AN 545: Design Guidelines and Timing Closure Techniques for HardCopy ASICs

© July 2010 AN-545-2.1

This application note covers topics from a timing closure perspective, for the successful migration to HardCopy® ASICs from Altera's FPGAs. The first section covers metastability, synchronous and asynchronous resets, and synchronizing an asynchronous reset. This section also describes techniques for passing control and data signals across clock domains using a handshake mechanism and FIFO.

The next section covers the concept of timing requirements at the system level (I/O constraints) and describes in detail what you must do to fully and properly constrain your design (I/O and internal timing constraints). This section focuses on timing constraints by introducing a few basic commands available in the TimeQuest Timing Analyzer that are the most critical for any design. This section also describes how you can properly constrain your design using Altera's TimeQuest Timing Analyzer available in the Quartus® II software.

The following are required reading for the "*Timing Analysis*" section in volume 3 of the *Quartus II Handbook*:

- *TimeQuest Analyzer Quick Start Tutorial*
- *Quartus II TimeQuest Timing Analyzer*
- *Switching to the TimeQuest Timing Analyzer*

The final section describes timing closure techniques for HardCopy ASICs; for example, design partitioning, incremental compilation, Quartus II optimization settings, and timing closure for the interfaces.

Top 10 Issues for Designs Migrating to HardCopy ASICs

When migrating to HardCopy ASICs, there are a few timing constraint situations that warrant closer examination. The top 10 issues that are frequently overlooked when targeting designs to the HardCopy ASIC are as follows:

- Virtual clocks—virtual clocks are necessary for designs migrating to HardCopy ASICs to account for proper clock uncertainties between inter-clock transfers, intra-clock transfers, and I/O transfers. For more information about virtual clocks, refer to"The Need for Virtual Clocks" on page 19.
- Clock uncertainty—clock uncertainty is required in designs migrating to HardCopy ASICs. For more information about clock uncertainty, refer to "Clock Uncertainty" on page 25.
- Special situations for input and output delays—there are many instances where input or output delay constraints may not be required; for example, a dedicated clock output from a phase-locked loop (PLL). For more information about input and output delays, refer to"Handling Special Situations for Input and Output Delays" on page 30.

- The `set_clock_groups` command—this is an elegant way to cut timing paths for clock domains that are not related, but the primary use of `set_clock_groups` is to identify asynchronous clock groups for the purpose of crosstalk analysis. For more information about the `set_clock_groups` constraint, refer to "The set_clock_groups Command" on page 32.
- Using minimum and maximum delays—minimum and maximum delay constraints are frequently misused. The most common misuse is to constrain a path with either the minimum delay constraint only or the maximum delay constraint only. For more information about the correct use of minimum and maximum delays, refer to "Using the set_min_delay and set_max_delay Constraints" on page 36.
- Multicycle paths—multicycle constraints are another area where designers make mistakes. While setting multicycle path constraints, it is very important to examine the waveform to validate the proper clock relationship. The waveform view tool in the TimeQuest Timing Analyzer GUI aids in reviewing these constraints. For more information about multicycle paths, refer to "Multicycle Paths" on page 38.
- Synopsys Design Constraint (**.sdc**) file tricks and tips—the TimeQuest Timing Analyzer is Tcl-based; designers that know Tcl scripting can do many things. For example, clock uncertainty is a requirement in HardCopy ASICs and not a requirement in the FPGA. How do you address this issue without having to create separate **.sdc**s for the HardCopy ASICs and the FPGA? For more information about **.sdc**s, refer to ".sdc Tricks and Tips" on page 41.
- Reading multiple **.sdc**s—there may be a need to read in multiple **.sdc**s for timing analysis and timing closure in your design due to the use of Altera's intellectual property (IP); for example, the ALTMEMPHY megafunction or other third party IP. Using the TimeQuest Timing Analyzer makes this easy to accomplish. For more information about reading multiple **.sdc**s, refer to "Reading Multiple .sdcs" on page 41.
- Metastability-safe reset design—you can design metastability-safe reset structures, synchronize resets, and safety sequence resets. For more information about metastability-safe reset design and sequencing resets, refer to "Metastability Safe Reset Design" on page 11.
- HardCopy performance improvement—the FPGA is a prototype vehicle; it may not need to run at a desired frequency. Because the HardCopy ASIC is a production vehicle, it may need to run at faster speeds. What can the designer do within the Quartus II software to meet this criterion? For more information about HardCopy performance improvement, refer to "HardCopy Performance Improvement" on page 47.

"Appendix A: Design Example" on page 56 details the topics just reviewed.

Design Guidelines

This section describes how to handle multi-clock domain designs for best design practices and guidelines for timing analysis. Wherever possible, hints, tips, and features in the Quartus II software are highlighted.

Metastability

The output of an edge-triggered flipflop has two valid states: high and low. To ensure reliable operation, designs must meet flipflop timing requirements for both setup and hold. Whenever a signal violates the setup and hold requirements of a flipflop, the output of the flipflop can be metastable. Metastable outputs oscillate, or hover, between high and low states for a brief period of time and eventually settle down to either a low state or high state. This can cause the system to fail.

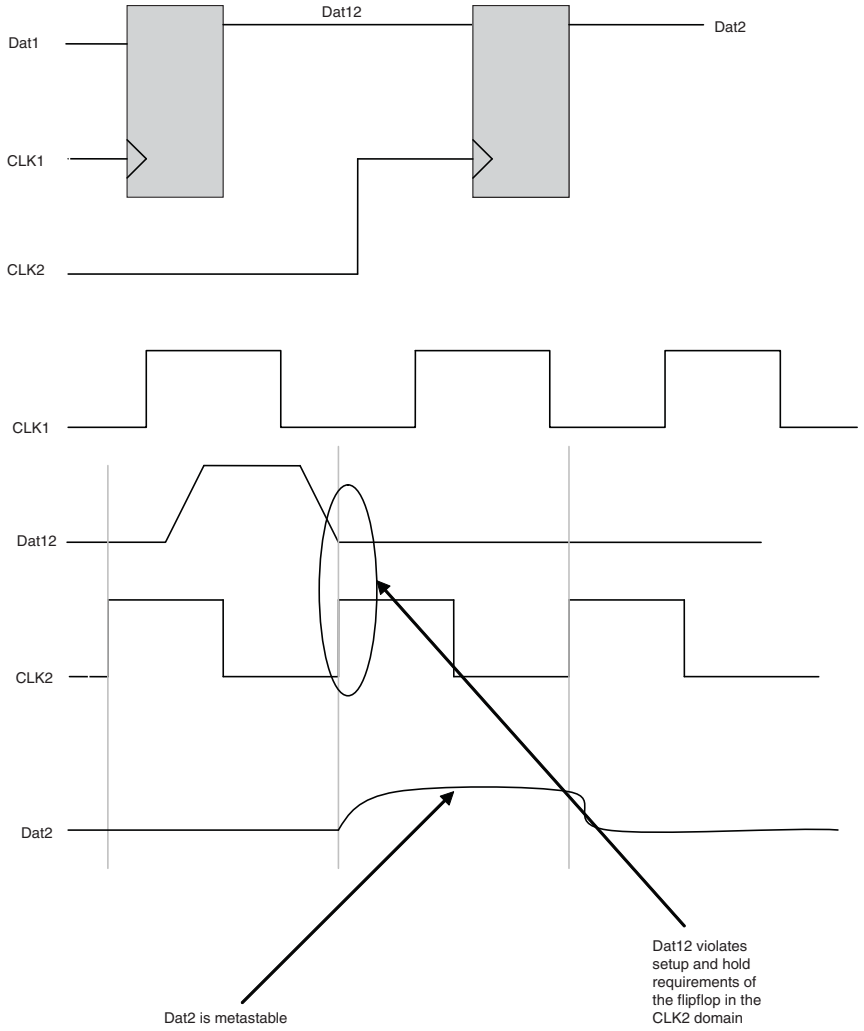
In multi-clock designs, ensure that signals crossing clock domains are properly synchronized. Typically, this involves using a synchronizer, a handshake mechanism, or a FIFO.

Synchronizer

A synchronizer samples an asynchronous signal and outputs a version of the signal that has transitions synchronized to the local clock.

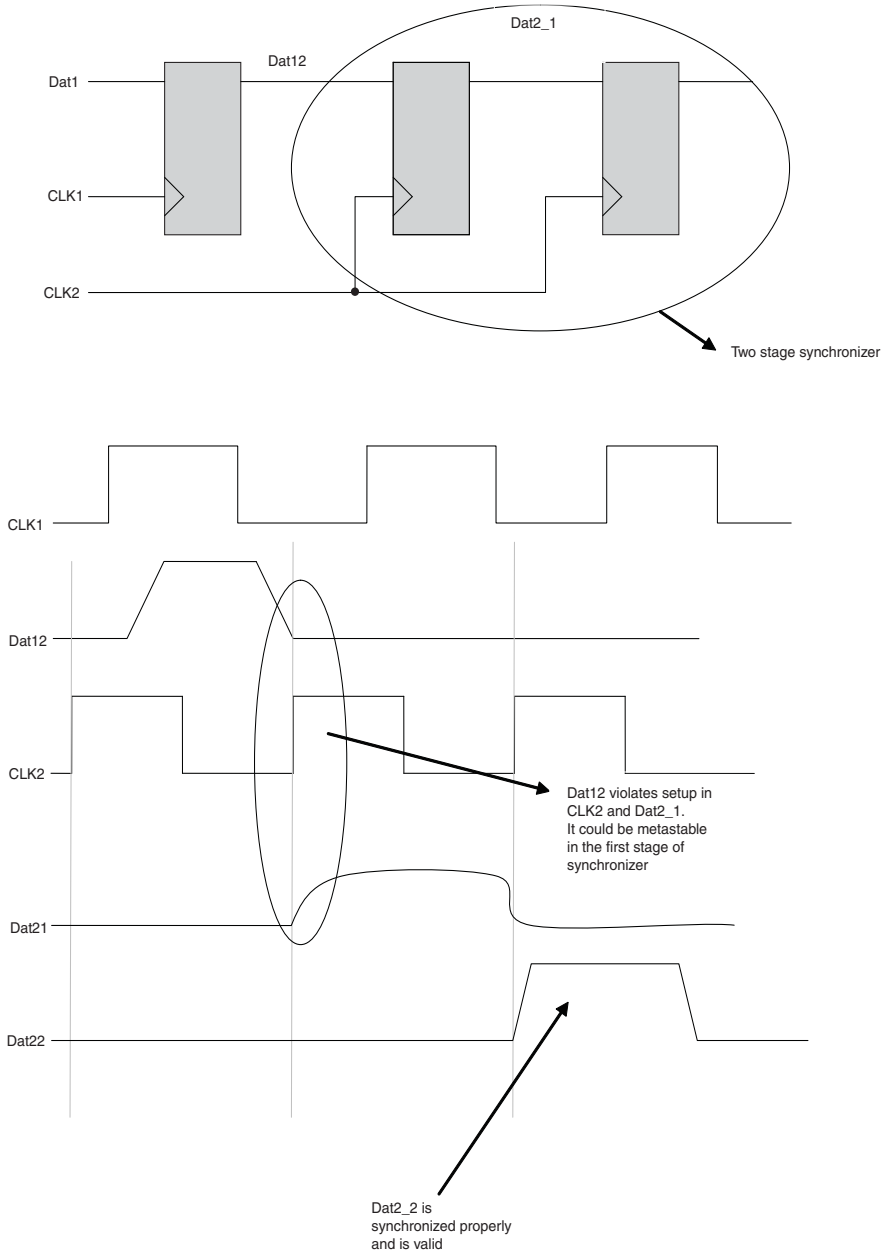
Figure 1 shows an example where a potential metastable state for a signal generated in one clock is sampled too close to the rising edge of another clock domain. If the metastable output is feeding to a state machine, the design might not function.

Figure 1. Synchronization Failure for a Signal Crossing Clock Domains



One way to avoid metastability is by using a synchronizer. The most common synchronizer used is a two-flipflop synchronizer or a two-stage synchronizer, as shown in Figure 2.

Figure 2. Two-Stage Synchronizer



Proper signal naming conventions reduce problems when running static timing analysis. Set the false paths for the signal crossing clock domain using wildcards. For the output of the first stage, you only have to perform minimum time (hold) checks.

The number of synchronizing flipflops required depends on the technology library.

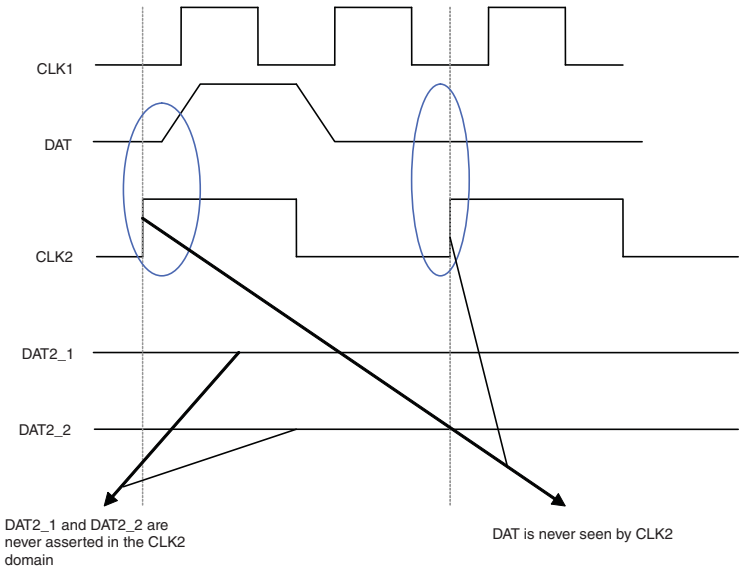
You can use the Quartus II software to analyze the mean time between failures (MTBF) due to metastability in your design. For more information, refer to the *Managing Metastability with the Quartus II Software* chapter in volume 1 of the *Quartus II Handbook*.

If you are concerned that a particular asynchronous signal in your design may lead to metastability, use the Quartus II software to determine and optimize the MTBF for the registers that synchronize that signal. You can improve the MTBF by adding additional synchronizer registers as required.

How to Synchronize a Signal from a Fast Clock Domain to a Slow Clock Domain

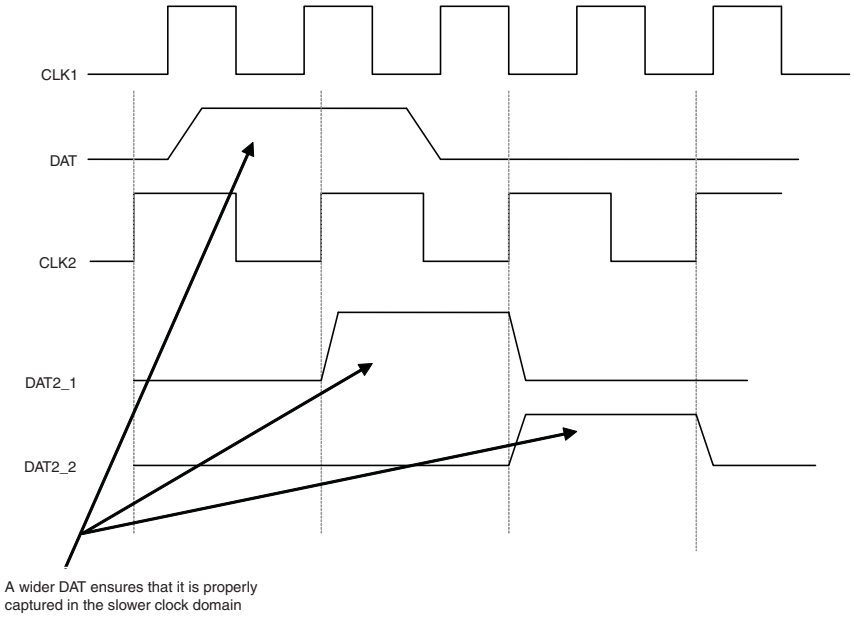
When synchronizing a signal from a fast clock domain to a slow clock domain, ensure that the signal is properly captured in the slow clock domain. Figure 3 shows an example where the signal is transferred from a fast clock domain to a slow clock domain. Because the signal is asserted for only one clock period in the fast clock domain, the signal is not captured in the slow clock domain.

Figure 3. Incorrect Way to Pass Signal from a Fast Clock Domain to a Slow Clock Domain



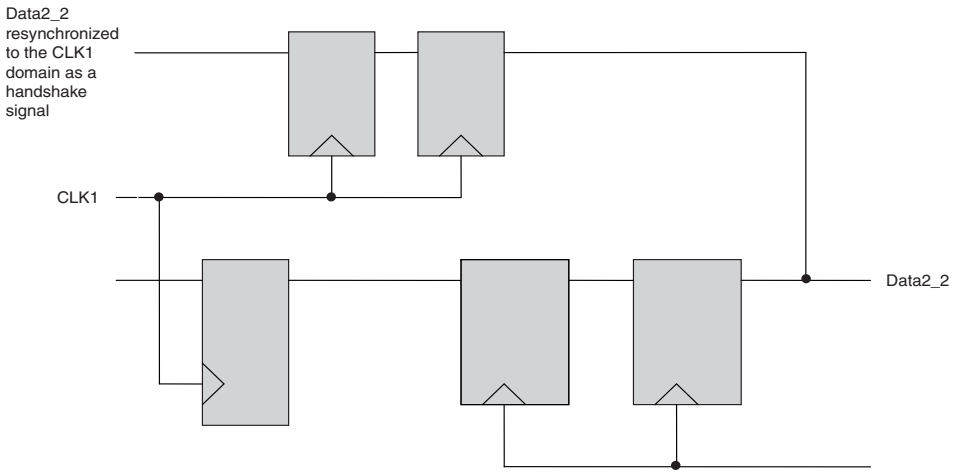
An easy solution is to ensure that the signal in the fast clock domain is asserted for a period that exceeds the cycle time of the clock in the slow clock domain, as shown in Figure 4. The signal must exceed the cycle time of the slow clock to ensure proper operation across process, voltage, and temperature (PVT) variations and any skew that might arise either in the data or clock signals.

Figure 4. Passing a Signal from a Fast Clock Domain to a Slow Clock Domain



Another solution is to use a handshake mechanism to ensure that the slower clock domain captures the data. This involves passing the DAT2_2 signal back to the fast clock domain as an acknowledge (ack) signal using two synchronous flipflops. If the ack signal does not come back within a certain number of fast clock cycles, the data can be retransmitted to the slow clock domain, as shown in Figure 5.

Figure 5. Handshake Mechanism for Signal Crossing Clock Domains



Another way to ensure that the control signal is properly synchronized in the slow clock domain is to de-assert the control signal in the fast clock domain after DAT2_2 is re-synchronized in the fast clock domain. This is described in "Datapath Synchronization" on page 9.

How to Pass Multiple Control Signals Between Clock Domains

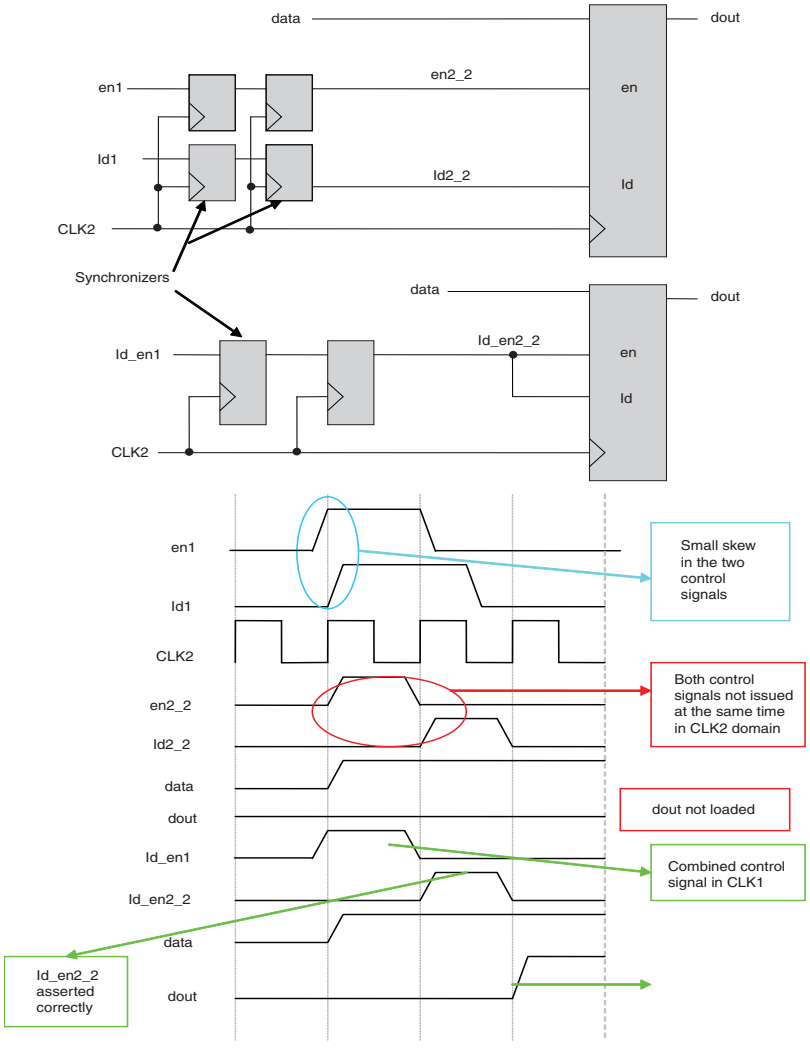
When passing multiple control signals across clock domains, be sure to sequence the control signals properly. Using synchronizers alone is not sufficient. Pay attention to the order in which the control signals are presented in the receiving clock domain to see if it is important. The following example describes the problem.

Passing Two Simultaneously Required Control Signals Across Clock Domains

In Figure 6, the control signals `ld2_2` and `en2_2` are simultaneously required for the signal data to be loaded into the register in the CLK2 clock domain. A small skew, due to either board or PVT variations on the two control signals (`ld1` and `en1`) in the CLK1 clock domain, can cause incorrect data (or no data) to be loaded in the register in the CLK2 clock domain.

The solution to this problem is to combine the two control signals into one control signal in the two clock domains, as shown in Figure 6.

Figure 6. Passing Multiple Control Signals Between Clock Domains



There are several scenarios where having to pass more than one control signal between clock domains is necessary. The solution is to limit the number of control signals crossing domains. Use one or, at most, two control signals from the sending clock domain to generate the other control signals in the receiving clock. Simulating your design with random delays to the control signals is one way to identify potential design problems. Relying only on timing analysis may not be sufficient.

