A STANDARDIZED PROCEDURE FOR CLOSING TIMING ON OpenHPSDR FPGA FIRMWARE DESIGNS

by Joe A. Martin, PhD, K5SO

August 26, 2014

Introduction

This document has been prepared to serve as a "field guide" for those first learning to close timing on firmware designs using Altera's Quartus II firmware development program. It is focused on use of the TimeQuest timing analyzer utility within that program to achieve timing closure for Field Programable Gate Array (FPGA) firmware designs in the context of the Open High Performance Software Designed Radio (OpenHPSDR) project.

The author makes no claim to be an expert in this area. The procedure detailed in this document derives from the author's experiences obtained while learning to use the Quartus II development tools on OpenHPSDR firmware designs. The information contained herein may well contain errors or omissions and will certainly contain personal bias on the part of the author. If official information is desired regarding the Altera's development tools and how to use them properly the reader is referred to the formal documents provided by Altera. Much of the information provided here originates directly from ideas acquired by the author while reading "TimeQuest Users Guide" by Ryan Scoville[1] and, to a lesser extent, Altera's "TimeQuest Timing Analyzer: Quick Start Tutorial".[2]

At the outset the reader should understand that the development of a successful firmware design consists of two distinct activities; creation of a high-level logic design for the firmware and, almost independently, establishment of a set of suitable timing constraints for the design. The first activity is accomplished by using the Verilog programming language and is not covered in this document. Here we assume that the Verilog design already exists and that the design simply needs a suitable timing constraint file to be created for it. The second activity, that of establishing a procedure to achieve timing closure, is the principal focus here. It is not necessary that the reader know anything much about Verilog programming as the timing constraint file does not use the Verilog programming language.

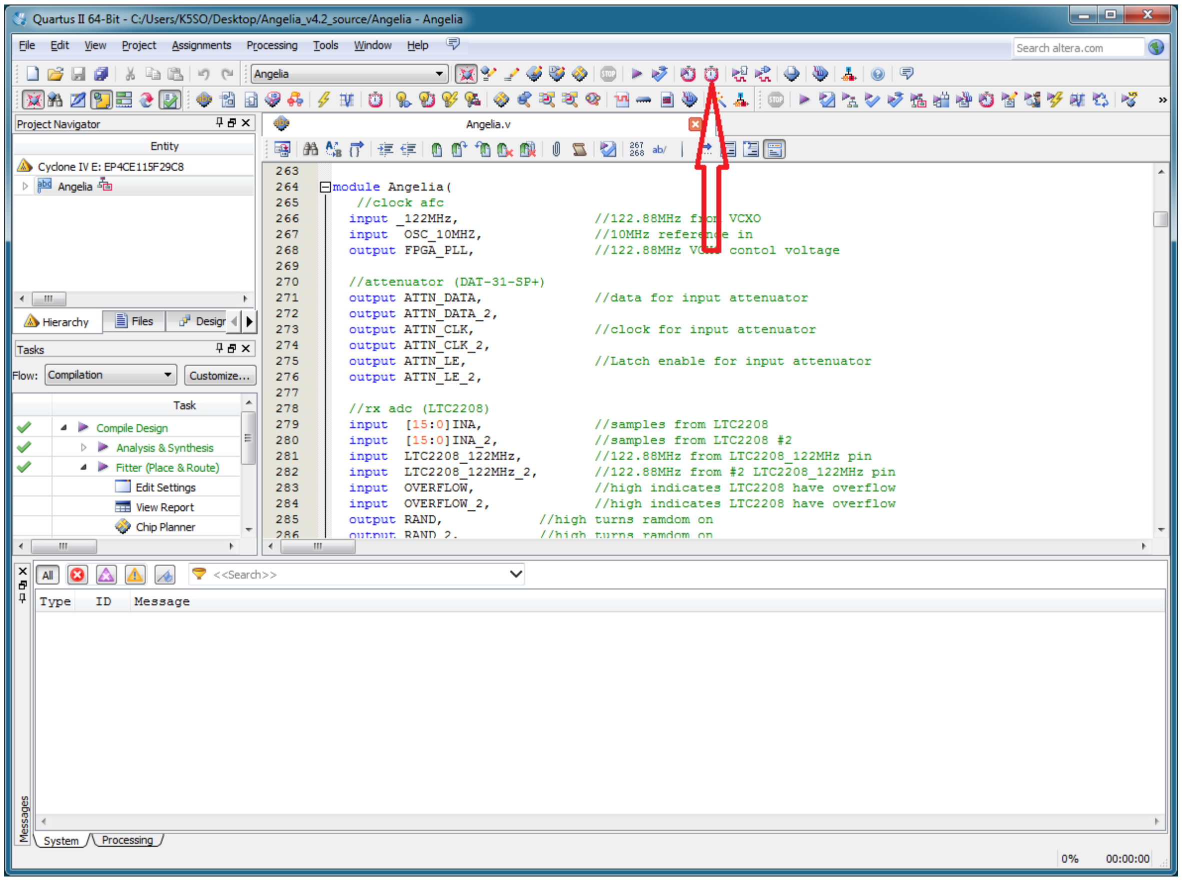
It is perhaps useful to note that there is no unique timing closure solution for any given firmware design. Many different possible timing closure solutions can adequately satisfy the timing requirements for any given firmware design. This fact makes it sometimes confusing to those attempting to "close timing" on firmware designs. It is the goal of this document to provide a single standardized procedure toward achieving at least one of the many possible suitable closure solutions for any arbitrary OpenHPSDR firmware design, using a limited number of the Altera constraint types that are available. The reader should take note that there is no claim made here that this standardized procedure is the "best" procedure one could take. The goal is simply to offer a workable method that beginners may use to achieve timing closure on OpenHPSDR firmware designs while they gain experience in the process. It is expected that as the beginner gains experience a personalized procedure to close timing will naturally evolve and rather quickly rigid adherence to this one will become unnecessary.

Running the Quartus II TimeQuest Utility

Timing analyses and timing adjustments for an OpenHPSDR FPGA firmware design are performed using the TimeQuest utility within the Altera Quartus II program. A primary function of the Quartus II program is to provide tools to write, edit, compile and produce the various files associated with an FPGA firmware design. The TimeQuest utility within the Quartus II program is used to analyze the design with respect to clocks, signal paths, and signal ports and to assist in properly constraining those items.

To begin using the TimeQuest utility, first open an existing firmware design in the main Quartus II program and click on the TimeQuest icon on the tool bar, as shown in the screenshot below:

