

PrimeTime
Static Timing Analysis Tool

George Michael

Scholarly Paper – Fall 2006

Dr. Gaj

Dr. Barnes

George Mason University

Table of Contents

Introduction	3
Introduction to TCL.....	4
Timing Violations	6
Capacitance and Transition Violations.....	8
Temperature, Voltage, and Process	9
Timing Constraints	9
Clocks.....	10
Input/Output Timing.....	11
False Paths.....	13
Multi-Cycle Paths.....	14
Path Delay Calculation	15
Cell Delay.....	15
Net Delay	16
Generating PrimeTime Reports	17
Report_Timing	17
-from and -to.....	18
Report_Constraints	19
Report_clock_timing	19
Generic PrimeTime Script	20
Conclusion	22
References	23

Introduction

The question of how to ensure that when an ASIC is placed and routed, that every path in the design does not violate setup and hold time specification of the flip flops? This is where Synopsys's PrimeTime comes into the ASIC design flow. PrimeTime is a full chip static analysis tool that can fully analyze a multimillion gate ASIC in a short amount of time. The main advantage of PrimeTime is that does not use test vectors to simulate the critical path. This vector-less approach removes the possibility that not all critical paths are identified when writing the delay vectors. With many of the commands the same as Design-Compiler, making the addition of PrimeTime to the ASIC design flow very easy because the designer does not have a steep learning curve. PrimeTime has the ability to analyze a design over various temperatures, voltages, and process variations, will make a designer feel that they have produced a very robust design.

What makes PrimeTime so flexible and able to fit into complex design flows is that Primetime works with several file formats. PrimeTime will read in Verilog, VHDL, or EDIF netlists along with numerous delay formats, standard delay format (SDF) and standard parasitic format (SPEF). PrimeTime uses the proprietary database (db) files of the standard cell and macro library to determine the delay through the cell and transition of the output pin. The last file that PrimeTime needs is the Synopsys Design Constraints (SDC) file, which defines your ASIC to PrimeTime.

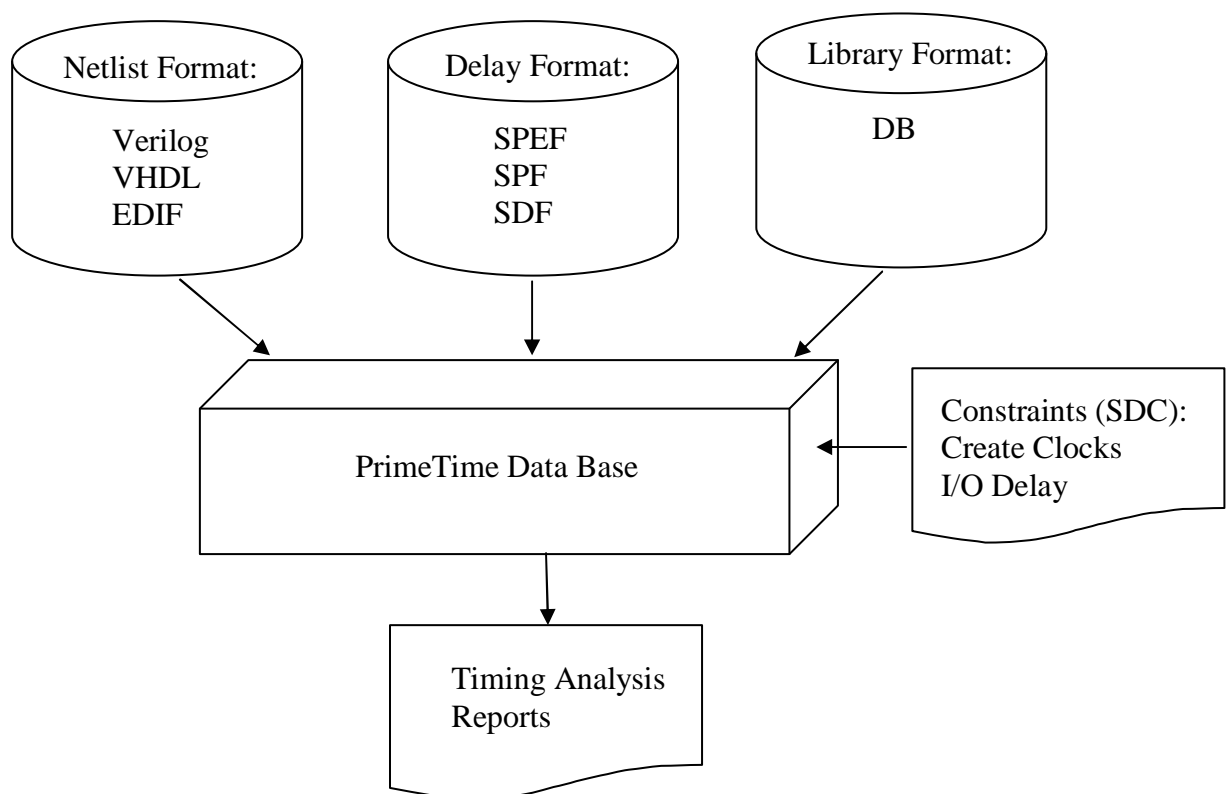


Figure 1 – PrimeTime Top Level Description

Introduction to TCL

PrimeTime is controlled through a scripting language called Tcl (tool command language). Tcl is pronounced like "tickle". Tcl is a very powerful but simple open source language that was created by John Ousterhout at the University of California at Berkeley [1]. PrimeTime has included some very usage variables (collections) that make interfacing with the database very easy. When running PrimeTime it is a good programming practice to create a script instead of typing each command individually. There are numerous websites on the internet and books that covers the tcl language in more detail then what is presented in this paper.

