

Qsys Introduction Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 2. QSYS INTRODUCTION LAB	50
2.1 Set Up the Quartus II Project	51
2.1.1 Create a new Quartus II Project.....	51
2.2 Build the Hardware Design	53
2.2.1 Launch Qsys	53
2.2.2 Build up the rest of the system.....	54
2.2.3 Generate RTL for compilation.....	61
2.3 Running Analysis and Synthesis.....	63
2.3.1 Adding the Qsys system to the Quartus II Project	63
2.4 Timing Constraints	63
2.4.1 Adding Timing Constraints.....	64
2.5 Constrain the device	66
2.5.1 Pin Assignments	66
2.5.2 General Assignments.....	67
2.6 Compiling the Design	69
2.6.2 Download the Configuration File to the DECA board	70
2.7 Testing your design.....	74
2.7.1 Debugging a Qsys Design	74
2.8 System Console - Dashboards	77
2.8.1 Push Button Dashboard.....	77
2.8.2 Temperature Dashboard.....	78

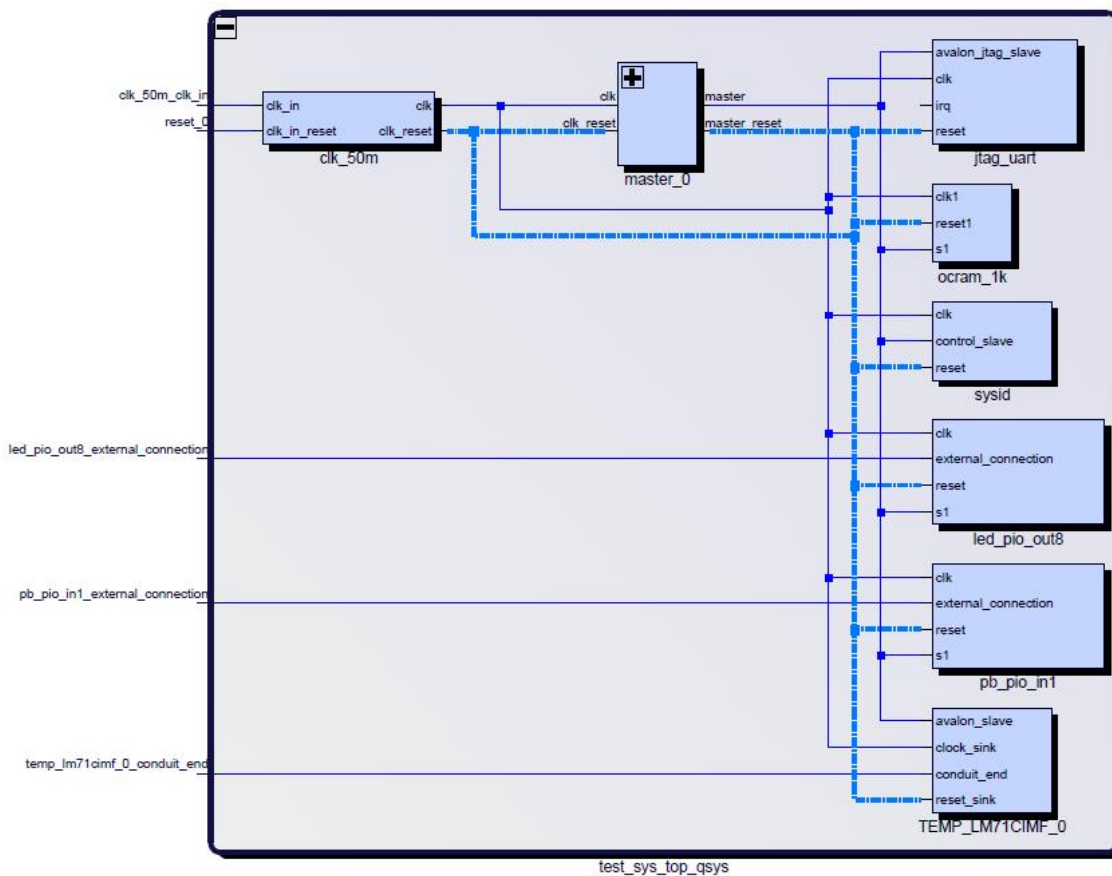
LAB 2. QSYS INTRODUCTION LAB

Overview: In this lab, you will build your first Qsys design to implement a simple bus master used to peek and poke the Qsys peripheral registers.

This lab takes you through a design from the ground up. Steps include:

- Creating a Quartus II project
- Use Qsys to add peripherals to a design
- Add pin and timing constraints
- Compile and program the device
- Launching System Console, and
- Interacting with TCL scripts to adjust register values

The lab will build up a Qsys system that will look like this:

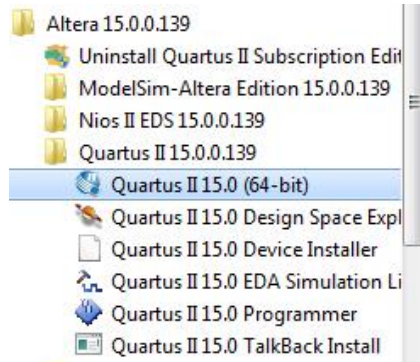


2.1 Set Up the Quartus II Project

In this module, you will create a Quartus II project for your Qsys design.


2.1.1 Create a new Quartus II Project

2.1.1.1 Launch Quartus II 15.0 (64-bit) from the Start menu, if you haven't already



2.1.1.2 Create a new project using the New Project Wizard. Click **File** → **New Project Wizard**

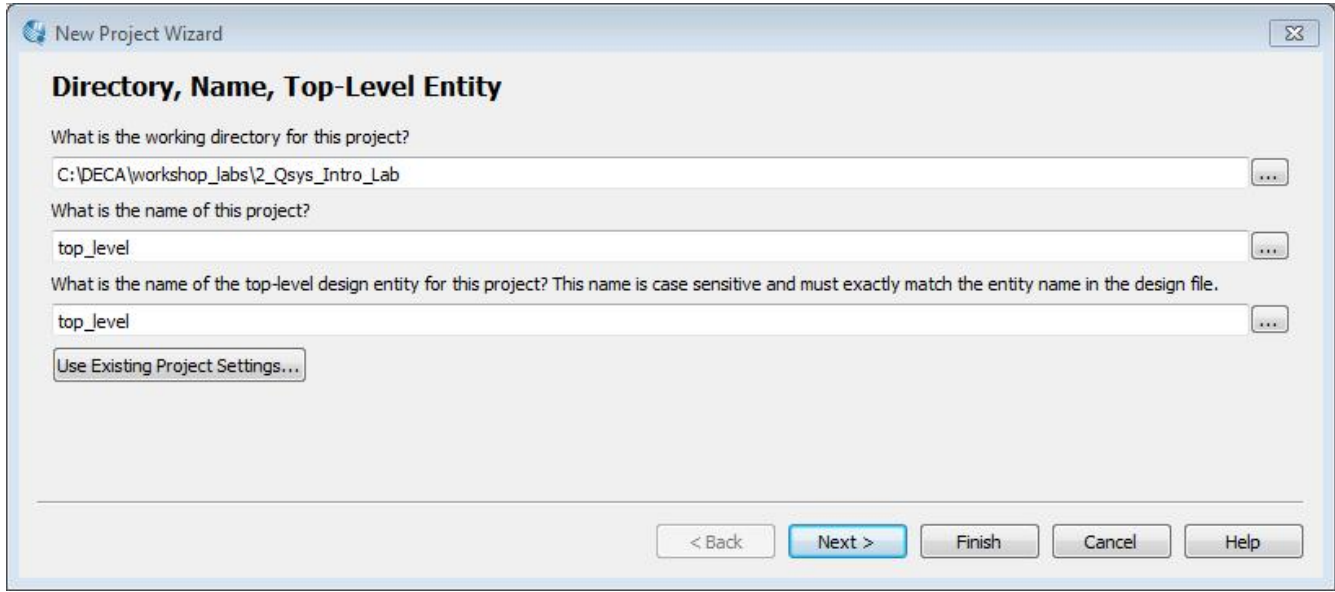
2.1.1.3 Configure the New Project Wizard directory, name, and top-level entity information.

2.1.1.4 Click on the  button and browse to the directory where you extracted the lab files (for example `c:\DECA\workshop_labs\2_Qsys_Intro_Lab`)

2.1.1.5 Specify the name of the project: `top_level`

2.1.1.6 Specify the name of the top level entity: `top_level`

(It is a common naming convention to include the word "top" in the top-level design entity to make it clear and obvious which entity is at the top of the hierarchy.)

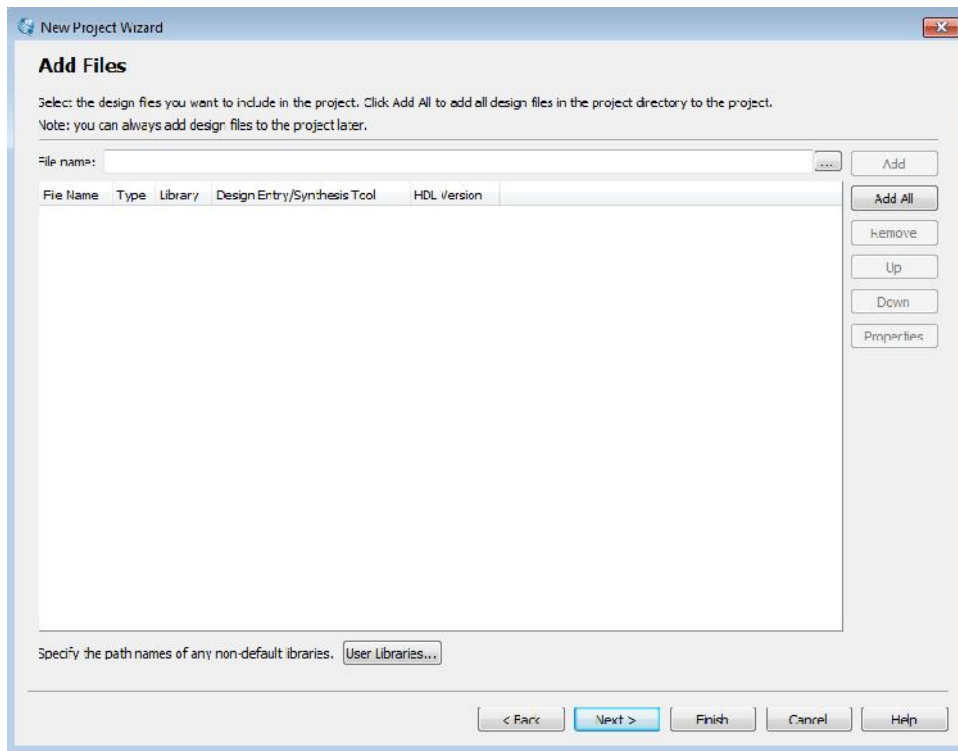


2.1.1.7 Click 

2.1.1.8 On the Project Type page, select "Empty Project" and click 

2.1.1.9 Add source files to the project

The Add Files window will appear. For this lab, new design files will be created so no files will be added. For other designs, files can be added here.

