White Paper

TimeQuest Timing Analyzer: Native SDC Support for Timing Analysis of FPGA-Based Designs

Introduction

The field programmable gate array (FPGA) market has changed significantly in the past few years. Advances in silicon process technologies continue to augment both FPGA density and speed. As a result, an increased number of high-performance commercial applications, typically targeted at ASICs or ASSPs, now can be successfully, efficiently, and economically implemented in FPGAs. To achieve target performance, FPGA design engineers have adopted new clocking requirements (via clock multiplexing design schemes) and implemented design interfaces (such as source-synchronous clocks) that are difficult to analyze using traditional FPGA timing analysis.

This paper describes the new requirements that FPGA design software must satisfy to quickly and efficiently perform timing analysis and achieve timing closure. (For the purpose of this paper, timing analysis and static timing analysis are the same tasks performed with the same static tools. For brevity's sake, we will refer simply to timing analysis.) Productivity requirements include the adoption of an industry-standard timing analysis methodology, and this white paper will illustrate a few practical applications of the industry-standard Synopsys Design Constraints (SDC) format. Additionally, productivity is enhanced by integrating the timing analysis engine with the place-and-route engine. Altera has used these requirements to develop and implement Quartus® II TimeQuest timing analyzer, a new, ASIC-strength static timing analysis tool with native SDC support. This paper will conclude with a short description of the timing analyzer's main features.

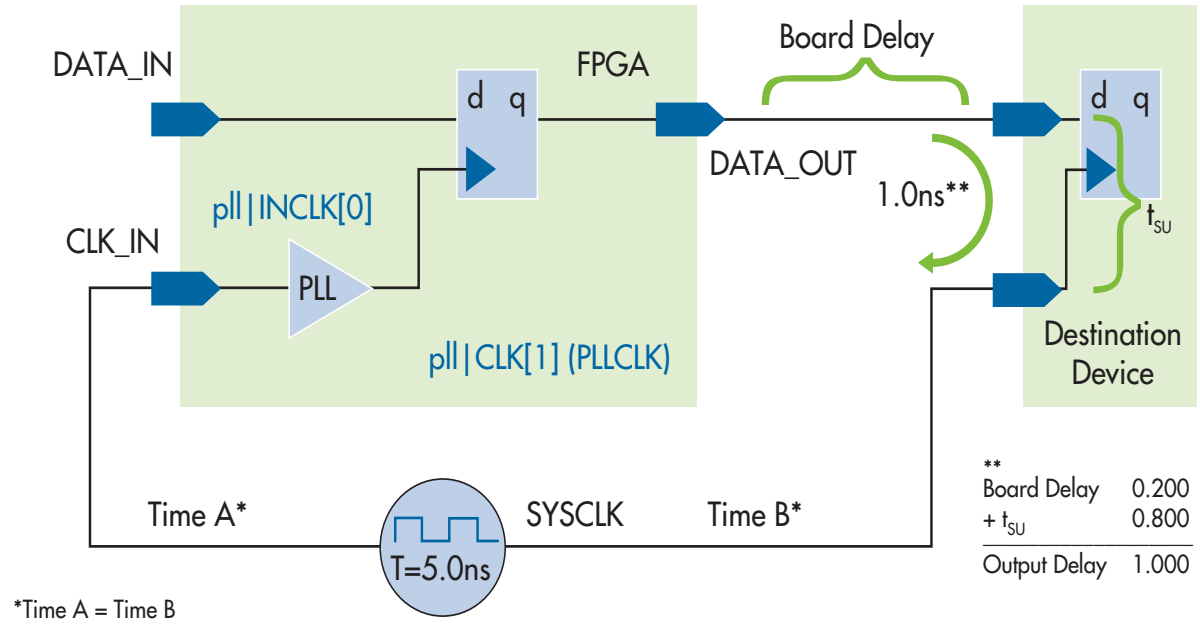
Native SDC Support + Timing Engine Integration = More Efficient Timing Analysis

Static timing analysis is a method of analyzing, debugging, and validating the timing performance of a design. It is used in conjunction with functional verification (typically simulation or prototyping) to verify the correct design operations. After synthesis or placement-and-routing (which can include physical synthesis) is performed, engineers run a timing analysis to check for timing violations that may occur.

