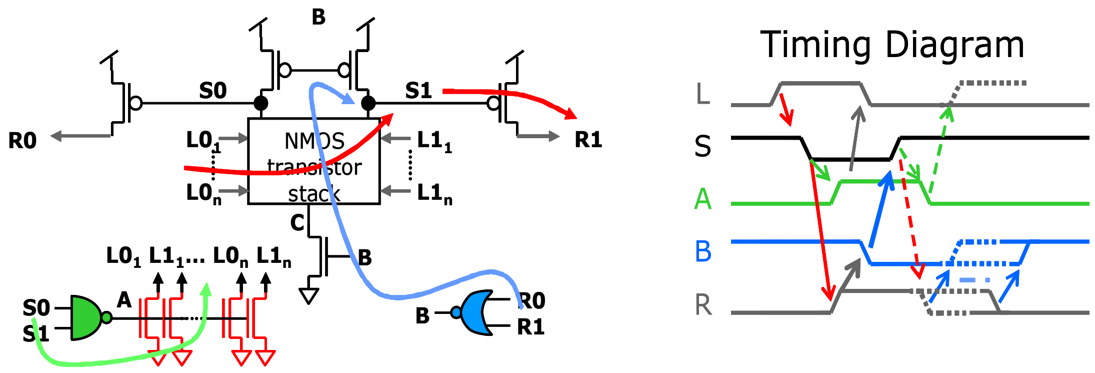


Figure 2-1: General structure of a static single track full buffer and its timing diagram

When there is no token on the right channel (R) (R is low meaning the channel is empty), the right environment enables the domino logic to process a new token. When a new token arrives at the left channel (L goes high) it is processed by lowering the state signal "S", which creates an output token on the right channel (R goes high) and asserts "A" removing the token from the left channel via reset NMOS transistors. The presence of the output token on the right channel restores the state signal and deactivates the NMOS transistor at the bottom of the N-stack thus disabling the stage from firing while the output channel is busy. The NAND gate is commonly called the state completion detector (SCD) and the NOR gate is called the right completion detector (RCD). A more detailed schematic of a SSTFB buffer is shown in Figure 2-2.

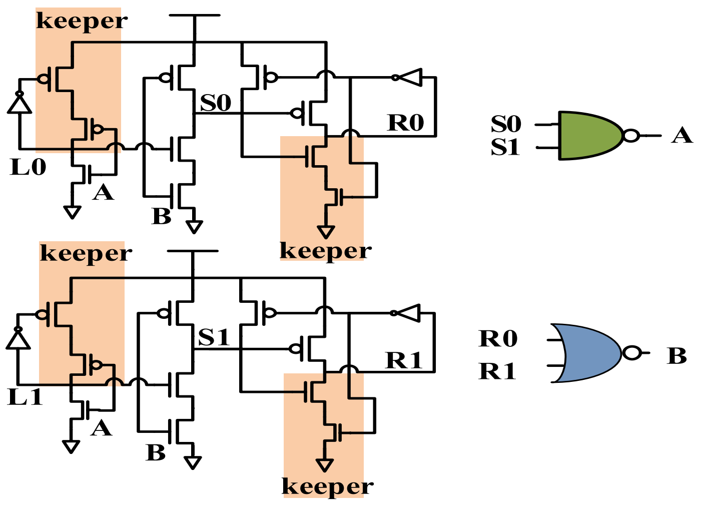


Figure 2-2: Static STFB dual-rail buffer

Notice that after the sender drives the line high, the receiver is responsible for actively keeping the line high (via the input keepers) until it wants to drive it low. Similarly, after the receiver drives the line low, the sender is responsible for actively keeping the line low until it wants to drive it high (via the output keepers). The line is always statically driven and no fight with staticizers exists. This hand-off technique enables the hold circuitry to be sized to a suitable strength creating a tradeoff between performance/power/area and robustness to noise. The inverters in the hold circuitry can be also be skewed such that they turn on early creating an overlap between the driving and hold logic. This overlap avoids the channel wire being in a tri-state condition thus making the circuit family more robust to noise. The overlap also helps ensure that the channel wires are always driven close to the power supplies further increasing noise margins.

The local cycle time of the static STFB template is 6 transitions with a forward latency of 2 transitions. It is called a full-buffer because each buffer stage can hold one token. The template is very flexible and can be expanded to implement different functionalities by enabling multiple 1ofN input channels, arbitrary NMOS pull down logic and multiple 1ofN output channels [5][6][7].

2.3. Pre-Charged Half Buffers

Pre-Charged Half Buffer (PCHB) is a QDI template developed at Caltech [9][10]. A single PCHB stage uses dual rail domino logic pre-charged by a single asynchronous control signal 'en' instead of two signals 'en' and 'pc' as in [10][11]. The domino block produces a dual rail output channel R and a control signal V which is used to detect its validity. Figure 2-3 shows a basic block diagram of PCHB. In general, a pipeline stage can contain multiple logic blocks to support multiple inputs or multiple outputs. Each logic block detects the validity of its outputs and produces a signal V. The control block detects the validity of the inputs and combines it with the V signals from all the logic blocks to generate the acknowledgement Le. The acknowledgement signals from the next stages are combined using a tree of c-elements and then combined with Le out of the controller to generate en for the logic block.

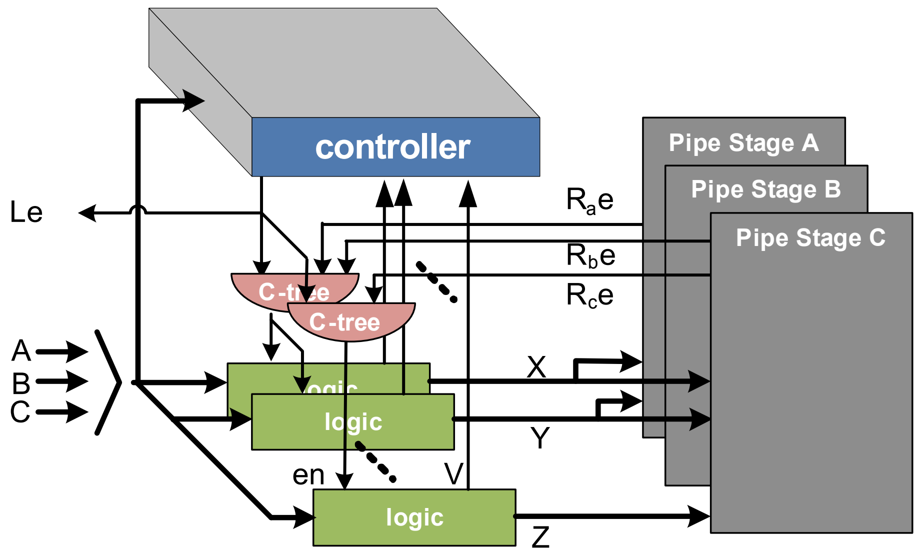


Figure 2-3: Block diagram for the PCHB template

The circuit relies on a relative timing constraint (shown by the thick green lines in Figure 2-4) for correct operation. In words, the input to the domino block should go low (neutral) before a rising transition on the control signal 'en' occurs. If this relative timing constraint is violated, the domino block may evaluate with old data creating an extra token on the output channel. Note that there are similar relative timing constraints associated with all the rails. These timing assumptions are a relaxed interpretation of the underlying isochronic fork assumption at the data inputs of this template and were verified to be sufficient for correctness by Ken Stevens using his tool RT-Analyze [15]. This template also has several timing loops (dotted lines) spanning a single to multiple stages which together dictate throughput.

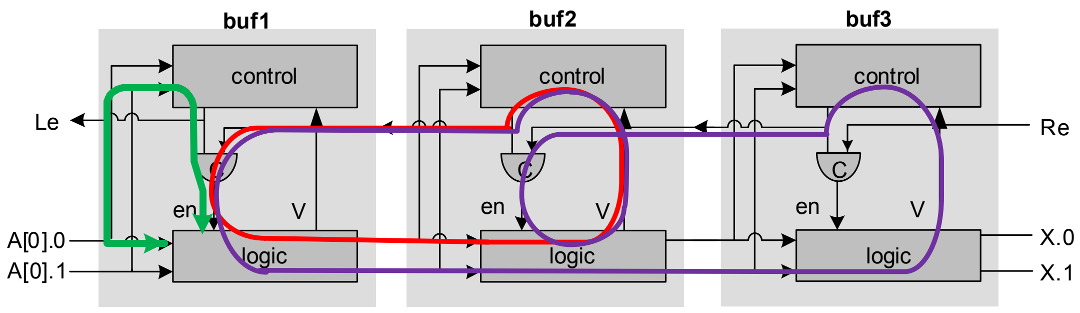


Figure 2-4: Three stage PCHB pipeline highlighting the relative timing constraint and the timing loops

2.4. Multi-Level Domino

MLD also uses dual rail domino to implement the logic blocks and a separate control block like in PCHB. A pipeline stage can contain *n* levels of domino blocks, *n-1* levels of which are pre-charged and evaluated by a single control signal ‘en’ while the n^{th} level of domino is pre-charged and enabled by different control signals ‘pc’ and ‘eval’ respectively as shown in Figure 2-5. The important feature of this template is that the last level of domino per pipeline stage does not pre-charge until the next pipeline stage has consumed the token. This makes the template fast as the first *n-1* domino blocks can start evaluating early with new data without having to wait for the next stage to consume the previous data. The last level of domino per stage is a special logic cell that generates a valid signal and is referred to as a v_logic cell. The pre-charge validity detector (PVD) as the name suggests is pre-charged low by the controller and only detects the validity of the stage by combining the valid signals from all the v_logic domino cells in that stage. The v_logic cells and the PVD are pre-charged simultaneously. It is important to note here that the neutrality of the data rails is not checked explicitly. This results in a relative timing constraint on the precharge delay and minimum pulse width constraints on ‘en’ and ‘pc’. In particular, the data inputs to the domino logic block should go neutral before the ‘en’ to the domino block goes high. This is to prevent the domino block from re-evaluating with old data. The minimum pulse width constraints are that the pre-charge signals ‘en’ and ‘pc’ should be low for a sufficiently long amount of time so as to pre-charge the domino blocks completely.

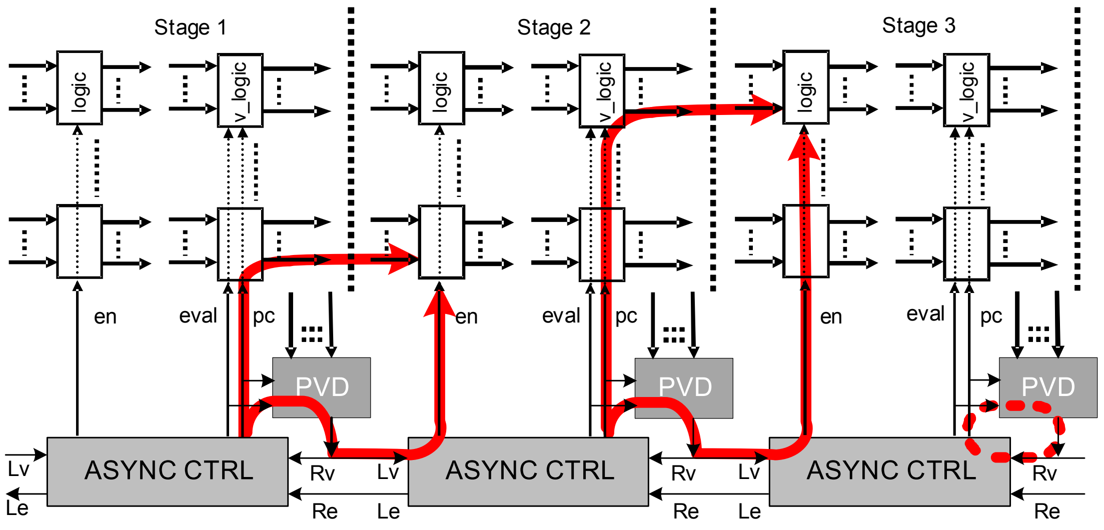


Figure 2-5: Block diagram of a multi-level domino pipeline highlighting the relative timing constraint (thick red lines) and the minimum pulse width constraints (dotted red lines).

2.5. Hierarchical design style

On the other end of design styles is the hierarchical design style which is much more flexible in terms of the circuit structure. Standard cells are put together in a hierarchical manner to create meaningful circuits that match a given specification. Silistix UK Ltd. is a startup which provides solutions for self-timed on-chip interconnect. They use a standard cell library with some added special cells such as C-elements and mutual exclusion elements necessary in asynchronous design [2]. They design delay insensitive circuits which are premised on the fact that they will function correctly regardless of the delays on the gates and the wires. An example circuit is the well-known sequencer also known as the Q-element [11] that implements enclosed handshaking on channel B and is shown in Figure 2-1.

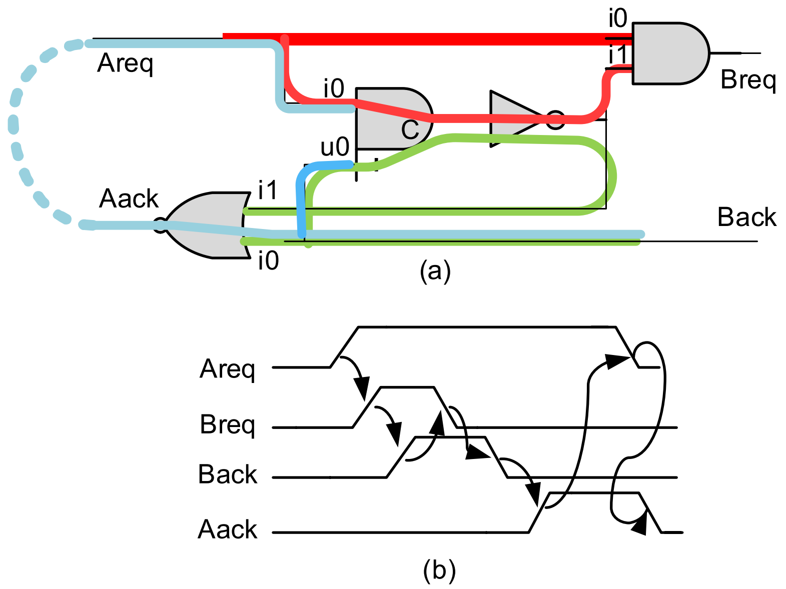


Figure 2-6: S-element with (a) RT constraints and (b) timing diagram

The S-element has two internal timing constraints (highlighted in red and green) and one external timing constraint (blue).

- Internal constraint 1 (red): This constraint forks from input Areq of the S-element. Input i0 of the AND gate should fall before input i1 of the AND gate rises. If this constraint is not satisfied, the AND gate can turn on creating a false request on channel B.

- Internal constraint 2 (green): This constraint forks from input Back of the S-element. Input i0 of the NOR gate should rise before input i1 of the NOR gate falls. If this constraint is not satisfied, the NOR gate can evaluate to a logic ‘1’ acknowledging falsely the completion of the handshake on channel B.

- External constraint (blue): This constraint forks from input Back of the S-element. Input u0 of the asymmetric C-element should fall before input i0 of the C-element rises in order to return the S-element back to its reset state. If this constraint is not satisfied, then a new request on channel A can be lost. The dashed arrow indicates that the constraint traverses paths in the environment of the S-element.

Chapter 3

3. Library Characterization for SSTFB

The industry standard format for representing delay and power information of a library is the Liberty format. It supports several delay models of which we chose the non-linear delay as it provides a reasonable tradeoff between accuracy and complexity. This delay model uses lookup tables indexed by input slew and/or load capacitance. There are four main steps involved in delay and power characterization. The first is defining timing arcs for the cell being characterized. The second is creating a suitable simulation environment for spice measurements. The third step is measuring power consumption and the last step is measuring pin capacitances. In addition, we would like to automate this flow as much as possible. Unlike synchronous standard cells, for which commercial library characterization tools are available, the effort has to be implemented from scratch and semi-automated. The following three subsections address these challenges and how they were solved in more detail. In addition, it is necessary to measure the correct supply currents to accurately characterize internal power.