USE "ZOOM" IN YOUR PDF VIEWER TO ENLARGE THE FIGURES



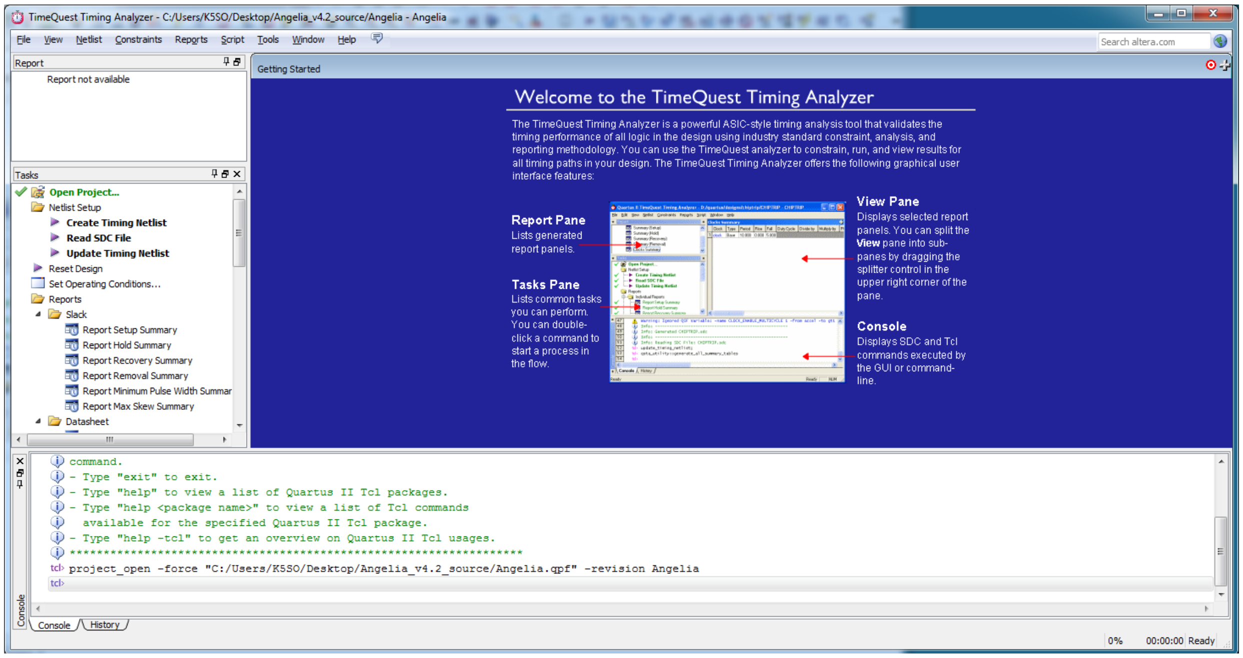
```
module Angelia(
  //clock afc
 input _122MHz,                //122.88MHz from VCXO
 input  OSC_10MHZ,             //10MHz reference in
 output FPGA_PLL,              //122.88MHz VCXO contol voltage

 //attenuator (DAT-31-SP+)
 output ATTN_DATA,             //data for input attenuator
 output ATTN_DATA_2,
 output ATTN_CLK,              //clock for input attenuator
 output ATTN_CLK_2,
 output ATTN_LE,               //Latch enable for input attenuator
 output ATTN_LE_2,

 //rx adc (LTC2208)
 input  [15:0]INA,             //samples from LTC2208
 input  [15:0]INA_2,           //samples from LTC2208 #2
 input  LTC2208_122MHz,        //122.88MHz from LTC2208_122MHz pin
 input  LTC2208_122MHz_2,      //122.88MHz from #2 LTC2208_122MHz pin
 input  OVERFLOW,              //high indicates LTC2208 have overflow
 input  OVERFLOW_2,            //high indicates LTC2208 have overflow
 output RAND,           //high turns ramdom on
 output RAND_2,         //high turns ramdom on
```

Fig. 1. Main Quartus II display, red arrow indicates the icon used to launch the TimeQuest utility.

Click the icon to open the TimeQuest Timing Analyzer window, shown below:



Welcome to the TimeQuest Timing Analyzer

The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in the design using industry standard constraint, analysis, and reporting methodology. You can use the TimeQuest analyzer to constrain, run, and view results for all timing paths in your design. The TimeQuest Timing Analyzer offers the following graphical user interface features:

Report Pane
Lists generated report panels.

Tasks Pane
Lists common tasks you can perform. You can double-click a command to start a process in the flow.

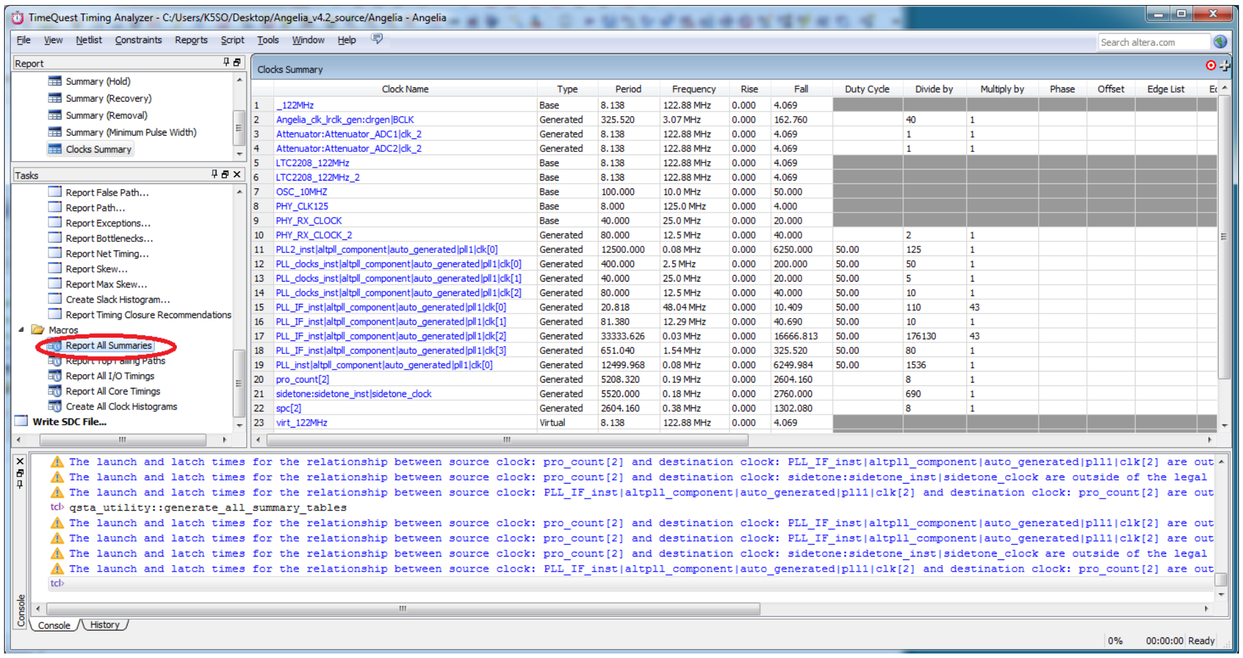
View Pane
Displays selected report panels. You can split the **View** pane into sub-panes by dragging the splitter control in the upper right corner of the pane.

Console
Displays SDC and Tcl commands executed by the GUI or command-line.

```
command.
- Type "exit" to exit.
- Type "help" to view a list of Quartus II Tcl packages.
- Type "help <package name>" to view a list of Tcl commands
  available for the specified Quartus II Tcl package.
- Type "help -tcl" to get an overview on Quartus II Tcl usages.
*******************************************************************
tcl> project_open -force "C:/Users/K5SO/Desktop/Angelia_v4.2_source/Angelia.qpf" -revision Angelia
tcl>
```

Fig. 2. TimeQuest Timing Analyzer main window at startup of the TimeQuest utility.

Using the TimeQuest utility it is possible to analyze the firmware design via a wide variety of tools and reports that the utility provides. To begin to work with a firmware design it is oftentimes best to perform an initial overall analysis. This can be achieved by double-clicking on the “Report All Summaries” task, which generates several informative reports, as shown below:



Clocks Summary

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase	Offset	Edge List	Ec
1	_122MHz	Base	8.138	122.88 MHz	0.000	4.069							
2	Angelia_clk_lrclk_gen:clrgen|BCLK	Generated	325.520	3.07 MHz	0.000	162.760		40	1				
3	Attenuator:Attenuator_ADC1|clk_2	Generated	8.138	122.88 MHz	0.000	4.069		1	1				
4	Attenuator:Attenuator_ADC2|clk_2	Generated	8.138	122.88 MHz	0.000	4.069		1	1				
5	LTC2208_122MHz	Base	8.138	122.88 MHz	0.000	4.069							
6	LTC2208_122MHz_2	Base	8.138	122.88 MHz	0.000	4.069							
7	OSC_10MHZ	Base	100.000	10.0 MHz	0.000	50.000							
8	PHY_CLK125	Base	8.000	125.0 MHz	0.000	4.000							
9	PHY_RX_CLOCK	Base	40.000	25.0 MHz	0.000	20.000							
10	PHY_RX_CLOCK_2	Generated	80.000	12.5 MHz	0.000	40.000		2	1				
11	PLL2_inst|altpll_component|auto_generated|pll1|clk[0]	Generated	12500.000	0.08 MHz	0.000	6250.000	50.00	125	1				
12	PLL_clocks_inst|altpll_component|auto_generated|pll1|clk[0]	Generated	400.000	2.5 MHz	0.000	200.000	50.00	50	1				
13	PLL_clocks_inst|altpll_component|auto_generated|pll1|clk[1]	Generated	40.000	25.0 MHz	0.000	20.000	50.00	5	1				
14	PLL_clocks_inst|altpll_component|auto_generated|pll1|clk[2]	Generated	80.000	12.5 MHz	0.000	40.000	50.00	10	1				
15	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	Generated	20.818	48.04 MHz	0.000	10.409	50.00	110	43				
16	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[1]	Generated	81.380	12.29 MHz	0.000	40.690	50.00	10	1				
17	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2]	Generated	33333.626	0.03 MHz	0.000	16666.813	50.00	176130	43				
18	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[3]	Generated	651.040	1.54 MHz	0.000	325.520	50.00	80	1				
19	PLL_inst|altpll_component|auto_generated|pll1|clk[0]	Generated	12499.968	0.08 MHz	0.000	6249.984	50.00	1536	1				
20	pro_count[2]	Generated	5208.320	0.19 MHz	0.000	2604.160		8	1				
21	sidetone:sidetone_inst|sidetone_clock	Generated	5520.000	0.18 MHz	0.000	2760.000		690	1				
22	spc[2]	Generated	2604.160	0.38 MHz	0.000	1302.080		8	1				
23	virt_122MHz	Virtual	8.138	122.88 MHz	0.000	4.069							

```
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl> qsta_utility::generate_all_summary_tables
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl>
```