Creating variables in tcl is the same as creating variables in the c-shell UNIX environment.

```
pt_shell> set home_dir "/user/home/"
```

A very powerful feature of PrimeTime is that UNIX environment variables can be passed to PrimeTime. The same PrimeTime variable can be initialized from the UNIX environment variable \$HOME.

```
pt_shell> set home_dir $env(HOME)
```

When a design is read into PrimeTime, new design dependant variables (collections) are automatically created. The new variables allow for faster and simpler scripting of design constraints. For the following examples, let's use a straightforward design seen in figure 2.

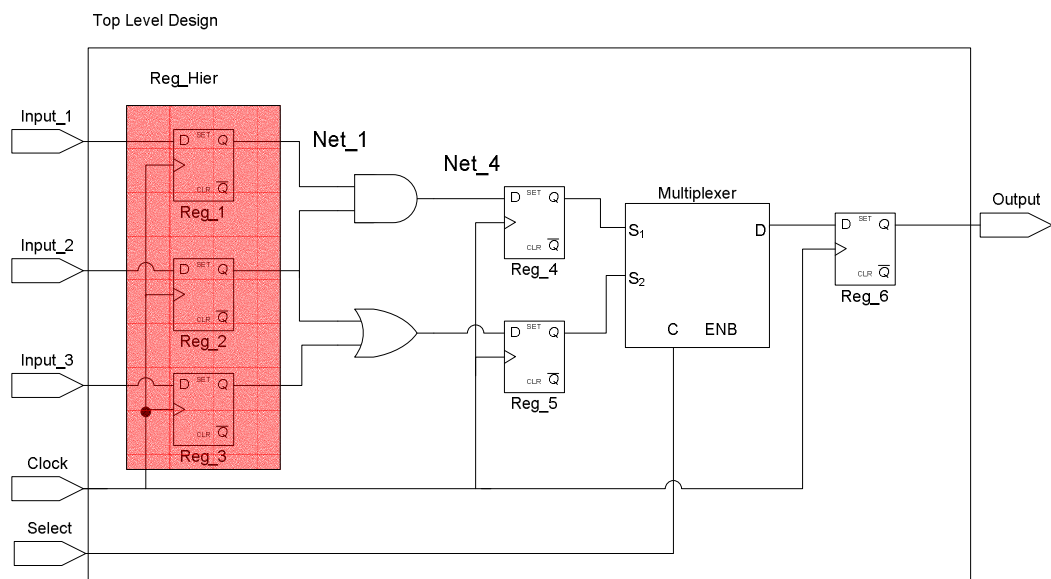


Figure 2 - Simple ASIC Design

This design has 5 primary inputs and 1 primary output and several register to register paths. Two new variables are created containing every primary input and output from the design.

```
pt_shell> all_input
          {Input_1 Input_2 Input_3 Clock Select}
pt_shell> all_output
          {Output}
```

When setting a constraint on a primary input, the users can either specify all the ports or individual ports by using `get_port` variable.

```
pt_shell> set_false_path -from [all_input] \
                    -to [all_output]

pt_shell> set_false_path -from [get_port Input_1] \
                    -to [all_output]
```

Every instance and net is also placed in variable in the database which can be accessed by the user. One note here, when setting a constraint, such as false path or generated clock, always set the constraint from the instance and not the net. Nets can be renamed or removed during the normal place and route process thereby creating errors in your PrimeTime scripts.

```
# Do not set the constraint using net names.
pt_shell> set_false_path -from [get_net net1] \
                    -to [get_net net4]

# The proper way to set a constraint is by using the
# instance and pin names.
pt_shell> set_false_path -from [get_pin reg_heir/reg_1/Q] \
                    -to [get_pin reg_4/D]
```

PrimeTime will accept regular expression when setting constraints, so if the user wants to apply the constraint to several instances or pins. With the constraint below both `reg_hier/reg_0` and `reg_hier/reg_1` instances would have a false path attribute set to any pin of instance `reg_4`.

```
pt_shell> set_false_path -from [get_pin reg_heir/reg_*/Q] \
                    -to [get_pin reg_4/*]
```

Any question about a PrimeTime command the user has can be found by typing `man` and the command name. If the user only wants to see the command inputs without the command description, simply type the command followed by `“-help”`

```
pt_shell> man set_false_path
pt_shell> set_false_path -help
```

Timing Violations

When PrimeTime analyzes a design, it is checking that each path meets the setup and hold specification for the design library that you specify. Along with setup and hold, PrimeTime will also check that transition to and from a gate is not outside of the timing table and that the capacitance on the output of a cell is not too large. PrimeTime will time every path twice to ensure that path does not cause a setup or hold violation whether the path is rising or falling.

Setup violations happen when data changes less than t_{Setup} nanoseconds before the rising edge of the clock. Hold violations are similar to setup violations but data changes less than t_{Hold} after the rising edge of the clock. There is a window around every rising clock edge that has a width of $(t_{\text{Setup}} + t_{\text{Hold}})$ where the data can not change. Changing the data inside this window will cause metastability inside of the flip flop. Depending on which side of the clock edge the data changes, determines if the data path violates setup or hold time.

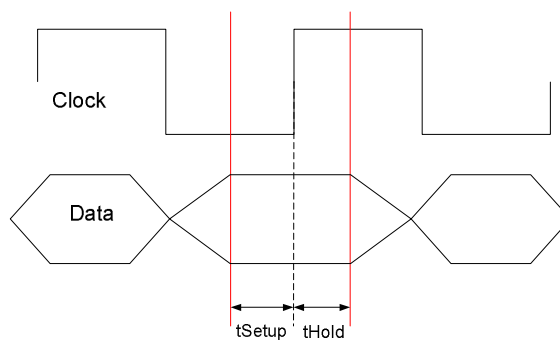


Figure 3 - Setup and Hold Diagram

Every path in the design is a register to registers, even input and output paths. The time through the combination logic must be less than one clock period minus the clock uncertainty, flip flop setup time, and time it takes to transfer the data to the Q pin when clock goes high. Clock uncertainty counts for clock skew (phase difference) between the launching flip flop and the capturing flip flop [3].