3.1. Defining Timing Arcs

A set of timing arcs need to be identified that capture the behavior of SSTFB. The causality between the timing arcs is formalized in a marked graph. As an example, Figure 3-1 illustrates the marked graph model of a buffer cell surrounded by a bit generator and bucket in which the specific data rails have been abstracted. Our future plans include constraining the short circuit between senders and receivers. This requires detailed knowledge of when the driver circuits are active and motivated us to explicitly model the internal signals S and A as extra bi-directional pins. Notice that the dashed edges represent the behavior of the environment whereas the solid edges refer to the behavior of the cell. The '+' symbol is used to indicate a rising transition and data going valid, '-' for a falling transition and data going neutral, '0Z' for a low to tri-state transition and '1Z' for a high to tri-state transition.

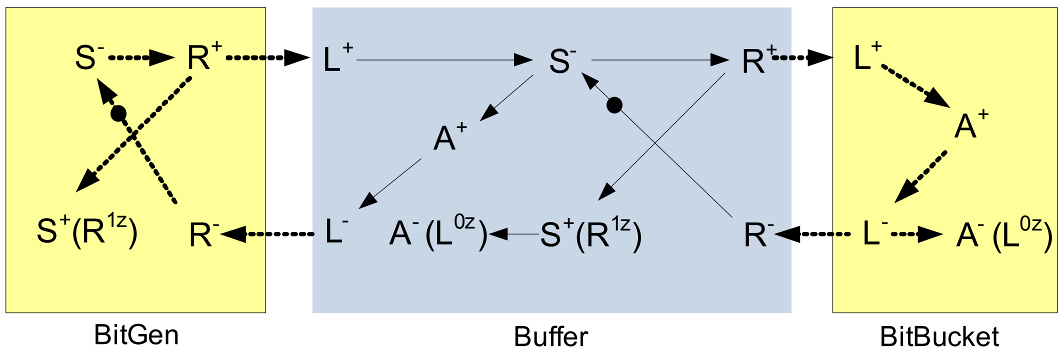


Figure 3-1: Abstract marked graph of a SSTFB buffer surrounded by a bit generator and generator

3.2. Creating a Simulation Environment

3.2.1. Input waveforms

Traditionally, the input driver uses a ramped linear waveform which is not desirable as it can by itself contribute to 5-10% delay error. Commercial library characterization tools use one of two approaches to generate real-world input waveforms: the pre-driver method or a pre-driver generated ‘real’ non-linear waveform. A buffer is often recommended for use as the pre-driver cell as shown in Figure 3-2(a). For asynchronous circuits, special cells are needed generate and consume input and output tokens to/from the circuit under test (CUT). Commonly, these cells which form the environment around the CUT are called bitgen and bucket respectively, shown in Figure 3-2(b). The bitgen implicitly takes care of the input waveform generation. The input slew is controlled by an adjustable capacitor C_S and the output load is controlled by the capacitor C_L.

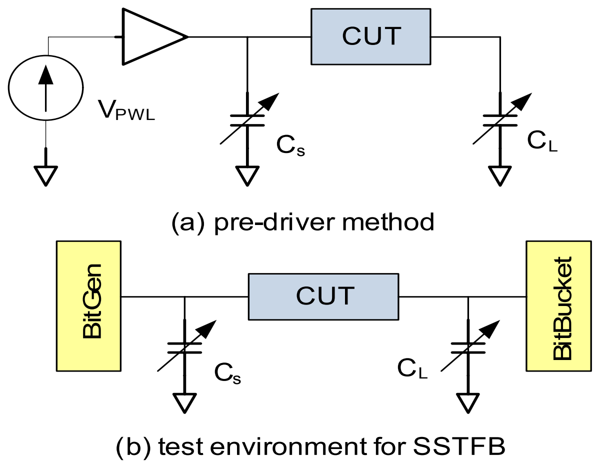


Figure 3-2: Test setup for (a) synchronous circuits (b) asynchronous circuits

3.2.2. Input Slew and Load Capacitances

The selection of input slew and load capacitance indices along with creating real-world input waveforms directly impacts the accuracy of the characterization. Delay behaves non-linearly and non-monotonically with input slew. The design usage space should be bounded by carefully selecting the minimum and maximum input slew and load capacitance values to minimize delay calculation error due to interpolation and extrapolation. The output load model can be simplified by assuming a lumped capacitance. The output load index must be based on the cell drive strength. The tables should have enough points for both input slew and output load index selections so as to cover non-linear or non-monotonic regions.

In the proposed flow, the minimum load capacitance was zero and the maximum was calculated such that the cell operated within pre-determined voltage swing levels. The input slew values were computed for each cell in the library based on the selected load capacitance values. The load capacitance on internal pins is fixed. Consequently timing arcs from input pins to the S and A pins need only be modeled as a 1D table (1x6) based on the input slew. However, arcs from the state pins S to the output pins R are modeled as a 2D table (6x6) based on both slew on S and output load.

3.3. Measuring power

The main challenges for power characterization are partitioning the currents drawn through the supply amongst timing arcs for the dynamic component, modeling short circuit current and the effects of crosstalk. The Liberty format measures internal energy per timing arc which includes short-circuit power. Power analysis tools convert this internal energy to internal power by dividing by the system cycle time. They also add short-circuit energy and switching energy, the latter calculated as the energy required for switching the total net capacitance on the nets. The dynamic internal energy component of energy for an arc can be calculated using the following equation:

$$E_{arc} = \frac{((I_{vdd/gnd} - I_{leakge}) * Vdd) * T}{N} \qquad (1)$$

where, $I_{vdd/gnd}$ is the average current measured through specific voltage sources associated with the timing arc, $I_{leakage}$ is the current measured when the circuit is idle, Vdd is the supply voltage, T is total simulation trace time and N is the number of tokens processed in time T. We added 0V voltage sources to Vdd segments of the extracted placed-and-routed netlist to measure the currents responsible for charging internal cell nodes. We added 0V voltage sources to segments of Gnd to measure the short-circuit current associated with charging output nodes (e.g., the R0/R1 nets). As an example, consider the SCD in a buffer. To measure the internal energy associated with the timing arc $S0^- \rightarrow A^+$, we insert a 0V source as shown in Figure 3-3.

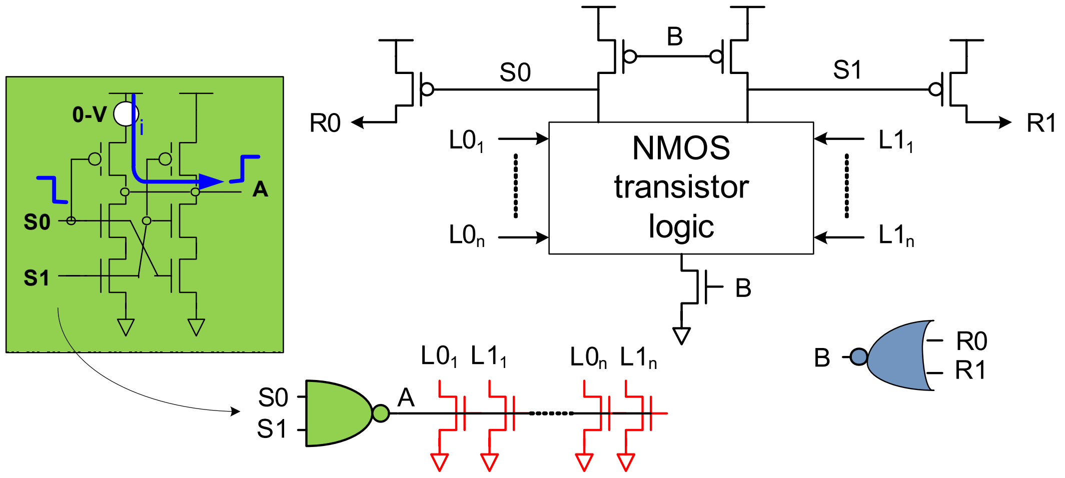


Figure 3-3: Addition of a 0V source to measure the internal energy associated with the timing arc $S0^- \rightarrow A^+$

In general, the measured currents associated with each token value can be partitioned among the associated timing arcs that must occur for each such token processed. For cells with a single-input channel, however, we partitioned currents into one power arc for each output accessed by an arbitrarily-chosen single related pin. For cells with multiple input channels in which multiple power arcs existed for a given output, we accounted for the power of all arcs in each arc. In this case, the power analysis tool chooses one such power arc depending on the timing of the related pins. This leads to a small amount of error because we are essentially assuming the slew on all input channels is identical.

3.4. Measuring Pin Capacitances

The liberty format requires pin capacitances for all input/output pins. A standard delay matching technique was used to measure the pin capacitances as shown in

Figure 3-4 and Figure 3-5. To measure the pin capacitance of pin A0 of the buffer, the delay d_1 from $S0^-$ of the bitgen to $A0^+$ of the buffer was measured. The buffer was then replaced by a variable capacitor and its value swept until delay d_2 from $S0^-$ of bitgen to $R0^+$ of bitgen matched delay d_1. The capacitance at which the delays match gives the capacitance of pin A0.

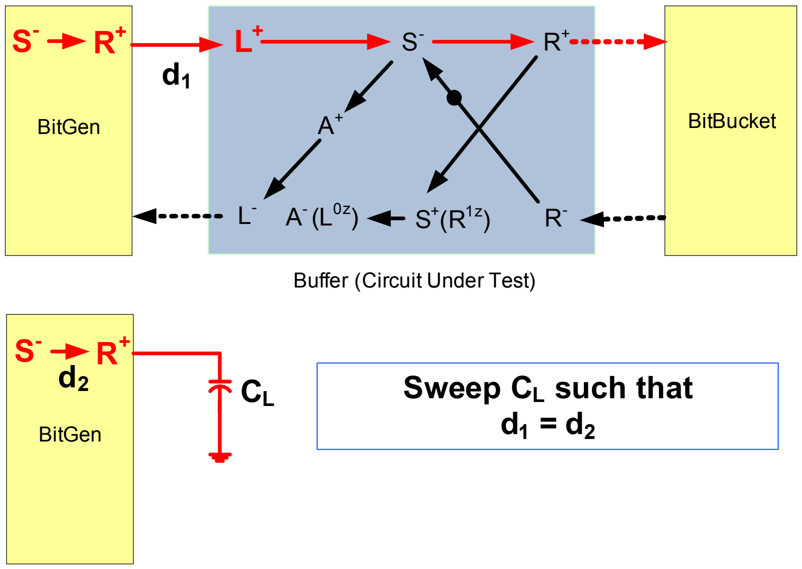


Figure 3-4: Measuring pin capacitance of pin A0 of a buffer

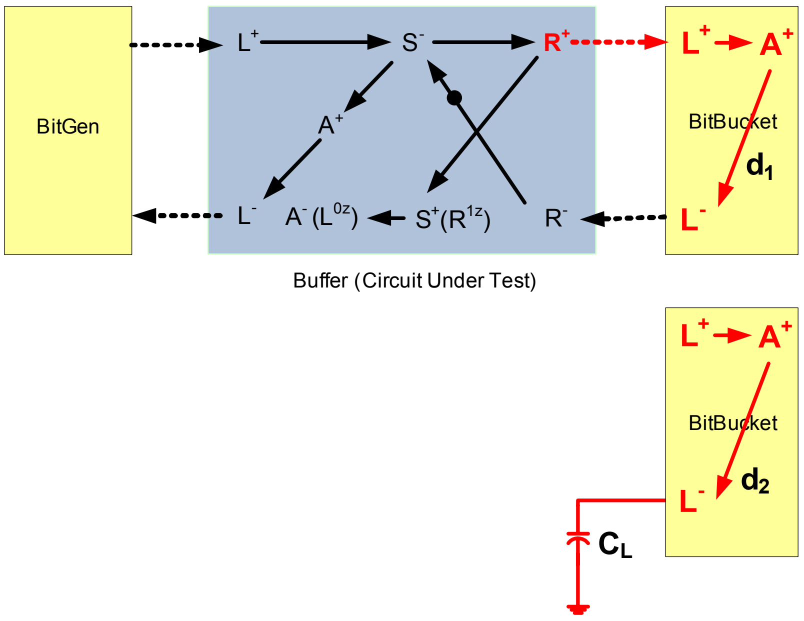


Figure 3-5: Measuring pin capacitance of pin R0 of a buffer

To measure the pin capacitance of pin R0 of the buffer, the delay d_1 from A^+ of the bucket to $R0^-$ of the buffer was measured. The buffer was then replaced by a variable capacitor and its value swept until delay d_2 from A^+ of the bucket to $A0^-$ of the bucket matched delay d_1. The capacitance at which the delays match gives the capacitance of pin R0. For proof of concept, the delays were matched only at 50% of supply voltage but ideally the delay should be matched at several points for a more accurate capacitance value.

There are some subtleties related to HSpice that should be taken care of while measuring the pin capacitances on channel R. In the second set-up where a load capacitor C_L is attached to the left channel of the bit bucket, one of the rails of the left channel should be initialized to the logic value ‘1’ using the .IC card in HSpice. Also, the internal pin A should be initialized to the logic value ‘0’. At time 0, the NOR gate shown in Figure 3-6 (a) evaluates to a ‘0’ causing the NANd gate to evaluate to a ‘1’. This causes a fight on the internal pin A creating a fake rising slew on it. Also, the state of NReset does not impact the value of pin A. If fake input channel rails L0’ and L1’ are created and tied low as in Figure 3-6 (b), the output of the NOR gate doesn’t control the NAND gate anymore. Initially, NReset is held high so as to prevent the bucket from consuming the token on the false rail L0 as A is 0. Once NReset is pulled low, the NAND gate turns on causing A to rise and pull the false rail L0 low. Pin A now has a real rising slew on it.

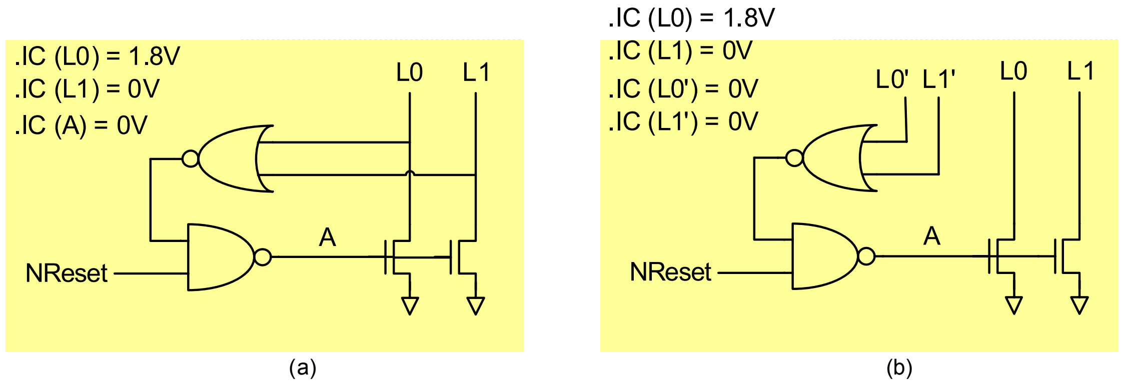


Figure 3-6: SSTFB BitBucket schematic (a) incorrect HSpice set-up (b) correct HSpice set-up

3.5. Library Generation

Using the above concepts, the flow illustrated in Figure 3-7 is used for complete timing and power characterization. Spice netlists of the cells were fed to Hspice along with perl-generated stimulus files. These stimulus files contain measure statements for delay, slew, and energy. Data is extracted from the output of Hspice which is in the .mt# file format and automatically converted to the liberty format. For ease of characterization, we sometimes assumed symmetry to estimate delays and slews of one data rail using measured data from the other rail, introducing a small amount of error due to small differences in the layout between rails.