Fig. 3. TimeQuest display after initiating the "Report All Summaries" task.

Summary reports are useful for a variety of purposes. As an example, the "Clocks Summary" shown in Figure 3 permits you to see what clocks are in the firmware design and what their characteristics are. Likely, if you are a beginner, there are many more clocks in the list than you at first expected there to be, especially if you initially assumed that the one or two crystal oscillator modules on the hardware board are the only clocks. A variety of summary reports can be viewed after clicking the "Report All Summaries" icon but we needn't explore those at this time; we will examine and use several report types later.

A critical initial step in working to close timing on a firmware design is to ensure that you have all clocks, and signal paths and signal ports "constrained". That is, you must let TimeQuest know what clocks and paths exist in the design and what their characteristics are. TimeQuest can automatically detect many of these items, but not all, therefore it is essential that you "constrain" all clocks and paths/ports properly. Further, TimeQuest may find "clocks" in the design that you might not initially consider to be clocks when examining the logic design. TimeQuest finds these easily and reports them, listing them as "unconstrained clocks" if you don't have entries for them in the Synopsis Design Constraint file for the project; that is, the *.sdc file.

By "constrain" it is meant that an appropriate entry in the *.sdc file exists for that item. Both TimeQuest and the main Quartus II program use the *.sdc file extensively to gain information about how to produce files for and analyze the firmware design. Therefore in order to achieve a successful firmware design it is important that you have appropriate entries in the *.sdc file and that you haven't, for example, inadvertently omitted clocks that are actually present. The "*" notation in "*.sdc" refers to the project name. OpenHPSDR examples of *.sdc files are Angelia.sdc, Hermes.sdc, Orion.sdc, etc.

The *.SDC file

Quartus II uses the Synopsis Design Constraint file (*.sdc file) to define clocks, to group clocks of common origin, and to inform the Quartus II fitter/router and TimeQuest analyzer of paths that should have specific delays applied to them during the fitting/routing process performed by Quartus II at compile time. It is important, even critical, that the definitions and specifications in the *.sdc file are correct, else Quartus II will not be able to correctly produce the files necessary to load and run the Verilog logic design successfully in the FPGA. The *.sdc file is editable using the TimeQuest utility or, in fact, using any text editor.

Creating Initial Constraints

The task of preparing a suitable *.sdc file for the firmware design is to identify what clocks are present in the design and what their characteristics are. Specifically, Quartus II and the TimeQuest analyzer need to know, via the information in the *.sdc file, whether the clocks used in the design are generated outside of the FPGA or generated inside the FPGA, whether they are to be treated as synchronous or asynchronous with respect to other clocks in the design, and what the clock frequencies are. A wide range of other characteristics can be specified for each clock via the entries in the *.sdc file but specifying only the characteristics mentioned above is usually sufficient for our tasks. Certain default assumptions will be made by Quartus II and TimeQuest if you fail to specify some parameters. If you happen to omit defining a clock, particularly an externally generated clock, Quartus II/TimeQuest may or may not detect it to alert you that the clock is “unconstrained”; that is, the characteristics of the clock are unspecified. If no alert is given, TimeQuest has ignored that such a clock exists in the design. This circumstance is a bad thing as none of the signal paths associated with that clock will be analyzed by the timing wizard. Further, those paths could easily have timing failures that prevent the design from successfully running on the intended hardware.

In our standardized procedure the *.sdc file for any firmware design should contain one or more of the following constraints and/or constraint types. No other constraint types are necessary. Additional constraint types are offered by Altera for use in the *.sdc file but only this particular subset of constraint types is used here:

create_clock ...
derive_pll_clocks
derive_clock_uncertainty
create_generated_clocks ...

set_clock_groups ...
set_input_delay ...
set_output_delay ...
set_max_delay ...
set_min_delay ...

Each of these constraint types is briefly described below.

create_clock
This is a one line statement describing an externally generated clock signal; that is, a clock that is generated outside of the FPGA chip. Each clock that is generated outside the FPGA should have a *create_clock* constraint in the *.sdc file. Externally generated clocks are termed “Base Clocks” in the timing reports.

Typical example:

create_clock -name PHY_CLK125 -period 8.000 [get_ports {PHY_CLK125}]

In this example the clock named “PHY_CLK125” is defined as having a period of 8 nSec (i.e., it is a 125 MHz clock). The name of the clock is the same name as the port for the clock. This is commonly done but we should recognize that there is a difference between a clock and its port; they are actually different entities. As a practical matter, this distinction is not important for us to worry about in this document, however. An equivalent syntax to use for this example is

create_clock -name PHY_CLK125 -period 8.000 PHY_CLK125

As noted above, clocks defined using the *create_clock* constraint are termed “Base Clocks”. There is a special case of this constraint type that is used specifically for clocks that are used to clock signals into the FPGA from an outside source (such as data from an ADC). These special-case clocks are called “virtual clocks”, instead of base clocks, and, to distinguish them from base clocks, they have no port designation specified in the *create_clock* constraint.

Typical example of this special clock constraint:

create_clock -name virt_PHY_CLK125 -period 8.000

In this example the name of the clock has been given as “*virt...*” as a helpful indicator to the reader to realize at a glance that this clock is a virtual clock that is used for clocking signals into the FPGA. The term “virt” is not needed by Quartus II or the TimeQuest utility as the fact that no port name is specified in the *create_clock* constraint automatically makes this clock a virtual clock by default and the clock will be reported by TimeQuest as a virtual clock in the Clocks Summary report regardless of what you call the clock.

As you will see later, virtual clocks are used in the *set_input_delay* constraint. It is not necessary that the beginner understand the subtleties associated with this special constraint type. As a practical matter it is only necessary that the beginner recognize that input signals into the FPGA should be using clocks that are designated as virtual clocks.

Output signals do not have this requirement; therefore virtual clocks are not used in the *set_output_delay* constraint which is described later.

derive_pll_clocks
This is a single-statement constraint with no modifiers. It instructs Quartus II/TImeQuest to automatically generate constraints for all clocks that are created by a PLL in the FPGA. All of our OpenHPSDR firmware designs should include the *derive_pll_clocks* constraint in the *.sdc file.

Example:

derive_pll_clocks

derive_clock_uncertainty
This is a single-statement constraint with no modifiers. It instructs Quartus II/TimeQuest to automatically calculate the clock uncertainties in the design and use them in the routing/fitting and timing analyses. All of our HPSDR firmware designs should include the *derive_clock_uncertainty* constraint in the *.sdc file.

Example:

derive_clock_uncertainty

create_generated_clocks
This is a one line constraint for each clock that is generated within the FPGA chip, excluding clocks that are direct outputs from a PLL in the FPGA.

Typical example:

create_generated_clock -name PHY_RX_CLOCK_2 -source PHY_RX_CLOCK -divide_by 2 PHY_RX_CLOCK_2

In this example the generated clock, *PHY_RX_CLOCK_2*, is specified as being half the frequency of the source clock *PHY_RX_CLOCK*.