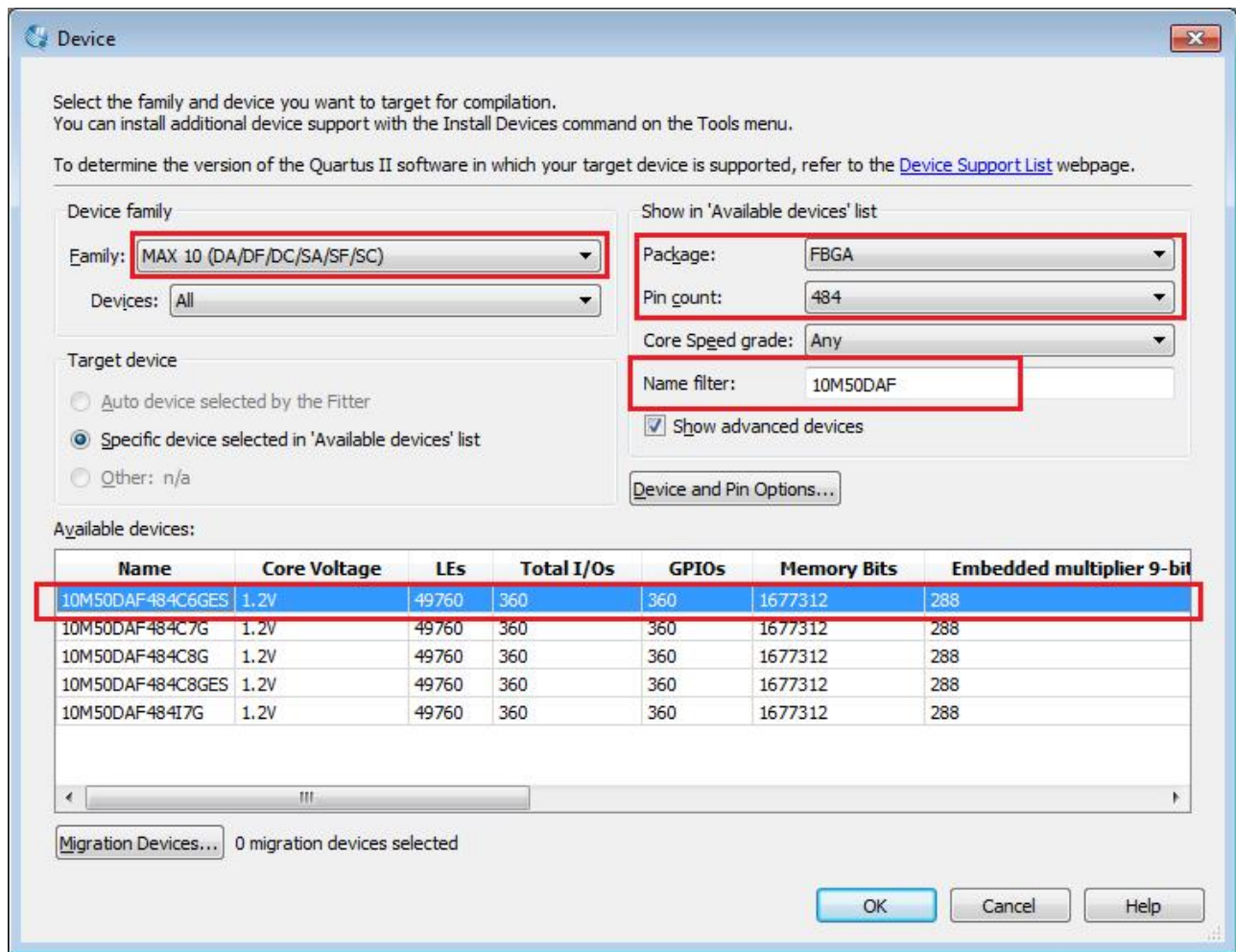


2.1.1.10 Click **Next >**

2.1.1.11 Specify Family and Device Settings

Rather than using the pull down menus to select the correct family, enter the part number in the Name Filter text box.

The part number is **10M50DAF484C6GES**.



After making your selection, look at your kit and confirm that the part number marked on your device matches your selection. Click **Finish**

2.2 Build the Hardware Design

Overview: In this module, you will create and add component to a system, make connections where required, assign clocks and generate the system.

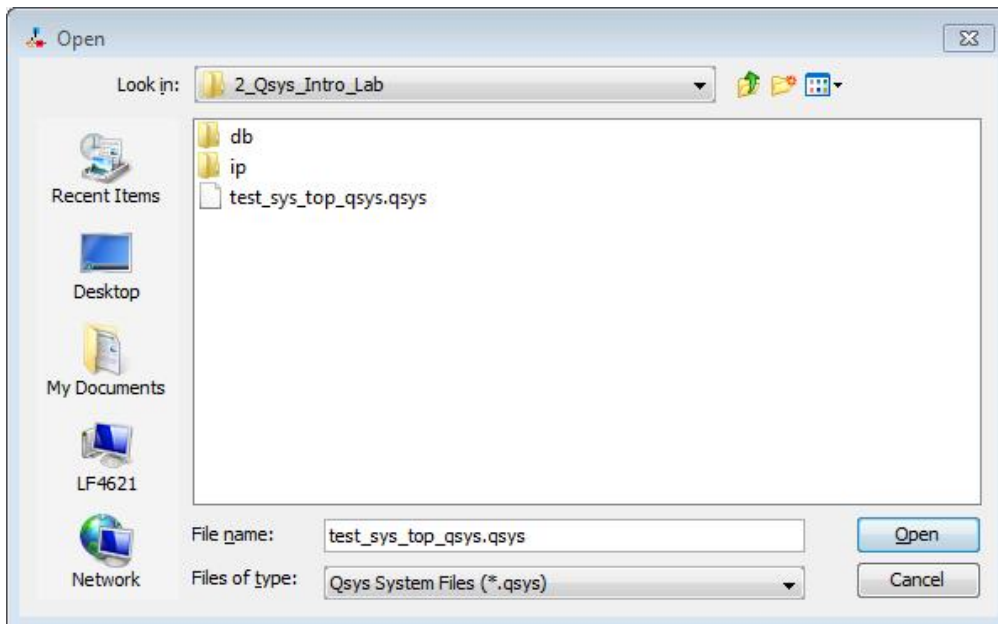
2.2.1 Launch Qsys

Qsys is a high level system integration tool that allows you to quickly build a system using Altera's IP blocks as well as custom components. The tool automatically creates interconnect logic between the components and allows for easy design reuse.

A Qsys system is made up of a number of components and the automatically generated, high performance interconnect between them. Qsys allows you to connect components on an interface level, rather than a signal by signal level. Qsys understands the different types of interfaces and will only allow connections between interfaces of the same type (i.e. a data master connects to a data slave, clock source to clock sink, etc...).

2.2.1.1 From the Tools menu in the main Quartus II window, select Qsys (**Tools → Qsys**).

2.2.1.2 After Qsys launches, open the file named: `test_sys_top_qsys.qsys`



Select Close once the System is completed successfully.

2.2.2 Build up the rest of the system

2.2.2.1 There will be various components that are already included in the Qsys system, while others will need to be added and configured. The Qsys system will look like this:

The screenshot shows the Qsys System Contents window. On the left is a hierarchical tree of components. On the right is a table listing the components and their properties.

Use	Component	Name	Description	Export	Clock	Base	Addr	IRQ	Tags
<input checked="" type="checkbox"/>	clk_50m	clk_50m	Clock Source						
		clk_in	Clock Input	clk_50m_clk_in	exported				
		clk_in_reset	Reset Input	reset_0					
		clk	Clock Output	clk_50m					
		clk_reset	Reset Output	clk_50m					
<input checked="" type="checkbox"/>	master_0	master_0	JTAG to Avalon Master Bridge						
		clk	Clock Input	clk_50m					
		clk_reset	Reset Input	clk_50m					
		master	Avalon Memory Mapped Master	clk_50m					
		master_reset	Reset Output	clk_50m					
<input checked="" type="checkbox"/>	ocram_1k	ocram_1k	On-Chip Memory (SRAM or ROM)						
		clk	Clock Input	clk_50m					
		s1	Avalon Memory Mapped Slave	clk_50m		# 0x0000_0000	0x0000_03FF		
		reset	Reset Input	clk_50m					
<input checked="" type="checkbox"/>	sysid	sysid	System ID Peripheral						
		clk	Clock Input	clk_50m					
		reset	Reset Input	clk_50m					
		control_slave	Avalon Memory Mapped Slave	clk_50m		# 0x0000_0400	0x0000_041F		
<input checked="" type="checkbox"/>	jtag_uart	jtag_uart	JTAG UART						
		clk	Clock Input	clk_50m					
		reset	Reset Input	clk_50m					
		avalon_slave	Avalon Memory Mapped Slave	clk_50m		# 0x0000_0440	0x0000_044F		

Below the table is a Messages window with the following content:

- 1 Warning**
 - The top sender `jtag_uart_irq` is not connected to any interrupt devices. System: `jtag_irq`
- 2 Info Messages**
 - System ID is not assigned automatically. Edit the System ID parameter to provide a unique ID. System: `sysid`
 - Time stamp will be automatically updated when the component is generated. System: `sysid`

There are 5 components already connected as seen above.

The Clock Source IP (`clk_50m`) is the clock source connected to our 50 MHz clock input coming into the FPGA (`MAX10_CLK1_50`).

The JTAG to Avalon Master Bridge (`master_0`) enables you to read and write to memory-mapped slaves connected to the bridge using your JTAG connection via the USB-Blaster II.

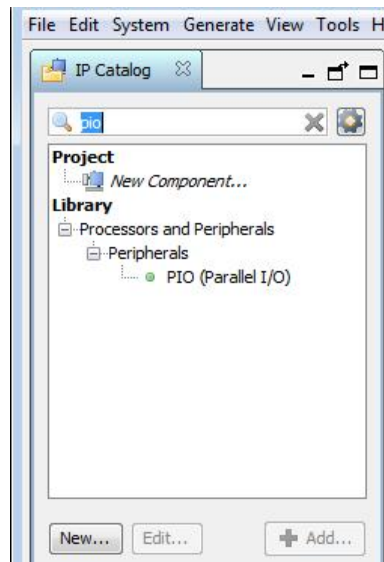
The On-Chip Memory (`ocram_1k`) is a memory mapped slave peripheral instantiated with 1KB of ram (using just one MAX@10 memory block)


The System ID peripheral (`sysid`) is an IP that provides the Qsys system with a unique ID. It is an important peripheral to include when targeting soft and/or hard processors in the Qsys system because it allows the software development tools to validate that the software application is built for the correct hardware system. Basically, it will not allow software to be executed on an incompatible hardware configuration.

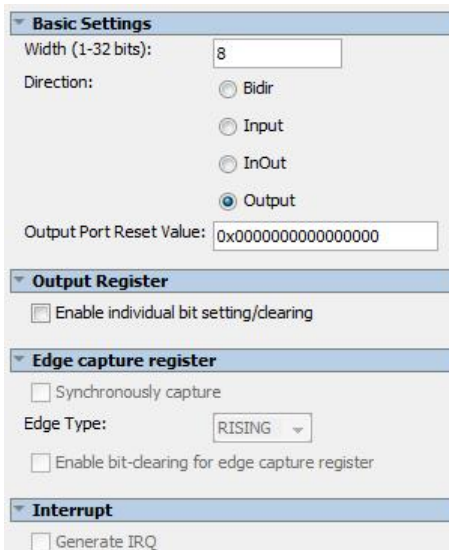
JTAG UART (`jtag_uart`) is a JTAG-based UART that processors use to print characters back to the console

2.2.2.2 Configure and add LED's


In Qsys, under the Library section of the IP Catalog, type: `pio` such as:



Double-click the PIO (Parallel I/O) (or use the  button). A dialog box to configure the component will open. There are eight LED's so set the width to 8 and set the direction to Output

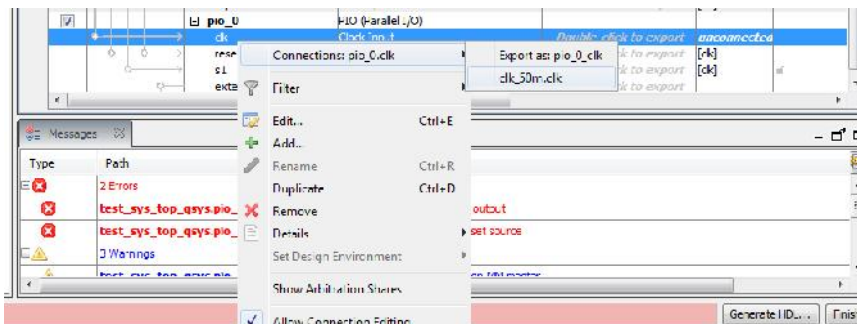


Select Finish. Your newly created PIO component will look like:

<input checked="" type="checkbox"/>		pio_0 clk reset s1 external_connection	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	unconnected [clk] [clk]
-------------------------------------	--	---	---	--	--------------------------------------

There will be errors related to the clock and reset, where the signals are not connected. These will be resolved in the following steps.

