



Five Years Out

Max10 DECA Workshop Manual

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

WORKSHOP MANUAL TABLE OF CONTENTS

LAB 1.	FPGA INTRODUCTION LAB	6
LAB 2.	QSYS INTRODUCTION LAB	50
LAB 3.	EMBEDDED SYSTEMS LAB	81
LAB 4.	GESTURE SENSOR LAB	144
LAB 5.	MAX10 ADC DATA CAPTURE	202
LAB 6.	USB 2.0 TO SDHC LAB	242
LAB 7.	INTERACT WITH DECA USING BLE AND WI-FI.....	270
LAB 8.	MIPI TO HDMI LAB	294
LAB 9.	APPENDIX.....	308

FPGA Introduction Lab

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 1. FPGA INTRODUCTION LAB	6
1.1 Getting Started	6
1.1.1 Getting your DECA Kit	6
1.1.2 Review the DECA Development Platform	6
1.2 Create a Quartus II Project	7
1.2.1 Create a Project in Quartus II	7
1.3 Build the Design	12
1.3.1 Block Diagram.....	12
1.3.2 Components of the Design	13
1.3.3 IP Catalog	13
1.3.4 Create the PLL	13
1.3.5 Configure the PLL	14
1.3.6 Create the Counter	19
1.3.7 Creating the Multiplexer	21
1.3.8 Adding Components to the Schematic file	25
1.3.9 Connect the Components	28
1.3.10 Analysis and Synthesis	37
1.3.11 Adding Timing Constraints.....	38
1.3.12 Adding Pin Assignments	39
1.3.13 Compiling the Design	43
1.3.14 Configuring the Device.....	43
1.3.15 Testing the Design	46

LAB 1. FPGA INTRODUCTION LAB

Overview: This Lab will guide the student through the complete design cycle from Design Entry to Configuring the MAX 10 on the DECA board.

If you are a new user, this Lab will walk you through the steps in the Quartus II tool suite to build a design, compile it, and download it to a board.

1.1 Getting Started

1.1.1 Getting your DECA Kit

If you are attending a DECA Workshop, you will receive your DECA kit in the 3-in-1 evaluation kit bundle when you arrive at the workshop location. If you are working on this lab independently, a DECA kit can be purchased through your Arrow sales representative or at parts.arrow.com.



Make sure you have a USB cable to connect the on-board USB Blaster to your laptop. If you are attending the workshop, USB cables are included in your evaluation kit bundle.

1.1.2 Review the DECA Development Platform

Review the components on the DECA board. This development board provides a full system built around the MAX10, including external memory, LEDs, sensors, buttons, and power supplies.



There are many components on the DECA board including the LEDs, capacitive push buttons, USB, Ethernet, HDMI and MIPI Interfaces.

This simple FPGA design will use a PLL, a push-button and a counter to output a value to the LEDs.

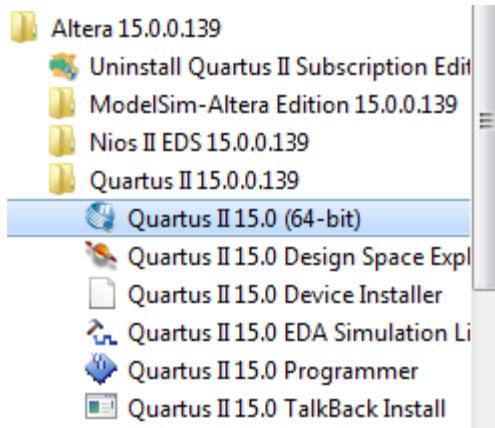
1.2 Create a Quartus II Project

In this module, you will create an FPGA design using the Quartus II Tool Suite. You will then compile this design and download it to the DECA board.

1.2.1 Create a Project in Quartus II

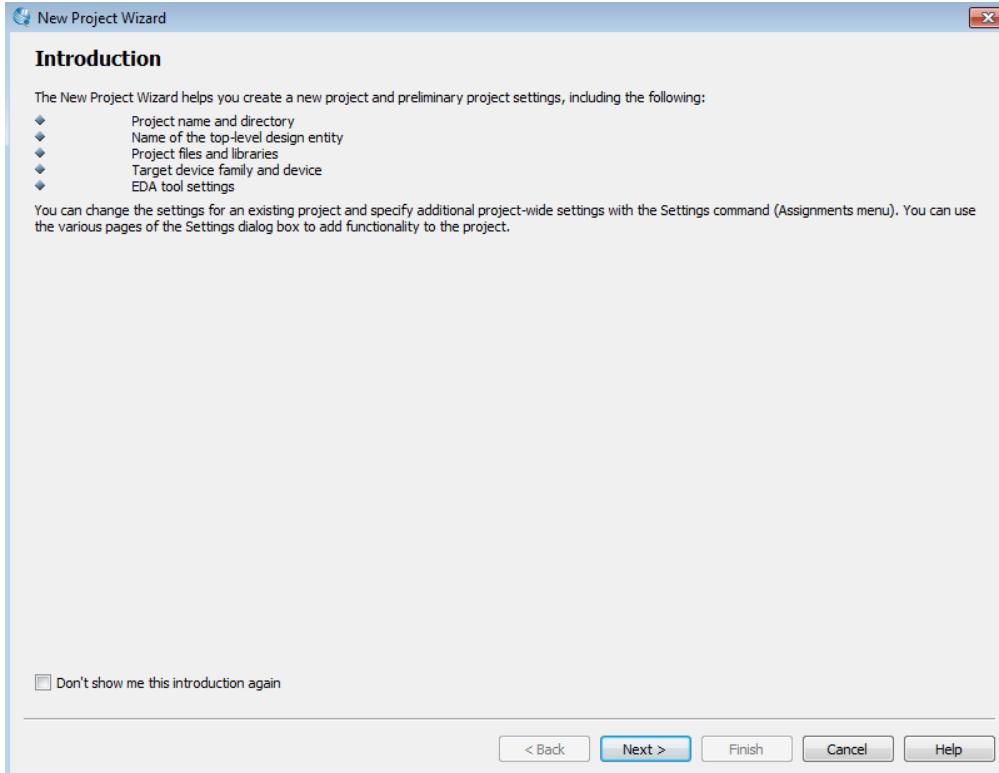
The project folder for this lab will reside to the location where you extracted the lab files. Generally, this location will be `C:\DECA\workshop_labs\1_FPGA_Intro_Lab`

1.2.1.1 Launch Quartus II 15.0 (64-bit) from the Start menu.



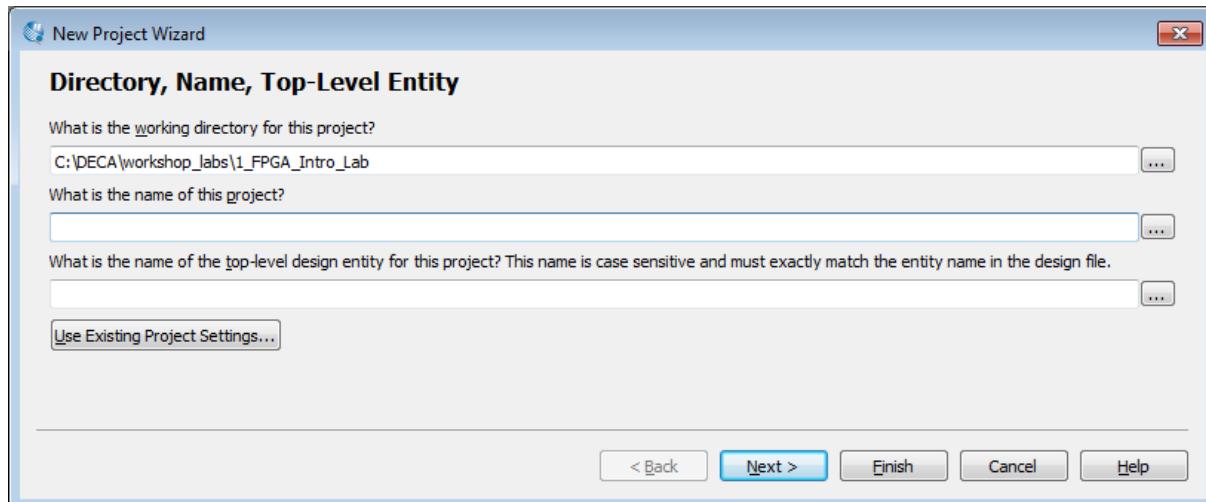
1.2.1.2 In the Quartus tool Create a new project: **File → New Project**

The New Project Wizard walks you through the project settings, such as the name, directories, files, directories, device family, and other settings. These settings can be changed later if needed.

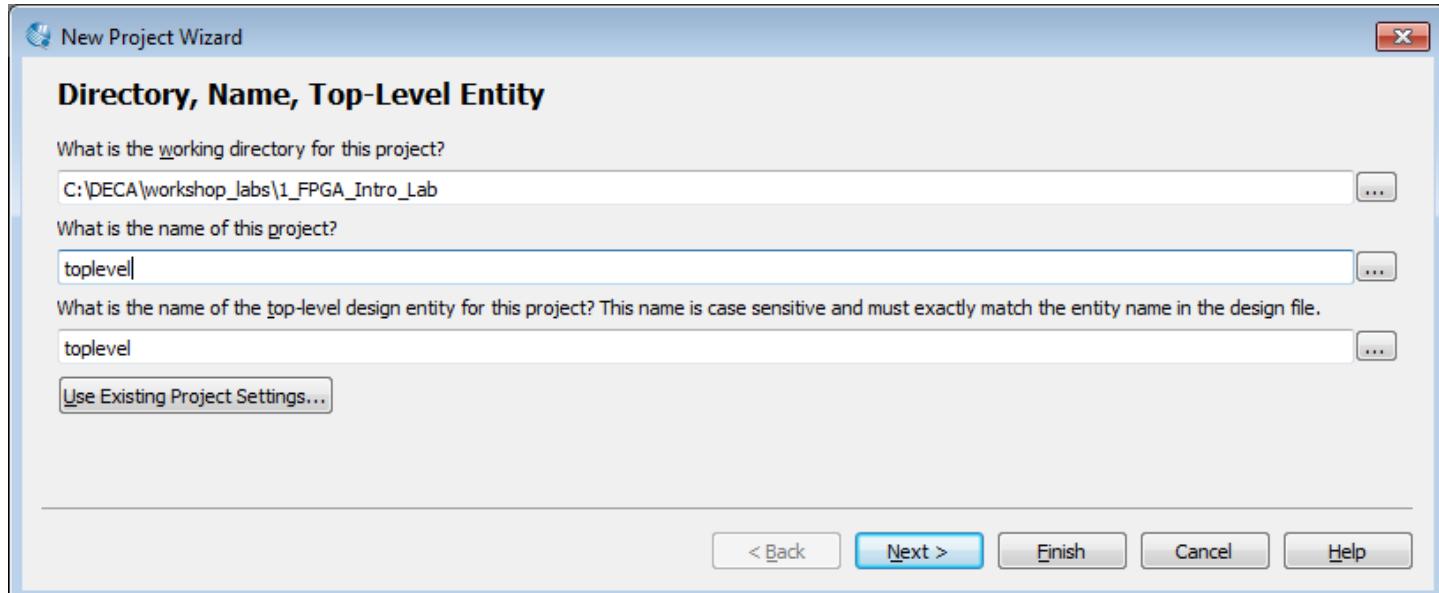


Click Next

1.2.1.3 Browse to the Project directory: C:\DECA\workshop_labs\1_FPGA_Intro_Lab



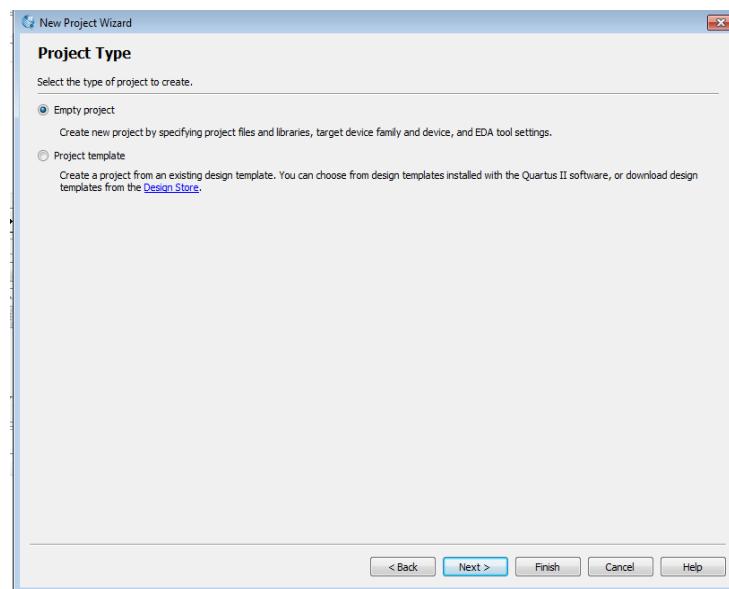
1.2.1.4 Enter the project Name "toplevel"



Click Next

1.2.1.5 Project Type

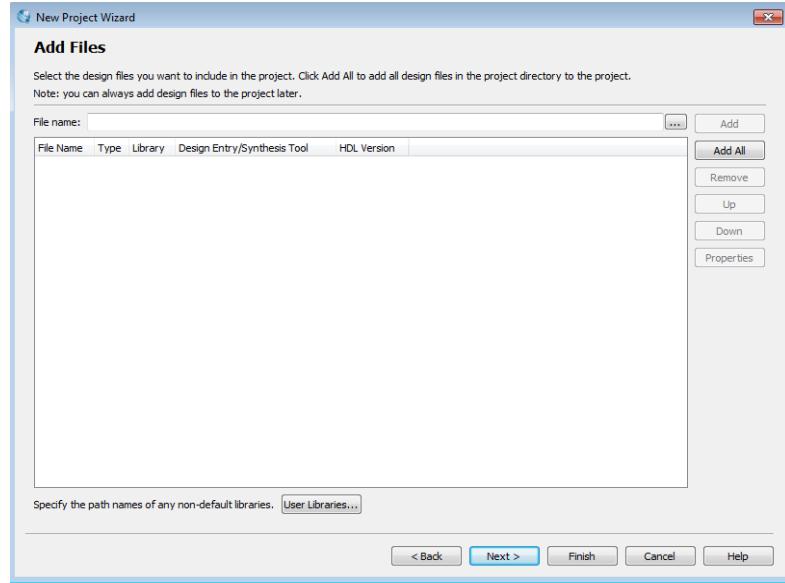
This page will be to create the Project type. In this lab, a new project will be created, and thus the default settings of empty project should be selected.



Click Next

1.2.1.6 Add Project Files

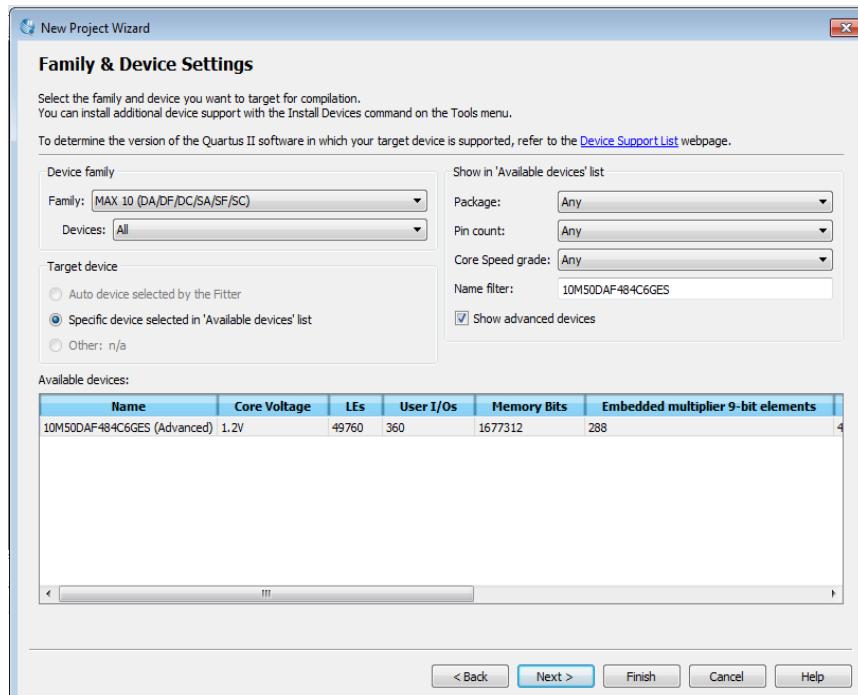
The Add File window will appear. For this lab, new design files will be created so no files will be added. For other designs, files can be added here.



Click Next

1.2.1.7 Select DECA Board MAX 10 Device Part Number

In the Family and Device Settings, use the pull down menu to select the family as Max 10. Then in the Name Filter, enter: **10M50DAF484C6GES**

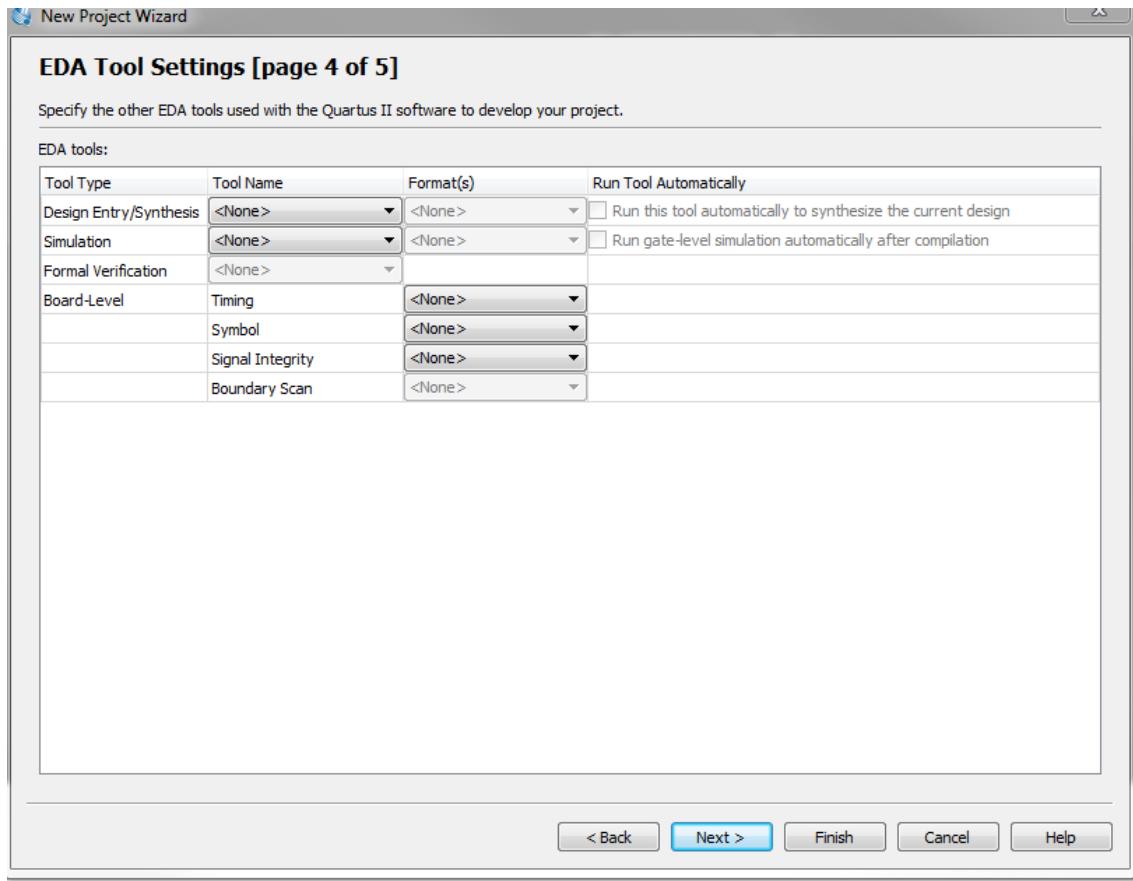


Rather than entering the exact part number, the pull-down menus can be used to select the correct family, package, pin count, and speed grade. Quartus II will use these settings to compile the design, and also provide the programming file that you will use later to program the device.

Click Next

1.2.1.8 EDA Tool Settings

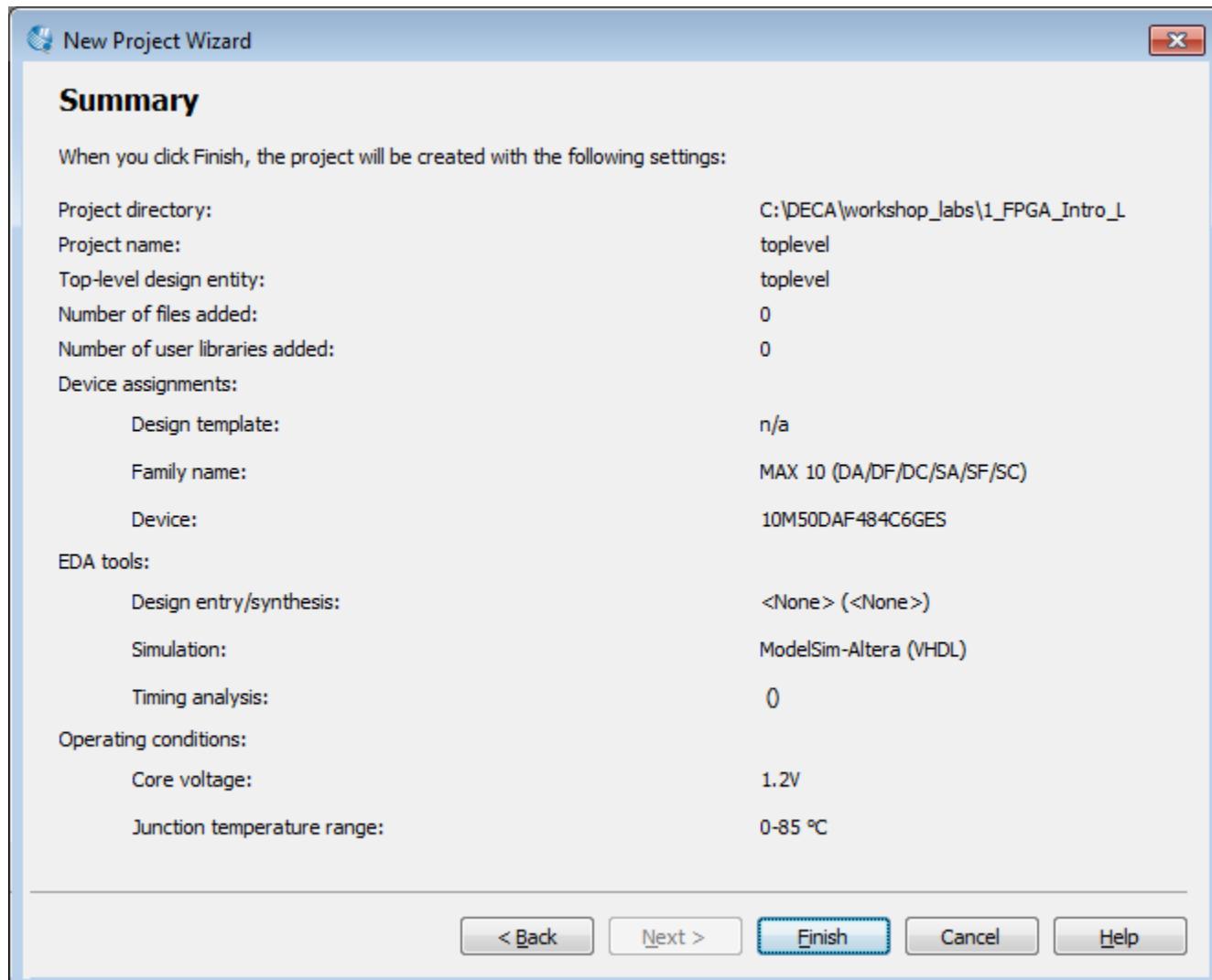
In the EDA tool Settings window, disable any EDA tools, if there are any present. EDA tools are third party tools that work with Quartus II for design entry, simulation, verification and board-level timing. For this workshop, no EDA software will be used, as only Quartus II will be used.



Click Next

1.2.1.9 Project Summary Page

This is the Summary Page that showing the settings Quartus II will use for this Project. These settings can be changed, as needed at a later time.



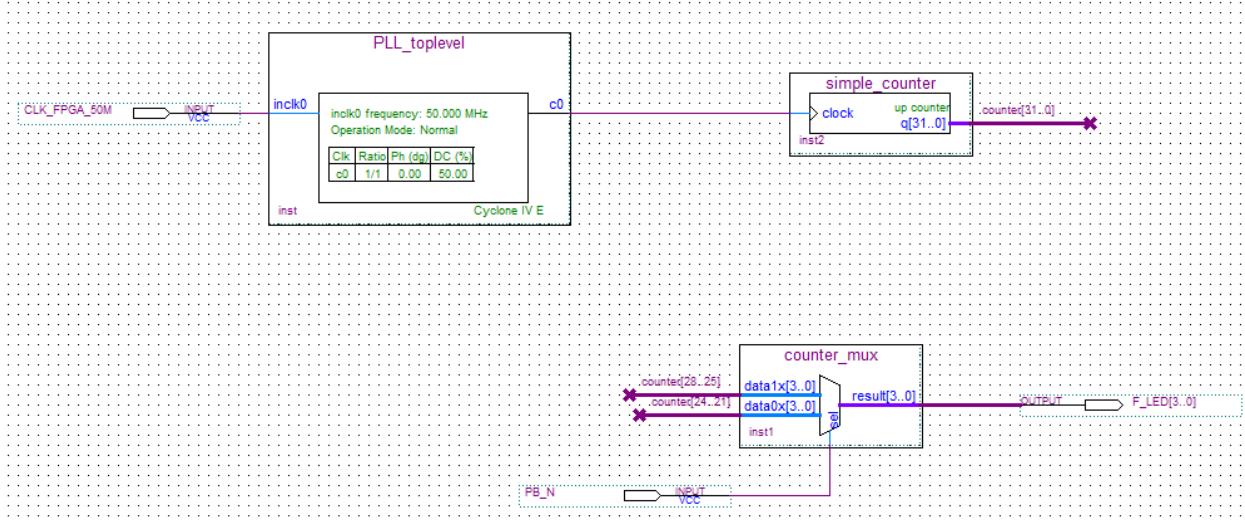
Click Finish.

1.3 Build the Design

Overview: In this section, you will create the components to a design, make connections, set the pins and compile the design.