The SDC format is an industry standard used to describe the timing constraints in which the design is expected to operate. As more commercial applications move from ASICs to FPGAs, an increasing number of ASIC design engineers are developing new chips or derivatives using FPGA-based design software and technologies. These engineers are already very familiar with the SDC-based timing analysis methodology. Often these constraints are already available for many of the blocks being re-implemented into designs targeting FPGAs, so re-using the same timing constraints provides a clear productivity advantage. Further time can be saved by avoiding any errors associated with the translation process, whether manual or automated, from SDC to the constraints format supported by the targeted FPGA-based design software timing analysis tool.

Another key aspect of SDC-based timing analysis is that the constraints are described using the tool command language (Tcl) and follow Tcl syntax rules. Therefore, SDC-based timing analysis lends itself to scripting, and empowers designers to automate timing-related tasks. This is one of the reasons why SDC-based timing analysis is the preferred methodology when designing structured ASIC devices, such as the Altera® HardCopy® II device family. In addition, complex timing constraints for designs containing source-synchronous interfaces (such as DDR and DDR2) and clock-multiplexing design structures can be conveniently expressed using the SDC format.

These combined benefits enable Quartus II design software users to increase their productivity and are providing a comparable productivity advantage to that which occurred when they transitioned from schematic entry to hardware description languages such as Verilog and VHDL. One of the reasons for the SDC format's popularity is that it is intuitive and easy to learn. Figure 1 shows a basic example of these constraints.




```
create_clock -name SYSCLK -period 5.000 [get_ports CLK_IN]
create_generated_clock -name PLLCLK \
    -source [get_ports CLK_IN] \
    [get_pins pll|CLK[1]]
# The next line assumes zero skew (e.g., Time_A == Time_B)
set_output_delay -clock SYSCLK 1.000 [get_ports DATA_OUT]
```

Figure 1. SDC Timing Analysis Application: Synchronous Interface

A phase-locked loop (PLL) is used to provide the clock signal to the output register of this FPGA design. The input of the PLL is the clock signal `SYSCLK`. Board layout considerations show that it takes 0.2 ns for the output of the FPGA to arrive at the external device. As shown in Figure 1, it takes 1 ns for the output of the FPGA to arrive at the external device relative to the clock port of the external device (this includes the board delay and the µtsu). If this "data arrival time" is met, the correct timing behavior will be guaranteed.

These timing constraints can be expressed by first writing the clock definition itself. This can be performed with the SDC command `create_clock`:

```
create_clock -name SYSCLK -period 5.000 [get_ports CLK_IN]
```

The designer will then have to specify that the output of the PLL is also a clock generated from the clock at the input port `CLK_IN`.

```
create_generated_clock -name PLLCLK \
    -source [get_ports CLK_IN] \
    [get_pins pll|CLK[1]]
```

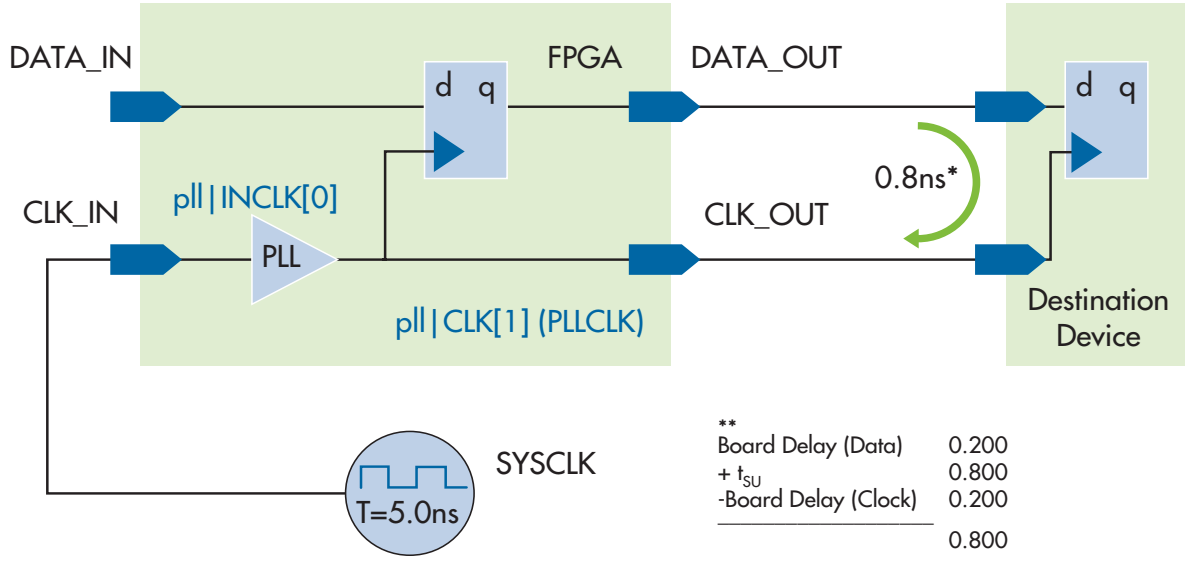
The delay between the output port `DATA_OUT` and the external device, relative to `SYSCLK`, is expressed using the set_output_delay command:

```
set_output_delay -clock SYSCLK 1.000 [get_ports DATA_OUT]
```

