

This chapter describes the Quartus® II software features that are used with SOPC Builder, including the following:

- “Quartus II IP File”
- “Quartus II Incremental Compilation” on page 5-1
- “TimeQuest Timing Analyzer” on page 5-2

Quartus II IP File

The Quartus II IP File (**.qip**) generated by SOPC Builder provides the Quartus II software with all required information about your SOPC Builder system. SOPC Builder creates the **.qip** during system generation and adds a reference to it in the Quartus II Settings File (**.qsf**).

The **.qip** file includes references to the following information:

- HDL files used in the SOPC Builder system
- TimeQuest Timing Analyzer Synopsys Design Constraint (**.sdc**) files
- Component definition files for archiving purposes

The **.qip** file is based on Tcl scripting syntax and is similar to the **.qsf** file. The information required to process most components is included in the system's single **.qip** file. Some complex components provide their own **.qip** file, in which case the system's **.qip** file references the component **.qip** file.




The **.qip** file is normally added to your project automatically by SOPC Builder. If it does not get added automatically you can add the file in the same way that you add other source files to your project. You can also have a **.qip** file for each component in your design. When you generate a design, each **.qip** is pulled into the main **.qip** file for your system by reference.

Quartus II Incremental Compilation

SOPC Builder supports the Quartus II incremental compilation feature, which allows you to separately compile isolated portions, or partitions, of a design. From within the Quartus II software, you can designate an entire SOPC Builder system as a design partition, or you can designate individual SOPC Builder components as design partitions.




Changing the parameters of a component and regenerating your system only prompts other partitions within the same system to recompile if the HDL in that partition depends on the changed parameters. The HDL you generate for the Nios® II processor is optimized as related to components to which the Nios II processor is connected.

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

TimeQuest Timing Analyzer

Altera recommends the TimeQuest Timing Analyzer in the Quartus II software for analysis of all new designs. SOPC Builder automatically generates a TimeQuest `.sdc` constraints file for SOPC Builder systems and components. In most cases, you use the TimeQuest constraints to declare false paths for signals that cross clock domains within a component, so that the TimeQuest Timing Analyzer does not perform normal setup and hold analysis for them. You can add `.sdc` files for custom components, using **Add Files** command on **HDL Files** tab in the Component Editor. Turn on the **Synth** option and turn off the **Synth** option.

The Classic Timing Analyzer was primary in earlier versions of the Quartus II software. However, Altera now recommends that you constrain designs before compilation, because the TimeQuest Timing Analyzer reports any unconstrained paths by default during the compilation process.

 Refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook* for further description of the TimeQuest Timing Analyzer. Refer to the *Switching to the Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook* for a description of the benefits of using the TimeQuest Timing Analyzer rather than the Classic Timing Analyzer. Refer to *TimeQuest Example: Basic SDC Example* on www.altera.com for a working example of using the TimeQuest Timing Analyzer. Refer to *TimeQuest Design Examples* on www.altera.com for further details about how to constrain different types of circuits for the TimeQuest Timing Analyzer.

Analyzing PLLs

You must constrain PLL clocks for proper analysis by the TimeQuest Timing Analyzer. You can define clocks generated by PLLs using one of the following methods:

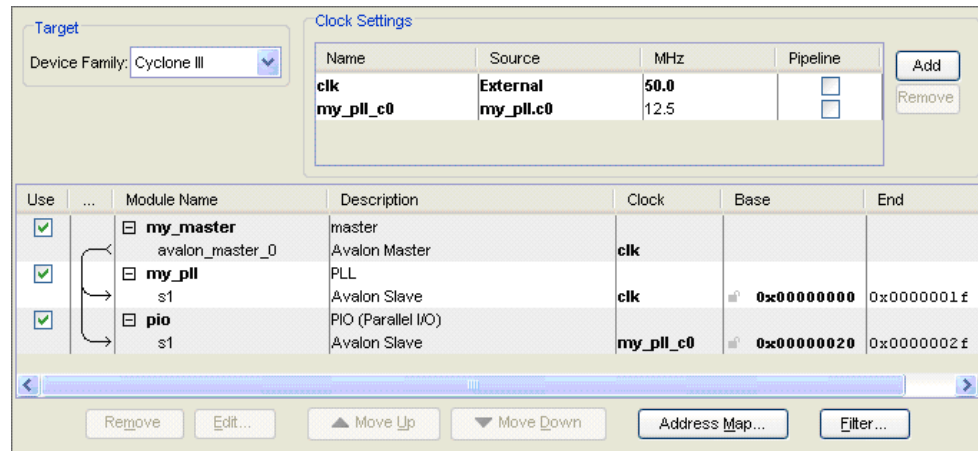
- Use the `derive_pll_clocks` command to derive clocks for all PLL outputs in the design. This is the best method.
- Use the `create_generated_clock` command to designate each clock output.
- Use the `-create_base_clocks` option of the `derive_pll_clock` assignments to designate the base clock feeding the PLL.

The following example focuses on the use of the `derive_pll_clocks` assignment, because this method automatically defines clock frequencies and phase shifts.