Datapath Synchronization

Using synchronizers is not an accepted practice for datapath synchronization, particularly when multiple data bits are involved. When you use synchronizers, skew in the data bits can lead to data being captured incorrectly. For datapath synchronization, use a handshake mechanism or FIFOs. In the Quartus II software, view the datapath transfers by using the `report_clock_transfers` command. Be sure to view the Design Assistant messages. Example 1 and Example 2 show typical Design Assistant Warnings for datapath synchronization.

Example 1. Design Assistant Warning, D101

```
Critical Warning: (High) Rule D101: Data bits are not synchronized when
transferred between asynchronous clock domains. Found <number>
asynchronous clock domain interface structure(s) related to this rule
```

Example 2. Design Assistant Warning, D103

```
Critical Warning: (High) Rule D103: Data bits are not correctly
synchronized when transferred between asynchronous clock domains. Found
<number> asynchronous clock domain interface structure(s) related to
this rule
```

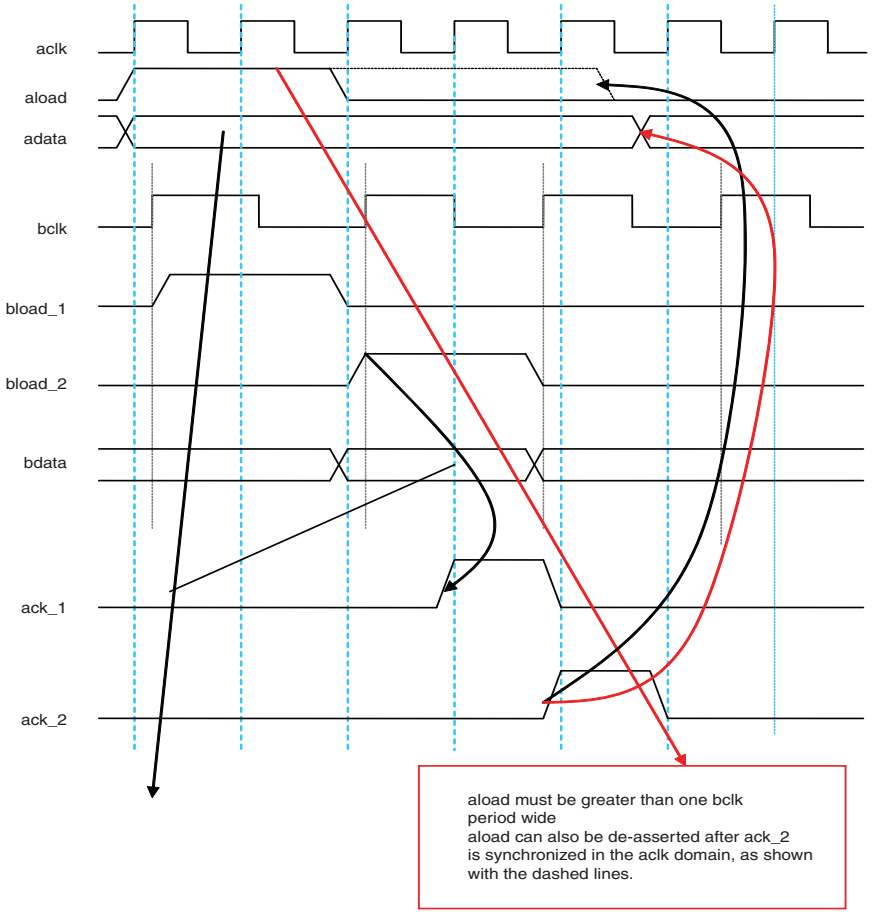
Handshake Mechanism for Datapath Synchronization Between Clock Domains

Whenever data is passed between clock domains, use a handshake mechanism to ensure proper synchronization. However, the disadvantage of using the handshake mechanism is that the more the control signals are used for handshaking, the longer the latency is to pass data between clock domains.

In most cases, a simple two-way handshake sequence is sufficient. In the sending clock domain, both the data and control signal are sent to the receiving clock domain. After synchronizing the control signal, the receiver clocks the data into a register. The control signal in the sending clock domain must be greater than one cycle of the receiving clock domain for it to properly synchronize. The control signal is then sent back to the sender as an `ack` signal. When the `ack` signal is received, the sender can change the value being driven on the data bus. You can also de-assert the control signal from the sending clock domain when the `ack` signal is received from the receiving clock domain. This ensures that control has been captured correctly in the receiving clock domain.

Figure 7 shows the handshake mechanism between two clock domains for passing data. Latency is caused because the next data word cannot be transmitted until the ack signal is received in the sending clock domain.

Figure 7. Handshake Mechanism for Transmitting Data Between Two Clock Domains



Using a FIFO for Datapath Synchronization Between Clock Domains

Another way to pass data between clock domains is to use a FIFO. The FIFO is a dual-port memory used for storage: one port is clocked by the sender; the other port is clocked by the receiver. The sender writes data into the FIFO and the receiver pulls data out from the FIFO. The control signals in a FIFO are typically the full flag, empty flag, and sometimes the half-full flag. Some designers also use two more flags, which indicate when the FIFO is almost full and almost empty. However, this can be difficult when generating accurate full and empty flags.

The advantage of using a FIFO is low latency when compared with the handshake mechanism.

Use the MegaWizard® Plug-In Manager to generate the FIFO of the proper width and depth.

When migrating from an Altera® FPGA to a HardCopy Series ASIC for FIFOs, keep the PLL M and N counters the same in both revisions because the jitter values and static phase error depends on the M and N counters. Choosing the same M and N counter values between the FPGA and HardCopy revision also limits the variation in the analog component. You can change the phases.

Metastability Safe Reset Design

Review your design prior to selecting a reset scheme. Choosing to use synchronous reset or asynchronous reset depends on your chip, board, or system requirements. Debugging asynchronous reset (or reset in general) can be problematic with silicon. When resets are de-asserted asynchronously, there can be recovery and removal problems. ASIC designers spend a lot of time ensuring their reset logic works correctly by running simulations both on the register transfer level (RTL) and gate-level netlist.

Reset recovery time refers to the time between de-asserted reset and when the clock signal goes high again. Violating recovery time causes metastability on register outputs.

Removal problems can occur if there are slight differences in propagation delays in either (or both) the reset signal or clock signal, which can cause some registers to exit the reset state before the others.

In the Quartus II software, review the Design Assistant Critical Warnings and Warnings and Information Only messages for reset-related issues. Example 3 shows a sample Design Assistant Warning.

Example 3. Design Assistant Warning

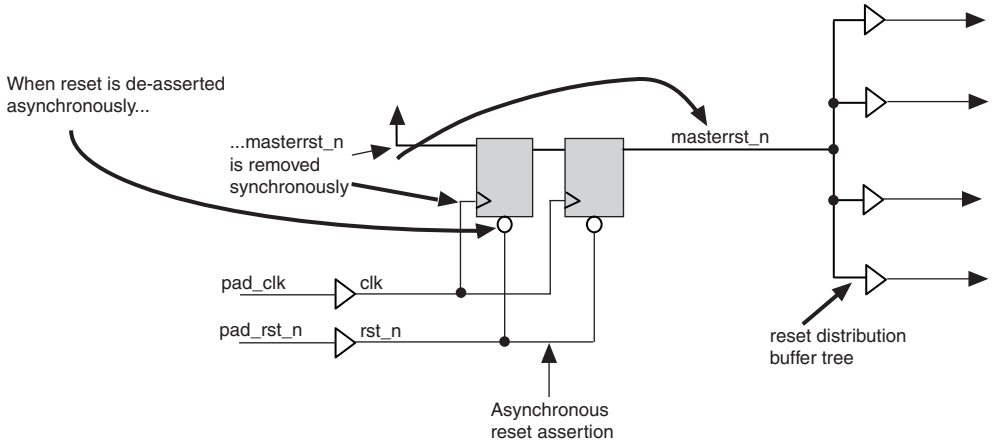
```
Warning: (Medium) Rule R102: External reset should be synchronized using
two cascaded registers. Found <number> node(s) related to this rule.
```

Altera recommends synchronizing asynchronous reset in your design. The following section describes a technique to synchronize asynchronous resets and the proper sequence for resets in your design. A few examples of RTL coding styles are also described.

Synchronizing an Asynchronous Reset

Figure 8 shows how to synchronize an asynchronous reset signal.

Figure 8. Synchronizing an Asynchronous Reset Signal *(Note 1)*



Note to Figure 8:

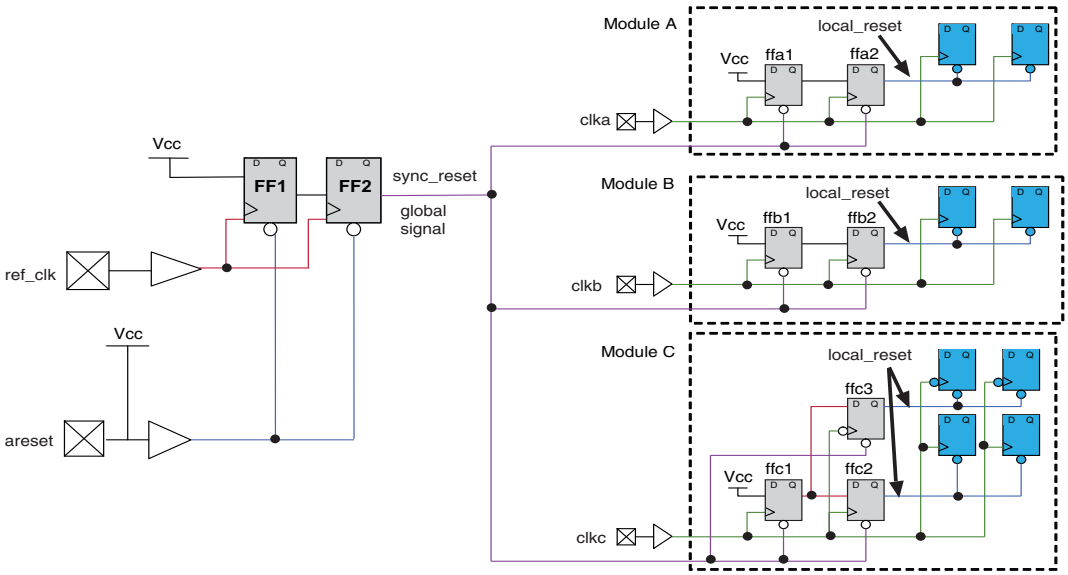
(1) Source from Cummings, Mills, Golson; "Asynchronous & Synchronous Reset Design Techniques"; SNUG Boston 2003; Synopsys Inc.

In Figure 8, the reset signal (`rst_n`) from the pad is connected to the `ACLR` pins of the flipflop. The D-input of the first stage flipflop is tied high. When reset is de-asserted asynchronously, `masterrst_n` is removed synchronously within the design. In this scheme, the reset distribution buffer tree is very similar to the clock-tree itself.

Because the reset signal typically has a large fan-out, you may want to promote this net to the appropriate global or regional clock network in the Quartus II software. For more information about nets that have high fan-out but are not promoted to either global or regional clock networks, in the Quartus II software, look in the **Fitter resource** section under **Non-Global High Fan-out signals**. You can promote some of these high fan-out nets to global or regional clock networks using the Assignment Editor. This improves recovery and removal times for the reset signal.

You can then use synchronized reset (`masterrst_n`) in a design with multiple clock domains, as shown in Figure 9. Within each clock domain, the reset signal is synchronized again.

Figure 9. Synchronizing an Asynchronous Reset in Multiple Clock Domains



At the top level, the `areset` signal is asynchronously asserted and the reset signal propagates through the reset retiming registers (highlighted in gray) as they are reset down to the user logic (highlighted in blue).

In this example, `ref_clk` is a system-level input reference clock to the device and `areset` is a chip-level asynchronous reset.

When `areset` is de-asserted, both FF1 and FF2 hold their outputs low until the next clock arrival of `ref_clk`. In the event that the `ref_clk` edge arrives close to the time `areset` is removed from FF1 and FF2, there is no metastability risk on FF2 because the D-input value is already at a stable value.

At the first subsequent clock edge of `ref_clk` after `areset` is de-asserted, FF1 updates the `Q` pin to logic 1, while FF2 remains logic 0, holding the system in reset for an additional clock cycle.

At the second subsequent edge of `ref_clk`, FF2 updates the `Q` pin to logic 1, while the global reset signal propagates it to each block of the design, where the local reset retiming stages reside.

The reset signal of Module A (or Module B) is metastability safe for the same reason the top-level reset circuit is metastability safe, regardless of when `clka` (or `clkb`) arrives relative to `ref_clk`. The second stage ffa2 has a static value on its input even if the clock arrives too close to the removal of `global_reset`.

Use separate local reset circuits for Module A and B because they reside in separate clock domains. The purpose is to retime resets within local clock domains to allow a full clock period for recovery and removal timing closure.

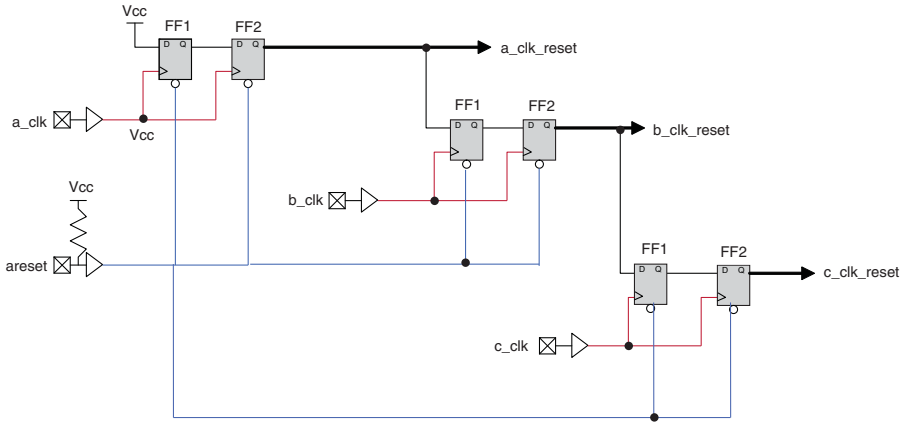
In Module C, both clock edges are used because of design requirements; however, a half clock cycle recovery and removal requirement is often difficult to meet in high-speed clock domains. The solution is to split the second stage flipflops (ffc2 and ffc3) by the clock edge and allow reset of user logic to have a full clock cycle to meet recovery and removal timing.

In Figure 9, reset is removed in the three clocks domains in a non-coordinated fashion. In non-coordinated reset removal, the order in which each of the clock domains come out of reset does not matter.

Proper Sequencing of Resets in a Design

Use the scheme shown in Figure 10 in instances where the reset must be properly sequenced between modules.

Figure 10. Reset Sequencing Between Modules



In this scheme, the logic in the `a_clk` domain is reset first, followed by the logic in the `b_clk` domain, followed by the logic in the `c_clk` domain. Ensure that data is not accessed from a module until that module completely comes out of reset.

Notice the signal naming conventions used in Figure 10 for the resets in the different clock domains.

A good design practice is to separate resets in your design. For example, isolating the PLL reset from a chip reset in the RTL allows you to reset the PLL and logic independently. This allows a safe reset of the PLL to allow it to re-lock in a loss-of-lock situation in a design. You could hold the RTL in reset while the PLL is reset and locks again. After lock is re-established, you can release the reset of the core logic driven by the PLL to receive a good clock signal. Another example is to use reset as a way to save dynamic power by holding a block of logic in reset when not in use, thus limiting the toggle activity in the block of logic to just the clock tree.

Conclusion, References, and Required Reading

This section describes a number of design guidelines. However, there are many scenarios not covered by this application note. For more information, refer to the following articles:

- Clifford E. Cummings and Don Mills, *Synchronous Resets? Asynchronous Resets? I am So Confused! How Will I Ever Know Which to Use*?, Synopsys Users Group Conference, San Jose, CA, April 2002. Also available at www.sunburst-design.com/papers
- Clifford E. Cummings, *Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs*, Synopsys Users Group Conference, San Jose, CA, March 2001. Also available at www.sunburst-design.com/papers
- Clifford E. Cummings and Peter Alfke, *Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons*, Synopsys Users Group Conference, San Jose, CA, April 2002. also available at www.sunburst-design.com/papers

The following application notes are required reading:

- *AN 469: Stratix III Design Guidelines*
- *AN 519: Stratix IV Design Guidelines*
- *AN 536: Design Guidelines for Preparing HardCopy II ASICs*

Timing Constraints

This section provides an overview of TimeQuest Timing Analyzer constraints.

For a full description of the constraints, refer to the *Timing Analysis* section of volume 3 of the *Quartus II Handbook*.

TimeQuest Timing Analyzer

The Quartus II TimeQuest Timing Analyzer is a powerful ASIC-style static timing analysis tool that you can use as a sign-off tool for Altera FPGAs and HardCopy ASICs. The TimeQuest Timing Analyzer uses industry-standard constraint, analysis, and reporting methodologies. Use the timing analysis engine as a GUI or a command-line interface to constrain, analyze, and report results for all timing paths in your design.

This application note includes some required basic timing constraints. These constraints are described in the following section.

For a detailed description of the timing commands, refer to the Help menu in the Quartus II software or *Volume 3: Verification* in the *Quartus II Handbook*.

This application note assumes a basic knowledge of **.sdc**-style constraints. For more information about the Quartus II TimeQuest Timing Analyzer, refer to the *Timing Analysis*" section in Volume 3 in the *Quartus II Handbook*.

You can use the Quartus II software to analyze the MTBF due to metastability in your design. For more information, refer to the *Managing Metastability with the Quartus II Software* chapter in volume 1 of the *Quartus II Handbook*.

If you are concerned that a particular asynchronous signal in your design may lead to metastability, use the Quartus II software to determine and optimize the MTBF for the registers that synchronize that signal. You can improve the MTBF by adding additional synchronizer registers as required.

Constraints

For HardCopy ASICs, it is very important to fully constrain your design. Failure to properly and fully constrain your design may cause device failure when migrating to HardCopy ASICs. This section describes how to best constrain your design for completeness.

To perform static timing analysis, specify all timing requirements. The basic constraints in a design are clocks and clock uncertainty, input and output delays, and timing exceptions (false paths, multi-cycle paths, and minimum and maximum delays).

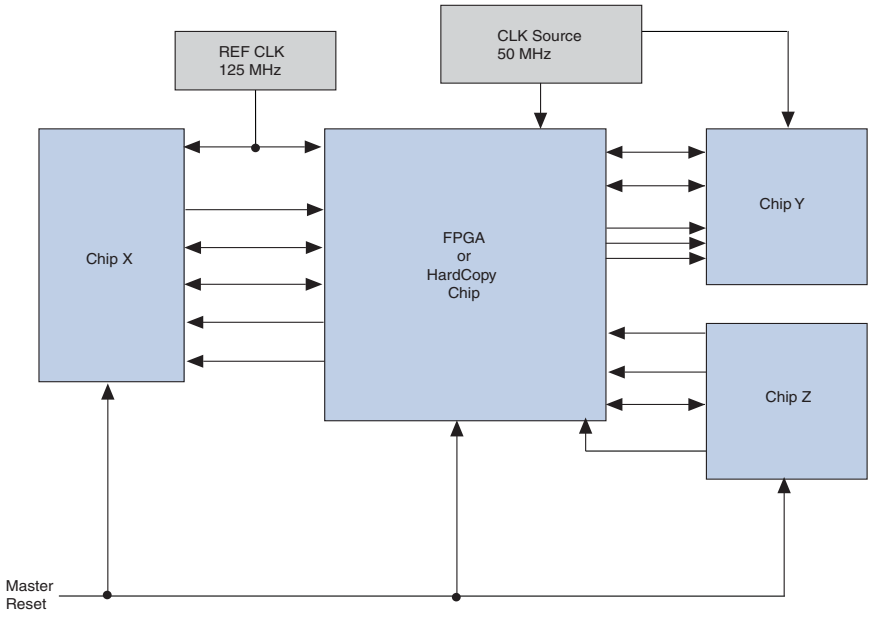
All HardCopy ASIC designs must use the TimeQuest Timing Analyzer as the default timing analysis engine. If you still use the Classic Timing Analyzer, refer to the *Switching to the TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Constraining Your Design

Start constraining your design by creating a system-level block diagram. The block diagram identifies the various chips, their interfaces, and the timing requirements for overall system performance. After this, define your clocks, inputs, outputs, bi-directional pins, and reset scheme (asynchronous or synchronous) to be employed system-wide or within a chip/design.

Figure 11 shows a sample system block diagram.

Figure 11. Sample System Block Diagram



Defining I/O Constraints

After you understand the basics, continue by choosing which FPGA prototype device and HardCopy ASIC migration device to use. You can then plan your pin placements, I/O drive strength, capacitive loading, and I/O standards. If you based your timing or performance requirements on the data sheets of the interfacing chips, your design goals may have to be revisited based on the data you have collected so far.

The Quartus II software automatically defaults the I/O drive strengths based on the I/O standard you use. After you decide which FPGA to prototype and which HardCopy companion device to use, revisit the I/O drive strength requirement.

You can change the default setting of the I/O drive strength in the **Device Setting** tab (on the Assignments menu, click **Device**). Altera recommends changing the I/O drive strengths for various I/O standards through the Pin Planner. However, if you change the I/O drive strength using the Assignment Editor, you will achieve the same results.

Clocks and Clock Uncertainty

The following sections describe clocks and clock uncertainty.

Clocks

Begin constraining your design by defining the clocks. Typically, the board has a crystal oscillator as the reference clock, which is fed to a PLL (either external or inside an Altera FPGA or HardCopy ASIC). When using an external PLL, you must consider PLL jitter and clock latencies of the source.

For clocks, use the `create_clock` and `create_generated_clock` commands available in the Quartus II TimeQuest Timing Analyzer. The `create_clock` and `create_generated_clocks` commands assume ideal clocks and do not take into account board effects, skew due to the clock network, PLL jitter, static phase errors, and so on. If you use an internal PLL, you can model uncertainties (latency, jitter, and skew) by using the `set_clock_uncertainty` command. This is described in the following section.

After you have defined your clocks using the `create_clock` command, use the `derive_pll_clocks` command (if you use the available PLLs in the FPGA or HardCopy ASIC) to automatically derive the generated clocks in your design. This is a good way to automatically create generated clocks because these commands check your design and PLL settings, and automatically generate the internally generated clocks or PLL output clocks for phase shift and `M` and `N` counter values, thus preventing user errors.

When using internal PLLs, Altera recommends using the PLL reconfig megafunction. This allows you to change the phase shift at a later time and to get a better estimate of board parameters.

The PLL reconfig megafunction also allows you to have different phase settings between the FPGA revision and the HardCopy ASIC revision. For more information about how to have different PLL settings between the FPGA and HardCopy revisions, refer to *AN432: Using Different PLL Settings Between Stratix II and HardCopy II Devices*.