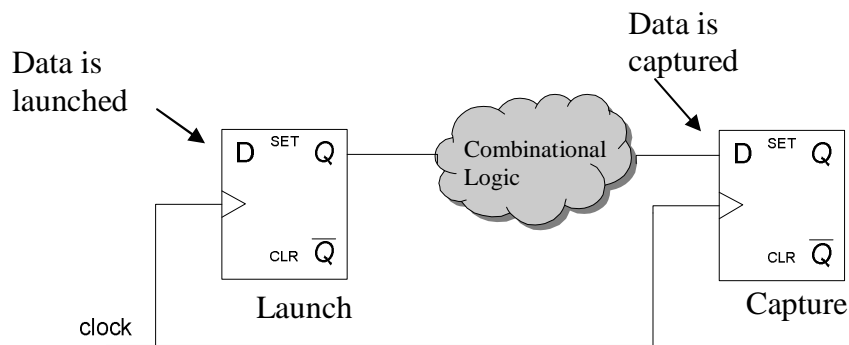


Figure 4 - Generic Register to Register Path

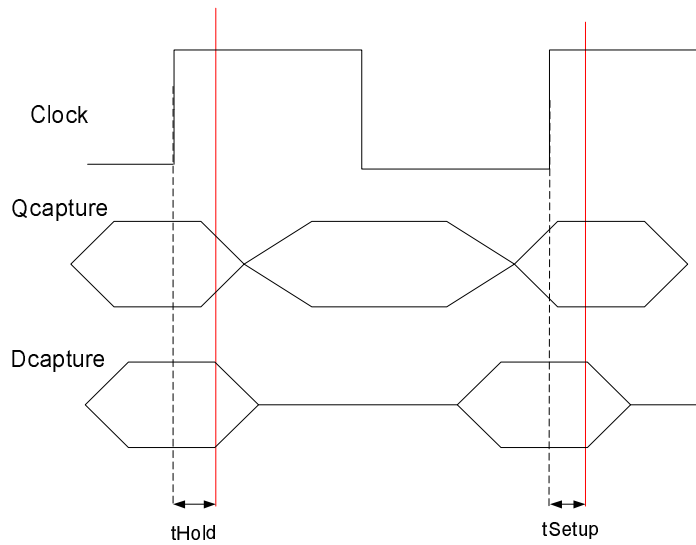


Figure 5 - Timing of Generic Register to Register Path

$$T_{\text{Combinational logic}} < (\text{Clock Period} - FF_{\text{launch}(\text{clk} \rightarrow Q)} - \text{Clock Uncertainty} - \text{Flip Flop } t_{\text{Setup}})$$

Calculating $T_{\text{combinational logic}}$ is very simple. The delay though the cell is added to the time thought the net. For given path below in figure 5,

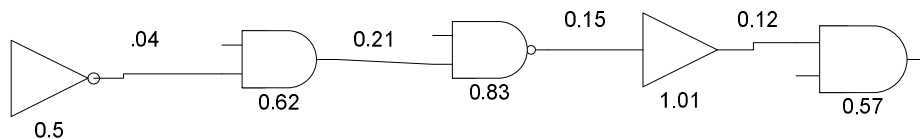


Figure 6 - Calculating $T_{\text{Combinational logic}}$

$$T_{\text{Combinational logic}} = (0.5 + 0.04 + 0.62 + 0.21 + 0.83 + 0.15 + 1.01 + 0.12 + 0.57) = 4.05 \text{ ns.}$$

How the delay through a book is calculated will be cover later on in this paper. The delay though the net is calculated by PrimeTime from data that is contained in the delay format file like a SPEF or SPF. A SPEF or SPF file contains extracted capacitance and resistance information based on the routing in the design.

Setup violations are essentially where the data path is too slow compared to the clock speed at the capture latch. With that in mind there are several things a designer can do to fix the setup violations.

- Reduce the amount of buffering in the path.
- Replace buffers with 2 inverters place farther apart
- Reduce larger than normal capacitance on a book's output pin.
- Increase the size of books to decrease the delay through the book.

- Make sure clock uncertainty is not too large for the technology library that you are using.
- Reduce clock speed. This is a poor design technique and should be used as a last resort.

Hold violations have the opposite problem of setup. Hold violations are caused when the data is not held long enough at the launch flip flop to ensure that data was clocked in completely. To state the problem differently, hold violations are caused when data is too fast when compared to the clock speed. If hold violations are not fixed before the chip is made, there is nothing that can be done post fabrication to fix hold problems unlike setup violation where the clock speed can be reduced.

To fix hold violations in the design, the designer needs to simply add more delay to the data path. This can be done by

- Adding buffers/inverter pairs/delay cells to the data path.
- Decreasing the size of certain buffers in the data path. It is better to reduce the buffers closer to the capture flip flop because there is less likelihood of affecting other paths and causing new errors.
- Add more capacitance to the output pin of buffers with light capacitance.

One thing to note when fixing setup or hold violations. Only make modification to the data path. Adjusting register location or removing/adding buffers to the clock path will fix the violation that you are looking at but will cause more violations that were not present before. The designer will end causing more violations than they fix.

Capacitance and Transition Violations

When a signal takes too long transiting from one logic level to another, a transition violation is reported. The violation is a function of the node resistance and capacitance [3]. The designer has two simple solutions to fix the transition violations. The first is to increase the drive capacity of the buffer to increase the voltage swing or decrease the capacitance and resistance by moving the source gate closer to sink gate. The second option is to increase the width of the route at the violation instance pin. This will decrease the resistance of the route and fix the transition violation.