 `derive_pll_clocks` generates clocks for all PLLs in the Quartus II hardware project, not just for the PLLs in the SOPC Builder system.

The SOPC system shown in [Figure 5-1](#) illustrates the use of the `derive_pll_clocks` assignment in the case of a single clock input and one PLL using a single output.

Figure 5-1. Example SOPC System



After running the following commands in the TimeQuest Timing Analyzer, two clocks are generated:

```
create_clock -name master_clk -period 20 [get_ports {clk}]
derive_pll_clocks
```

The TimeQuest Timing Analyzer analyzes and reports performance of the constrained clocks in the Clocks Summary report. This displays a report as shown in Figure 5-2.

Figure 5-2. Clocks Summary Report

Clocks Summary			
	Clock Name	Type	Period
1	master_clk	Base	20.000
2	the_my_pll the_pll altpll_component auto_generated pll1 clk[0]	Generated	80.000

master_clk is defined by the create_clock command, and the_my_pll clock is derived from the derive_pll_clocks command.

Analyzing Slow Asynchronous I/O Paths

If you use slow asynchronous I/O in an SOPC Builder system, such as PIO and UART peripherals, you do not have to analyze these paths because they are asynchronous to the clock that is used to capture or output data. In this case you must designate false paths to produce an accurate analysis.

For outputs, set a false path between the launch clock and the output. For inputs, a false path should be set between the input and the latching clock. For bidirectional signals, set a false path from the launching clock to the bidirectional pin and also from the bidirectional pin to the latching clock. Launch and latch clocks are typically the clocks associated with the SOPC Builder module that includes the I/O.

For the system described in the PLL section, the following command sets false paths for the PLL outputs:

```
set_false_path -to [get_ports {*_pio[*]}]
```

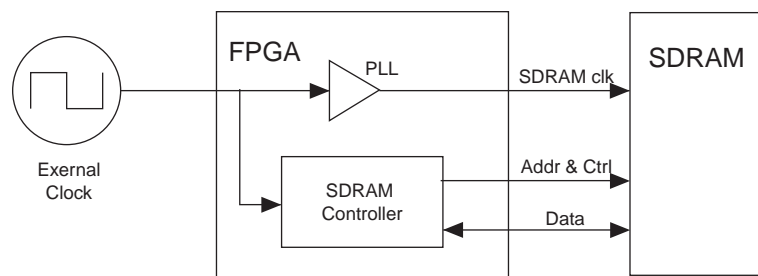
Because design contains a 4-bit PIO, filter `*_pio[*]` includes the following I/O pins.

- `out_port_from_the_pio[0]`
- `out_port_from_the_pio[1]`
- `out_port_from_the_pio[2]`
- `out_port_from_the_pio[3]`

Analyzing Single Data Rate SDRAM and SSRAM

Single data SDRAM interfaces in SOPC Builder typically use the type of circuit shown in Figure 5-3. You can use a PLL to fine tune the phase shift to the external memory to meet I/O timing requirements.

Figure 5-3. Typical Single Data Rate SDRAM Circuit



To constrain this interface, you must create a clock that is recognized by the external SDRAM; then you must set the I/O timing relative to that clock.

[Example 5-1](#) shows how to constrain a PLL output clock and set a Tcl variable for that clock.

Example 5-1. Constraining PLL Output Clock

```
create_clock -period 20.000 -name ext_clk [get_ports {clk}]
derive_pll_clocks
set sdr_clk\my_pll_inst|altpll_component|auto_generated|pll1|clk[0]
```

You can then use the `create_generated_clock` command to define a clock as recognized by the external memory. This generated clock automatically adds delays associated with routing to the clock output pin and the delay of the pin itself. You must also account for some board delay due to the PCB trace between the FPGA and SDRAM by using the `offset` option.

The following command shows the creation of the `sdr_clk_pin` generated clock derived from the output pin `sdr_clk` clock. A 0.5 ns offset accounts for PCB routing delay.

```
create_generated_clock -name sdr_clk_pin -source $sdr_clk \
-offset 0.5 [get_ports {sdr_clk}]
```

There may be some uncertainty associated with the PCB delay not accounted for in this command. The uncertainty can be included in the I/O constraints that are specific to input or output and minimum or maximum delays.

The I/O constraints must be defined in relation to the data sheet for the external memory. Figure 5-4 shows part of a data sheet for an SDRAM device with the worst case input and output timing highlighted for a CAS latency of 3.