When using an internal PLL, choose a crystal source that is easily multiplied and divided to get the desired frequency. The higher the `M` and `N` counter values, the greater the jitter and static phase errors.

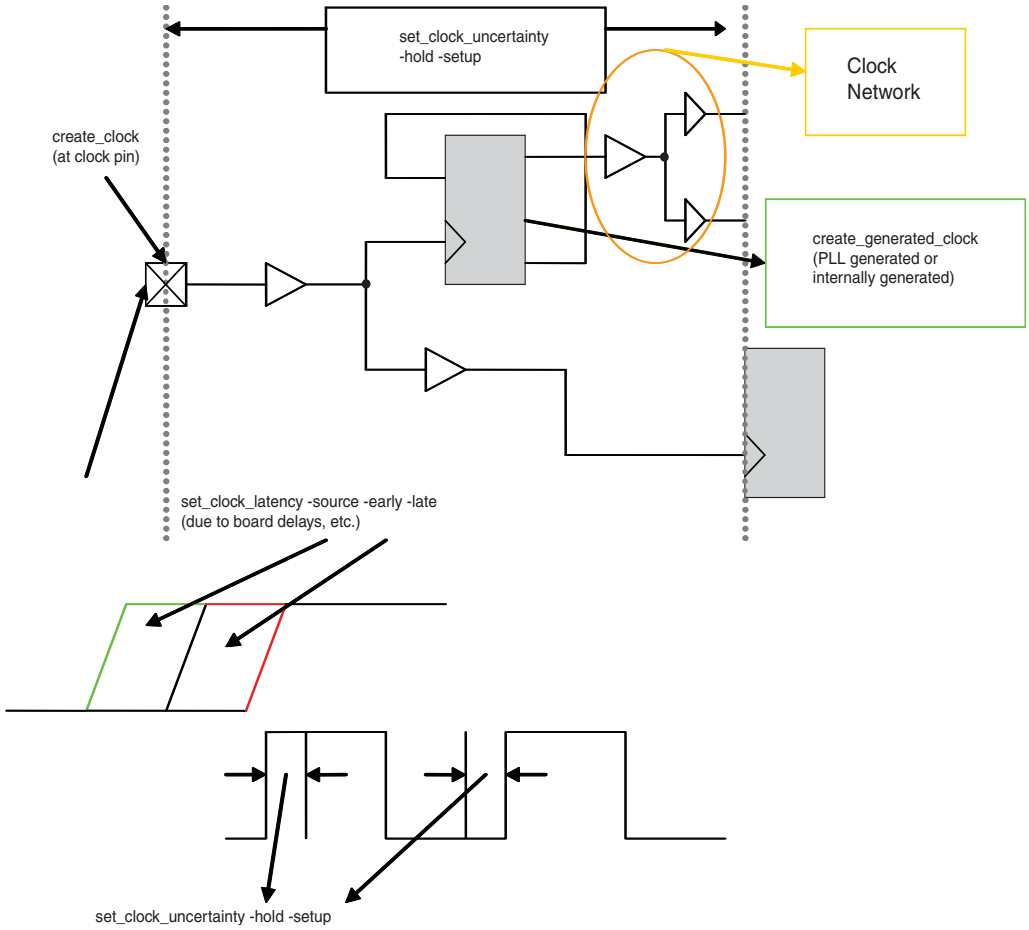
Beginning with Quartus II software version 7.1, automatic self reset on loss-of-lock is no longer supported in Stratix II devices and HardCopy II ASICs.

If your system qualification requires that you run tests above or below the nominal operating frequency, enter the highest frequency for PLL outputs when configuring the PLL megafunction. This ensures the voltage-controlled oscillator (VCO) of the PLL is properly centered to minimize jitter and static phase error.

When planning pin placements on the PLL, place the clocks on dedicated clock input pins. When placed on dedicated clock pins, the clock signals use global or regional clock networks, thus reducing skew and latency.

Figure 12 shows commands associated with clocks that are available in the TimeQuest Timing Analyzer in the Quartus II software.

Figure 12. Clocks and Clock Uncertainty and the Various Commands in the TimeQuest Timing Analyzer



Creating a Clock Using the create_clock Command

The `create_clock` command defines a clock.

Usage: `create_clock [-h | -help] [-long_help] [-add] [-name <clock_name>] -period <value> [-waveform <edge_list>] [<targets>]`

The Need for Virtual Clocks

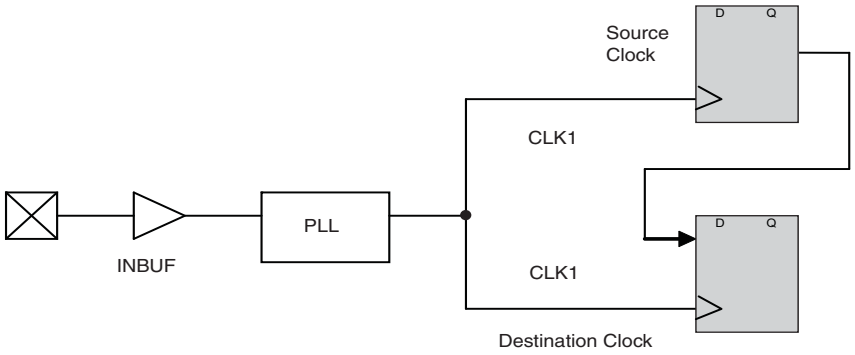
Clock transfers are classified as follows:

- "Intra-Clock Transfer"
- "Inter-Clock Transfer"
- "I/O Transfers"

Intra-Clock Transfer

Intra-clock transfer occurs when the source and destination clocks come from the same PLL, I/O, or clock pins, as shown in Figure 13.

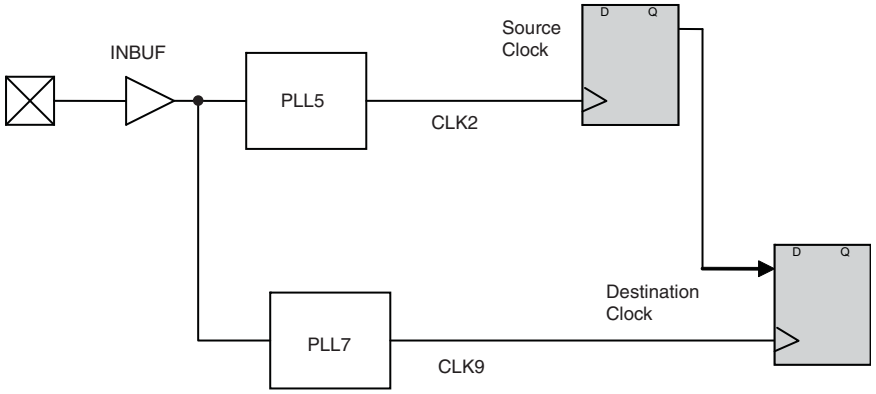
Figure 13. Intra-Clock Transfer



Inter-Clock Transfer

Inter-clock transfer occurs when the source and destination clocks come from different PLLs or I/O clock pins, as shown in Figure 14.

Figure 14. Inter-Clock Transfer



I/O Transfers

I/O transfers are between an off-chip output pin and an on-chip input pin. There can also be an I/O transfer from an on-chip output pin and an off-chip input pin. These are shown in Figure 15 and Figure 16.

Figure 15. Input Transfer

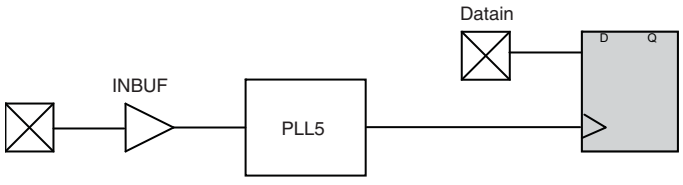
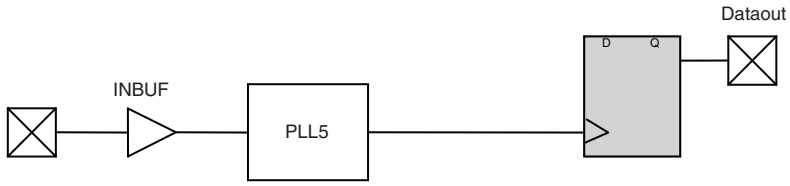


Figure 16. Output Transfer



For clock uncertainties to be accurately accounted for in each of the transfers, create virtual clocks. This separates the clock domains for the I/O and core. Do this to set the correct clock uncertainty numbers in the clock uncertainty constraints, which differ for the core and I/O transfers. The benefit of using virtual clocks is that it allows you to make separate and distinct clock uncertainty constraints for each unique clock transfer. Without using virtual clocks, the clock uncertainty values are incorrect for I/O timing transfers and you may have difficulty closing timing.

There is an exception for source synchronous output clocks because the `set_output_delay` constraint must be set relative to the output clock.

The TimeQuest Timing Analyzer still calculates the proper clock uncertainty in this case using `derive_clock_uncertainty`.

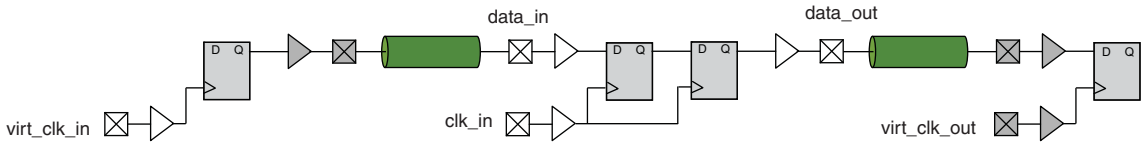
"Case 1: No PLL Involved" and "Case 2: With PLL" on page 23 describe the correct virtual clock use. Clock uncertainty values are different between the two implementations in each case.

Case 1: No PLL Involved

Figure 17 shows an instance where no PLL is involved. In this case, there is an input transfer, an intra-clock core transfer, and an output transfer. Traditionally, you would set the input delay and output delay with `clk_in` as the reference clock. This intra-clock transfer appears the same as the I/O clock transfer. Incorrect clock uncertainty is applied for I/O transfers.

Virtual clocks for input transfer and output transfer provide separate and distinct clock transfers from the `clk_in` core to the intra-clock transfer.

Figure 17. Case 1: No PLL Involved



No Virtual Clocks

Example 4 shows the constraints where no virtual clocks are created.

Example 4. Constraints with No Virtual Clocks

```
create_clock -period 10 -name clk_in [get_ports {clk_in}]
set_input_delay -clock [get_clocks {clk_in}] -max 2 [get_ports {data_in}]
set_input_delay -clock [get_clocks {clk_in}] -min 0 [get_ports {data_in}]
set_output_delay -clock [get_clocks {clk_in}] -max 2 [get_ports {data_out}]
set_output_delay -clock [get_clocks {clk_in}] -min 0 [get_ports {data_out}]
```

The resulting clock transfers are:

From `clk_in` to `clk_in` only

The resulting clock uncertainty is:

From `clk_in` to `clk_in` Setup: 150 ps Hold: 50 ps

Virtual Clocks

Example 5 shows the constraints with virtual clocks.

Example 5. Constraints with Virtual Clocks

```
create_clock -period 10 -name clk_in [get_ports {clk_in}]
create_clock -period 10 -name virt_clk_in
create_clock -period 10 -name virt_clk_out
set_input_delay -clock [get_clocks {virt_clk_in}] -max 2 [get_ports {data_in}]
set_input_delay -clock [get_clocks {virt_clk_in}] -min 0 [get_ports {data_in}]
set_output_delay -clock [get_clocks {virt_clk_out}] -max 2 [get_ports
{data_out}]
set_output_delay -clock [get_clocks {virt_clk_out}] -min 0 [get_ports
{data_out}]
```

The resulting clock transfers are:

- From `clk_in` to `clk_in`
- From `virt_clk_in` to `clk_in`
- From `clk_in` to `virt_clk_out`

The resulting clock uncertainty is:

- From `clk_in` to `clk_in` Setup: 150 ps Hold: 50 ps
- From `virt_clk_in` to `clk_in` Setup: 130 ps Hold: 130 ps
- From `clk_in` to `virt_clk_out` Setup: 130 ps Hold: 130 ps

As you can see from "Case 1: No PLL Involved" on page 21, not only are the clock transfers separated, but the clock uncertainty numbers are different for the intra-clock core transfer and the output I/O transfer.

Case 2: With PLL

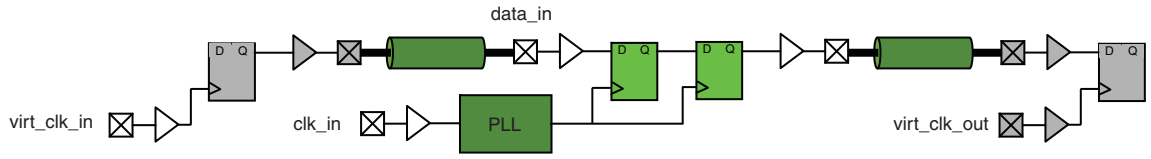
Traditionally, you would set the input and output delay with `clk_in` as the reference clock.

Separate transfers happen in this case:

- `clk_in` to PLL
- PLL to PLL
- PLL to `clk_in`

Virtual clocks for input and output transfer provide separate and distinct clock transfers from `clk_in` to core, intra-clock transfer of the core PLL clock, and core PLL clock to the output. Using virtual clocks separates a presumed inter-clock core transfer from the `clk_in` to the PLL from an I/O transfer to and from the PLL, as shown in Figure 18.

Figure 18. Case 2: With PLL



No Virtual Clocks

The following are constraints without using virtual clocks:

Example 6. Constraints with No Virtual Clocks

```
create_clock -period 10 -name clk_in [get_ports {clk_in}]
set_input_delay -clock [get_clocks {clk_in}] -max 2 [get_ports {data_in}]
set_input_delay -clock [get_clocks {clk_in}] -min 0 [get_ports {data_in}]
set_output_delay -clock [get_clocks {clk_in}] -max 2 [get_ports {data_out}]
set_output_delay -clock [get_clocks {clk_in}] -min 0 [get_ports {data_out}]
derive_pll_clocks
```

The resulting clock transfers are:

- From `clk_in` to `pll_clk0`
- From `pll_clk0` to `pll_clk0`
- From `pll_clk0` to `clk_in`

The resulting clock uncertainty (based on example design PLL settings) is:

- From clk_in to pll_clk0 Setup: 270 ps Hold: 230 ps
- From pll_clk0 to pll_clk0 Setup: 100 ps Hold: 50 ps
- From pll_clk0 to clk_in Setup: 230 ps Hold: 270 ps

Virtual Clocks

Example 7 shows the constraints with virtual clocks.

Example 7. Constraints with Virtual Clocks

```
create_clock -period 10 -name clk_in [get_ports {clk_in}]
create_clock -period 10 -name virt_clk_in
create_clock -period 10 -name virt_clk_out
set_input_delay -clock [get_clocks {virt_clk_in}] -max 1 [get_ports {data_in_a* data_in_b*}]
set_input_delay -clock [get_clocks {virt_clk_in }] -min 0 [get_ports {data_in_a* data_in_b*}]
set_output_delay -clock [get_clocks {virt_clk_out }] -max 1 [get_ports {data_out* }]
set_output_delay -clock [get_clocks {virt_clk_out }] -min 0 [get_ports {data_out* }]
derive_pll_clocks
```

The resulting clock transfers are:

- From virt_clk_in to pll_clk0
- From pll_clk0 to pll_clk0
- From pll_clk0 to virt_clk_out

The resulting clock uncertainty (based on example design PLL settings) is:

- From virt_clk_in to pll_clk0 Setup: 150 ps Hold: 100 ps
- From pll_clk0 to pll_clk0 Setup: 100 ps Hold: 50 ps
- From pll_clk0 to virt_clk_out Setup: 100 ps Hold: 150 ps

Using virtual clocks results in lower and more accurate clock uncertainty values. This not only makes timing closure easier but it also allows you to separate the I/O transfers from the core transfers. When timing closure is difficult, you can easily identify which part of the design—the I/O or core—is not meeting timing.

The derive_pll_clocks Command

This command identifies PLLs or similar resources in the design and creates generated clocks for their output clock pins. Multiple generated clocks may be created for each output clock pin if the PLL is using clock switchover—one for the inclk[0] input clock pin and one for the inclk[1] input clock pin.

Usage: `derive_pll_clocks [-h | -help] [-long_help] [-create_base_clocks] [-use_net_name]`

The create_generated_clock Command

This command defines an internally generated clock. If `-name` is not specified, the clock name is the same as the first target in the list or collection. The clock name is used to refer to the clock in other commands.

Usage: `create_generated_clock [-h | -help] [-long_help] [-add] [-divide_by <factor>] [-duty_cycle <percent>] [-edge_shift <shift_list>] [-edges <edge_list>] [-invert] [-master_clock <clock>] [-multiply_by <factor>] [-name <clock_name>] [-offset <time>] [-phase <degrees>] -source <clock_source> [<targets>]`

Table 1 lists the `create_generated_clock` commands and their definitions.

Table 1. create_generated_clock Commands

Command	Definition
`-add`	Add clock to an existing clock node.
`-divide_by <factor>`	Division factor.
`-duty_cycle <percent>`	Specifies the duty cycle as a percentage of the clock period.
`-edge_shift <shift_list>`	List of the edge shifts.
`-edges <edge_list>`	List of the edge values.
`-invert`	Invert the clock waveform.
`-master_clock <clock>`	Specifies the source node clock.
`-multiply_by <factor>`	Multiplication factor.
`-name <clock_name>`	Name of the generated clock.
`-offset <time>`	Specifies the offset as an absolute time shift.
`-phase <degrees>`	Specifies the phase shift in degrees.
`-source <clock_source>`	Source node for the generated clock.
`<targets>`	List or collection of targets.

Clock Uncertainty

You can use the `derive_clock_uncertainty` command to calculate clock uncertainties.

This command automatically calculates clock uncertainties and applies them to the design using the `set_clock_uncertainty` command. The calculated clock uncertainty values are based on I/O buffer, static phase errors (SPE), and jitter in the PLL's clock networks and core noises.

For HardCopy designs, the HardCopy Design Center requires that you set clock uncertainty constraints.

For the FPGA prototype, Altera recommends clock uncertainty constraints.

Input and Output Delays

For HardCopy devices, you must constrain all input and output pins—both the timing requirements and capacitive loads. Use the Assignment Editor to specify the capacitive loads on the pins, or use the Advanced I/O Timing board modeling utility to build your board's RLC network for all outputs and bidirectional signals in your design.

When specifying timing requirements, use the `set_input_delay` and `set_output_delay` commands available in the TimeQuest Timing Analyzer to constrain your input and output pins.

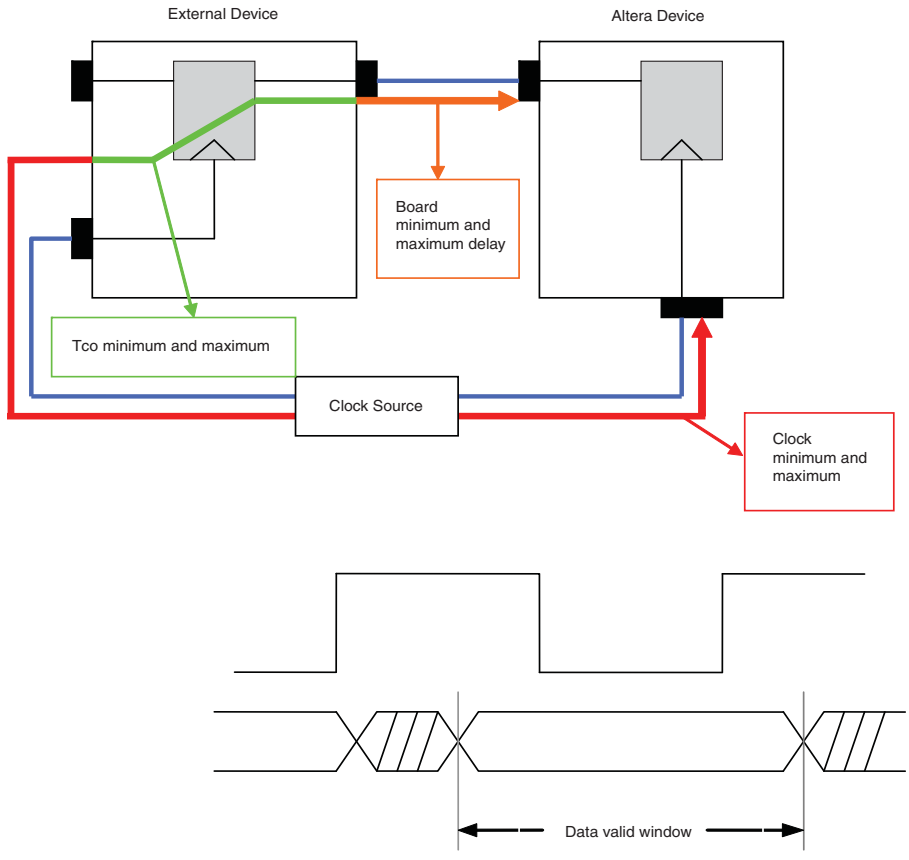
To estimate board delays, follow the guidelines described in *AN 366: Understanding I/O Output Timing for Altera Devices*.

Input Delay

The `set_input_delay` constraint specifies the data arrival time at the input pin of a device with reference to the clock.

Figure 19 shows an input delay path.

Figure 19. Input Delay Path and Data Valid Window



The equations for input delay are as follows:

`set_input_delay` (max/setup) = board max + T_{CO} max - clock min (ext > Altera Device)

`set_input_delay` (min/hold) = board min + T_{CO} min - clock max (ext > Altera Device)

The T_{CO} minimum and maximum values are obtained from the data sheet of the external device and the board minimum and maximum values; the clock minimum and maximum values are obtained from the board parameters.

Usage: `set_input_delay [-h | -help] [-long_help] [-add_delay] -clock <name> [-clock_fall] [-fall] [-max] [-min] [-reference_pin <name>] [-rise] [-source_latency_included] <delay> <targets>`

Table 2 lists the `set_input_delay` commands and their definitions.

Table 2. set_input_delay Commands

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-add_delay`	Add to the existing delays instead of overriding them.
`-clock <name>`	Clock name.
`-clock_fall`	Specifies that the input delay is relative to the falling edge of the clock.
`-fall`	Specifies the falling input delay at the port.
`-max`	Applies value as the maximum data arrival time.
`-min`	Applies value as the minimum data arrival time.
`-reference_pin <name>`	Specifies a port in the design to which the input delay is relative.
`-rise`	Specifies the rising input delay at the port.
`-source_latency_included`	Specifies that the input delay includes added source latency.
`<delay>`	Time value.
`<targets>`	List of input port type objects.

You can specify input delays relative to a port (`-reference_pin`) in the clock network. Clock arrival times to the reference port are added to data arrival times. Non-port reference pins are not supported.

