# Using PrimeTime in LSI Logic's FlexStream™ Design Flow

Robert Landy
Yoon Kim

LSI Logic
Milpitas, CA

landy@lsil.com
ykim@lsil.com

## Abstract

For large or complex System-on-a-Chip designs, which often consist of over one million gates, full chip gate-level dynamic simulation is becoming increasingly time consuming and verification coverage is steadily diminishing.

LSI uses static timing analysis to ensure that a design will meet timing.

Synopsys PrimeTime has been qualified for static timing analysis signoff within the LSI Logic Flexstream flow.

This presentation will focus on how LSI has fully integrated Primetime:

1. LSI Flexstream flow (lsidelay, lsimemory).
2. LSI design center organization has been fully trained with Primetime
3. LSI has numerous success stories using the above-mentioned design flow.

## 1.0 Introduction

Designers in the past have spent a huge portion of their overall design time performing functional and timing verification of the design. Using dynamic verification method, a designer would have to create an extremely large number of timing vectors to verify the timing paths in the design. As the size and complexity of designs increase however it's becoming almost impossible for a designer to create such a vector set to achieve complete coverage.

Static verification offers numerous advantages over dynamic verification in the runtime, capacity and exhaustive coverage and hence is much better suited to handle today's high-capacity and high-complexity designs.

LSI has qualified PrimeTime as a signoff static timing analysis tool to allow customers to analyze, debug, and validate the timing performance of a design.

## 2.0 Static Timing Analysis Using Primetime in FlexStream design flow

This section describes how LSI recommends its customers to use PrimeTime and outlines the step-by-step procedure of performing static timing analysis on a design using PrimeTime in the LSI FlexStream design environment.

Performing static timing analysis using PrimeTime can be broken into 3 sessions:

1. Generate input files needed by PrimeTime in FlexStream environment.
2. Create setup files for PrimeTime.
3. Perform analysis and generate report.

### 2.1 Input files

The following input files which are needed by PrimeTime need to be generated in LSI's FlexStream design environment.

**Netlist file**:

The netlist for the design can be in Verilog, VHDL, or .db format. LSI supplies the lsinetlist program to translate netlists into Verilog/VHDL. Netlist translations must be done using lsinetlist due to name mapping and back annotation concerns.

**Technology library files**:

PrimeTime uses the same technology library files (`.synlib`) that are used for synthesis by Synopsys Design Compiler. The user needs to specify the search path and link path for the libraries so that PrimeTime knows where to look for them.

**Stamp memory models**:

Stamp memory models, if the design requires them, are generated by running `lsimemory` with the -ptime option.

```
lsimemory -mcell 4p_lp_2r2w_11p422la -word 128 -bits 56 -bpw word -
ptime -freq 100 -cellname rr128x56
```

This generates the .mod and .data files that are needed to compile the Stamp memory models in PrimeTime.

The Stamp models may also contain mode definitions (for modes such as read, write, read/write, etc.) to be used in mode analysis of the design.

**SDF files**:

The Standard Delay Format (SDF) file is generated by running `lsidelay` with the `-ptime_verilog` or `-ptime_vhdl` option, depending on whether a Verilog or VHDL netlist is used. LSI requires that the design be tested in best-case, worst-case, and test conditions, so `lsidelay` generates three SDF files, one for each set of conditions.

```
lsidelay -top_module dcp_bs_c_dp -language verilog -nets dcp_bs_c_dp.v
-technology lcbg10p -delaycase wccom tst bccom -ptime_verilog
```