This example assumes that there is zero clock skew between the two devices. In Figure 1, the clock skew is the difference between "time A", the time it takes to reach the `CLK_IN` port from the external clock source, and "time B", the time it takes to reach the external device from the same external clock source. For designs operating at low frequencies, a small amount of skew may not matter. However, skew is one of the major limiting factors hindering synchronous designs from operating at high frequencies. Skew is also important for hold checks, regardless of the frequency. Ensuring that "time A" and "time B" are the same, thereby eliminating skew between data and clock signals, can be very difficult and expensive to achieve in practice at high clock frequencies.

To overcome the frequency limitation of the synchronous interface, designers have started adopting "source-synchronous" interface architectures. These are interfaces in which both the data and clock signals are sourced by the host device. Since both clock and data signals are sent together, they are subject to similar physical effects on the board. This virtually eliminates issues associated with skew and results in higher interface speeds.

Figure 2 shows an extension of the PLL example shown earlier. In this case, the output of the PLL (`PLLCLK`) is also used to source the clock for the external device. The output delay requirement in this case is 2.3 ns.




```
create_clock -name SYSCLK -period 5.000 [get_ports CLK_IN]
create_generated_clock -name PLLCLK \
    -source [get_ports CLK_IN] \
    [get_pins pll|CLK[1]]
set_output_delay -clock PLLCLK \
    -reference_pin [get_ports CLK_OUT]
    0.800 \
    [get_ports DATA_OUT]
```

Figure 2. SDC Timing Analysis Application: Source Synchronous Interface

As we did earlier, we define the clock with the SDC command `create_clock`, and we specify that the output of the PLL is a generated clock.

```
create_generated_clock -name PLLCLK \
    -source [get_ports CLK_IN] \
    [get_pins pll|CLK[1]]
```

With SDC, it is possible to specify the output delay of a signal using another pin or port of the device as a reference. In this case, the clock path delays from SYSCLK to CLK_OUT are accounted for automatically by the timing analyzer.

```
set_output_delay -clock PLLCLK \
    -reference_pin [get_ports CLK_OUT]
    0.800 \
    [get_ports DATA_OUT]
```

This example demonstrates the importance that native SDC support has for any ASIC-strength FPGA timing analyzer. The support for SDC provides all the sophisticated constraints controls that engineers require when performing timing analysis of high-performance FGPA-based design. Table 1 shows the SDC constructs supported in the TimeQuest timing analyzer in Quartus II software version 6.0.

Table 1. Quartus II Native SDC Support

Type	SDC Commands
Collections	all_clocks
	all_inputs
	all_keepers *(1)*
	all_outputs
	all_registers
	get_cells
	get_clocks
	get_keepers *(1)*
	get_nets
	get_nodes *(1)*
	get_pins
	get_ports
	get_registers *(1)*
Clocks	create_clock
	create_generated_clock
	derive_clocks
	derive_pll_clocks *(1)*
	set_clock_groups
	set_clock_latency
	set_clock_uncertainty
Constraints	set_input_delay
	set_output_delay
Exceptions	set_false_path
	set_max_delay
	set_min_delay
	set_multicycle_path

Note:
(1) These SDC commands are TimeQuest timing analyzer-specific extensions.