1.3.1 Block Diagram

The final system that will be built as described with the following steps, will look as follows when complete:



1.3.2 Components of the Design

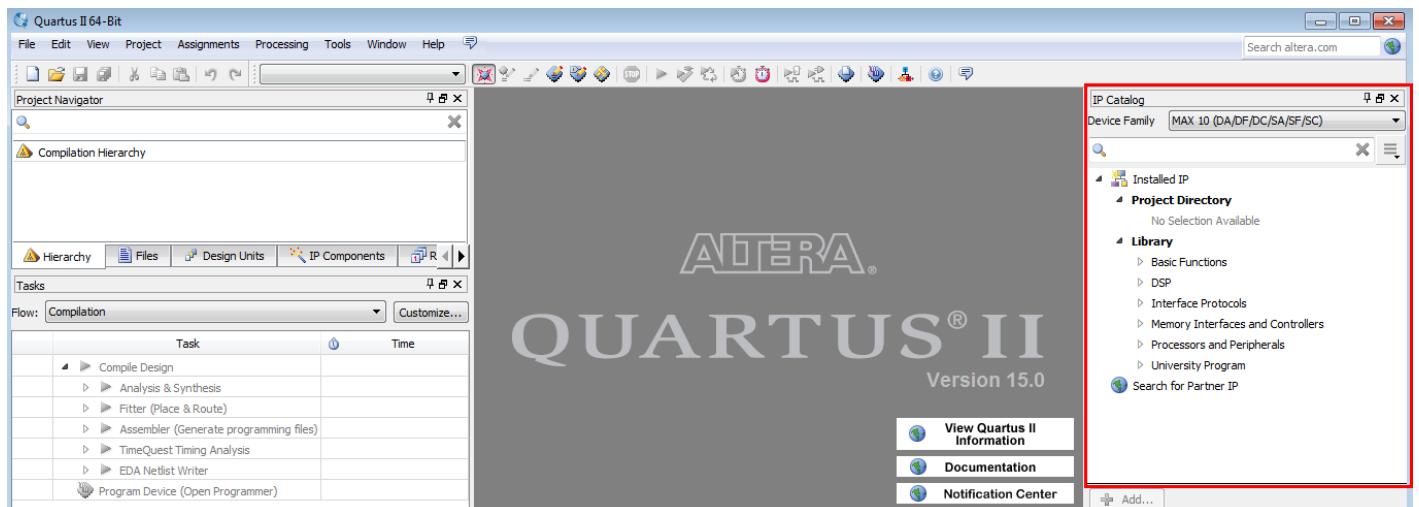
There are three components in the system: a PLL, a counter, and a mux. The components, in the following steps, will be built separately and then connected together. A push button on the board controls the mux. The mux in turn control which of the counter outputs (slow counting or fast counting) will be shown on the LEDs.

There are different ways to create components, such as RTL or schematic. In this lab, schematics will be used.

There are also different ways for entering schematics such as Qsys and IP Catalog. This lab will focus on the IP Catalog.

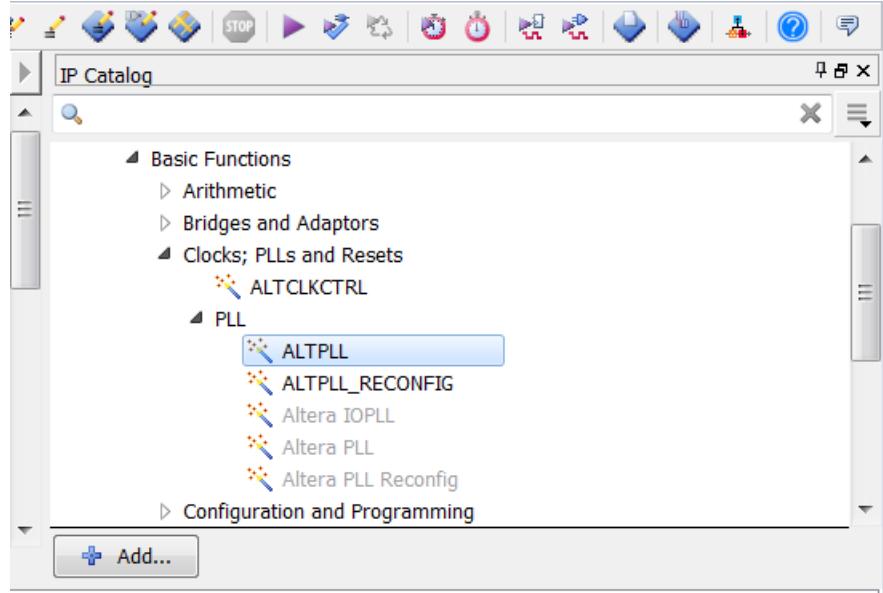
1.3.3 IP Catalog

The IP Catalog allows you to create and modify design files with custom variations. The IP catalog window is open by default when you open Quartus. If it is not present, you can open it by selecting **Tools → IP Catalog**



1.3.4 Create the PLL

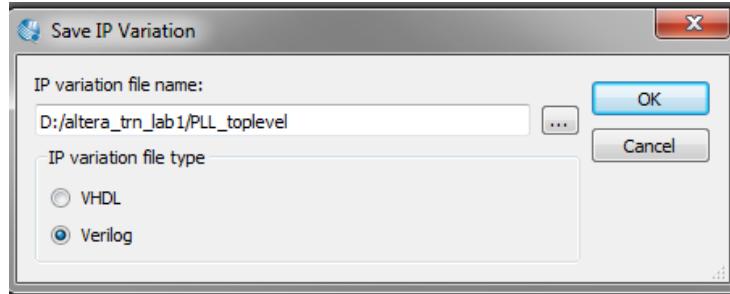
In the IP Catalog, browse for ALTPLL, via: **Basic Functions → Clocks; PLLs and Resets → PLL**



1.3.4.1 Click ALTPLL, where ALTPLL stands for Altera Phase Locked Loop

1.3.4.2 Click Add

When the Save IP Variation window appears, enter the file name variation as PLL_toplevel and select Verilog (VHDL could be used as well). Both Verilog and schematic code will be created.



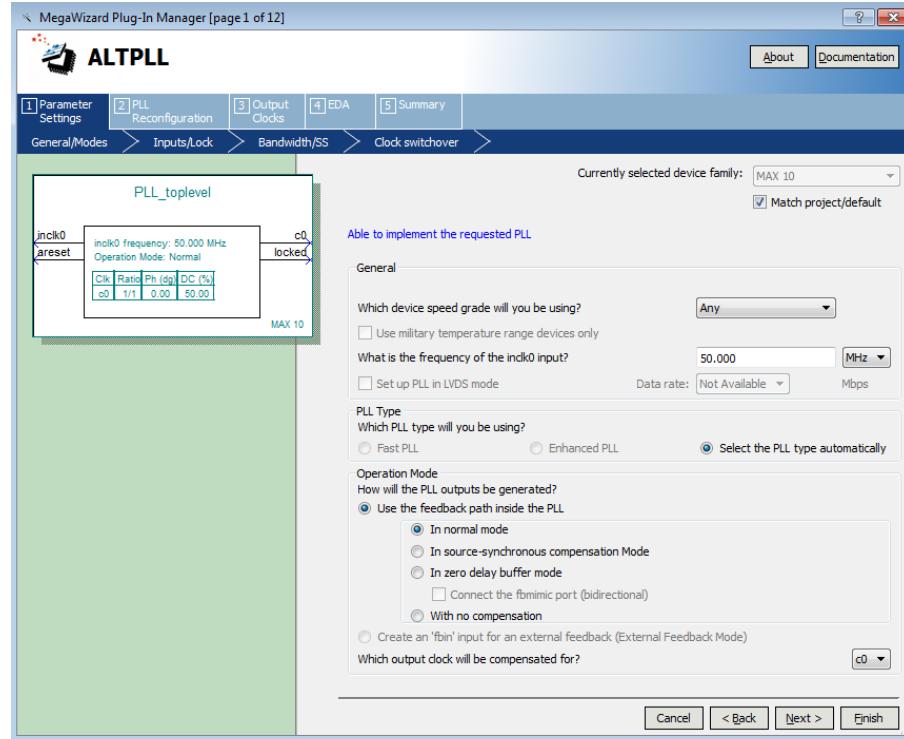
1.3.4.3 Click OK

1.3.5 Configure the PLL

The next step is to configure the PLL Component.

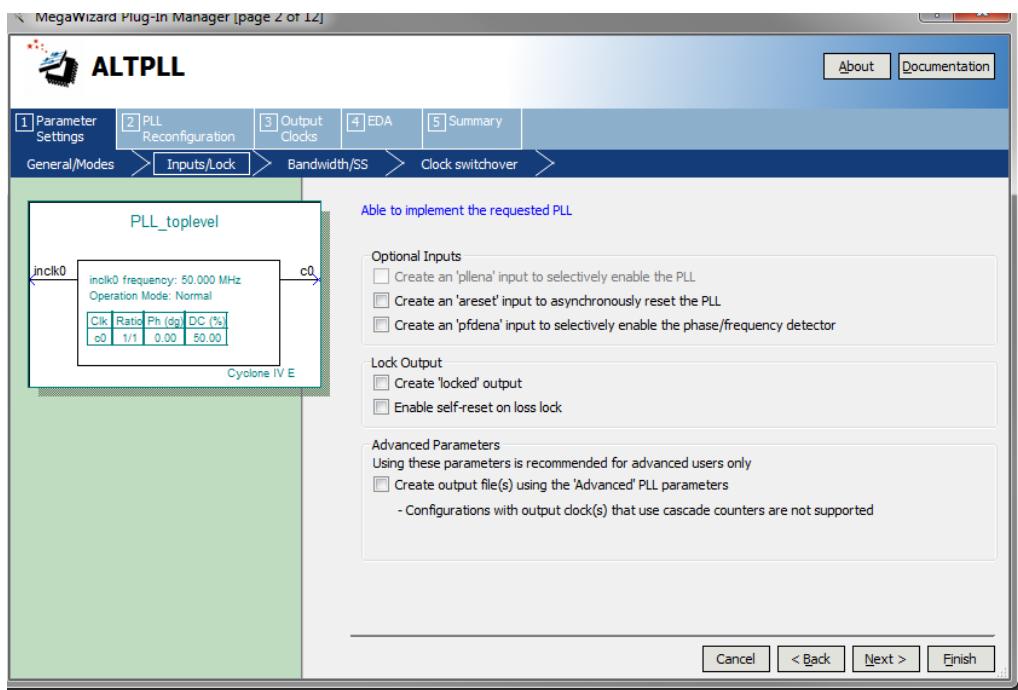
1.3.5.1 Enter the PLL reference clock frequency to match the clock input on the DECA board. Since we have 50 MHz coming into the FPGA, the inclk0 input to be 50 MHz.

1.3.5.2 The setting should look like:



1.3.5.3 Click Next

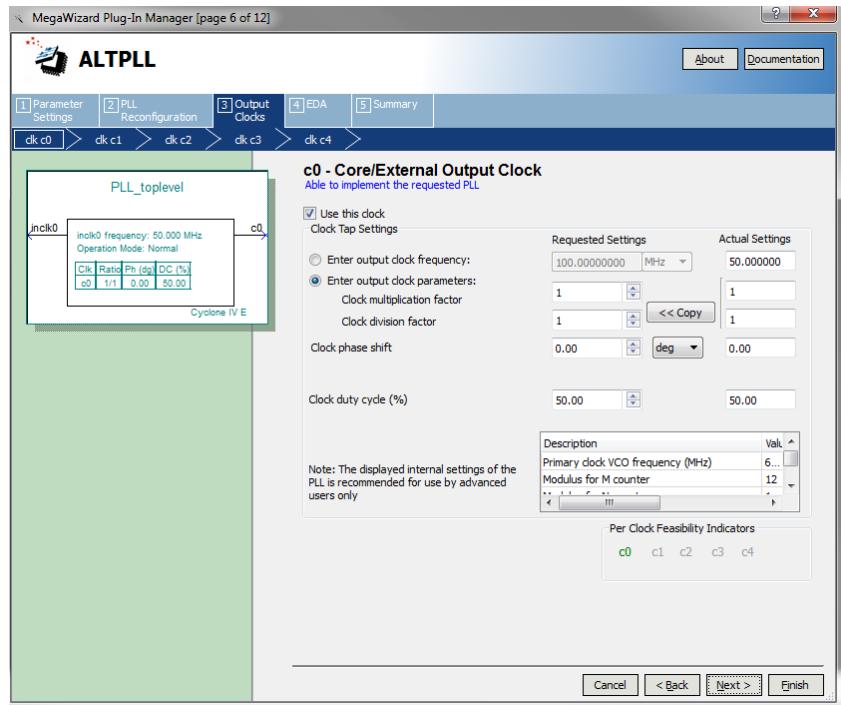
1.3.5.4 Simplify the PLL, by disabling the areset and locked outputs



1.3.5.5 Click Next

1.3.5.6 Continue to select Next to go through the various options (e.g. Pages 3 to Pages 5), but leaving the default options as they are. The Page numbers can be seen on the top of the windows.

1.3.5.7 On page 6, (c0 - Core/External Output Clock) ensure that the output clock is set to one for the clock multiplication and one for the clock division. For simplification, there is one input to the PLL (50 MHz), and one output of the PLL (50 MHz).



1.3.5.8 Click Next.

1.3.5.9 Click Next for Pages 7 to 11, as the default settings are to be used.

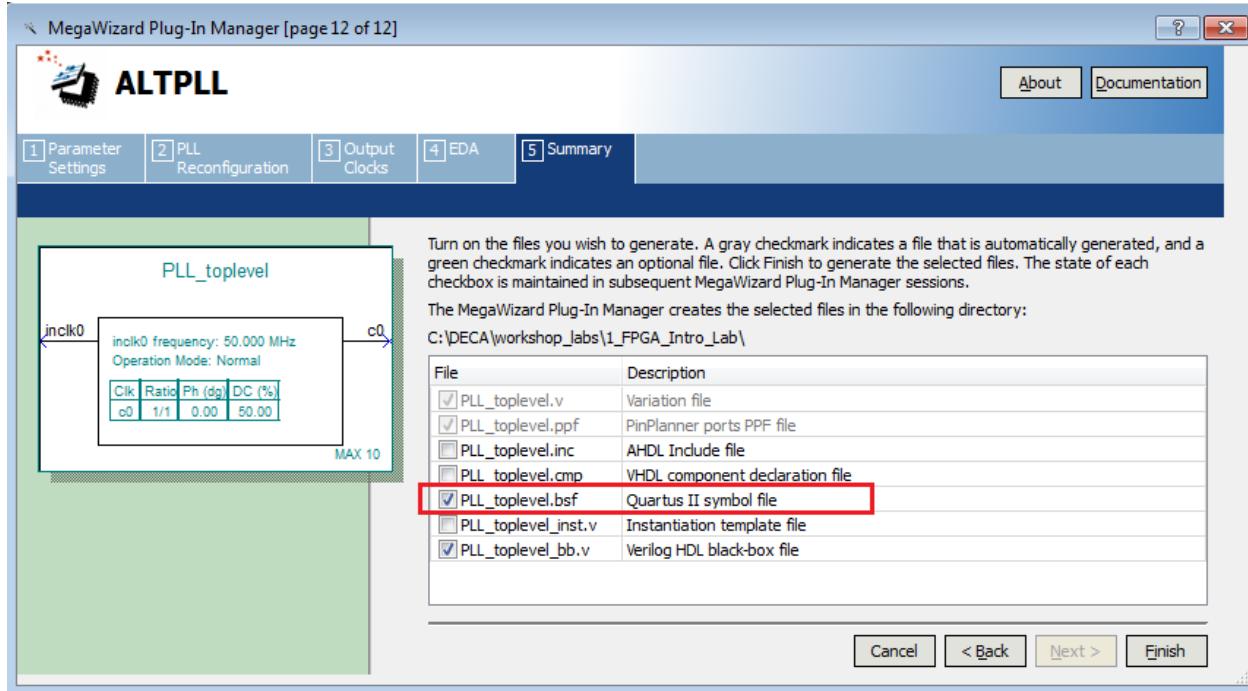
1.3.5.10 Click Next.

On Page 12, there is a list of output files that will be generated. The default settings are seen below:

The MegaWizard Plug-In Manager creates the selected files in the following directory: C:\DECA\workshop_labs\1_FPGA_Intro_Lab\	
File	Description
<input checked="" type="checkbox"/> PLL_toplevel.v	Variation file
<input checked="" type="checkbox"/> PLL_toplevel.ppf	PinPlanner ports PPF file
<input type="checkbox"/> PLL_toplevel.inc	AHDL Include file
<input type="checkbox"/> PLL_toplevel.cmp	VHDL component declaration file
<input type="checkbox"/> PLL_toplevel.bsf	Quartus II symbol file
<input type="checkbox"/> PLL_toplevel_inst.v	Instantiation template file
<input checked="" type="checkbox"/> PLL_toplevel_bb.v	Verilog HDL black-box file

Since this design will be done in a schematic, you will need to select the **PLL_toplevel.bsf** checkbox. The .bsf file provides a symbol that can be used in the schematic design we will be creating later.

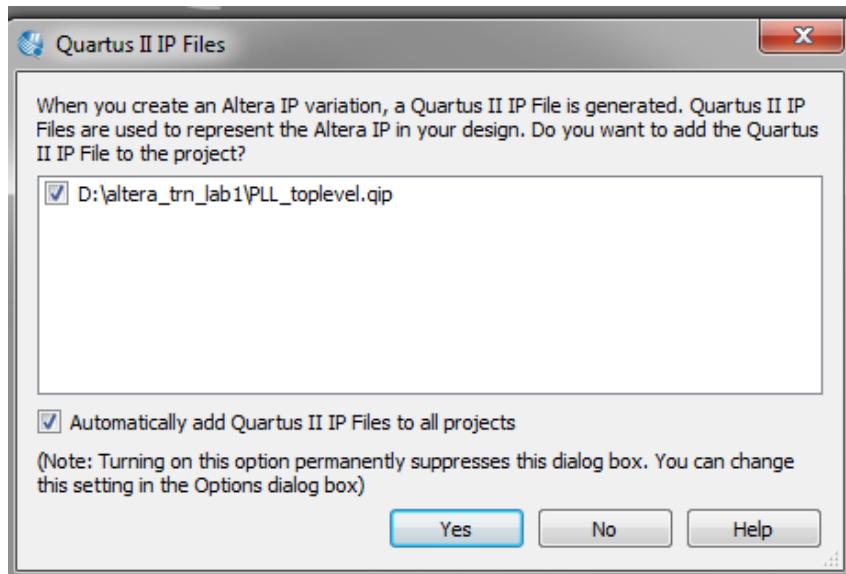
1.3.5.11 The view should now look like this:



1.3.5.12 Click Finish.

The PLL (1st component) will now be created.

Note if this is the first time that you are using this version of Quartus II, you might see a pop-up Window for Quartus II IP Files that states:



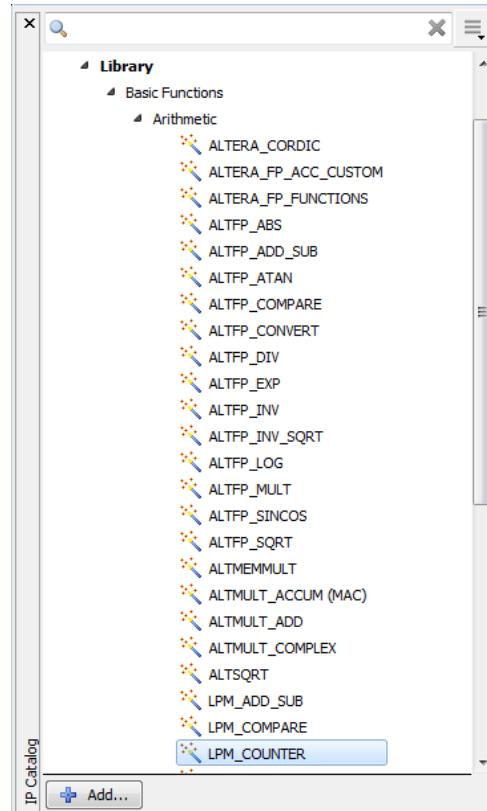
1.3.5.13 Select Automatically Add Quartus II IP Files to all projects.

1.3.5.14 Click Yes to allow all of the IP to automatically be added to the project, and so that this message will not be seen for other designs.

1.3.6 Create the Counter

The next step is to create a counter which will drive the LEDS on the DECA Board

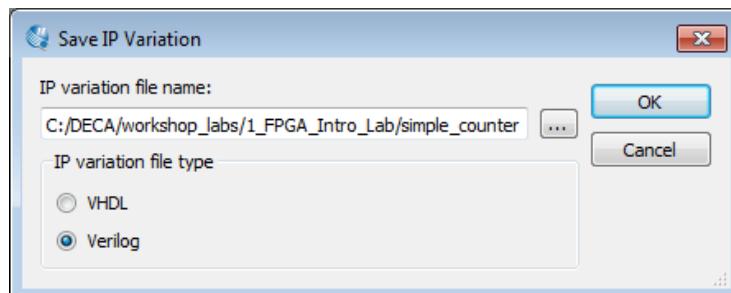
1.3.6.1 To create this counter, select the IP Catalog and Expand the Basic Functions → Arithmetic and select LPM_COUNTER.



(LPM stands for Library of Parameterized Modules.).

1.3.6.2 Click Add

1.3.6.3 When the Save IP Variation pop up appears, enter **simple_counter** and select Verilog as below

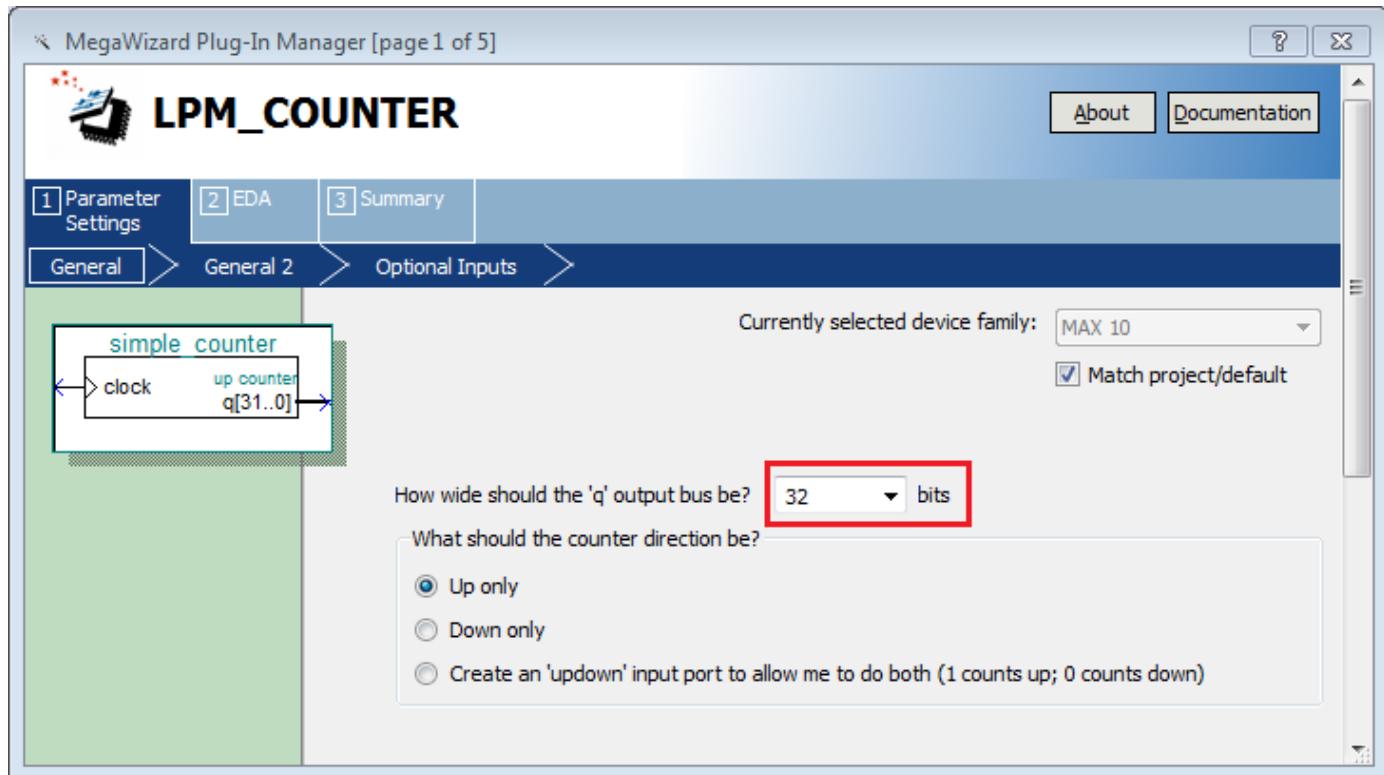


1.3.6.4 Click OK.

1.3.6.5 The next step is to increase the size of the counter to a number of bits large enough to divide down the clock so we can see the LEDs toggling.

1.3.6.6 Change this number to 32

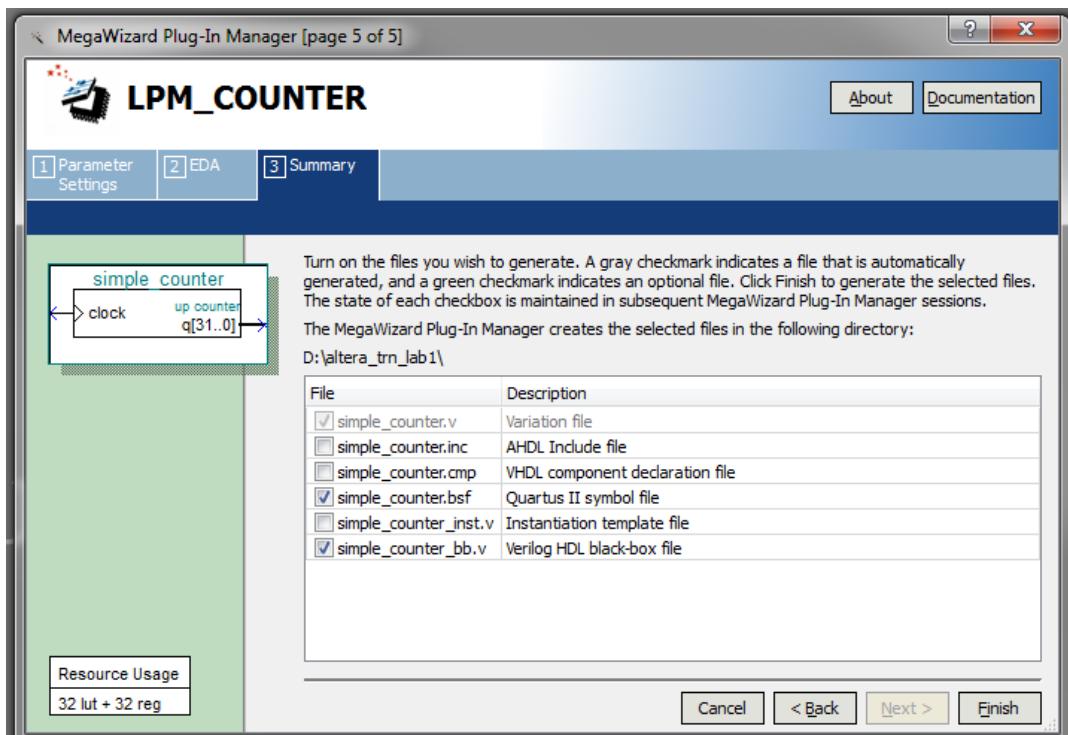
1.3.6.7 Allow the counter to be Up only, so the LEDS will show the counters counting up. The window should now look like:



1.3.6.8 Select Next repeatedly until reaching Page 5.

Select the `simple_counter.bsf` checkbox to generate a symbol for our schematic design.