PLL-generated clocks are automatically constrained via the *derive_pll_clocks* constraint mentioned above and should not be explicitly constrained in the *.sdc file.

set_clock_groups
This constraint is used to specify which clocks may be treated as asynchronous with respect to other clocks in the design; clocks listed in the group are to be treated as being synchronous with respect to each other however.

Typical example:

```
set_clock_groups -asynchronous -group {LTC2208_122MHz  \
                                       _122MHz} \
                               -group {PHY_CLK125 \
                                       PHY_RX_CLOCK }
```

In this example there are two groups of clocks specified, with two clocks in each group. The clocks within a group are related to each other (i.e., they have a common clock source as their origin) but groups are to be treated by Quartus II/TimeQuest as being asynchronous to all other clocks in the firmware design.

Note that the *set_clock_groups* constraint is unnecessary if all clocks in the design are synchronous with each other, as Quartus II/TimeQuest treats all clocks as synchronous with each other by default unless they are explicitly specified otherwise in a *set_clock_groups* constraint.

set_input_delay
This constraint type is used to initially set time delays for input signal paths that use a particular clock. Many paths/ports can be included into one statement as long as all use the same clock in the design. Two statements are required, one to set the maximum input delay and one to set the minimum input delay.

Typical example:

set_input_delay -add_delay -max -clock virt_PHY_RX_CLOCK 1.000 {PHY_MDIO PHY_RX[*] RX_DV PHY_INT_N}

set_input_delay -add_delay -min -clock virt_PHY_RX_CLOCK -1.000 {PHY_MDIO PHY_RX[*] RX_DV PHY_INT_N}

In this example a maximum input delay of 1.000 nSec is to be assigned to the input paths and input ports that are listed between the {} braces, which includes the multiple input signals of data array *PHY_RX[*]*, and a minimum input delay of -1.000 nSec is to be assigned to those same input paths and ports.

The +1.000/-1.000 nSec delays for I/O paths are arbitrary values here and are simply assigned as default values in order to initially constrain the I/O paths. These default values seem to work fine for our OpenHPSDR I/O paths/ports and no further adjustment is usually required. Should an adjustment be required due a “failing path” report from the timing wizard, the offending path(s) can be adjusted further using the “*set_max_delay*” and/or the “*set_min_delay*” constraints described below, as these latter two constraints are overriding constraints and will replace any other delay constraints that may exist in the *.sdc file, including these initial default values that have been specified in the *set_input_delay* constraints.

set_output_delay
This constraint type specifies initial values for output delays that are to be assigned to specific signals leaving the FPGA; as above in the *set_input_delay* constraint example, two statements are required, one to set the maximum output delay and one to set the minimum output delay for the signal paths or signal ports.

Typical example:

set_output_delay -add_delay -max -clock _122MHz 1.000 {ADCCLK ADCMOSI ATTN_CLK DACD[*]}

set_output_delay -add_delay -min -clock _122MHz -1.000 (ADCCLK ADCMOSI ATTN_CLK DACD[*]}

In this example a maximum input delay of 1.000 nSec is assigned to the output paths and output ports that are listed between the {} braces, which includes the multiple output signals from a DAC, (i.e., DACD[*]), and a minimum output delay of -1.000 nSec is to be assigned to those same output paths and ports.

As noted above in the *set_input_delay* constraint discussion, the +1/-1 nSec delays for I/O paths are arbitrary values and are used simply as default values in order to initially constrain the I/O paths. Nevertheless, the default values appear to work fine for our OpenHPSDR I/O paths/ports and no further adjustment is usually required. Should an adjustment be required due to a “failing path” report from the timing wizard, the timing of the failing path(s) can be adjusted further using the “*set_max_delay*” and/or the “*set_min_delay*” constraints described below, as these latter two constraints are overriding constraints and will replace any other delay constraints that may exist in the .sdc file, including the default values that have been set here.

set_max_delay
This constraint is the primary means in our procedure to adjust timing for paths that fail to meet “setup” or “recovery” timing requirements.

When beginning the timing closure process there will be no *set_max_delay* constraints in the *.sdc file. This constraint type is used to adjust timing for paths that the TimeQuest timing wizard reports as “failing paths”. There will ultimately be one of these constraints in the *.sdc for each failing path, or group of failing paths, reported by the timing wizard that fails to meet “setup” timing requirements or “recovery” timing requirements.

Typical example:

set_max_delay -from PHY_CLK125 -to _122MHz 10

In this example a fixed 10 nSec time delay is to be applied to all signals that are launched from the *PHY_CLK125* clock domain and latched into the *_122MHz* clock domain. Such a constraint, using a 10 nSec delay value, might be used to correct a worst case “setup” timing failure or “recovery” timing failure of around -8 nSec slack (i.e., negative slack) as reported in a “failing path” report, resulting in around 1 or 2 nSec positive slack for the signals instead. An explanation of how one determines the delay value to specify in a constraint like this is discussed later in this document.

Paths that are reported as having “negative slack” are failing paths; that is, paths with negative slack values do not meet required timing requirements. Paths that are reported with positive slack values meet timing requirements.

set_min_delay
This constraint is the primary means in our procedure to adjust timing for paths that fail to meet “hold” or “removal” timing requirements. One statement is used for each path or path group.

Typical example:

set_min_delay -from LTC2208_122MHz -to PHY_CLK125 -2

in this example a fixed -2 nSec time delay is to be applied to all signals that are launched from the *LTC2208_122MHz* clock domain and are latched into the *PHY_CLK125* clock domain. Such a constraint might be used to correct a worst case “hold” or “removal” timing failure of around -1 nSec slack (negative slack) for those signals, resulting in around 1 nSec positive slack instead. Explanation of how one determines how much delay to use in a constraint like this is discussed later in this document.

Timing Reports
TimeQuest can produce numerous reports regarding signal path timing in a firmware design; so many in fact that it sometimes seems overwhelming to the beginner to know what reports to request to aid in the quest to close timing on a particular firmware design. We suggest that beginners use of only this subset of the reports and tasks that are available, in an attempt to lessen the apparent complexity of the effort. There are only a handful of reports and tasks that are needed to be able to successfully close timing on our OpenHPSDR firmware designs. These are all accessible in the “Tasks” window of the TimeQuest main display and include the following:

Reset Design
Report Clocks
Report Unconstrained Paths
Report Ignored Constraints
Report Timing...
Report Top Failing Paths
Report All I/O Timings
Report All Core Timings

Each of these will be discussed as we describe our standardized procedure for closing timing.

A Standardized Procedure For Closing Timing

The process of closing timing on a firmware design is, by necessity, an iterative process. This is the case for this standardized procedure as well. The intrinsic iterative nature of obtaining timing closure is due to the fact that the TimeQuest timing wizard works on the last compiled version of the project. When multiple changes are made to the .sdc file the timing wizard performs its new analysis of the "changed" design without re-compiling the design, unless you manually recompile the design before asking the wizard to analyze things. Performing a new compile after each timing adjustment is very inefficient and time-consuming for us to do therefore it is not recommended, as compile times can be quite long for complex designs. Without a fresh compile after making timing changes the changes are analyzed as if the last compiled fit/route information for the placement of the design into the FPGA is appropriate for those changes while in reality any change that is made in terms of adding/subtracting delays will naturally change the fitting and routing of signals within the FPGA.

Sometimes the modified delays that are introduced create insignificant effects in the fit/route placements and everything still meets timing when the final recompiling is done, but often that is not the case and slight additional modifications are needed after the new compile is performed to account for the slight differences in fit/route that Quartus II uses to incorporate the requested changes. Usually only a few paths are affected in such a situation so minor timing adjustments are all that are typically needed to reach complete timing closure.