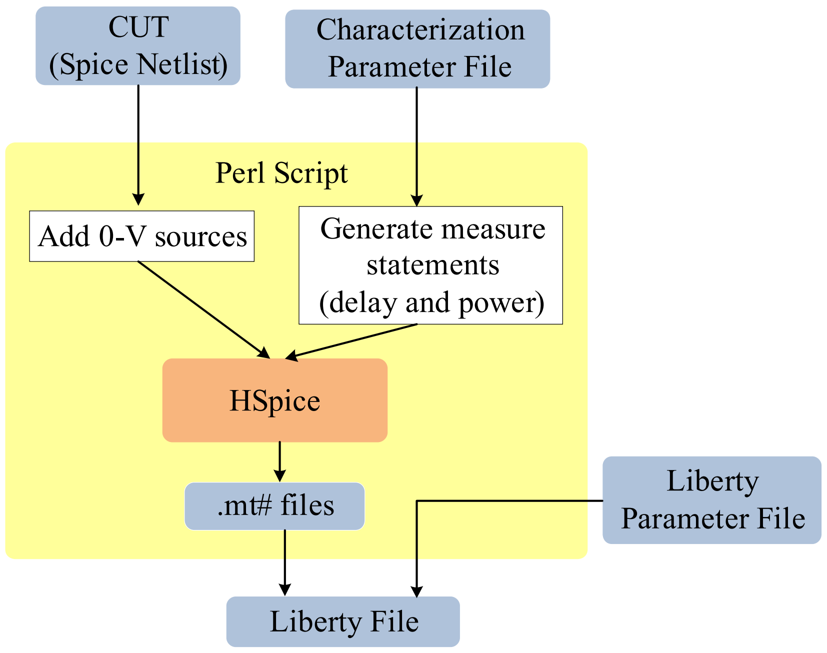


Figure 3-7: Characterization flow

3.6. Validation Results

To validate the quality of our characterization, we used several representative unbalanced non-linear pipelines commonly seen in asynchronous design (Figure 3-8). The validation flow is shown in Figure 3-9. Cadence's SoC Encounter is used to perform delay calculation for each timing arc instance in the routed netlist using the liberty description, recording the resulting arc delays in a standard delay format (.sdf) file. The .sdf file along with the Verilog netlist is simulated in Cadence's NC-Verilog simulator. This generates a timing-accurate value change dump (.vcd) file that records the signal activity. The .vcd file is then fed back into SoC Encounter to compute simulation-based power analysis. The results of the simulation-based timing and power analysis are compared to golden Hspice simulations.

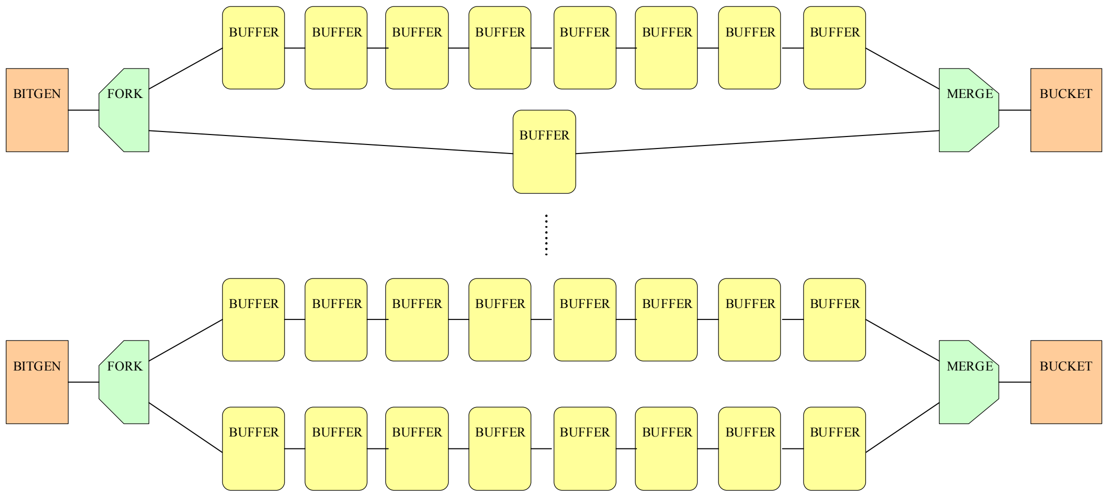


Figure 3-8: Unbalanced non-linear pipelines used for validating the characterization flow.

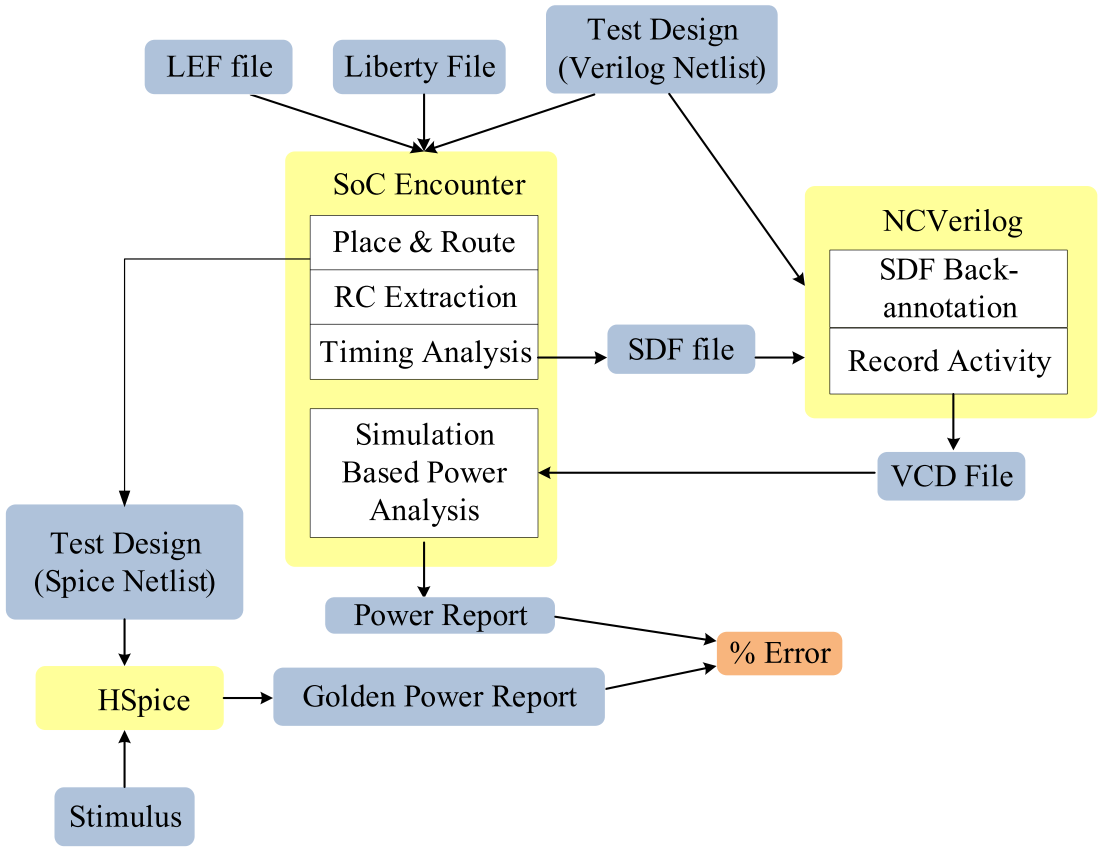


Figure 3-9: Validation flow

Figure 3-10 and Figure 3-11 show the performance and power dissipation as measured by Hspice and the Encounter-based flow using our prototype SSTFB library. The performance plot shows an interesting counter-intuitive result that maximum throughput is achieved when the short and long paths of the fork-join structure are somewhat unbalanced, i.e., the short path has 4 buffers while the long path has 8 buffers. This fact can be attributed to the “free-slack” associated with the buffers that are faster than the FORK and JOIN cells. More balanced fork-join pipelines are slightly slower due to increased wire delay and consume more energy.

The experimental results show a maximum error between the encounter estimated and Hspice golden numbers of 7.1%. We believe much of this error can be attributed to the limited slew propagation during SDF generation due to the loops and bi-directional pins in the SSTFB .lib model.

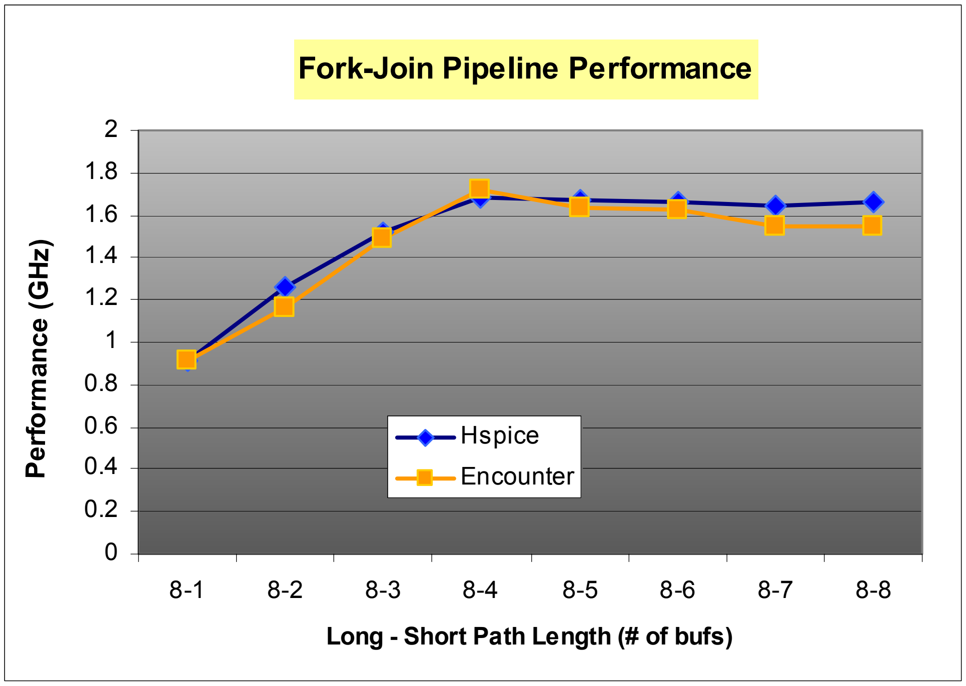


Figure 3-10: Fork-join pipeline experiment – performance

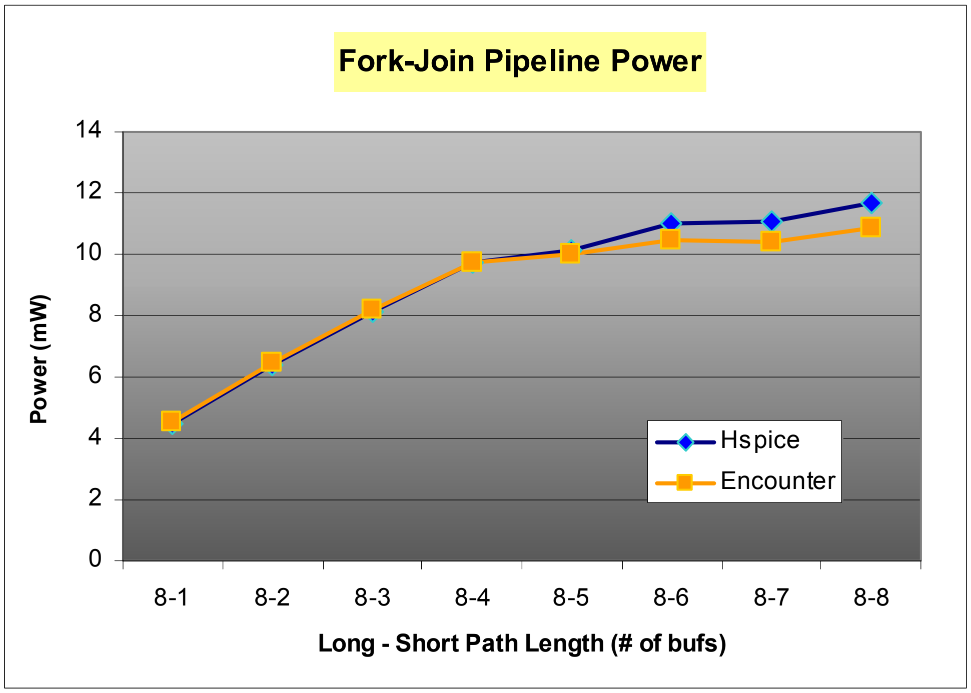


Figure 3-11: Fork-join pipeline experiment - power

Chapter 4

4. Static Timing Analysis for Asynchronous Circuits

PCHB and MLD templates are representative asynchronous circuit templates with timing loops and relative timing constraints but without the more complex features like bi-directional pins present in SSTFB. Due to this reason we chose to demonstrate the STA flow on PCHB and MLD templates and leave the extension to SSTFB as future work. Three steps constitute the static timing analysis flow for any template based design. The first step is to capture the timing constraints (relative and minimum pulse width) within PrimeTime. The second step is to break combinational loops which should be done in a manner that doesn't break paths of interest. The last step is to do performance verification. All these steps are described in detail in the following sections. For simplification, the various steps of the flow are explained using three-stage PCHB and MLD linear pipelines. For the DI circuits at Silistix, only constraint verification was carried out.

4.1. Modeling relative timing constraints

4.1.1. Non-sequential data checks

A hold or a setup check between two data signals is called a non-sequential constraint [14]. Synopsys PrimeTime allows the designer to check such a constraint using a command called *set_data_check*. In the synchronous world, one of these data signals D1 might be used to capture data on the other data signal D2 as shown in Figure 4-1. The command shown will check if the rising transition on D1 occurs a setup time 1.5ns before the rising edge on D2. The command constrains pin D1 relative to pin D2 and thus the names constrained pin and related pin. This command compares the slowest path to the constrained pin to the fastest path to the related pin to verify that the specified setup time was met. This is synonymous to checking if the latest arriving data signal sets up a setup time before the earliest edge of the clock. For a more accurate post-layout analysis, PrimeTime's on-chip variation (OCV) mode can be used to account for PVT variations between the two paths of interest. .

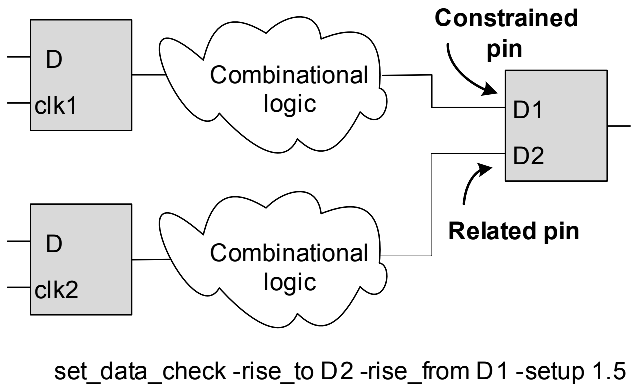


Figure 4-1: Non-sequential data to data check

The relative timing constraints stemming from a fork can be easily modeled using the set_data_check command and modeling the ends of the fork as constrained and related pins. The start of the fork is commonly referred to as the point of divergence (POD). In the case of PCHB and MLD, the data rails are the constrained pins and the enable pins 'en' are the related pins. The path from the POD to the constrained pin is referred to as the short path and the path from the POD to the related pin is referred to as the long path. The commands for a three stage PCHB pipeline, a three stage MLD pipeline and the S-element are listed in Figure 4-2. There are six RT constraints for the three-stage PCHB pipeline and four RT constraints for the three-stage MLD pipeline. For the purpose of illustration, an arbitrary setup margin of 0.5ns was chosen for the data checks.

```
set_data_check -rise_from buf1/en -fall_to buf1/A[0].0 -setup 0.5
set_data_check -rise_from buf1/en -fall_to buf1/A[0].1 -setup 0.5
set_data_check -rise_from buf2/en -fall_to buf2/A[0].0 -setup 0.5
set_data_check -rise_from buf2/en -fall_to buf2/A[0].1 -setup 0.5
set_data_check -rise_from buf3/en -fall_to buf3/A[0].0 -setup 0.5
set_data_check -rise_from buf3/en -fall_to buf3/A[0].1 -setup 0.5
```

(a)

```
set_data_check -rise_from buf2/eval -fall_to buf2/A[0].0 -setup 0.5
set_data_check -rise_from buf2/eval -fall_to buf2/A[0].1 -setup 0.5
```

(b)

```
set_data_check -rise_from i3/i1 -fall_to i3/i0 -setup 0.5
set_data_check -fall_from i5/i1 -rise_to i5/i0 -setup 0.5
```

(c)

Figure 4-2: Modeling the relative timing constraints using set_data_check for (a) PCHB template (b) MLD template (c) S-element

4.1.2. Associated challenges

There are several challenges involved in modeling RT constraints in PrimeTime. Typical STA tools that are built with synchronous circuits in mind expect flip flops enabled by a global clock signal at the launch and capture points of data. The timing constraint is that the data reaches its capture point from its launch point in less than a clock cycle. Without a global clock signal and flip flops to create data launch and capture points, it is impossible to model the constraints described in Chapter 2 in PrimeTime.