Starting with clk, right-click the clk in the pio_0 peripheral:



Select `clk_50m.clk`. The patch panel of the PIO will now be connected to the clock input. To complete the peripheral connections, repeat the step but this time:

- Connect the `pio_0.reset` to `clk50m.clk_reset`
- Connect the `pio_0.reset` to the `master_0.master_reset`

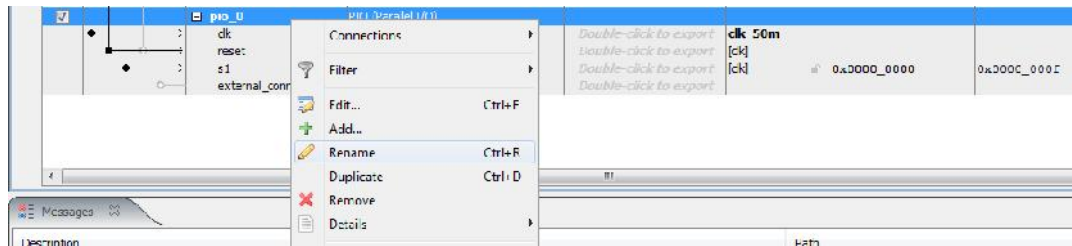
Connecting the reset to both the `clk` and `master` reset domains allow this pio to be reset if either reset occurs.

- Connect the `pio_0.s1` to `master_0_master`.

There will be an error that states:



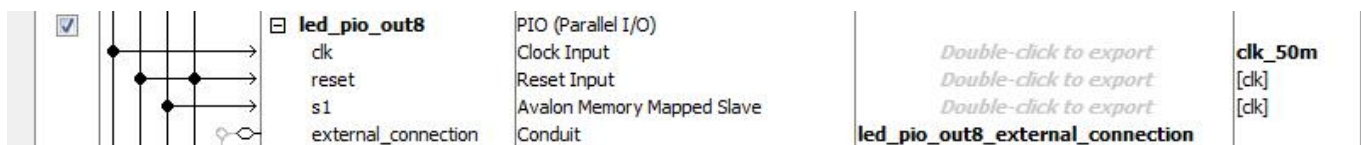
This error will be removed later on. The next step is to change the name of the component. To change the name, right click on the `pio_0` and select Rename:



Change the name to be `led_pio_out8`.

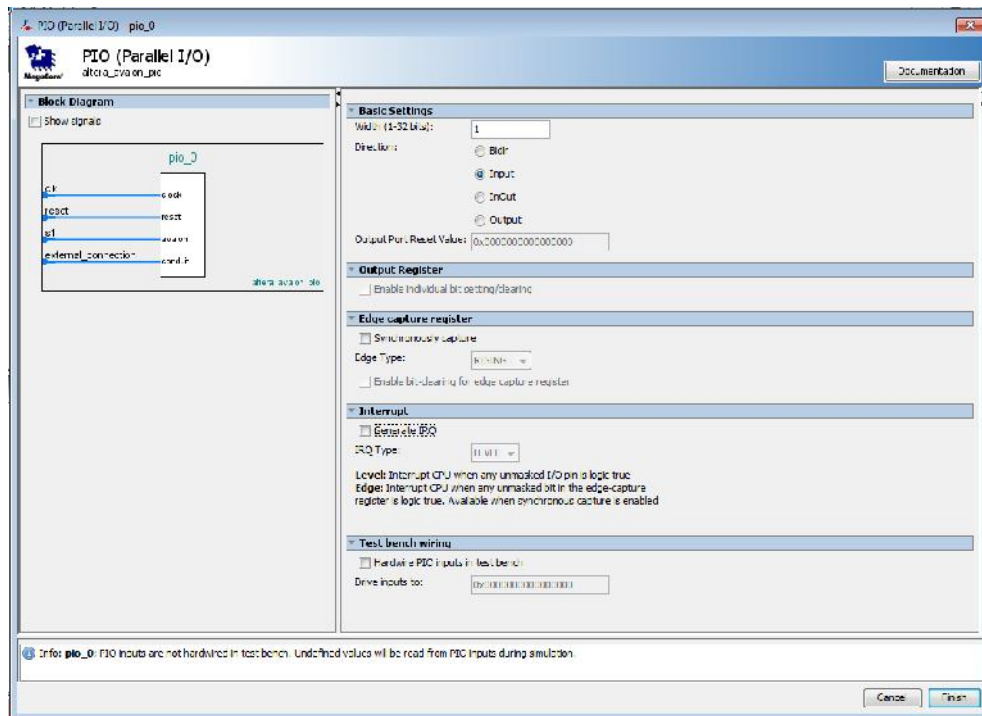
The next step is to export the led signals to the top-level design so that we can connect these signals to FPGA pins. To do this, double-click on the export signal. The name of `led_pio_out_8_external_connection` should now be present.

The completed peripheral should now look like this:



2.2.2.3 Configure and add push-button IO

Repeating the same procedure, we will add a PIO but this time set it up as an input with a width of one (1)



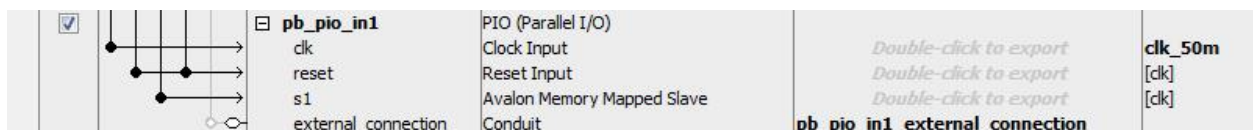
Select Finish to complete the generation of the new pio peripheral.

Similar to the led pio, you need to connect the pio.clk, pio.reset, and pio.s1 ports.

You will encounter more overlapping memory address errors that will be addressed later.

Right-click and change the pio_0 name to **pb_pio_in1**.

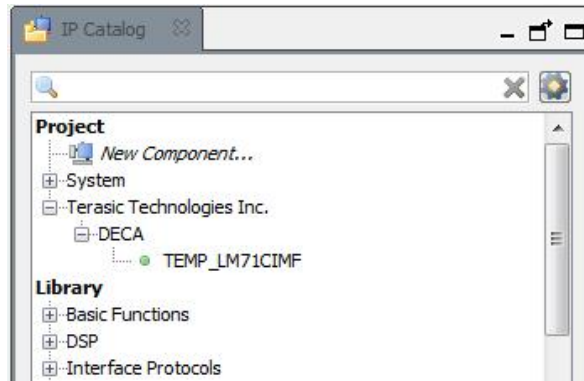
Lastly, we need to export the pio to the top-level design. Double click on the export signal. The signal of **pb_pio_in1_export_connection** should automatically be created. The completed pb_pio_in1 peripheral is shown here:



2.2.2.4 Configure and add the temperature sensor

The LM71 temperature sensor is mounted in the 'hot' area of the board and connects to the FPGA via a SPI-like interface. Terasic has created a custom component to interface to this temperature sensor.

In the library section of the Qsys window, expand the Terasic Technologies Inc. → DECA (Delete any text in the IP Catalog search field to see this IP)

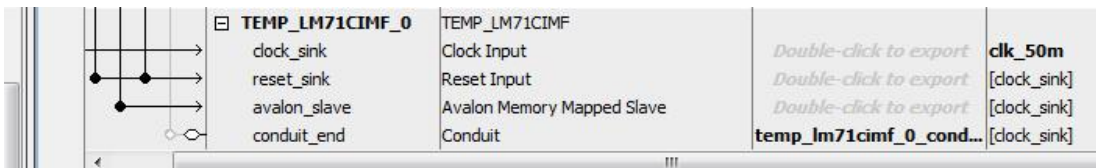


Select the TEMP_LM71CIMF and select Add... then Finish

The next step, similar to other peripherals, is to connect the clk, reset, and control port. Connect these signals:

- Temp_LM71CIMF_0.clock_sink → clk_50m.clk
- Temp_LM71CIMF_0.reset_sink → clk_50m_clk.reset
- Temp_LM71CIMF_0.reset_sink → master_0_master_reset
- Temp_LM71CIMF_0.avalon_slave → master_0.master
- Export the interface to the top-level. Double-click to export the signals and select the default name

The LM71 peripheral will look like this:

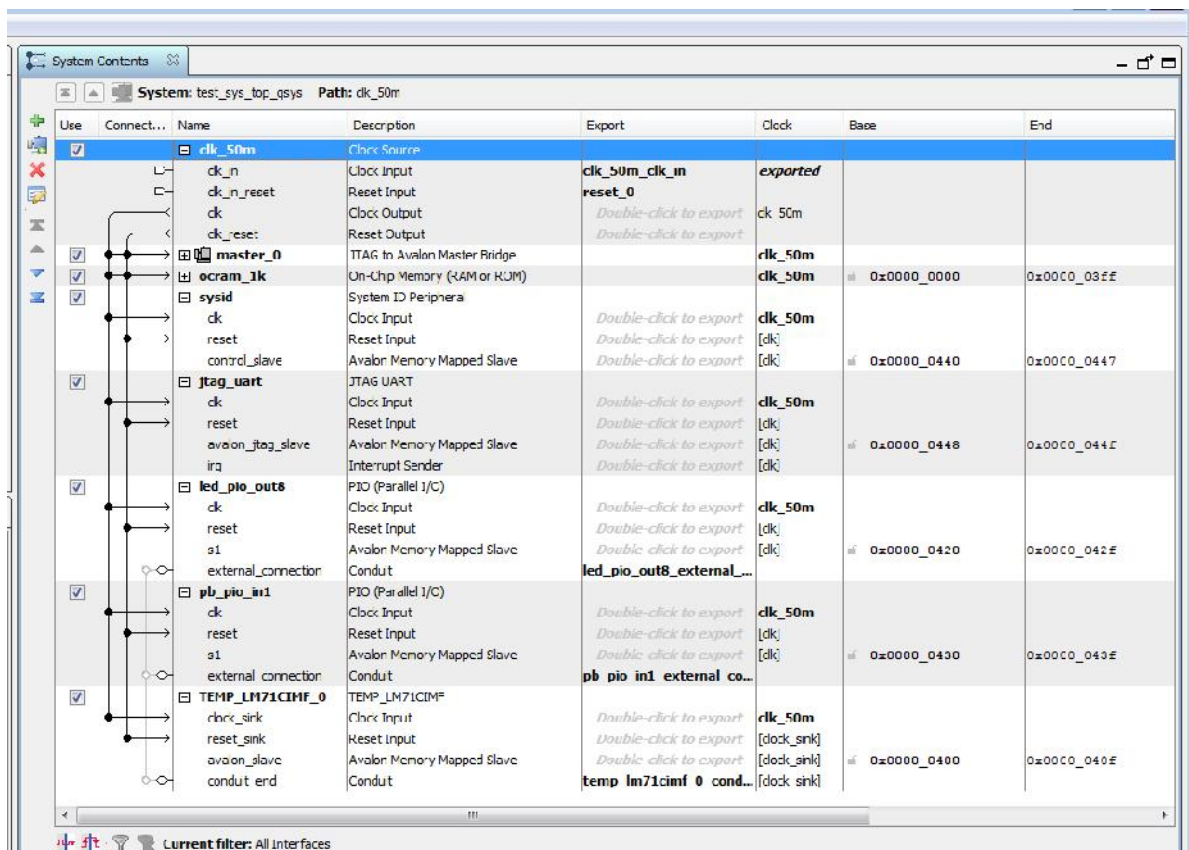


Note that there will be an error about the overlapping peripherals in the memory space. This will be resolved next.