An additional important requirement for a timing analysis tool is the effective integration of static timing analysis with the place-and-route engine in order to achieve highest quality of results (QoR). This requirement is met by integrating the place-and-route engine with the timing analysis engine. A major benefit of this integration is that it

facilitates the diagnosis and debugging of the design's critical paths. Integration enables internal nodes to remain "observable", and thus makes it easier for the design engineer to trace any given node back to the original timing constraints. Lack of integration typically causes node name mismatches between the original netlist and the netlist generated by the place-and-route engine.

TimeQuest Timing Analyzer

With the Quartus II 6.0 release, Altera has introduced a new timing analyzer aimed at FPGA designers creating complex clocking schemes and source-synchronous interfaces, and at ASIC designers already familiar with the SDC format. Quartus II design software users can easily analyze the examples above using the TimeQuest timing analyzer.

The TimeQuest timing analyzer has a fast on-demand and interactive reporting interface, which saves designers' time by enabling them to quickly analyze critical paths and request more detailed timing analysis only when needed. The TimeQuest timing analyzer's fast on-demand reporting is complemented by a powerful graphical user interface (GUI) that reports the timing analysis results in an intuitive, easy-to-understand graphical format, further enhancing designer productivity. Advanced designers have unlimited access to all the TimeQuest functionality via its Tcl interface.

The TimeQuest GUI assists with the task of specifying timing constraints. Figure 3 shows how the designer can create a clock definition using the TimeQuest timing analyzer. A similar interface is used to specify additional timing constraints such as input delays, output delays, or exception constraints such as false paths and multicycle paths.

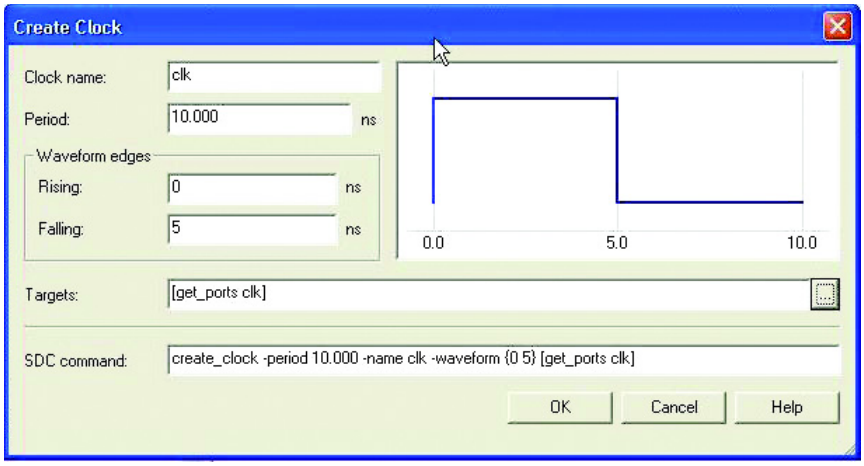


Figure 3. TimeQuest Timing Analyzer's Create Clock Dialog Window

This dialog window enables Quartus II users to write constraints using the SDC format. With native SDC support, TimeQuest users adopt a powerful industry-standard timing analysis methodology and achieve a higher degree of productivity due to the use (and re-use) of SDC and Tcl-based scripts. Designers already proficient with SDC can enter timing constraint and exception commands directly in the constraints file (*.sdc). For these designers, TimeQuest timing analyzer also supports a powerful Tcl interface, shown in Figure 4. With it, designers can automate repetitive timing analysis tasks, which is particularly useful when porting ASIC designs into FPGAs.