1.3.6.9 The screen should now look like:



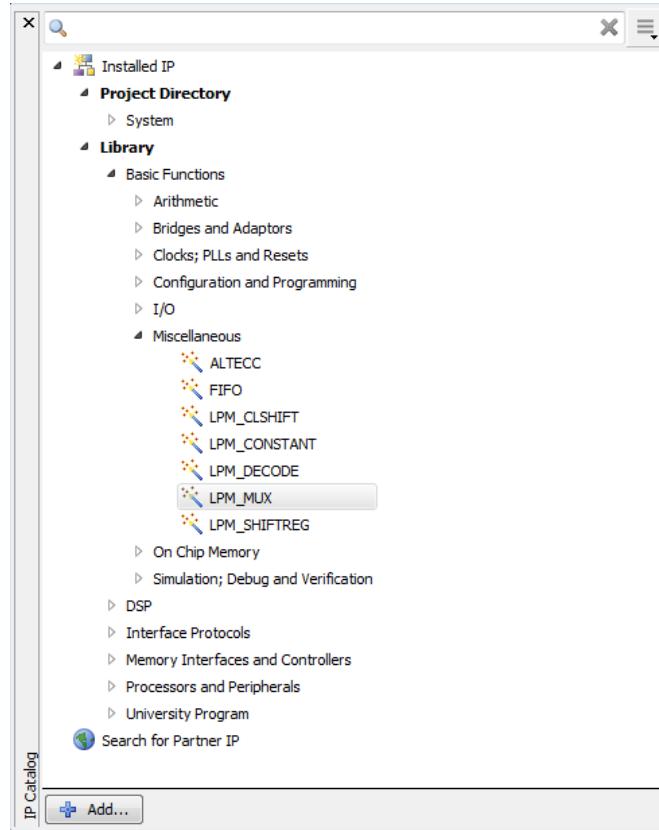
1.3.6.10 Click Finish

The Counter is now created.

1.3.7 Creating the Multiplexer

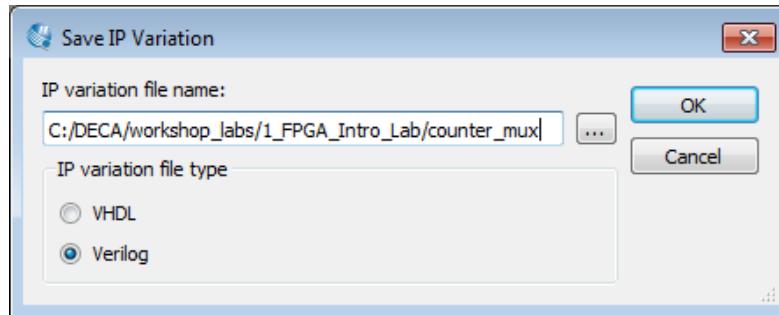
The next step is to create a mux component. This mux would be used along with a push button on the board to control the speed of the counter, where the counter outputs will be seen on the LEDs.

1.3.7.1 To create this mux, select IP Catalog and expand Basic Function → Miscellaneous and select LPM_MUX



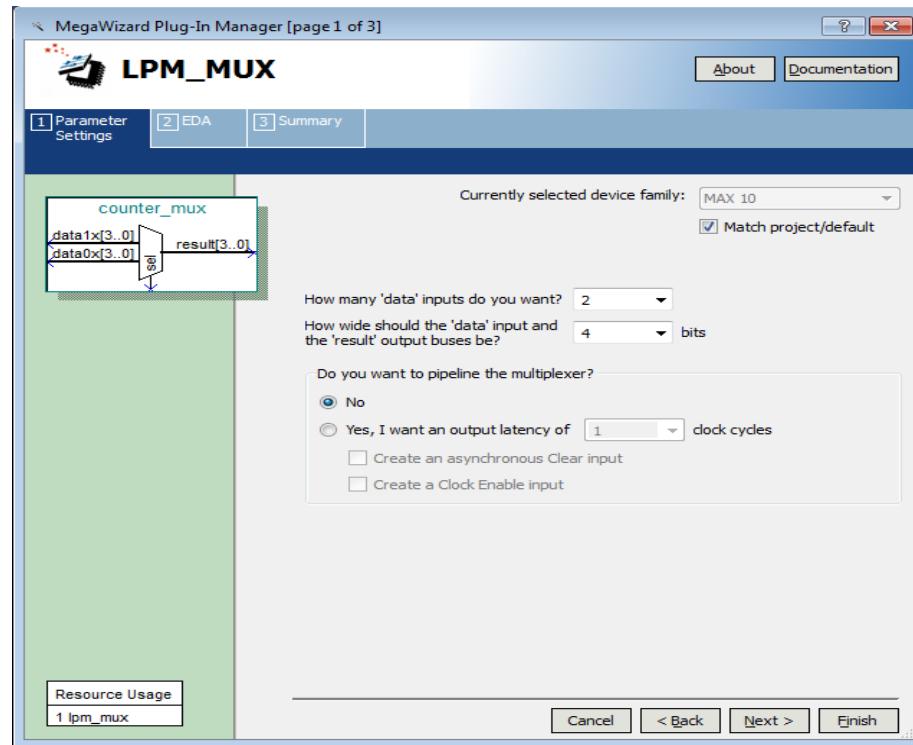
1.3.7.2 Click Add

1.3.7.3 In the Save IP Variation, enter the name of counter_mux and the File type is to be Verilog.



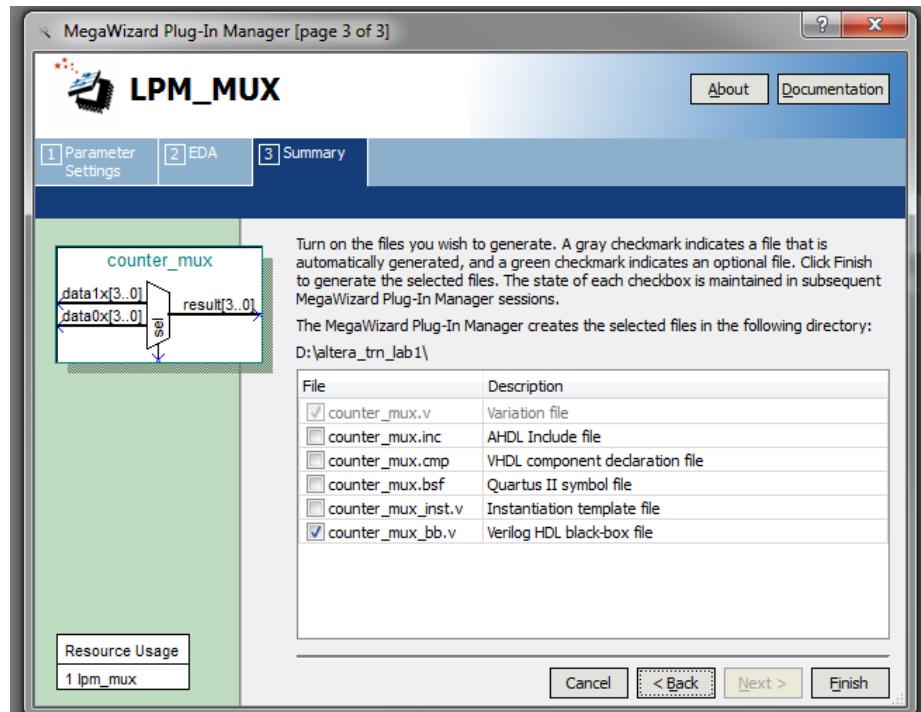
1.3.7.4 Click OK

1.3.7.5 Select **2** data inputs and the width of the input and output buses to be **4** bits. The reason for 4 bits is that 4 LEDs will be changing (showing the count values)



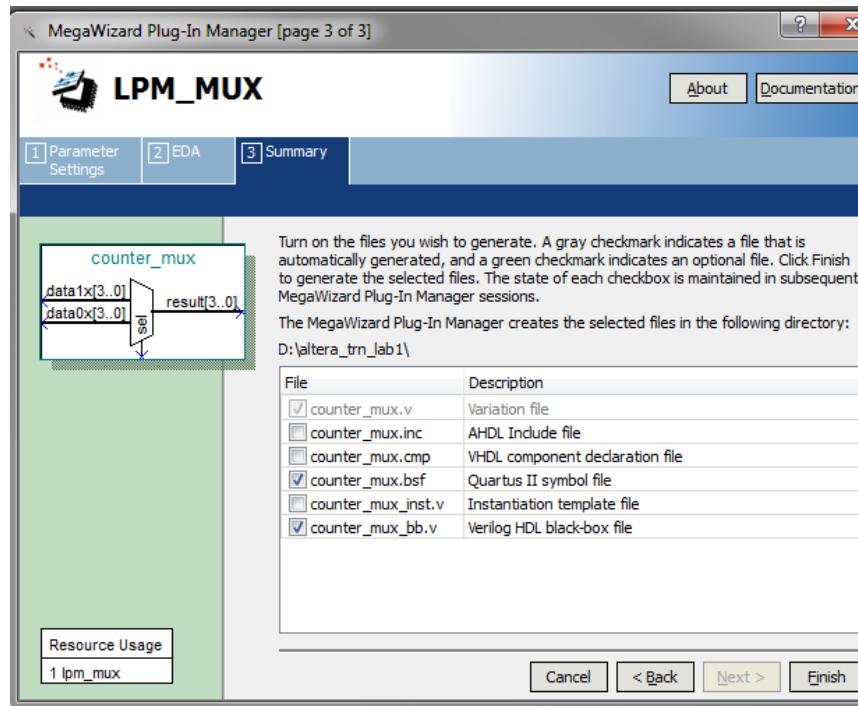
1.3.7.6 Click Next

1.3.7.7 Continue to select Next until Page 3



1.3.7.8 On this page, select **counter_mux.bsf** so that a symbol will be generated.

1.3.7.9 The view of this now looks like:

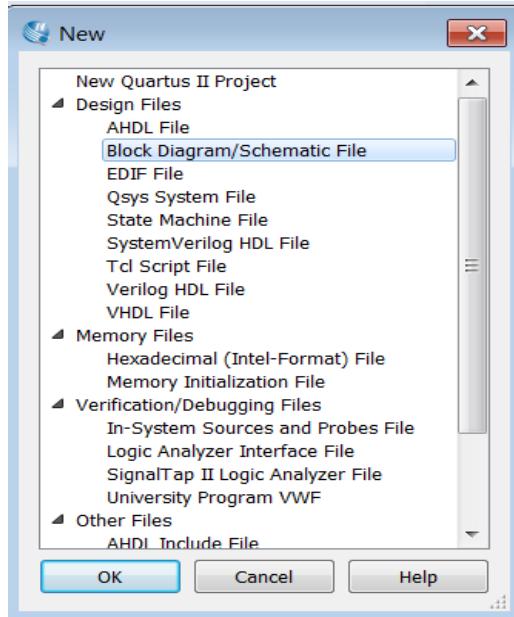


Click Finish

1.3.8 Adding Components to the Schematic file

The next step is to connect the components together.

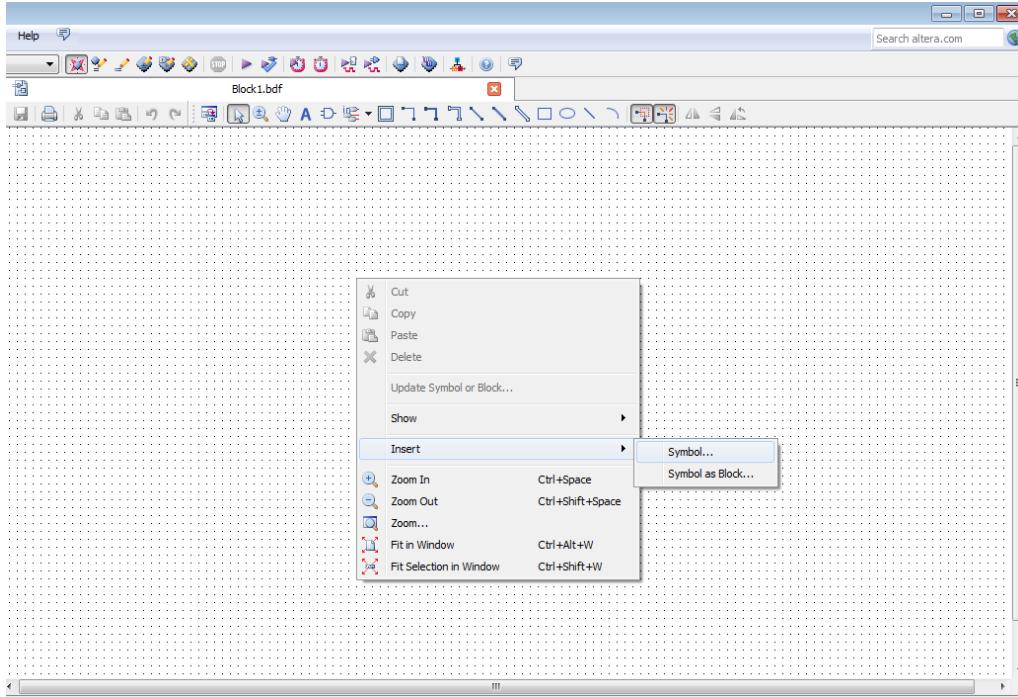
- 1.3.8.1 To do so, select the File menu - then select New and select Block Diagram/Schematic File



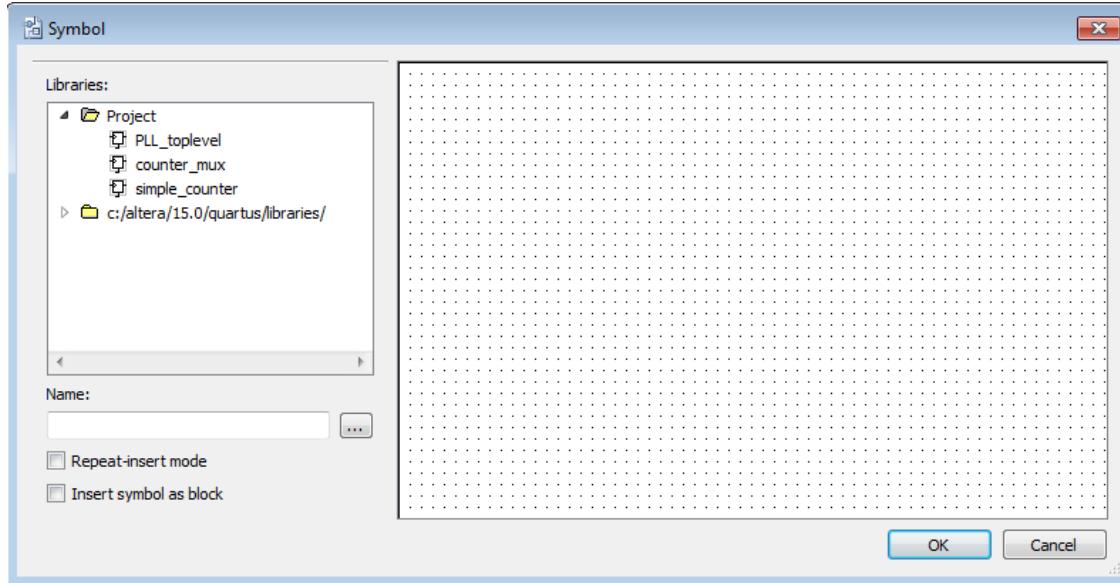
- 1.3.8.2 Click OK.

A new schematic page will be created, where the components can be added.

1.3.8.3 Right click on the schematic page, and select Insert Symbol, as seen below



1.3.8.4 Expand Project, where the three components that were created, can now be seen

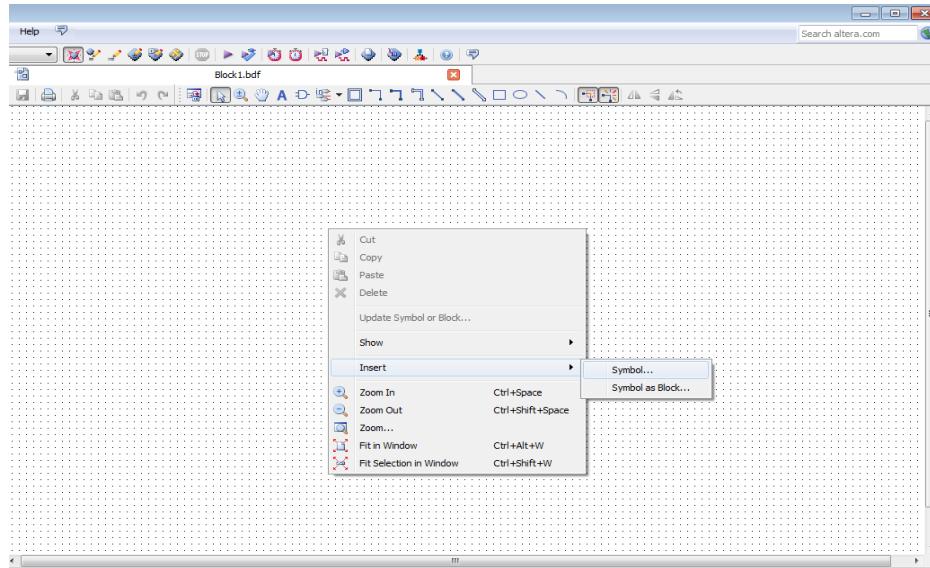


1.3.8.5 Select PLL_toplevel

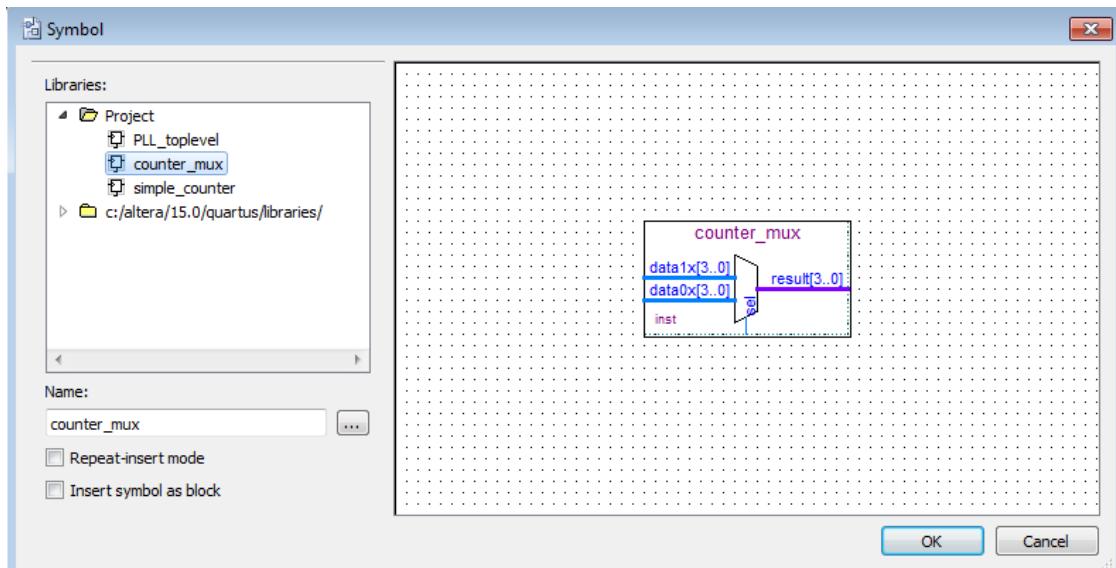
1.3.8.6 Click OK.

The PLL component is now added to the schematic page. The PLL component can be added anywhere for now.

- 1.3.8.7 To add the counter_mux, right click on the schematic page, select Insert Symbol, and select Symbol.
 (The order of adding components to the schematic does not matter, as the connections will in later steps connect the components correctly together)



- 1.3.8.8 Select the counter_mux, as shown



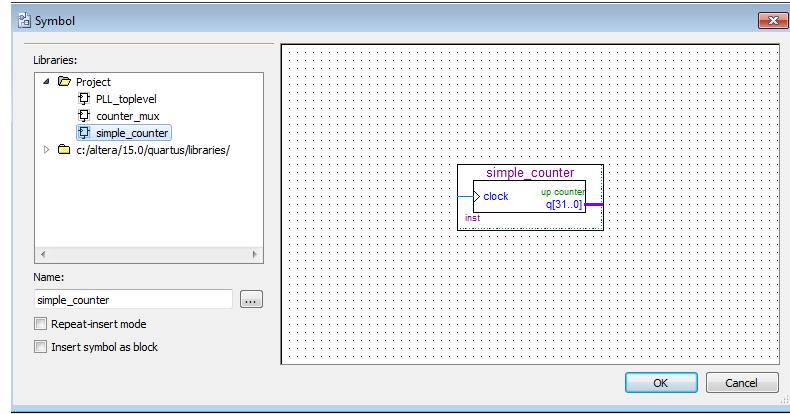
- 1.3.8.9 Click OK.

- 1.3.8.10 Put the counter_mux anywhere for now in the schematic page.

- 1.3.8.11 To add the counter, right click on the schematic page, select Insert Symbol, and select Symbol.

(The order of adding components to the schematic again does not matter, as the connections will in later steps connect the components correctly together)

1.3.8.12 Select the simple counter:

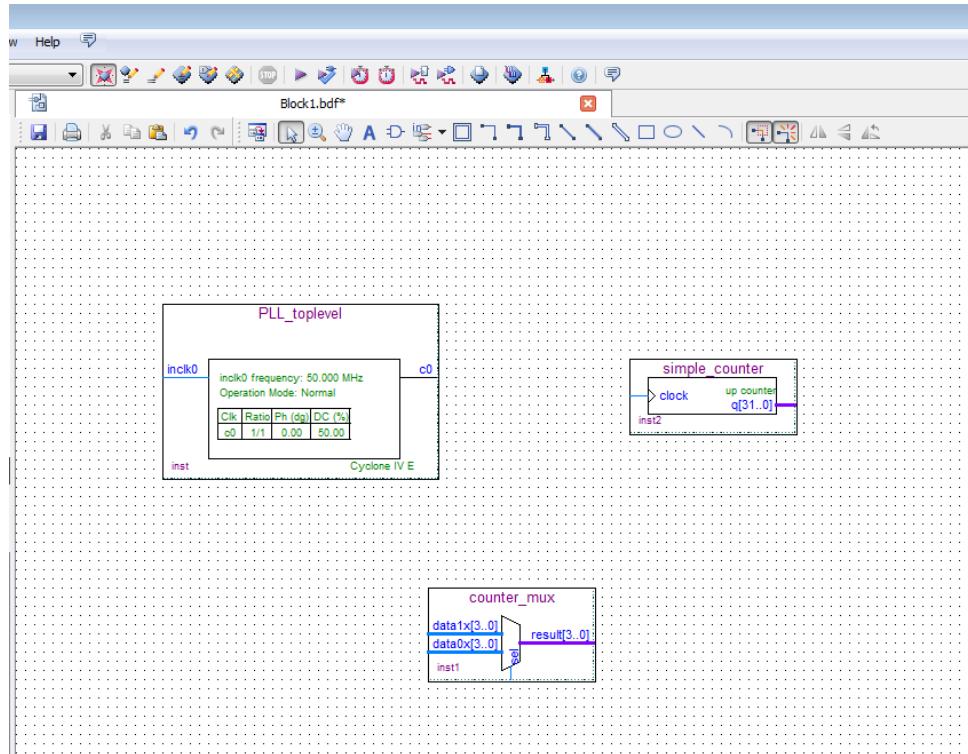


1.3.8.13 Click OK.

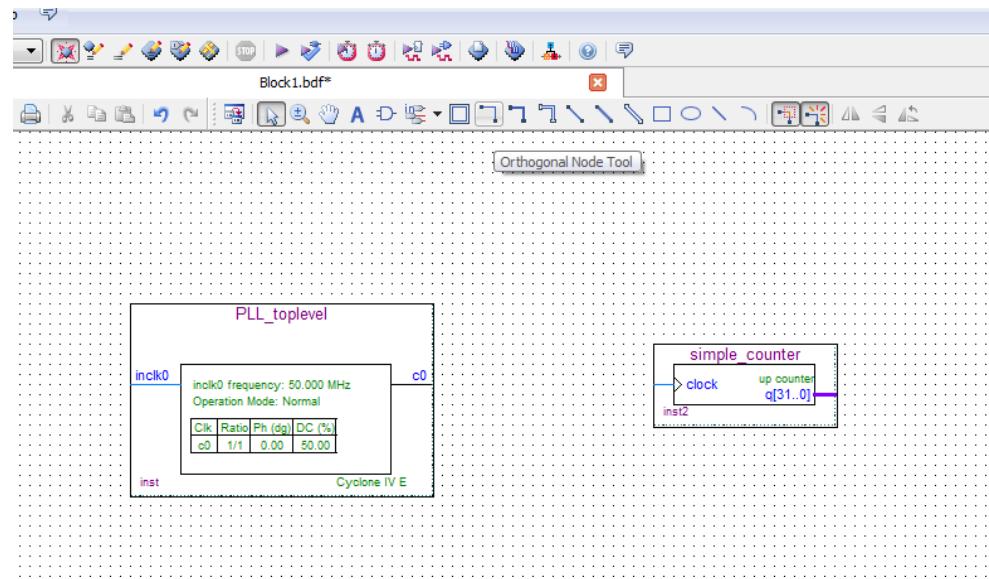
1.3.8.14 Put the simple_counter anywhere for now in the schematic page.

1.3.9 Connect the Components

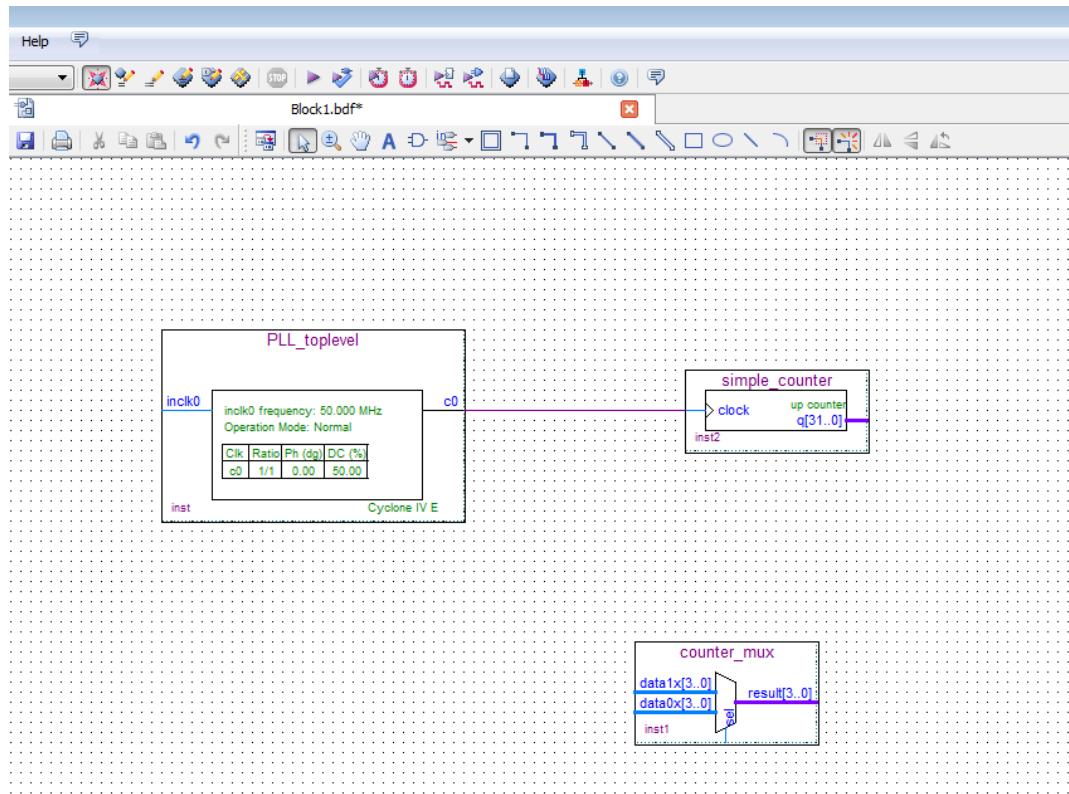
Since the components need to be correctly connected together to make the design, move the three components so that they look like the following. To move the components, select the component and drag it into the correct place.