A standardized procedure, the description of which is the principal goal of this document, consists of the following seven steps:

1) Verify that the firmware design is constrained,

2) Verify that no constraints are being ignored by TimeQuest,

3) Examine the worst-case failing paths,

4) Adjust timing on the worst-case launch/latch clock domains, initially using a single adjustment for all the signals from the launch clock domain to the latch clock domain by adding an “overriding” constraint in the form of *set_max_delay* or *set_min_delay* as appropriate to resolve the mis-timings,

5) Verify that the timing adjustment corrected the issue for this specific group of signal paths,

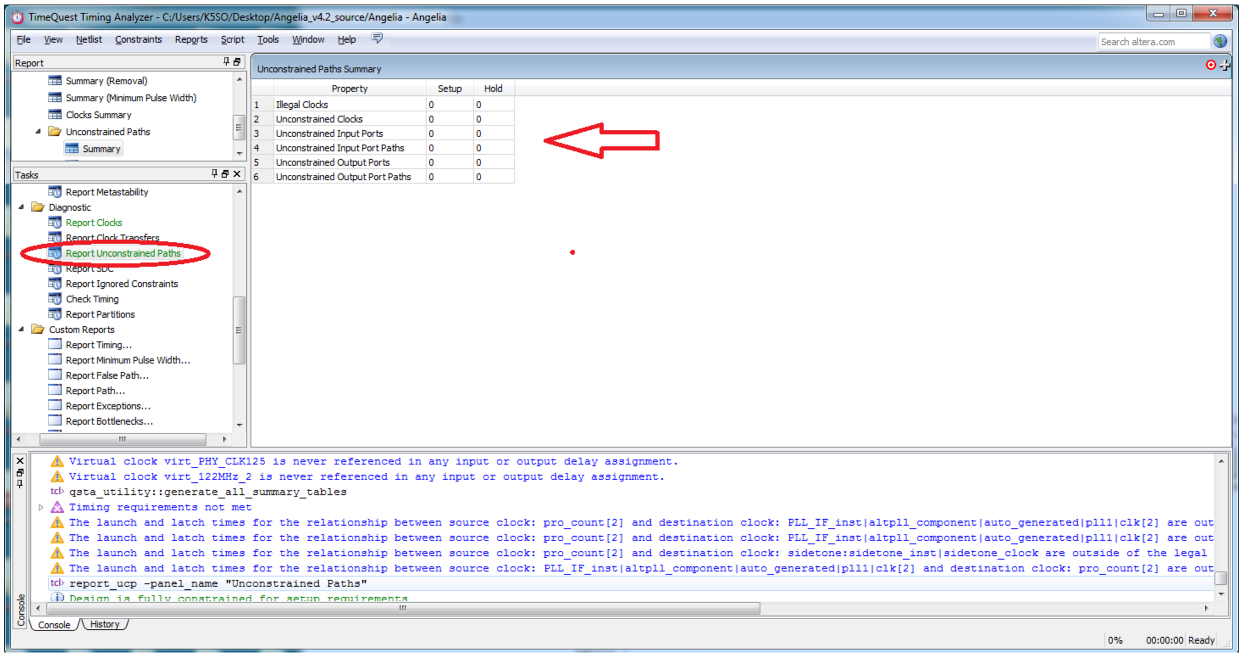
6) Go to 3) above and do the same thing for the next group of failing paths until the timing wizard no longer finds failing paths,

7) Recompile the project and if additional failing paths have been introduced due to the new fitting/routing that was done by Quartus II then repeat the steps above until timing is fully met after the fresh compile is completed.

Each of these steps is explained more fully below.

1) verify that the firmware design is fully constrained

Before beginning to adjust any timing paths in the firmware design it is always best to confirm that everything is fully constrained and acceptable to Quartus II/TimeQuest by generating the “Report Unconstrained Paths” report, as shown below:



Unconstrained Paths Summary

	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	0	0
4	Unconstrained Input Port Paths	0	0
5	Unconstrained Output Ports	0	0
6	Unconstrained Output Port Paths	0	0

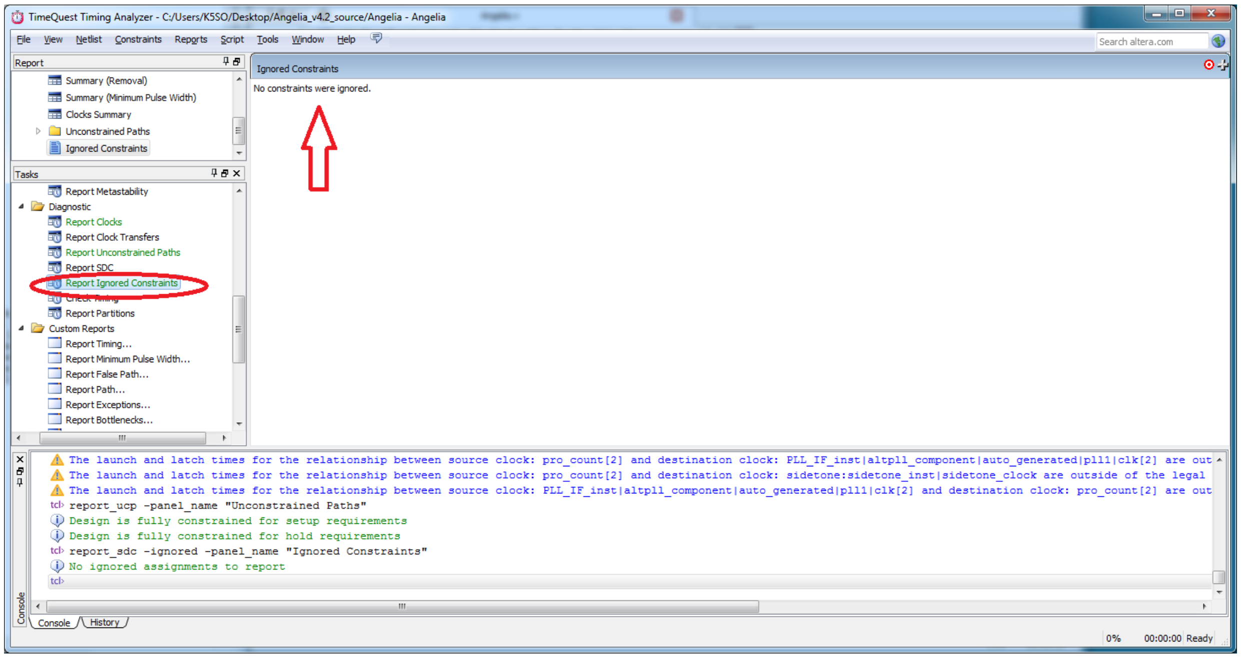
```
Virtual clock virt_PHY_CLK125 is never referenced in any input or output delay assignment.
Virtual clock virt_122MHz_2 is never referenced in any input or output delay assignment.
tcl> qsta_utility::generate_all_summary_tables
Timing requirements not met
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl> report_ucp -panel_name "Unconstrained Paths"
Design is fully constrained for setup requirements
```

Fig. 4. Generate the "Report Unconstrained Paths" report to verify that everything is constrained.

If any unconstrained paths are reported in this report then constrain them before proceeding further with the timing closure task by inserting appropriate constraints in the *.sdc file, as described earlier in this document.

2) verify that no constraints are being ignored

Generate the "Report Ignored Constraints" report to verify that you don't have a syntax error in any of your constraints in the *.sdc file, as shown below:



No constraints were ignored.