The capacitance on a node is a combination of the fan-out of the output pin and the capacitance of the net. This check ensures that the device does not drive more capacitance than the device is characterized for. The violation can be removed by increasing the drive strength of the buffer or by buffering some of the fan-out paths to reduce the capacitance seen by the output pin.

Temperature, Voltage, and Process

The interplay of voltage, temperature, and process on a design only complicates matters by increasing the number of PrimeTime runs. These three things do complicate matters but not as much as it might seem at first. The problem can be reduced by understanding which condition causes the best and worst case conditions and testing the design accordingly. If we graph voltage, temperature, and process individually against performance, we will be able to see what causes the best and worst case design circumstances.

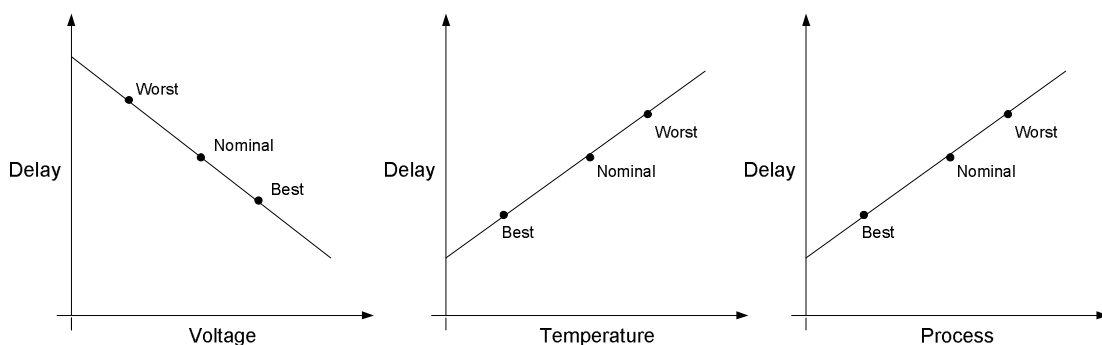


Figure 7 - Delay vs. Voltage, Temperature, and Process

From the graphs, it is visible that the best case or least delay is when the voltage is high, temperature is low, and process is fast. Therefore the worst case or most delay would be the contrary, low voltage, high temperature, and a slow process [2]. With the knowledge that increased delay causes more setup violations and decreased delay causes more hold violations, we can accurately fix setup and hold violations across voltage, temperature, and process. If we bound our design by the worst and best case conditions we can analyze the design only 2 times instead of 3^3 or 27 times.

- Use the worst case delay when testing for setup violations.
- Use the best case delay when testing for hold violations.

Timing Constraints

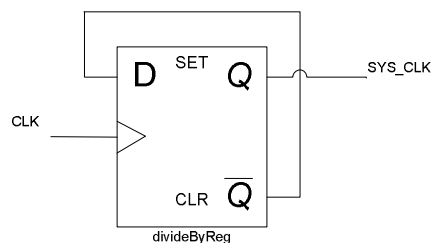
Timing constraints are how the designer tells PrimeTime about the timing behavior of the ASIC. There are three main constraints that must be defined to guarantee that the fabricated ASIC functions as the designer envisioned it to. The three minimum constraints the design must supply are defining the clock, input delay, and output delay. When the clocks are defined, all register to register paths are assumed to be constrained in one clock cycle. The input delay defines the time from the output of another ASIC through a primary input to a register. An output delay is the same as input delay but from the output of a register through a primary output to the input of another ASIC. These three constraints define the behavior of the ASIC to PrimeTime.

Clocks

To define a clock in PrimeTime, it is a function of the input port, clock frequency, and duty cycle. With a place and routed design, the clock port is no longer directly to every flip flop. A clock tree is grown, and PrimeTime must be told to propagate the clock signal through the clock tree. To define a clock with a frequency of 500 MHz or 2 ns period in PrimeTime the following command is used:

```
pt_shell> create_clock -period 2 [get_port clk]
```

There are cases when an internal clock is not defined from a primary input, such as a divide by counter for a pll for example [3]. To define an internal clock from a pin on an instance



b

Figure 8 - Divide By Register

```
pt_shell> create_generated_clock -name divide -source clk \  
-divide_by 2 [get_pins divideByReg/Q]
```

It is a good design practice to add extra margin into the design. The PrimeTime command, `set_clock_uncertainty`, adds the extra padding to both the setup and hold calculations to every path in the design. During synthesis `set_clock_uncertainty` is used to account for delay between the clock branches (skew). A designer can use this fact to over constrain the design during synthesis and relax the constraint during timing closure to ensure that the design will meet timing when physical design is complete.

```
pt_shell> set_clock_uncertainty .1 [all_clocks]
```

Primetime can treat the clock paths as ideal or analyze the clock path from the primary clock input to register clock pin. The `set_propagated_clock` command specifies that PrimeTime realized the latency for each clock path. This command should be used during post route analysis.

```
pt_shell> set_propagated_clock [all_clocks]
```

Input/Output Timing

PrimeTime assumes that the data signal arrives at the primary input at time 0 and does not constrain the output paths at all. The PrimeTime commands `set_input_delay` and `set_output_delay` are used to constrain the input and output ports respectively. Calculating the I/O delay is very simple remembering that all paths are register to register.

The `set_input_delay` specifies the amount of time it takes for the signal to propagate from the external logic to the primary input [3]. When this command is used it is necessary to specify a clock signal that the input signal is relative to. PrimeTime will calculate the amount of time it takes for the signal to propagate through the internal logic.