1.3.9.1 Now select the Node Tool:

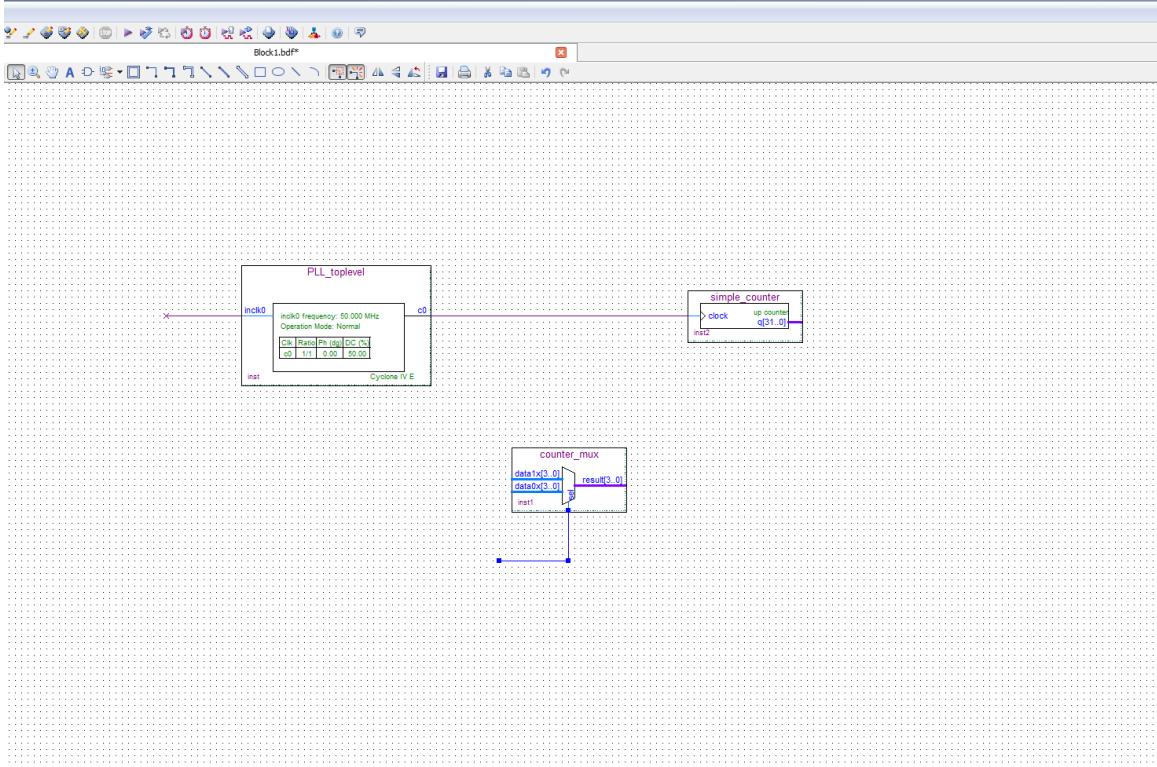


1.3.9.2 connect the c0 of the PLL_toplevel to the simple_counter :



This will mean that a single signal (c0) is connected to the simple_counter (clock).

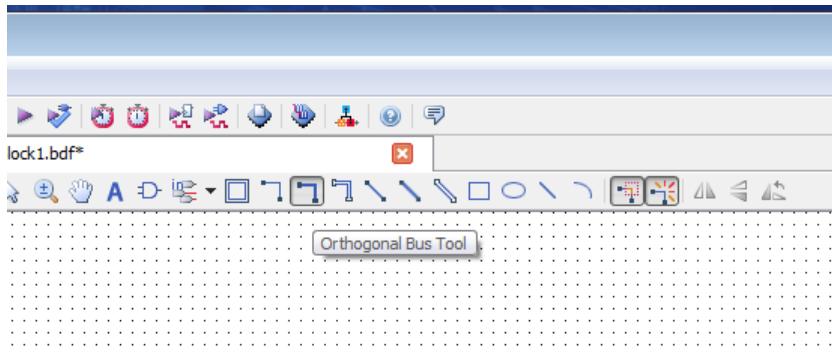
1.3.9.3 Using the Node line, make a connection on the counter_mux as:



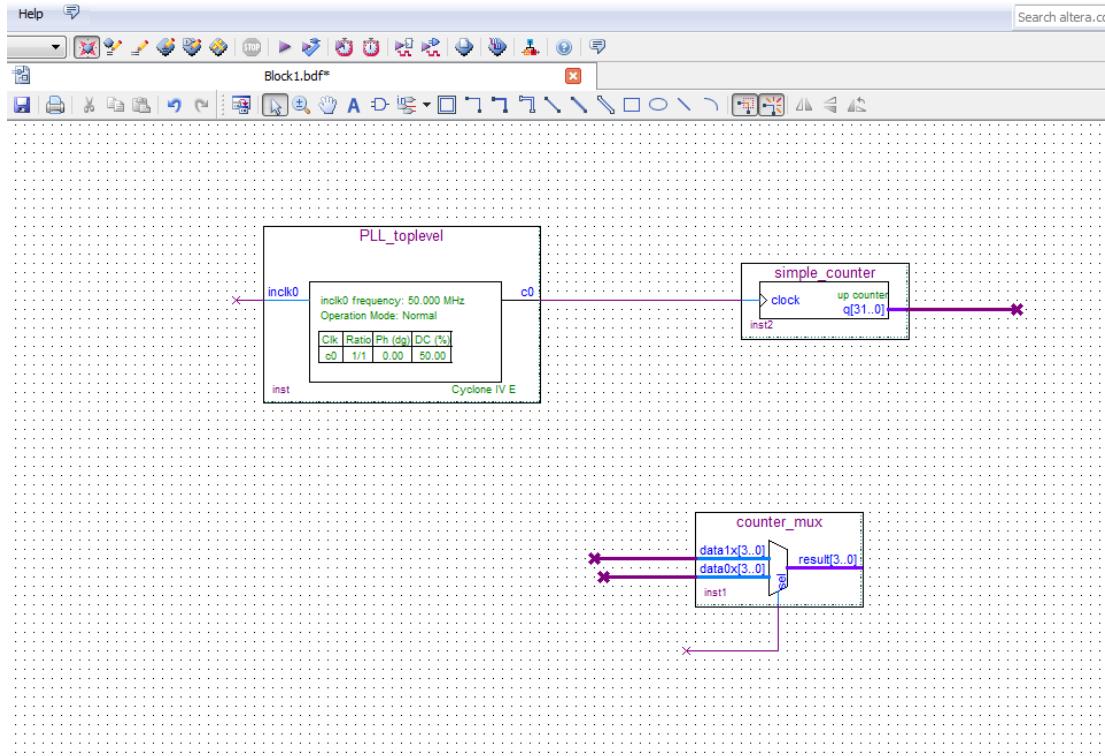
This makes it easier to add the input pin to the select of the mux.

Since the outputs of the counter and the input to the counter_mux are buses (multiple signals), the Bus tool will be needed.

1.3.9.4 Select the orthogonal bus tool:



1.3.9.5 Using the Bus tool, connect the counter outputs and the mux inputs as the following:

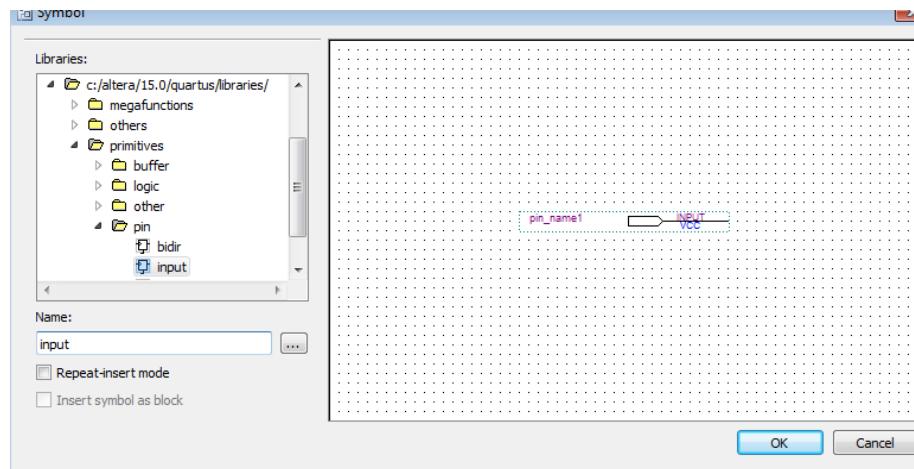


The next step will be to add pins (input and output pins).

1.3.9.6 Select the Arrow button (Selection Tool) as seen below:



1.3.9.7 Then select anywhere free in the schematic, select **Insert → Insert Symbol** and then under the Name write the word **input**

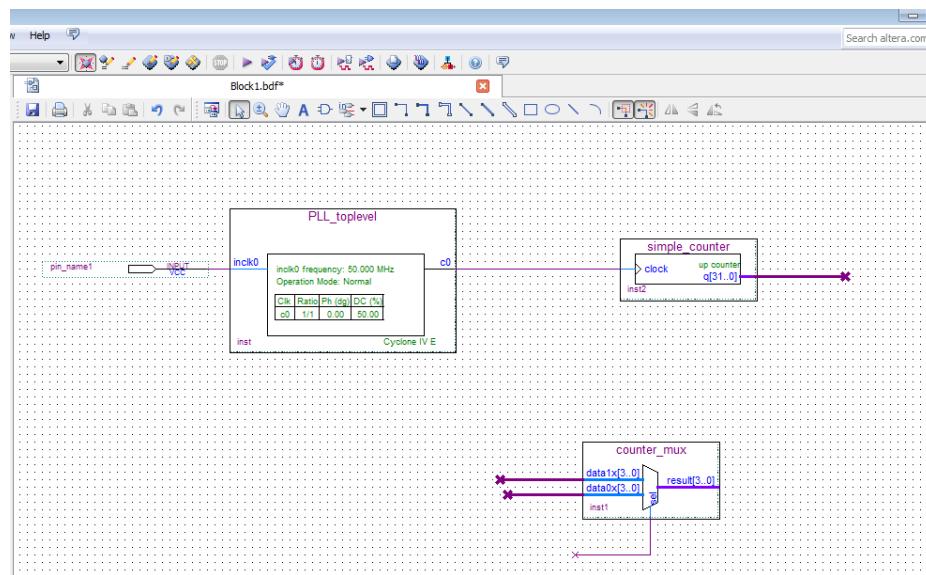


An input pin will appear.

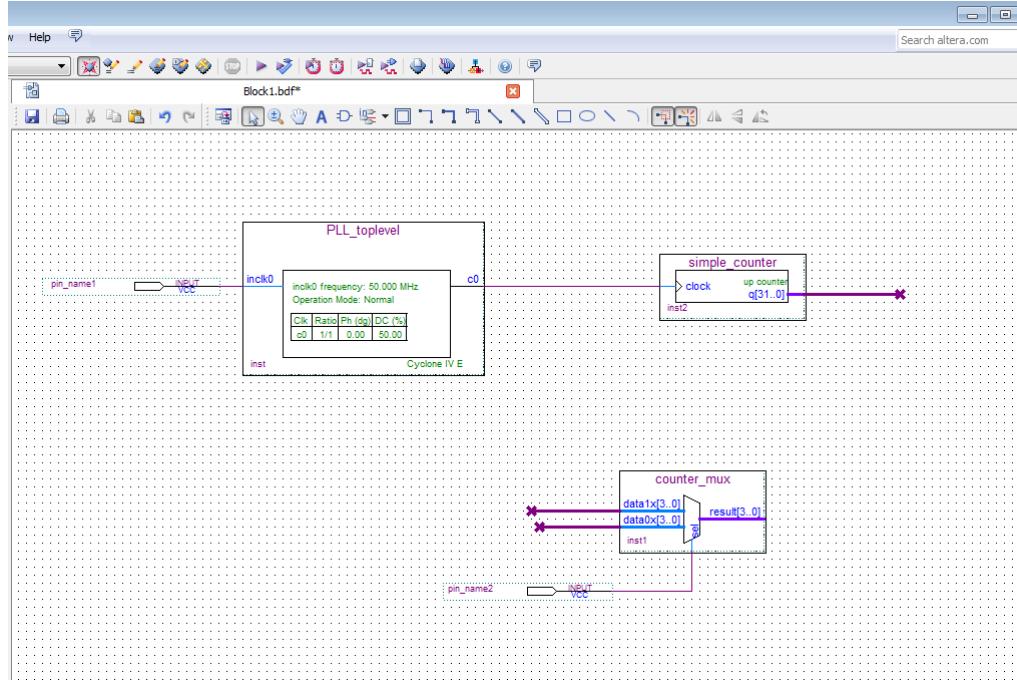
1.3.9.8 Click OK.

1.3.9.9 Connect the input pin to the PLL_toplevel as below.

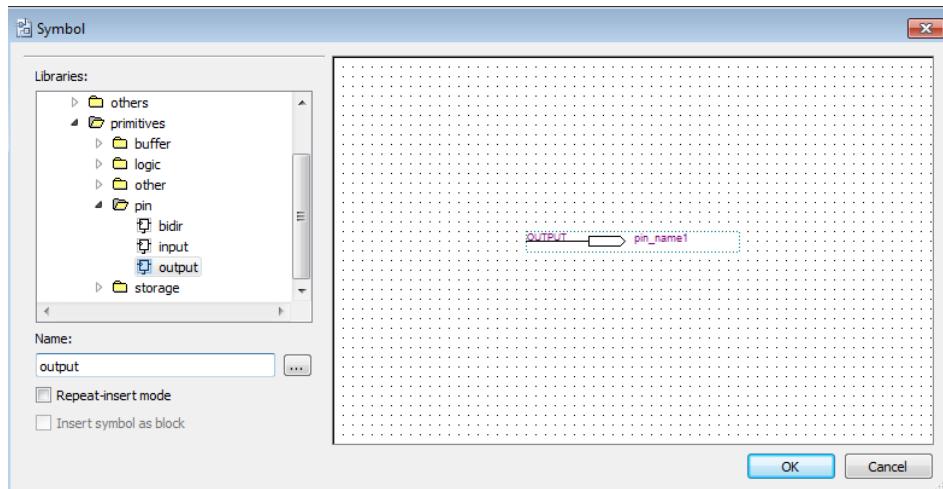
To make the connection, connect the input pin to the inclk0 of PLL_toplevel and then drag to the left. (This will make the connection for you).



1.3.9.10 Copy and paste the input pin and then connect it to the counter_mux:



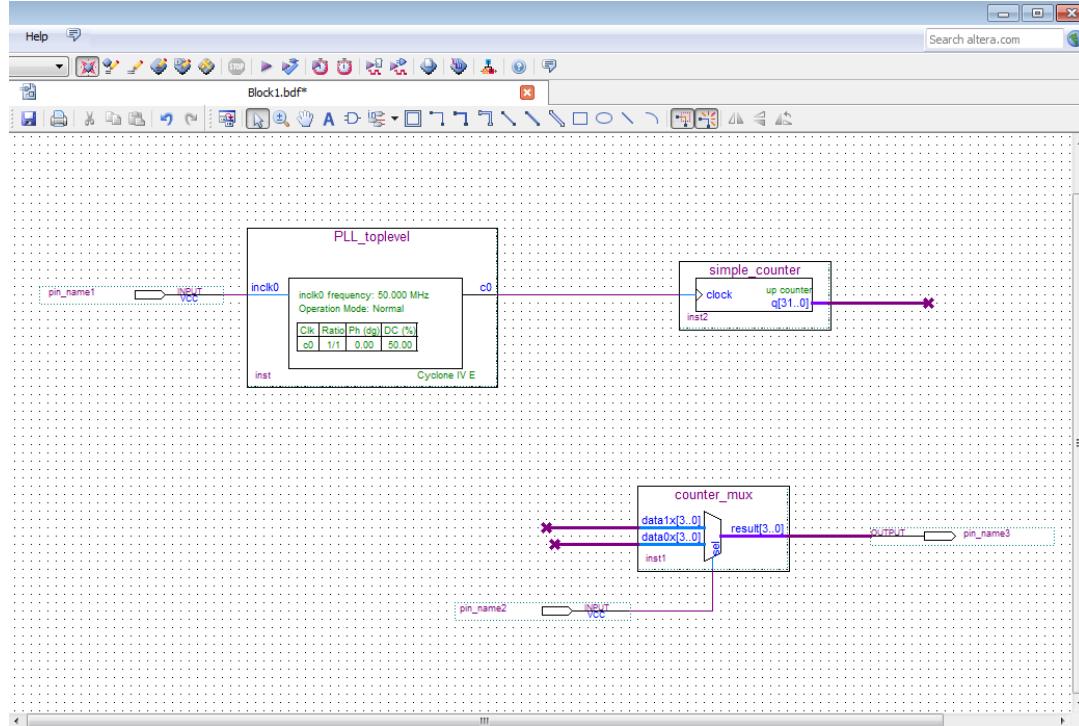
1.3.9.11 The next step is to add output pins. To add an output pin, right click on the schematic, select **Insert** → **Insert Symbol** and enter under the name output. This is an output pin



1.3.9.12 Click OK.

Rather than drawing the bus line manually, put the output pin next to the counter_mux output (result[3..0]) and then drag to the right. A bus connection will be created.

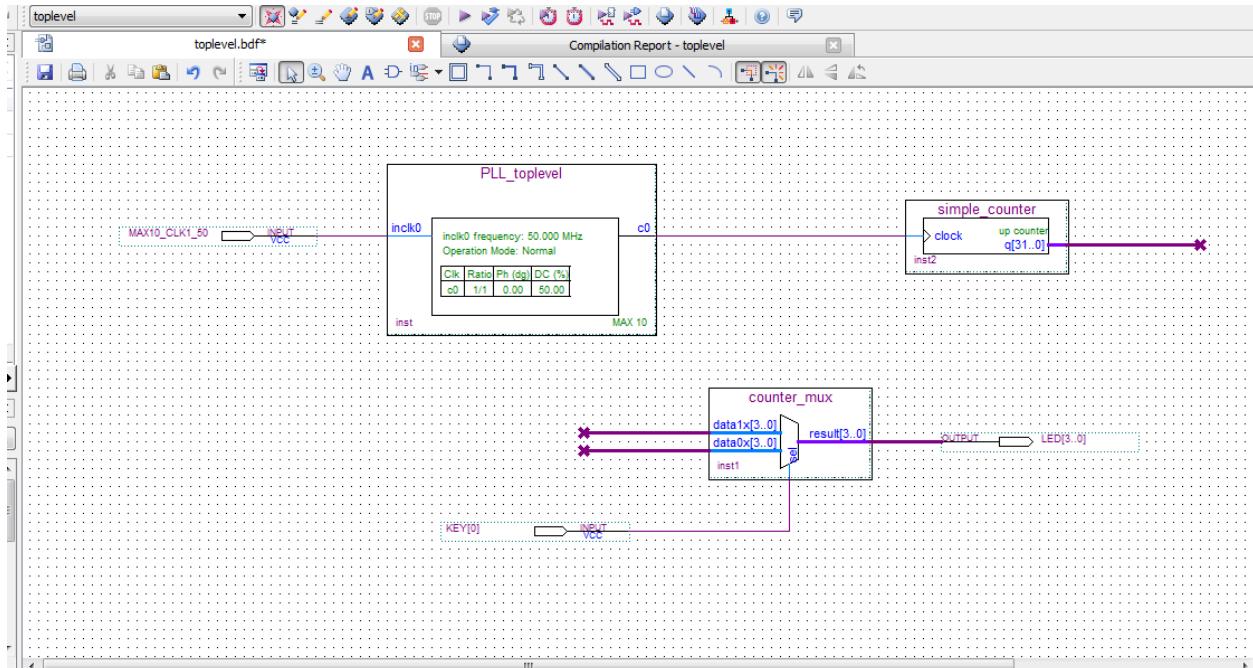
1.3.9.13 The output pin to the counter should now look like this:



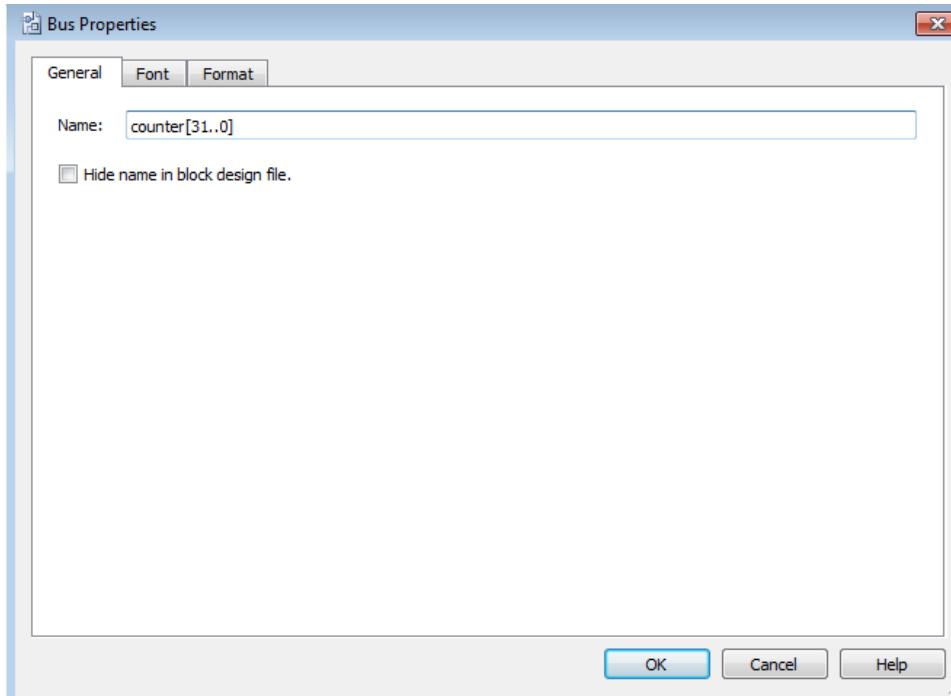
1.3.9.14 The next step will be to name the input and output pins, as well as the connection of the counter and mux.

- Select on the pin_name1 (input pin to the PLL_toplevel) and type the name to be **MAX10_CLK1_50**.
- Select on the pin_name2 (input to the counter_mux) and type the name to be **KEY[0]**.
- Select on the pin_name3 (output of the counter_mux) and type the name to be **LED[3..0]**.

1.3.9.15 The schematic should now look like this:



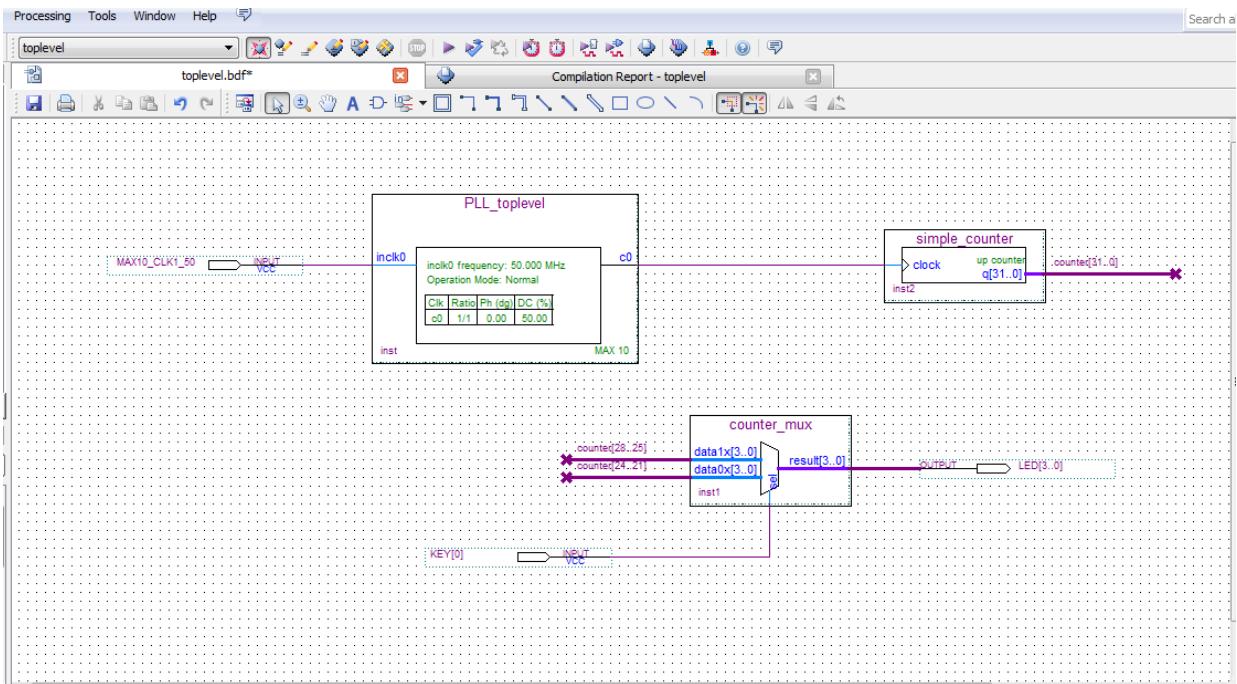
1.3.9.16 Label the connection of the counter and the mux. Select on the output bus from the simple_counter, right-click and select Properties. Add the name as counter[31..0] as seen below:



1.3.9.17 Click OK

1.3.9.18 This will label the bus signals.

1.3.9.19 Select on the upper bus signal of the counter_mux right-click and select Properties. Add the name as counter[28..25]. Select the lower bus signal of the counter_mux, right click and select Properties. Add the name as counter[24..21] as seen below.