Specifying input delays relative to a reference pin are supported in the TimeQuest Timing Analyzer, but not in the Astro Place and Route tool. Altera's HardCopy Design Center modifies such constraints by adding a generated clock to the reference pin and using that clock for the **-clock** option of the input/output delay, removing the **-reference_pin** option. You are notified of any constraint modifications.

In some cases, the input delay can include clock source latency. By default, the clock source latency of the related clock is added to the input delay value, but when the **source_latency_included** option is specified, clock source latency is not added because it was factored into the input delay value.

Use maximum input delay (`-max`) for clock setup checks or recovery checks; use minimum input delay (`-min`) for clock hold checks or removal checks. If only `-min` or `-max` (or neither) is specified for a given port, the same value is used for both. To fully constrain your design, specify both the minimum and maximum input delays for all your inputs.

You can also specify separate rising (`-rise`) and falling (`-fall`) arrival times at the port. If only one `-rise` and `-fall` is specified for a given port, the same value is used for both.

By default, `set_input_delay` removes any other input delays to the port except for those with the same `-clock`, `-clock_fall`, and `-reference_pin` combination. You can specify multiple input delays relative to different clocks, clock edges, or reference pins by using the **-add_delay** option.

The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or TimeQuest-extension substitution rules.

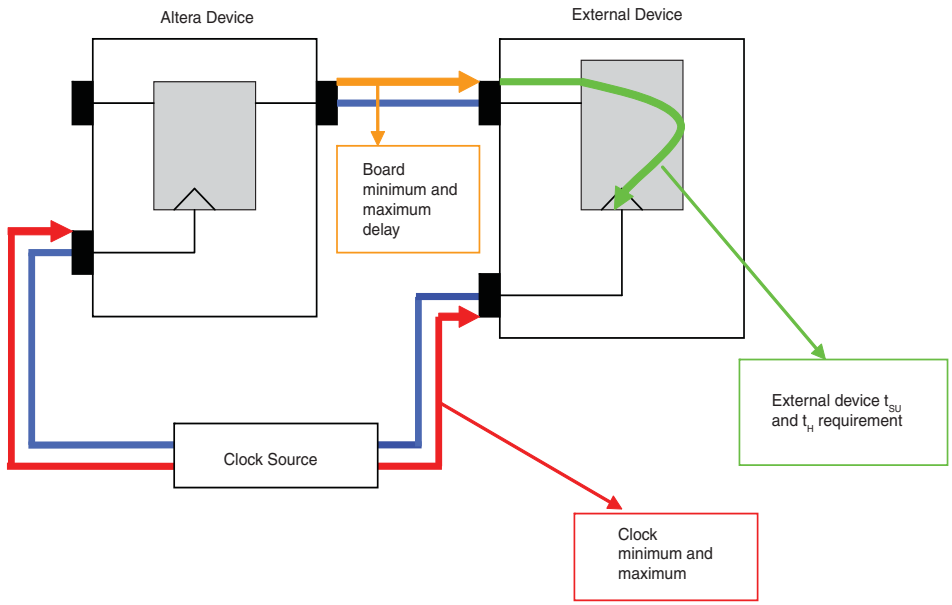
For more information about the `use_timequest_style_escaping` command, refer to the Quartus II Software Help.

Output Delay

The `set_output_delay` constraint specifies the data requirement at the device pin with respect to a clock specified by the **-clock** option. The clock must refer to a clock name in the design.

Figure 20 shows an output delay path.

Figure 20. Output Delay Path



The equations for output delay are as follows:

`set_output_delay` (max/setup) = board max + t_{SU} - clock min (Altera Device > ext)

`set_output_delay` (min/hold) = board min + t_H - clock max (Altera Device > ext)

The t_{SU} and t_H values are obtained from the data sheet of the external device and the board minimum and maximum values; the clock minimum and maximum values are obtained from the board parameters.

With these values, you can constrain your output delays as follows:

```
set_output_delay -max -from <clock> <value> <output pin(s)>
```

```
set_output_delay -min -from <clock> <value> <output pin(s)>
```

Usage: `set_output_delay [-h | -help] [-long_help] [-add_delay] -clock <name> [-clock_fall] [-fall] [-max] [-min] [-reference_pin <name>] [-rise] [-source_latency_included] <delay> <targets>`

Table 3 lists the `set_output_delay` commands and their definitions.

Table 3. set_output_delay Commands

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-add_delay`	Add to the existing delays instead of overriding them.
`-clock <name>`	Clock name.
`-clock_fall`	Specifies that the output delay is relative to the falling edge of the clock.
`-fall`	Specifies the falling output delay at the port.
`-max`	Applies value as the maximum data arrival time.
`-min`	Applies value as the minimum data arrival time.
`-reference_pin <name>`	Specifies a port in the design to which the input delay is relative.
`-rise`	Specifies the rising output delay at the port.
`-source_latency_included`	Specifies that the output delay includes added source latency.
`<delay>`	Time value.
`<targets>`	Collection or list of output ports.

If the output delay is specified relative to a simple generated clock (a generated clock with a single target), the clock arrival times to the generated clock are added to the data required time.

You can specify output delays relative to a port (`-reference_pin`) in the clock network. Clock arrival times to the reference port are added to the data required time. Non-port reference pins are not supported.

Specifying output delays relative to a reference pin are supported in the TimeQuest Timing Analyzer, but not in the Astro Place and Route tool. Altera's HardCopy Design Center modifies such constraints by adding a generated clock to the reference pin and using that clock for the **-clock** option of the input and output delay, removing the **-reference_pin** option. You are notified of any constraint modifications.

Output delays can include clock source latency. By default, the clock source latency of the related clock is added to the output delay value, but when the **-source_latency_included** option is specified, the clock source latency is not added because it was factored into the output delay value.

Use maximum output delay (`-max`) for clock setup checks or recovery checks; use minimum output delay (`-min`) for clock hold checks or removal checks. If only one of the `-min` and `-max` (or neither) is specified for a given port, the same value is used for both.

You can specify separate rising (`-rise`) and falling (`-fall`) required times at the port. If only one of the `-rise` and `-fall` times is specified for a given port, the same value is used for both.

Be sure to specify both the minimum and maximum output delays for outputs in your design to be fully constrained.

For input and output delays, you may have to create a virtual clock to reference the clock for the input or output pins to account for the proper clock uncertainties.

Handling Special Situations for Input and Output Delays

The following are a few special cases where input and output delay constraints are not required:

- Clock output pins
  - Unless you need a specific T_{CO}/min-T_{CO} requirement, clock output pins do not need `set_output_delay` constraints
  - TimeQuest Timing Analyzer detects a generated clock on the output pin
  - Clock port acting as an end point for datapaths, no output delay, minimum and maximum delays, or false-path exceptions found
  - Unless you require a specific T_{CO}/min-T_{CO} requirement, clock output pins do not require `set_max_delay` or `set_min_delay` constraints
  - Using the wrong start-point and/or end-point for the internal clock path is dangerous
- No input or output timing requirement?
  - Use `set_false_path`
- Static input or output monitoring signals?
  - Use `set_false_path`
- Asynchronous signals?
  - Use `set_false_path` or `set_max_delay/set_min_delay`, depending on your situation

Timing Exceptions

By default, the TimeQuest Timing Analyzer assumes that data launched at the starting point of a path is captured by the next rising edge of the clock at the end point. If this is not the case for some paths in your design, you must specify timing exceptions. Failure to do so results in timing violations.

The timing exception commands available in the TimeQuest Timing Analyzer are:

- `set_false_path` ("False Paths" on page 31)
- `set_min_delay` ("Maximum and Minimum Delays" on page 33)
- `set_max_delay` ("Maximum and Minimum Delays" on page 33)
- `set_multicycle_path` ("Multicycle Paths" on page 38)

False Paths

False paths are paths in your design that can be ignored during timing analysis. Typically, these paths cross clock domains and occur in the first stage of the synchronizer. You can use the `set_false_path` command to ignore these paths for timing analysis.

Usage: `set_false_path [-h | -help] [-long_help] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-hold] [-rise_from <names>] [-rise_to <names>] [-setup] [-through <names>] [-to <names>]`

Table 4 lists the `set_false_path` commands and their definitions.

Table 4. set_false_path Commands

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching).
`-hold`	Specifies the `false_path` value (applies only to the clock hold or removal checks).
`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-setup`	Specifies the `false_path` value (applies only to the clock setup or recovery checks).
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching).
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching).

Before you set a large number of false paths in your design, understand the reasons for eliminating those paths from timing analysis. Using wildcards can result in fewer lines in your constraint file, but you must ensure that valid paths are not inadvertently set as false paths.

The set_clock_groups Command

Clock groups provide a quick and convenient way to specify which clocks are not related. Asynchronous clocks are those that are completely unrelated (for example, have different ideal clock sources). Exclusive clocks are those that are not active at the same time (for example, multiplexed clocks). The TimeQuest Timing Analyzer treats both options, `-exclusive` and `-asynchronous`, as if they were the same; however, the difference is important to HardCopy ASICs for signal integrity analysis.

Using `set_clock_groups` causes all the clocks in any group to be cut from all clocks in every other group. This command is equivalent to calling `set_false_path` from each clock in every group to each clock in every other group, and vice versa. This makes `set_clock_groups` easier to specify for cutting clock domains. Using a single-group option notifies the TimeQuest Timing Analyzer to cut this group of clocks from all other clocks in the design, including clocks that are created in the future.

Usage: `set_clock_groups [-h | -help] [-long_help] [-asynchronous] [-exclusive] -group <names>`

Table 5 lists the `set_clock_groups` commands and their definitions.

Table 5. set_clock_groups Commands

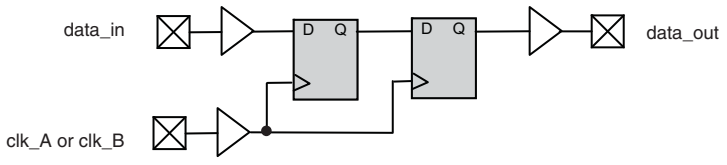
Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-asynchronous`	Specify mutually exclusive clocks (same as the `-exclusive` option).
`-exclusive`	Specify mutually exclusive clocks.
`-group <names>`	Valid destinations.

The set_clock_groups -exclusive Command

If your design has unrelated clocks, use the `set_clock_groups -exclusive` command to declare an exclusive relationship between clocks. This method is more efficient than setting false paths.

This constraint is typically used in place of the `set_false_path` constraint in which the clocks are mutually exclusive. For example, consider the design shown in Figure 21.

Figure 21. set_clock_groups -exclusive Command



In this example, `clk_A` and `clk_B` cannot exist on the clock network at the same time. Therefore, use the `set_clock_groups -exclusive` command as follows:

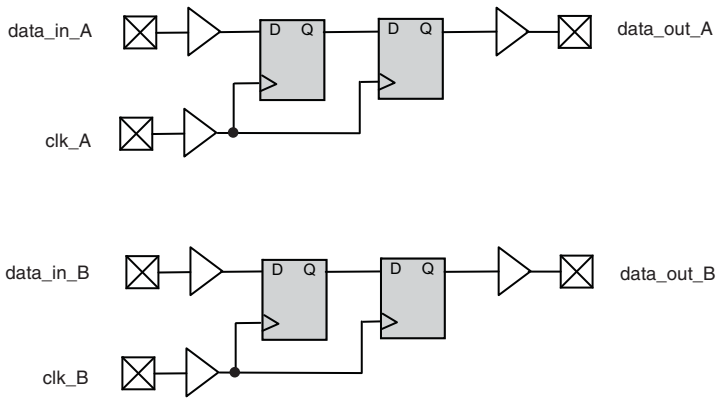
```
set_clock_groups -exclusive -group {clk_A} -group {clk_B}
```

The set_clock_groups -asynchronous Commands

You can use the `set_clock_groups -asynchronous` commands for clocks that have no synchronous relationship with each other.

For example, consider the design shown in Figure 22.

Figure 22. set_clock_groups -asynchronous Commands



In this example, `clk_A` and `clk_B` can be present in the design at the same time, but have no synchronous relationship with each other (inside or outside the chip). Use the following constraints for this case:

```
set_clock_groups -asynchronous -group {clk_A} -group {clk_B}
```

You can use the `set_clock_groups -exclusive` constraint in this case, but that is not the correct choice. Using the `set_clock_groups -asynchronous` constraint is the right choice because Altera's HardCopy Design Center uses PrimeTime-SI to determine the worst-case crosstalk coupling noise possible in analysis. PrimeTime-SI must know which clocks are truly asynchronous so that their worst-case coupling can be determined over overlapping periods. PrimeTime-SI performs crosstalk analysis on synchronous clock domains as well, but the crosstalk effect may be insignificant for clocks that have a synchronous relationship but are out-of-phase with each other, such as two clocks coming from the same PLL.

Maximum and Minimum Delays

The `set_max_delay` and `set_min_delay` commands set the path delay of the specified path to a certain restricted value.

The set_min_delay Constraint

The effect of the minimum delay constraint is similar to changing the hold relationship (launching clock edge - latching clock edge), except that you can apply it to input or output ports without input or output delays assigned to them. Minimum delays are always relative to any clock network delays (if the source or destination is a register) or any input or output delays (if the source or destination is a port). Therefore, input delays and clock latencies are added to the data arrival times. Clock latencies are also added to the data required times. Output delays are subtracted from the data required times.

Usage: `set_min_delay [-h | -help] [-long_help] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-rise_from <names>] [-rise_to <names>] [-through <names>] [-to <names>] <value>`

Table 6 lists the `set_min_delay` commands and their definitions.

Table 6. set_min_delay Commands

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching).
`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching).
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching).
`<value>`	Time Value.

The set_max_delay Constraint

The maximum delay is similar to changing the setup relationship (latching clock edge - launching clock edge), except that you can apply it to the input or output ports without input or output delays assigned to them. Maximum delays are always relative to any clock network delays (if the source or destination is a register) or any input or output delays (if the source or destination is a port). Therefore, input delays and clock latencies are added to the data arrival times. Clock latencies are also added to the data required times. Output delays are subtracted from the data required times.

Usage: `set_max_delay [-h | -help] [-long_help] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-rise_from <names>] [-rise_to <names>] [-through <names>] [-to <names>] <value>`

Table 7 lists the `set_max_delay` commands and their definitions.

Table 7. set_max_delay Commands (Part 1 of 2)

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching).
`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching).

Table 7. set_max_delay Commands (Part 2 of 2)

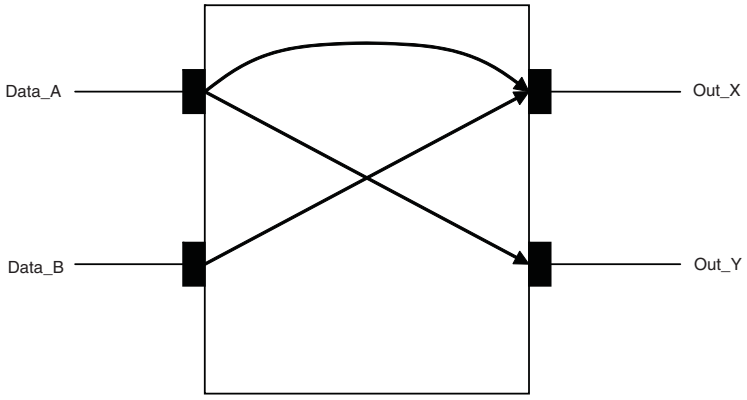
Command	Definition
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching).
`<value>`	Time Value.

When using minimum and maximum delays, use explicit collections for targets, such as:

- Use [`get_ports { <pin name> }`] for I/O targets of constraints
- Do not use [`get_keepers *`] for core targets

In combinational timing circuits, a path exists from a primary input port to a primary output port. This type of circuit does not contain registers. Therefore, it does not require a clock for constraint specification. You only need the maximum and minimum delay from the primary input port to the primary output port to constrain the path for timing requirements, as shown in Figure 23.

Figure 23. Minimum and Maximum Delays for Feed-Through Paths



In this example, assume:

`Data_A` to `Out_X` is 2.25 ns

`Data_A` to `Out_Y` is 3.125 ns

`Data_B` to `Out_X` is 4.25 ns

The commands for the maximum and minimum delays are as follows:

```
set_max_delay -from [get_ports Data_A] -to [get_ports Out_X]
2.25

set_max_delay -from [get_ports Data_A] -to [get_ports Out_Y]
3.125

set_max_delay -from [get_ports Data_B] -to [get_ports Out_X]
4.25

set_min_delay -from [get_ports Data_A] -to [get_ports Out_X] 0

set_min_delay -from [get_ports Data_A] -to [get_ports Out_X] 0

set_min_delay -from [get_ports Data_B] -to [get_ports Out_X] 0
```

Apply the minimum and maximum delay constraint exception to an output port that does not use an output delay constraint. In this case, the setup summary and hold summary report the slack for these paths. Because there is no clock associated with the output port, no clock is reported for these paths. In this case, you cannot report timing for these paths.

To report timing using clock filters for output paths with the minimum and maximum delay constraints, use the `set_output_delay` command for the output port with a value of 0 using an existing clock from the design or a virtual clock as the clock reference in the `set_output_delay` command.

For the minimum and maximum delay constraints used for paths from register-to-register, which are clocked by PLL generated clocks, the PLL phase shifts (on the clocks that feed these registers) are not considered in the timing reports by default. PrimeTime-SI only considers the absolute delay between these registers. Also, frequency and board trace information with respect to the system clock are not considered. You cannot use proper clock uncertainty constraints because no clock is involved.

Using the set_min_delay and set_max_delay Constraints

There are three cases where you can use minimum and maximum delays:

- Port to register
- Register to port
- Port to port (also known as the T_{PD} path)

Always use both `set_max_delay` and `set_min_delay`.

The following scenarios may arise in your design when specifying minimum and maximum delays.

Specifying the Minimum and Maximum Delay for Pin-to-Register (t_{SU} and t_H)

In this case, you are specifying a minimum and maximum delay from a port to a register for input setup and hold checks. In such cases, you are constraining the maximum and minimum data arrival time without taking into account the latch clock arrival time.

For example, consider the following constraint, assuming the PLL clock is actually a 10 ns clock:

```
set_max_delay 6 -from [get_ports {in_data_a*} ]
```

This constraint sets the maximum arrival time for the data signals `in_data_a*` regardless of the arrival time of the latch clock. Therefore, the latch edge is set at 6 ns and the clock is propagated based on the latch edge time.

Consider the following constraint:

```
set_min_delay 0.5 -from [get_ports {in_data_a*}]
```

You are setting the minimum arrival time for the data signals `in_data_a*` regardless of the arrival time of the latch clock. In this case, the hold check latching clock from the PLL is really at time 0 ns, but the minimum delay requirement is 0.5 ns. Therefore, the latch edge is set at 0.5 ns and the clock is propagated based on that latch edge time.

Specifying the Minimum and Maximum Delay for Register-to-Pin (max/min T_{CO})

In this case, you are specifying the minimum and maximum data arrival time regardless of the arrival time of the latch clock for performing proper output setup and hold checks.

Consider the following maximum delay constraint:

```
set_max_delay 8 -from [get_ports {data_out_b*}]
```

This constraint sets up the maximum arrival time for the signals `data_out_b*` from a register to an output pin regardless of the arrival time of the latch clock. Therefore, the latch edge is set at 8 ns and the latch clock is propagated based on that time.

Similarly, the following constraint sets up the hold check:

```
set_min_delay 0.5 -from [get_ports {data_out_b*}]
```

The preceding constraint, which holds the check latching clock from the PLL, is really at time 0 ns, but the minimum delay requirement is 0.5 ns. Therefore, the latch edge is set at 0.5 ns and the clock is propagated based on that latch edge time.

Specifying the Minimum and Maximum Delay for Pin-to-Pin (max/min T_{PD})

In this case, no clock is involved so the latch edge is set as the maximum delay requirement time. The following is an example:

```
set_max_delay 10 -from [get_ports {data_in}] -to [get_ports
{data_out}]

set_min_delay 0 -from [get_ports {tpd_in}] -to [get_ports
{tpd_out}]
```

Specifying the Minimum and Maximum Delay for Register-to-Register (internal path)

For register-to-register paths, the minimum and maximum delay overrides the natural derived setup and hold requirement between the launch and latch clocks.

The following are the constraints for the maximum delay:

```
set_max_delay 8 -from [get_registers {data_2_in_b_reg* }] -to
[get_registers {data_2_out_reg* }]
```

In this case, the default requirement is a 10 ns clock period, but the maximum delay requirement is 8 ns. Clock propagation is factored in for both launch and latch paths. This is more significant in inter-clock transfers.

The following are the constraints for the minimum delay:

```
set_min_delay 0.5 -from [get_registers {data_2_in_b_reg* }] -to
[get_registers {data_2_out_reg* }]
```

In this case, the default requirement is 0 ns because it is an intra-clock transfer, but the minimum delay requirement is 0.5 ns. Clock propagation is factored in for both launch and latch paths. This is more significant in inter-clock transfers.

For cases where you want to use `set_max_delay` and `set_min_delay` to establish an I/O timing requirement (t_{SU}, t_H, t_{CO}, and $t_{CO\text{-}min}$), you must constrain the port using `set_input_delay/output_delay` with a virtual clock. The delay value can be 0 for `-max/min` in `set_output_delay/set_input_delay` because `set_max_delay/set_min_delay` is used to override the setup and hold requirement and thereby establishing the t_{SU}, t_H, t_{CO}, and $t_{CO\text{-}min}$ requirement. Because you set your requirement in `set_max/min_delay`, you do not need to specify a value for the `set_input_delay` or `set_output_delay` constraint, but you still must use a virtual clock to make the clock transfer be correctly identified as an I/O transfer. In this way, `derive_clock_uncertainty` applies uncertainty correctly on this path.

For core paths, you can also use multicycle or clock uncertainty constraints instead of using the `set_max_delay` and `set_min_delay` constraints.

Multicycle Paths

By default, the TimeQuest Timing Analyzer assumes that data launched at a path starting point is captured at the path end point by the next clock edge at the end point. However, some paths may take multiple clock cycles from launch to capture. If you correctly apply multicycle-path timing exceptions to the path, the TimeQuest Timing Analyzer checks for the arrival of data at the appropriate clock edge.

Usage: `set_multicycle_path [-h | -help] [-long_help] [-end] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-hold] [-rise_from <names>] [-rise_to <names>] [-setup] [-start] [-through <names>] [-to <names>] <value>`

Table 8 lists the `set_multicycle_path` commands and their definitions.