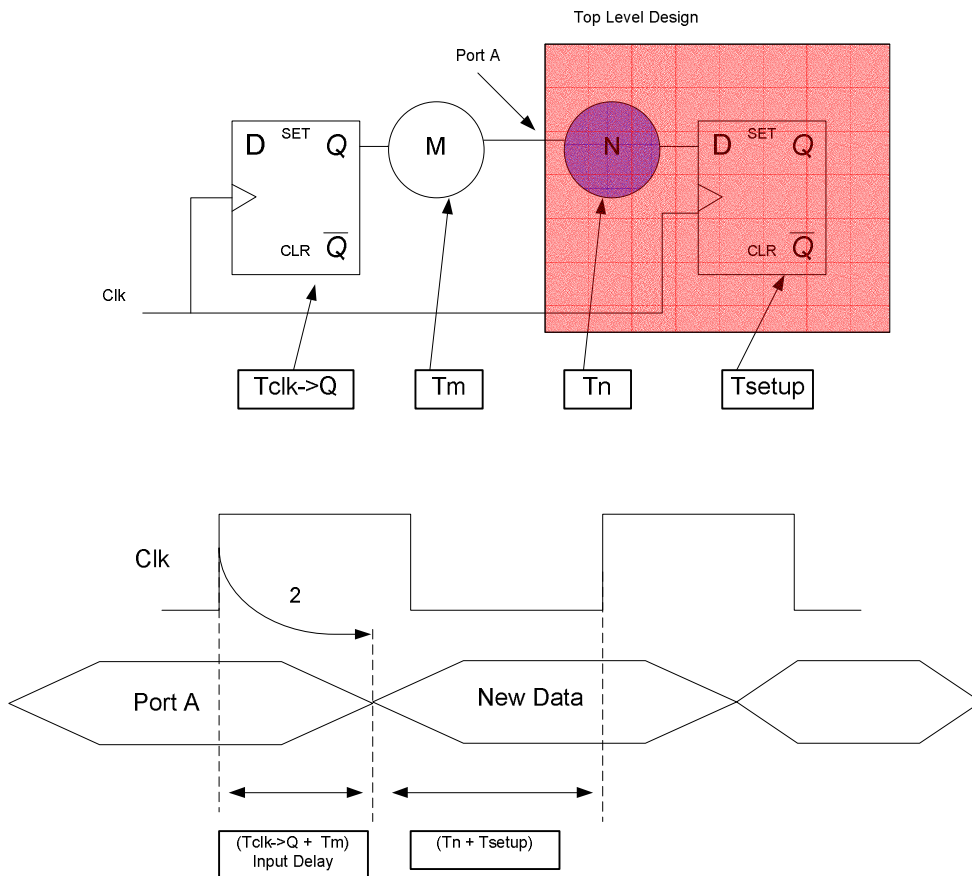


Figure 9 - Input Delay Example

```
pt_shell> create_clock -period 20 [get_port clk]
pt_shell> set_input_delay 2 -clock clk [get_port A]
```

The output path is constrained using the `set_output_delay` command. The time that the designer specifies the amount time the external logic needs. PrimeTime will calculate how long the signal takes to propagate through the internal logic [3]. As in the `set_input_delay` command the output signal must be referenced against a clock signal to calculate the setup/hold times.

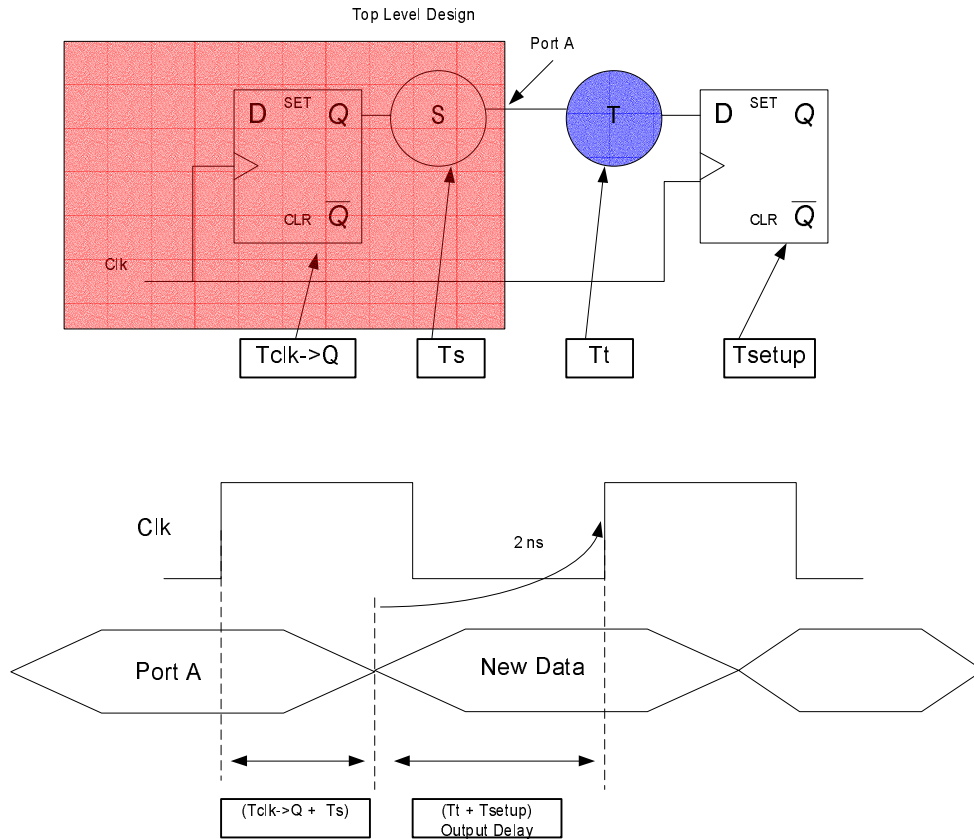


Figure 10 - Output Delay Example

```
pt_shell> create_clock -period 20 [get_port clk]
pt_shell> set_output_delay 2 -clock clk [get_port A]
```

False Paths

There are paths in a design where a designer would not want the timing arcs to be calculated [3]. These paths are either not relevant to functional operation of the circuit or paths which are impossible to exercise. Another case where a false path is needed is when data is launched off of a clock and captured on a different asynchronous clock domain.