One approach to this is to model the POD and the POC as the launch and capture points respectively by creating virtual clocks on the PODs. Creating multiple clocks in the design poses other challenges. First, the related or the constrained pins can now originate from different clock domains in which case PrimeTime will check the paths separately and report all the combinations. In the case of the forks, both the short and the long paths start at the same point: the POD. Second, creating virtual clocks on pins in the design creates cut points in the corresponding timing graph. Thus any virtual clock that intersects a constraint causes a timing path in that constraint to be broken making it impossible to verify the two constraints in the same PrimeTime run. This problem is illustrated in Figure 4-3 where the two forks yield overlapping constraints. Specifically, the POD of the blue constraint associated with the fork in S-element 2 (POD2) intersects the long path of the red constraint associated with the fork in S-element 1. This problem is revisited in Section 4.3.

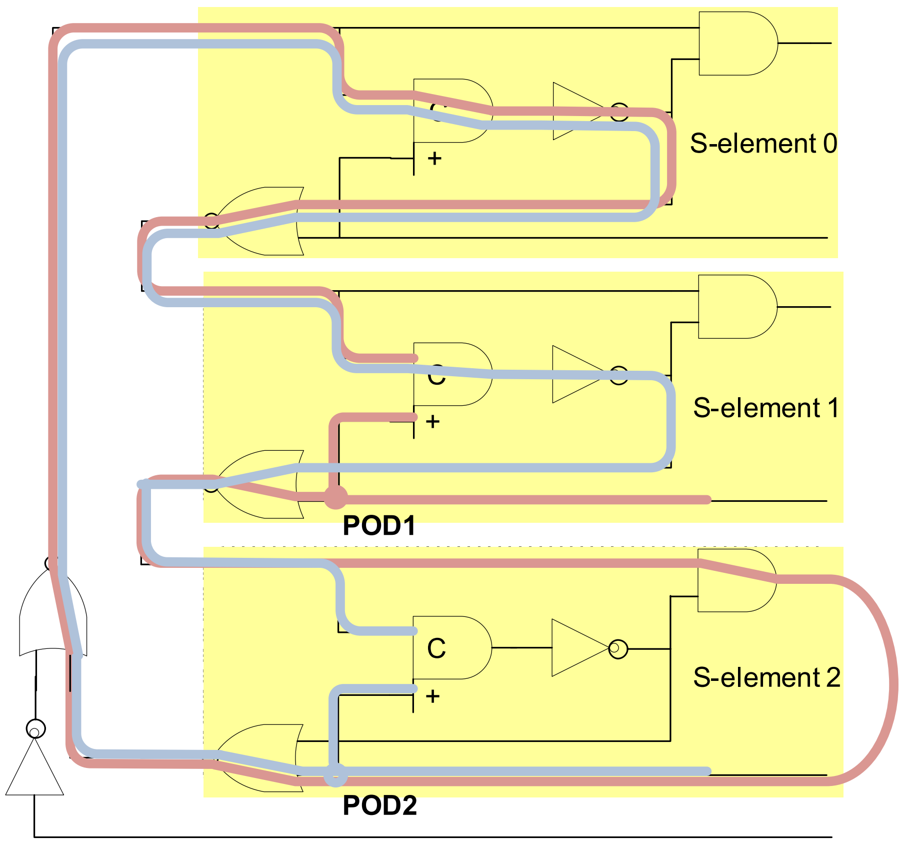


Figure 4-3: Configuration of three S-elements creating overlapping constraints

The solution to the first challenge is to specify the launching clock explicitly using the –clock option in the data check command. This has the added benefit of reducing the number of paths PrimeTime needs to analyze potentially reducing the run time. The downside is that the slew information on the POD is lost as the tool creates an ideal clock which can lead to over optimistic results. This effect can be mitigated by moving the POD back one level for every stage except the first. Figure 4-4 shows the modified commands that specify the clock domains and reflect the new PODs in the case of PCHB. An important thing to note here is that now there are two paths to the related pin: one through the true rail and one through the false rail. While checking for the constraint on the false rail, the path from the POD through the true rail should be disabled and vice versa to generate the correct timing reports.

```
set_data_check -clock buf1/A[0].0 -rise_from buf1/en -fall_to buf1/A[0].0 -setup 0.5
set_data_check -clock buf1/en  -rise_from buf2/en -fall_to buf2/A[0].0 -setup 0.5
set_data_check -clock buf2/en  -rise_from buf3/en -fall_to buf3/A[0].1 -setup 0.5
```
(a)

```
set_data_check -clock buf1/pc  -rise_from buf2/eval -fall_to buf2/A[0].0 -setup 0.5
```
(b)

```
set_data_check -clock ireq -rise_from i3/i1 -fall_to i3/i0 -setup 0.5
```
(c)

Figure 4-4: Modified set_data_check commands for (a) PCHB template (b) MLD template and (c) S-element

The external constraint on the S-element described earlier in Section 2.5. requires special handling as the long path can traverse many gates through many levels of hierarchy before converging at the POC. The long path may also branch off several times causing PrimeTime to analyze all the possible paths between the POD and the POC in order to find the fastest slow path. This can result in an exponential increase in run time and render the approach useless. In most cases with careful placement and routing, a few gate delays on the slow path provide enough setup margins. Thus the related pin can be moved closer to the POD reducing the number of paths that need to be inspected. This new related pin will be referred to as the fake related pin throughout the rest of the document. Starting close to the POD and moving away from it, multiple fake related pins can be selected and set_data_checks issued for each so that a good range of gate delays is included to make sure that the constraint passes verification in a single run and false negatives avoided. If a fork is encountered, paths on all branches should be traversed. In the example shown in Figure 4-5, where the POD is pin in2 and the POC is gate G, some of the possible fake related pins are U3/i1, U3/i1, U4/i1 and U5/i0.

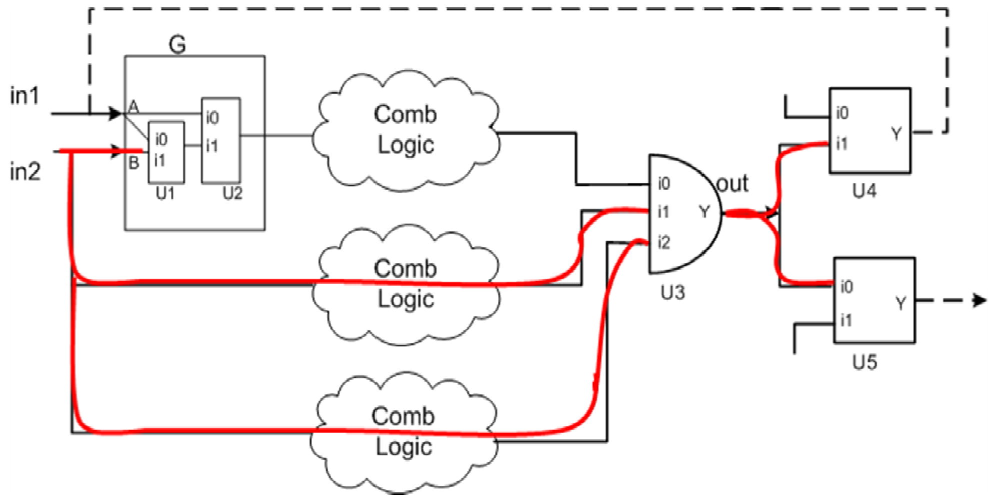
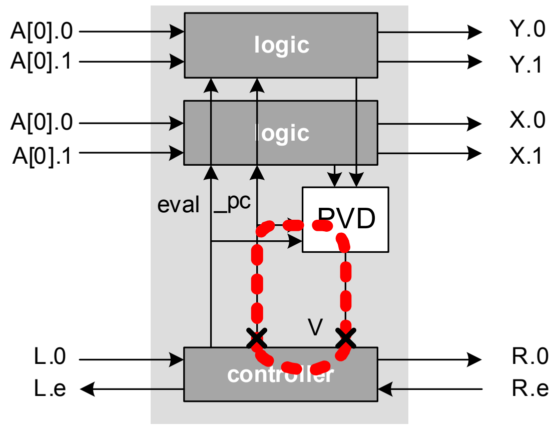


Figure 4-5: Multiple fake related pins per constraint

It is also important to note here that the –clock option does not allow us to specify the direction of the clock transition we are interested in. Thus it may find that the fastest slow path is from a falling edge of the clock when we are interested in the rising edge of the clock. This can easily be taken care of by disabling all the paths from the falling edge of the clock within PrimeTime.

4.2. Modeling the minimum pulse width constraints

The minimum pulse width constraint in MLD spans a timing loop as shown in Figure 2-5. This means that the constraint needs to be decomposed into at least two segments depending on how the loop is broken. The natural approach to modeling this in PrimeTime is by constraining it as a minimum delay constraint using the command set_min_delay. An example command set is shown in Figure 4-6 where the loop is broken at the boundaries of the control block yielding two minimum delay constraints.



```
set_min_delay 1 -fall_from buf1/buf_v_logic/_pc
                -fall_to buf1/buf_v_logic/V

set_min_delay 1 -fall_from buf1/buf_v_logic/V
                -rise_to buf1/buf_v_logic/_pc
```

Figure 4-6: Modeling minimum pulse width constraint

4.3. Breaking timing loops

PrimeTime has two loop breaking techniques: static loop breaking and dynamic loop breaking. With static loop breaking, the tool automatically breaks loops by disabling timing arcs during the initial timing analysis. These loop breaks persist throughout the primetime run until another run is initiated. Dynamic loop breaking on the other hand does not permanently disable any timing arcs and guarantees that all paths will be reported. Both these techniques don't work for asynchronous circuits due to several reasons.

Static loop breaking has the potential for disabling paths of interest resulting in incorrect timing reports. Consider the example in Figure 4-7 where there are two paths to the constrained pin, labeled sp1 and sp2. There is also a combinational loop that includes path sp2 which is the slowest short path. If PrimeTime disables a timing path through any of the inverters in path sp2, then path sp1 will become the new slowest short path resulting in an incorrect data check report. If there was only one short path sp2, then PrimeTime would not find any paths for the data check report.

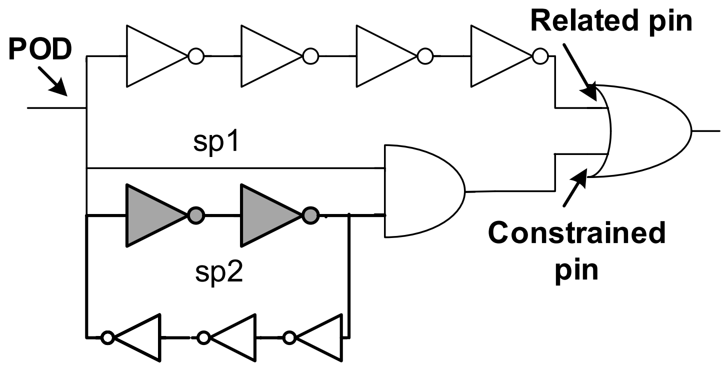


Figure 4-7: Static loop breaking

The problem with dynamic loop breaking is that the loop breaking flexibility is accompanied by heavy memory usage and large run-time making it impractical for even a medium sized design. Due to these reasons, we are forced to break loops explicitly by disabling timing paths carefully so as not to break any paths of interest. This is done with the command *set_disable_timing*.

4.3.1. Loop breaks for template-based designs

The regular structure of the circuits built using PCHB and MLD templates proves to be advantageous while breaking timing loops. By disabling certain timing arcs for some of the library cells, all timing loops can be broken without breaking any paths of interest. Figure 4-8 shows the loop breaks (dotted lines) for PCHB and MLD. The dots indicate the location of the clocks.

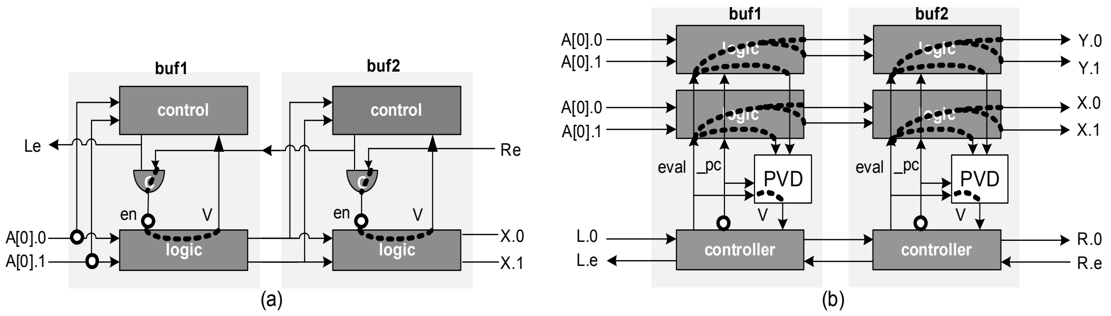


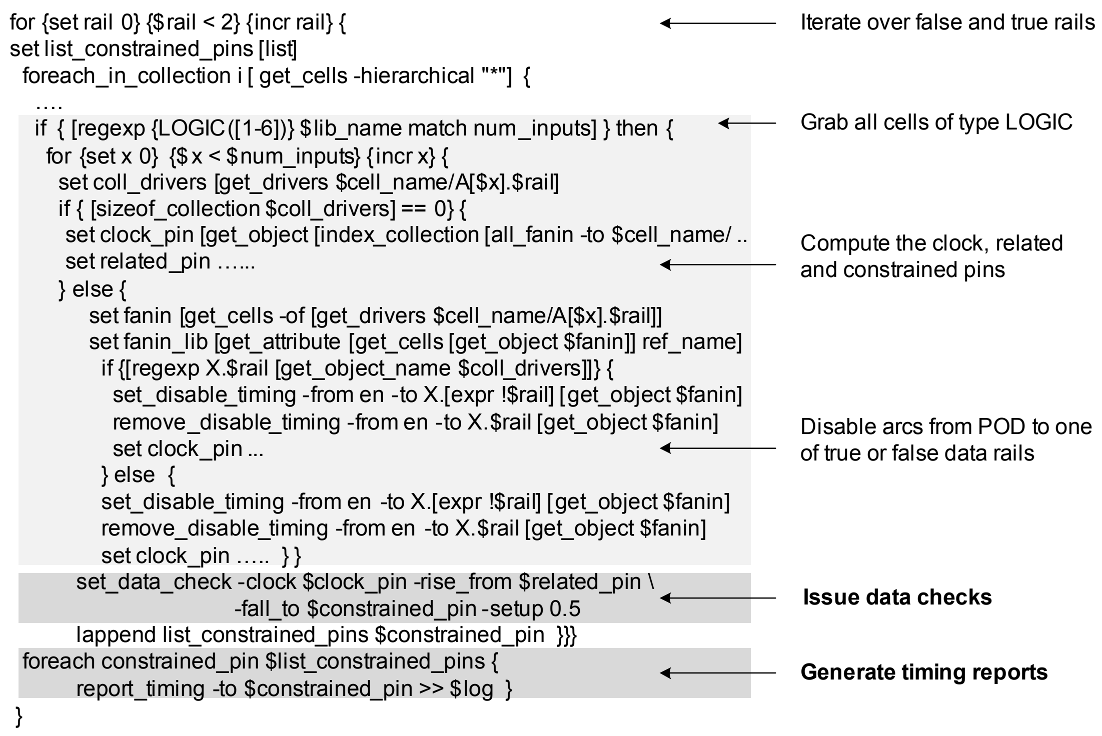
Figure 4-8: Clocks and loop breaks for (a) PCHB and (b) MLD templates

4.3.2. Loop breaks for hierarchical design style based circuits

The lack of any regular structure poses a bigger challenge in loop breaking. In a medium sized circuit, tens of thousands of feedback loops exist [9] making it very likely that there will be overlaps between loop breaks and timing constraints that need to be verified. These overlaps are in addition to the overlaps caused due to intersecting constraints discussed in section 4.1.2. A naïve solution to the overlapping constraint problem is to create and remove virtual clocks on the fly during a PrimeTime run. But this can be very costly in terms of run-time as every time a clock is created or removed, the timing graph internal to the tool changes evoking time consuming timing updates. Instead, a methodology is required which identifies a set of loop breaks such that a minimum number of constraints are disturbed and partitions the constraints into an optimal number of non-overlapping sets based on the loop breaks. A prototype flow was developed as part of the internship to demonstrate the feasibility of the proposed methodology.