2.2.2.5 Resets and warnings

There will be errors that the master has peripheral components that overlap. If you inspect the system closely, you will find that multiple peripherals share the same physical address.

To resolve the errors, select on the base address of the components to manually enter the addresses required for the lab. Ensure you follow the diagram **exactly**. You only need to enter the Base address, the End address is calculated automatically.



Use	Connect...	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk_50m	Clock Source				
		clk_n	Clock Input	clk_50m_clk_in	exported		
		clk_n_reset	Reset Input	reset_0			
		ck	Clock Output	Double-click to export	clk_50m		
		ck_reset	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		master_0	ITAG to Avalon Master Bridge		clk_50m		
<input checked="" type="checkbox"/>		ocram_1k	On-Chip Memory (RAM or ROM)		clk_50m	0x0000_0000	0x00C0_03FF
<input checked="" type="checkbox"/>		sysid	System ID Peripheral				
		ck	Clock Input	Double-click to export	clk_50m		
		reset	Reset Input	Double-click to export	[clk]		
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0440	0x00C0_0447
<input checked="" type="checkbox"/>		jtag_uart	JTAG UART				
		ck	Clock Input	Double-click to export	clk_50m		
		reset	Reset Input	Double-click to export	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0448	0x00C0_044E
		ira	Interrupt Sender	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		led_pio_outs	PIO (Parallel I/O)				
		ck	Clock Input	Double-click to export	clk_50m		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0420	0x00C0_042F
		external_connector	Conduit	led_pio_outs_external...			
<input checked="" type="checkbox"/>		pb_pio_in1	PIO (Parallel I/O)				
		ck	Clock Input	Double-click to export	clk_50m		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0430	0x00C0_043F
		external_connector	Conduit	pb_pio_in1_external co...			
<input checked="" type="checkbox"/>		TEMP_LM71CIMF_0	TEMP_LM71CIMF				
		clock_sink	Clock Input	Double-click to export	clk_50m		
		reset_sink	Reset Input	Double-click to export	[clock_sink]		
		avalon_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_sink]	0x0000_0400	0x00C0_040F
		conduit_end	Conduit	temp_lm71cimf_0 cond...	[clock_sink]		

Qsys has the ability to auto-assign base addresses (via the System menu). The reason we are manually entering these addresses is to ensure that the system is setup properly to correspond with our lab files that use these specific memory addresses to interact with the peripherals.

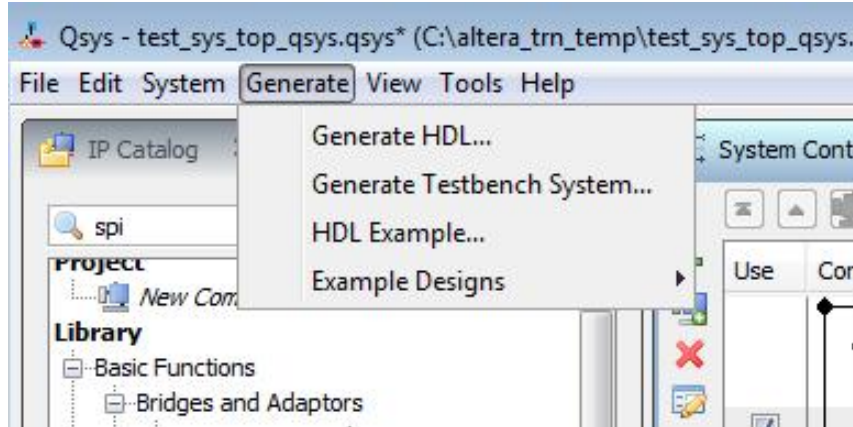
Should you have accidentally selected interrupts in the components, you may see warnings that irq (interrupts) are not connected to interrupt receiver. For this workshop, interrupts are not used. However, for other designs, interrupts can be implemented.

Double-check that the names of the peripherals match the above diagram and that the addresses match the diagram too.

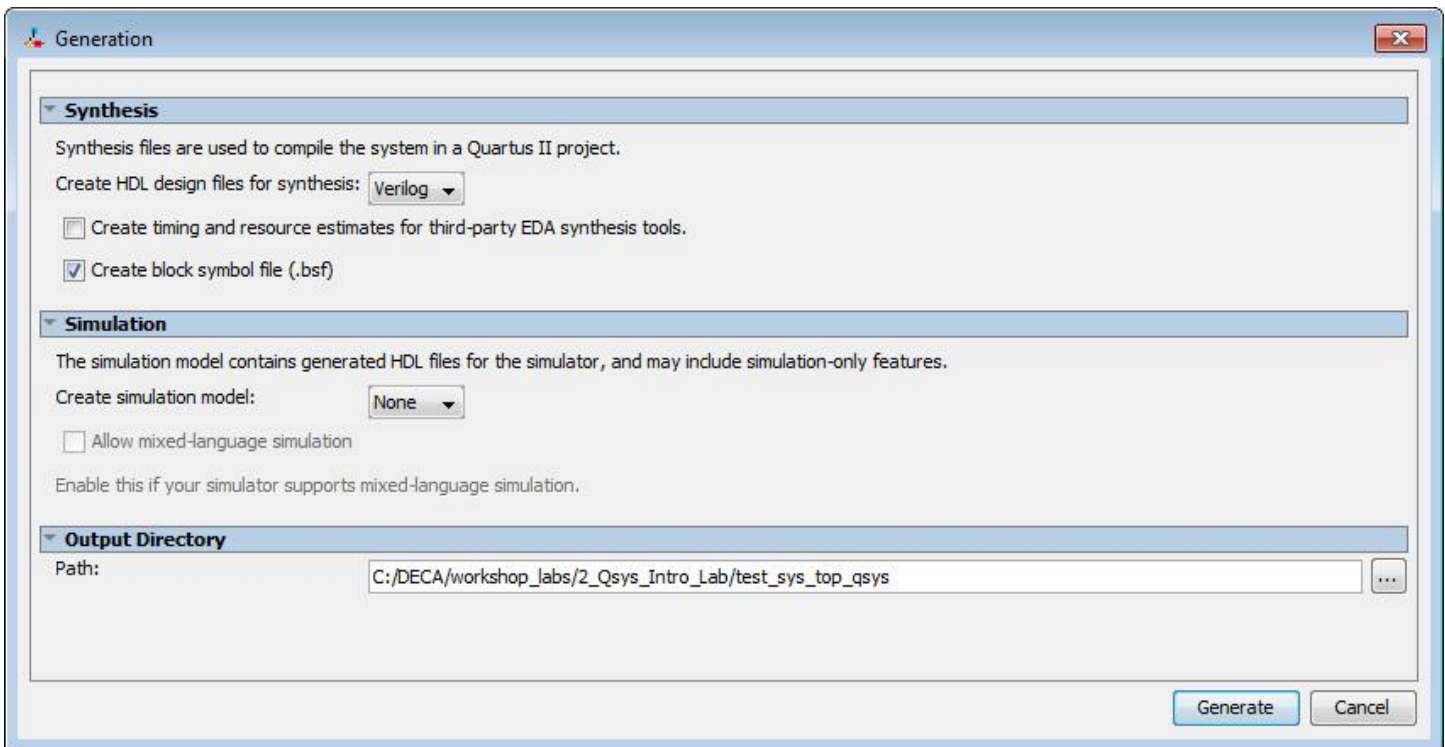
2.2.3 Generate RTL for compilation

This next step converts the Qsys system into code that Quartus can place and route during compilation. The RTL (register transfer level) can be generated in either VHDL or Verilog. Upon completion, these files are used by Quartus to place and route the design using the IP created by Qsys.

2.2.3.1 To do this from the Qsys menu, select **Generate** → **Generate HDL** as below:



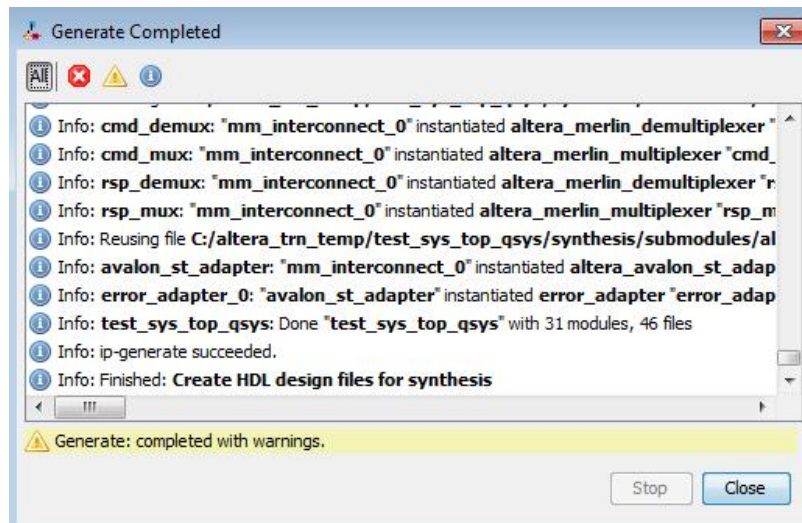
2.2.3.2 A new window, Generation, will appear. Select Verilog for the Create HDL design files for synthesis.



The output files generated by Qsys will default to the path shown in the GUI above. Keep the path the same as shown. By using a sub-directory, the files are neatly organized for use by Quartus. The directory above will be used in the next step.

2.2.3.3 Select Generate

After Generation is complete, you should have no errors.



If there are errors, they will need to be resolved before proceeding to the next step.

2.2.3.4 Select Close.


2.2.3.5 Close Qsys

2.3 Running Analysis and Synthesis

2.3.1 Adding the Qsys system to the Quartus II Project

The system created in Qsys now needs to be added to your Quartus project so that it can be instantiated in the top-level design file. You can think of the Qsys system as a module or component as you would in any other FPGA design. Qsys generates IP "pointer" files for both synthesis (.qip) and simulation (.sip) that will point Quartus to all the necessary design files needed to synthesize or simulate the Qsys system.

2.3.1.1 Within Quartus, select **Project** → **Add/Remove Files in Project** from the Quartus II menu.

2.3.1.2 Click the browse button  to the synthesis directory noted above (it should be `c:\DECA\workshop_labs\2_Qsys_Intro_Lab\test_sys_top_qsys\synthesis\`) and select `test_sys_top_qsys.qip`.

2.3.1.3 Click "Add" to add the .qip file to the project. Click "Apply" and "OK".

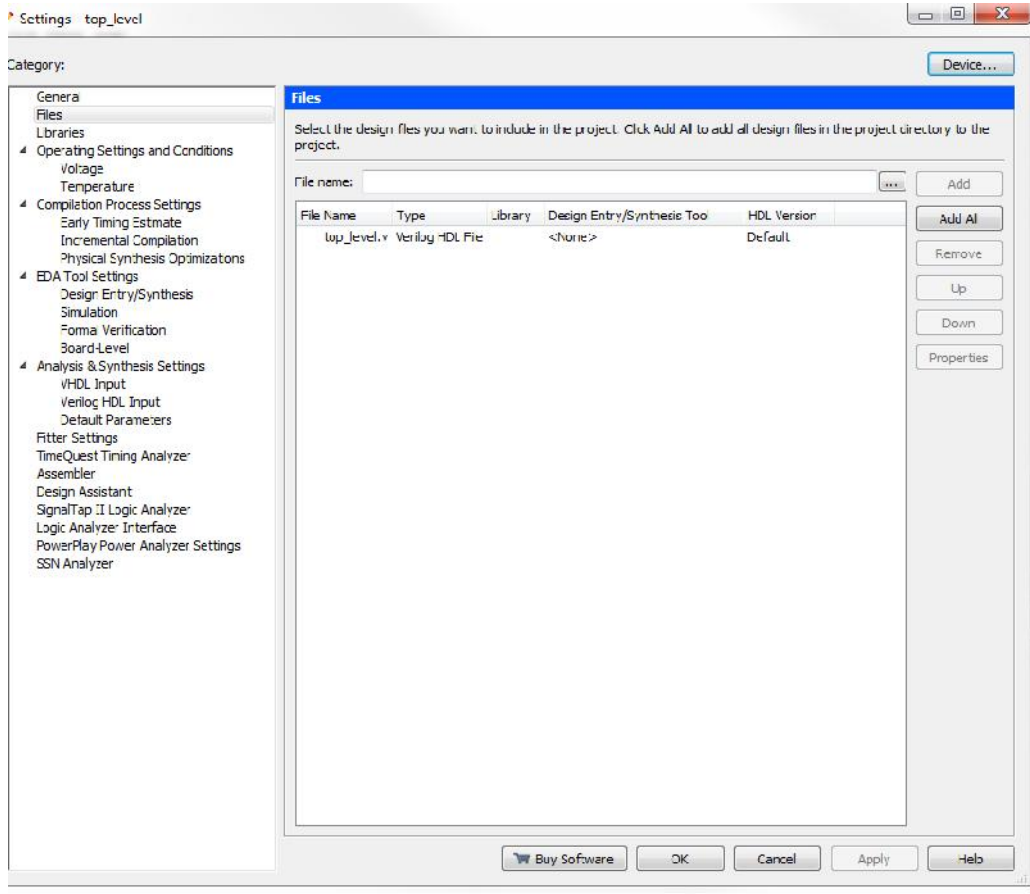
2.4 Timing Constraints

Timing Constraints tell the software of what the timing (eg clock fmax, etc) should be for this design. The timing constraints have already been created for this design, and they are in the lab file, `top_level.sdc`. SDC file is a Synopsys design constraint file.