In figure 11 shown below, it is not possible for a signal that starts Mux_1/S1 and propagates to Mux_2/D through the Mux_2/S1 input pin. The same can be said if a signal starts at Mux_1/S2 can not travel through the Mux_2/S2 pin. This is an example of a logical false path. The timing arcs must be removed from this example.

```
pt_shell> set_false_paths -through Mux_1/S1 \  
-through Mux_2/S1  
  
pt_shell> set_false_paths -through Mux_2/S2 \  
-through Mux_2/S2
```

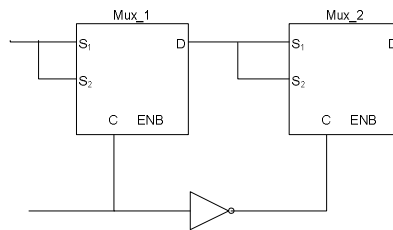


Figure 11 - Logical False Path Example

When data is launched from one clock domain and captured on a different asynchronous domain, metastability can acquire. The potential errors can be sent to the next set of registers causing errors to ripple through the design. There are several EDA tools that can identify and correct clock domain crossing by inserting synchronizing cells. To ensure that PrimeTime does not check for setup and hold in figure 12 the following PrimeTime commands should be used. It should be noted that a false path must be created from one clock domain to another and vice versa. If this is not done potential errors can occur.

```
pt_shell> set_false_path -from clk_a -to clk_b  
pt_shell> set_false_path -from clk_b -to clk_a
```

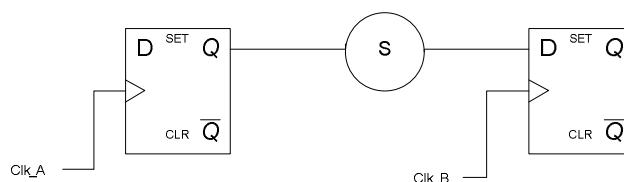


Figure 12 - Clock Domain Crossing False Path

Multi-Cycle Paths

A multi-cycle path is when a signal takes more than one clock cycle to propagate to the end point [3]. When you use this command you specify how many cycles until the signal is captured. There is one wrinkle that the design must be aware when using this command. The default hold check will be performed one cycle before the setup check. PrimeTime must be configured to check for hold violations $N_{\text{multi-cycles}}$ cycles before the capture edge.

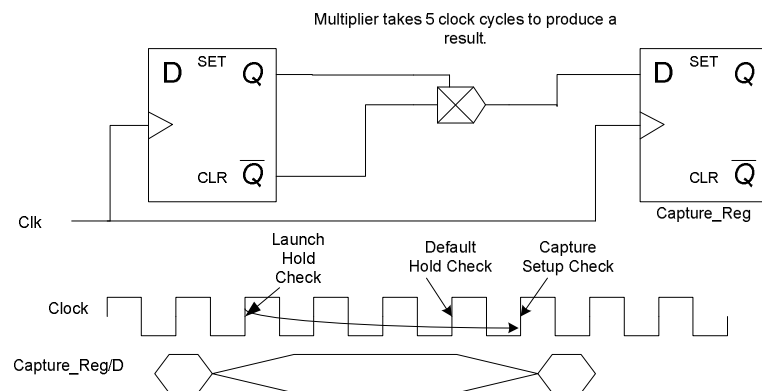


Figure 13 - Multi-Cycle Example

```
pt_shell> set_multicycle_path -setup 5 \  
                               -to [get_pins Capture_Reg/D]  
  
pt_shell> set_multicycle_path -hold 0 \  
                               -to [get_pins Capture_Reg/D]
```

Path Delay Calculation

Cell Delay

In the previous section, cell and net delay were assumed for easy calculations of total path delay. In this section, it will be explained how PrimeTime calculated the delay and what components are needed for the calculations. Before PrimeTime starts calculating path and clock delays, it first breaks each path into timing arcs [3]. Each timing arc contributes to either cell or net delay.

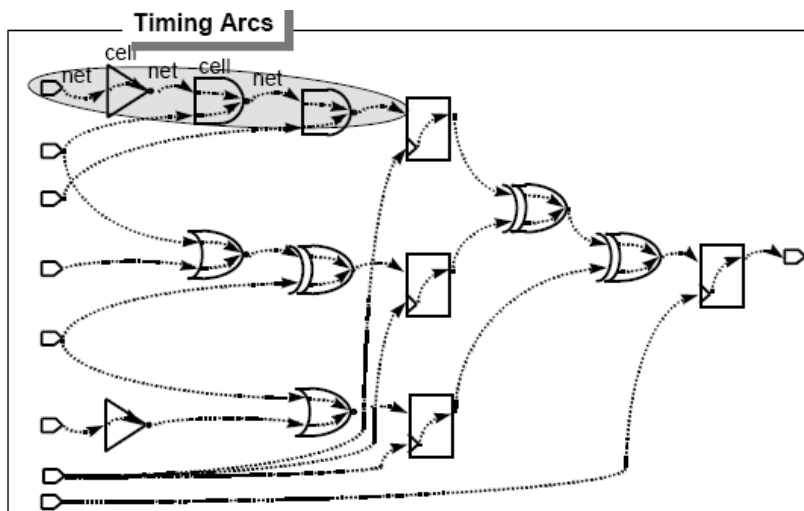


Figure 14 - Timing Arcs
Picture provided by Synopsys

As shown in figure 6, the total path delay is the addition of all the net and cell delay. When PrimeTime analyzes a path, it must keep track of edge sensitivity or unateness [3]. If PrimeTime used the only largest or smallest delay when calculating the path delay, the results would be overly pessimistic resulting in false setup and hold violations. This is why PrimeTime must know if the input and output of the cell is rising/falling for each path. Also, since PrimeTime does not use test vectors, each path is analyzed twice, once with a rising input and once with a falling input.