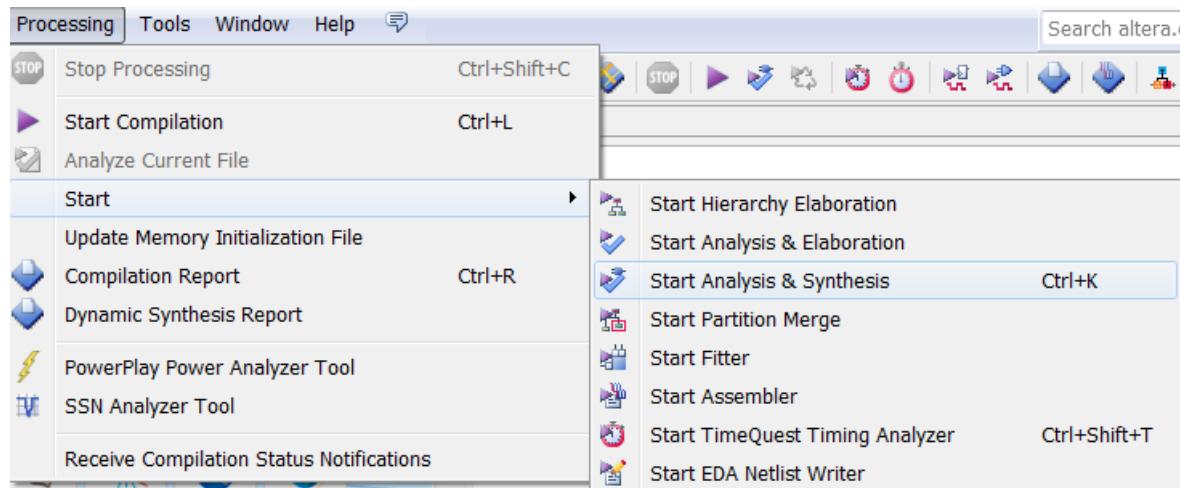


Even though the buses are not connected together by wires, the names of counter tell Quartus II to connect the signals together. Overall, the key will toggle between displaying higher 4 bits of the counter and 4 lower bits of the counter. The signals of the counter that are not connected will not be used by Quartus II.

1.3.9.20 Save your design. Use the File menu and select Save, and save it as `toplevel.bdf`. Click Save

1.3.10 Analysis and Synthesis

The next step is to run Analysis and Synthesis to ensure that there are no errors.: **Processing → Start → Analysis and Synthesis.**



There should be no errors. If there are errors, fix the error message and re run Analysis and Synthesis

1.3.11 Adding Timing Constraints

Timing Constraints tell the Quartus what the timing requirements are for this design. Timing Constraints are required in every CPLD/FPGA design.

To add timing, select **File → New** and under the Other File section, select Synopsys Design Constraints File and select OK.

Type the following lines in this new file:

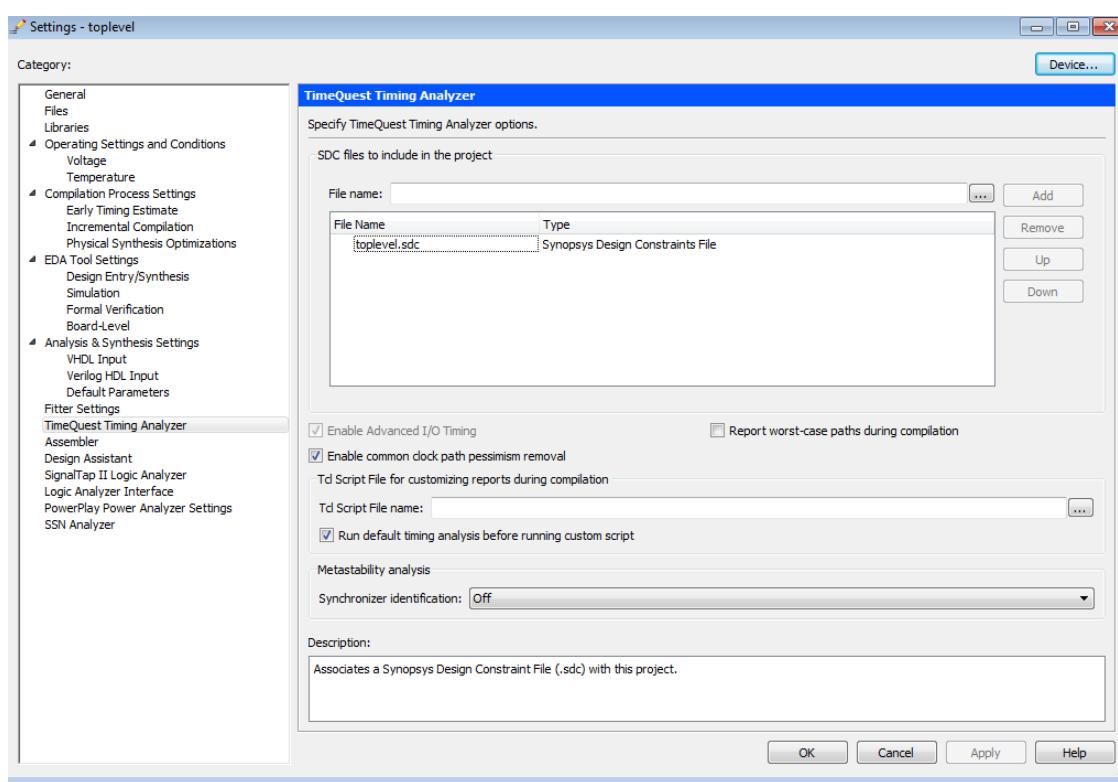
```
create_clock -name MAX10_CLK1_50 -period 20.000 -waveform { 0.000
10.000 } [get_ports { MAX10_CLK1_50}]

derive_pll_clocks

derive_clock_uncertainty
```

(Three separate lines will be needed).

- 1.3.11.1 The first line tells Quartus II that the clock, MAX10_CLK1_50 is 20 ns (50 MHz), with a 50/50 duty cycle (waveform of 0 10). It assigns the MAX10_CLK1_50 to a pin (port) in the .sdc format.
- 1.3.11.2 The second line tells the software to look if there are any PLLs, and if so, automatically derive the clock multiplication/division of the outputs of the PLL even if they are used internally within the CPLD/FPGA.
- 1.3.11.3 The third line tells the software to automatically determine the internal clock uncertainty. No clock is ideal, and thus there will be some internal jitter within the FPGA associated with it.
- 1.3.11.4 Use **File → Save** to save it as **toplevel.sdc**.
- 1.3.11.5 Ensure that this file is added to the **Project: Assignments → Settings** and select TimeQuest Timing Analyzer. The **toplevel.sdc** should be added already by default. (If it is not, it will need to be added).



1.3.12 Adding Pin Assignments

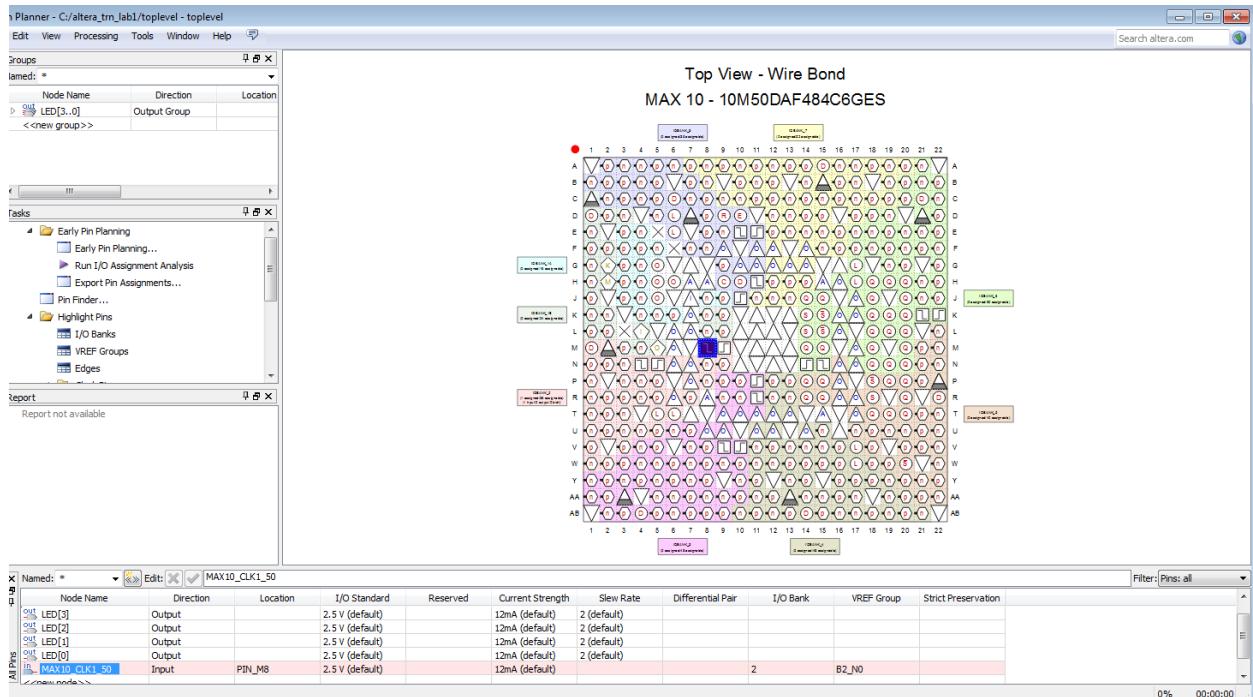
Before the design can be downloaded, pin assignments that match the hardware on the board are needed. There are different ways to do this, such as the Pin Planner, Assignment Editor, and text files.

The following steps will show one of these ways – the Pin Planner. Since there are only 6 pins that need to be assigned, the Pin Planner can be used. If many pins are needed, other ways can be used such as the Quartus Assignment Editor, or by importing constraints from a text file or spreadsheet.

1.3.12.1 Open the Pin planner: Assignments → Pin Planner. A new window will appear as seen below.



1.3.12.2 To make pin assignments, select the MAX10_CLK1_50 (node name) on the bottom portion (last node of the bottom window) and drag and drop it to pin M8. The window should now look like this:

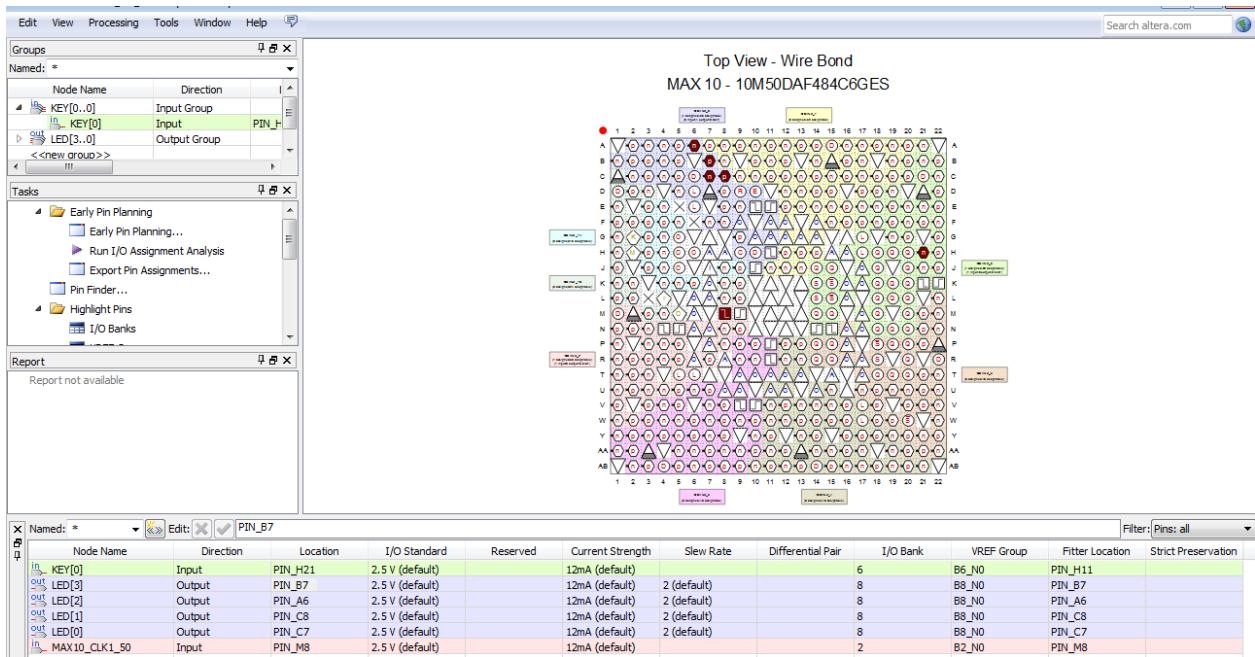


Note the Location for MAX10_CLK1_50 is now set to Location Pin_M8 (as seen in blue color in the top view of the FPGA).

1.3.12.3 The other pins need to be assigned as well. They can be assigned by dragging and dropping or even clicking in the location column (bottom of the window) and then using the pull down menu to assign the pins as follows (in order to match the board).

Node Name	Pin Location
KEY[0]	PIN_H21
LED[3]	PIN_B7
LED[2]	PIN_A6
LED[1]	PIN_C8
LED[0]	PIN_C7

1.3.12.4 After completing the pin assignments, the screen should now look like this:



The specific pins are now selected, but the I/O standards now need to be set as well. The switch, LEDS, and clock pins are different I/O standards (due to how they are connected on the board). The KEY[0] is a 1.5 V Schmitt Trigger, the LEDs are 1.2V and the clock pin is 2.5V.

These I/O standards can be set in the Pin Planner, by selecting the I/O standard at the bottom of the page, and change the 2.5 V (default) to the specific I/O standard.

1.3.12.5 The settings should now be as shown:

Top View - Wire Bond
MAX 10 - 10M50DAF484C6GES

Groups

Node Name	Direction	Group
KEY[0..0]	Input Group	PIN_H
KEY[0]	Input	PIN_H
LED[3..0]	Output Group	PIN_B
out LED[3]	Output	PIN_B
out LED[2]	Output	PIN_A
out LED[1]	Output	PIN_C
out LED[0]	Output	PIN_C

Tasks

- Early Pin Planning
 - Early Pin Planning...
 - Run I/O Assignment Analysis
 - Export Pin Assignments...
- Pin Finder...
- Highlight Pins
 - I/O Banks

Report

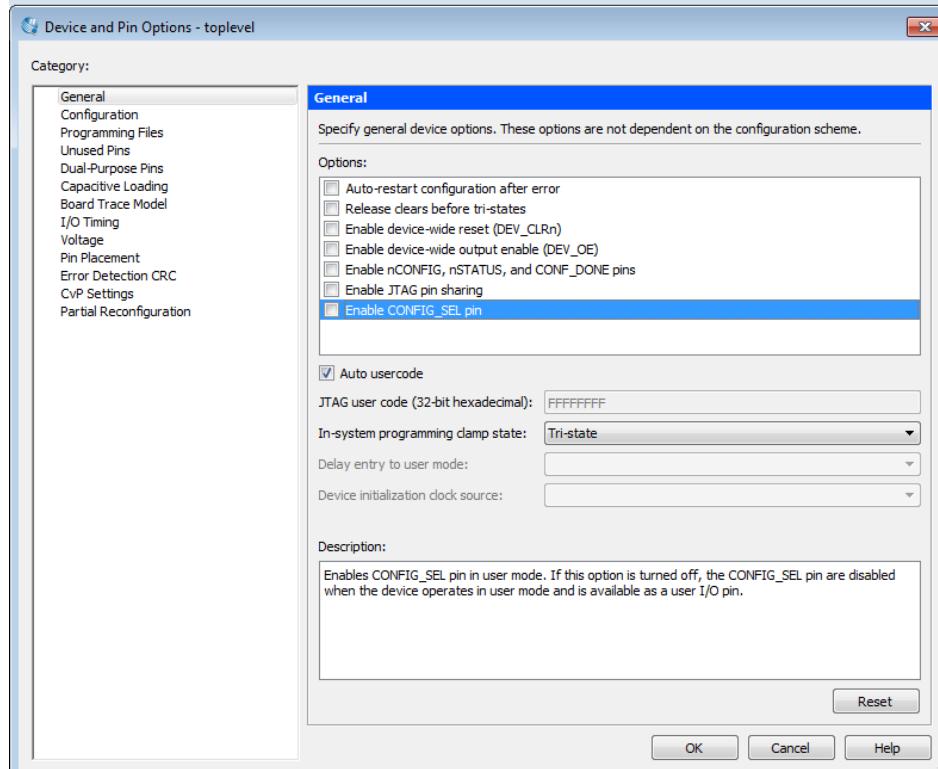
Report not available

Named: *

Node Name	Direction	Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	I/O Bank	VREF Group	Strict Preservation
in KEY[0]	Input	PIN_H21	1.5 V Schmitt Trigger		12mA (default)			6	B6_N0	
out LED[3]	Output	PIN_B7	1.2 V		12mA (default)	2 (default)		8	B8_N0	
out LED[2]	Output	PIN_A6	1.2 V		12mA (default)	2 (default)		8	B8_N0	
out LED[1]	Output	PIN_C8	1.2 V		12mA (default)	2 (default)		8	B8_N0	
out LED[0]	Output	PIN_C7	1.2 V		12mA (default)	2 (default)		8	B8_N0	
in MAX10_CLK1_50	Input	PIN_M8	2.5 V		12mA (default)			2	B2_N0	

1.3.12.6 Close this Pin Planner window. The pin assignments are automatically saved.

1.3.12.7 For the DECA board, another setting will need to be changed regarding one bank. Select **Assignments → Device** and select Device and Pin Options. Under General, disable all of the general device options. The window should now look like:



1.3.12.8 Click OK.

1.3.12.9 Select OK for the Device window.

1.3.13 Compiling the Design

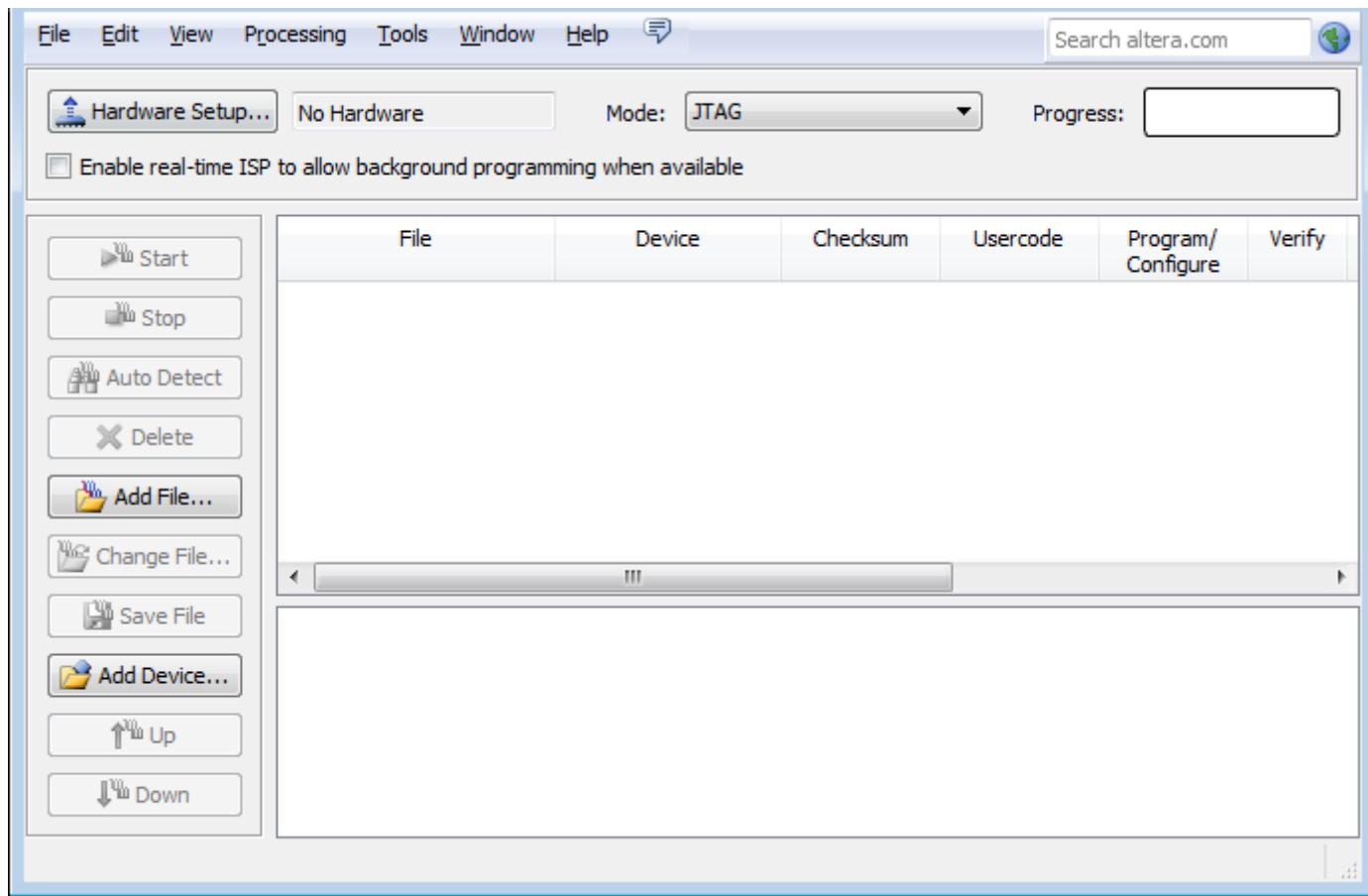
The next step is to compile the complete design. This step will verify there are no errors, create internal databases, as well as create a programming file that will be used in the next step.

1.3.13.1 Compile the design: **Processing → Start Compilation**

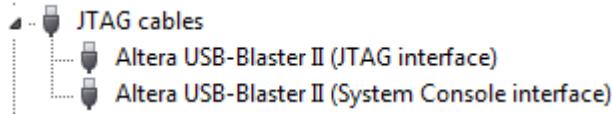
If there are errors, they will need to be resolved and re-compiled before the design can be programmed to the board. When Compiling finishes and there are no errors, there will be a message at the bottom of the window that states: Full Compilation was successful.

1.3.14 Configuring the Device

1.3.14.1 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.



1.3.14.2 Connect your DECA board to your PC using a USB cable. Be sure to connect it to the mini-USB connector labeled **USB2_J10** (on the bottom right of the board). Since the USB Blaster II driver software should already be installed, the Windows's Device Manager should display two entries under "JTAG Cables".

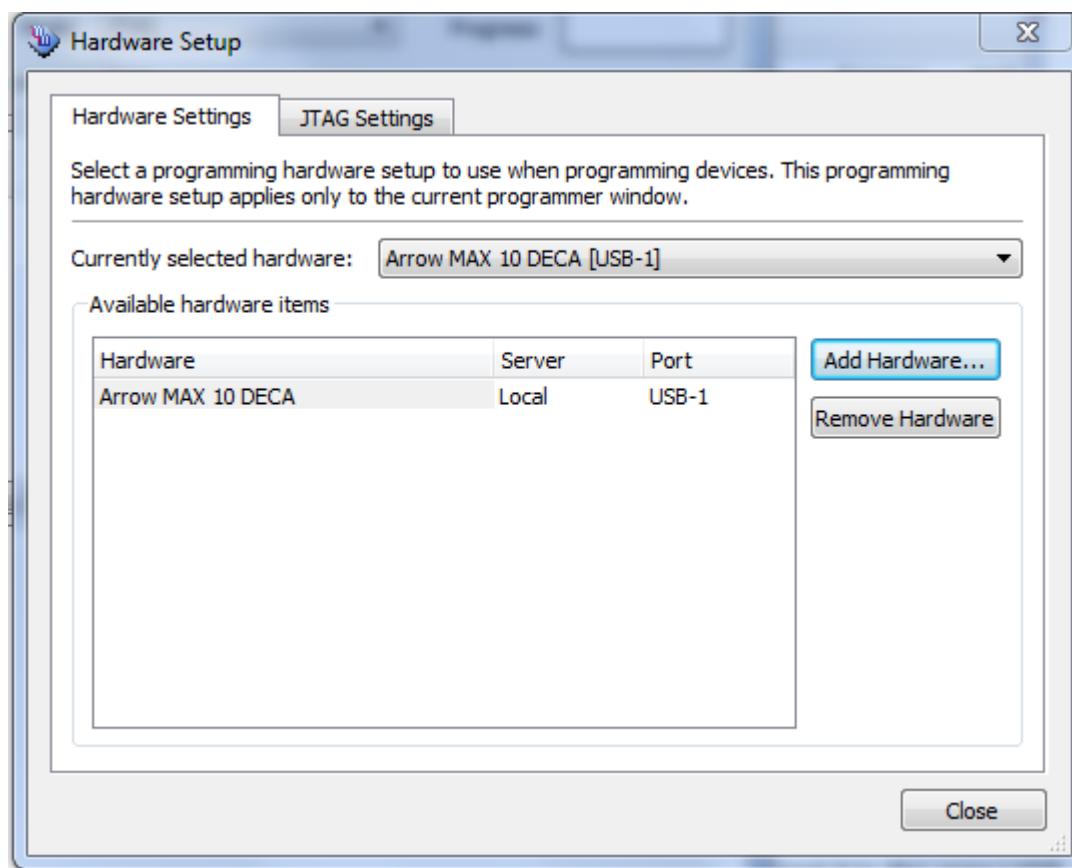


You should see a few LEDs light up on your DECA including the blue LED labeled **3.3v** and the green LED labeled **CONF_D**.

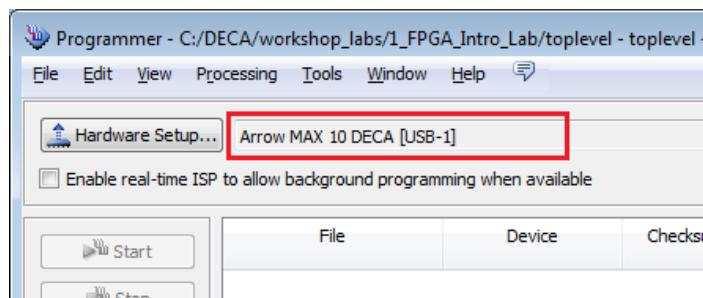


If the Device Manager shows an unconfigured USB Blaster, if Windows tries to look for drivers, or if the LEDs on the DECA do not light up, ask your workshop trainer for help.

- 1.3.14.3 In the Programmer window, click Hardware Setup and double-click the Arrow MAX10 DECA entry in the Hardware Settings tab. 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.