4.3.3. Fully automated scripts

PrimeTime supports Tcl and many useful internal commands which can be used to gain information about the design being verified. Portions of the fully automated RT constraints verification scripts for PCHB is shown in Figure 4-9. Internal PrimeTime commands are used to get all cells of type LOGIC, get connectivity information to determine the POD, the related and the constrained pins to create the constraints and apply them automatically. Some cells such as token buffers need special handling and are not shown in the figure for simplicity. Note the first for loop that divides the verification of the constraints into two sets. One set for the false rails and the other for the true rails. This is because the long path traverses the data rails and the timing arcs from the POD to one of the rails is disabled However, in the case of MLD all the constraints can be verified in a single set as the long path does not go through the data rails. This also enables us to generate the timing reports associated with both the rails at once at the end of the script. RT verification and performance verification scripts for PCHB and MLD are written in a similar manner. In the case of Silistix' circuits, a script that verified the relative timing constraints associated with all the S-elements in the design was developed. The script partitions the constraints into non-overlapping sets by comparing the POD of a constraint to all the related pins associated with all the other constraints. A match means that the two constraints should be put in different sets. With this approach, a timing update per set of constraints is required compared to a timing update per constraint which leads to large run times. Please refer to Appendix A for all the scripts.



```
for {set rail 0} {$rail < 2} {incr rail} {
set list_constrained_pins [list]
  foreach_in_collection i [ get_cells -hierarchical "*"] {
    ….
    if { [regexp {LOGIC([1-6])} $lib_name match num_inputs] } then {
      for {set x 0} {$x < $num_inputs} {incr x} {
        set coll_drivers [get_drivers $cell_name/A[$x].$rail]
        if { [sizeof_collection $coll_drivers] == 0} {
         set clock_pin [get_object [index_collection [all_fanin -to $cell_name/ ..
         set related_pin …...
        } else {
             set fanin [get_cells -of [get_drivers $cell_name/A[$x].$rail]]
             set fanin_lib [get_attribute [get_cells [get_object $fanin]] ref_name]
               if {[regexp X.$rail [get_object_name $coll_drivers]]} {
                 set_disable_timing -from en -to X.[expr !$rail] [get_object $fanin]
                 remove_disable_timing -from en -to X.$rail [get_object $fanin]
                 set clock_pin ...
               } else  {
               set_disable_timing -from en  -to X.[expr !$rail] [get_object $fanin]
               remove_disable_timing -from en  -to X.$rail [get_object $fanin]
               set clock_pin …..  } }
         set_data_check -clock $clock_pin -rise_from $related_pin \
                             -fall_to $constrained_pin -setup 0.5
         lappend list_constrained_pins $constrained_pin  }}}
  foreach constrained_pin $list_constrained_pins {
         report_timing -to $constrained_pin >> $log  }
}
```

Iterate over false and true rails
Grab all cells of type LOGIC
Compute the clock, related and constrained pins
Disable arcs from POD to one of true or false data rails
Issue data checks
Generate timing reports

Figure 4-9: Relative timing constraints verification script for PCHB

4.4. Modeling performance

As mentioned earlier, the cycle time of these pipelines is reduced to verifying the timing of the loops. Because of the limitations of STA tools, these timing loops need to be decomposed into two or more segments by creating cut points. The delay of these segments should then be verified using maximum delay constraints with the command set_max_delay. The approach then is as follows:

- Identify a minimal set of cut-points that break all timing loops.
- Constrain all segments with a maximum delay constraint where the maximum delay value is the sum of the specified delays of all timing arcs in the segment
- If all segments meet timing, then all cycle time constraints are met.

Notice that the latency of a design is simultaneously verified as long as segments that begin at primary inputs and/or end at primary outputs are identified, constrained, and verified. We note that this decomposition does force some conservativeness as time borrowing across segments is disallowed. In the case of the PCHB pipeline, creating cut-points at all the enable pins of the logic blocks cuts all the timing loops.

4.5. Results

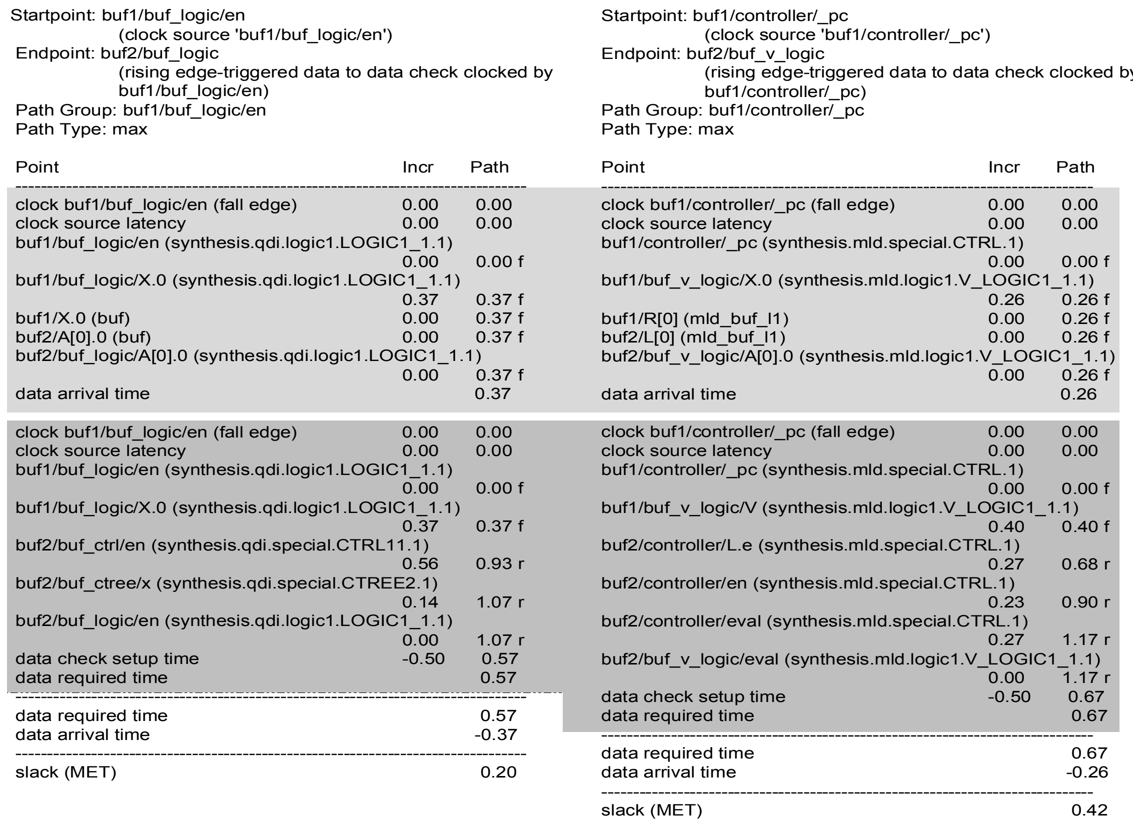
The proposed flow was fully automated using tcl and run on several ISCAS benchmarks that were synthesized to a synchronous image netlist and then automatically translated to PCHB and MLD pipelines. The run times for constraint verification and performance verification on a Sun Blade 1000 machine with 2GB RAM are shown in Table 1 and Table 2. Sample timing reports for relative timing constraints and performance verification for the simple linear pipeline test cases are shown in Figure 4-10 and Figure 4-11. For the purpose of illustration, an arbitrary setup margin of 0.5ns was used for constraint verification and an arbitrary maximum delay value of 2ns was used to constrain the segments for performance verification. In this example, the RT constrain is satisfied with a left-over margin of 0.2ns for PCHB and 0.42ns for MLD. The first half of the constraint verification report shows the short path and the second half of the report shows the long path for the data check. In the case of performance verification, the segment shown satisfies the maximum delay constraint with a left-over margin of 0.39ns for PCHB and 1.93ns for MLD.

Table 1: Run times for RT constraints verification

ISCAS benchmark	PCHB		MLD	
	No. of Constraints	Run Time (sec)	No. of Constraints	Run Time (sec)
s298	230	8	212	5
s400	374	14	296	8
s1423	1616	115	1456	63
c3540	6870	17 min	7560	11 min

Table 2: Run times for performance verification

ISCAS benchmark	PCHB		MLD	
	No. of Constraints	Run Time (sec)	No. of Constraints	Run Time (sec)
s298	963	6	4613	18
s400	1447	12	6268	18
s1423	4574	41	112727	5 min
c3540	23414	210	747426	1 hr 19 min



```
Startpoint: buf1/buf_logic/en
            (clock source 'buf1/buf_logic/en')
Endpoint: buf2/buf_logic
            (rising edge-triggered data to data check clocked by
            buf1/buf_logic/en)
Path Group: buf1/buf_logic/en
Path Type: max

Point                                                Incr     Path
--------------------------------------------------------------------------------
clock buf1/buf_logic/en (fall edge)                  0.00     0.00
clock source latency                                 0.00     0.00
buf1/buf_logic/en (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.00     0.00 f
buf1/buf_logic/X.0 (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.37     0.37 f
buf1/X.0 (buf)                                       0.00     0.37 f
buf2/A[0].0 (buf)                                    0.00     0.37 f
buf2/buf_logic/A[0].0 (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.00     0.37 f
data arrival time                                             0.37

clock buf1/buf_logic/en (fall edge)                  0.00     0.00
clock source latency                                 0.00     0.00
buf1/buf_logic/en (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.00     0.00 f
buf1/buf_logic/X.0 (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.37     0.37 f
buf2/buf_ctrl/en (synthesis.qdi.special.CTRL11.1)
                                                     0.56     0.93 r
buf2/buf_ctree/x (synthesis.qdi.special.CTREE2.1)
                                                     0.14     1.07 r
buf2/buf_logic/en (synthesis.qdi.logic1.LOGIC1_1.1)
                                                     0.00     1.07 r
data check setup time                               -0.50     0.57
data required time                                            0.57
--------------------------------------------------------------------------------
data required time                                            0.57
data arrival time                                            -0.37
--------------------------------------------------------------------------------
slack (MET)                                                   0.20
```

(a)

```
Startpoint: buf1/controller/_pc
            (clock source 'buf1/controller/_pc')
Endpoint: buf2/buf_v_logic
            (rising edge-triggered data to data check clocked by
            buf1/controller/_pc)
Path Group: buf1/controller/_pc
Path Type: max

Point                                                Incr     Path
--------------------------------------------------------------------------------
clock buf1/controller/_pc (fall edge)                0.00     0.00
clock source latency                                 0.00     0.00
buf1/controller/_pc (synthesis.mld.special.CTRL.1)
                                                     0.00     0.00 f
buf1/buf_v_logic/X.0 (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                     0.26     0.26 f
buf1/R[0] (mld_buf_l1)                               0.00     0.26 f
buf2/L[0] (mld_buf_l1)                               0.00     0.26 f
buf2/buf_v_logic/A[0].0 (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                     0.00     0.26 f
data arrival time                                             0.26

clock buf1/controller/_pc (fall edge)                0.00     0.00
clock source latency                                 0.00     0.00
buf1/controller/_pc (synthesis.mld.special.CTRL.1)
                                                     0.00     0.00 f
buf1/buf_v_logic/V (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                     0.40     0.40 f
buf2/controller/L.e (synthesis.mld.special.CTRL.1)
                                                     0.27     0.68 r
buf2/controller/en (synthesis.mld.special.CTRL.1)
                                                     0.23     0.90 r
buf2/controller/eval (synthesis.mld.special.CTRL.1)
                                                     0.27     1.17 r
buf2/buf_v_logic/eval (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                     0.00     1.17 r
data check setup time                               -0.50     0.67
data required time                                            0.67
--------------------------------------------------------------------------------
data required time                                            0.67
data arrival time                                            -0.26
--------------------------------------------------------------------------------
slack (MET)                                                   0.42
```

(b)

Figure 4-10: Relative timing constraints verification report for (a) PCHB and (b) MLD templates

```
Startpoint: buf1/buf1_logic/en
          (internal path startpoint)
Endpoint: buf3/buf1_logic/en
          (internal path endpoint)
Path Group: **default**
Path Type: max

Point                                                  Incr    Path
---------------------------------------------------------------------
input external delay                                   0.00    0.00 r
buf1/buf1_logic/en (synthesis.qdi.logic1.LOGIC1_1.1)
                                                       0.00    0.00 r
buf1/buf1_logic/X.0 (synthesis.qdi.logic1.LOGIC1_1.1)
                                                       0.39    0.39 r
buf1/X.0 (buf)                                         0.00    0.39 r
buf2/A[0].0 (buf)                                      0.00    0.39 r
buf2/buf1_logic/X.1 (synthesis.qdi.logic1.LOGIC1_1.1)
                                                       0.37    0.76 r
buf2/X.1 (buf)                                         0.00    0.76 r
buf3/A[0].1 (buf)                                      0.00    0.76 r
buf3/buf1_logic/V (synthesis.qdi.logic1.LOGIC1_1.1)
                                                       0.32    1.09 r
buf3/buf1_ctrl/en (synthesis.qdi.special.CTRL11.1)
                                                       0.31    1.40 f
buf3/buf1_ctree/x (synthesis.qdi.special.CTREE2.1)
                                                       0.22    1.61 f
buf3/buf1_logic/en (synthesis.qdi.logic1.LOGIC1_1.1)
                                                       0.00    1.61 f
data arrival time                                              1.61

max_delay                                              2.00    2.00
output external delay                                  0.00    2.00
data required time                                             2.00
---------------------------------------------------------------------
data required time                                             2.00
data arrival time                                             -1.61
---------------------------------------------------------------------
slack (MET)                                                    0.39
```

(a)

```
Startpoint: buf2/buf_v_logic/X.0
          (internal path startpoint)
Endpoint: buf3/buf_v_logic/X.1
          (internal path endpoint)
Path Group: **default**
Path Type: max

Point                                                  Incr    Path
---------------------------------------------------------------------
input external delay                                   0.00    0.00 r
buf2/buf_v_logic/X.0 (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                       0.00    0.00 r
buf2/R[0] (mld_buf_l1)                                 0.00    0.00 r
buf3/L[0] (mld_buf_l1)                                 0.00    0.00 r
buf3/buf_v_logic/X.1 (synthesis.mld.logic1.V_LOGIC1_1.1)
                                                       0.07    0.07 r
data arrival time                                              0.07

max_delay                                              2.00    2.00
output external delay                                  0.00    2.00
data required time                                             2.00
---------------------------------------------------------------------
data required time                                             2.00
data arrival time                                             -0.07
---------------------------------------------------------------------
slack (MET)                                                    1.93
```

(b)

Figure 4-11: Performance verification report for (a) PCHB and (b) MLD templates

Chapter 5

5. Conclusions and Future Work

A fully characterized asynchronous library not only supports back-annotated simulation-based power and timing analysis. It also enables timing driven place and route, performance and power driven synthesis and ECO flows. Moreover, characterized asynchronous libraries are a necessary pre-cursor to extending STA-based timing sign-off to these designs. The work presented in this thesis demonstrates the issues, feasibility, and potential accuracy associated with characterizing static STFB circuits. This is quite promising because the SSTFB circuits have among the most complex timing relationships of the many different proposed asynchronous design styles and have promising characteristics for application in low-power high-performance SoC interconnects.