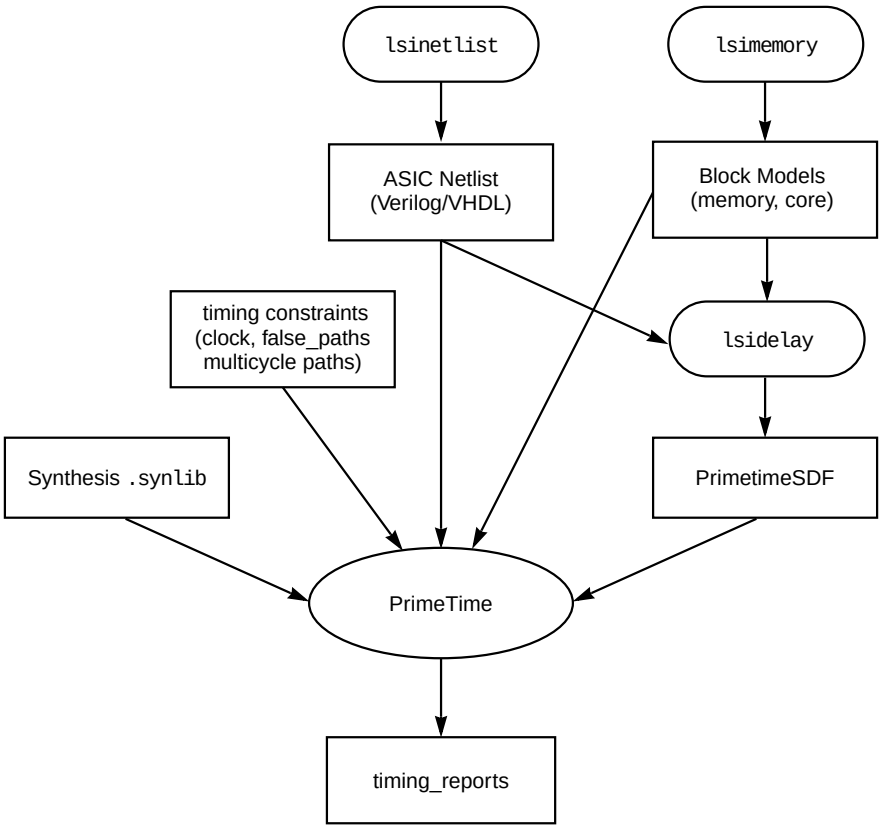


Figure 1. FlexStream Primetime flow.

## 2.2 Design environment and timing constraints

This section describes step-by-step procedures needed to setup design environment and various timing constraints before the actual analysis is performed and report generated.

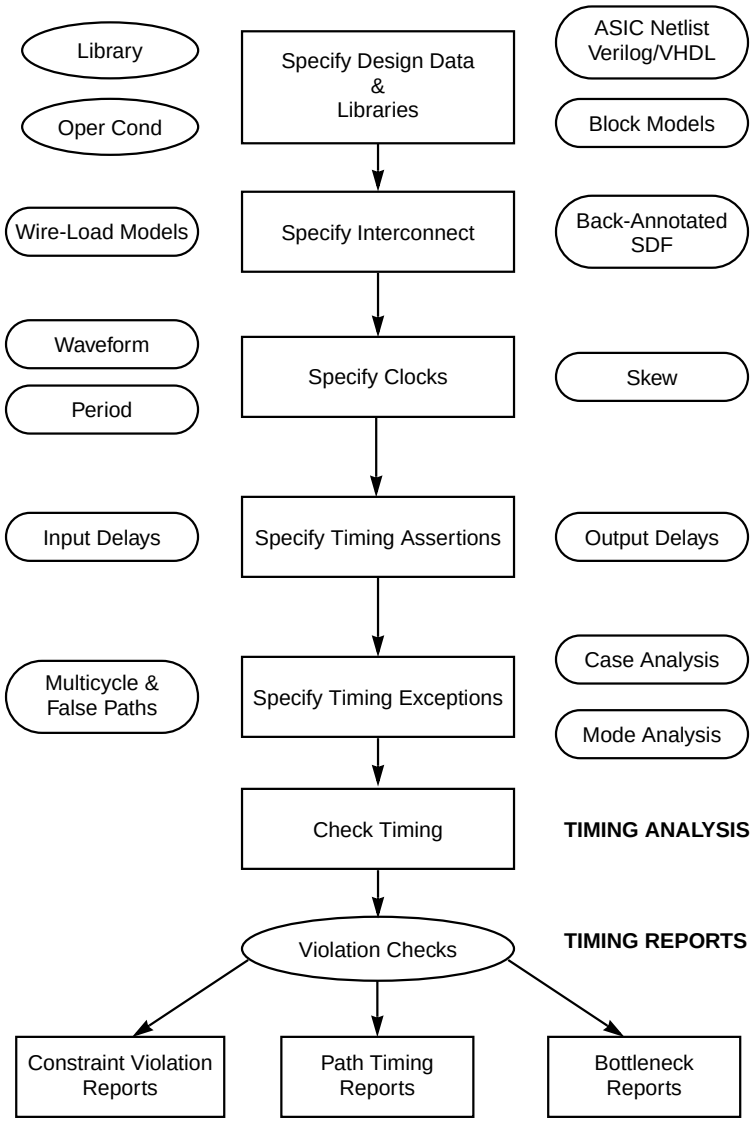


Figure 2. Static timing analysis Flow Using Primetime

Step 1. Compile the Stamp memory models, and then include them in the search and link paths.

```
compile_stamp_model -remove_internal_arcs -library_cell -model_file
rr128x56.mod -data rr128x56.data -output rr128x56
```

**Note:** If you do not use the `-remove_internal_arcs` and `-library_cell` options, then PrimeTime creates a `.db` file that is not consistent with the SDF and causes problems in back-annotation.