Figure 5-4. AC Characteristics from SDRAM Device Data sheet

AC Characteristics Parameter	Symbol	-6		-7		Units	Notes
		Min	Max	Min	Max		
Access time from CLK (pos. edge)	CL = 3	^t AC (3)	5.5		5.5	ns	
	CL = 2	^t AC (2)	7.5		8	ns	
	CL = 1	^t AC (1)	17		17	ns	
Address hold time		^t AH	1		1	ns	
Address setup time		^t AS	1.5		2	ns	
CLK high-level width		^t CH	2.5		2.75	ns	
CLK low-level width		^t CL	2.5		2.75	ns	
Clock cycle time	CL = 3	^t CK (3)	6		7	ns	23
	CL = 2	^t CK (2)	10		10	ns	23
	CL = 1	^t CK (1)	20		20	ns	23
CKE hold time		^t CKH	1		1	ns	
CKE setup time		^t CKS	1.5		2	ns	
CS#, RAS#, CAS#, WE#, DQM hold time		^t CMH	1		1	ns	
CS#, RAS#, CAS#, WE#, DQM setup time		^t CMS	1.5		2	ns	
Data-in hold time		^t DH	1		1	ns	
Data-in setup time		^t DS	1.5		2	ns	
Data-out High-Z time	CL = 3	^t HZ (3)	5.5		5.5	ns	10
	CL = 2	^t HZ (2)	7.5		8	ns	10
	CL = 1	^t HZ (1)	17		17	ns	10
Data-out Low-Z time		^t LZ	1		1	ns	
Data-out hold time		^t OH	2		2.5	ns	

The mapping of external memory timing to FPGA I/O delays is shown in Table 5-1. This table also shows whether the minimum or maximum PCB routing delay should be used, which must be added to the FPGA delay constraints.

Table 5-1. External Memory Timing

Memory Timing	FPGA Timing	PCB Routing
Max clock to out	Max input delay	Max
Min clock to out	Min input delay	Min
Min setup	Max output delay	Max
Min hold	Min output delay (-ve)	Min

Note to Table 5-1:

- (1) The constraint for minimum output delay is actually 0 – Min hold.

You can use the `set_input_delay` and `set_output_delay` commands to set the I/O constraints. In the following examples, a common PCB routing delay of $0.5\text{ns} \pm 0.1\text{ns}$ is used, which adds a 0.4 ns or 0.6 ns delay to the paths. [Example 5-2](#) illustrates the use of these commands.

Example 5-2. `set_input_delay` and `set_output_delay` commands

```
set_input_delay -clock sdram_clk_pin -max [expr 5.5 + 0.6] <ports>
set_input_delay -clock sdram_clk_pin -min [expr 2.5 + 0.4] <ports>
set_output_delay -clock sdram_clk_pin -max [expr 2.0 + 0.6] <ports>
set_output_delay -clock sdram_clk_pin -min [expr -1 + 0.4] <ports>
```

In this example, `<ports>` represent a list of I/O ports for the relevant constraints as shown in [Example 5-3](#).

Example 5-3. `<ports>`

```
set_output_delay -clock sdram_clk_pin -max [expr 2.0 + 1.2] \
[get_ports {cas_n ras_n cs_n we_n addr[*]}]
```

You can use multiple `set_input_delay` and `set_output_delay` commands to set different delays for different I/O.

Analyzing Tristate Bridges and Asynchronous Devices

This section discusses the timing constraints associated with the Avalon tristate bridge and asynchronous external devices, such as the CFI Flash and user tristate components. These components typically have slower performance requirements compared with the FPGA, and SOPC Builder generates logic within the interface to control timing across multiple clock cycles. You define the tristate component's timing parameters by entering data for setup, wait, and hold times.

For the interface types previously discussed, the timing is controlled by a state machine that is generated based on setup, wait, and hold settings you specify in the component editor. Because data sheet values for the FPGA are used in calculating the timing, the constraints simply ensure the data sheet timing is met. Adding these constraints ensures that issues associated with data sheet misinterpretation and fitting problems that affect I/O timing are captured.


The TimeQuest Timing Analyzer uses constraints that are based upon the timing of the external device.




For further information on how to convert older FPGA-centric constraints into system-centric constraints, refer to *Switching to the Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Analyzing DDR and DDR2 Memories

When using DDR, DDR2, or DDR3 memory with Cyclone® III, Stratix® III, and Stratix IV families, you must use the corresponding High-Performance Controller MegaCore® function. You can use the MegaWizard™ Plug-In Manager interface to parameterize these functions and generate timing constraints in the form of `.sdc` files. You must ensure that the constraints file associated with the MegaCore function is included in the project for timing analysis. You can add an `.sdc` file to the project by clicking **Add/Remove Files in Project** on the Project menu in the Quartus II software.

 As these MegaCore functions make use of the `derive_pll_clocks` command, conflicts may occur if your `.sdc` file also uses these constraints.

 For more design examples, refer to [TimeQuest Design Examples](http://www.altera.com) on www.altera.com. Also, *AN: 433 Constraining and Analyzing Source-Synchronous Interfaces* describes source synchronous constraints for the TimeQuest Timing Analyzer.

Document Revision History

Table 5-2 shows the revision history for this chapter.

Table 5-2. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	<ul style="list-style-type: none"> ■ Corrected <code>set_output_delay</code> expression for the SDRAM clock pin minimum in Example 5-2 on page 5-6. 	
March 2009, v9.0.0	<ul style="list-style-type: none"> ■ No changes to content from previous release. 	—
November 2008, v8.1.0	<ul style="list-style-type: none"> ■ No changes to content from previous release. ■ Changed page size to 8.5 x 11 inches 	—
May 2008, v8.0.0	Initial release.	Information moved from other chapters and consolidated here.