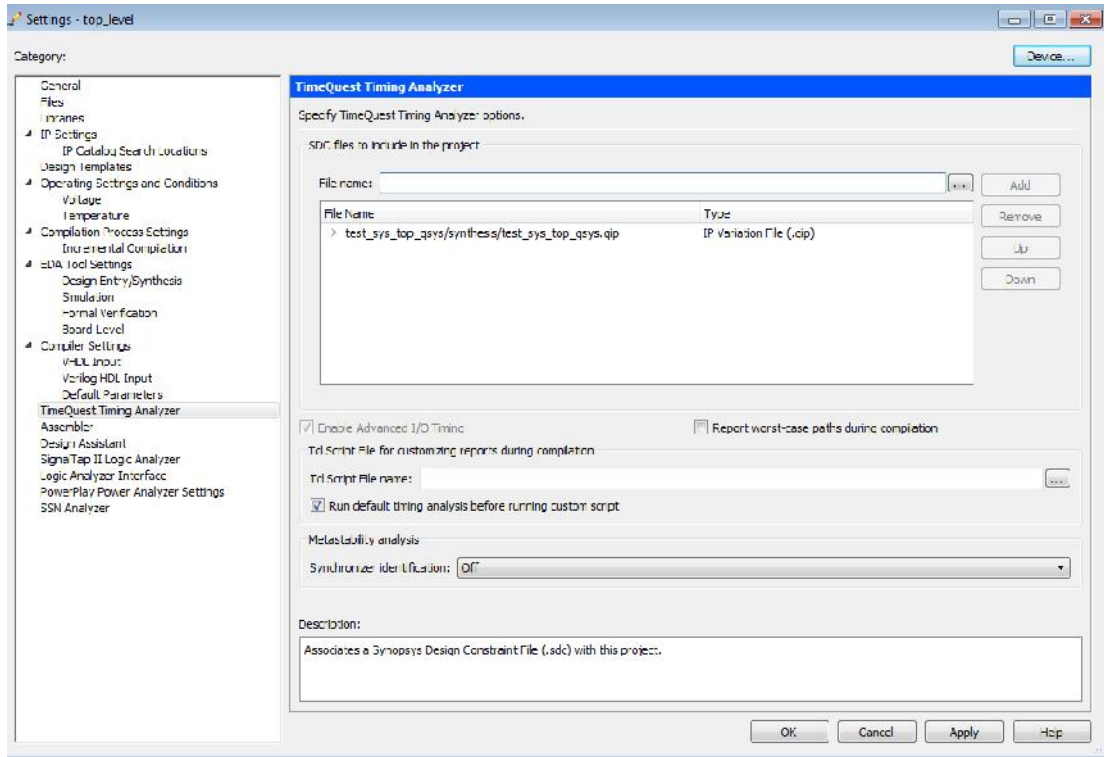
2.4.1 Adding Timing Constraints

2.4.1.1 Add/Remove SDC file

Within Quartus II, select Project - Add/Remove files in Project.

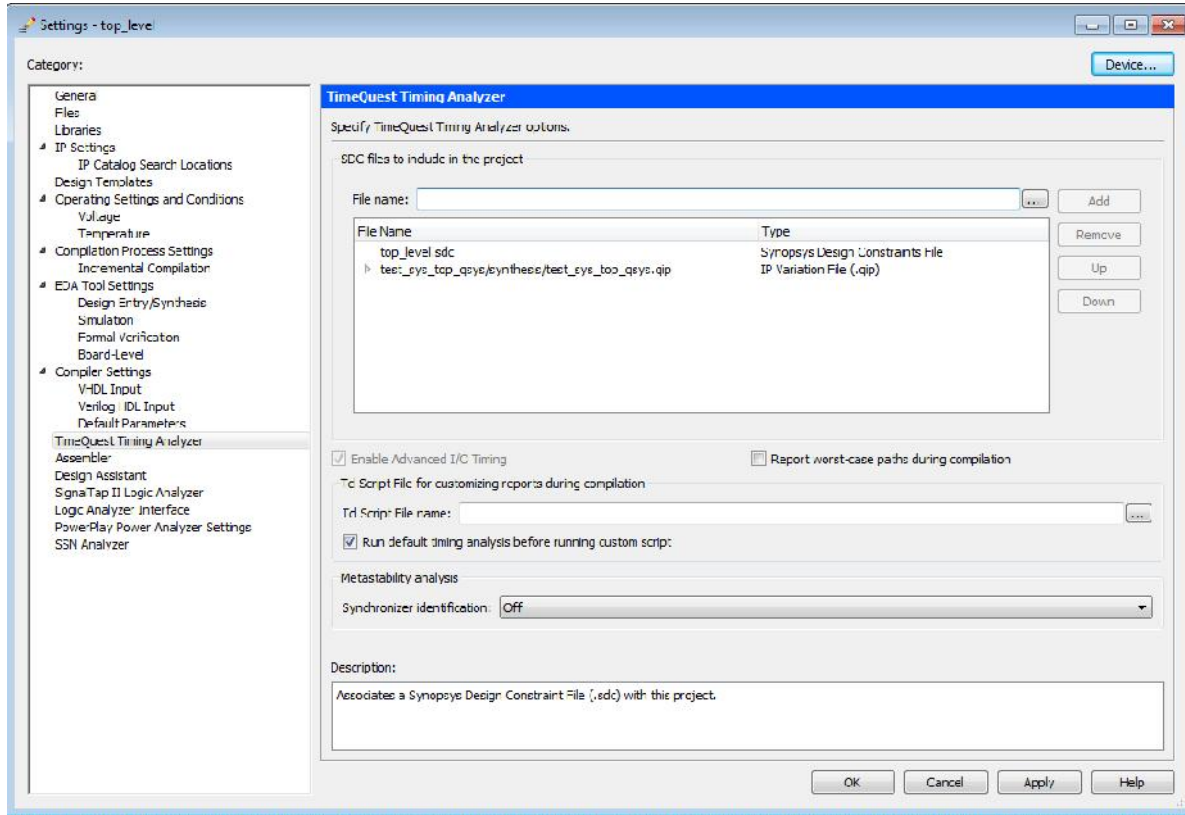


Select the TimeQuest Timing Analyzer on the left hand side of the window.



Using the Browse button, select the top_level.sdc file. (You may need to scroll up or down a directory path).

Select Add. Confirm the file was added to the file list.



Select Apply.

Select OK.

2.5 Constrain the device

2.5.1 Pin Assignments

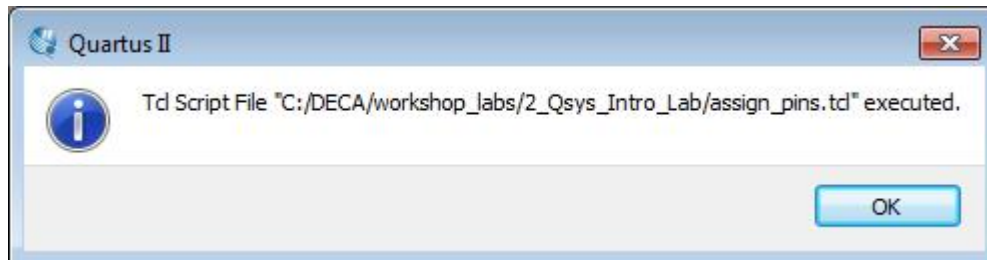
Before the design can be downloaded, pin assignments that match the hardware on the board are needed. There are different ways to do this, such as using Pin Planner, Assignment Editor, and other methods. In this lab a .tcl (script) file is used, because there are many pin locations as I/O assignments.

2.5.1.1 Within Quartus II, select **Tools** → **TCL Scripts**

2.5.1.2 Select `assign_pins.tcl`

(The other .tcl files are used for other steps in this workshop)

2.5.1.3 Select Run. A message window should appear that states:



2.5.1.4 Click OK to close this message window

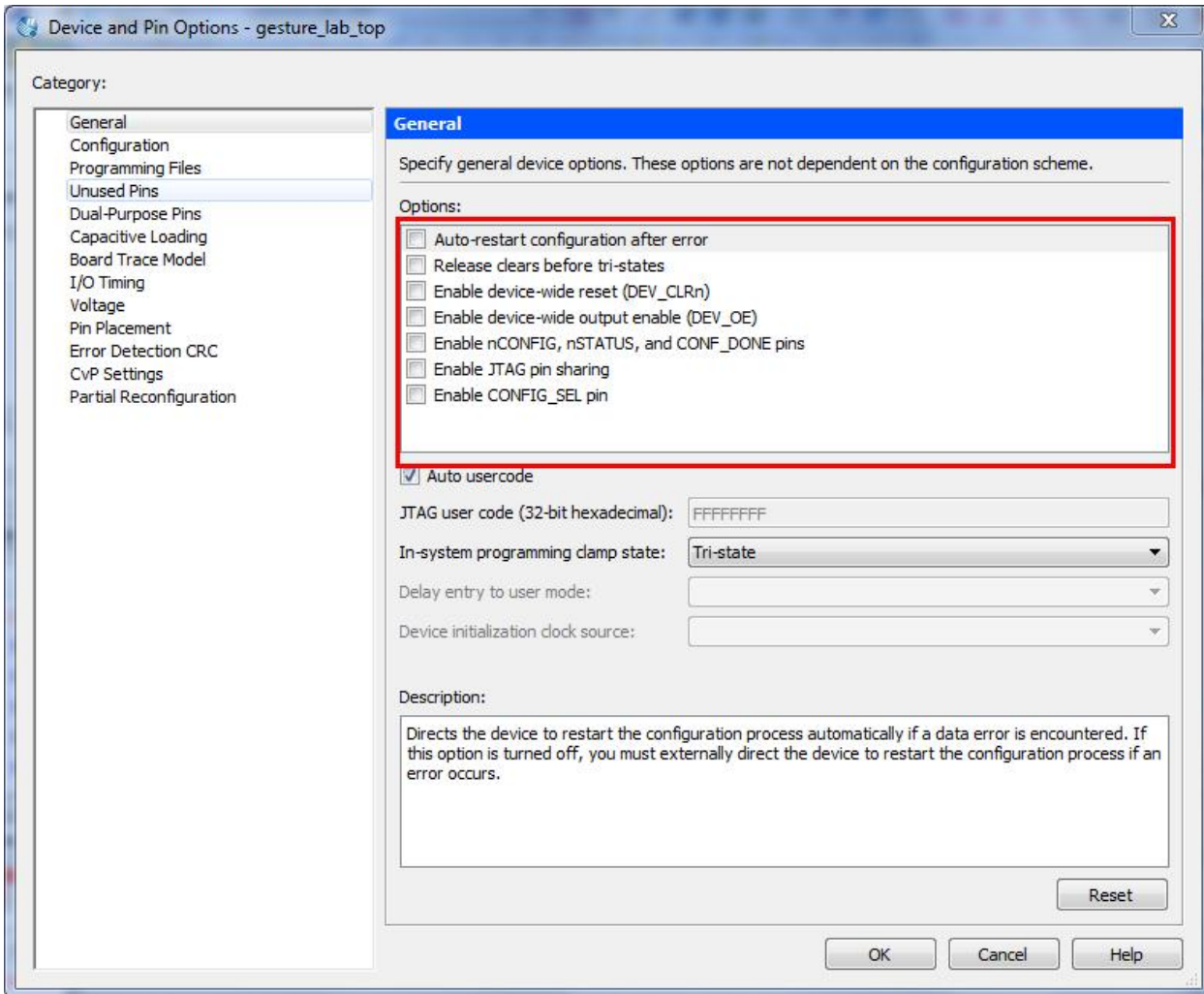
2.5.1.5 Click OK to close the TCL script window.