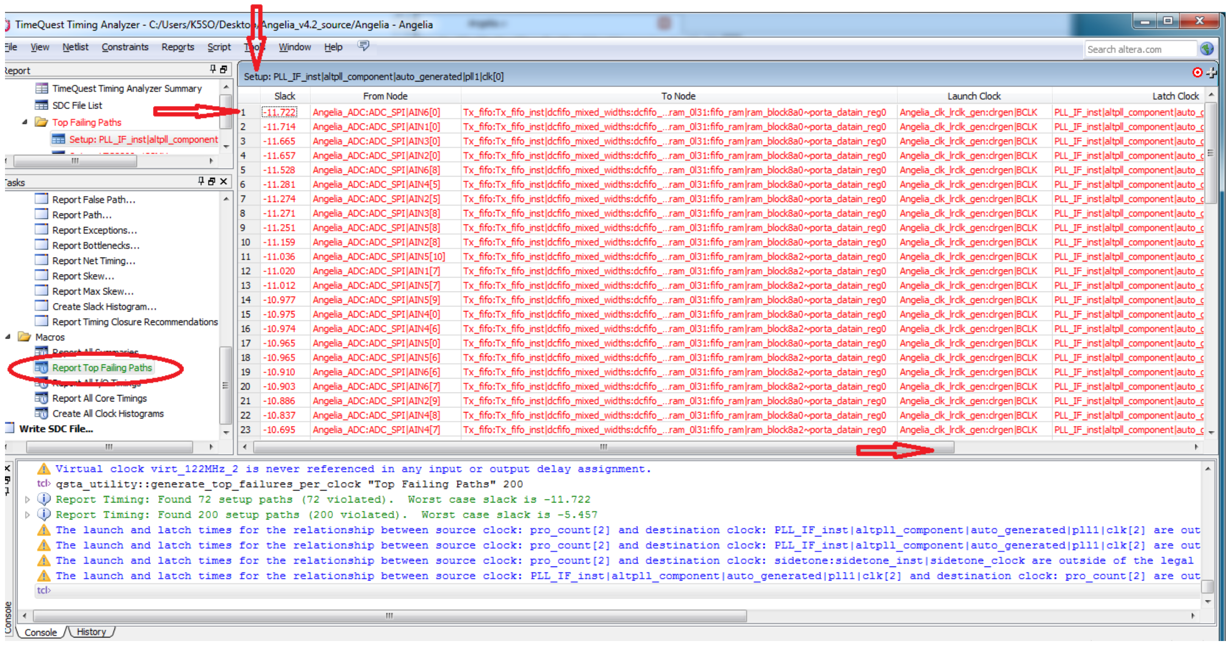
```
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl> report_ucp -panel_name "Unconstrained Paths"
Design is fully constrained for setup requirements
Design is fully constrained for hold requirements
tcl> report_sdc -ignored -panel_name "Ignored Constraints"
No ignored assignments to report
tcl>
```

Fig. 5. Generate the "Report Ignored Constraints" report to ensure you don't have a syntax error in any of your constraints.

If you have errors in your *.sdc file, or if TimeQuest doesn't know what to do with some of your constraints, you will see them listed in this report. If you are unable to keep TimeQuest from ignoring your constraint you may as well remove it from your *.sdc file because it will be ignored anyway and serves no purpose being in the file.

3) examine the worst-case failing paths

Now run the “Report Top Failing Paths” report to examine the details of the worst case failing paths. An example of the report is shown below with numerous failing paths detected:



Setup: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]

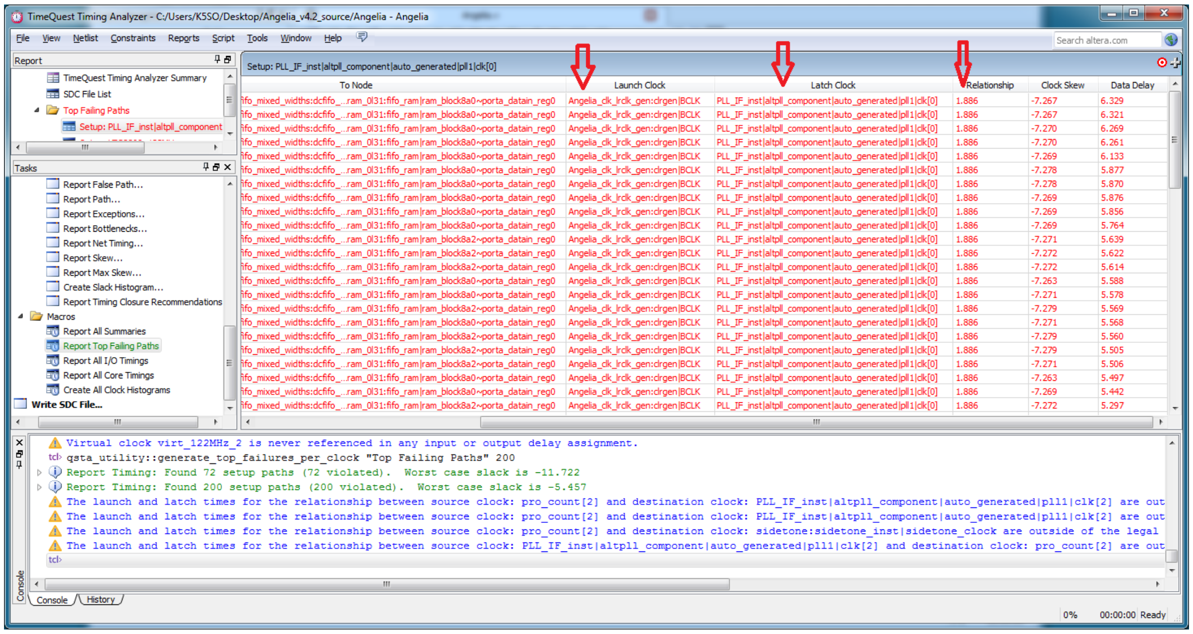
	Slack	From Node	To Node	Launch Clock	Latch Clock
1	-11.722	Angelia_ADC:ADC_SPI|AIN6[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
2	-11.714	Angelia_ADC:ADC_SPI|AIN1[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
3	-11.665	Angelia_ADC:ADC_SPI|AIN3[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
4	-11.657	Angelia_ADC:ADC_SPI|AIN2[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
5	-11.528	Angelia_ADC:ADC_SPI|AIN6[8]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
6	-11.281	Angelia_ADC:ADC_SPI|AIN4[5]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
7	-11.274	Angelia_ADC:ADC_SPI|AIN2[5]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
8	-11.271	Angelia_ADC:ADC_SPI|AIN3[8]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
9	-11.251	Angelia_ADC:ADC_SPI|AIN5[8]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
10	-11.159	Angelia_ADC:ADC_SPI|AIN2[8]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
11	-11.036	Angelia_ADC:ADC_SPI|AIN5[10]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
12	-11.020	Angelia_ADC:ADC_SPI|AIN1[7]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
13	-11.012	Angelia_ADC:ADC_SPI|AIN5[7]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
14	-10.977	Angelia_ADC:ADC_SPI|AIN5[9]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
15	-10.975	Angelia_ADC:ADC_SPI|AIN4[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
16	-10.974	Angelia_ADC:ADC_SPI|AIN4[6]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
17	-10.965	Angelia_ADC:ADC_SPI|AIN5[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
18	-10.965	Angelia_ADC:ADC_SPI|AIN5[6]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
19	-10.910	Angelia_ADC:ADC_SPI|AIN6[6]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
20	-10.903	Angelia_ADC:ADC_SPI|AIN6[7]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
21	-10.886	Angelia_ADC:ADC_SPI|AIN2[9]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
22	-10.837	Angelia_ADC:ADC_SPI|AIN4[8]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c
23	-10.695	Angelia_ADC:ADC_SPI|AIN4[7]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_c

```
Virtual clock virt_122MHz_2 is never referenced in any input or output delay assignment.
tcl> qsta_utility::generate_top_failures_per_clock "Top Failing Paths" 200
Report Timing: Found 72 setup paths (72 violated).  Worst case slack is -11.722
Report Timing: Found 200 setup paths (200 violated).  Worst case slack is -5.457
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl>
```

Fig. 6. Generate the “Report Top Failing Paths” report to see the details of the top failing paths.

In this report you can see that there are numerous failing paths that don’t meet timing. Notice that this report is reporting “setup” timing failures, as indicated by the labeling of the report with the term “Setup:” (red arrow). Next notice that the worst case timing failure is -11.722 nSec (negative slack).