Step 2. Specify the libraries and models to be used, by setting the search path and the link path. The search path must include the location of the FlexStream synthesis libraries (the same libraries used by Synopsys Design Compiler for synthesis) as well as the location of the compiled Stamp memory models. The link path should include internal caches ("*"), as well as the Stamp memory model (if any) and the technology synthesis library (both internal and I/O).

```
set search_path ". ~fs10/lib3p/synopsys/lcbg10p /lsi/home/jswami/
PrimeTime/TESTCASES/NDS/PrimeTime/STAMP_LIB"

set link_path { * lcbg10p_wccomv.db lcbg10piov.db rr256x32syn_lib.db
rr256x8_lib.db }
```

Step 3. Read in the ASIC design either as a Verilog or a VHDL file.

```
read_verilog {./dcp_bs_c_dp.v }
```

Step 4. Link the design.

```
link_design {-verbose}
```

Make sure that the design has been linked properly by looking at the output of the `link_design` command. If there are any unresolved cells, check and confirm the reason before proceeding with the analysis.

Step 5. Set the current design to be the design you want to analyze (if it is a hierarchical design).

```
current_design dcp_bs_c_dp
```

Step 6. Read in the PrimeTime SDF file with the `-analysis_type` switch set to `on_chip_variation`:

```
read_sdf -analysis_type on_chip_variation dcp_bs_c_dpbccom.ptimeSDF
```

**Note:** The `on_chip_variation` command is used typically when we need to read in different delay values due to different process, voltage, and temperature (PVT) conditions in a single chip. However, we do not need this capability, because, in the LSI design methodology, we use separate SDF files for different PVT conditions (for example, for bccom, wccom and tst conditions, we would have three separate SDF files). Rather, we use the `on_chip_variation` command to read in delay values that differ between the same two points depending on the condition of other inputs, such as asynchronous set paths, test enable constraints, or custom IOs with multiple modes. In this case, the SDF file contains the delay values in the format *`min:typ:max`*, where the *`min`* place-holder is used for the minimum delay between the two points, the *`max`* place-holder for the maximum delay and the *`typ`* place holder is the average of the other two delays.

For example if you have

(SETUP (posedge D) (posedge CP) (1:2:3) (1:2:3))

the CP→Q delay has three different values, in the form *min:typ:max*. PrimeTime uses the `max` delay value for setup checks and the `min` delay value for hold checks.

**Note:** On-chip variation is *not* set as an operating condition using the set_operating_conditions command or from the attributes->operating_conditions on the GUI but rather with the `read_sdf` command.

Step 7. After the SDF file has been read in, execute the following command to make sure there are no problems with back-annotation:

```
report_annotated_delay -list_not_annotated
```

This command reports all net delays and cell delays and whether or not they have been back-annotated. Make sure all the cell arcs and internal net arcs have been back-annotated. Some net arcs to primary outputs or from primary inputs might *not* have been back-annotated, because the SDF does not include these arcs.

The following command reports all the timing checks and whether or not they have been back-annotated from the SDF.

```
report_annotated_check -list_not_annotated
```

Once again make sure that all the checks in the SDF have been back-annotated.

Step 8. Specify clock information in a separate file so that it can be modified and re-read easily without having to modify the entire setup. The clock is generated by the `create_clock` command, with the name of the clock, the period, the waveform, and (optionally) the name label, specified as parameters:

```
create_clock clk_g -period 8 -waveform {0 4}
```

In PrimeTime (unlike MOTIVE), the designer may specify a divided clock. For example, if the master clock runs at 100 MHz, the following commands generate a 50-MHz divided clock:

```
create_clock clk_g -period 10 -waveform {0 5}

create_generated_clock -divide_by 2 -source clk_g [get_pins xtop/YTOP/
xa5/xa0/cntb_reg2\[2\]/Q] -name clk_g_div
```