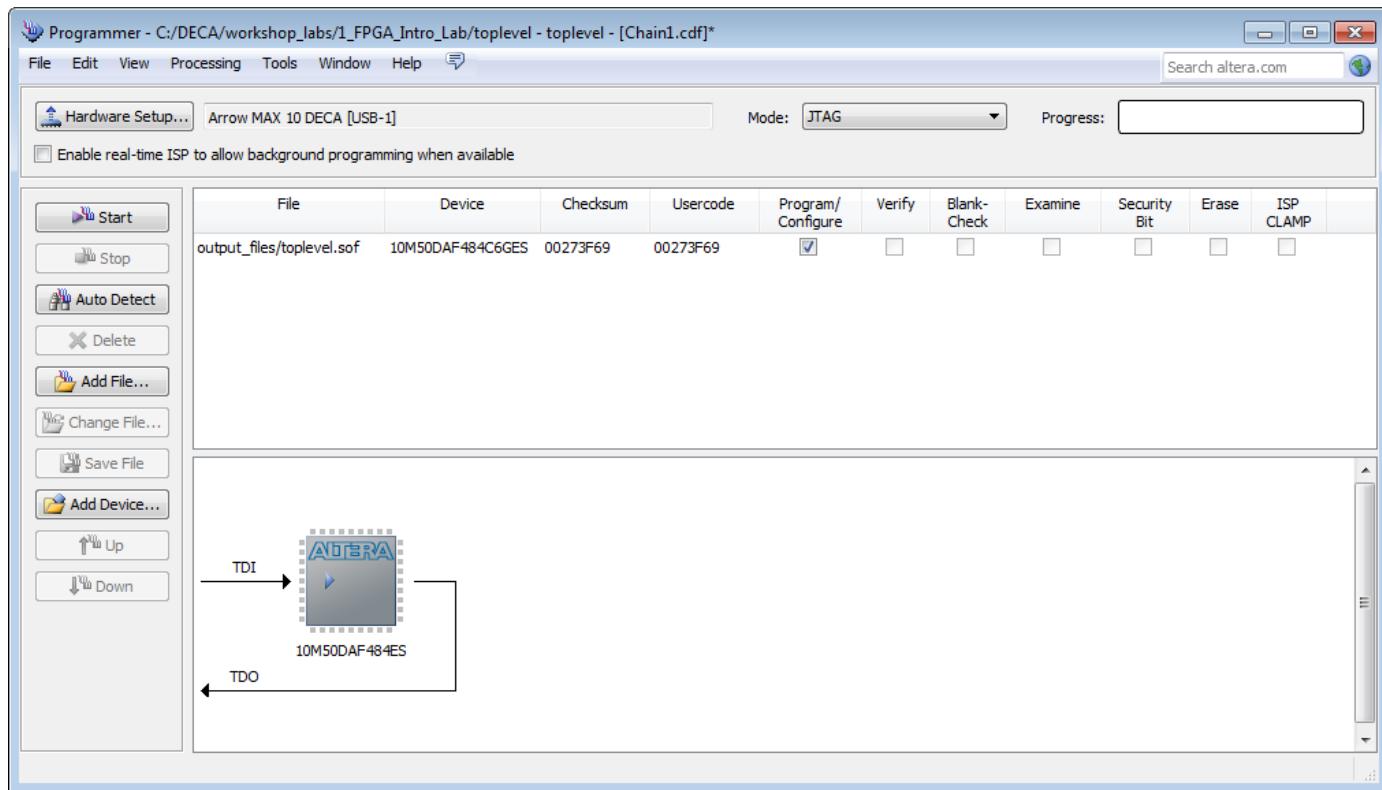


- 1.3.14.4 The programming window should now have a Hardware Setup such as:



1.3.14.5 Click "Add File..." and navigate to <project_directory>/output_files/ in your compilation directory. Open the **toplevel.sof** file.

1.3.14.6 The window should now look like this:



1.3.14.7 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).

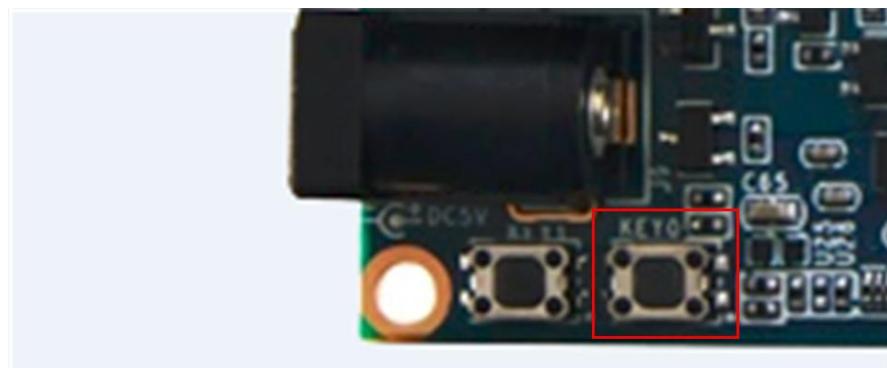
Progress: 100% (Successful)

The design is now programmed into the FPGA.

1.3.15 Testing the Design

On the board, by default, the LEDS should now toggle in a slow counting sequence. Note the LEDs are active low so the counter appears to be decrementing, not incrementing.

- 1.3.15.1 Push and hold the push button, KEY0, to see the LEDs now toggle in a very fast counting sequence. KEY0 is the second button to the right under the power connector, as seen below:



Releasing the push buttons should have the LEDs toggle in the slow counting sequence again.

After you are finished testing your design, you can disconnect the USB connector, and close Quartus II.

CONGRATULATIONS! YOU HAVE COMPLETED YOUR DESIGN.

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

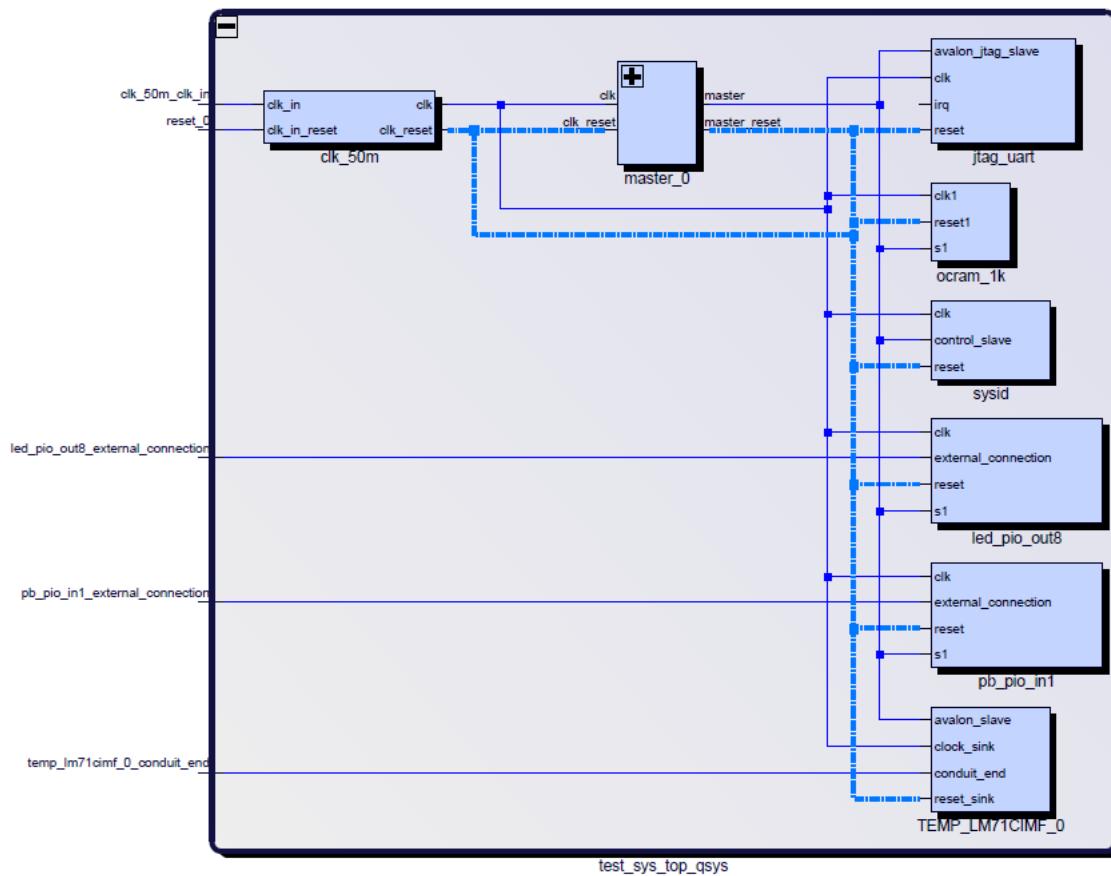
LAB 2. QSYS INTRODUCTION LAB

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:

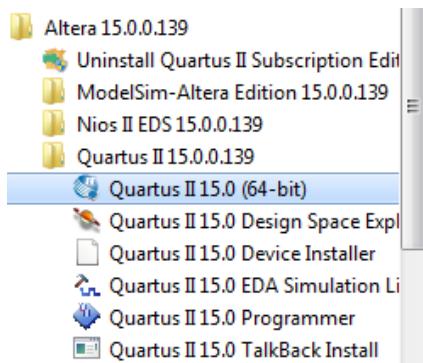


2.1 Set Up the Quartus II Project

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**

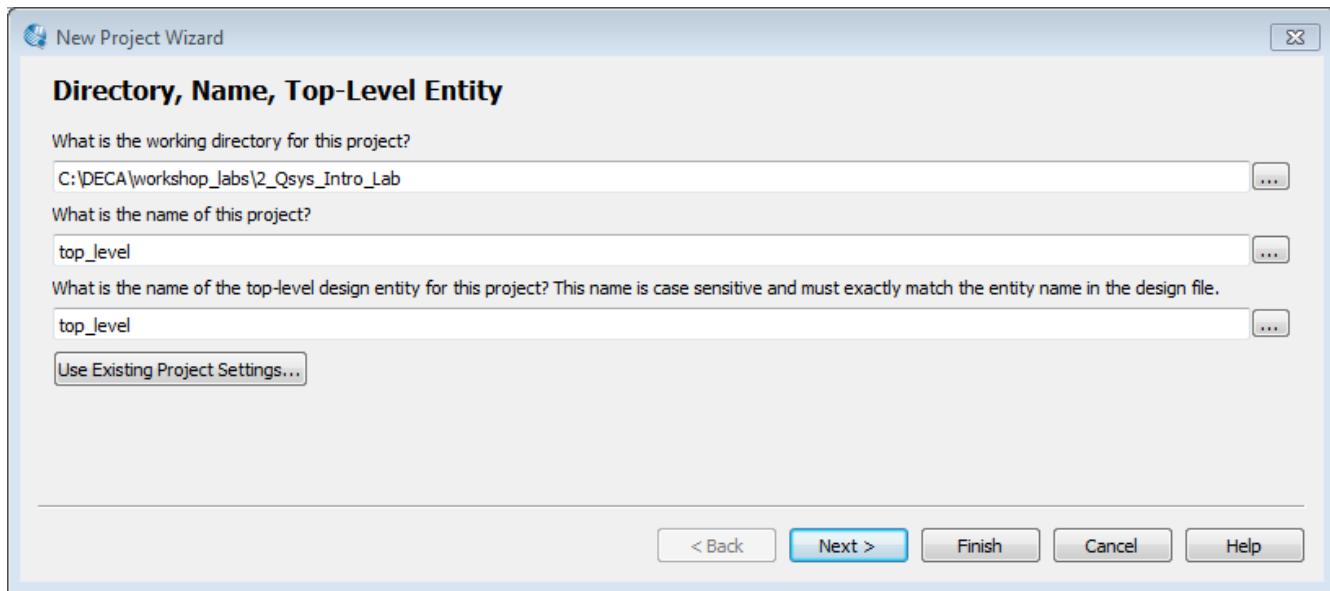
2.1.1.3 Configure the New Project Wizard directory, name, and top-level entity information.

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.)

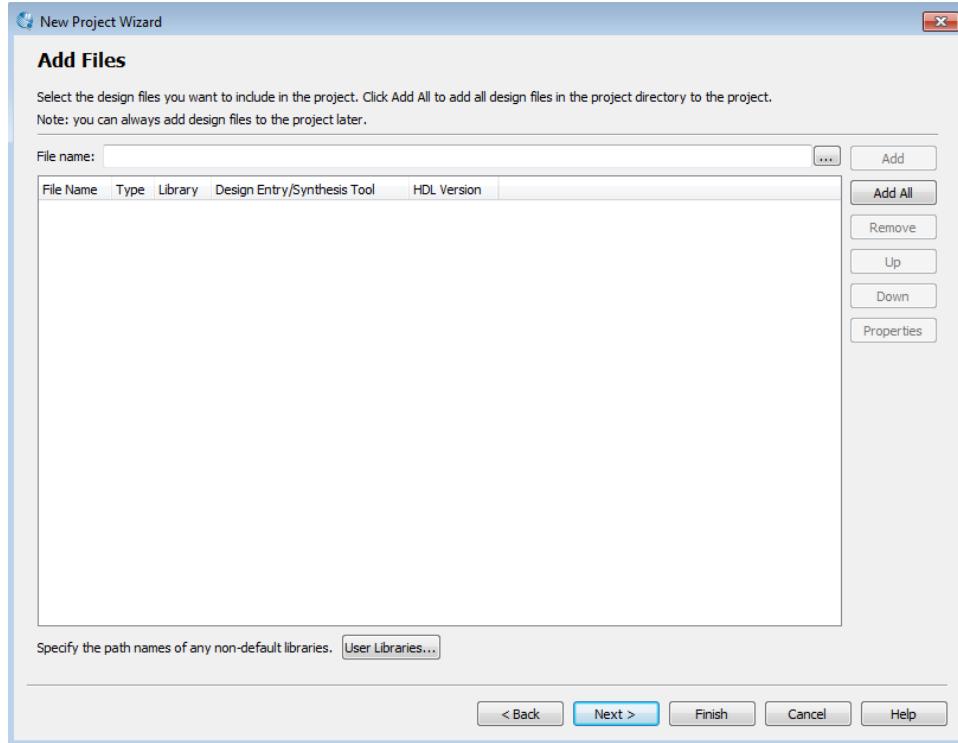


2.1.1.7 Click **Next >**

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

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.

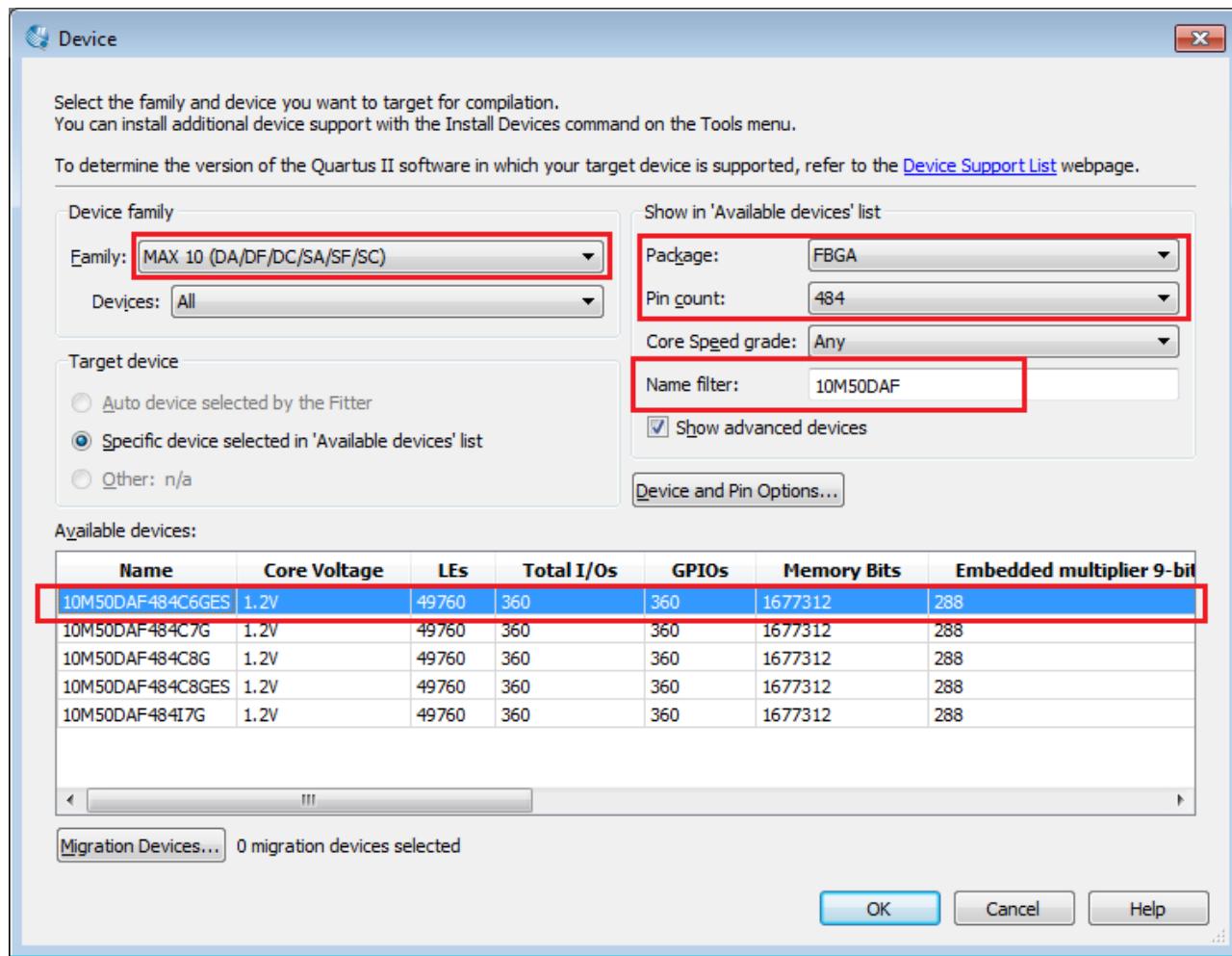


2.1.1.10 Click [Next >](#)

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.

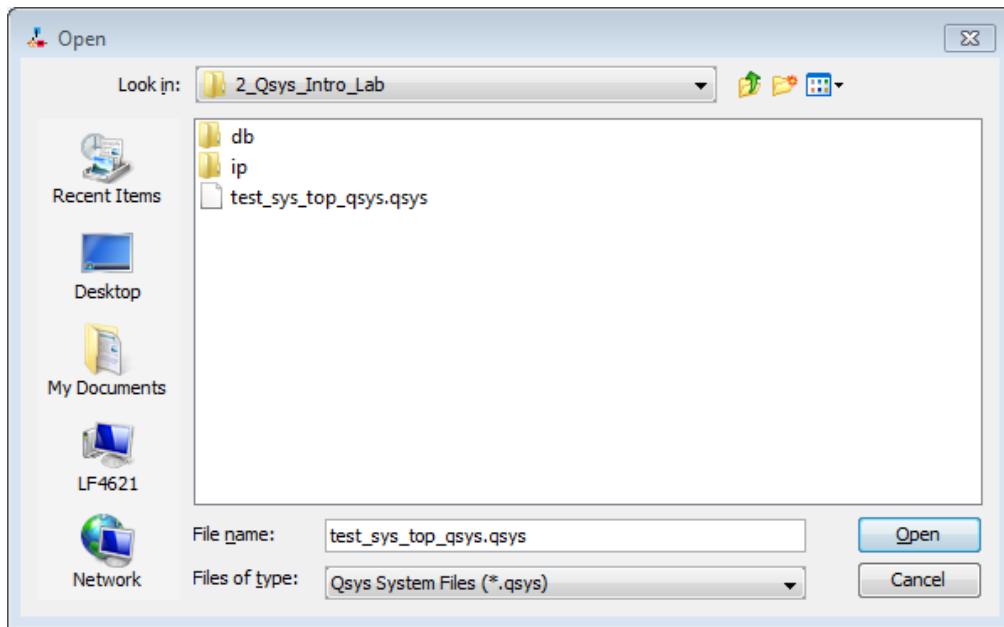
2.2.1 Launch Qsys

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**).

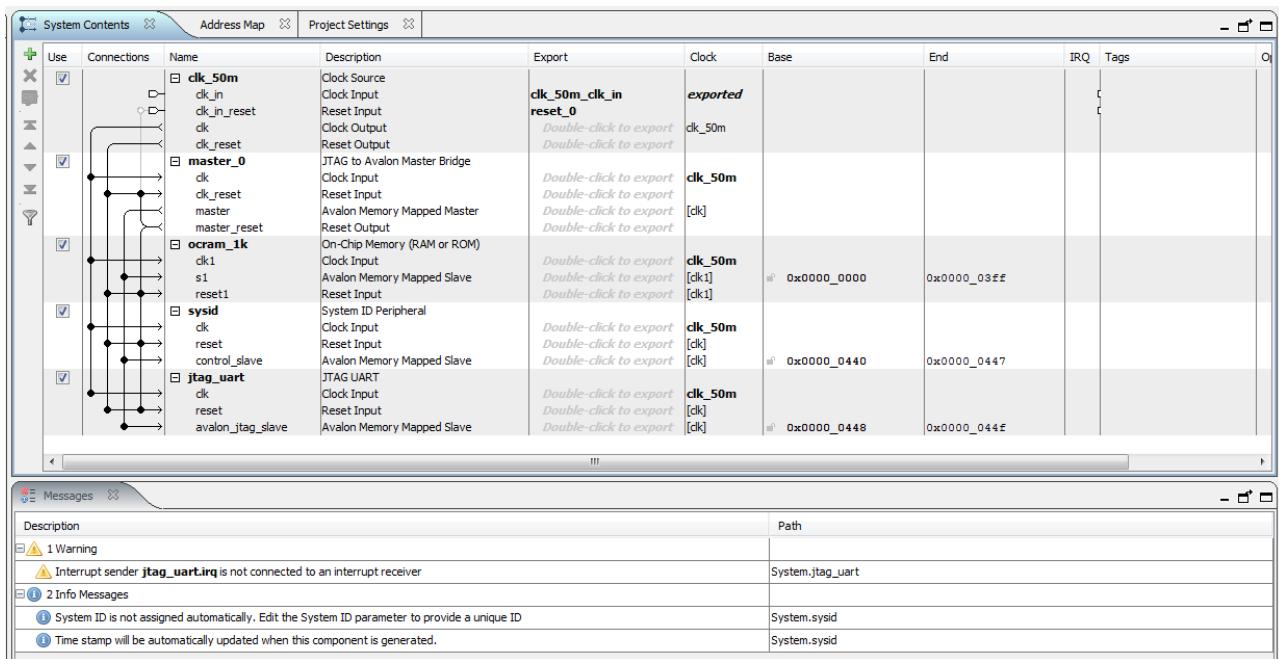
2.2.1.2 After Qsys launches, open the file named: **test_sys_top_qsys.qsys**



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:



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.

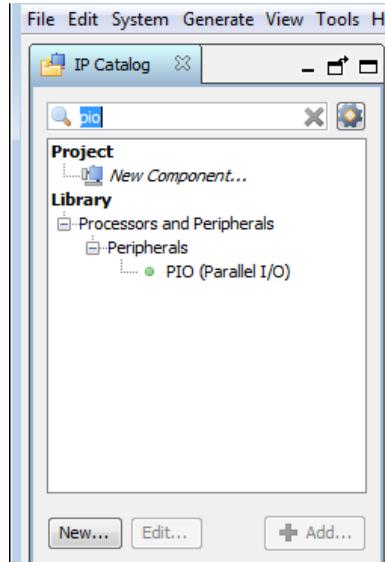
The On-Chip Memory (`ocram_1k`) is a memory mapped slave peripheral instantiated with 1KB of ram (using just one MAX®10 memory block)

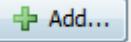
The System ID peripheral (`sysid`) is an IP that provides the Qsys system with a unique ID. It is an important peripheral to include when targeting soft and/or hard processors in the Qsys system because it allows the software development tools to validate that the software application is built for the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

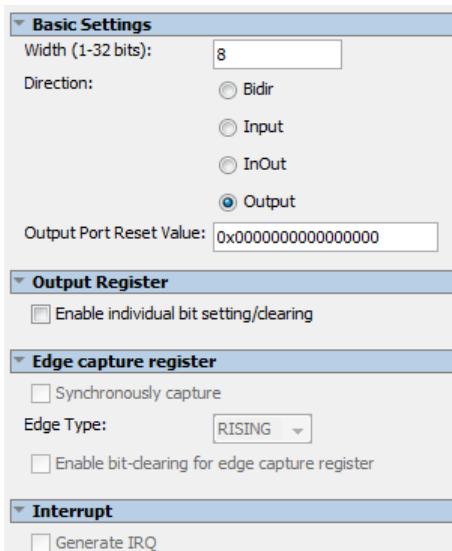
JTAG UART (`jtag_uart`) is a JTAG-based UART that processors use to print characters back to the console

2.2.2.2 Configure and add LED's

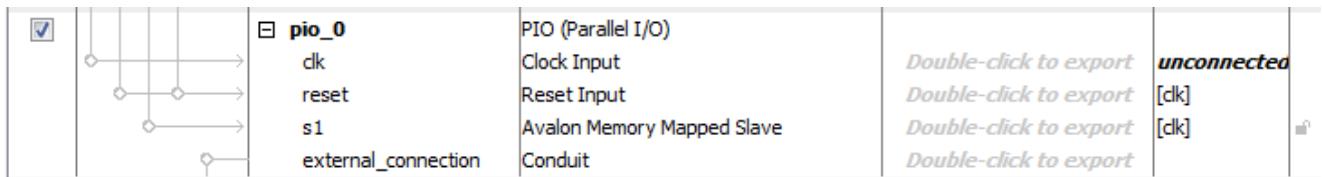
In Qsys, under the Library section of the IP Catalog, type: `pio` such as:



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

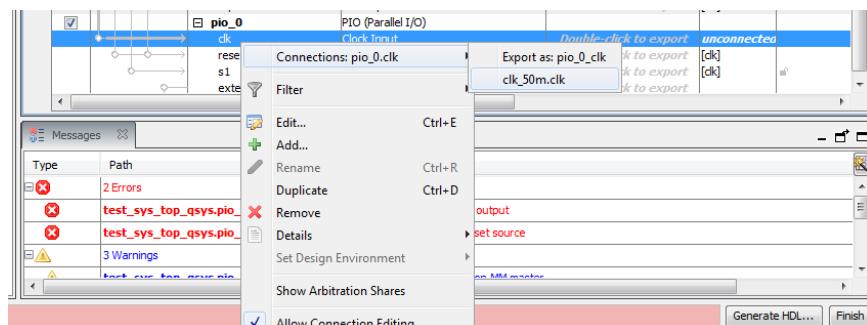


Select Finish. Your newly created PIO component will look like:



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:



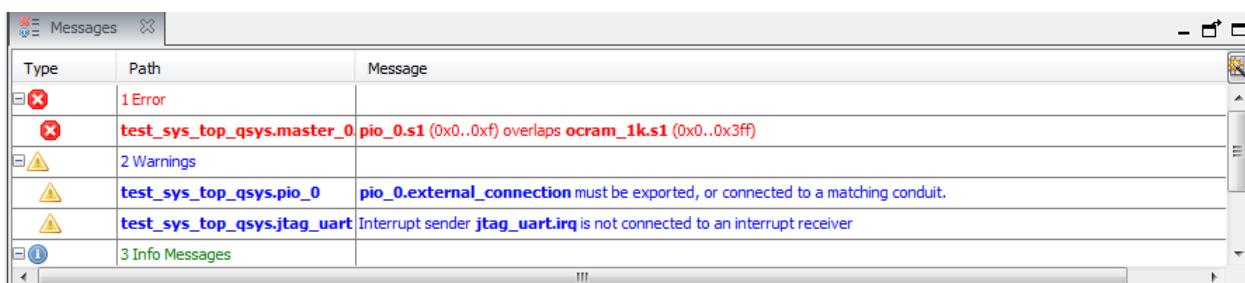
Select clk_50m.clk. The patch panel of the PIO will now be connected to the clock input. To complete the peripheral connections, repeat the step but this time:

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

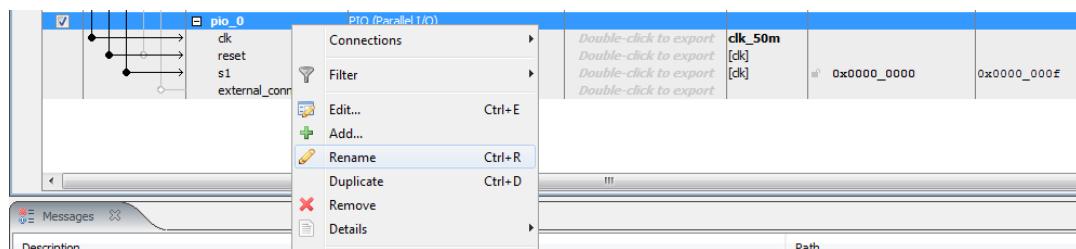
Connecting the reset to both the *clk* and *master* reset domains allow this pio to be reset if either reset occurs.