Now we move the display window so that you can see the right edge of the report, as show below:


Setup: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]

To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.267	6.329
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.267	6.321
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.270	6.269
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.270	6.261
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.269	6.133
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.278	5.877
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.278	5.870
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.269	5.876
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.269	5.856
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.269	5.764
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.271	5.639
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.272	5.622
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.272	5.614
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.263	5.588
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.271	5.578
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.279	5.569
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.271	5.568
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.279	5.560
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.279	5.505
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.271	5.506
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.263	5.497
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.269	5.442
fifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a2~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]	1.886	-7.272	5.297

```
Virtual clock virt_122MHz_2 is never referenced in any input or output delay assignment.
tcl> qsta_utility::generate_top_failures_per_clock "Top Failing Paths" 200
Report Timing: Found 72 setup paths (72 violated).  Worst case slack is -11.722
Report Timing: Found 200 setup paths (200 violated).  Worst case slack is -5.457
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] are out
The launch and latch times for the relationship between source clock: pro_count[2] and destination clock: sidetone:sidetone_inst|sidetone_clock are outside of the legal
The launch and latch times for the relationship between source clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2] and destination clock: pro_count[2] are out
tcl>
```

Fig. 7. Observe which clock is launching the signals and which clock is latching the signals and what the "relationship" value is for the worst case failing path.

For this worst case failing path we see that the "Launch Clock" is "Angelia_clk_lrclk_gen:clrgenlBCLK", the "Latch Clock" is "PLL_IF_instl altpll_componentlauto_generatedlpll1lclk[0]", and the "relationship" value is "1.886" nSec. In step 4) we'll use these values to determine a crude timing adjustment value that we should make to resolve this worst case timing failure, which will also resolve all of the the less-worst-case paths shown in the report that use these same clocks at the same time.

4) adjust timing on signal paths of worst-case launch/latch clock domains

For the OpenHPSR firmware projects it appears (from experience) that a reasonable positive slack value to strive for in correcting failing path timing is around 1 nSec positive slack. For this example, a rough estimate for the amount of setup delay that will be needed to achieve a positive slack value near 1 nSec is

$$\begin{aligned} \text{adjustment delay} &\sim \text{desired slack - negative slack + relationship} \\ &= 1 \text{ nSec} - (-11.722 \text{ nSec}) + 1.886 \text{ nSec} = 14.608 \text{ nSec} \\ &\sim 15 \text{ nSec} \end{aligned}$$

Due to the fact that in this example the timing failure is a "setup" timing failure, as opposed to any other type of timing failure, we will resolve it using a *set_max_delay* constraint. (Note: if the timing failure were a "hold" timing failure or a "removal" timing failure we would use a *set_min_delay* constraint to resolve it instead)

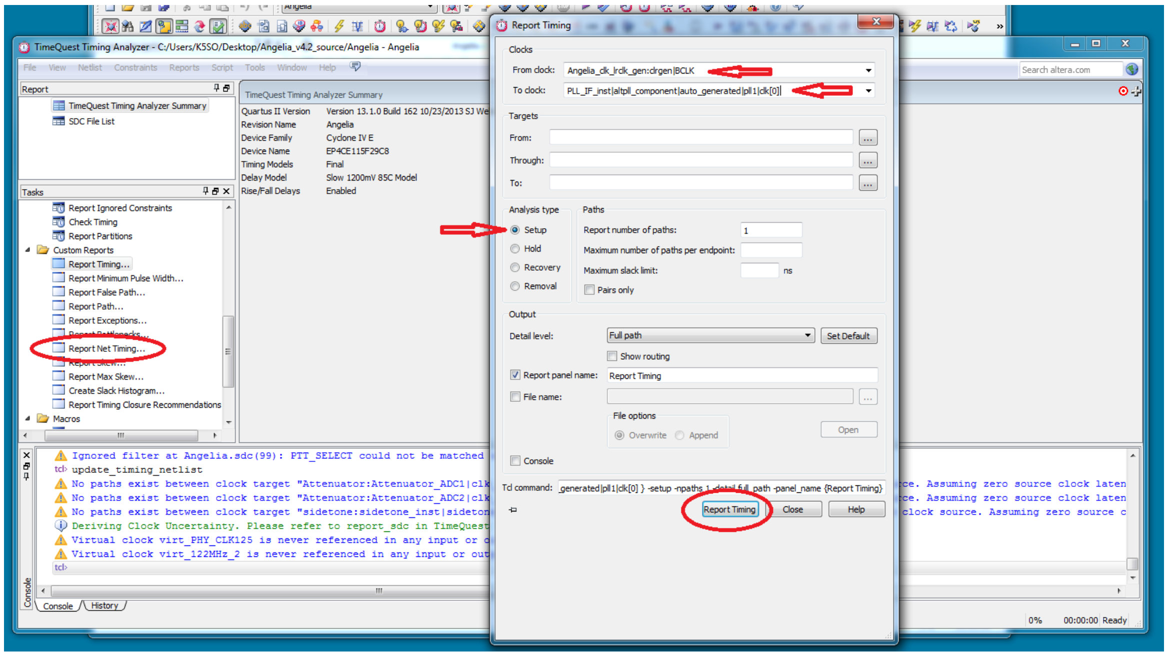
The constraint that we add to the *.sdc file is then

set_max_delay -from Angelia_clk_lrclk_gen:clrgen|BCLK -to PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0] 15

Note especially that this standardized procedure does not advocate attempting to change ALL signals paths to have the 1 nSec positive slack target value described above; on the contrary, this arbitrary positive slack value is suggested as a target value to strive for only on paths that are failing to meet the minimum timing requirements, i.e., those reported by the timing wizard as having negative slack values.

5) verify that the adjustment corrected the issue

When the above constraint is added to the *.sdc file, the file SAVED, and TimeQuest reset by double clicking “Reset Design” in the Tasks window, we can examine the effect our new constraint has on the timing of these paths by using the “Report Timing...” report, as shown below:


TimeQuest Timing Analyzer - C:/Users/K5SO/Desktop/Angelia_v4.2_source/Angelia - Angelia
File View Netlist Constraints Reports Script Tools Window Help
Report
TimeQuest Timing Analyzer Summary
SDC File List
Tasks
Report Ignored Constraints
Check Timing
Report Partitions
Custom Reports
Report Timing...
Report Minimum Pulse Width...
Report False Path...
Report Path...
Report Exceptions...
Report Net Timing...
Report Max Skew...
Create Slack Histogram...
Report Timing Closure Recommendations
Macros
Console
History
0%
00:00:00 Ready
Report Timing
Report Timing
Close
Help
Set Default
Open

TimeQuest Timing Analyzer Summary

Quartus II Version	Version 13.1.0 Build 162 10/23/2013 SJ We
Revision Name	Angelia
Device Family	Cyclone IV E
Device Name	EP4CE115F29C8
Timing Models	Final
Delay Model	Slow 1200mV 85C Model
Rise/Fall Delays	Enabled

Clocks
From clock: Angelia_clk_lrclk_gen:clrgen|BCLK
To clock: PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]

Targets
From:
Through:
To:

Analysis type: [✓ Setup] Hold Recovery Removal

Paths
Report number of paths: 1
Maximum number of paths per endpoint:
Maximum slack limit: ns
☐ Pairs only

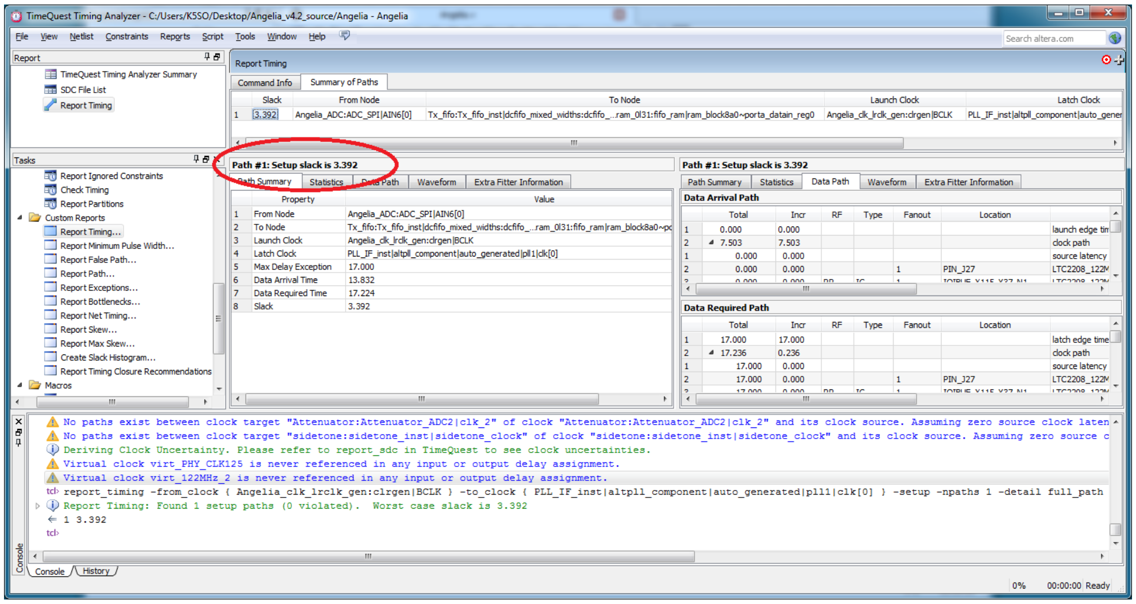
Output
Detail level: Full path
☐ Show routing
☑ Report panel name: Report Timing
☐ File name:
File options: [✓ Overwrite] Append
☐ Console