Step 9. If you have a post-layout SDF then always specify the `set_propagated_clock` command so that the actual propagation delays resulting from cell and net delays in the clock network are taken into account.

```
set_propagated_clock clk_g
```

If you do not have post-layout SDF, you can set the clock latency using the `set_clock_uncertainty` command.

```
set_clock_uncertainty -setup 1.2 -hold 0.45 {get_clocks CLK1]
```

The set_propagated_clock command can be used for all the clocks in the design both internal and external, and it also takes into account the delay between the source and destination clocks.

```
set_propagated_clock [all_clocks]
```

Step 10. Specify all the input and output constraints for the design, using the set_input_delay and set_output_delay commands. These commands correspond to the TIN and TSH statements in the MOTIVE .ref file.

```
set_input_delay   2.4 -min -rise -add_delay -clock clk_g [all_inputs]
set_output_delay 0.0 -max -rise -add_delay -clock clk_g [all_outputs]
```

If the -min -rise options are omitted, then the delay is applied to all the edges. If a given input or output pin is dependent on more than one clock, then the input/output delays for each reference clock must be set.

Step 11. If the design has any multicycle paths, specify them using the set_multicycle_path command, preferably in a separate file (mcp.pt).

```
set_multicycle_path 2 -setup -hold -from ff1/CP -to ff2/D
```

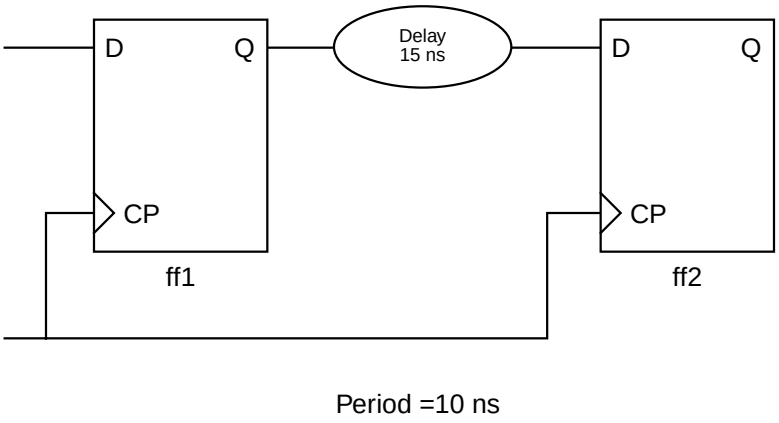


Figure 3. Example of multicycle path schematic

Step 12. If the design has false paths, specify them using the set_false_path command, preferably in a separate file (false_path.pt).

```
set_false_path -from ff1/CP -to ff2/D -setup -hold
set_false_path -from [get_clock clk1] -to [get_clock clk2] -setup -
hold
```

Step 13. If you have disabled any timing paths using a “constant declaration,” then use the `set_case_analysis` command, preferably in a separate file (const.pt). This is similar to the `CONST` statement in MOTIVE.

```
set_case_analysis 0 TS_ATPG
```

**Note:** In PrimeTime, the case can be set only to 0 or 1. There is no “X” option as in MOTIVE.

**Example with Combinatorial Element:** For a multiplexer with two inputs and one select (as shown in Figure 5), the following command is used:

```
set_case_analysis 0 Sel
```

This command instructs PrimeTime to select only the timing arc A → Z for analysis and to disregard the arc B → Z.

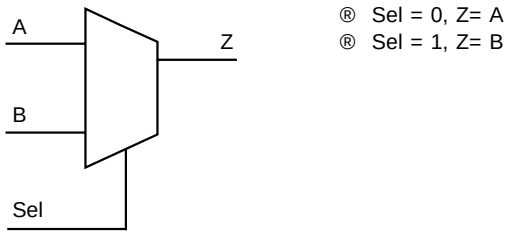


Figure 4. Example of Using set_case_analysis with a Combinatorial Element

**Example with Sequential Element:** The following command is commonly used for disabling the scan chain (see Figure 6). To disable the scan chain, propagate the test-mode constant value to the scan pin of the flip-flop, as follows:

```
set_case_analysis 0 TI
```

This command causes PrimeTime to forego setup and hold analysis on the CP → TI arc.