This thesis also presents a fully-automated flow for pre and post layout static timing analysis for two template-based asynchronous circuits using a commercial STA tool. The proposed flow is not specific to a design and can be used to verify the timing of any design built using PCHB or MLD templates. A prototype constraint verification flow for hierarchical design style based asynchronous circuits is also presented. The performance verification flow verifies a fixed delay model that allows time borrowing across gates in a segment but not across segments.

Potential extensions to this work include the application of the timing constraints to performance-driven place-and-route as well as the application of both the characterization and the static timing analysis methodologies to other non-standard circuits. This includes extending our STA flow to single-track templates with bi-directional pins, such as SSTFB, and bundled-data designs, as well as synchronizers typically found in SoC systems. In addition, a tighter connection between the STA flow and automatic generation of relative-timing constraints to verify the completeness of the constraints tested, actively manage the run-times of the STA tool, and reduce the conservativeness of the constraints would also be valuable.

Bibliography

[1] N. Andrikos, L. Lavagno, D. Pandini, C. P. Sotiriou, "A Fully-Automated Desynchronization Flow for Synchronous Circuits", *Design Automation Conference,* , pp. 982-985, June 2007

[2] W.J.Bainbridge, A.Bardsley, R.W.McGuffin, "System-on-Chip Design using Self-timed Networks-on-Chip", Whitepaper, www.silistix.com.

[3] J. Cortadella, A. Kondratyev, L. Lavagno, C. P. Sotiriou, "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 10, pp. 1904-1921, October 2006

[4] M. Ferretti and P. A. Beerel, "High Performance Asynchronous Design Using Single-Track Full-Buffer Standard Cells", *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 6, pp. 1444-1454, June 2006.

[5] M. Ferretti and P. A. Beerel, "Single-Track Asynchronous Pipeline Templates using 1-of-N Encoding", *DATE'02*, pp. 1008-1015, March 2002.

[6] P. Golani, G. D. Dimou, M. Prakash, P. A. Beerel, "Design of a High-Speed Asynchronous Turbo Decoder", *ASYNC 2007*, pp. 49-59, March 2007.

[7] P. Golani and P. A. Beerel, "High-Performance Noise Robust Asynchronous Circuits", *Proc. of ISVLSI*, pp. 173-178, March 2006.

[8] P. Golani and P. A. Beerel, "Back-Annotation in High-Speed Asynchronous Design" *Journal of Low Power Electronics 2*, pp. 37–44, 2006.

[9] D. Jackson, J. Bainbridge, Silistix UK Ltd., Personal Communication.

[10] A. M. Lines, "Pipelined asynchronous circuits" *Master's thesis,* California Institute of Technology, 1996.

[11] Alain J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits", In C. A. R. Hoare, editor, Developments an Concurrency and Communzcataon, UT Year of Programming Series, pp. 1-64. Addison-Wesley, 1990.

[12] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, T. K. Lee, “The Design of an Asynchronous MIPS R3000 Microprocessor”, *Proc. 17th Conference on Advanced Research in VLSI*, pp. 164-181, 1997.

[13] M. A. Pena, J. Cortadella, A. Kondratyev, E. Pastor, “Formal Verification of Safety Properties in Timed Circuits”, *6th International Symposium on Asynchronous Circuits and Systems*, pp. 2–11, 2000.

[14] PrimeTime User Guide: Advanced timing analysis, Version Y-2006.06, Synopsys Inc., June 2006.

[15] K. Stevens, University of Utah, Personal Communication.

[16] K. Stevens, R. Ginosar, S. Rotem, “Relative Timing”, *ASYNC*, vol II, no 1, pp 129-140, April 1999.

[17] E. Sutherland, J. K. Lexau, “Designing Fast Asynchronous Circuits”, *7th International Symposium on Asynchronous Circuits and Systems*, pp 184–193, March 2001.

[18] H. Zheng, C. J. Myers, D. Walter, S. Little, T. Yoneda, “Verification of Timed Circuits with Failure Directed Abstractions”, *21st International Conference on Computer Design*, pp 28–35, 2003

Appendix A

A.1. Relative timing constraints verification script for PCHB

```
##############################################################
# Primetime script to verify isochronic forks
##############################################################
set verilog_netlist c3540.qdi.v
set top_level_module c3540

echo "Script starting at [date]"
set_message_info -id PTE-070 -limit 1
set_message_info -id PTE-060 -limit 1
set_message_info -id PTE-012 -limit 1
set_message_info -id UITE-130 -limit 1
set_message_info -id UITE-137 -limit 1

#set variable so only non-inverted is propagated
set timing_non_unate_clock_compatibility true

#Don't report warnings PTE-060
set timing_disable_clock_gating_checks true

set log run2.log
source ~/silistix/summer_2007_MP_work/tcl/get_drivers_loads.tcl
source ~/silistix/summer_2007_MP_work/tcl/custom_procedures.tcl

read_lib ~mallikap/proteus/proteus_sta/qdi_example/primetime_USC/qdi.lib

read_verilog $verilog_netlist
set link_path "PROTEUS $verilog_netlist"
link_design -keep_sub_designs $top_level_module

echo "Starting circuit analysis at [date]" >> $log
## Create clocks
foreach_in_collection l [ get_cells -hierarchical *]  {
        set lib_name_for_clock [get_attribute -class cell $l ref_name]
        set cell_name_for_clock [get_object $l]
        if  { [regexp {LOGIC([1-6])} $lib_name_for_clock]} then {
                set clock_pin [get_pin $cell_name_for_clock/*en*]
                create_clock $clock_pin -period 100
        } elseif  { [regexp {BUF} $lib_name_for_clock]} then {
                set clock_pin [get_pin $cell_name_for_clock/R.0]
                create_clock $clock_pin -period 100
                set clock_pin [get_pin $cell_name_for_clock/R.1]
                create_clock $clock_pin -period 100
        }
}
```

```
foreach_in_collection k [all_inputs ] {create_clock $k -period 100}
report_clocks

##############################################################
# Break loops
##############################################################
foreach_in_collection x [get_lib_cells PROTEUS/*LOGIC*] {
            set_disable_timing -from en -to V $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*CTREE2*] {
            set_disable_timing -from a[1] -to x $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*CTREE3*] {
            set_disable_timing -from a[0] -to x $x
            set_disable_timing -from a[1] -to x $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*CTREE4*] {
            set_disable_timing -from a[0] -to x $x
            set_disable_timing -from a[1] -to x $x
            set_disable_timing -from a[2] -to x $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*BUF*] {
            set_disable_timing -from R.e -to R.0 $x
            set_disable_timing -from R.e -to R.1 $x
            set_disable_timing -from L.0 -to R.0 $x
            set_disable_timing -from L.1 -to R.1 $x
}
set count_num_logic_cells 0
set num_constraints 0

##############################################################
# Add set_data_checks for all false rails in all cells of type LOGIC
# Generate set_data_check report for each false rail
##############################################################
for {set rail 0} {$rail < 2} {incr rail} {
set list_constrained_pins [list]

  foreach_in_collection i [ get_cells -hierarchical "*"]  {
    set lib_name [get_attribute $i ref_name]
    set cell_name [get_object $i]

    ## if library name of $cell_name matches LOGIC
    if  { [regexp {LOGIC([1-6])} $lib_name match num_inputs] } then {
      set count_num_logic_cells [expr ($count_num_logic_cells + 1)]
      echo "\n-------------------------------------------" >> $log
      echo "Processing logic cell : $cell_name" >> $log
      echo "[date]" >> $log
      for {set x 0}  {$x < $num_inputs} {incr x} {
            echo "\nProcessing rail A[$x].$rail" >> $log
```

```
# get the drivers of the data rail A[$x].$rail
# for logic cell $cell_name
# get_drivers will return a collection of size 0
# if the data rail is a primary input
# otherwise the size will be a +ve integer
set coll_drivers [get_drivers $cell_name/A[$x].$rail]
echo "coll_drivers: [get_object_name $coll_drivers]" >> $log

# if the size of coll_drivers is 0,
# then the clock pin is the data rail itself.
# But we need the top level net or actual primary input
# using all_fanin to get that - this command causesa timing update
# and thus is slow. But it might be ok as it is only used for
# primary inputs.
if { [sizeof_collection $coll_drivers] == 0} {
  set clock_pin [get_object [index_collection [all_fanin -to $cell_name/A[$x].$rail -flat] 1]]
  echo "-- Found a primary input as fanin" >> $log
  set related_pin $cell_name/en
  set constrained_pin $cell_name/A[$x].$rail
  echo "\tClock: $clock_pin" >> $log
  echo "\tR_pin: $related_pin" >> $log
  echo "\tC_pin: $constrained_pin" >> $log
} else {
  # if the size of coll_drivers is not equal to 0,
  # get the cell that drives the data rail
  set fanin [get_cells -of [get_drivers $cell_name/A[$x].$rail]]
  query_objects $fanin >> $log
  set fanin_lib [get_attribute [get_cells [get_object $fanin]] ref_name]

  # if the cell is of type BUF or TOK_BUF
  # the corresponding output rail of that cell is the clock pin
  # if the cell is of type LOGIC
  # the pin 'en' of that cell is the clock pin
  if {[regexp {BUF} $fanin_lib match]} {
    echo "-- Found special cell as fanin: [get_object $fanin]" >> $log

    # Get the clock pin
    if {[regexp {R.$rail} [get_object_name $coll_drivers]]} {
          set clock_pin [get_object $fanin]/R.$rail
    } else {
          set clock_pin [get_object $fanin]/R.[expr !$rail]
    }
    set related_pin $cell_name/en
    set constrained_pin $cell_name/A[$x].$rail
    echo "\tClock: $clock_pin" >> $log
    echo "\tR_pin: $related_pin" >> $log
    echo "\tC_pin: $constrained_pin" >> $log
  } else {
    echo "-- Found logic cell as fanin: [get_object $fanin]" >> $log

    # disable en to theother rail
```

```
                if {[regexp X.$rail [get_object_name $coll_drivers]]} {
                  #---------------------------------------------------------
                  echo "-- Rails not twisted rail - $rail" >> $log
                  if {[set_disable_timing -from en -to X.[expr !$rail] [get_object $fanin]]} {
                        echo "-- Arc disabled from en to X.[expr !$rail] on [get_object $fanin]" >> $log
                  } else {
                        echo "*ERROR*:Couldn't disable en to X.[expr !$rail] on [get_object $fanin]" >>
$log
                  }
                  remove_disable_timing -from en -to X.$rail [get_object $fanin]
                  set clock_pin [get_object $fanin]/en
                  set related_pin $cell_name/en
                  set constrained_pin $cell_name/A[$x].$rail
                  echo "\tClock: $clock_pin" >> $log
                  echo "\tR_pin: $related_pin" >> $log
                  echo "\tC_pin: $constrained_pin" >> $log
                } else  {
                  echo "-- Rails twisted rail - $rail" >> $log
                  #---------------------------------------------------------
                  if {[set_disable_timing -from en -to X.[expr !$rail] [get_object $fanin]]} {
                        echo "-- Arc disabled from en to X.[expr !$rail] on [get_object $fanin]" >> $log
                  } else {
                        echo "*ERROR*:Couldn't disable en to X.[expr !$rail] on [get_object $fanin]" >>
$log
                  }
                  remove_disable_timing -from en -to X.$rail [get_object $fanin]
                  set clock_pin [get_object $fanin]/en
                  set related_pin $cell_name/en
                  set constrained_pin $cell_name/A[$x].[expr !$rail]
                  echo "\tClock: $clock_pin" >> $log
                  echo "\tR_pin: $related_pin" >> $log
                  echo "\tC_pin: $constrained_pin" >> $log
                }
               }
        }
            if {![set_data_check -clock $clock_pin \
                        -rise_from $related_pin \
                           -fall_to $constrained_pin -setup 0.5]} {
              echo "*ERROR*:set_data_check failed on $constrained_pin" >> $log
            } else {lappend list_constrained_pins $constrained_pin}
      }
     }
   }
##############################################################
# Generate timing reports
##############################################################
  echo "Starting report_timing at [date]" >> $log
  set num_constraints [expr $num_constraints + [llength $list_constrained_pins]]
  foreach constrained_pin $list_constrained_pins {
            echo "Timing report for  : $cell_name" >> $log
            echo "Constrained pin: $constrained_pin" >> $log
            report_timing -to $constrained_pin >> $log
  }
  echo "Finished report_timing at [date]" >> $log
}
echo "Finished circuit analysis at [date]" >> $log
echo "Number of logic cells = $count_num_logic_cells" >> $log
echo "Number of constraints verified = $num_constraints" >> $log
exit
```