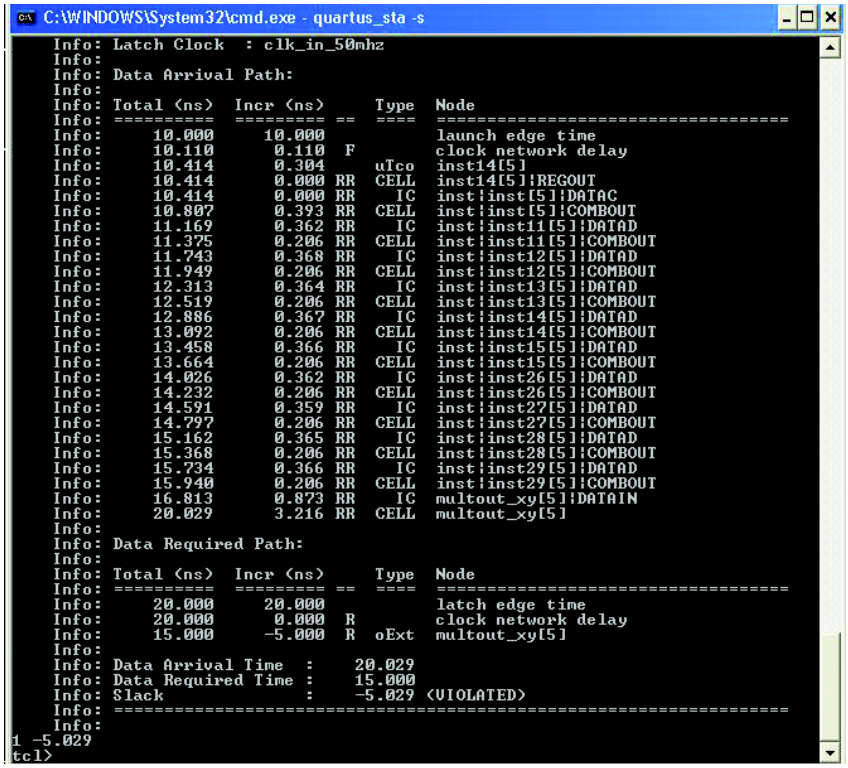


Figure 4. TimeQuest Timing Analyzer's Tcl Interface

After specifying timing constraints, the designer performs a detailed timing analysis. The TimeQuest GUI includes a tasks pane (Figure 5) that provides easy access to commonly performed tasks, such as netlist setup and generation of timing reports. The tasks pane shows a workflow illustrating processes to be completed before timing sign-off. Engineers who are new to static timing analysis will find this pane helpful, while advanced users will appreciate its quick and convenient access to TimeQuest analysis and report functionality features.

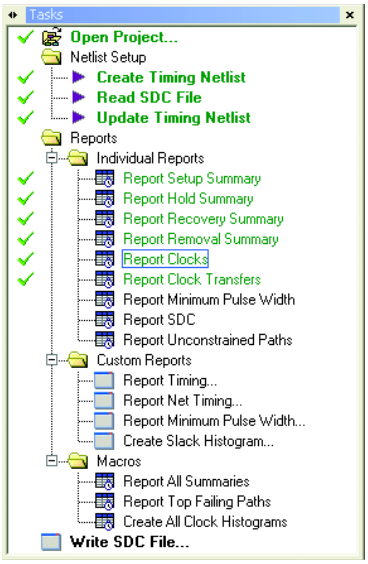


Figure 5. TimeQuest Timing Analyzer's Tasks Pane

TimeQuest timing analyzer has very fast interactive reporting capabilities. While viewing the slack report, the designer can ask for more information about a failing clock domain. Figure 6 illustrates the result of this query.

In the TimeQuest View Pane, the designer accesses all the details concerning the selected path. The pie chart provides a quick view into whether any reported timing is due to logical (cell) or interconnect (ic) delays. The FPGA designer decides then whether to further optimize the interconnect delays, or—in the event of a high cell delay—to perform design/architectural changes prior to recompilation.