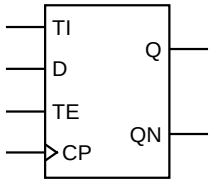


Figure 5. Example of Using set_case_analysis with a Sequential Element

Step 14. If the design has any memories or cores, then you might need to specify modes, such as read, write, read/write, and so on. This is a two-step procedure:

**a)** First, define all the modes that are present in the design (memory or core) either in the `pt_shell` or in the Stamp memory model itself.

```
define_design_mode {read, write, fetch}
```

**b)** Set the particular mode for which you want to analyze the design:

```
set_mode read ROM
set_mode write RAM
set_mode fetch CPU
```

In order to see the different kinds of modes in the design use the `report_mode` command.

```
report_mode
```

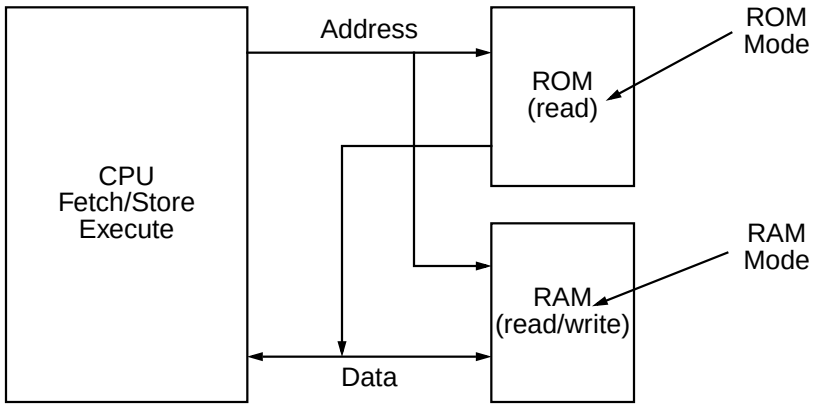


Figure 6. Mode analysis schematic

The reason for using mode analysis is to enable only a limited set of the timing arcs in the design. For example, if the memory is in read mode, and you run the `report_timing` command, PrimeTime performs the analysis only on the timing arcs that are enabled during read mode as defined in the Stamp memory model file and ignores all the other timing arcs, thereby accurately reflecting the timing of the design in the READ mode.

Step 15. Run the `check_timing` command on the design to make sure all the variables have been set as desired. If any violations are reported, you need to fix them before proceeding with the analysis.

```
check_timing -verbose
```

The following violations might be reported:

- Clock pins driven by multiple clocks: typically this happens if there is a mux clock and selection pin has not been constrained.
- Unclocked registers: all registers clock pins must be driven by a clock, ensuring the whole analysis will be conducted on all registers.
- Unconstrained endpoints: unconstrained input, output and bidirectional ports.

## 2.3 Analysis and reports

You can generate many different kinds of reports based on the needs of the design and the specific type of analysis being performed. After all the reports have been generated, make sure that the following have been completely eliminated:

- Flip-flop timing violations (setup, hold and pulsewidth)
- Output setup/hold violations (setup/hold violations on the imaginary flip-flop at the output of a chip. The setup/hold check will be made at the input of this imaginary flip-flop. This process is similar to the TSH statement in MOTIVE.)

The following reports are mandatory for the LSI Logic static timing analysis flow:

### `Report_constraint -all_violators -verbose`

This command reports the worst violator (setup, hold, or pulsewidth) for each flip-flop or latch in the design.

In order to generate an exhaustive report for all paths to all sequential element violations, use the following two commands:

```
report_timing -slack_lesser_than 0.0 -to [all_registers -data_pins] -
nworst 100000 -path summary
```

```
report_timing -slack_lesser_than 0.0 -to [all_registers -async_pins] -
nworst 100000 -path summary
```

For example, if a flip-flop has six data paths to its data pin and each path causes a violation, then the first command reports all six data paths as violating. Thus, this command allows the user to create a complete list of violators for all paths and all flip-flops.

Make sure that no setup, hold or pulsewidth violations are reported. To eliminate these violations, make modifications to the design or the timing setup.

Determine whether any `clock_gating` check violations are reported, Primetime will report timing violations on the and gate if the gated clock (CK1) is misshaped. For example, in the diagrams below, a clock gating hold violation will cause either a glitch at the trailing edge of the clock pulse or a clipped clock pulse (as shown in Figure 8).