Table 8. set_multicycle_path Commands

Command	Definition
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-end`	Specifies that the multicycle is relative to the destination clock waveform (the default).
`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching).
`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching).
`-hold`	Specifies that the multicycle value applies to the clock hold or removal checks.
`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching.
`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching).
`-setup`	Specifies that the multicycle value applies to the clock setup or recovery checks (default).
`-start`	Specifies that the multicycle is relative to the source clock waveform.
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching).
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching).
`<value>`	Number of clock cycles.

For example, a path containing a large combinational logic block (a hardware multiplier), might take two clock cycles. In the absence of timing exceptions, the TimeQuest Timing Analyzer reports a setup violation for this type of circuit because the data would not be available at the next clock edge. A multicycle timing exception set on this path can make the setup check occur at the correct time.

Figure 24 shows an example of a multicycle path for the following commands:

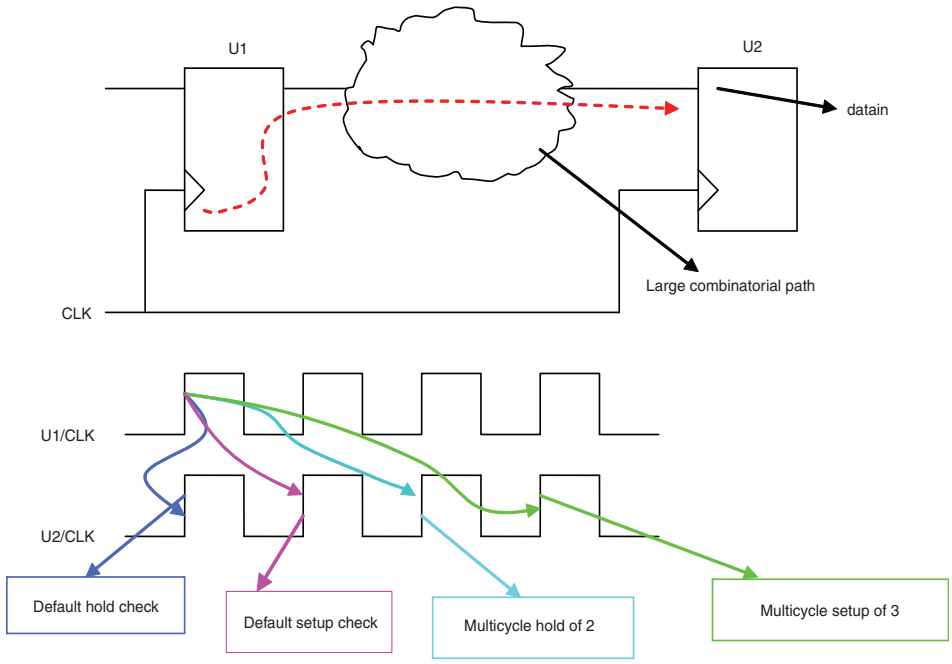
```
set_multicycle_path -setup 3 -from u1|clk -to u22|datain
```

For the preceding command, the setup check is performed at the third clock edge at the destination register, `u22/datain`, after the launch edge.

```
set_multicycle_path -hold 2 -from u1|clk -to u22|datain
```

For the preceding command, the hold check is performed at the second clock edge at the destination register (`u22|datain`) after the launch edge.

Figure 24. Multicycle Example *(Note 1)*



Note to Figure 24:

(1) The multicycle hold of 2 and multicycle setup of 3 are shown only to illustrate which clock edge the setup or hold checks perform. This does not imply that if you have a multicycle setup of 3. You must constrain multicycle hold of 2. Multicycle setup and hold constraints are design dependent.

After specifying your multicycle paths, view them visually and examine the clock edge relationships in the TimeQuest Timing Analyzer GUI to ensure they are correct. Pay attention to the launch and latch clock edges, phase shifts, and clock periods involved.

Using `-start` or `-end` makes a difference, especially with different clock periods for launch and latch clocks. Use the Waveform View tool in the TimeQuest Timing Analyzer GUI to assess if the correct edge relationships for setup and hold are being timed.

You must analyze and constrain cross-clock domain multicycle paths—never assume they are correct by default—particularly when dealing with I/O clock transfers (virtual clocks) and different clock periods for launch and latch clocks.

Exception Priority

If you apply multiple exceptions to a path and there are conflicts, the question arises as to which one the TimeQuest Timing Analyzer will accept. The TimeQuest Timing Analyzer follows the exception priority, from highest to lowest, as follows:

- False paths
- Maximum and minimum delays
- Multicycle paths

For more information about how to use these commands, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

The .sdc

Timing constraints are located in a file that is in **.sdc** format. Constraint files are unlimited. For example, you could have one **.sdc** for constraining your I/Os and clocks and another **.sdc** for clock uncertainty and yet another **.sdc** for setting up false paths, minimum and maximum delays, and multicycle paths. Or you could have **.sdc**s for each design partition and one **.sdc** for the top-level design. You can also have **.sdc**s for slow and fast corners and for FPGA and HardCopy ASIC revisions.

Alternatively, you could choose to have just one **.sdc** for your entire design, with smart Tcl scripts to manage the FPGA and HardCopy ASIC revisions. Remember to add all the relevant **.sdc**s to the design file list in the Quartus II software.

In your **.sdc**s, add comments about how the values were obtained for the constraints.

Altera recommends the following **.sdc** structure:

1. Constraints for clocks.
2. Input and output delays.
3. False path, maximum and minimum delays, and multicycle paths.
4. Clock uncertainty constraints for the HardCopy ASIC using the Tcl construct shown in Example 8.

Example 8. Clock Uncertainty Constraints for Step 4

```
if { $::TimeQuestInfo(family) == "HardCopy II"} {
   # HCII Specific constraints can be placed here
   # For example
   # Clock Uncertainty(CU) which is only needed in HCII
   derive_clock_uncertainty
}
```

.sdc Tricks and Tips

In cases where the FPGA and HardCopy ASIC constraints are different, you can use the constructs just described. For example, for performance improvements in the HardCopy ASIC, you may want to define the clocks differently in the FPGA and HardCopy revisions. In such circumstances, define the base clocks for the HardCopy ASIC within the Tcl construct, then use an `else` clause to define the FPGA base clocks, as shown in Example 9.

Example 9. Base Clocks for the HardCopy ASIC with the else Clause for the FPGA Base Clocks

```
if { $::TimeQuestInfo(family) == "HardCopy II"} {
   # The base clock runs faster in the HC revision
   set period 5
   set waveform {0 2.5}
   } else {
   set period 8
   set waveform {0 4}
 }
create_clock -name iclk -period $period -waveform $waveform
```

Reading Multiple .sdcs

Typically, in a large design, you may have to read-in multiple **.sdc**s. The different **.sdc**s may be due to incorporating third-party IP or Altera IP; for example, the ALTMEMPHY megafunction or HardCopy-specific **.sdc** constraints.

The TimeQuest Timing Analyzer makes reading in multiple **.sdc**s easy. For example, consider this scenario:

- A top-level design that has the **top.sdc** set of constraints
- Separate **.sdc**s for Stratix II and HardCopy II revisions that must be read into the TimeQuest Timing Analysis
- Only the HardCopy II revision must have `derive_clock_uncertainty` added

How do you write the **.sdc** constraints to make this scenario work seamless in the Quartus II software flow? You could list the individual **.sdc**s in the TimeQuest Timing Analyzer settings, but you would have to change them after you create the HardCopy II companion revision to read in the HardCopy-specific constraints.

The following is a better alternative:

Nest the **.sdc** commands within your main **.sdc** using Tcl commands and specify unique constraints, depending on which device family you are compiling in. This way you only have to specify **top.sdc** in your TimeQuest Timing Analyzer settings in your FPGA revision and you do not have to make a settings change for the HardCopy companion revision to change the **.sdc**s. The correct **.sdc**s are read in automatically, depending on which device family you are presently compiling.

Within **top.sdc**, add in the commands shown in Example 10.

Example 10. top.sdc Additions

```
create_clock -period 10 [get_ports {refclk}]
#
# SDC exception to only execute these constraints when Quartus detects you are
in the
# Stratix II device
if { $::TimeQuestInfo(family) == "Stratix II"} {
   read_sdc ddr_settings_sii.dwz.sdc
   # insert any Stratix II specific timing constraints here
}
# SDC exception to only execute these constraints when Quartus detects you are
in the #HardCopy II device
if { $::TimeQuestInfo(family) == "HardCopy II"} {
   read_sdc ddr_settings_hcii.dwz.sdc
   derive_clock_uncertainty
   # insert any additional HardCopy specific timing constraints here
}
```

Sample Template for Your .sdcs

Assume that your design's name is "demo_design".

Example 11 shows the template for **demo_design_constraints.sdc**.

Example 11. demo_design_constraints Template (Part 1 of 3)

```
#############################################################################
#
# FILE: Constraints file
# VENDOR: Altera
# PROGRAM: Quartus II
# VERSION: Version 7.2 Internal Build 207 04/04/2007 SJ Full Version
# DATE: Wed Apr 11 12:03:26 2007
# This file to be used for both the FPGA and the HardCopy Revision
```

Example 11. demo_design_constraints Template (Part 2 of 3)

```
##############################################################################
#*************************************************************
# Create Clock
#*************************************************************
#Constraints for your Clocks would be in this section
#*************************************************************
# Create Generated Clock
#*************************************************************
# PLL and internally generated clocks would be in this section
#*************************************************************
# Set Clock Latency
#*************************************************************
# Based on your board parameters you may or not have clock latency constraints
#*************************************************************
# Set Input Delay
#*************************************************************
# Constrain all your Input pins in this section.
# Separate the input delays according to clk domain. If possible explain how
the
# numbers were obtained!
#*************************************************************
# Set Output Delay
#*************************************************************
# Constrain all your Output pins in this section
#*************************************************************
# Set Clock Groups
#*************************************************************
# Constrain unrelated clocks in your design in this section
#*************************************************************
# Set False Path
```

Example 11. demo_design_constraints Template (Part 3 of 3)

```
#**************************************************************
# If your design has any false paths, set them in this section
# You may want to use set_clock_groups -exclusive instead
#**************************************************************
# Set Multicycle Path
#**************************************************************
# Multicycle paths can be constrained in this section
#**************************************************************
# Set Maximum Delay
#**************************************************************
#Constrain your designs Max delays here
#**************************************************************
# Set Minimum Delay
#**************************************************************
# Constrain your designs min delays here.
#**************************************************************
# Set Clock Uncertainty
#**************************************************************
# Clock Uncertainty for the HardCopy revision goes here
if { $::TimeQuestInfo(family) == "HardCopy II"} {
# HCII Specific constraints can be placed here
# For example
# Clock Uncertainty(CU) which is only needed in HCII
# FPGA Timing model is pessimistic so CU not needed - you can still model CU in
FPGA
# if you want to, just use the following constraint outside of this TCL
construct.
derive_clock_uncertainty
}
```

Summary

This section described how to constrain your design for both timing and input and output loads and also described some of the commands available in the TimeQuest Timing Analyzer. It offered useful tips for successful timing closure. Although it is important to fully constrain your design, do not over constrain your design. If you want additional guard band for your design, consider clock uncertainty constraints. Remember to constrain all your inputs and outputs and take special care of signals crossing clock domains. Use the `report_ucp` command to report a list of unconstrained paths in your design.

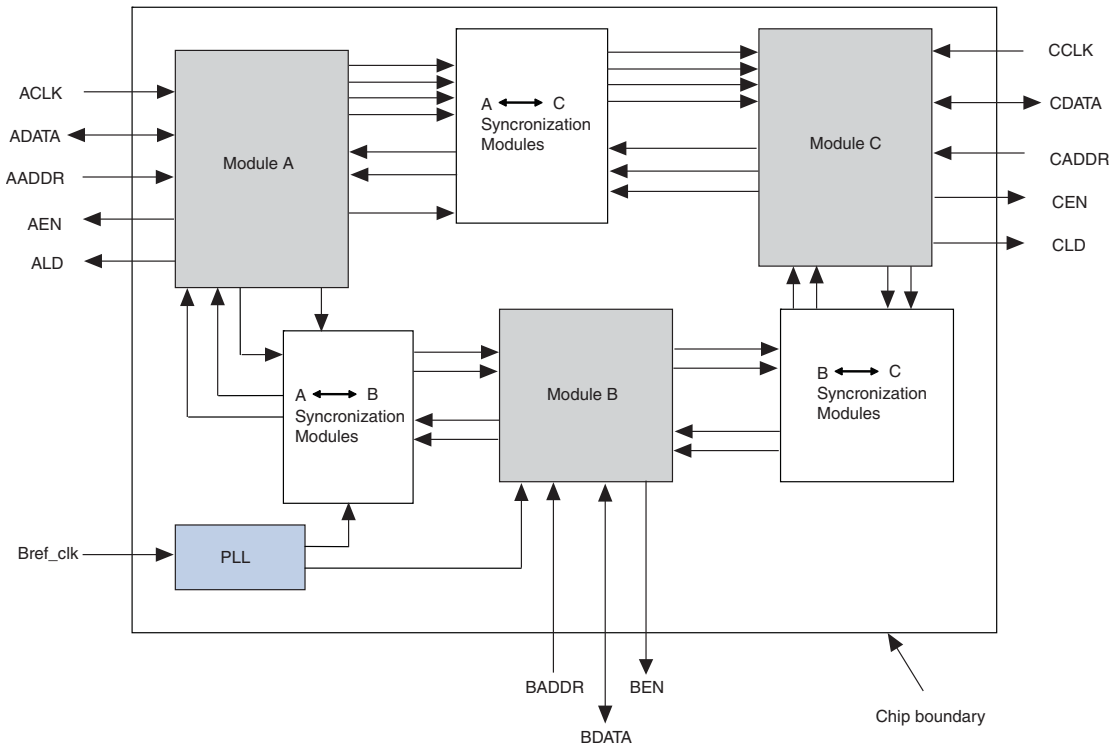
Timing Closure Techniques for Designs Migrating to HardCopy ASICs

This section describes techniques for achieving timing closure for your design migrating to HardCopy ASICs.

Design Partition and Signal Naming Convention

To make timing closure easier, use proper design partitioning. You can make timing closure simpler on single-clock modules. No matter how many clock domains you have in your design, by properly partitioning your design, it is possible to have only one clock per module. For signals crossing clock domains, separate synchronization module(s) are created, as shown in Figure 25.

Figure 25. Design Partitioning



In Figure 25, synchronization modules can be either synchronizers or FIFOs. You need a synchronizer or FIFO for every clock transfer between clock domains. For example, one synchronizer for the transfer from the ACLK to BCLK domain and one synchronizer for the transfer from the BCLK to ACLK domain. All modules have only one clock except the synchronization modules, which have multiple clocks.

In this methodology, you can easily employ a signal naming convention. All signals that do not interact with other clock domains are named so. For example, an ald signal only interacts in the aclk domain.

All signals that cross clock domains are also properly named. For example, a signal that crosses the clock domain ACLK to BCLK is named a2b_dat. An ack signal that crosses from the domain BCLK to ACLK is named b2a_ack, and so on. Timing analysis is simpler with this type of signal naming convention. All signals that cross clock domains are easily identified and can have false paths (or clock groups) set for timing analysis without trouble.

In addition to making timing closure easier, the advantages of design partitioning are better floor planning for synthesis reduction in area, performance, and potentially lower power. The synthesis tool or engine is able to do a better job because there is only one clock in the module and the design can be easily optimized for area, speed, or power.

Incremental Compilation and Floorplanning

As described in the previous section, if your design is well partitioned, you can use incremental compilation to optimize your design to meet your system performance requirements. Incremental compilation combined with floorplanning is a very powerful feature that can assist you in achieving your performance goals.

To use the full benefits of incremental compilation, organize your design so that the hierarchy and RTL code is easily partitioned along logical hierarchy boundaries. Within the Quartus II software, create design partitions to specify which blocks will be compiled independently as partitions. After compilation, the Quartus II Analysis and Synthesis and the Fitter engines create separate netlists for each partition. For example, after Analysis and Synthesis, the netlist is a post-synthesis netlist and after the Fitter, the netlist is a post-fit netlist. Select which netlist type to preserve for each partition during subsequent compilation. If RTL changes are required for a particular partition in the next compile, only that partition is recompiled, reducing compilation time.

In addition to reduced compilation time, incremental compilation also preserves performance for unchanged design blocks and enables true team-based designs.

Incremental compilation has two flows: a top-down flow and a bottom-up flow. For designs migrating to HardCopy ASICs, only the top-down flow is supported.

Incremental compilation produces the best results when the design is properly floorplanned. A floorplan represents the layout of the physical resources on the device. In the Quartus II software, create LogicLock™ regions to constrain blocks of a design to a particular region of the device. A LogicLock region is a rectangular area in the device with a user-defined or Fitter-defined size and location on the device.

For more information about incremental compilation, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* and *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapters in volume 1 of the *Quartus II Handbook*.

You may ask, if my design is well-partitioned, why do I need to floorplan the design as well? Design partitions are logical entities based on the design hierarchy, whereas LogicLock regions are physical placement assignments that constrain logic to a rectangular region on the device. In the absence of LogicLock assignments, the Fitter places the logic anywhere on the device. To control the placement of the logic from a design partition and isolate it to a particular region in the device, you must assign logical design partitions to a physical region in the device floorplan using LogicLock assignments.

Floorplan location planning is very important to ensure good quality results when compiling a design that migrates to a HardCopy ASIC using the full top-down incremental compilation flow. Altera recommends creating a floorplan for timing-critical partitions. However, you can choose not to implement floorplan assignments for partitions that are not timing-critical.

For more information about floorplans and LogicLock regions, refer to the *Analyzing and Optimizing the Design Floorplan* chapter in the volume 2 of the *Quartus II Handbook*.

HardCopy Performance Improvement

Altera's HardCopy ASIC migration flow allows you to gain significant performance improvement over the FPGA due to its unique architecture and die-size reduction. In fact, some customers have achieved twice the performance than the equivalent FPGA with a significant reduction in power. Because the FPGA is a prototype vehicle and the HardCopy ASIC is a production vehicle, many designers choose a low-speed FPGA for validating the RTL and, after FPGA validation, go to production with a HardCopy ASIC at the desired performance target.

Performance improvement is design dependent.

Performance improvement over the FPGA brings an added complexity to the timing constraints. The base clocks may need to be different, as do the PLL settings. The Quartus II software, together with the TimeQuest Timing Analyzer engine, makes constraining your design easy. The following section describes how you can have a single **.sdc** to analyze timing for the low-speed FPGA and HardCopy ASIC running at your target performance requirement. It also describes how you can set the PLL settings differently between the FPGA and the HardCopy ASIC. Finally, this section describes when you cannot achieve performance improvement in the HardCopy ASIC.

Constraints for Performance Improvement

The following is a scenario that may be possible in your design. Assume a reference clock (`ref_clk`) must run at 10 ns in the FPGA and 5 ns in the HardCopy ASIC. The reference clock (`ref_clk`) feeds a PLL, which then generates the required clocks running at 100 MHz in the FPGA and 200 MHz in the HardCopy ASIC. In this case, Example 12 shows the constraints you would have in your **.sdc**.

Example 12. .sdc Constraints for Performance Improvement

```
if { $::TimeQuestInfo(family) == "HardCopy II"} {
# The base clock runs faster in the HC revision
create_clock -name ref_clk -period 5 -waveform {0 2.5}
}
else {
# FPGA clock runs slower
create_clock -name ref_clk -period 10 -waveform {0 5}
}
derive_pll_clocks
```

In the preceding constraint example, the `M` and `N` counter values and the phase offsets of the PLL are assumed to be identical.

Performance improvement almost always requires the instantiated PLLs in the FPGA and HardCopy ASICs to operate using different settings. You can easily accommodate different settings for the FPGA and HardCopy ASIC by manipulating the PLLs' `M` and `N` counter values. However, be aware that the revision compare can fail because the tool detects different PLL settings in the FPGA and HardCopy devices.

If you use the guidelines outlined in *AN432: Using Different PLL Settings between Stratix II and HardCopy II Devices*, your design can have a significant performance improvement in the HardCopy ASIC and also achieve a successful HardCopy Revision Comparison report.

Performance Improvement Expectations and Guidance

HardCopy ASIC architecture is similar to the FPGA architecture in the areas of I/O and hard IP blocks, such as M4Ks, MRAM, and DSP hard macros. The I/O performance of the HardCopy ASIC is similar to the FPGA performance. Also, hard IP blocks, such as M4Ks, MRAM, and DSP hard macros, are similar to the FPGA performance. Therefore, no performance improvement in these areas is expected.

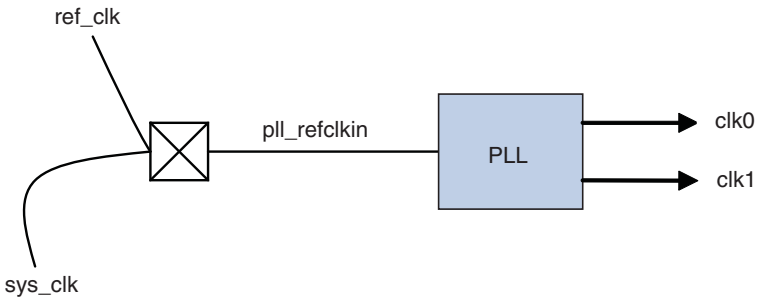
Also, performance improvement is design dependent. For example, a design with many carry chains does not see significant performance improvement because the carry chain logic array block (LAB) implementation is already highly optimized and efficient in the FPGA and HardCopy ASIC—there is nothing left to improve. In this instance, the only area where performance improvement can be expected is the core logic. Core logic improvement comes mostly from faster interconnect delays, not from cell delay improvements.

Multi-Frequency Analysis

Altera and Altera's customers are increasingly creative when it comes to using the HardCopy ASIC as a production vehicle. One customer designed a HardCopy ASIC that went into production on two different systems. Depending on which system the board plugged into, the firmware powered-up differently—the PLL settings changed by the boot code using the PLL reconfig megafunction—one board powered up at 200 Mhz, and another powered up at 250 Mhz. The customer wanted timing analysis to be run at both frequencies and sign-off was based on both operating frequencies. This was accomplished using the technique described in the previous section. The following section describes two other techniques that you can use to obtain the same results.

Figure 26 shows two base clocks to a reference clock pin of a PLL

Figure 26. Two Base Clocks to a Reference Clock Pin of a PLL



In this situation, define the two base clocks to a PLL input clock using the following constraints:

```
create_clock -name ref_clk -period 10 -waveform {0 5}
[get_ports pll_refclkin]
```

```
create_clock -name sys_clk -period 8 -waveform {0 4} [get_ports
pll_refclkin] -add
```

Because the two base clocks are being defined to the same input pin (`pll_refclkin`) of the PLL, you cannot use the `derive_pll_clocks` constraint because the `derive_pll_clocks` macro cannot determine what the master clock is for the generated clocks `clk0`, `clk1`, and so on.