2.5.2 General Assignments

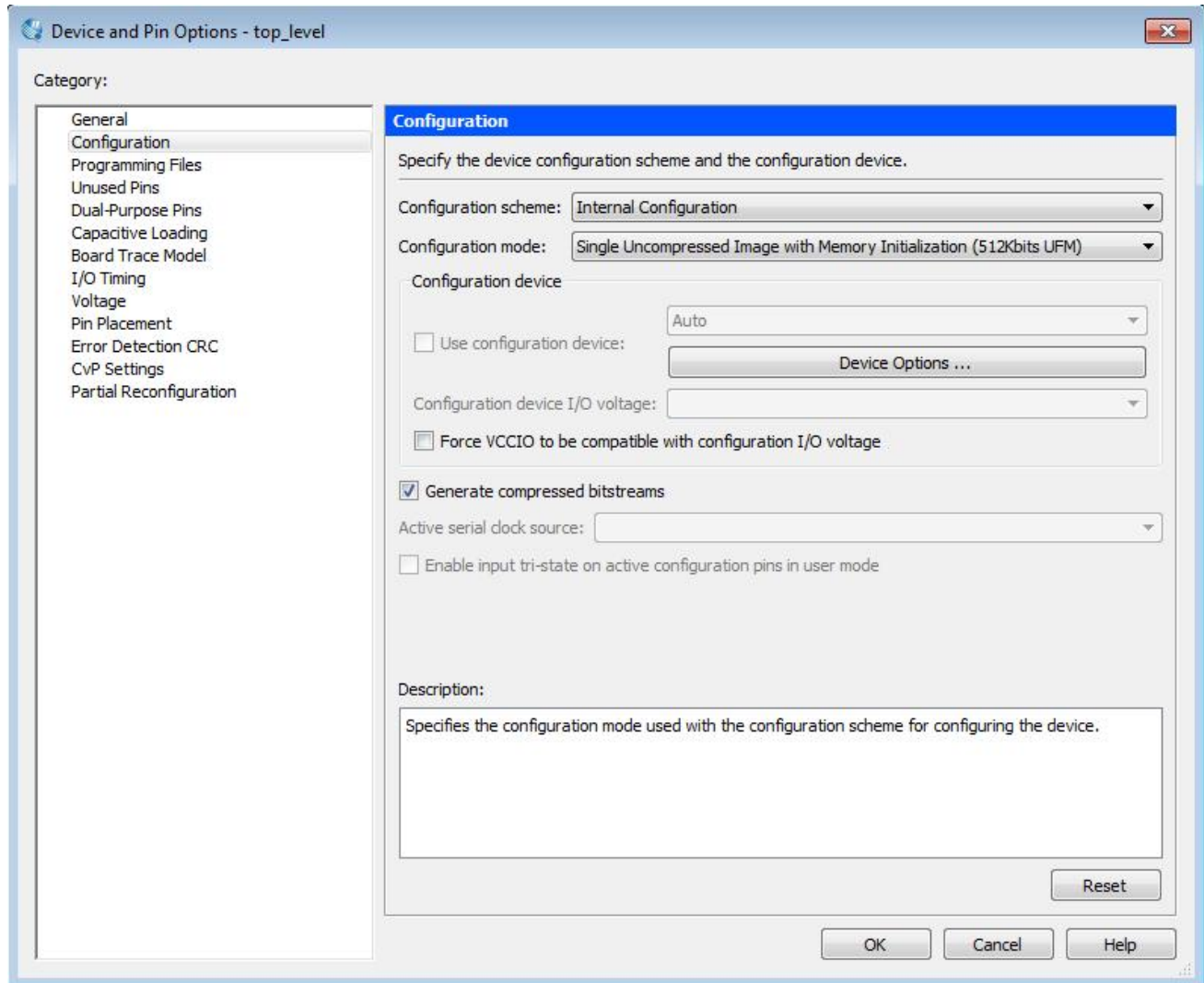
With the hardware design complete, a few device settings need to be changed and the project can be compiled to create a configuration file. Specifically, some optional configuration pins need to be disabled to match the DECA board hardware. If this step is not completed, the VCCIO required by those optional pins will not match the VCCIO of other pins of the DECA board, and Quartus will generate an error during the Fitter processing.

2.5.2.1 Open the Device settings window from **Assignments** → **Device...** and click "Device and Pin Options".

2.5.2.2 Unselect all of the checkboxes in the Options box in the General category so that they match the following.



- 2.5.2.3 Under the Configuration category, select "Single Uncompressed Image with Memory Initialization (512Kbits UFM)" as the Configuration mode and ensure the other settings match as follows.



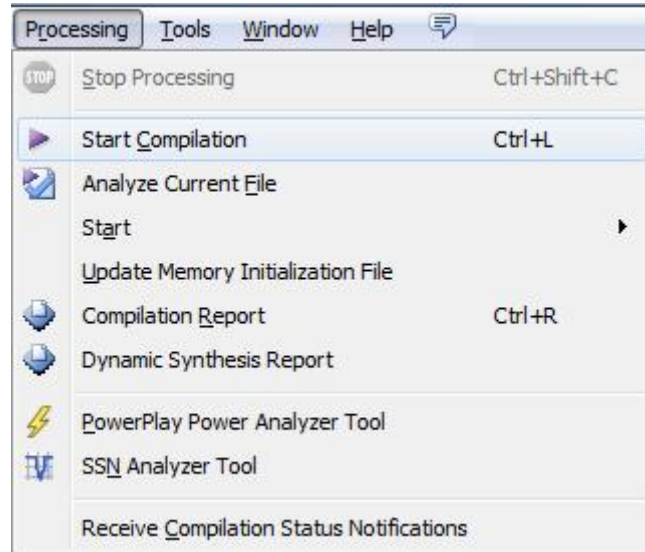
- 2.5.2.4 Click OK to close the Device and Pin Options window

- 2.5.2.5 Click OK to close the Device window

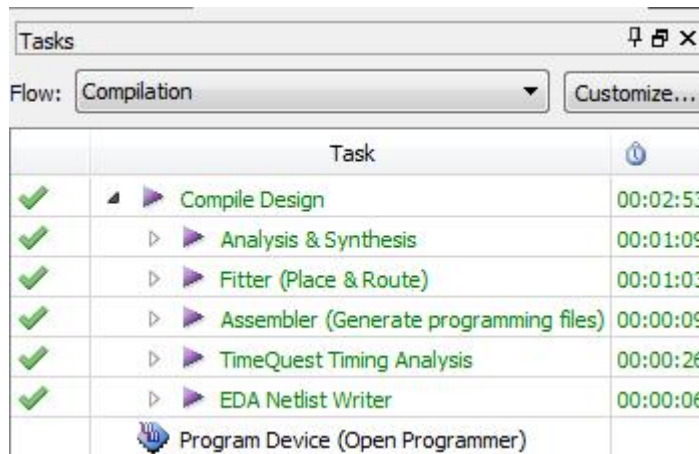
2.6 Compiling the Design

The next step is to compile the complete design. This step will verify there are no errors, create internal databases, as well as create a programming file that will be used in the next step.

2.6.1.1 Start the compilation: **Processing** → **Start Compilation** or double-click Compile Design in the Task window.



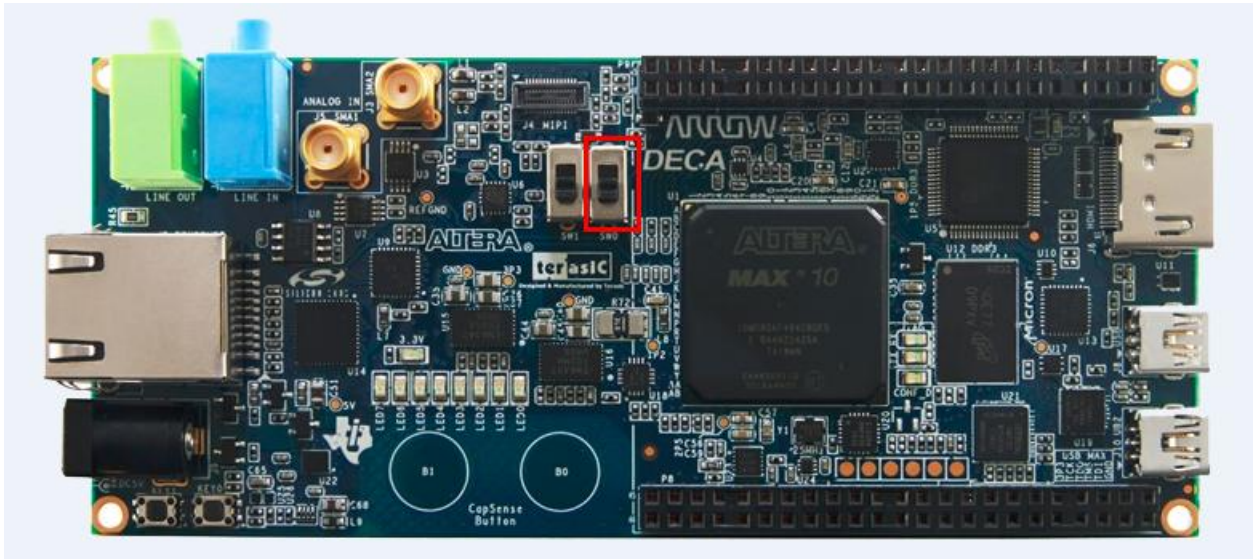
2.6.1.2 After a few minutes, the compilation should complete with no errors