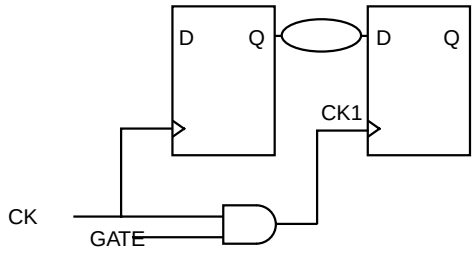


Figure 7. Example Circuit for clock_gating Check Violations

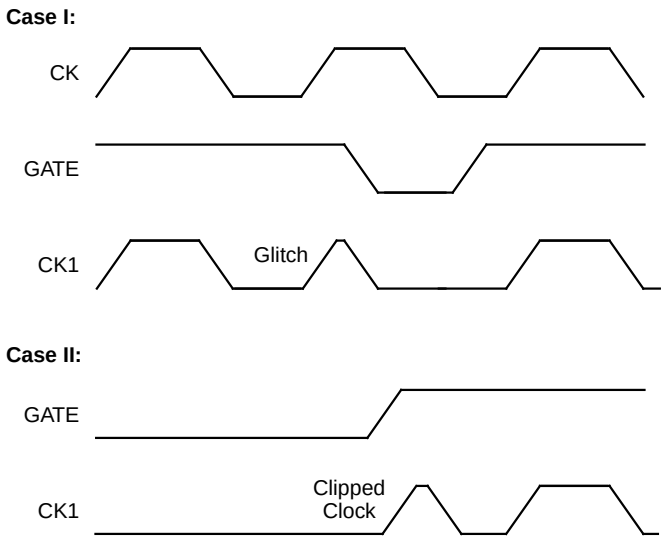


Figure 8. Modified Waveforms

### `report_disable_timing -nosplit`

This command reports all the disabled pins and timing arcs and details why they have been disabled. The command is useful for determining whether the arcs have been disabled for a valid reason. If necessary, modify the design so that all mistakenly disabled arcs are re-enabled and included in the analysis.

### `report_exceptions`

This command displays the timing exceptions within the design that PrimeTime ignores or considers valid. Make sure that these exceptions are acceptable. If the exceptions are not acceptable, make changes in the setup so that they are not excluded from the analysis. Some timing arcs might be overlooked because the user had set a specific case on some pins, or there might have been timing loops which have been disabled. Make sure that these exceptions are acceptable and were set deliberately.

### `report_case_analysis`

This command lists out all the ports and pins that have been set to a constant value in order to disable some timing arcs. Make sure this list is consistent with your settings.

### `report_bottleneck`

This command reports the cell(s) in the design that have multiple timing violations. You can use this information to modify those particular cells to improve performance. This report also contains a list of cells that have the greatest number of violations, and it indicates one of the following bottleneck costs associated with each of these cells:

- path count

- path cost
- endpoint slack count

The user can chose which cost-histogram to view: path count, path cost, or endpoint slack count. Get a schematic of the offending cell with its fan-in and fan-out. If excessive net-capacitance is found to be the cause of these problems, manually up-size the cells and increase their drive strength.

## 3.0 PrimeTime at LSI

### Qualification

PrimeTime has been a qualified signoff static timing analysis tool at LSI since July of 1998. LSI's PrimeTime qualification team worked very closely with Synopsys to improve the quality of the tool by providing feedback from users' point of view and Synopsys has been very responsive in implementing LSI's requests. LSI is currently working on qualifying 1998.08-PT2 version of PrimeTime.

### Deployment

LSI is aggressively deploying PrimeTime to the field as well as internal product groups by offering frequent training classes (through Synopsys), providing various application notes on how to do static timing analysis using PrimeTime and by providing scripts to convert MOTIVE setup files to PrimeTime files.

PrimeTime has been used as a static timing analysis signoff tool in various applications such as Computer, Communications and Consumer products.

### PrimeTime advantages

PrimeTime's ease of use and relatively short learning curve allow designers to generate valid static timing analysis runs in hours as opposed to days or weeks in previous static timing analysis tools. In addition, customers that are familiar with the synthesis environment will find using PrimeTime a natural extension of design_analyzer for running static timing analysis.

## 4.0 Summary

In order to provide full timing verification for today's high-capacity, high-complexity designs, LSI has qualified Synopsys' PrimeTime as a signoff static timing analysis tool.

This paper outlines procedures LSI recommends to customers on performing static timing analysis using PrimeTime: generating input files in FlexStream environment, creating setup files in PrimeTime, and generating and analyzing reports.

Finally, qualification, deployment and usage of PrimeTime at LSI is briefly discussed.