- Connect the pio_0.s1 to master_0_master.

There will be an error that states:



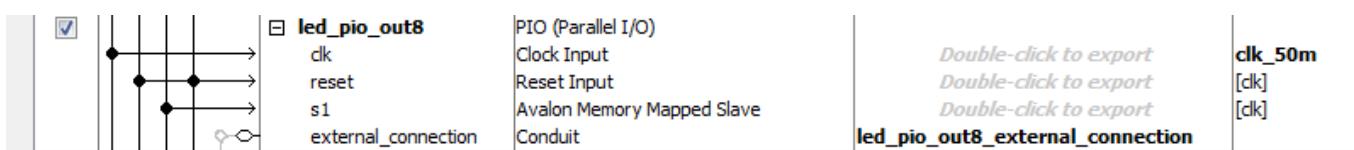
This error will be removed later on. The next step is to change the name of the component. To change the name, right click on the pio_0 and select Rename:



Change the name to be **led_pio_out8**.

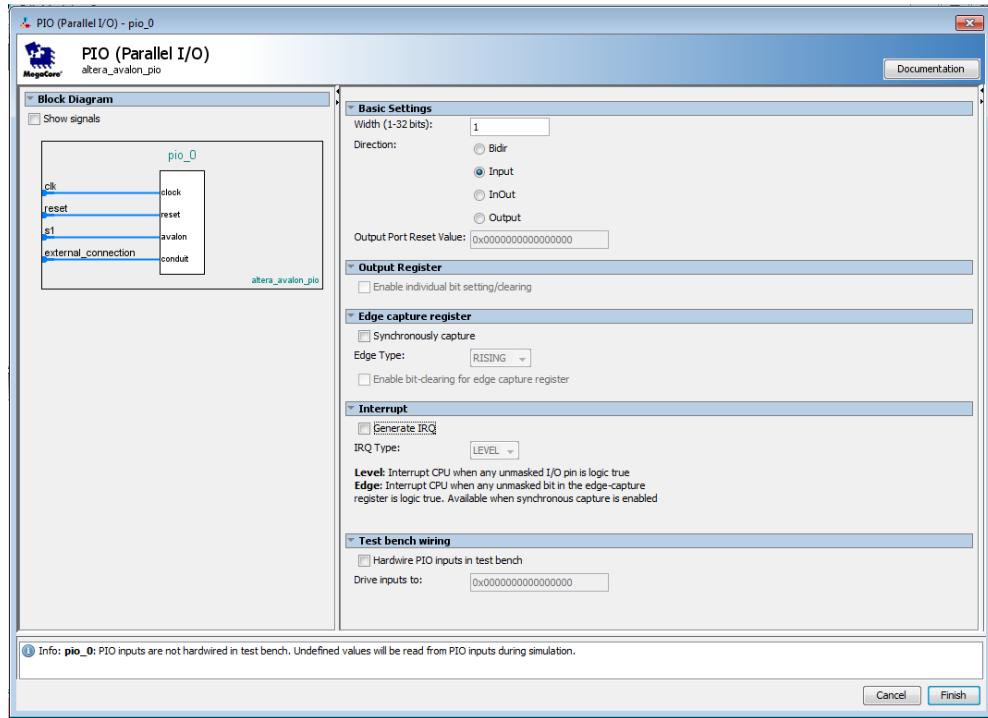
The next step is to export the led signals to the top-level design so that we can connect these signals to FPGA pins. To do this, double-click on the export signal. The name of **led_pio_out_8_external_connection** should now be present.

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)



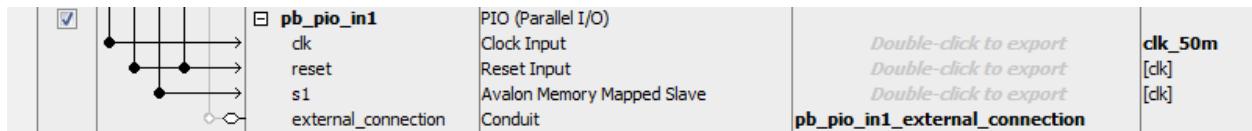
Select Finish to complete the generation of the new pio peripheral.

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.

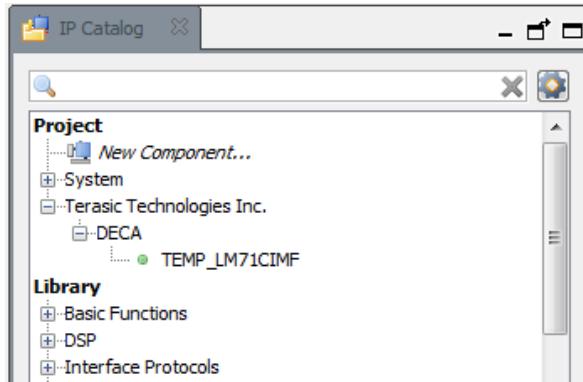
Lastly, we need to export the pio to the top-level design. Double click on the export signal. The signal of **pb_pio_in1_export_connection** should automatically be created. The completed pb_pio_in1 peripheral is shown here:



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)

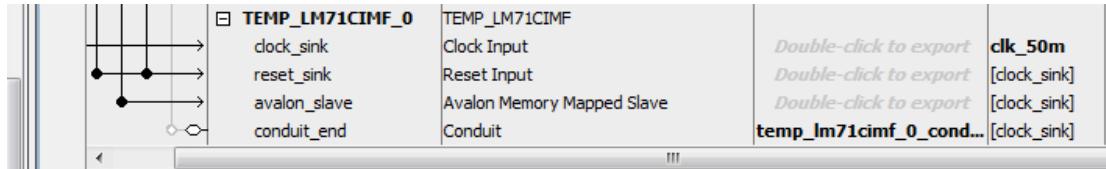


Select the TEMP_LM71CIMF and select Add... then Finish

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:

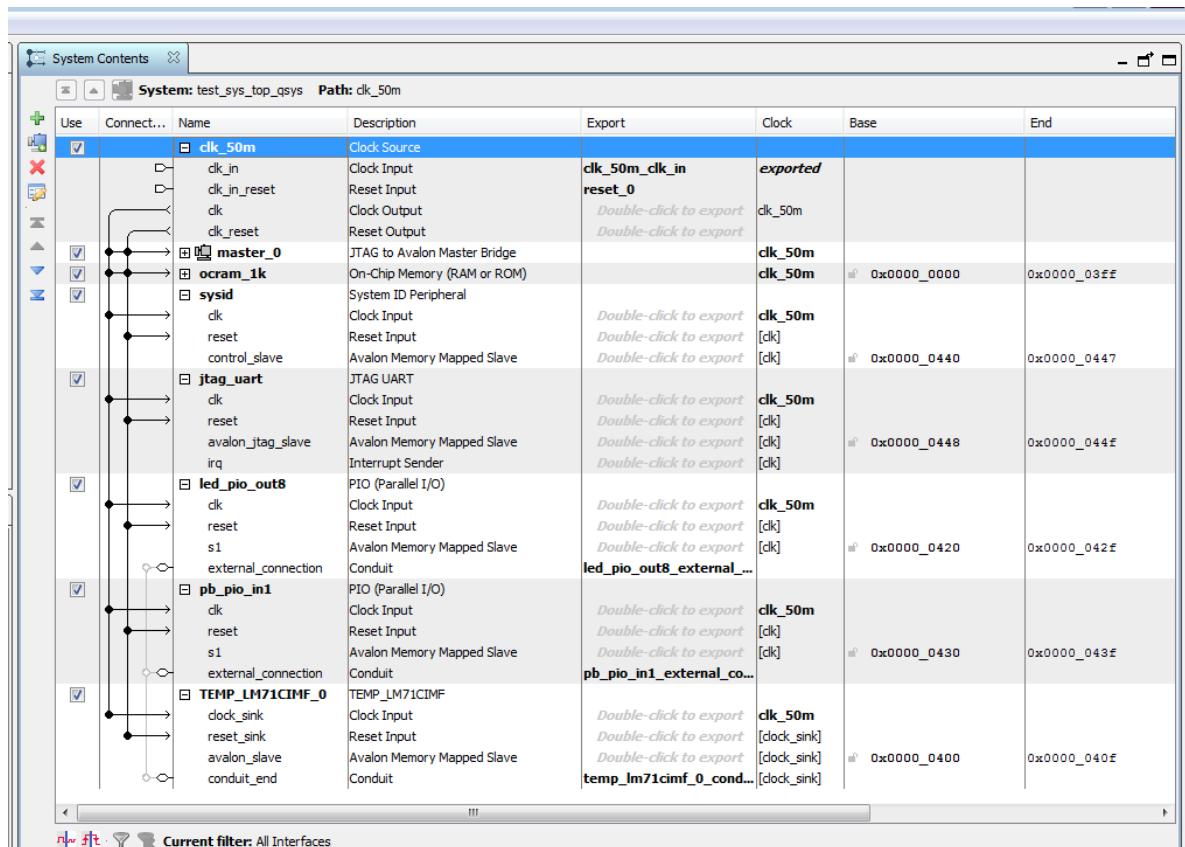


Note that there will be an error about the overlapping peripherals in the memory space. This will be resolved next.

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.



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.

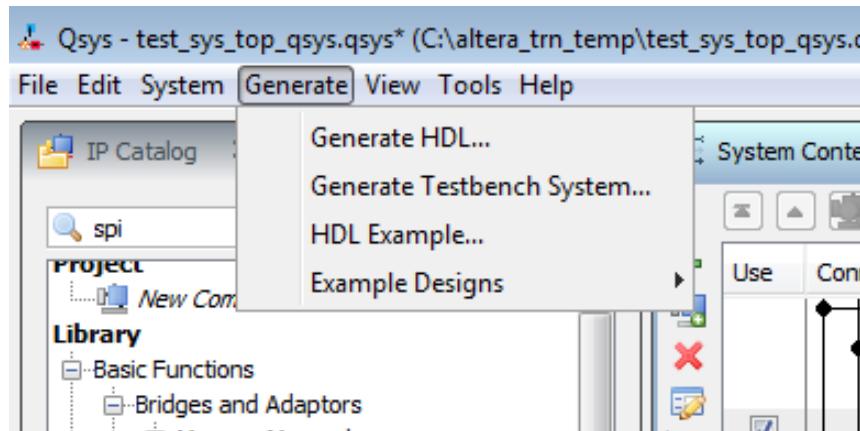
Should you have accidentally selected interrupts in the components, you may see warnings that irq (interrupts) are not connected to interrupt receiver. For this workshop, interrupts are not used. However, for other designs, interrupts can be implemented.

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

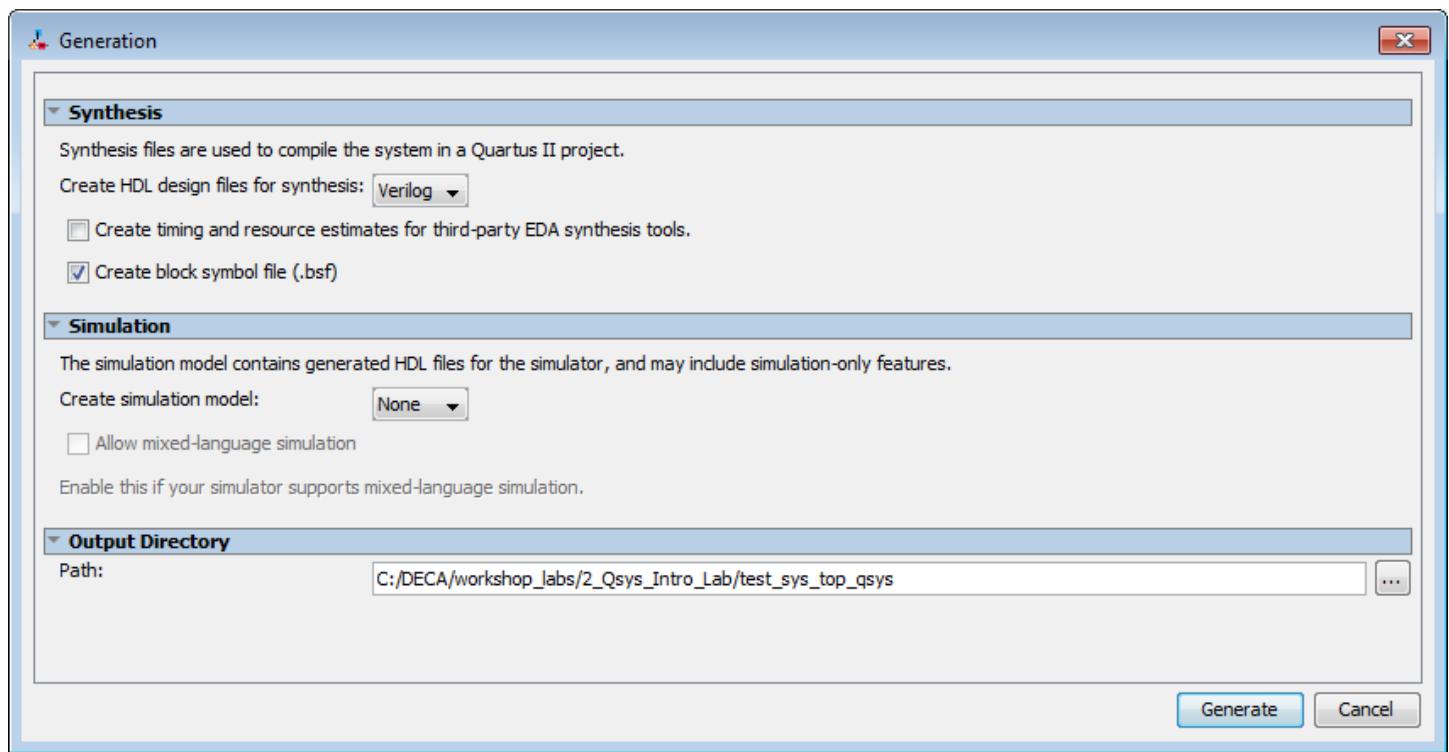
2.2.3 Generate RTL for compilation

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:



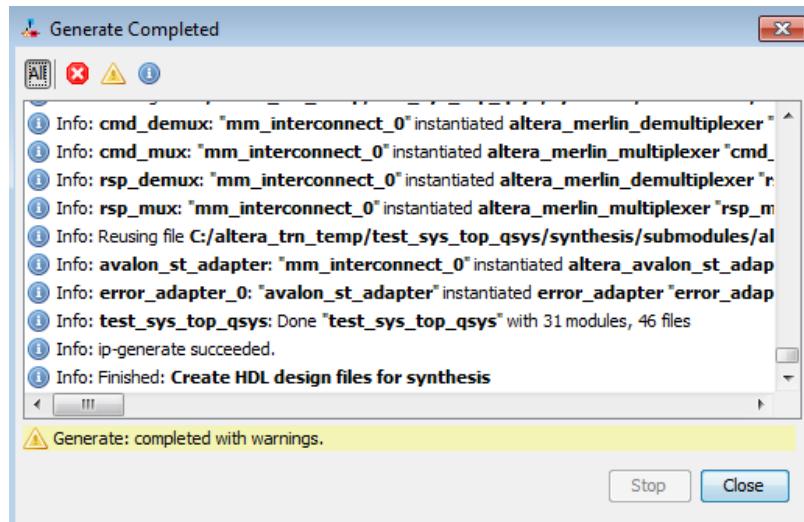
2.2.3.2 A new window, Generation, will appear. Select Verilog for the Create HDL design files for synthesis.



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".

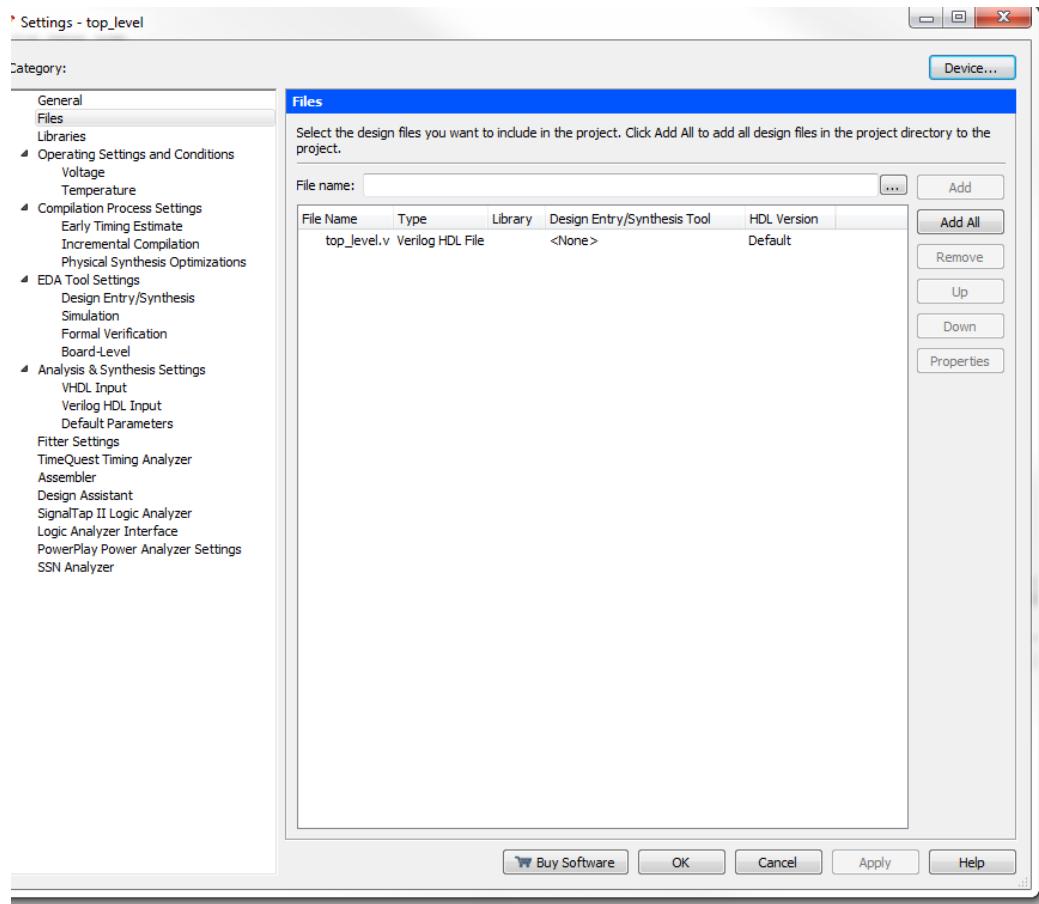
2.4 Timing Constraints

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.

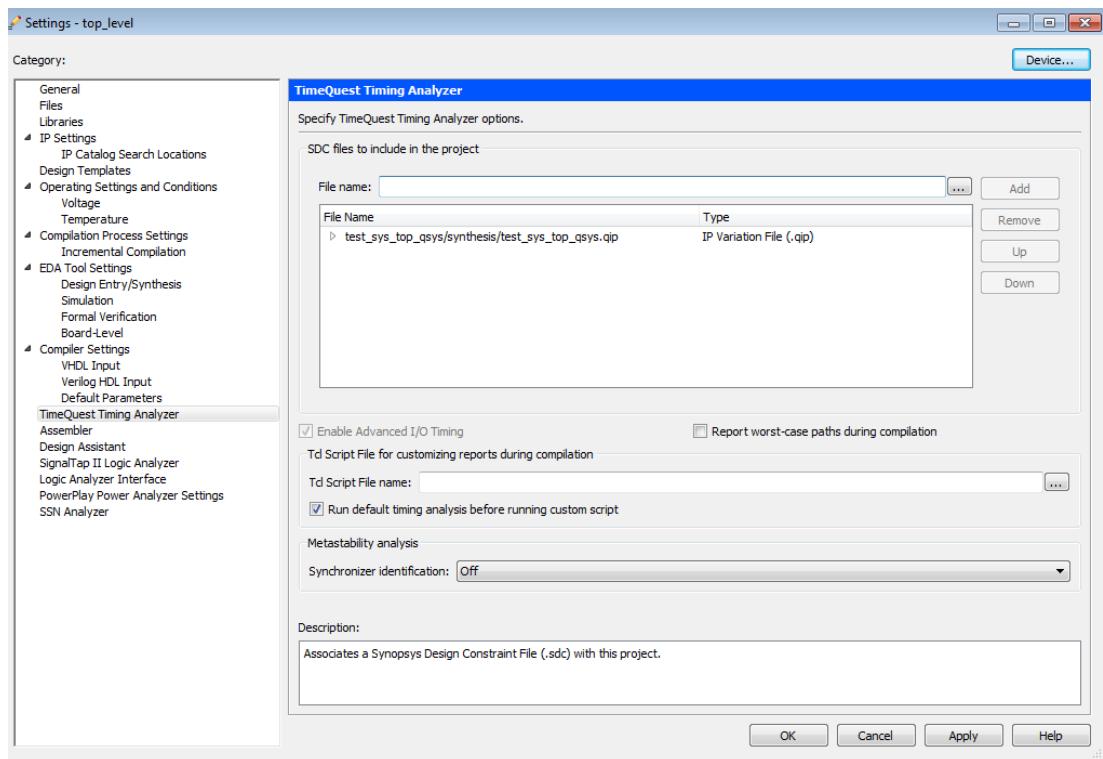
2.4.1 Adding Timing Constraints

2.4.1.1 Add/Remove SDC file

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

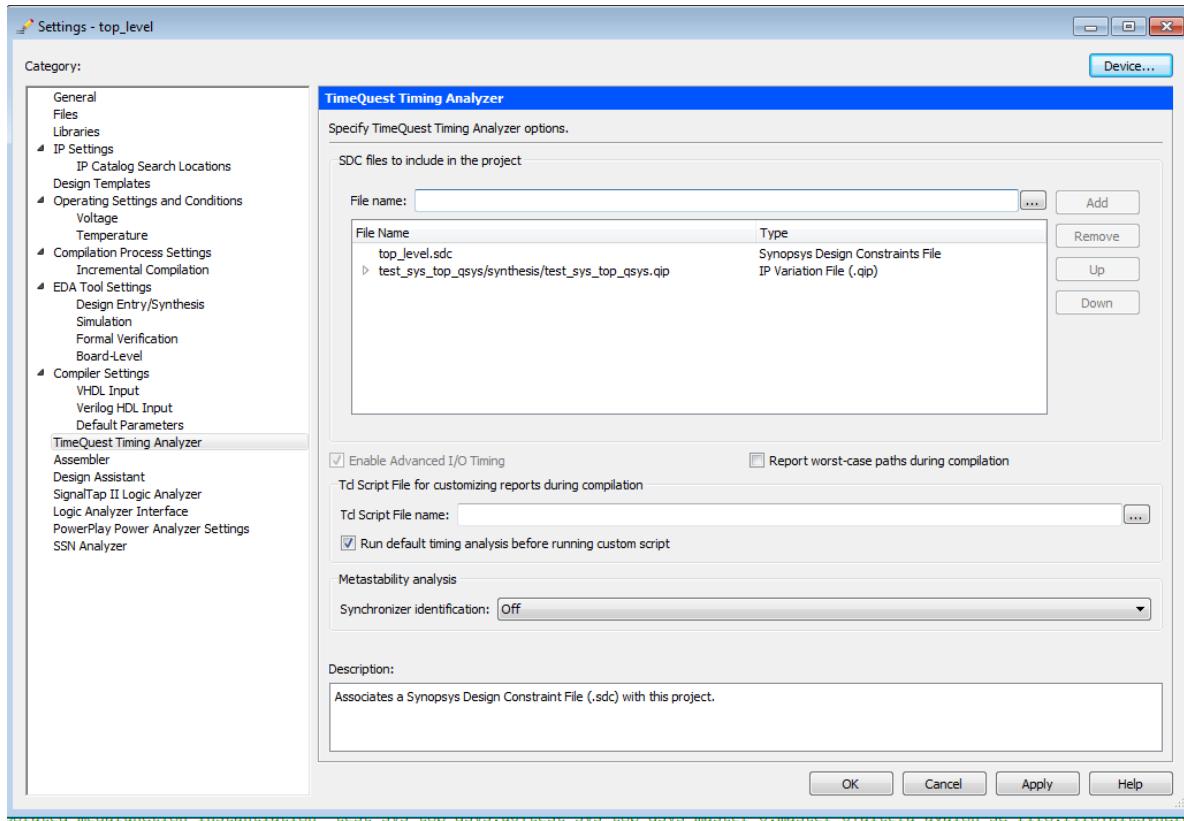


Select the TimeQuest Timing Analyzer on the left hand side of the window.



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

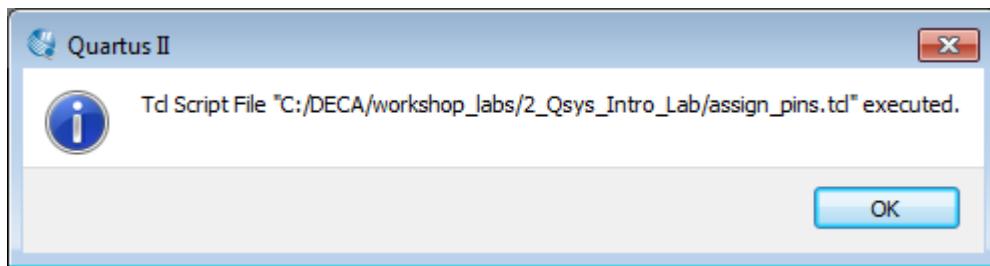
Before the design can be downloaded, pin assignments that match the hardware on the board are needed. There are different ways to do this, such as using Pin Planner, Assignment Editor, and other methods. In this lab a .tcl (script) file is used, because there are many pin locations as I/O assignments.