Therefore, instead of using the `derive_pll_clocks` macro, use the `create_generated_clock` constraint for each of the PLL-generated clocks for both the base clocks with the `-add` option shown in Example 13.

Example 13. create_generated_clock Constraint (with the -add Option)

```
#Generated Clocks with ref_clk as the base clock

create_generated_clock -name {ref_clk_gen0} -master_clock
{ref_clk} -source [get_pins
{inst_ref|inst_refclk|ref_clk_pll_inst|altpll_component|pll|inc
lk[0]}] -multiply_by 1 -duty_cycle 50.000 [get_pins
{inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|clk[0]}] -add

create_generated_clock -name {ref_clk_gen1} -master_clock
{ref_clk} -source [get_pins
{inst_ref|inst_refclk|ref_clk_pll_inst|altpll_component|pll|inc
lk[0]}] -multiply_by 1 -duty_cycle 50.000 -phase 117.000
[get_pins {inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|clk[1]}] -add

#Generated Clocks with sys_clk as the base clock

create_generated_clock -name {sys_clk_gen0} -master_clock
{sys_clk} -source [get_pins {
inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|inclk[0]}] -multiply_by 1 -duty_cycle
50.000 [get_pins { inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|clk[0]}] -add

create_generated_clock -name {sys_clk_gen1} -master_clock
{sys_clk} -source [get_pins {
inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|inclk[0]}] -multiply_by 1 -duty_cycle
50.000 -phase 90.000 [get_pins {
inst_ref|inst_refclk|ref_clk_pll_inst
|altpll_component|pll|clk[1]}] -add
```

In the preceding example, to minimize user error for the PLL-generated clocks within the TimeQuest Timing Analyzer GUI, define one of the base clocks using the `derive_pll_clocks` constraint and use the `write_sdc -expand` command to get the generated clocks and repeat the step for the other base clock.

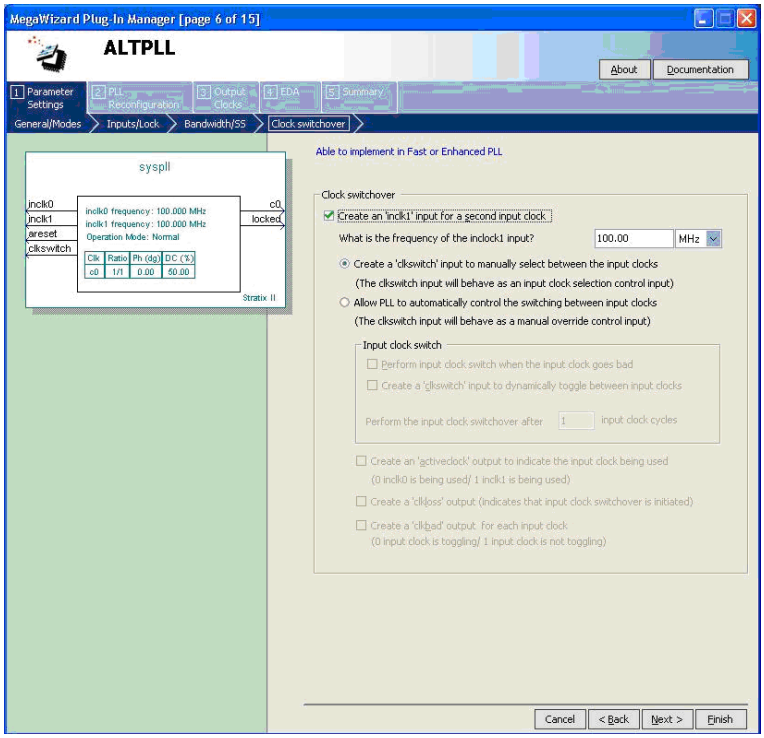
If the PLL settings are different between the two base clocks, use the guidelines outlined in *AN432: Using Different PLL Settings between Stratix II and HardCopy II Devices*.

Because the two base clocks are unrelated, cut the timing paths between the different base clocks and the corresponding generated clocks by using the `set_clock_groups -exclusive` constraint.

Using the Clock Switchover Feature in the ALTPLL Megafunction

Use the clock switchover feature in the ALTPLL megafunction to meet performance requirements on systems that operate at different frequencies. Figure 27 shows the ALTPLL megafunction with the clock switchover feature enabled.

Figure 27. ALTPLL Megafunction with Clock Switchover Enabled



The reference clock of the PLL can be either `inclk0` or `inclk1` and is selected by the `clkswitch` signal. Depending on your board and system requirements, you can assert the `clkswitch` signal by the bootcode. With this feature, you do not need explicitly generated clocks on the outputs of the PLL. The two base clocks are defined and using the `derive_pll_clocks` macro takes care of the generated clocks. The only constraints that you must add are the set of timing paths that have to be cut using the `set_clock_groups` constraint.

Example 14 shows the set of constraints.

Example 14. Additional set_clock_groups Constraints

```
create_clock -name ref_clk -period 10 -waveform {0 5[get_pins {
inst_ref|inst_refclk|ref_clk_pll_inst |altpll_component|pll|inclk[0]}]

create_clock -name sys_clk -period 8 -waveform {0 4} [get_pins {
inst_ref|inst_refclk|ref_clk_pll_inst |altpll_component|pll|inclk[1]}]

derive_pll_clocks
```

For more information about the ALTPLL megafunction, refer to the *ALTPLL Megafunction User Guide*.

Quartus II Software Optimizations

Migrating from the FPGA to the HardCopy ASIC includes mapping ALM logic blocks to Hcell macros. The FPGA hold time requirements are guaranteed by design. In the HardCopy ASIC, due to different clock structures and Hcell macros that are faster than the equivalent ALM logic blocks, the Quartus II software or the HardCopy Design Center (HCDC) back-end tools may have to insert buffers to guarantee hold time requirements are met for all operating conditions and process variations.

In general, the Quartus II Fitter must be set to **Optimize Fast Corner Timing** and **Fix Hold Time for All Paths**. However, for some hard-to-fit designs that are close to or just out of routing resources, you can set the Quartus II Fitter to NOT **Optimize Fast Corner Timing**. You can also try the **Fix Hold Time for I/O and Tpd paths**.

Contact the HardCopy Design Center before using these settings to ensure that the back-end tools are able to close hold time requirements. For other techniques to fit difficult-to-fit designs, refer to *AN 453: HardCopy II Fitting Techniques*.

Figure 28 shows the Quartus II Fitter set to meet the hold time requirements.

Figure 28. Quartus II Fitter with Hold Time Requirements Set

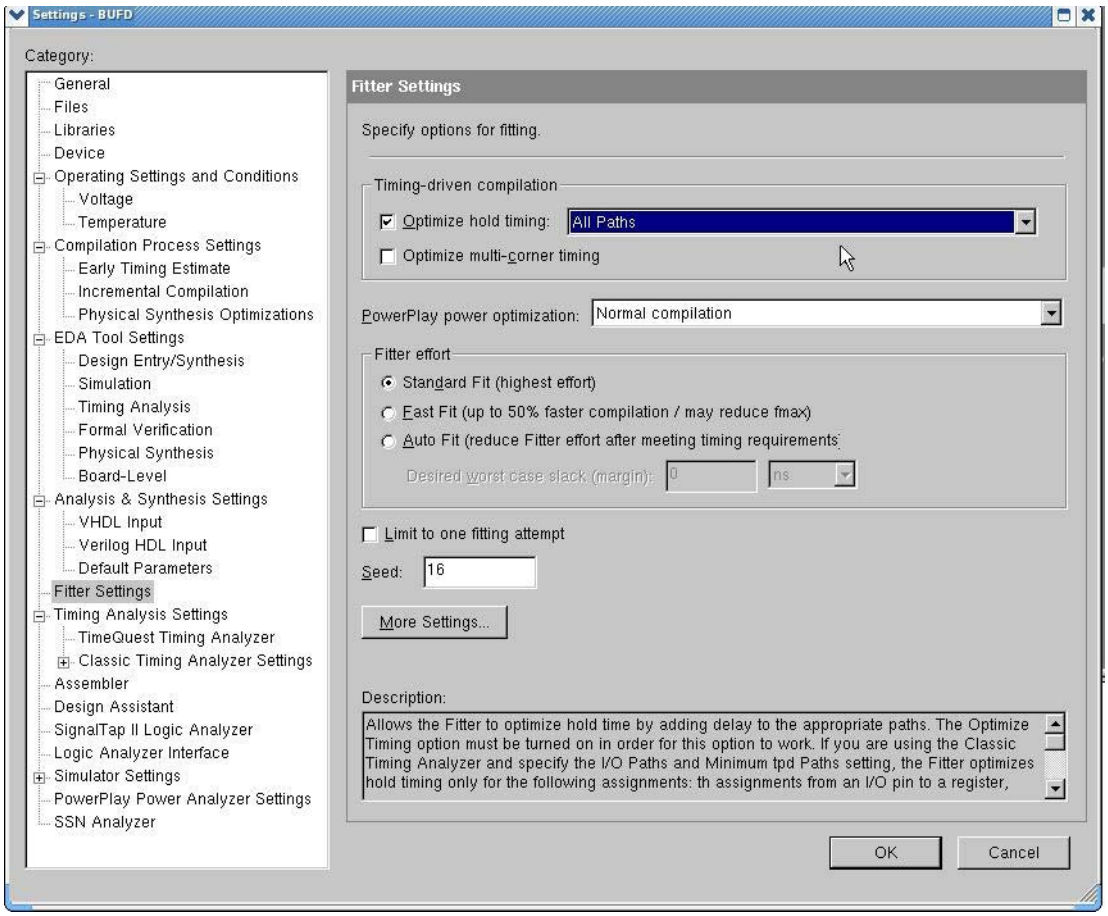
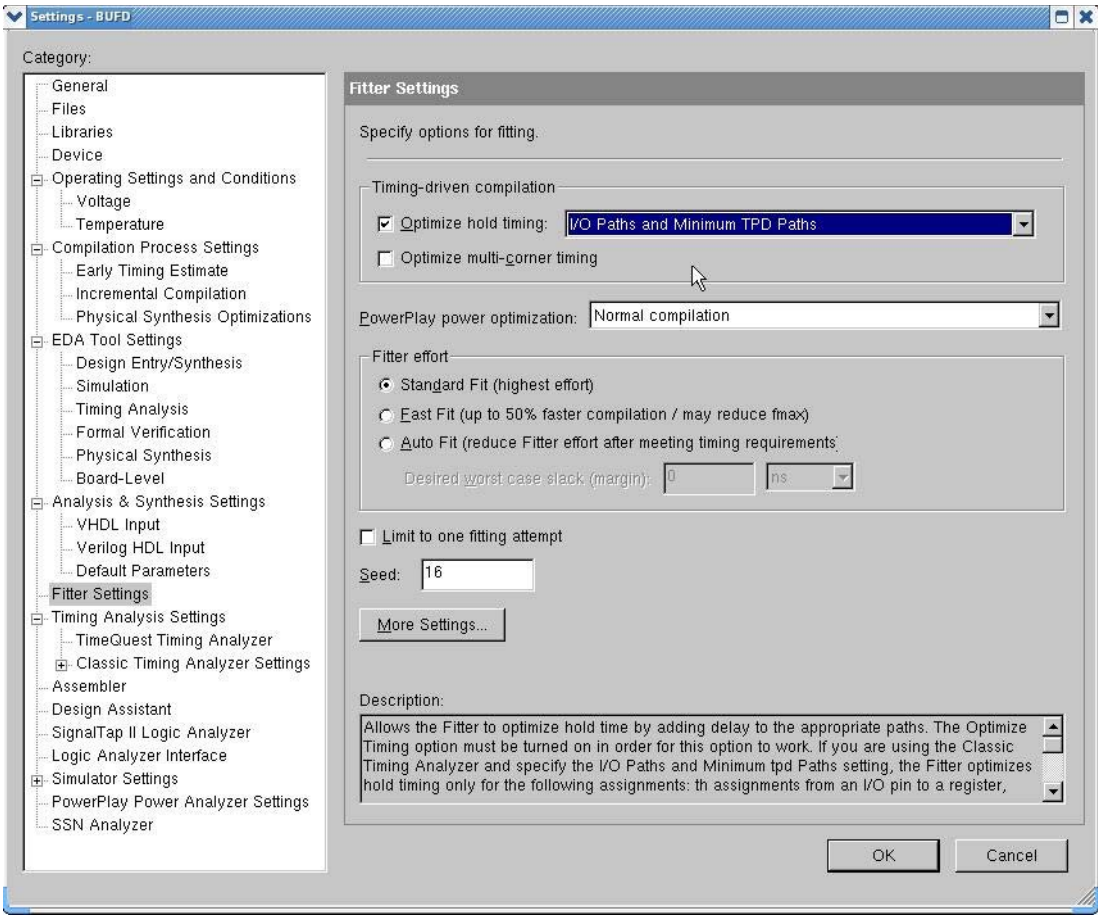


Figure 29 shows the Quartus II Fitter set to let the HardCopy Design Center back-end tools perform the optimizations necessary to meet the hold time requirements.

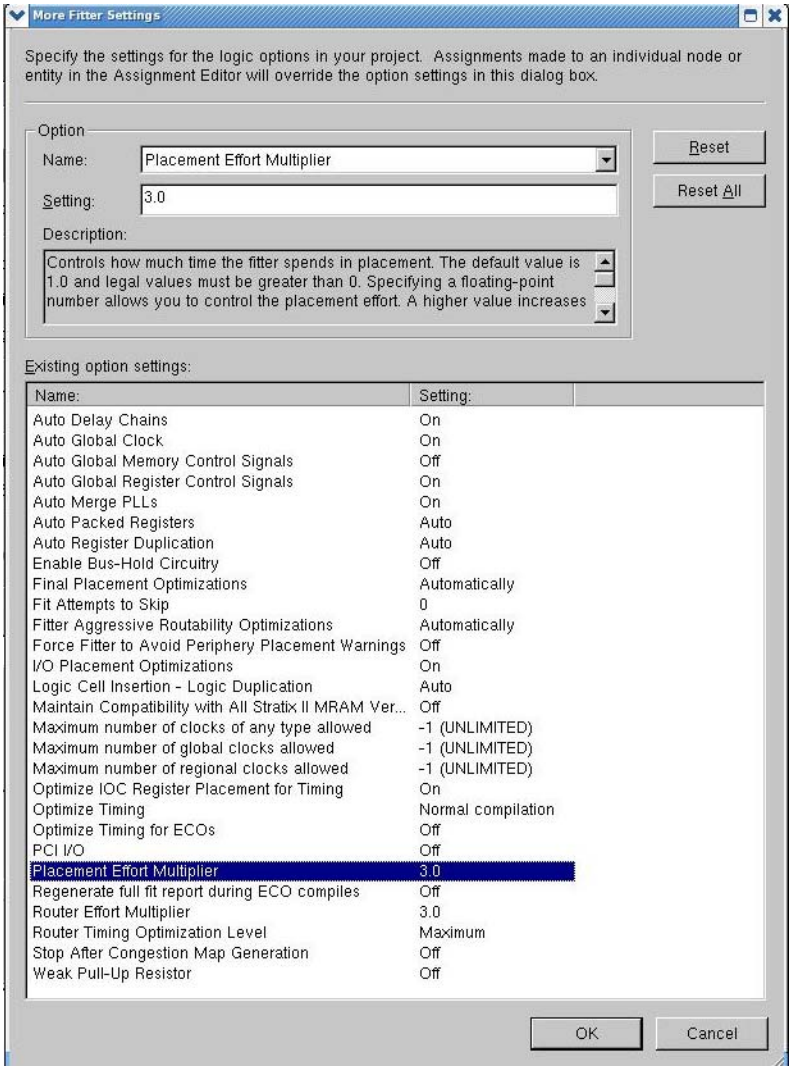
Figure 29. Quartus II Fitter Settings for Fixing the Hold Time Requirements for the I/O Paths and Minimum TPD Paths Only



Another useful setting for getting a hard-to-route design to fit is the **Placement Effort Multiplier** setting. Increasing the multiplier to as much as **3.0** can produce a better starting placement and improve routing.

Figure 30 shows **Placement Effort Multiplier** set to **3.0**.

Figure 30. Quartus II Fitter with the Placement Effort Multiplier Set to 3.0



There is also a **Routing Effort Multiplier** setting. Increasing this setting increases compile time and does not significantly improve routing in HardCopy designs. Therefore, Altera recommends that you leave this setting at **1.0**.

For more information about the Quartus II Optimization settings, refer to the Area, Timing and Power Optimization section in volume 2 of the *Quartus II Handbook*.

Timing Closure for Interfaces

The following sections describe timing closure for interfaces.

External Memory Interface

Altera FPGAs and HardCopy ASICs support a wide variety of external memory interfaces, including DDR, DDR2, and DDR3. Altera also offers reference designs for some variations of RLDRAM and QDR SRAM. Altera recommends using the ALTMEMPHY megafunction for external memory interfaces. For instances with lower frequency requirements and lower latency requirements, you can use the Legacy Static PHY. Contact your local Field Applications Engineer for more information about which PHY is more suitable for your design.

Information about external memory interfaces is available in the following user guides and application notes:

- *AN 413: Using Legacy Integrated Static Data Path and Controller Megafunction with HardCopy II Structured ASICs*
- *AN 463: Using ALTMEMPHY Megafunction with HardCopy II Structured ASICs*
- *AN 325: Interfacing RLDRAM II with Stratix II, Stratix & Stratix GX devices*
- *AN 326: Interfacing QDRII+ & QDRII with Stratix II, Stratix II GX, Stratix, & Stratix GX Devices*
- *RLDRAM II Controller MegaCore Function User Guide*
- *QDRII SRAM Controller MegaCore Function User Guide*

Source Synchronous Interfaces

Consider the following when using a source synchronous interface. You must consider whether to use edge-aligned or center-aligned data:

- On the receiving side of a center-aligned interface, you can use the clock directly to capture data.
- On the receiving side of an edge-aligned interface, you can use a PLL to shift the clock 90°.
- On the sending side of an edge-aligned interface, you can use one clock for clocking out both the clock signal and data signals.
- On the sending side of a center-aligned interface, you can use separate PLL taps to shift the output clock by 90°.

Altera recommends using the **Source Synchronous Mode** feature for the PLL on the receiving side of the interface. This ensures that the clock and datapaths maintain their alignment from the pins of the device to the capturing register(s).

For more information about timing source synchronous interfaces, refer to *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

For information about using an RGMII source synchronous interface, refer to *AN 477: Designing RGMII Interface with FPGA and HardCopy Devices*.

Appendix A: Design Example

Appendix A describes an design example called “demo_design”. The design reviews the steps that you must complete for the design to be considered “HardCopy Ready”. HardCopy Ready is the term used for a design that has passed all the necessary steps for the HardCopy Design Center handoff. The design example is small and can be compiled for both FPGA and HardCopy devices in less than 15 minutes.

After completing this exercise, you will be able to:

- Be proficient in ASIC-style timing constraint creation
- Understand how to use the HardCopy Advisor and make appropriate changes to complete all the necessary steps for HardCopy Ready status
- Identify missing or incorrect timing constraints and apply them to the design
- Review the various reports (Synthesis, Fitter, and Timing reports) and identify logic structures that are not acceptable for HardCopy ASICs
- Review I/O usage from the Fitter and identify I/O usage not acceptable for HardCopy ASICs
- Review Design Assistant warnings and recommend changes or actions
- Apply clock uncertainty constraints using the `derive_clock_uncertainty` constraint

Things to consider before you begin:

- Some timing constraints are incorrect or missing in the archive provided. Review the system specification to understand the timing requirements. Add or change the timing requirements to the appropriate ASIC-style constraint
- Use the TimeQuest Timing Analyzer. The timing constraints are in **.sdc** format
- Do not add more false paths to the design. Assume that any timing path not cut is legitimate
- Design Assistant warnings will be of the category that you must review and waive. You do not have to change any RTL to fix Design Assistant warnings

The design archive is called *demo_design_original***.qar**. Restore this archive file with the Quartus II software version 7.1. The migration path is an EP2S30F484C4 FPGA to an HC210F484C HardCopy ASIC.

In the “Sample HardCopy Design Review Document” on page 74 (also known as the HardCopy Worksheet), log your findings on the design. If a step is missing, proceed with running that step and record in your Worksheet what step you ran to progress the design to HardCopy Ready status. The following are some steps that you may have to take:

1. Use the system-level timing budget block diagram in Figure 31 on page 58 and Table 9 on page 59 to derive the input and output minimum and maximum delay.
2. Unarchive the *demo_design_original***.qar** file using the Quartus II design software.

3. Review the **.sdc**s for the FPGA device and HardCopy II ASIC. Compare the timing numbers in your Worksheet and make corrections in **demo_design_constraints.sdc** and **demo_design_cu.sdc**.
   a. Note which constraints are missing and add them. You can compile the design and review the Timing report to look for unconstrained paths.
   b. Note the use of the `derive_pll_clocks` command. Use this command instead of the `create_generated_clocks` command, because the `derive_pll_clocks` command avoids errors (typically typos) when generating the clock commands.
   c. The `clkb_o` source synchronous output clock does not need a generated clock to check timing on that interface.
4. Review clock uncertainty to the timing constraints in the **demo_design_cu.sdc** file.
5. Identify issues and make the appropriate changes to make the design HardCopy Ready. Record these changes on the HardCopy Worksheet.

   ☞ If there are problems in the FPGA that stops it from compiling in HardCopy II ASICs, identify the problems, correct them, and run the HardCopy II compilation. When complete, identify all information requested on the HardCopy Worksheet and create a list of what you need to change. Use the HardCopy II Advisor to identify what steps were not run and correct the settings as needed. Use the HardCopy II Advisor to launch subsequent runs of the compiler and various steps, as needed, to familiarize yourself with this tool. Carefully document the changes you made to the Sample Design Review Document.

6. For actual customer designs, Design Assistant warnings are either waived or RTL modifications are made. In general, the designer best understands the intended operation of the circuitry and can make the best judgments on what design structures can cause different operations of the FPGA devices versus the HardCopy ASIC. For this exercise, you can wave the reset and synchronization type Design Assistant warnings.
7. BONUS SECTION: Altera recommends the `alt_pll_reconfig` PLL wrapper for HardCopy ASIC designs. Add this wrapper around your PLLs. For this exercise, bond the pins out of these blocks (there are two PLLs) to the primary I/O. Normally, customers map the PLL reconfiguration register space into their CPU interface configuration registers (or whatever means they use at the system level to configure their design).
8. BONUS SECTION: You will find that the FPGA device does not meet performance. This makes the design a candidate for HardCopy II performance improvement. To allow the FPGA device and HardCopy II ASIC to meet timing, you can slow the clock down for the FPGA device and keep the HardCopy II clock speed the same, but maintain two different PLL settings.