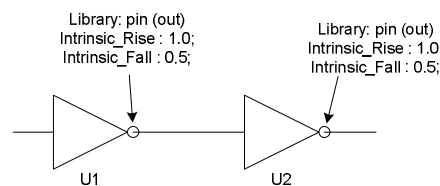


Figure 15 - Edge Sensitivity

Cell delay is stored in files called Synopsys database files or db files. Database files are compiled liberty files. Liberty files are an open standard file which contains 2

dimensional non-linear lookup tables for both cell delay and output transitions. The lookup tables contain two inputs, output load on the cell and the input transition to the cell. The output transition becomes the input transition to the next cell. PrimeTime will use a first order approximation if the input transition or output load fall within two of the table points [4]. Table sizes can vary from a 5x5 matrix to 7x7 matrix.

Cell Delay (ns)		Output Load (fF)			
		0	1	2	3
Input Transition (ns)	0	2	3	5	6
	10	3	3.5	6	7
	20	5	8	9.4	12

Cell Delay = 6ns

Output Transition (Vendor Units)		Output Load (fF)			
		0	1	2	3
Input Transition (ns)	0	1	19	18	23
	10	2	12	28	32
	20	4	15	29	38

Output Transition = 28 units

Figure 16 - Non Linear Lookup Table Example

Database files are read into PrimeTime by the link_path variable. Link_path is a space-delimited variable so multiple database files can be read in at a time.

```
pt_shell> link_path "stdCell_ss.db rams_ss.db"
```

For more information of generation of liberty files please refer to the scholarly paper “Standard Cell Based ASIC Library Development Flow” by Saadat Khan [4].

Net Delay

The net delay is calculated by PrimeTime by an internal delay calculator. A user can choose not to use the PrimeTime internal delay calculator and use a third party EDA tool [3]. Cadence’s SignalStorm will read in spef/spf, analyze the design, and produce SDF (standard delay format) files which can be used by PrimeTime. The following command is used to read the parasitic file into PrimeTime.

```
pt_shell> read_parasitics -format SPEF top_level.spef.gz
```

Generating PrimeTime Reports

Report_Timing

Reporting the information stored in the PrimeTime database allows to user to identify timing violations and make informed engineering changes to the design. PrimeTime provides extensive commands to report setup, hold, and design rules violations. To reduce the size and complexity of the PrimeTime reports, it is recommended to break the design into groups. There are 4 main groups that can be seen in figure 17 [3].

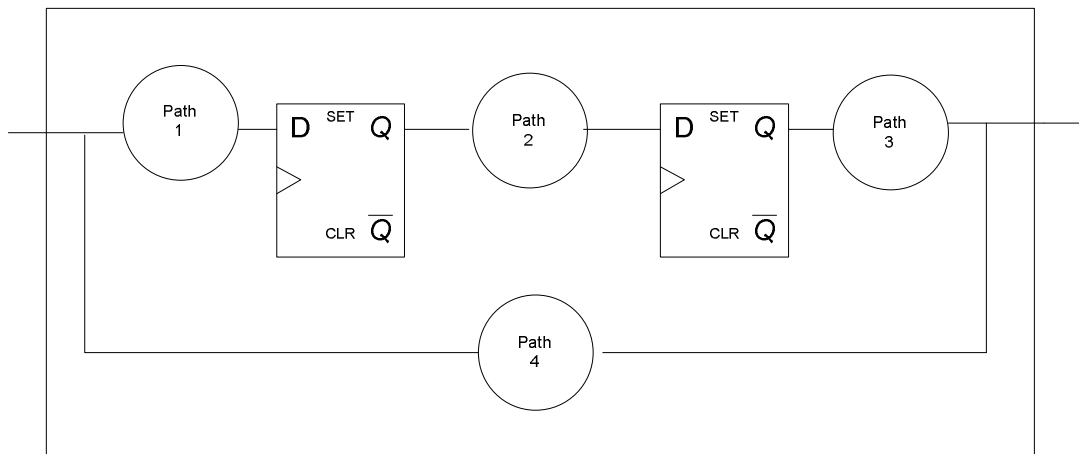


Figure 17 - Path Grouping
Path 1: Input to Registers
Path 2: Register to Register
Path 3: Register to Output
Path 4: Input to Output

Note that each path begins at a valid start point and end points. Valid start points are primary inputs port and clock pins of the register. Valid end points are primary output ports and all input ports on registers (this includes enable and scan in pins) except for clock pins. To reduce the amount of code a design must write, PrimeTime has built in collections (variables) which make grouping to and from the registers in the design very simple [3]. Using the all_register collection makes it straightforward to find and report input to registers, register to register, and register to out paths. Please

```
pt_shell> report_timing -from [all_registers -clock_pins] \  
                    -to [all_registers -data_pins]
```

There are many options the user can specify when calling the `report_timing` command. It is recommended that the user review the manual for this command because it is used with every design.

```
Pt_shell> man report_timing
```

Several of the more common command options will be presented in this paper.

-from and -to

When this command is used on the paths contained in this option will be reported. If this command is not used PrimeTime will default to the longest path (critical path) in the design.

-path full_path

This option reports not only the data path but the launching and capturing clock path. `Set_propagated_clocks` must be set for this option to properly report the clock paths.