2.6.2 Download the Configuration File to the DECA board

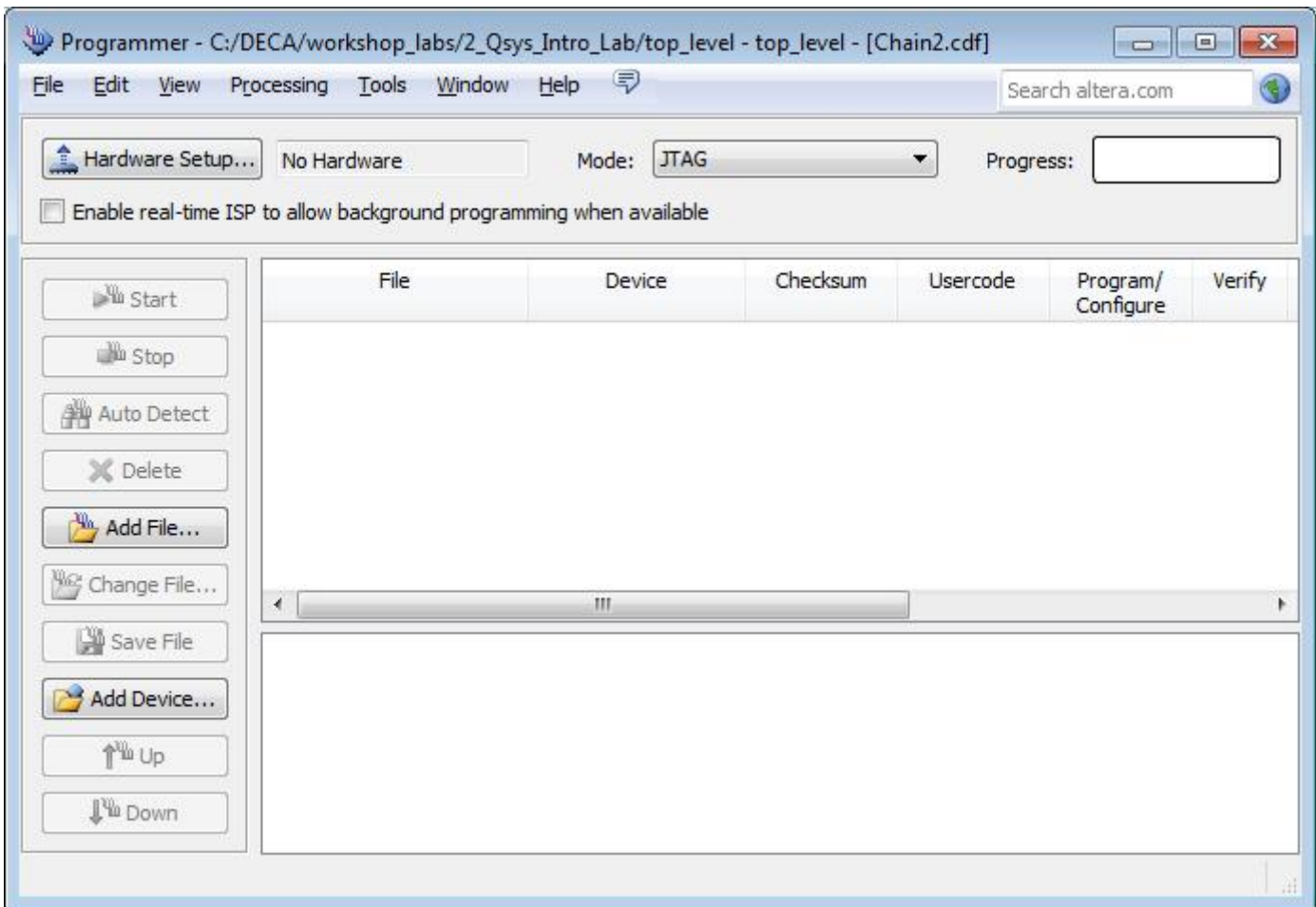
Now that the hardware design is complete and has been converted into a configuration file, the DECA needs to be programmed. Before doing so, we need to ensure the switch used to reset the logic is in the inactive state.

2.6.2.1 Set SW0 to low as shown here:

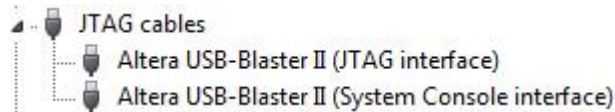


The logic within the FPGA can be reset by sliding the switch (SW0) up. To release reset, slide SW0 back down to the default position

2.6.2.2 Open the Quartus II Programmer from **Tools → Programmer** or double-click on Program Device (Open Programmer) from the Tasks pane. Since the DECA isn't connected yet, the Programmer should show a blank configuration.



2.6.2.3 Connect your DECA board to your PC using a USB cable. Be sure to connect it to the mini-USB connector labeled **UB2 J10** (on the bottom right of the board). Since the USB Blaster II driver software should already be installed, the Window's Device Manager should display two entries under "JTAG Cables".

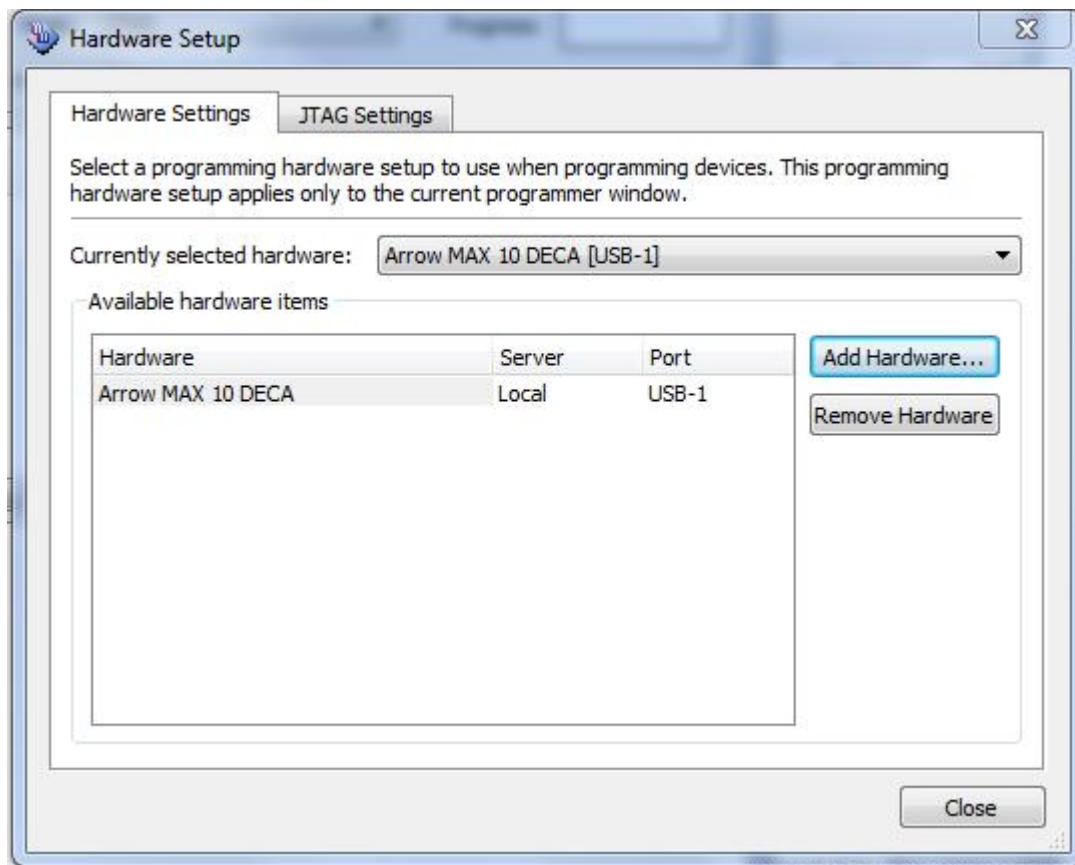


You should see a few LEDs light up on your DECA including the blue LED labeled 3.3V and the green LED labeled CONF_D.

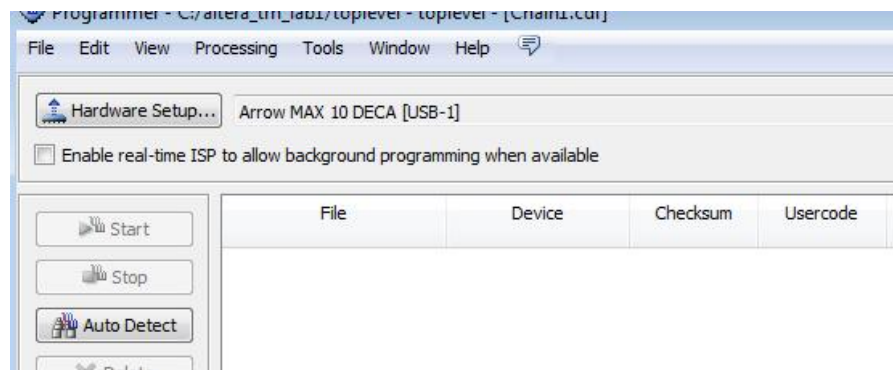


If the Device Manager shows an unconfigured USB Blaster, if Windows tries to look for drivers, or if the LEDs on the DECA do not light up, ask your workshop trainer for help.

- 2.6.2.4 In the Programmer window, click Hardware Setup and double-click the Arrow MAX10 DECA entry in the Hardware pane. The Currently Selected Hardware drop-down should now show Arrow MAX10 DECA [USB-1]. Depending on your PC, the USB port number may be different. Click Close.

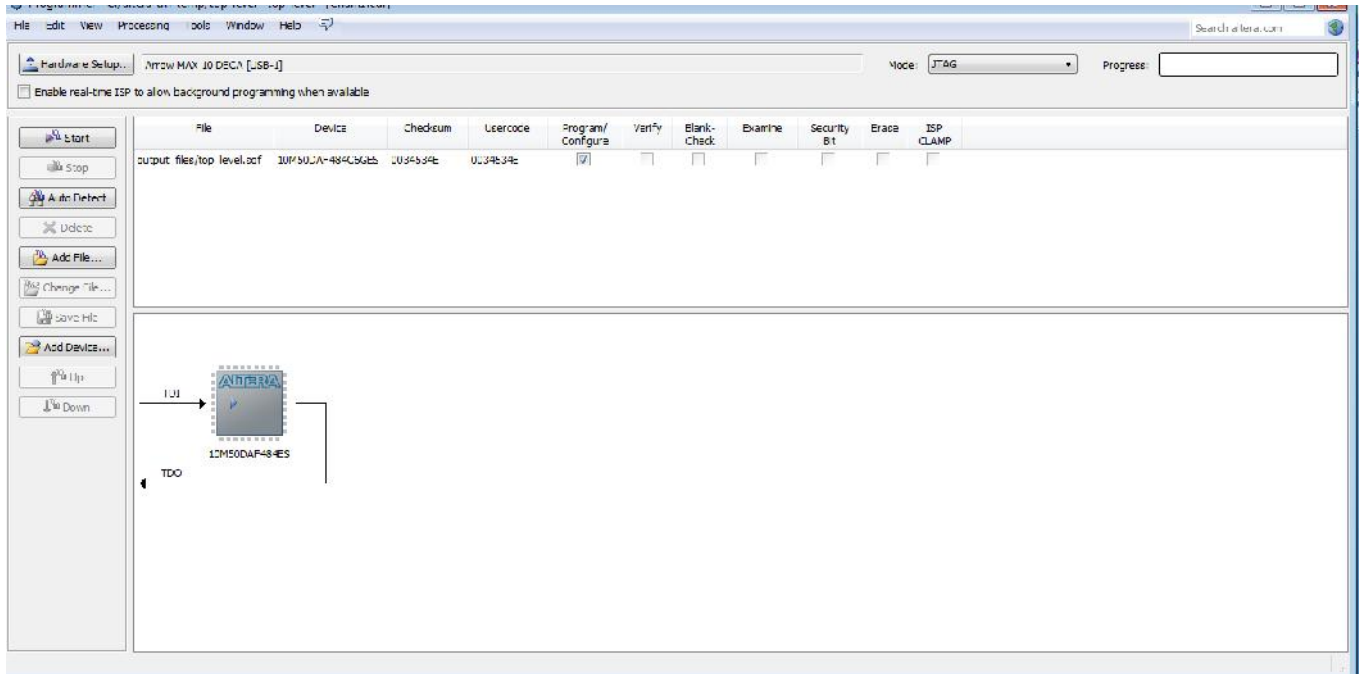


- 2.6.2.5 The programming window should now have a Hardware Setup such as:



- 2.6.2.6 Click "Add File..." and navigate to `<project_directory>/output_files/` in your compilation directory. Open the `top_level.sof` file.

The window should now look like this:



2.6.2.7 Make sure the Program/Configure checkbox is checked and click Start to program the DECA. You should see the **CONF_D** LED toggle briefly to indicate that the configuration is complete and the Progress bar should reach 100% (Successful).