A.2. Performance verification script for PCHB

```
###############################################################
# Primetime script to verify performance
###############################################################
set verilog_netlist s1423.qdi.v
set top_level_module s1423

echo "Script starting at [date]"
set_message_info -id PTE-070 -limit 1
set_message_info -id PTE-060 -limit 1
set_message_info -id PTE-012 -limit 1
set_message_info -id UITE-130 -limit 1
set_message_info -id UITE-137 -limit 1

#set variable so only non-inverted is propagated
set timing_non_unate_clock_compatibility true

#Don't report warnings PTE-060
set timing_disable_clock_gating_checks true
set log run.log
source ~/silistix/summer_2007_MP_work/tcl/get_drivers_loads.tcl
source ~/silistix/summer_2007_MP_work/tcl/custom_procedures.tcl
read_lib ~mallikap/proteus/proteus_sta/qdi_example/primetime_USC/qdi.lib
read_verilog $verilog_netlist
set link_path "PROTEUS $verilog_netlist"
link_design -keep_sub_designs $top_level_module

echo "Starting circuit analysis at [date]" >> $log
set from_to_list [list]

foreach_in_collection m [all_inputs] {
  if { ![regexp {RESET} [get_object_name $m] match]} {
  set loads [get_loads $m]
  foreach_in_collection n $loads {
    set cell_name [get_object_name [get_cells -of $n]]
    set lib_name [get_attribute -class cell $cell_name ref_name]
    if  { [regexp {LOGIC} $lib_name match]} then {
      echo "\n------------------------------------------" >> $log
      echo "Processing boundary logic cell : $cell_name" >> $log

      set from_pin [get_object_name $m]
      set to_pin $cell_name/X.0
      echo "From pin : $from_pin" >> $log
      echo "To pin : $to_pin \n" >> $log
      lappend from_to_list [list $from_pin $to_pin]
      if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
      }
      set to_pin $cell_name/X.1
      echo "From pin : $from_pin" >> $log
      echo "To pin : $to_pin \n" >> $log
      lappend from_to_list [list $from_pin $to_pin]
      if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
      }

    } elseif { [regexp {BUF} $lib_name match num_inputs]} then {
      echo "\n-------------------------------------------" >> $log
      echo "Processing boundary BUF cell : $cell_name" >> $log
      set from_pin [get_object_name $m]
      set to_pin $cell_name/R.0
      echo "From pin : $from_pin" >> $log
      echo "To pin : $to_pin \n" >> $log
      lappend from_to_list [list $from_pin $to_pin]
      if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
      }
      set to_pin $cell_name/R.1
      echo "From pin : $from_pin" >> $log
      echo "To pin : $to_pin \n" >> $log
      lappend from_to_list [list $from_pin $to_pin]
      if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
      }
    }
  }
}
}

foreach_in_collection l [ get_cells -hierarchical *]  {
  set lib_name [get_attribute -class cell $l ref_name]
  set cell_name [get_object $l]

  if  { [regexp {LOGIC([1-6])} $lib_name match num_inputs]} then {
    echo "\n-------------------------------------------" >> $log
    echo "Processing logic cell : $cell_name" >> $log

    for {set x 0} {$x < $num_inputs} {incr x} {
      echo "\nProcessing rail A[$x].0" >> $log
      set coll_drivers [get_drivers $cell_name/A[$x].0]
      echo "coll_drivers: [get_object_name $coll_drivers]" >> $log
      #set max delay constraint from input rail to output rail
      echo "Max delay constraints from datain to dataout" >> $log

      if {[sizeof_collection $coll_drivers] != 0} {
              #from_pin is not a primary input
              set from_pin [get_object $coll_drivers]
        set to_pin $cell_name/X.0
        echo "From pin : $from_pin" >> $log
        echo "To pin : $to_pin \n" >> $log
        lappend from_to_list [list $from_pin $to_pin]
        if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
        }

        set to_pin $cell_name/X.1
        echo "From pin : $from_pin" >> $log
        echo "To pin : $to_pin \n" >> $log
        lappend from_to_list [list $from_pin $to_pin]
        if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
              echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin"  >> $log
        }
        #--------------------------------------------
        #set max delay constraint from en to previous en
        echo "Max delay constraints from en to previous en" >> $log
        set from_pin $cell_name/en
        echo "From pin : $from_pin" >> $log
        set to_cell [get_object_name [get_cells -of [get_object_name [get_drivers $cell_name/A[$x].0]]]]
        set to_cell_lib_name [get_attribute -class cell $to_cell ref_name]
        if { [regexp {LOGIC} $to_cell_lib_name match]} {
              set to_pin $to_cell/en
          echo "To pin : $to_pin \n" >> $log
          lappend from_to_list [list $from_pin $to_pin]
        } elseif  { [regexp {BUF} $to_cell_lib_name match]} {
              set to_pin $to_cell/R.e
          echo "To pin : $to_pin \n" >> $log
          lappend from_to_list [list $from_pin $to_pin]
        }
        if {![set_max_delay 1.0 -fall_from $from_pin -rise_to $to_pin]} {
               echo "*Error* set_max_delay -fall_from  $from_pin -rise_to $to_pin" >> $log
        }
      }
    }
    #--------------------------------------------
    #set max delay constraint from en to output rail
    echo "Max delay constraints from en to dataout" >> $log
    set from_pin $cell_name/en
    set to_pin $cell_name/X.0
    echo "From pin : $from_pin" >> $log
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
    if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
      echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
    }
    set to_pin $cell_name/X.1
    echo "From pin : $from_pin" >> $log
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
    if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
       echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
    }
```

```
  #--------------------------------------------
  #set max delay constraint from output rail to en
  echo "Max delay constraints from dataout to en" >> $log

  set to_pin $cell_name/en
  set from_pin $cell_name/X.0
  echo "From pin : $from_pin" >> $log
  echo "To pin : $to_pin " >> $log
  lappend from_to_list [list $from_pin $to_pin]
  if {![set_max_delay 1.0 -rise_from $from_pin -fall_to $to_pin]} {
    echo "*Error* set_max_delay -rise_from  $from_pin -fall_to $to_pin" >> $log
  }

  set from_pin $cell_name/X.1
  echo "From pin : $from_pin" >> $log
  echo "To pin : $to_pin " >> $log
  lappend from_to_list [list $from_pin $to_pin]
  if {![set_max_delay 1.0 -rise_from $from_pin -fall_to $to_pin]} {
    echo "*Error* set_max_delay -rise_from  $from_pin -fall_to $to_pin" >> $log
  }
}

if  { [regexp {BUF} $lib_name match]} then {
  echo "\n--------------------------------------------" >> $log
  echo "Processing logic cell : $cell_name" >> $log
  echo "\nProcessing rail L.0" >> $log
  set coll_drivers [get_drivers $cell_name/L.0]
  echo "coll_drivers: [get_object_name $coll_drivers]" >> $log
  #----------------------------------------------------
  #set max delay constraint from input rail to output rail
  echo "Max delay constraints from datain to dataout" >> $log

  if {[sizeof_collection $coll_drivers] != 0} {
    #from_pin is not primary input
    set from_pin [get_object $coll_drivers]
    set to_pin $cell_name/R.0
    echo "From pin : $from_pin" >> $log
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
    if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
      echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
    }
    set to_pin $cell_name/R.1
    echo "From pin : $from_pin" >> $log
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
    if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
      echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
    }
    #--------------------------------------------
    #set max delay constraint from R.e to previous en
  set from_pin $cell_name/R.e
  set to_cell [get_object_name [get_cells -of [get_object_name [get_drivers $cell_name/L.0]]]]
  set to_cell_lib_name [get_attribute -class cell $to_cell ref_name]
  if { [regexp {LOGIC} $to_cell_lib_name match]} {
    echo "Max delay constraints from R.e to previous en/R.e" >> $log
    echo "From pin : $from_pin" >> $log
        set to_pin $to_cell/en
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
  } elseif  { [regexp {BUF} $to_cell_lib_name match]} {
    echo "Max delay constraints from R.e to previous en/R.e" >> $log
    echo "From pin : $from_pin" >> $log
        set to_pin $to_cell/R.e
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
  }
  if {![set_max_delay 1.0 -fall_from $from_pin -rise_to $to_pin]} {
        echo "*Error* set_max_delay -fall_from  $from_pin -rise_to $to_pin" >> $log
  }
}

#--------------------------------------------
#set max delay constraint from R.e to output rail
echo "Max delay constraints from R.e to dataout" >> $log

set from_pin $cell_name/R.e
set to_pin $cell_name/R.0
echo "From pin : $from_pin" >> $log
echo "To pin : $to_pin \n" >> $log
lappend from_to_list [list $from_pin $to_pin]
if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
  echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" \>> $log
}
set to_pin $cell_name/R.1
echo "From pin : $from_pin" >> $log
echo "To pin : $to_pin \n" >> $log
lappend from_to_list [list $from_pin $to_pin]
if {![set_max_delay 1.0 -rise_from $from_pin -rise_to $to_pin]} {
  echo "*Error* set_max_delay -rise_from  $from_pin -rise_to $to_pin" >> $log
}

#--------------------------------------------
#set max delay constraint from output rail to R.e
echo "Max delay constraints from dataout to R.e" >> $log
set from_pin $cell_name/R.0
set to_pin $cell_name/R.e
echo "From pin : $from_pin" >> $log
echo "To pin : $to_pin \n" >> $log
lappend from_to_list [list $from_pin $to_pin]
if {![set_max_delay 1.0 -rise_from $from_pin -fall_to $to_pin]} {
      echo "*Error* set_max_delay -rise_from  $from_pin -fall_to $to_pin" >> $log
    }
    set from_pin $cell_name/R.1
    echo "From pin : $from_pin" >> $log
    echo "To pin : $to_pin \n" >> $log
    lappend from_to_list [list $from_pin $to_pin]
    if {![set_max_delay 1.0 -rise_from $from_pin -fall_to $to_pin]} {
      echo "*Error* set_max_delay -rise_from  $from_pin -fall_to $to_pin" >> $log
    }
  }
}

for {set p 0} {$p < [llength $from_to_list]} {incr p} {
    echo "Timing report from [lindex $from_to_list $p 0] to [lindex $from_to_list $p 1]" >> $log
    report_timing -from [lindex $from_to_list $p 0] -to [lindex $from_to_list $p 1]
}

echo "Number of constraints verified = [llength $from_to_list]" >> $log
echo "Finished constraint verification at [date]" >> $log
exit
```

A.3. Relative timing constraints verification script for MLD

```
################################################################
# Primetime script to verify RT Constraints for MLD
################################################################
set verilog_netlist c3540.mld.v
set top_level_module c3540

echo "Script starting at [date]"
set_message_info -id PTE-070 -limit 1
set_message_info -id PTE-060 -limit 1
set_message_info -id PTE-012 -limit 1
set_message_info -id UITE-130 -limit 1
set_message_info -id UITE-137 -limit 1

#set variable so only non-inverted is propagated
set timing_non_unate_clock_compatibility true

#Don't report warnings PTE-060
set timing_disable_clock_gating_checks true

set log run.log
source ~/silistix/summer_2007_MP_work/tcl/get_drivers_loads.tcl
source ~/silistix/summer_2007_MP_work/tcl/custom_procedures.tcl
read_lib ~mallikap/proteus/proteus_sta/mld_example/mld.lib
#read_lib ~mallikap/proteus/proteus_sta/mld_example/threebufs_ring_mld/mld.lib
read_verilog $verilog_netlist
set link_path "PROTEUS $verilog_netlist"
link_design -keep_sub_designs $top_level_module

echo "Starting circuit analysis at [date]" >> $log
## Create clocks
foreach_in_collection l [ get_cells -hierarchical *]  {
        set lib_name_for_clock [get_attribute -class cell $l ref_name]
        set cell_name_for_clock [get_object $l]
        if  { [regexp {CTRL} $lib_name_for_clock]} then {
                set clock_pin [get_pin $cell_name_for_clock/_pc]
                create_clock $clock_pin -period 0
        }
}

foreach_in_collection k [all_inputs ] {create_clock $k -period 0}
report_clocks

################################################################
# Break loops
################################################################
foreach_in_collection x [get_lib_cells PROTEUS/*V_LOGIC*] {
        set_disable_timing -from eval -to V $x
        set_disable_timing -from eval -to X.0 $x
```

```
            set_disable_timing -from eval -to X.1 $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*TOK_BUF*] {
            set_disable_timing -from eval -to V $x
            set_disable_timing -from eval -to R.0 $x
            set_disable_timing -from eval -to R.1 $x
            set_disable_timing -from L.0 -to R.0 $x
            set_disable_timing -from L.1 -to R.1 $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*.LOGIC*] {
            set_disable_timing -from en -to X.0 $x
            set_disable_timing -from en -to X.1 $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*CTRL*] {
#           set_disable_timing -from L.0 -to L.e $x
}
foreach_in_collection x [get_lib_cells PROTEUS/*COMPLETE*] {
            set_disable_timing -from eval -to V $x
}
check_timing -verbose >> $log
report_disable_timing >> $log
set count_num_logic_cells 0
set num_constraints 0

#################################################################
# Add set_data_checks for all false rails in all cells of type LOGIC
# Generate set_data_check report for each false rail
#################################################################
for {set rail 0} {$rail < 2} {incr rail} {
set list_constrained_pins [list]

  foreach_in_collection i [ get_cells -hierarchical "*"]  {
    set lib_name [get_attribute $i ref_name]
    set cell_name [get_object $i]

    ## if library name of $cell_name matches LOGIC
    if  { [regexp {V_LOGIC} $lib_name] } then {
      echo "\n-------------------------------------------" >> $log
      echo "Processing v_logic cell : $cell_name" >> $log
      set clock_pin [get_drivers $cell_name/_pc]
      echo "Clock pin : [get_object $clock_pin]" >> $log
      set coll_fanout [get_loads $cell_name/X.$rail]
      echo "\nFanout for X.$rail : " >> $log
      query_objects $coll_fanout >> $log

      if {[sizeof_collection $coll_fanout] == 0} {
            echo "Found primary output as fanout. " >> $log
      } else {
        foreach_in_collection c_pin $coll_fanout {
              lappend list_constrained_pins $c_pin
              set fanout_cell [get_cells -of $c_pin]
```

```
                echo "Fanout cell : [get_object_name $fanout_cell]" >> $log
                set fanout_cell_lib [get_attribute $fanout_cell ref_name]
                if  { [regexp {V_LOGIC} $fanout_cell_lib] || \
                   [regexp {TOK_BUF} $fanout_cell_lib] } then {
                           set related_pin [get_object $fanout_cell]/eval
                           echo "Related pin : $related_pin" >> $log
                           echo "Constrained pin : [get_object $c_pin]" >> $log
                } elseif {[regexp {.LOGIC} $fanout_cell_lib]} {
                           set related_pin [get_object $fanout_cell/en]
                           echo "Related pin : $related_pin" >> $log
                           echo "Constrained pin : [get_object $c_pin]" >> $log
                }
                if {![set_data_check -clock [get_object $clock_pin] \
                                     -rise_from $related_pin \
                                     -fall_to $c_pin -setup 0.5]} {
                  echo "*ERROR*:set_data_check failed on $c_pin" >> $log
                }
        }
      }
    }
  }
################################################################
# Generate timing reports
################################################################
  echo "Starting report_timing at [date]" >> $log
  set num_constraints [expr $num_constraints + [llength $list_constrained_pins]]
  foreach constrained_pin $list_constrained_pins {
            echo "Constrained pin: [get_object $constrained_pin]" >> $log
            report_timing -to $constrained_pin >> $log
  }
  echo "Finished report_timing at [date]" >> $log
}
echo "Finished circuit analysis at [date]" >> $log
echo "Number of constraints verified = $num_constraints" >> $log
exit
```