   The reference for this flow is in *AN 432: Using Different PLL Settings Between Stratix II and HardCopy II Devices*.

Figure 31. Demo Design Block Diagram

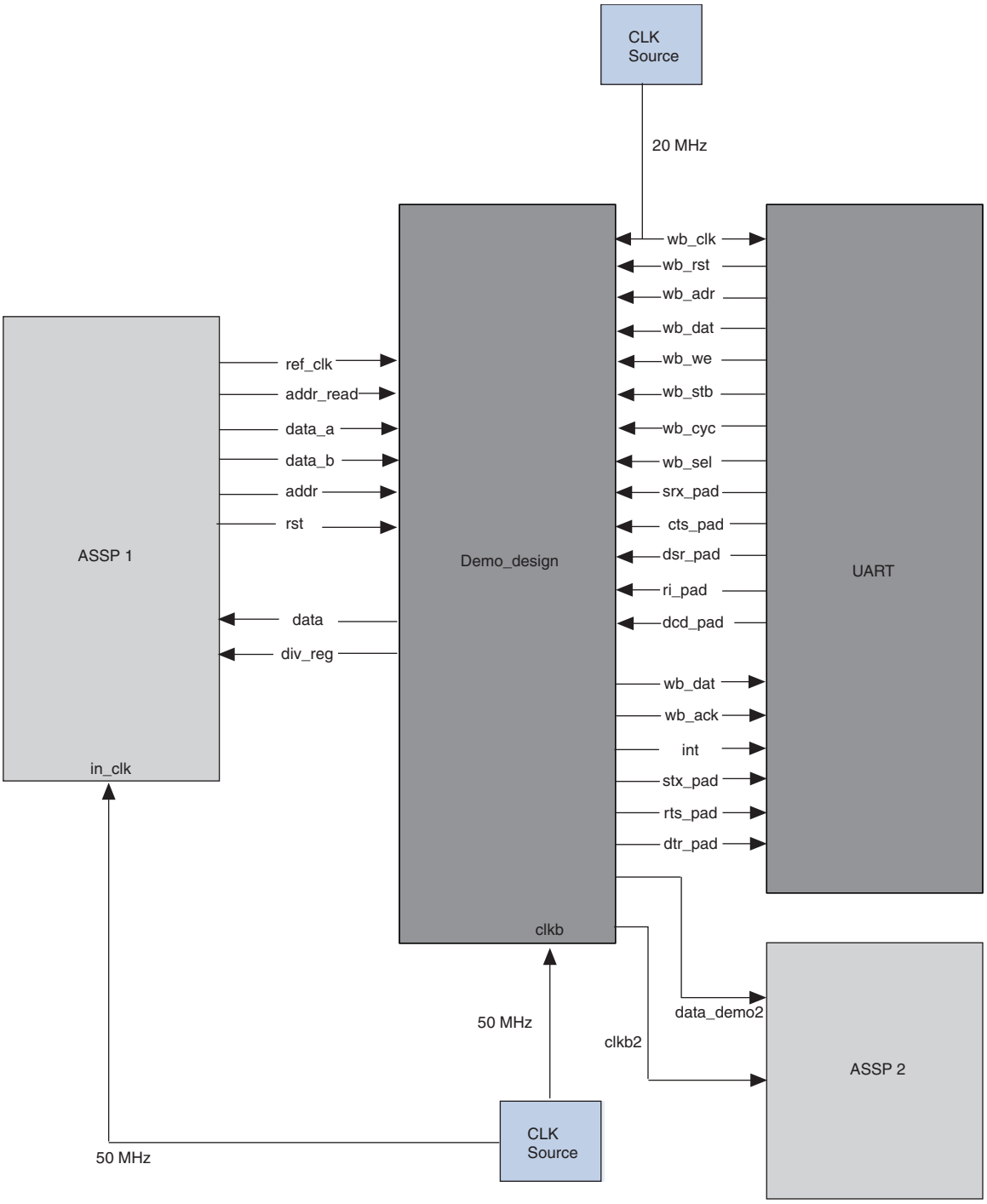


Table 9. System Specification *(Note 1), (2), (3), (4)*

Timing Specifications for the Interfaces				FPGA and HardCopy II Timing Specification SDC Constraint					
		t_{SU}	t_H	T_{CO} Max +0.5 ns	T_{CO} Min -0.5 ns	Input Max	Input Min	Output Max	Output Min
ASSP1 Timing Spec in clk (50 MHz) > ref_clk									
in_clk	addr			6 ns	2 ns				
in_clk	addr_read			13 ns	3 ns				
in_clk	data_a			8 ns	2 ns				
in_clk	data_b			8 ns	2 ns				
in_clk	resetn			3 ns	2 ns				
in_clk	div_reg	7 ns	0 ns					8.75 ns	1.25 ns
in_clk	data	5 ns	1 ns					10.75 ns	2.2 ns
ASSP2 Timing Spec in clkb (100 MHz)									
clkb2	data_demo	4 ns	1 ns					5.25	-0.25 ns
UART Timing Spec in wb_clk (20 MHz)									
wb_clk	wb_rst			8 ns	2 ns				
wb_clk	wb_adr			8 ns	2 ns				
wb_clk	wb_dat(in)			8 ns	2 ns				
wb_clk	wb_we			8 ns	2 ns				
wb_clk	wb_stb			8 ns	2 ns				
wb_clk	wb_cyc			8 ns	2 ns				
wb_clk	wb_sel			8 ns	2 ns				
wb_clk	srx_pad			8 ns	2 ns				
wb_clk	cts_pad			8 ns	2 ns				
wb_clk	dsr_pad			8 ns	2 ns				
wb_clk	ri_pad			8 ns	2 ns				
wb_clk	dcd_pad			8 ns	2 ns				
wb_clk	wb_dat(out)	8 ns	2 ns						
wb_clk	wb_ack	8 ns	2 ns						
wb_clk	int	8 ns	2 ns						
wb_clk	stx_pad	7 ns	0 ns						
wb_clk	rts_pad	7 ns	0 ns						
wb_clk	dtr_pad	7 ns	-1 ns						

Notes to Table 9:

(1) All clock signals have 1 ns board delay from source to destination.

(2) All data signals have 2 +/- 0.25 ns board delay from source to destination.

(3) ASSP1 has 15 pF of pin load, ASSP2 has 25 pF of pin load, and UART has 11 pF of pin load.

(4) Board trace cap for data signals is 10 pf; Board trace cap for clock signals is 5 pf.

The source equations for Table 9 are as follows:

- `set_input_delay_max = bd_max + tco_max - clock_min(ext -> FPGA)`
- `set_input_delay_min = bd_min + tco_min - clock _max(ext -> FPGA)`
- `set_output_delay_max = bd_max + Tsu - clock_min(FPGA -> ext)`
- `set_output_delay_min = bd_min - Th - clock_max(FPGA -> ext)`

Demo Design Walkthrough

The following sections describe the demo design process.

Step 1: System Timing Requirements

The first step is to review the block diagram in Figure 31 on page 58 and the timing specifications in Table 9 on page 59. You must complete the timing specification table. Start by filling in Table 9 on page 59 with the input and output delay values for setup and hold.

ASSP1 > FPGA (or HCII)

Input delays:

- `set_input_delaymax = boardmax + Tcomax - clockmin (external device > FPGA [or HCII)])`
- `set_input_delaymin = boardmin + Tcomin - clockmax (external device > FPGA [or HCII])`

Board Delay:

In Table 9 on page 59, the board delay specified in row two is as follows:

All data signals have 2 +/- 0.25 ns board delay from source to destination

This means that Boardmax = 2.25 ns and Boardmin = 1.75 ns

Clock Delay:

For setup, consider the earliest clock; for hold, consider the latest clock.

The board has a delay of 1 ns for the clock. The T_{COMAX} for `ref_clk` is 0.5 ns; the T_{COMIN} for `ref_clk` is -0.5 ns.

Therefore, Clockmin = 1.0 - 0.5 = 0.5 ns and Clockmax = 1.0 + 0.5 = 1.5 ns.

Apply the following to the preceding equations:

1. For the addr pin:

   set_input_delaymax = 2.25 + 6 - 0.5 = 7.75 ns

   set_input_delaymin = 1.75 + 2 - 1.5 = 2.25 ns

   Enter the results in Table 9 on page 59 for the addr pins.

   The reference clock for the addr pins is ref_clk; therefore, create a virtual clock for this source-synchronous input.

   The constraint for this input is as follows:

   ```
   set_input_delay -max -clock virtual_ref_clk_i 7.75 [get_ports {addr_i*}]
   ```

   ```
   set_input_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {addr_i*}]
   ```

2. For the addr_read pin:

   set_input_delay(max) = 2.25 + 13 -0.5 = 14.75 ns

   set_input_delay(min) = 1.75 + 3 - 1.5 = 3.25 ns

   Enter the results in Table 9 on page 59 for the addr_read pin.

   The constraints are as follows:

   ```
   set_input_delay -max -clock virtual_ref_clk_i 14.75 [get_ports {addr_read_i*}]
   ```

   ```
   set_input_delay -min -clock virtual_ref_clk_i 3.25 [get_ports {addr_read_i*}]
   ```

   Calculate the values for the input delays for the rest of the input pins on this interface.

Output Delays

```
set_output_delaymax = boardmax + Tsu -clockmin(FPGA [or HCII] > external device)
```

```
set_output_delaymin = boardmin - Th - clockmax(FPGA [or HCII] > external device)
```

Board Delay

In Table 9 on page 59, the board delay specified in row two is as follows:

- All data signals have 2 +/- 0.25 ns board delay from source to destination
- This means that Boardmax = 2.25 ns and Boardmin = 1.75 ns

Clock Delay

The clock delay for the outputs can be difficult. In Figure 31 on page 58, the outputs from the FPGA to ASSP1, data, and div_reg, have no reference clock. These outputs are referenced with respect to ref_clk. The question remains, what should the values for Clockmax and Clockmin be set to?

The answer is simple. For setup, consider the worst-case arrival clock; for hold, consider the best-case arrival of clock.

Clockmin = 1 - (-0.5) = 1.5 ns

Clockmax = 1 - 0.5 = 0.5 ns

`ref_clk` is from ASSP1 to FPGA (or HCII); therefore, the output delay equation is changed as follows:

- For the data signal:

```
set_output_delaymax = boardmax + Tsu + clockmin  = 2.25 + 5 +
1.5 = 8.75 ns

set_output_delaymin = boardmin - Th + clockmax = 1.75 - 1.0 +
0.5 = 1.25 ns
```

- For `div_reg`:

```
set_output_delaymax = boardmax + Tsu + clockmin  = 2.25 + 7 +
1.5 = 10.75 ns

set_output_delaymin = boardmin - Th -clockmax = 1.75 - 0.0 +
0.5 = 2.25 ns
```

Enter the above values in Table 9 on page 59.

The constraints for the output pins are as follows:

```
set_output_delay -max -clock virtual_ref_clk_i 8.75
[get_ports {data_o*}]

set_output_delay -min -clock virtual_ref_clk_i 1.25
[get_ports {data_o*}]

set_output_delay -max -clock virtual_ref_clk_i 10.75
[get_ports {div_reg_o*}]

set_output_delay -min -clock virtual_ref_clk_i 2.25
[get_ports {div_reg_o*}]
```

FPGA (or HCII) > ASSP2

Again, the out delay equation is as follows:

```
set_output_delaymax = boardmax + Tsu - clockmin(FPGA [or
HCII] > external device)

set_output_delaymin = boardmin - Th - clockmax(FPGA [or HCII]
> external device)
```

Board Delay

In Table 9 on page 59, the board delay specified in row two is as follows:

- All data signals have 2 +/- 0.25 ns board delay from source to destination
- This means that Boardmax = 2.25 ns and Boardmin = 1.75 ns

Clock Delay

For setup, consider the earliest clock; for hold, consider the latest clock.

The board has a delay of 1 ns for the clock.

Therefore, Clockmin = 1.0 = 0.5 ns and Clockmax = 1.0 ns.

```
set_output_delaymax = boardmax + Tsu + clockmin  = 2.25 + 4 -
1.0 = 5.25 ns
```

```
set_output_delaymin = boardmin - Th + clockmax = 1.75 - 1.0 -
1.0 = -0.25 ns
```

Enter the results in Table 9 on page 59.

The constraints for the output pins are as follows:

```
set_output_delay -max -clock clkb_o 5.25 [get_ports
{data_demo2_o*}]
```

```
set_output_delay -min -clock clkb_o -0.25 [get_ports
{data_demo2_o*}]
```

Calculate the input and output delays for the rest of the pins and enter them in Table 9 on page 59. Create the command for the constraints as you calculate them.

Table 10 lists the calculated values. Cross-check your results with this table to confirm they are correct.

Table 10. Final Timing Constraints After System Timing Review *(Note 1)*, *(2)*, *(3)*, *(4)* (Part 1 of 2)

Timing Specifications for the Interfaces				FPGA and HardCopy II ASIC Timing Specification SDC Constraint					
		t_{SU}	t_H	T_{CO} Max +0.5 ns	T_{CO} Min -0.5 ns	Input Max	Input Min	Output Max	Output Min
ASSP1 Timing Spec in clk (50 MHz) > ref_clk									
in_clk	addr			6 ns	2 ns	7.7 ns	2.25 ns		
in_clk	addr_read			13 ns	3 ns	14.75 ns	3.25 ns		
in_clk	data_a			8 ns	2 ns	9.75 ns	2.25 ns		
in_clk	data_b			8 ns	2 ns	9.75 ns	2.25 ns		
in_clk	data	5 ns	1 ns					8.75 ns	1.25 ns
in_clk	div_reg	7 ns	0 ns					10.75 ns	2.25 ns
in_clk	resetn			3 ns	2 ns	4.75 ns	2.25 ns		
ASSP2 Timing Spec in clkb (100 MHz)									
clkb2	data_demo	4 ns	1 ns					5.25	-0.25 ns
UART Timing Spec in wb_clk (20 MHz)									
wb_clk	wb_rst			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_adr			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_dat(in)			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_we			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_stb			8 ns	2 ns	10.25 ns	3.75		

Table 10. Final Timing Constraints After System Timing Review *(Note 1)*, *(2)*, *(3)*, *(4)* (Part 2 of 2)

Timing Specifications for the Interfaces		FPGA and HardCopy II ASIC Timing Specification SDC Constraint							
		t_{SU}	t_H	T_{CO} Max +0.5 ns	T_{CO} Min -0.5 ns	Input Max	Input Min	Output Max	Output Min
wb_clk	wb_cyc			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_sel			8 ns	2 ns	10.25 ns	3.75		
wb_clk	srx_pad			8 ns	2 ns	10.25 ns	3.75		
wb_clk	cts_pad			8 ns	2 ns	10.25 ns	3.75		
wb_clk	dsr_pad			8 ns	2 ns	10.25 ns	3.75		
wb_clk	ri_pad			8 ns	2 ns	10.25 ns	3.75		
wb_clk	dcd_pad			8 ns	2 ns	10.25 ns	3.75		
wb_clk	wb_dat(out)	8 ns	2 ns					10.25 ns	-0.25 ns
wb_clk	wb_ack	8 ns	2 ns					10.25 ns	-0.25 ns
wb_clk	int	8 ns	2 ns					10.25 ns	-0.25 ns
wb_clk	stx_pad	7 ns	0 ns					9.25 ns	1.75 ns
wb_clk	rts_pad	7 ns	0 ns					9.25 ns	1.75 ns
wb_clk	dtr_pad	7 ns	-1 ns					9.25 ns	2.75 ns

Notes to Table 9:

(1) All clock signals have 1 ns board delay from source to destination.
(2) All data signals have 2 +/- 0.25 ns board delay from source to destination.
(3) ASSP1 has 15 pF of pin load, ASSP2 has 25 pF of pin load, and UART has 11 pF of pin load.
(4) Board trace cap for data signals is 10 pf; Board trace cap for clock signals is 5 pf.

Writing the .sdc

Now that you have finalized system timing requirements, start writing the **.sdc**. For more information about how to create the **.sdc**s, refer to "The .sdc" on page 40. Notice the comments embedded in the **.sdc**. Example 15 shows the **.sdc** for system timing requirements.

Example 15. .sdc for System Timing (Part 1 of 5)

```
demo_design_constraints.sdc:

# This is the system level timing requirements SDC file for Demo Design.

# This file has the system timing requirements for both the FPGA and HCII
revisions

# The Clock Uncertainty Constraints are applicable for the HCII revision

#

# File Created by: Altera Engineer

# Original Date Created: 03/05/2007
```

Example 15. .sdc for System Timing (Part 2 of 5)

```
# Revision History:
   1.3: 05/03/2007: Altera Engineer1
   #HCII CU constraints removed and put in a separate file
   #Final system timing requirements.
   #
   #1.2: 03/11/2007: Altera Engineer2
   #ASSP1 Timing requirements changed
   #Based on final system timing requirements after meeting on
   #03/10/2007
   #
   #1.1: 03/04/2007: ASSP2 and UART timing requirements changed.
   #1.0: 03/01/2007: Initial system timing requirements.
# ####################
# Clocks
# These are the actual clocks in the design associated with ports
######################
create_clock -period 20 -name ref_clk_i [get_ports {ref_clk_i}]
create_clock -period 20 -name clkb_i [get_ports {clkb_i}]
create_clock -period 50 -name wb_clk_i [get_ports {wb_clk_i}]
######################
#  Virtual Clocks
# These are virtual clocks used for associating the IO timing
# These clocks allow a different clock uncertainty for IO paths versus core
timing paths
########################
create_clock -period 20 -name virtual_ref_clk_i
create_clock -period 20 -name virtual_clkb_i
create_clock -period 50 -name virtual_wb_clk_i
###########
# PLL Clocks
############
derive_pll_clocks
##########################################
# Create Generated Clock for source sync output
```

Example 15. .sdc for System Timing (Part 3 of 5)

```
# This clock is needed for the source synchronous output - Hint: this is wrong
way to

# handle this interface

##############################################

create_generated_clock -name clkb_o -source [get_pins
{pll2_inst|altpll_component|pll|clk[1]}] [get_ports {clkb_o}]

######################

# False Paths

######################

set_false_path -from [get_ports clkb_i] -to [get_ports {clkb_o}]

set_false_path -from [get_clocks {pll1_inst|altpll_component|pll|clk[0]}] -to
[get_clocks {pll2_inst|altpll_component|pll|clk[0]}]

##############################

#  Input Delay

##############################

######################

# ASSP1 ' Demo Design

######################

# 03/11/2007: Changed from 7.00 to 7.75 based on system timing review

set_input_delay -max -clock virtual_ref_clk_i 7.75 [get_ports {addr_i*}]

# Initial requirement

set_input_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {addr_i*}]

# 03/11/2007: Changed from 11.25 to 14.75 based on system timing review

set_input_delay -max -clock virtual_ref_clk_i 14.75 [get_ports {addr_read_i*}]

# 03/11/2007: Changed from 0 to 3.25 after system timing review

set_input_delay -min -clock virtual_ref_clk_i 3.25 [get_ports {addr_read_i*}]
```

Example 15. .sdc for System Timing (Part 4 of 5)

```
set_input_delay -max -clock virtual_ref_clk_i 9.75 [get_ports {data_a_i*}]
set_input_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {data_a_i*}]
set_input_delay -max -clock virtual_ref_clk_i 9.75 [get_ports {data_b_i*}]
set_input_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {data_b_i*}]
set_input_delay -max -clock virtual_ref_clk_i 4.75 [get_ports {resetn_i}]
set_input_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {resetn_i}]
######################
# UART ' Demo Design
######################
# 03/04/3007: Changed after system timing review
set_input_delay -max -clock virtual_wb_clk_i 10.25 \
   [get_ports {wb_rst_i wb_adr_i* wb_dat_i* wb_we_i \
   wb_stb_i wb_cyc_i wb_sel_i* srx_pad_i cts_pad_i  \
   dsr_pad_i ri_pad_i dcd_pad_i}]
set_input_delay -min -clock virtual_wb_clk_i 3.75  \
   [get_ports {wb_rst_i wb_adr_i* wb_dat_i* wb_we_i \
   wb_stb_i wb_cyc_i wb_sel_i* srx_pad_i cts_pad_i  \
   dsr_pad_i ri_pad_i dcd_pad_i}]
#############################################################
# Output Delay
#############################################################
######################
# Demo Design ' ASSP1
######################
set_output_delay -max -clock virtual_ref_clk_i 10.75 [get_ports {div_reg_o*}]
set_output_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {div_reg_o*}]
set_output_delay -max -clock virtual_ref_clk_i 10.75 [get_ports {data_o*}]
set_output_delay -min -clock virtual_ref_clk_i 2.25 [get_ports {data_o*}]

######################
# Demo Design ' ASSP2
######################
```

Example 15. .sdc for System Timing (Part 5 of 5)

```
## Max = bd_max + Tsu - clock_min(Demo Design -> ext) = 2.25 + 4 - 1
set_output_delay -max -clock clkb_o 5.25 [get_ports {data_demo2_o*}]
# Min = bd_min - Th - clock_max(Demo Design ->ext) = 1.75 - 1 - 1
set_output_delay -min -clock clkb_o -0.25 [get_ports {data_demo2_o*}]
######################
# Demo Design ' UART
######################
set_output_delay -max -clock virtual_wb_clk_i 10.25 \
   [get_ports {wb_ack_o int_o wb_dat_o*}]
set_output_delay -min -clock virtual_wb_clk_i -0.25 \
   [get_ports {wb_ack_o int_o wb_dat_o*}]
set_output_delay -max -clock virtual_wb_clk_i 9.25 [get_ports {stx_pad_o
rts_pad_o}]
set_output_delay -min -clock virtual_wb_clk_i 1.75 [get_ports {stx_pad_o
rts_pad_o}]
set_output_delay -max -clock virtual_wb_clk_i 9.25 [get_ports {dtr_pad_o}]
set_output_delay -min -clock virtual_wb_clk_i 2.75 [get_ports {dtr_pad_o}]
########################
## Clock Uncertainty
########################
if { $::TimeQuestInfo(family) == "HardCopy II"} {
# HCII Specific constraints can be placed here
# For example
# Clock Uncertainty(CU) which is only needed in HCII
# FPGA Timing model is pessimistic so CU not needed - you can still model CU in
FPGA
# if you want to, just use the following constraint outside of this TCL
construct.
derive_clock_uncertainty
}
```

The `derive_clock_uncertainty` command automatically calculates the clock uncertainties and constraints. They are saved in: *<project dir>*/***hardcopy_output***/*<project_name>*_***hcii.constraints*.sdc.**

Example 16 shows the clock uncertainty constraints in this file.