-delay {max|min}

PrimeTime will calculate either setup or hold depending on how the input to this option is set. If the input of the option is set to "max" then PrimeTime will analyze the design and calculate the setup times for all the paths. If the option is set to min then each path is checked for hold violations.

-nets/-cap/-trans

To include information about the nets in the timing reports the following options can be used.

- nets: This command tells PrimeTime to include the nets in the reports.
- cap: The lump capacitance of every net will be reported with this command.
- trans: Reports the input driving transition on the pin in the path.

-max_paths, -nworst

These two options control the number of paths that are printed in the reports. Since there could be hundreds of thousands of paths in the given design, it is only necessary to see the paths which are failing or close to failing.

-max_paths; This variable states the total number of paths to be reported per group. The default is one.

-nworst: This variable states the number of end points to be reported.

For an example of the `report_timing` command please see generic PrimeTime script section of this paper.

Report_Constraints

This command generates a summary of all paths that are violation setup and hold times as well as any cells that violation a design rule such as fanout, capacitance, and transition. Viewing this one report will tell you if changes will need to be made to your design.

```
pt_shell> report_constraints -all_violators
```

Report_clock_timing

Clock skew can be a cause of multiple timing failures. This command will report the skew, the difference between the longest and shortest clock insertion time, and allow the design to evaluate whether or not the clock tree must be resynthesized. This is a powerful command can save the designer from numerous timing closure spins.

```
Pt_shell> report_clock_timing -type skew -verbose
```

Generic PrimeTime Script

This section is intended to give a student or beginning designer a complete but adaptive script to begin working with PrimeTime. To run, each command can be cut and pasted in the PrimeTime command line individually or executed with one command from the Linux/Unix prompt.

```
[Linux] user@gmu>> pt_shell -f pt_script.tcl |& tee pt.log
```

A total of three scripts must be created, one for each timing corner.

```
# -----
# Library Declarations.
# -----
set search_path ". /proj/timing/etc"
set link_path "*"
lappend link_path "stdCell_tt.db"

# -----
# Read in Design
# -----

# Read in netlist
read_file -f verilog top_level.v

# Define top level in the hierarchy
current_design "top_level"

# Combine verilog and db files and identify any errors.
link_design

# Read in SPEF file
read_parasitics -quiet -format SPEF top_level.spef.gz

# -----
# Apply Constraints
# -----

# Read in timing constraints
read_sdc -echo top_level.sdc

# Propagate clocks and add uncertainty to setup/hold calculations
set_propagated_clock [all_clocks]
set_clock_uncertainty 0.2 [all_clocks]
```

```

# -----
# Time
# -----

set_operating_conditions -min WORST -max WORST

# Register to Register
report_timing -from [all_registers -clock_pins] \
              -to [all_registers -data_pins] -delay_type max \
              -path_type full_clock -nosplit \
              -max_paths 1 -nworst 1 \
              -trans -cap -net > tc_reg2reg_setup.rpt
report_timing -from [all_registers -clock_pins] \
              -to [all_registers -data_pins] -delay_type min \
              -path_type full_clock -nosplit \
              -max_paths 1 -nworst 1 \
              -trans -cap -net > tc_reg2reg_hold.rpt

# Register to Out
report_timing -from [all_registers -clock_pins] \
              -to [all_outputs] -delay_type max \
              -path_type full_clock -nosplit \
              -max_paths 1 -nworst 1 \
              -trans -cap -net > tc_reg2out_setup.rpt

report_timing -from [all_registers -clock_pins] \
              -to [all_outputs] -delay_type min \
              -path_type full_clock -nosplit \
              -max_paths 1 -nworst 1 \
              -trans -cap -net > tc_reg2out_hold.rpt

# In to Register
report_timing -from [all_inputs]
              -to [all_registers -data_pins] \
              -delay_type max \
              -path_type full_clock -nosplit \
              -max_paths 1 -nworst 1 -trans \
              -cap -net > tc_in2reg_setup.rpt

report_timing -from [all_inputs] \
              -to [all_registers -data_pins] \
              -delay_type min -path_type full_clock \
              -nosplit -max_paths 1 -nworst 1 \
              -trans -cap -net > tc_in2reg_hold.rpt

# All Violators - Find Cap/Tran Violations
# Summary of Setup/Hold Violations
report_constraints -all_violators > tc_all_viol.rpt

# Clock Skew
report_clock_timing -type skew -verbose > tc_clockSkew.rpt

exit

```

Conclusion

Static timing analysis is the most accurate approach to analyze the timing performance of a design before performing dynamic timing analysis through extensive testbenches. Synopsys PrimeTime software is user-interactive and allows users to apply timing constraints on their design. Most of the required timing characteristics of a design are provided in the database. However, the user needs to combine his/her timing analysis proficiency with the knowledge of the tool to extract different timing information, such as path delays, clock performance, and setup and hold time. The tcl language environment provides the users access the required information faster and more easily. A user needs to refer to the PrimeTime software release notes for the latest updates on the tool. In sum, this document, combined with the user manual, provides reference material for PrimeTime static timing analysis.

References

[1] Brent B. Welch, "Practical Programming in TCL and TK",
3rd Edition, Prentice Hall, 1999

[2] **Static timing analysis with rigorous exploitation of setup time margins**
Wortmann, A.; Simon, S.; Bergholz, W.; Muller, M.; Mader, D.;
Circuits and Systems, 2003. MWSCAS '03. Proceedings of the 46th IEEE International Midwest
Symposium on
Volume 3, 27-30 Dec. 2003 Page(s):1396 - 1399 Vol. 3

[3] Synopsys PrimeTime users guide

[4] GMU Scholarly Paper Presentation "Standard Cell Based ASIC Library Development Flow" Saadat A.
Khan