A.4. Performance verification script for MLD

```
##############################################################
# Primetime script to verify performance
##############################################################
set verilog_netlist c3540.mld.v
set top_level_module c3540

echo "Script starting at [date]"
set_message_info -id PTE-070 -limit 1
set_message_info -id PTE-060 -limit 1
set_message_info -id PTE-012 -limit 1
set_message_info -id UITE-130 -limit 1
set_message_info -id UITE-137 -limit 1

#set variable so only non-inverted is propagated
set timing_non_unate_clock_compatibility true

#Don't report warnings PTE-060
set timing_disable_clock_gating_checks true

set log run.log
source ~/silistix/summer_2007_MP_work/tcl/get_drivers_loads.tcl
source ~/silistix/summer_2007_MP_work/tcl/custom_procedures.tcl
read_lib ~mallikap/proteus/proteus_sta/mld_example/mld.lib
read_verilog $verilog_netlist
set link_path "PROTEUS $verilog_netlist"
link_design -keep_sub_designs $top_level_module

echo "Starting circuit analysis at [date]" >> $log
set list_ctrl [list]
set from_to_list [list]

foreach_in_collection l [ get_cells -hierarchical *]  {
  set lib_name [get_attribute -class cell $l ref_name]
  set cell_name [get_object $l]

  if  { [regexp {CTRL} $lib_name match num_inputs]} then {
    #echo "\n------------------------------------------" >> $log
    #echo "Processing ctrl cell : $cell_name" >> $log
       set ctrl_fanouts [list]
    set ctree_level1 [list]
    set ctree_level2 [list]
    set ctree_level3 [list]

    #process ctrl cell
    lappend ctrl_fanouts $cell_name

    #get driver of V which is the output of the PVD
    #get cells driven by PVD output
```

```
set loads_V [get_loads [get_driver $cell_name/V]]
#echo "loads_V :" >> $log
query_objects $loads_V >> $log
for {set j 1} {$j < [sizeof_collection $loads_V]} {incr j} {
  set i [get_object_name [index_collection $loads_V $j]]
  set cell_load_V [get_cells -of $i]
  set i_lib_name [get_attribute -class cell $cell_load_V ref_name]
  if  { [regexp {CTREE} $i_lib_name match]} then {
        lappend ctree_level1 $i
    #echo "-Found level1 CTREE [get_object_name [get_cells -of $i]]" >> $log
  } elseif { [regexp {CTRL} $i_lib_name match]} then {
        lappend ctrl_fanouts [get_object_name [get_cells -of $i]]
    #echo "-Found CTRL [get_object_name [get_cells -of $i]]" >> $log
  }
}
if { [llength $ctree_level1] !=0 } {
  #echo "--Processing ctree_level1" >> $log
  foreach i $ctree_level1 {
    #echo "--Processing $i" >> $log
    set temp1 [get_object_name [get_cells -of $i]]
        #echo "temp1: $temp1" >> $log
        set temp2 [get_loads  $temp1/x]
        if { [sizeof_collection $temp2] !=0} {
      set fanout_level1 [get_object_name [get_cells -of $temp2]]
      set fanout_level1_lib_name [get_attribute -class cell $fanout_level1 ref_name]
      if  { [regexp {CTREE} $fanout_level1_lib_name match]} then {
        lappend ctree_level2 $fanout_level1
        #echo "---Found level 2 CTREE $fanout_level1" >> $log
      } elseif { [regexp {CTRL} $fanout_level1_lib_name match]} then {
        lappend ctrl_fanouts $fanout_level1
        #echo "---Found CTRL $fanout_level1" >> $log
      }
        }
  }
}
if { [llength $ctree_level2] !=0 } {
  #echo "----Processing ctree_level2" >> $log
  #this list has cells only not ports
  foreach i $ctree_level2 {
    #echo "----Processing $i" >> $log
        set temp1 [get_loads  $i/x]
        if {  [sizeof_collection $temp1] !=0} {
      set fanout_level2 [get_object_name [get_cells -of $temp1]]
      set fanout_level2_lib_name [get_attribute -class cell $fanout_level2 ref_name]
      if  { [regexp {CTREE} $fanout_level2_lib_name match]} then {
        lappend ctree_level3 $fanout_level2
        #echo "-----Found level 3 CTREE $fanout_level2" >> $log
      } elseif { [regexp {CTRL} $fanout_level2_lib_name match]} then {
        lappend ctrl_fanouts $fanout_level2
        #echo "-----Found CTRL $fanout_level2" >> $log
      }
```

```
            }
        }
      }
      if { [llength $ctree_level3] !=0 } {
        #echo "------Processing ctree_level3" >> $log
        #this list has cells only not ports
        foreach i $ctree_level3 {
          #echo "------Processing $i" >> $log
                set temp1 [get_loads  $i/x]
                if {  [sizeof_collection $temp1] !=0} {
            set fanout_level3 [get_object_name [get_cells -of [get_loads $i/x]]]
            set fanout_level3_lib_name [get_attribute -class cell $fanout_level3 ref_name]
            if  { [regexp {CTREE} $fanout_level3_lib_name match]} then {
              lappend ctree_level4 $fanout_level3
              #echo "-------Found level 4 CTREE $fanout_level3" >> $log
            } elseif { [regexp {CTRL} $fanout_level3_lib_name match]} then {
              lappend ctrl_fanouts $fanout_level3
              #echo "-------Found CTRL $fanout_level3" >> $log
            }
                }
        }
      }
      lappend list_ctrl $ctrl_fanouts
    }
  }
################################################################
# 1. constrain segments from v_logic to v_logic based on list_ctrl
# 2. constrain segments from V(n) to eval(n)
# 3. constrain segments from V(n) to _pc(n)
################################################################
#echo "\n-----------------------------------------------------------------" >> $log
#echo "Setting up constraints from v_logic data rails to v_logic data rails\n" >> $log
#grab all v_logics driven by controller 1's eval

for {set p 0} {$p < [llength $list_ctrl]} {incr p} {
  #echo "-----------------------------------" >> $log
  #echo "Processing [lindex $list_ctrl $p 0]" >> $log
  set from2 [get_object_name [get_drivers [lindex $list_ctrl $p 0]/V]]
  #echo "from2 : $from2" >> $log
  set to4 $from2

  set coll_eval_loads [get_loads [lindex $list_ctrl $p 0]/eval]
  if {[sizeof_collection $coll_eval_loads]!=0} {
  query_objects $coll_eval_loads >> $log
  foreach_in_collection i $coll_eval_loads {
    set v_logic_cell [get_object_name [get_cells -of $i]]
    set v_logic_cell_lib [get_attribute -class cell $v_logic_cell ref_name]
      if { [regexp {V_LOGIC} $v_logic_cell_lib match]} {
        #echo "Found v_logic cell $v_logic_cell in fanin" >> $log
        #foreach of these v_logic
        #set it's dataouts as from_pins
  set from_false $v_logic_cell/X.0
  set from_true $v_logic_cell/X.1
  #echo "from_false $v_logic_cell/X.0" >> $log
  #echo "from_true $v_logic_cell/X.1" >> $log

  set to2 [get_object_name $i]
  #echo "to2: $to2" >> $log
  set_max_delay 8 -rise_from $from2 -fall_to $to2
  lappend from_to_list [list -rise_from $from2 -fall_to $to2]
  set to3 $v_logic_cell/_pc
  echo "to3: $to3" >> $log
  set_max_delay 8 -fall_from $from2 -rise_to $to3
  lappend from_to_list [list -fall_from $from2 -rise_to $to3]
      set from4 $v_logic_cell/V
      echo "from4 : $from4" >> $log
      echo "to4 : $to4" >> $log
  set_max_delay 8 -rise_from $from4 -rise_to $to4
  lappend from_to_list [list -rise_from $from4 -rise_to $to4]
} elseif { [regexp {BUF} $v_logic_cell_lib match]} {
  echo "Found tok cell $v_logic_cell in fanin" >> $log
  #foreach of these v_logic
  #set it's dataouts as from_pins
  set from_false $v_logic_cell/R.0
  set from_true $v_logic_cell/R.1
  echo "from_false $v_logic_cell/R.0" >> $log
  echo "from_true $v_logic_cell/R.1" >> $log

  set to2 [get_object_name $i]
  #echo "to2: $to2" >> $log
  set_max_delay 8 -rise_from $from2 -fall_to $to2
  lappend from_to_list [list -rise_from $from2 -fall_to $to2]

  set to3 $v_logic_cell/_pc
  echo "to3: $to3" >> $log
  set_max_delay 8 -fall_from $from2 -rise_to $to3
  lappend from_to_list [list -fall_from $from2 -rise_to $to3]

      set from4 $v_logic_cell/V
      echo "from4 : $from4" >> $log
      echo "to4 : $to4" >> $log
  set_max_delay 8 -rise_from $from4 -rise_to $to4
  lappend from_to_list [list -rise_from $from4 -rise_to $to4]
} else { continue}

for {set i 1} {$i < [llength [lindex $list_ctrl $p]]} {incr i} {
#grab all v_logics driven by fanout controller's eval
set coll_feval_loads [get_loads [lindex $list_ctrl $p $i]/eval]
foreach_in_collection j $coll_feval_loads {
  set fv_logic_cell [get_object_name [get_cells -of $j]]
  set fv_logic_cell_lib [get_attribute -class cell $fv_logic_cell ref_name]
  if { [regexp {V_LOGIC} $fv_logic_cell_lib match]} {
          echo "Found v_logic cell $fv_logic_cell in fanout" >> $log
          set to_false $fv_logic_cell/X.0
          set to_true $fv_logic_cell/X.1
          echo "to_false $fv_logic_cell/X.0" >> $log
          echo "to_true $fv_logic_cell/X.1" >> $log
          #issue set_max_delay on both rails
          lappend from_to_list [list -rise_from $from_false -rise_to $to_false]
          set_max_delay 2 -rise_from $from_false -rise_to $to_false
          lappend from_to_list [list -rise_from $from_false -rise_to $to_true]
          set_max_delay 2 -rise_from $from_false -rise_to $to_true
          lappend from_to_list [list -rise_from $from_true -rise_to $to_false]
          set_max_delay 2 -rise_from $from_true -rise_to $to_false
          lappend from_to_list [list -rise_from $from_true -rise_to $to_true]
          set_max_delay 2 -rise_from $from_true -rise_to $to_true
        } elseif { [regexp {BUF} $fv_logic_cell_lib match]} {
          echo "Found tok cell $fv_logic_cell in fanout" >> $log
          set to_false $fv_logic_cell/R.0
          set to_true $fv_logic_cell/R.1
          echo "to_false $fv_logic_cell/R.0" >> $log
          echo "to_true $fv_logic_cell/R.1" >> $log
          #issue set_max_delay on both rails
          lappend from_to_list [list -rise_from $from_false -rise_to $to_false]
          set_max_delay 8 -rise_from $from_false -rise_to $to_false
          lappend from_to_list [list -rise_from $from_false -rise_to $to_true]
          set_max_delay 8 -rise_from $from_false -rise_to $to_true
          lappend from_to_list [list -rise_from $from_true -rise_to $to_false]
          set_max_delay 8 -rise_from $from_true -rise_to $to_false
          lappend from_to_list [list -rise_from $from_true -rise_to $to_true]
          set_max_delay 8 -rise_from $from_true -rise_to $to_true
        } else { continue}
      }
    }
  }
}
}
################################################################
#constrain segments from V(n) to L.0(n+1) based on all fanouts of controller(n)
################################################################
echo "Setting up constraints from V(n) to L.0(n+1)\n" >> $log
for {set p 0} {$p < [llength $list_ctrl]} {incr p} {
  #echo "------------------------------------" >> $log
  #echo "Processing [lindex $list_ctrl $p 0]" >> $log
  for {set q 1} {$q < [llength [lindex $list_ctrl $p]]} {incr q} {
    set from [get_object_name [get_drivers [lindex $list_ctrl $p 0]/V]]
    #echo "from [get_object_name [get_drivers [lindex $list_ctrl $p 0]/V]]" >> $log
    set to [lindex $list_ctrl $p $q]/L.0
    #echo "to [lindex $list_ctrl $p $q]/L.0" >> $log
    set_max_delay 8 -rise_from $from -rise_to $to
    lappend from_to_list [list -rise_from $from -rise_to $to]
  }
}
```

```
for {set p 0} {$p < [llength $from_to_list]} {incr p} {
   #echo "Timing report from [lindex $from_to_list $p 1] to [lindex $from_to_list $p 3]" >> $log
   report_timing -from [lindex $from_to_list $p 1] \
                         -to [lindex $from_to_list $p 3]
}

echo "Number of constraints verified = [llength $from_to_list]" >> $log
echo "Finished constraint verification at [date]" >> $log
exit
```

A.5. Relative timing constraint verification for hierarchical design based circuits

```
# Script that verifies the external constraints on all S-elements
# Author: Mallika Prakash
# Date  : 07/03/07

set sh_enable_page_mode on
set timing_report_unconstrained_paths true
set timing_report_always_use_valid_start_end_points false
set_message_info -id PTE-070 -limit 1
set_message_info -id PTE-060 -limit 1
set_message_info -id PTE-012 -limit 1
set_message_info -id UITE-130 -limit 1
set_message_info -id UITE-137 -limit 1

#set variable so only non-inverted is propagated
set timing_non_unate_clock_compatibility true

#Don't report warnings PTE-060
set timing_disable_clock_gating_checks true

read_verilog design.v
read_verilog stm_artisan_sage.v
read_lib typical.lib
set link_path "typical stm_artisan_sage.v design.v"

# IMPORTANT
# link_design option -keep_sub_designs required to store all sub_designs in memory sothey can be
accessed later for set_data_checks

# top-level: fabric_adapt:
link_design -keep_sub_designs fabric_adapt
current_design fabric_adapt

# Source a new tcl proceure provided by synopsys: get_drivers and get_loads.
# These get the drivers and the loads of a net respectively.
# While creating clocks on the PODs, we want to make sure that they are
# created on actual ports and not on hierarchical pins which may be lost in
# a post-layout design.
source ../tcl/get_drivers_loads.tcl

set custom_tcl_path ~/primetime/tcl/
source $custom_tcl_path/custom_procedures.tcl
set logfile aug7.log
```

```
date > $logfile
##############################################################################
# Breaking some obvious feedback loops
##############################################################################
set_disable_timing -from A1 -to Y [get_lib_cells typical/OA21X4]
set_disable_timing -from A0 -to Y [get_lib_cells typical/AO21X4]
set_disable_timing -from A1 -to Y [get_lib_cells typical/AO22X4]
#set_disable_timing -from A1 -to Y [get_lib_cells typical/OAI21X4]
set_disable_timing -from A1 -to Y [get_lib_cells typical/OAI31X4]

##############################################################################
# Loop that retrieves all the different types of S-elements and finds all the
# instances for each type and creates a collection of them called coll_of_instances.
##############################################################################
echo "Creating collection of all S-element instances."  >> $logfile
set coll_of_selements [get_designs -hierarchical slbase_selement_1*]

# Create a collection of all the S-element instances in 4i4t
foreach_in_collection  x $coll_of_selements {
		set selement_type [get_object $x]
		append_to_collection  coll_of_instance  [get_cells  -hierarchical  -filter  "ref_name  ==
$selement_type"]
}

##############################################################################
    set curr_sub_set [list]
    foreach_in_collection  x $coll_of_instance {
		set instance [get_object $x]
		# create an empty list
		# list_header will store the clock/POD and the constrained pin
		# list_tail will store the fake related pins
		set list_header [list]
		set list_tail [list]
		set list_constraint [list]

        # get the instance name
		lappend list_header $instance

		# get the POD
		lappend list_header [get_object [get_drivers [get_pins $instance/oack]]]

		# append the constrained pin
		lappend list_header "$instance/I7_U1/A0"

		# create a collection of the fanouts
		set coll_of_related_pins [get_loads $instance/iack]

		# create a list of the fanouts using user-defined procedure coll_to_list
		set list_tail [coll_to_list $coll_of_related_pins]

		# concatenate the two lists to create a list that contains:
```

```
            # clock POD POC1 POC2 ...
            # append that to big_list
            set list_constraint [concat $list_header $list_tail]
            lappend curr_sub_set $list_constraint
            #echo "\n-------------------------------------------------------------" >> $logfile
            #echo "\nBuilding curr_sub_set: $curr_sub_set" >> $logfile
    }

set big_set [partition_constraints $curr_sub_set $logfile]

################################################################################
# Section that grabs a constraint from all the non-overlapping sets and
# verifies them with set_data_check commands
################################################################################
for {set sub_set_index 0} {$sub_set_index < [llength $big_set]} {incr sub_set_index} {
            echo "\n************| Starting verification of constraints in sub_set $sub_set_index
|************\n" >> $logfile
            set curr_sub_set [lindex $big_set $sub_set_index]

            # create clocks on the PODS
            echo "\nCreating clocks on the oack port of all S-elements of set $sub_set_index..." >> $logfile
            for {set constraint_index 0} {$constraint_index < [llength [lindex $big_set 0]]} {incr
constraint_index} {
               set constraint [lindex $curr_sub_set $constraint_index]
               set clock_pin [lindex $constraint 1]
               create_clock -period 0 [get_drivers [get_pins $clock_pin]]
            }
            set_false_path -rise_from [get_clocks *]
            echo "\nList of clocks created on the oack port of all S-elements of set $sub_set_index...." >>
$logfile
            report_clocks >> $logfile

            # Create clocks for the latches
            echo "\nCreating clocks for latches..."  >> $logfile
            create_clock [get_ports *clk*] -period 0

            for {set constraint_index 0} {$constraint_index < [llength [lindex $big_set 0]]} {incr
constraint_index} {
                        set constraint [lindex $curr_sub_set $constraint_index]
                        set instance_name [lindex $constraint 0]
                        set clock_pin [lindex $constraint 1]
                        set constrained_pin [lindex $constraint 2]
                        set num_related_pins [expr [llength $constraint]-3]
                        echo "\n-------------------------------------------------------------" >> $logfile
                        echo "S-element: $instance_name" >> $logfile
                        for {set related_pin_index 3} {$related_pin_index < [expr $num_related_pins+3]}
{incr related_pin_index} {
                                    set related_pin [lindex $constraint $related_pin_index]
                                    set_data_check -clock $clock_pin -rise_from $related_pin -fall_to
$constrained_pin -setup 0.2
```

```
                        set_data_check -clock $clock_pin -fall_from $related_pin -fall_to
$constrained_pin -setup 0.2
                        echo "clock_pin = $clock_pin" >> $logfile
                        echo "constrained_pin = $constrained_pin" >> $logfile
                        echo "related_pin = $related_pin\n" >> $logfile

                }

                set return_value [set_disable_timing -from A0 -to Y $instance_name/I7_U1]
                if {$return_value != 1} {
                        echo "\nThere was a problem with set_disable_timing on $instance_name"
>> $logfile
                } else {
                        echo "\nExecuted set_disable_timing on $instance_name successfully" >>
$logfile
                }
        }
        for {set constraint_index 0} {$constraint_index < [llength [lindex $big_set 0]]} {incr
constraint_index} {
            set constraint [lindex $curr_sub_set $constraint_index]
            set constrained_pin [lindex $constraint 2]
            set num_related_pins [expr [llength $constraint]-3]
            echo "num_related_pins = $num_related_pins" >> $logfile
            echo "report_timing launched at: [date]" >> $logfile
            report_timing -to $constrained_pin -nworst $num_related_pins  >> $logfile
            echo "report_timing completed at: [date]" >> $logfile
        }
        for {set constraint_index 0} {$constraint_index < [llength [lindex $big_set 0]]} {incr
constraint_index} {
            set return_value [remove_disable_timing -from A0 -to Y $instance_name/I7_U1]
            if {$return_value != 1} {
                echo "There was a problem with remove_disable_timing on $instance_name" >>
$logfile
            } else {
                echo "Executed remove_disable_timing on $instance_name successfully" >> $logfile
            }
            remove_clock *
        }
   }

echo " Run completed at [date]" >> $logfile
exit
```