Tcl command: _generated|pll1|clk[0] } -setup -npaths 1 -detail full_path -panel_name {Report Timing}

```
Ignored filter at Angelia.sdc(99): PTT_SELECT could not be matched
tcl> update_timing_netlist
No paths exist between clock target "Attenuator:Attenuator_ADC1|clk      ce. Assuming zero source clock laten
No paths exist between clock target "Attenuator:Attenuator_ADC2|clk      ce. Assuming zero source clock laten
No paths exist between clock target "sidetone:sidetone_inst|sideton      clock source. Assuming zero source c
Deriving Clock Uncertainty. Please refer to report_sdc in TimeQuest
Virtual clock virt_PHY_CLK125 is never referenced in any input or o
Virtual clock virt_122MHz_2 is never referenced in any input or out
tcl>
```

Fig. 8. TimeQuest displays after requesting a“Report Timing...” report to see what effect the timing adjustment achieved.

Double click on the “Report Timing” button to generate the report, as shown below:



	Slack	From Node	To Node	Launch Clock	Latch Clock
1	3.392	Angelia_ADC:ADC_SPI|AIN6[0]	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~porta_datain_reg0	Angelia_clk_lrclk_gen:clrgen|BCLK	PLL_IF_inst|altpll_component|auto_gener

Path #1: Setup slack is 3.392

	Property	Value
1	From Node	Angelia_ADC:ADC_SPI|AIN6[0]
2	To Node	Tx_fifo:Tx_fifo_inst|dcfifo_mixed_widths:dcfifo_...ram_0l31:fifo_ram|ram_block8a0~pc
3	Launch Clock	Angelia_clk_lrclk_gen:clrgen|BCLK
4	Latch Clock	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]
5	Max Delay Exception	17.000
6	Data Arrival Time	13.832
7	Data Required Time	17.224
8	Slack	3.392

Path #1: Setup slack is 3.392

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge tim
2	7.503	7.503					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_J27	LTC2208_122M

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	17.000	17.000					latch edge time
2	17.236	0.236					clock path
1	17.000	0.000					source latency
2	17.000	0.000			1	PIN_J27	LTC2208_122M

```
No paths exist between clock target "Attenuator:Attenuator_ADC2|clk_2" of clock "Attenuator:Attenuator_ADC2|clk_2" and its clock source. Assuming zero source clock laten
No paths exist between clock target "sidetone:sidetone_inst|sidetone_clock" of clock "sidetone:sidetone_inst|sidetone_clock" and its clock source. Assuming zero source c
Deriving Clock Uncertainty. Please refer to report_sdc in TimeQuest to see clock uncertainties.
Virtual clock virt_PHY_CLK125 is never referenced in any input or output delay assignment.
Virtual clock virt_122MHz_2 is never referenced in any input or output delay assignment.
tcl> report_timing -from_clock { Angelia_clk_lrclk_gen:clrgen|BCLK } -to_clock { PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0] } -setup -npaths 1 -detail full_path
Report Timing: Found 1 setup paths (0 violated).  Worst case slack is 3.392
1 3.392
tcl>
```

*Fig. 9. The "Report Timing" display showing that the set_max_delay constraint we added to the *.sdc file resolved the previous negative slack value, changing the path timing to a positive slack value of 3.3892 nSec instead.*

If desired, the positive slack value can be "tweaked" closer to the (arbitrarily selected)1 nSec positive slack value we were striving for by reducing the delay value specified in the *setup_max_delay* constraint from "15" to "13", which should yield a positive slack value for these paths near 1 nSec instead, if we wish to do that. In reality, the 3.3892 positive slack value is probably workable in the firmware design, in this particular example.

6) repeat from step 3) until no failing paths are found

The above procedure beginning with step 3) is repeated as long as failing paths are reported by the TimeQuest timing wizard.

7) re-compile the firmware design

After recompiling the project the TimeQuest timing report may show a few new failing paths, in which case the seven step procedure should be done again until such time that a fresh compile yields no failing paths. At that point, the timing closure task is completed.

Rules of Thumb

In employing this standardized procedure to close timing in a firmware design a few general "rules of thumb" are suggested to the beginner as being likely advisable to observe. These are:

i) Limit the timing adjustment values to the minimum value necessary to achieve positive slack in the neighborhood of 1 nSec positive slack.

ii) Use only positive values for delays in the *set_max_delay* constraints.

iii) Use only negative values for delays in the *set_min_delay* constraints.

iv) Should you encounter a situation in which timing is successfully closed according to the TimeQuest timing wizard but the code doesn't actually run on some boards, examine the "Report All I/O Timings" report and the "Report All Core Timings" report to see if there are positive slack values for any of those paths that are less than 0.5 nSec. If there are, adjust the timing on those paths such that the positive slack values are between 0.5 nSec and 1 nSec.

v) Should you achieve timing closure and the firmware still doesn't run, re-examine your clock definitions and clock groupings. Each clock group should include only those clocks which are related to each other in the sense that they all originate from the same clock source. Mis-grouping clocks in the *.sdc file can result in firmware that doesn't run on the hardware.

vi) Should the *set_max_delay* and *set_min_delay* constraints become "overly numerous" (say, many dozens of each of them) and you still are unable to achieve timing closure, you may be well advised to simply delete all of them, review your clock definitions, clock groupings, and I/O delay constraints to ensure that they reflect the actual hardware then start the timing closure task fresh again with no *set_max_delay* or *set_min_delay* constraints at all, carefully adding them back, in turn, as the timing wizard reports the failing paths.

vii) Do NOT attempt to change the positive slack values for ALL signal paths in the firmware design to the 1 nSec positive slack value that was suggested as being a suitable target value to strive for when correcting paths that are reported by the timing wizard as failing to meet timing requirements.

As mentioned earlier, it is expected that as the beginner gains more experience with the Altera development tools in working with firmware designs a personalized procedure will naturally evolve that better suits the beginner's personal preferences.

Summary

This document details a standardized iterative procedure for achieving timing closure in OpenHSPDR firmware designs. There is no claim made here that the procedure is infallible nor is there a guarantee that the procedure will always be successful. Nevertheless, the procedure has proven effective for the author in achieving timing closure for the firmware designs on a variety of OpenHPSDR and ANAN-series platforms, including Metis, Ozy, Mercury, Penelope, Hermes, Angelia, and Orion; using a variety of the Altera Cyclone-series FPGAs.

The procedure consists of a seven step iterative process that can be used to adjust timing on signal paths that have been reported by the TimeQuest timing wizard as failing to meet the minimum required timing requirements. It is hoped that the process described in this document can be useful to beginners, in particular, to lessen the perceived complexity of the timing closure process.

It is further hoped that this document will assist in encouraging more people within the OpenHSPDR community to become engaged in working with FPGA firmware designs.

References/Suggested Reading

1. “TimeQuest User Guide” by Ryan Scoville, Wiki Release 1.1, December 9, 2010

2. “TimeQuest Timing Analyzer: Quick Start Tutorial”, Altera document UG-TMQSTANZR-1.1, December 2009.