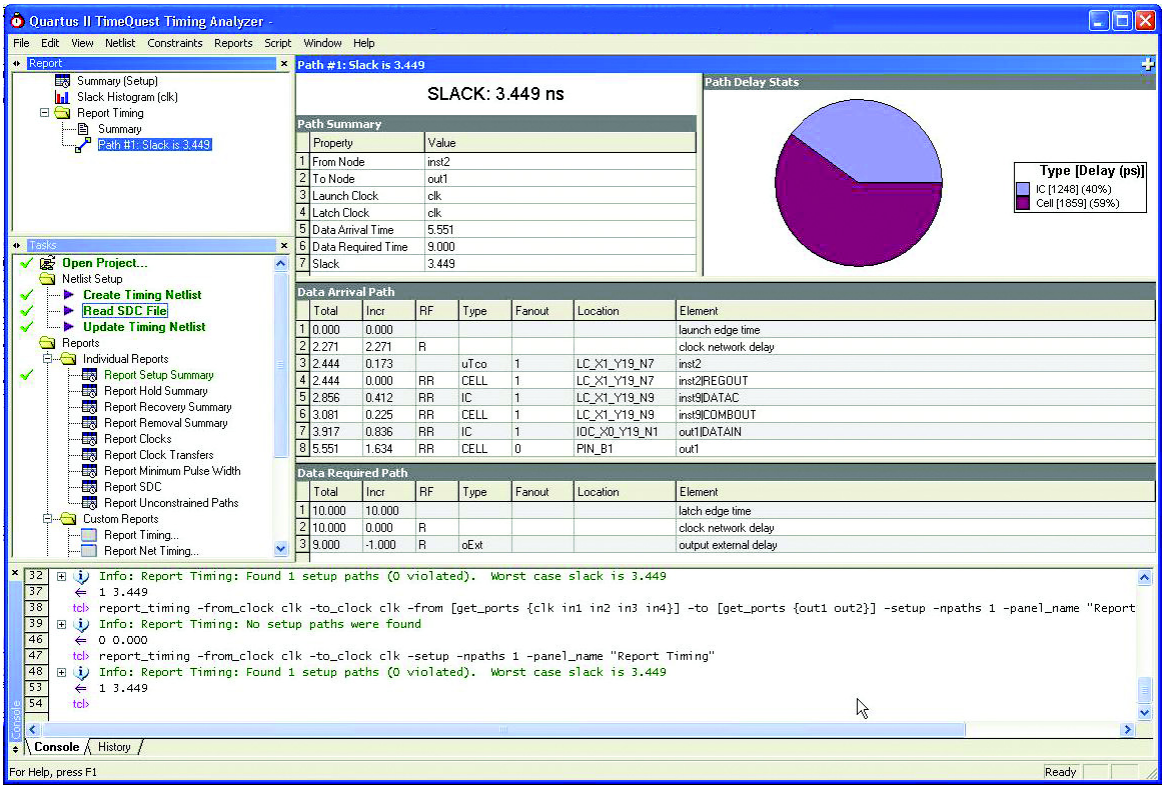


Figure 6. TimeQuest Timing Analyzer's Fast, Interactive Reporting Capabilities

Any of the nodes or paths analyzed with the timing analyzer can be cross-probed with the integrated floorplan, so that potential routing congestions can be analyzed by looking at the interconnect density between logic blocks.

Conclusion

Advances in process technologies enable the adoption of FPGAs in a wide range of applications, which traditionally belonged solely to the ASIC domain. To meet market requirements and achieve target performance, FPGA design engineers are adopting new design styles and complex clocking schemes, such as clock multiplexing in 10-Mbps, 100-Mbps, and Gigabit Ethernet applications. In addition, they are also embedding in their designs source-synchronous clock interfaces (such as those found in DDR and DDR2 interfaces) that are difficult to analyze using traditional FPGA timing analysis. On the methodology side, the designer's productivity is enhanced by the ability to use the industry-standard SDC format to specify design constraints. From a technology standpoint, the timing analysis engine must be integrated with the place-and-route engine, and have a powerful GUI that allows fast access to critical-path data to ensure fast and reliable timing closure.

These requirements were the driving force in the definition and development of TimeQuest timing analyzer, a new ASIC-strength timing analyzer with native support for the industry-standard SDC format. The timing analyzer is included in Quartus II software starting with version 6.0 of the subscription edition. With TimeQuest timing analyzer,

designers perform advanced timing verification on high-end FPGA-based designs, empowering them to quickly and efficiently reach and maintain timing closure.

Further Information

- Altera's TimeQuest timing analysis solution:
  www.altera.com/TimeQuest
- Timing analysis methodology is further discussed in Chapters 6 and 7 of the Quartus II Handbook, Volume 3
  http://www.altera.com/literature/quartus2/lit-qts-verification.jsp
- Quartus II development software and TimeQuest customer training:
  www.altera.com/training

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com