Example 16. Clock Uncertainty Constraints (Part 1 of 2)

```
#************************************************************

# Set Clock Uncertainty

#************************************************************

set_clock_uncertainty -setup -from [get_clocks {wb_clk_i}] -to [get_clocks
{wb_clk_i}] 0.200

set_clock_uncertainty -hold -from [get_clocks {wb_clk_i}] -to [get_clocks
{wb_clk_i}] 0.050

set_clock_uncertainty -setup -from [get_clocks {wb_clk_i}] -to [get_clocks
{virtual_wb_clk_i}] 0.180

set_clock_uncertainty -hold -from [get_clocks {wb_clk_i}] -to [get_clocks
{virtual_wb_clk_i}] 0.180

set_clock_uncertainty -setup -from [get_clocks {virtual_wb_clk_i}] -to
[get_clocks {wb_clk_i}] 0.180

set_clock_uncertainty -hold -from [get_clocks {virtual_wb_clk_i}] -to
[get_clocks {wb_clk_i}] 0.180

set_clock_uncertainty -setup -from [get_clocks {virtual_ref_clk_i}] -to \

   [get_clocks {pll1_inst|altpll_component|pll|clk[0]}] 0.160

set_clock_uncertainty -hold -from [get_clocks {virtual_ref_clk_i}] -to \

   [get_clocks {pll1_inst|altpll_component|pll|clk[0]}] 0.180

set_clock_uncertainty -setup -from [get_clocks
{pll2_inst|altpll_component|pll|clk[0]}] -to \

   [get_clocks {pll2_inst|altpll_component|pll|clk[0]}] 0.120

set_clock_uncertainty -hold -from [get_clocks
{pll2_inst|altpll_component|pll|clk[0]}] -to \

    [get_clocks {pll2_inst|altpll_component|pll|clk[0]}] 0.050

set_clock_uncertainty -setup -from [get_clocks
{pll2_inst|altpll_component|pll|clk[0]}] -to \

    [get_clocks {clkb_o}] 0.120

set_clock_uncertainty -hold -from [get_clocks
{pll2_inst|altpll_component|pll|clk[0]}] -to \

[get_clocks {clkb_o}] 0.050
```

Example 16. Clock Uncertainty Constraints (Part 2 of 2)

```
set_clock_uncertainty -setup -from [get_clocks
{pll1_inst|altpll_component|pll|clk[0]}] -to \

   [get_clocks {virtual_ref_clk_i}] 0.180

set_clock_uncertainty -hold -from [get_clocks
{pll1_inst|altpll_component|pll|clk[0]}] -to \

   [get_clocks {virtual_ref_clk_i}] 0.160

set_clock_uncertainty -setup -from [get_clocks
{pll1_inst|altpll_component|pll|clk[0]}] -to\

   [get_clocks {pll1_inst|altpll_component|pll|clk[0]}] 0.120

set_clock_uncertainty -hold -from [get_clocks
{pll1_inst|altpll_component|pll|clk[0]}] -to \

[get_clocks {pll1_inst|altpll_component|pll|clk[0]}] 0.050
```

Step 2: Setting Up the Design in the Quartus II Design Software

In this step, you will set up the pin capacitances shown in Table 9 on page 59 and set the drive strengths for the I/Os. Follow the HardCopy II Advisor for other required settings; for example, unused pins and enabling the Design Assistant. You will also review the **.sdc** that exists in the design and make corrections (or add new constraints).

1. Open the design archive using the current Quartus II Design software.
2. Set the TimeQuest Timing Analyzer as the default Timing Analyzer. Open the **.sdc** and check the constraints for wrong or missing constraints. Add the **.sdc**s to the **.sdc** list.
3. Use the Assignment Editor to add pin capacitances. Use the Pin Planner to plan your pin placements. At this point, it is a good idea to review your device pin-outs for clocks and other interfaces; for example, DDR and LVDS.

☞ Pay special attention to where you place your clock pins. For maximum performance, place the clock pins on dedicated clock I/Os.

4. Under **device settings** (go to **Assignments**, then **Device**), set the drive strengths. Review the guidelines about how to handle unused pins. The tool automatically defaults unused pins as inputs, tri-stated, with weak pull-ups. By default, the drive strength is set to 24 mA. You may need to change this value based on your board's signal integrity (SI) requirements.
5. Open the HardCopy II Advisor and correct the recommended settings. Use the HardCopy II Advisor for compiling the FPGA and HardCopy II revisions (Figure 32).
6. As you go through the flow, review the reports (Synthesis, Fitter, Timing, and Design Assistant). Mark your observations in the sample HardCopy Design Review Document.

Figure 32. HardCopy II Advisor

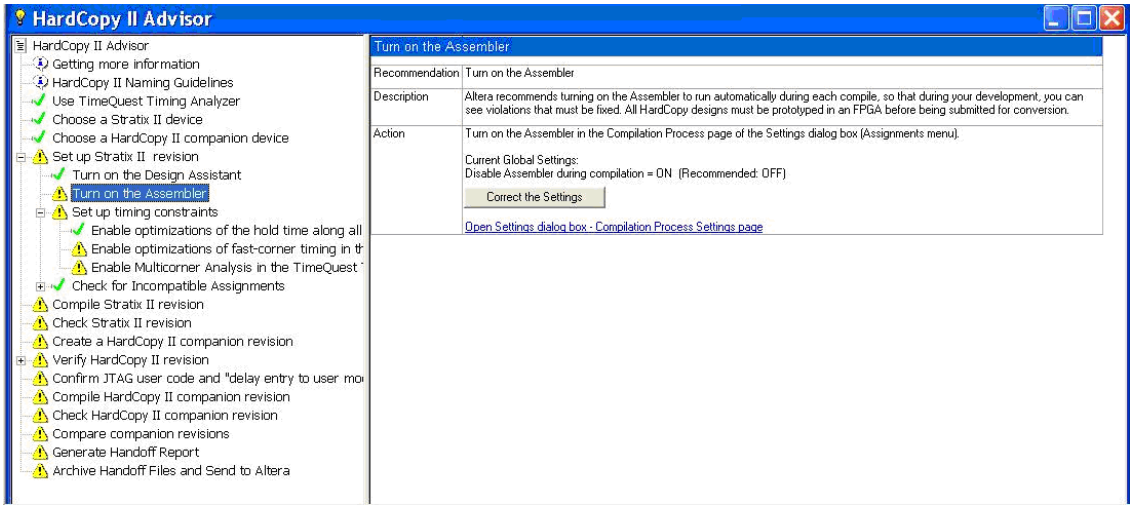
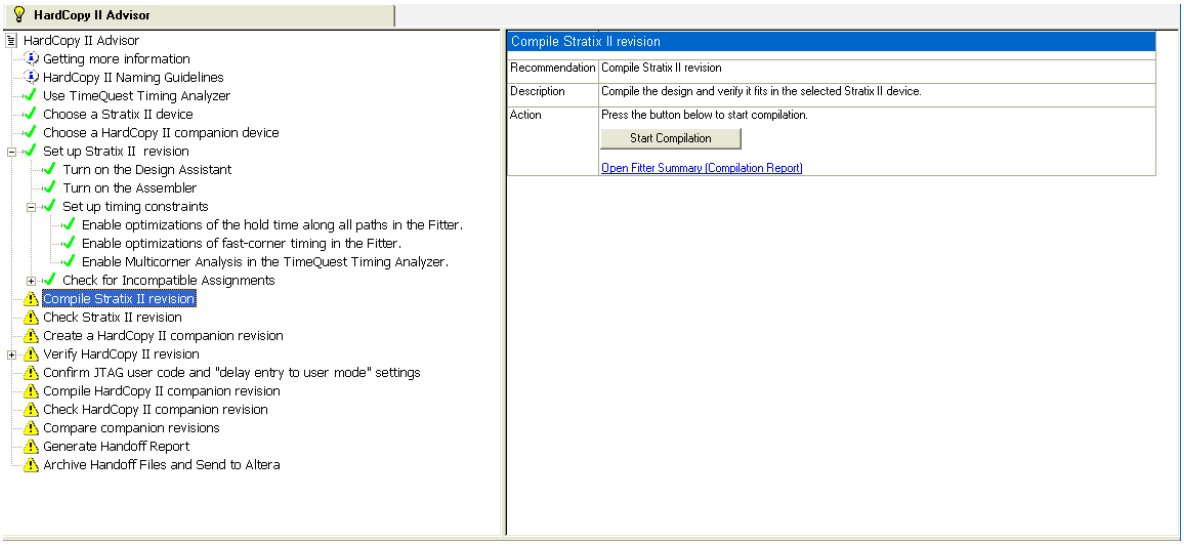


Figure 32 shows the HardCopy II Advisor before correcting the settings. To make the required correction, click **Correct the Settings**.

After you make the recommended changes for the FPGA revision, the HardCopy II Advisor reflects the changes, as shown in Figure 33.

Figure 33. HardCopy II Advisor after Making Recommended Changes



Step 3: Compile the FPGA Revision

Compile the FPGA revision by clicking **Start Compilation** in the HardCopy II Advisor.

Review the reports for Synthesis, Fitter, and Design Assistant warnings and critical warning messages. If some of these warnings can be waived, you can ignore them. However, pay close attention to critical warning messages.

For example, during synthesis, you can ignore the messages shown in Example 17.

Example 17. Example Messages that can be Ignored

```
Warning (10230): Verilog HDL assignment warning at uart_regs.v(694): truncated value with
size 32 to match size of target (1)
Warning (10230): Verilog HDL assignment warning at uart_regs.v(703): truncated value with
size 32 to match size of target (16)
Warning (10230): Verilog HDL assignment warning at uart_regs.v(705): truncated value with
size 32 to match size of target (16)
Warning (10230): Verilog HDL assignment warning at uart_regs.v(743): truncated value with
size 32 to match size of target (8)
```

You can suppress these messages using the message suppression feature in the Quartus II software. To suppress these types of messages, right click on the message, and click **Suppress**, then click **Suppress Similar messages**.

Example 18 shows a critical warning that needs immediate attention:

Example 18. Critical Warning Message

```
Critical Warning: WYSIWYG primitive
"example_ram:ram_inst1|altsyncram:ram_data_rtl_0|altsyncram_7do1:auto_generated|ram_bl
ock1a9" cannot use Memory Initialization File when migrating to HardCopy II device in
non-ROM operation mode
```

This critical warning has been issued because, for HardCopy Series Devices, memory initialization is not supported. The solution to this problem is to remove memory initialization.

Go through all the warning and critical warnings and note them in the Sample HardCopy Design Review Document. For your design to be HardCopy Ready, you must review all of the warnings and they must be either fixed or waived.

After you have gone through the Synthesis report, go through the Fitter report. Look for resource utilization and fan-outs that might affect performance. Look for warnings about pin placement, non-global high fan-out nets that can be promoted to global resources, and other areas in the Resource section of the Fitter. Example 19 shows an sample warning message in the Fitter report that you must resolve.

Example 19. Fitter Report Warning Message

```
Warning: No exact pin location assignment(s) for 94 pins of 94 total pin
```

The warning message is generated because the pins have not been assigned and the Fitter has automatically assigned them. Use the Pin Planner and assign the pins to match the board requirements.

Under the Resource section, review the non-global high fan-out nets and see if you can promote any of them to the global network. In this design, the resets are good candidates for global signal promotion.

Next, review the Timing report to see if you can make any improvements to the design. In this design, timing analysis fails due to a long divider path. A possible solution is to relax the timing requirement on this path, which results in decreased performance. The other possible solution is to redesign this module or add pipeline stages. Adding pipeline stages is a good choice.

Finally, review the Design Assistant warnings. Example 20 shows an example of Design Assistant warnings.

Example 20. Design Assistant Warning Messages

```
Critical Warning: (High) Rule D101: Data bits are not synchronized when transferred
between asynchronous clock domains. Found 8 asynchronous clock domain interface
structure(s) related to this rule.

Critical Warning: (High) Rule D103: Data bits are not correctly synchronized when
transferred between asynchronous clock domains. Found 2 asynchronous clock domain
interface structure(s) related to this rule.

Warning: (Medium) Rule R102: External reset should be synchronized using two cascaded
registers. Found 2 node(s) related to this rule.
```

Review these messages and take the necessary corrective action. For example, for the reset related warning, you can easily add synchronization registers.

After taking the necessary corrective actions, recompile the FPGA.

After successfully recompiling the FPGA revision, follow the steps in the HardCopy Advisor and create the HardCopy II companion revision.

Compile the HardCopy II revision while the FPGA device is still being verified to ensure the design actually fits in the targeted HardCopy II ASIC. If your design appears not to fit, refer to *AN 453: HardCopy II Fitting Techniques*.

Step 4: Compile the HardCopy II Revision

When you have made the necessary corrections in the FPGA revision, compiling the HardCopy II revision is straightforward. Review the report files in the HardCopy II revision again.

Use the HardCopy II Advisor to compile the HardCopy II revision. If the timing requirements are met, perform Revision Compare for the FPGA and HardCopy II revisions. If the Revision Compare passes, generate the required output files for handoff to the HardCopy Design Center.

If you have followed the steps previously outlined, both the FPGA and HardCopy II revisions should meet timing and pass Revision Compare. Other than some minor warnings and critical warnings which you must review, the design is ready for HardCopy Handoff.

If timing is not met, use the TimeQuest Timing Analyzer to report the paths. In some cases, you may have to promote some signals to the appropriate global or regional network to meet timing. If you still do not meet timing, revisit your constraints and check if they are valid. You may have to relax your constraints, re-write the RTL, or have the Quartus II software insert re-timing registers for the failing paths.

If the Revision Compare step fails, check the report to see why this failed and make the necessary corrections in both revisions. You may need to recompile both the FPGA and HardCopy II revisions again.

Step 5: The HardCopy Design Review Document

This section describes what Altera's HardCopy Design Center looks for in a design that is ready for tape-out. During the Design Review process, Altera's HardCopy Design Center reviews the design and tries to work with you to resolve issues or, if you choose to, waive some of the warnings and critical warnings. The HardCopy Design Review Document for this project is shown below.

If you have followed the guidelines described in the previous section, some of the warnings and observations may not be applicable in the HardCopy Design Review Document. The Design Review Document is representative of what a typical design review looks like when a design is reviewed by the HardCopy Design Center. In real designs, the Design Review Document is completed by one of the engineers at Altera's HardCopy Design Center.

Sample HardCopy Design Review Document

Revision: 1.0

Template Date: 11/16/2006

Customer: Altera

Customer Engineer Name: Altera Engineer1

Design Top Level Module: demo_design

Date: 8/15/2008

Altera FAE:

Meeting Attendees:

FPGA: EP2S30F484C4

HardCopy Device: HC210F484

Estimated DR1 Date: March 2007

Estimated DR2 Date: July 2007

Quartus Version: 7.1 Build 153

Patches Needed: None

ALUTs: 596/27104

Hcells: 6190 / 618893

IOs: 94/335

DSPs: 0

Memory Bits : 4096/1369728 (FPGA statistic) 4096 / 875520 (HCII statistic)

Synthesis / Analyzer Warnings: This design has many synthesis and analyzer warnings. Here are some examples:

1. Warning (10036): Verilog HDL or VHDL warning at uart_wb.v(192): object "wb_sel_is" assigned a value but never read
2. Warning (10230): Verilog HDL assignment warning at uart_regs.v(616): truncated value with size 32 to match size of target (1)
3. Warning: Port "wb_dat_i" on the entity instantiation of "uart_top_i" is connected to a signal of width 3. The formal width of the signal in the module is 8. Extra bits will be driven by GND.
4. Warning: Reduced register "addr_read_in_reg[7]" with stuck data_in port to stuck value GND
5. Warning: No clock transition on "uart_top:uart_top_i | uart_regs:regs | dl[10]" register due to stuck clock or clock enable
6. Warning: Output pins are stuck at VCC or GND
   a. Warning: Pin "stx_pad_o" stuck at VCC
7. Warning: Design contains 3 input pin(s) that do not drive logic
   a. Warning: No output dependent on input pin "wb_sel_i"
   b. Warning: No output dependent on input pin "srx_pad_i"
   c. Warning: No output dependent on input pin "dcd_pad_i"

These warrant a thorough investigation prior by the designer prior to final handoff.

Special IOs : 3.3V LVTTL, 1.8V SSTL2

24 mA drivers: There are no drive strengths manually specified. Some drive selected by QII are up to the max 24 mA amount and can be at risk of ringing and can cause SSN issues. Designer should manually add drive strengths through the assignment editor.

LVDS: None

Pin Cap: None specified. Pin cap is included in the specification and should be added by the designer.

Memory MIF files: The RAM: ram_inst1 included a RAM implied through the ram.v file. This was removed so that HCII compile could proceed. Check with designer if removal of MIF file is okay.

PLL # used: 2 out of 4 PLLs were used: 2 enhanced - PLL_5 and PLL_6.

PLL Counters: M counters are 12 and 14. charge pump current is >100 uA (good for low static phase error).

PLL Compensation Mode: Normal mode - for removal of clock insertion. No issue.

PLL Reconfig: This has not been added. Design should add the PLL reconfig blocks on both PLLs. Consult with designer when PLL reconfig can be added if if this can be verified in FPGA prototype prior to DR2.

Fit Report Warnings:

1. No output pin locations defined for any pin. Is this what the designer intended?
2. No pin cap specified; refer to the specification and add to the qsf file through the assignment editor.
3. stx_pad_o has VCC driving its datain port. Is this the design intent?

HardCopy II Advisor: Many items are not checked yet. Must rerun HC Advisor with proper settings and recompile FPGA and HCII.

HardCopy II Compile: Reran at Altera once RAM MIF file was removed. Have designer rerun in the future.

FPGA versus ASIC constraints: ASIC style using SDC and TQ. Some of the values look wrong. Check the values in the demo_design_constraints.sdc against the specification.

Unconstrained Paths: Yes, there are unconstrained paths. These need to be constrained. Specifically IOs that are listed as unconstrained: data_i and data_o. These should be added to demo_design_constraints.sdc.

Classic or TimeQuest: Timequest is used

CUT_DISABLE_TIMING_PATHS : NA for TQ flow

Timing Results: Failing pll1 clk0 setup for FPGA, but passing for HCII. FPGA fails by 13 ns. This is due to a long divider path that runs at 29 Mhz in the FPGA and 53 Mhz in the HCII device. Need to add CU on this path. Note that fast timing reports were not generated by designer.

Ran fast model timing with corrected timing constraints and see the following failing paths:

FPGA -

Slow Model - Fails divider paths by a lot. WC negative slack is -12.726 ns. This means the designer should run this clock domain slower in the FPGA or pipeline/rearchitect the divider.

There are also failures in the source synchronous output domain clkb_o. This fails by -0.942 ns.

Fast Model - All paths work. Will need to change phase shift on clkb_o to center the output window.

HCII -

Slow Model - Divider paths and all other core paths work. The IO setup path that failed in the FPGA works in the HCII device since paths tend to speed up in HCII.

Fast Model - There are hold time violations on paths within the same clock domain: PLL2:clk0

PLL Clock Uncertainty Spreadsheet: Was run but numbers need a review. Missing some clock domain transfers in the demo_design_cu.sdc file.

Hold Time Fix all Paths: On

Other Timing Warnings: None

DDR: None

Mhz: _____

Number of interfaces: _____

Width of interface: _____

PLLs per Interface: _____

PLL reconfig: _____

DTW Knowledge: _____

DLL Disable During Reads: _____

DLL Reconfig: _____

PLL Reconfig: Needs to be added to both PLLs.

Design Assistant: Was not originally enabled. After turning this feature on, the following types of violations were introduced:

1. D101: Data bits are not correctly synchronized when transferred between asynchronous clock domains.
2. Rule D103: Data bits are not correctly synchronized when transferred between asynchronous clock domains.
3. Rule R102: External reset should be synchronized using two cascaded registers.
4. Rule D102: Multiple data bits that are transferred across asynchronous clock domains are synchronized, but not all bits may be aligned in the receiving clock domain.

The above design assistant warnings need review with the designer to see if they can be waived or should be changed in the demo_design.

AIs / Comments:

1. The design contained a RAM with memory initialization specified. This is not supported in HardCopy. Modify ram.v to remove the memory MIF file reference.
2. The HardCopy II companion device was specified in the device setting but the HCII fitter was not run since the memory initialization file was there. After removing the MIF file, the design successfully compiled.

3. Correct the settings in the HardCopy II Advisor.
4. Review synthesis and analysis warnings with designer. RTL looks like it has some issues that only the designer can comprehend.
5. Add drive strengths in assignment editor. Avoid 24 mA drivers where necessary.
6. Why does stx_pad_o drive zero?
7. Add pinout constraints unless designer is comfortable with automated assignments for pin locations.
8. Add pin cap as per specification.
9. Add PLL reconfiguration wrappers around the 2 PLLs.
10. Review SDC constraints for correctness against the specification.
11. Add constraints to correct the unconstrained paths on data_i and data_o.
12. FPGA fails setup in PLL1 clk0 domain by large amount (-12 ns). Use performance improvement (lower constraints for FPGA versus HCII) or re-architect the divider.
13. Re-center PLL phase on FPGA output to make timing on the clkb_o domain setup. Recheck if this phase setting works on the HCII device.
14. HCII compile fails hold paths on pll1 clk0 but by small amount (-11ps) and is okay to waive.
15. Similarly the wb_clki path shows a hold time violator of -2ps and is okay to ignore.
16. Once all the action items have been resolved, the design is ready for submission to Altera's HardCopy Design Center for review.

Summary

This section described the FPGA compile and steps that must be taken to make the migration to the HardCopy Series Device a success. The section outlined timing constraints, setting up the FPGA and HardCopy II revisions in the Quartus II Design software, and achieving timing closure in both revisions. Also reviewed was the need to pay extra attention to the various warning messages and noting them in the HardCopy Worksheet. Some of these warnings require the designer to review them and make the necessary changes to the design or waive them.

Document Revision History

Table 11 shows the revision history for this application note.

Table 11. Document Revision History

Date	Version	Changes Made
July 2010	version 2.1	■ Changed "-use_tan_name" to "-use_net_name" in the "The derive_pll_clocks Command" section.
December 2009	version 2.0	■ Updated the "Synchronizer", "Synchronizing an Asynchronous Reset", "No Virtual Clocks", "Virtual Clocks", "Case 2: With PLL", "Timing Exceptions", "Design Partition and Signal Naming Convention", and "Multi-Frequency Analysis" sections. ■ Updated Figure 6, Figure 24, Figure 25, Figure 28, Figure 29, and Figure 30. Added color to graphics where needed. ■ Added "Document Revision History" section. ■ Minor text edits.
November 2008	version 1.0	Initial release.

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support