2.7 Testing your design

2.7.1 Debugging a Qsys Design

After completing an FPGA design, there are a variety of ways to test the functionality. We will be using System Console to test our design. System Console is a low level hardware debug tool that is built with Tcl and it runs Tcl scripts and commands. We can communicate to System Console through our JTAG connection.

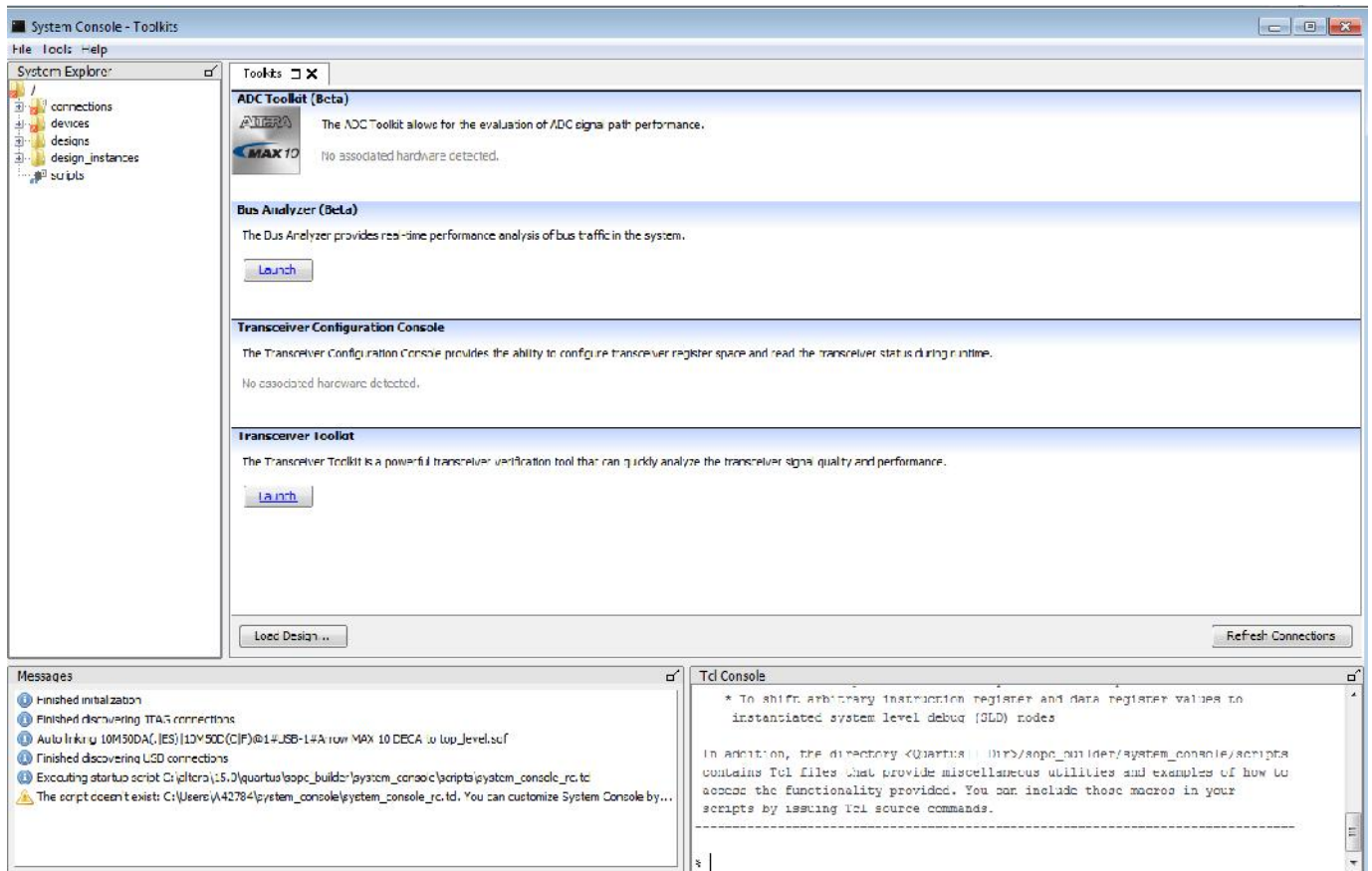
There are several ways to launch System Console. (either the **Tools → System Debug Tools → System Console...**, or via Qsys). For the lab, we will be using Qsys.

2.7.1.1 Launch Qsys (**Tools** → **Qsys**)

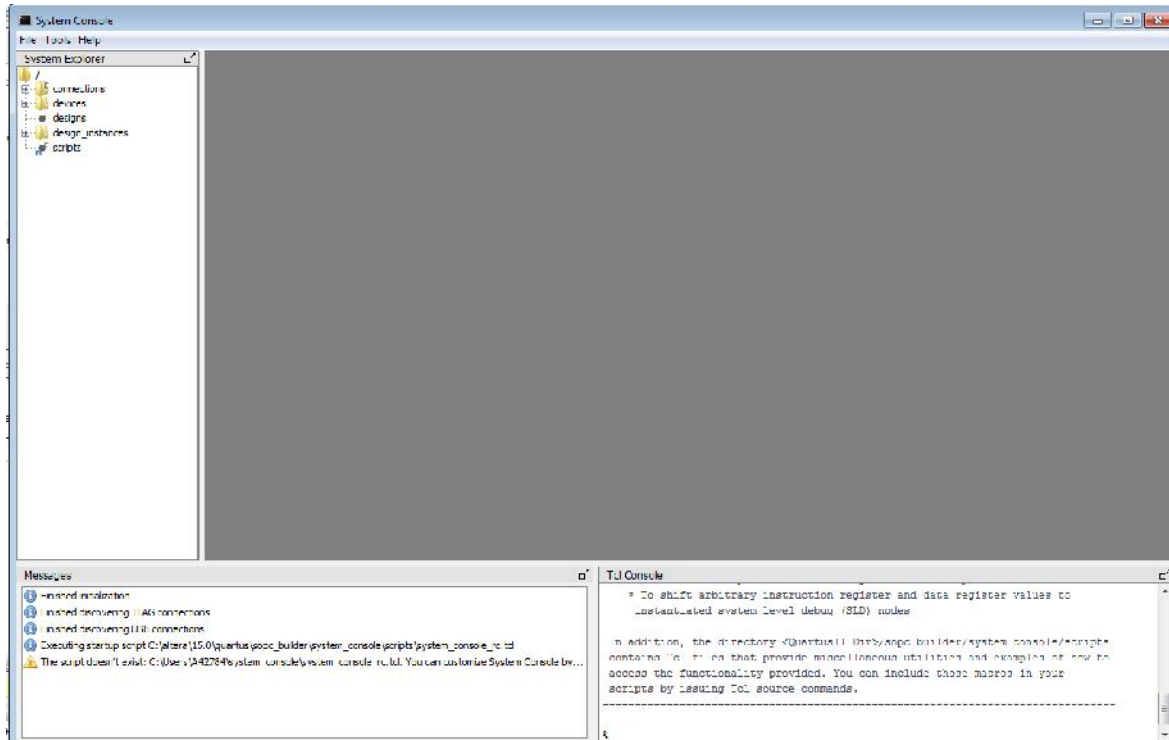
2.7.1.2 Open the `test_sys_top_qsys.qsys` file (you may need to browse to your top-level project folder)

2.7.1.3 Within Qsys, use the menu to select **Tools** → **System Console**.

2.7.1.4 A new window will appear as shown:



2.7.1.5 You can close the Toolkits tab as Toolkits will not be used in this design. Your window should now look like this:



2.7.1.6 To interact with the Qsys peripherals you can manually type TCL commands in the console window. For the purposes of this lab, several TCL scripts were created to automate this process.

2.7.1.7 There are two ways to launch TCL scripts. Either from the TCL console using `source myscript.tcl`, or via the menu **File** → **Execute Script**

2.7.1.8 Launch the first TCL script, `led_shifting.tcl`, by typing: `source led_shifting.tcl`

2.7.1.9 There will be an extremely fast LED shifting. (If you miss the shift, rerun the last TCL command by pressing the up arrow, then enter, or typing `source led_shifting.tcl` again)

2.7.1.10 Open the `led_shifting.tcl` in either Quartus II or your favorite text editor. Review the code

2.7.1.11 Modify the TCL script parameters section, specifically `delay` and `cycles`:

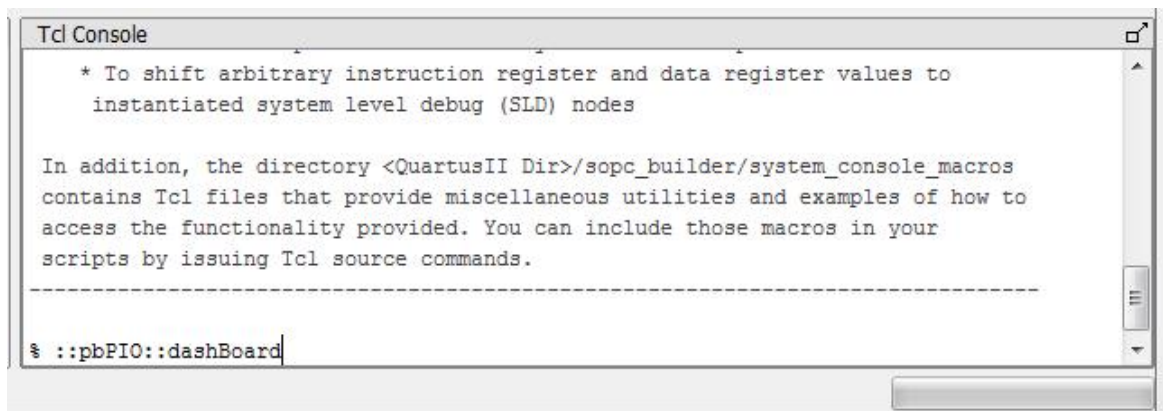
```
# Set the values to write to the LED pio.
set led_vals {1 2 4 8 16 32 64 128 128 64 32 16 8 4 2 1}
set delay 30000
set cycles 2
set max_loop_count [ expr $cycles * 16 ]
```

- 2.7.1.12 Experiment by changing the delay (say 120000), the number of cycles, or even the led_vals parameter. Save the TCL script and rerun the TCL in the System Console window as before.

2.8 System Console - Dashboards

2.8.1 Push Button Dashboard

- 2.8.1.1 With System Console, Dashboards can be displayed for visual diagnostics or controls.
- 2.8.1.2 We have to first setup the Push Button dashboard, by typing `source PB_dashboard.tcl`. There will be no dashboard available yet as we need to launch it.
- 2.8.1.3 Launch the Push Button dashboard by calling the function. Type: `::pbPIO::dashBoard`



```
Tcl Console

* To shift arbitrary instruction register and data register values to
  instantiated system level debug (SLD) nodes

In addition, the directory <QuartusII Dir>/sopc_builder/system_console_macros
contains Tcl files that provide miscellaneous utilities and examples of how to
access the functionality provided. You can include those macros in your
scripts by issuing Tcl source commands.

-----

% ::pbPIO::dashBoard
```

The current state of the LEDs is dark green and there is a zero for the PB Input History.

- 2.8.1.4 The next step is to see what happens if the pushbutton was selected. This is a two step process. The first step is to type `::pbPIO::dashBoard`, but do not press Enter until you hold down the push button (KEY0). The KEY0 button is the right button located under the power connector.



2.8.1.5 While holding the push-button, press enter to re-launch the Push Button Dashboard

You should now see the dashboard led illuminated. You can release the push-button as the value has been stored for display.

2.8.2 Temperature Dashboard

2.8.2.1 To view the temperature, we will launch the new dashboard. To do so, we have to setup the dashboard first

2.8.2.2 Type source `Therm_SPI_dashboard.tcl`

2.8.2.3 Launch the temperature sensor dashboard by typing: `::thermSPI::dashBoard`, and pressing enter
A dashboard will appear showing the temperature of the hot area of the board (where the power supplies reside)

CONGRATULATIONS!
YOU HAVE COMPLETED THE QSYS INTRODUCTION LAB!