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

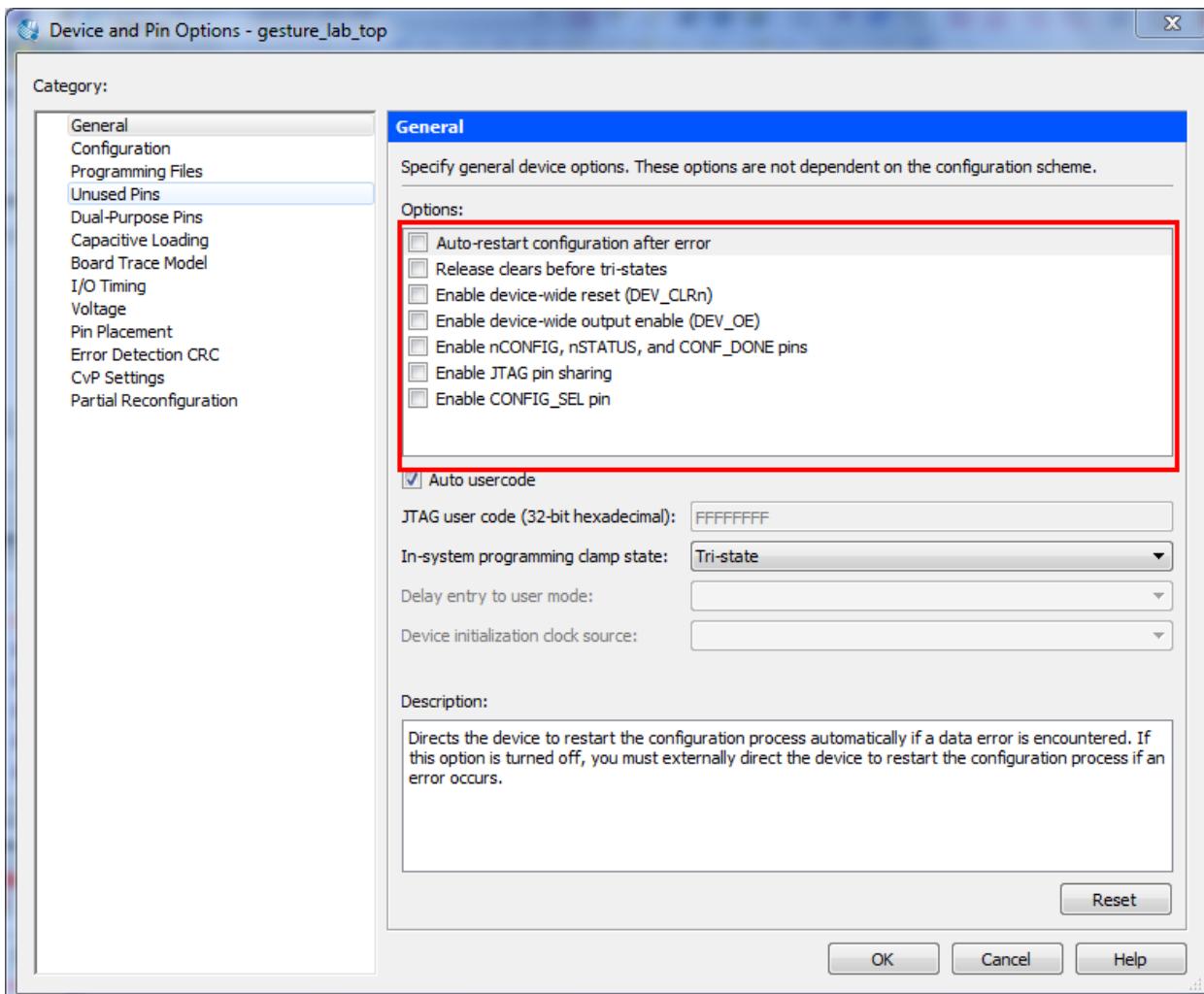
2.5.1.5 Click OK to close the TCL script window.

2.5.2 General Assignments

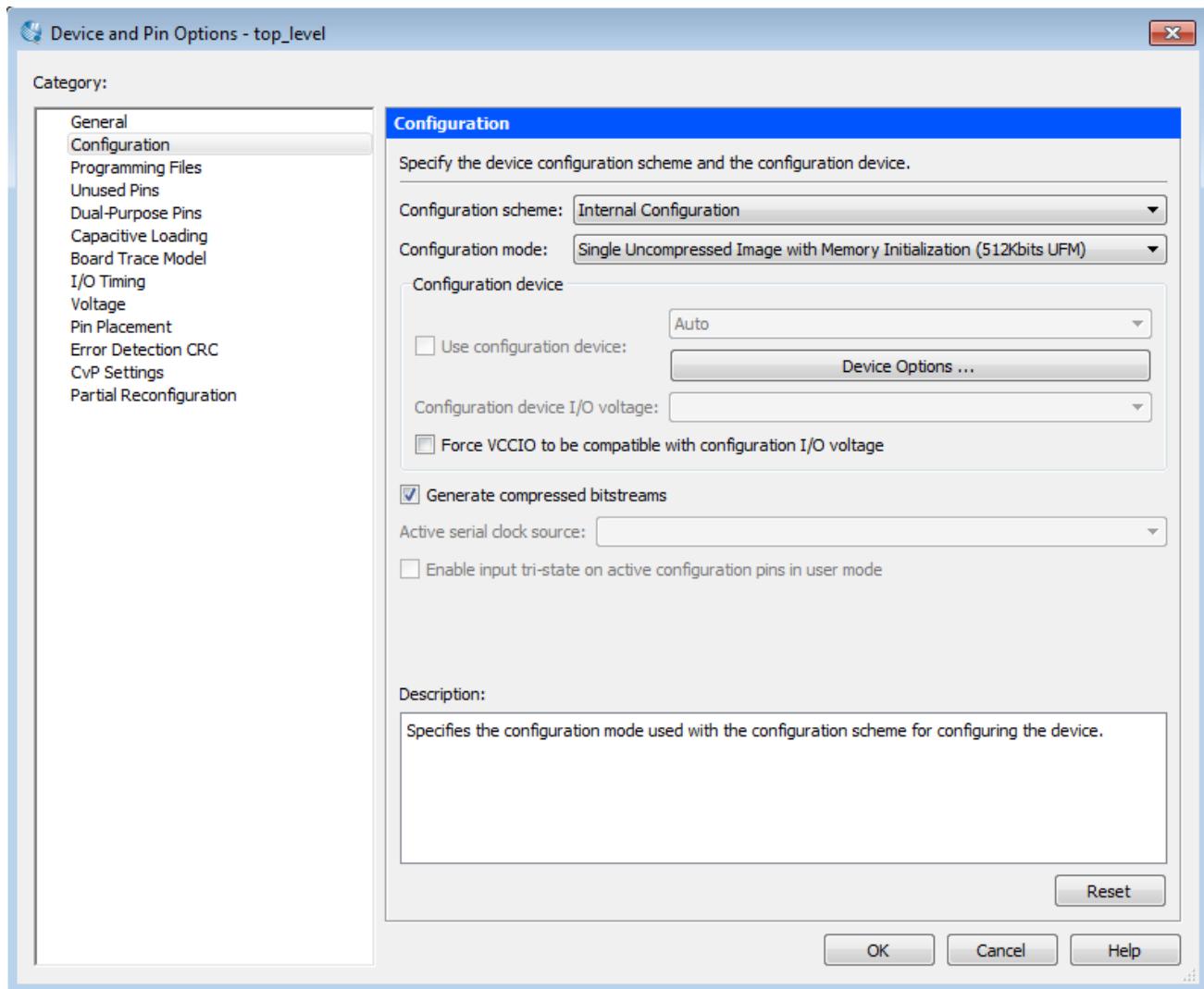
With the hardware design complete, a few device settings need to be changed and the project can be compiled to create a configuration file. Specifically, some optional configuration pins need to be disabled to match the DECA board hardware. If this step is not completed, the VCCIO required by those optional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



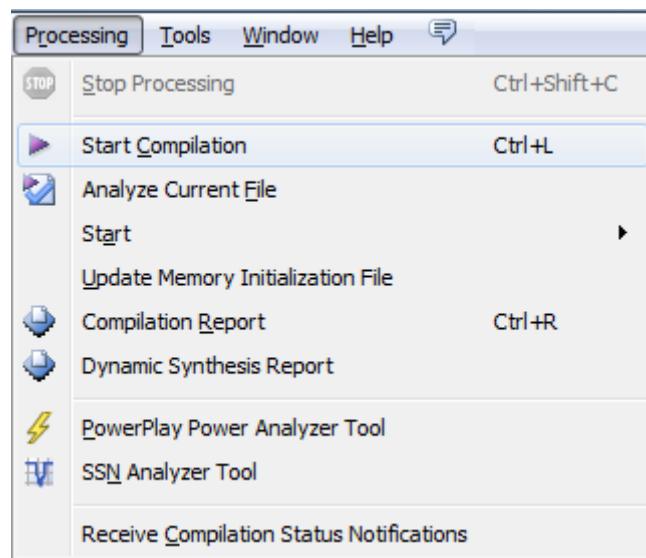
2.5.2.4 Click OK to close the Device and Pin Options window

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.



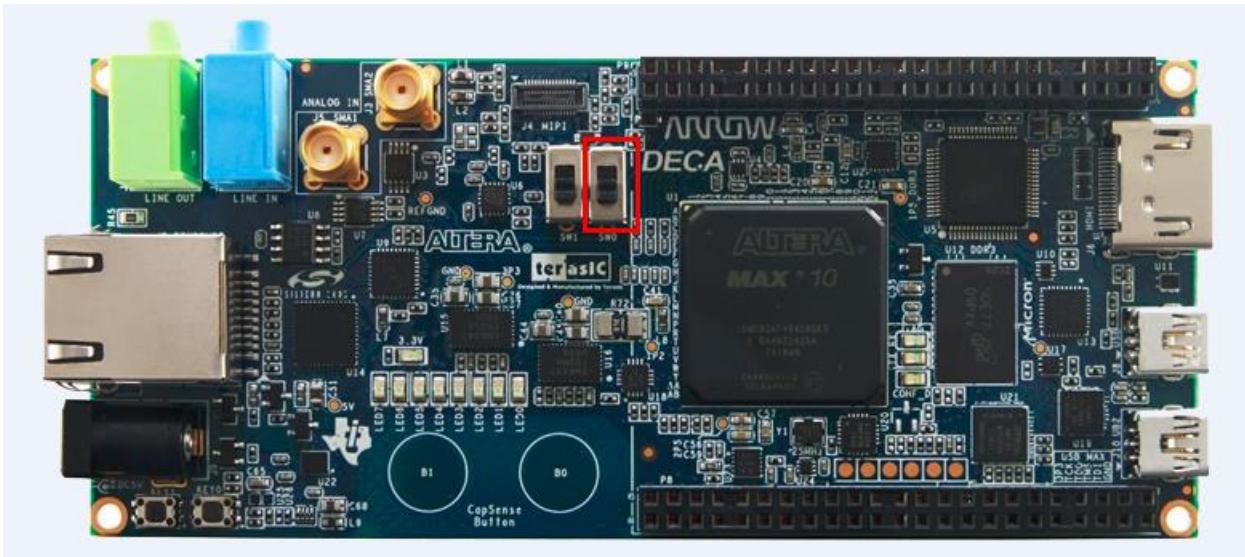
2.6.1.2 After a few minutes, the compilation should complete with no errors

Tasks		
Flow:	Compilation	Customize...
	Task	⌚
✓	▶ ▶ Compile Design	00:02:53
✓	▶ ▶ Analysis & Synthesis	00:01:09
✓	▶ ▶ Fitter (Place & Route)	00:01:03
✓	▶ ▶ Assembler (Generate programming files)	00:00:09
✓	▶ ▶ TimeQuest Timing Analysis	00:00:26
✓	▶ ▶ EDA Netlist Writer	00:00:06
	➡ Program Device (Open Programmer)	

2.6.2 Download the Configuration File to the DECA board

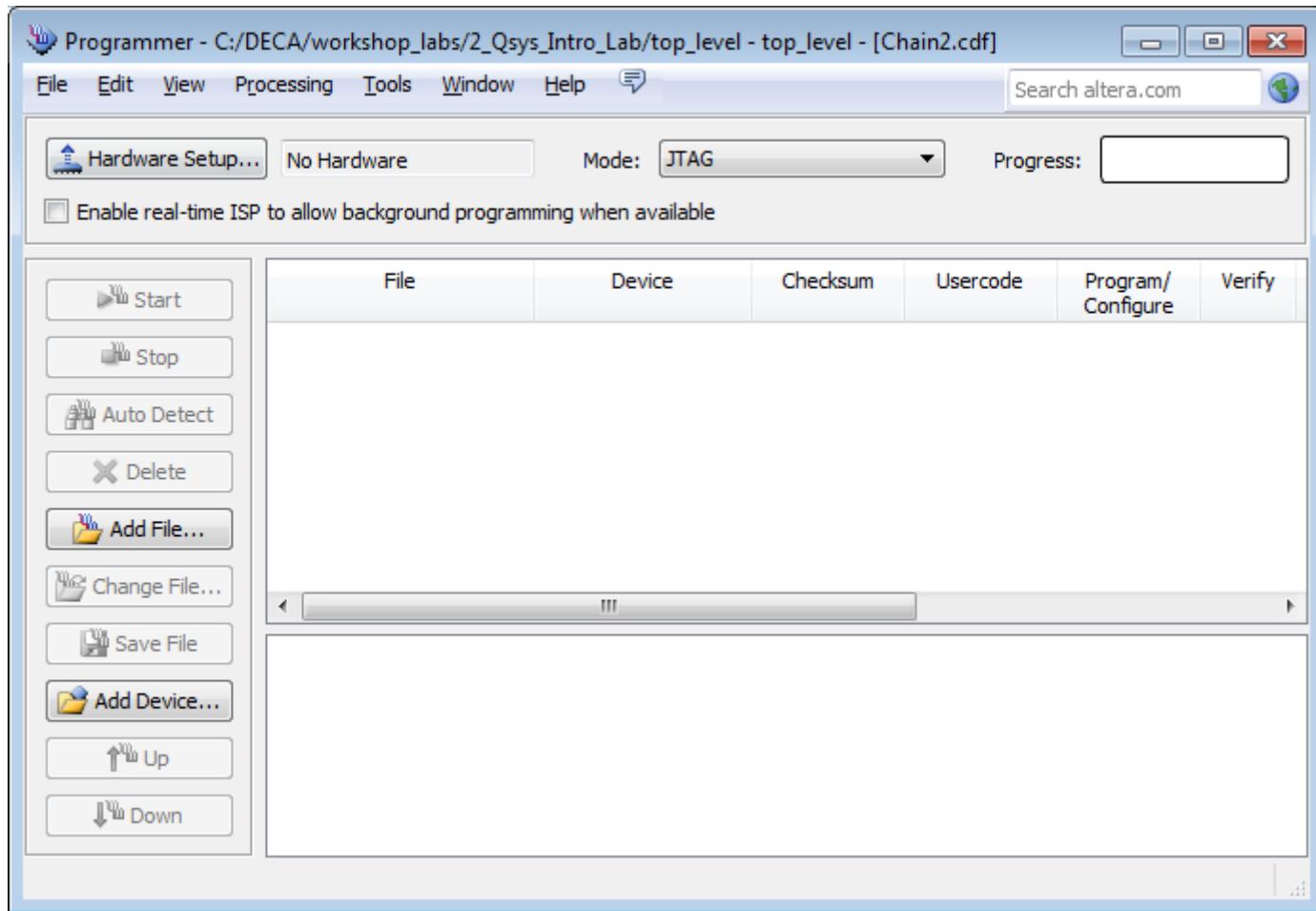
Now that the hardware design is complete and has been converted into a configuration file, the DECA needs to be programmed. Before doing so, we need to ensure the switch used to reset the logic is in the inactive state.

2.6.2.1 Set SW0 to low as shown here:

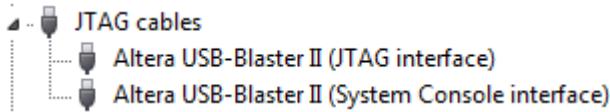


The logic within the FPGA can be reset by sliding the switch (SW0) up. To release reset, slide SW0 back down to the default position

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 **USB2 J10** (on the bottom right of the board). Since the USB Blaster II driver software should already be installed, the Windows's Device Manager should display two entries under "JTAG Cables".

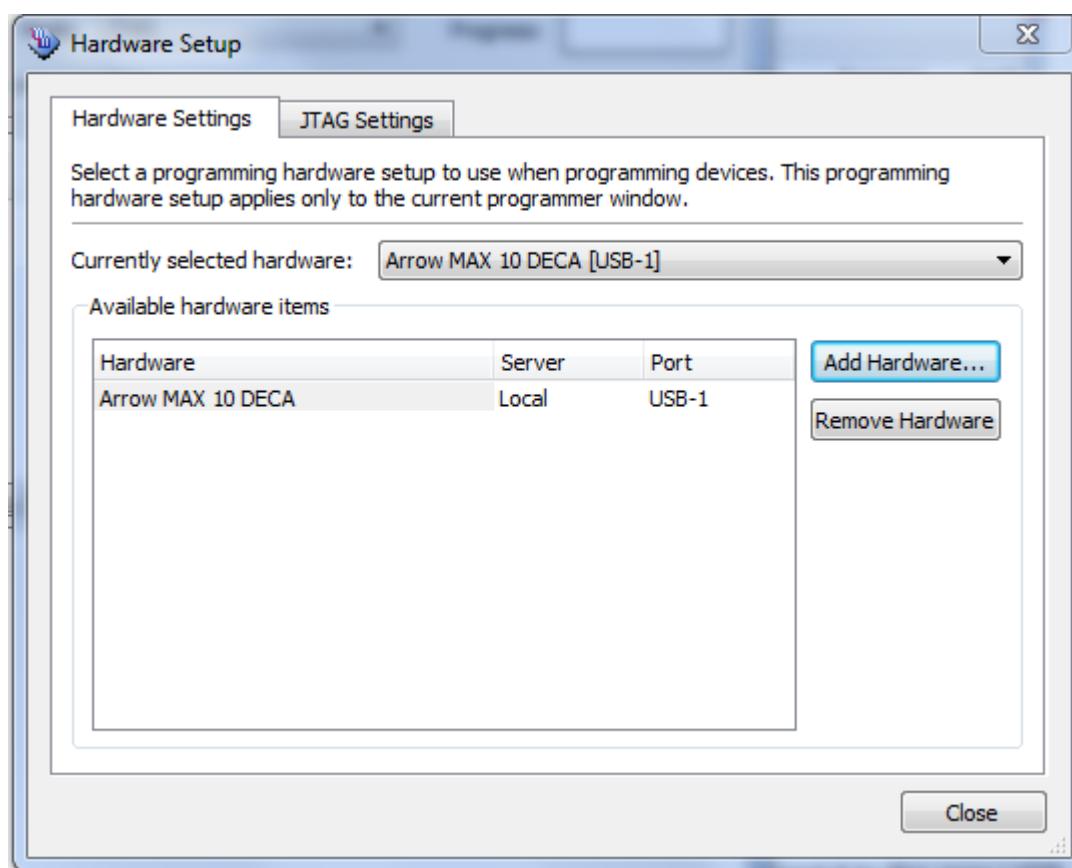


You should see a few LEDs light up on your DECA including the blue LED labeled **3.3V** and the green LED labeled **CONF_D**.

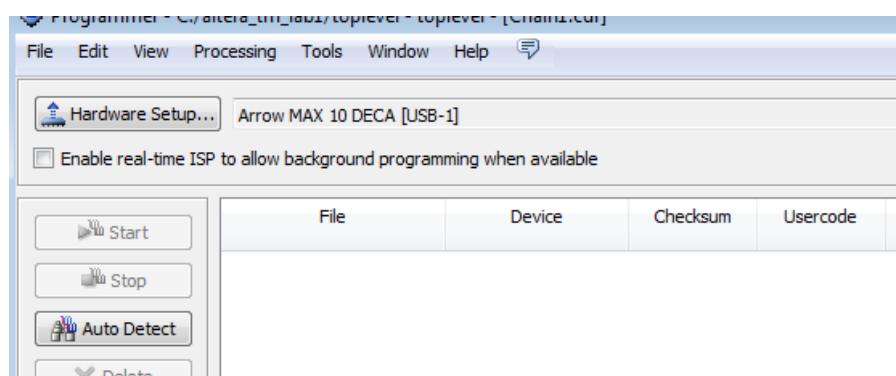


If the Device Manager shows an unconfigured USB Blaster, if Windows tries to look for drivers, or if the LEDs on the DECA do not light up, ask your workshop trainer for help.

- 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.

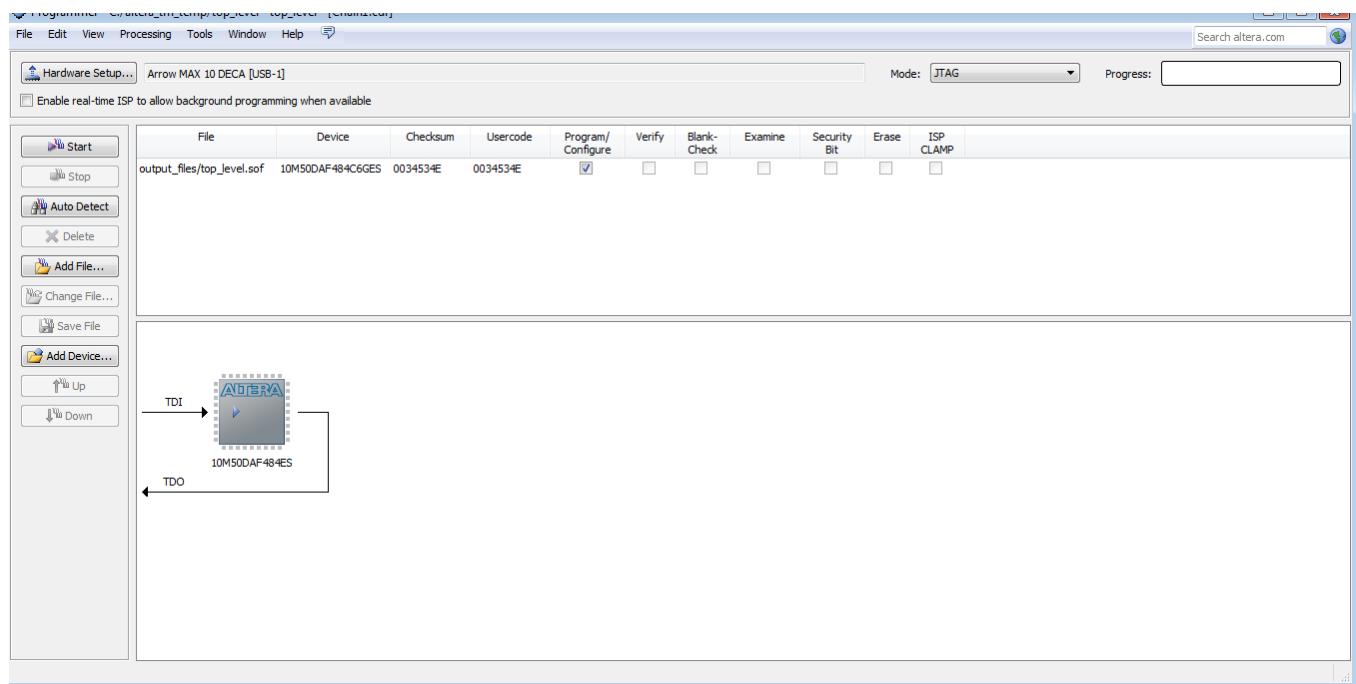


- 2.6.2.5 The programming window should now have a Hardware Setup such as:

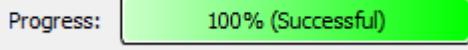


- 2.6.2.6 Click "Add File..." and navigate to <project_directory>/output_files/ in your compilation directory. Open the `top_level.sof` file.

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.

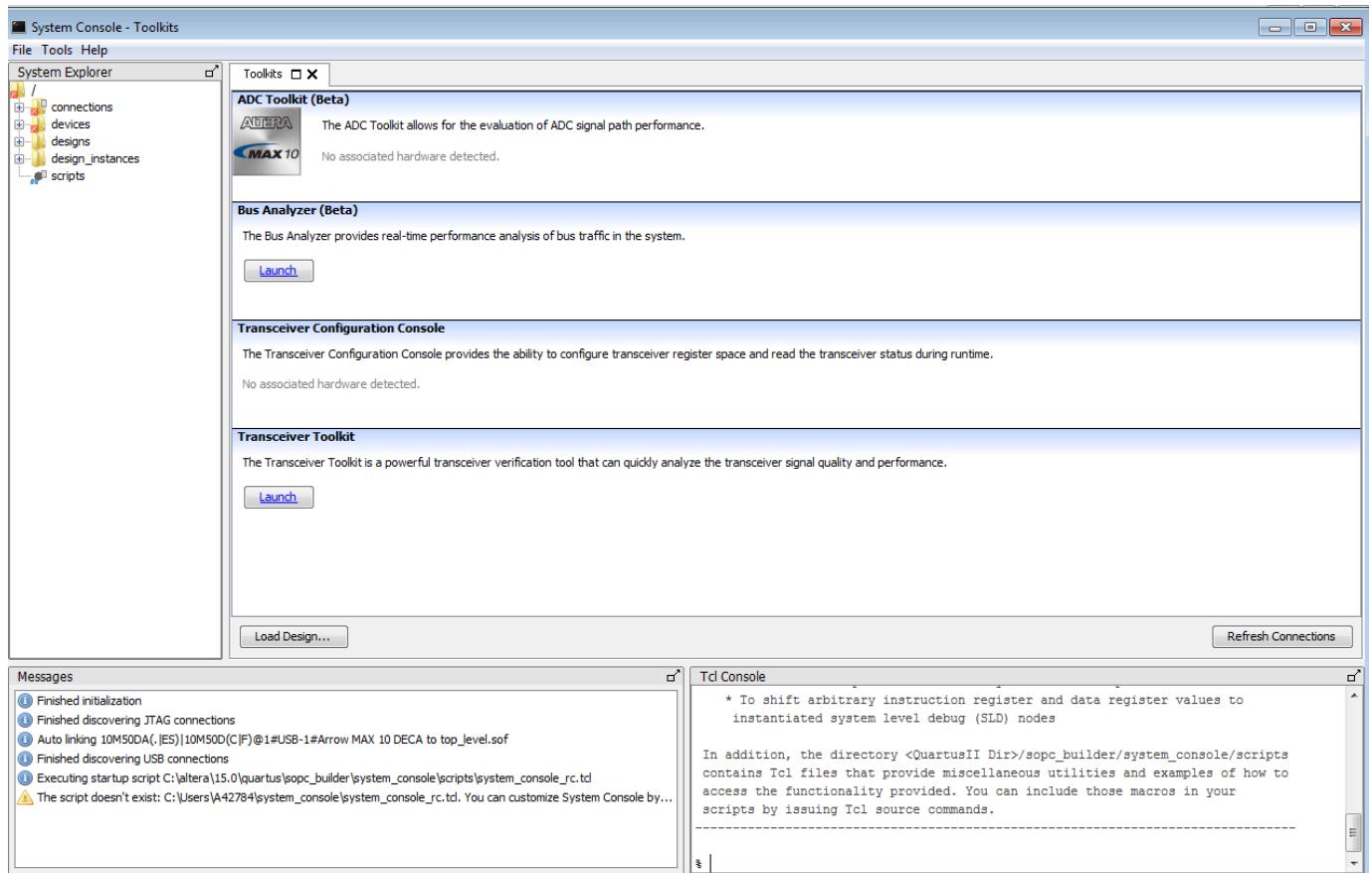
There are several ways to launch System Console. (either the **Tools → System Debug Tools → System Console...**, or via Qsys). For the lab, we will be using Qsys.

2.7.1.1 Launch Qsys (Tools → Qsys)

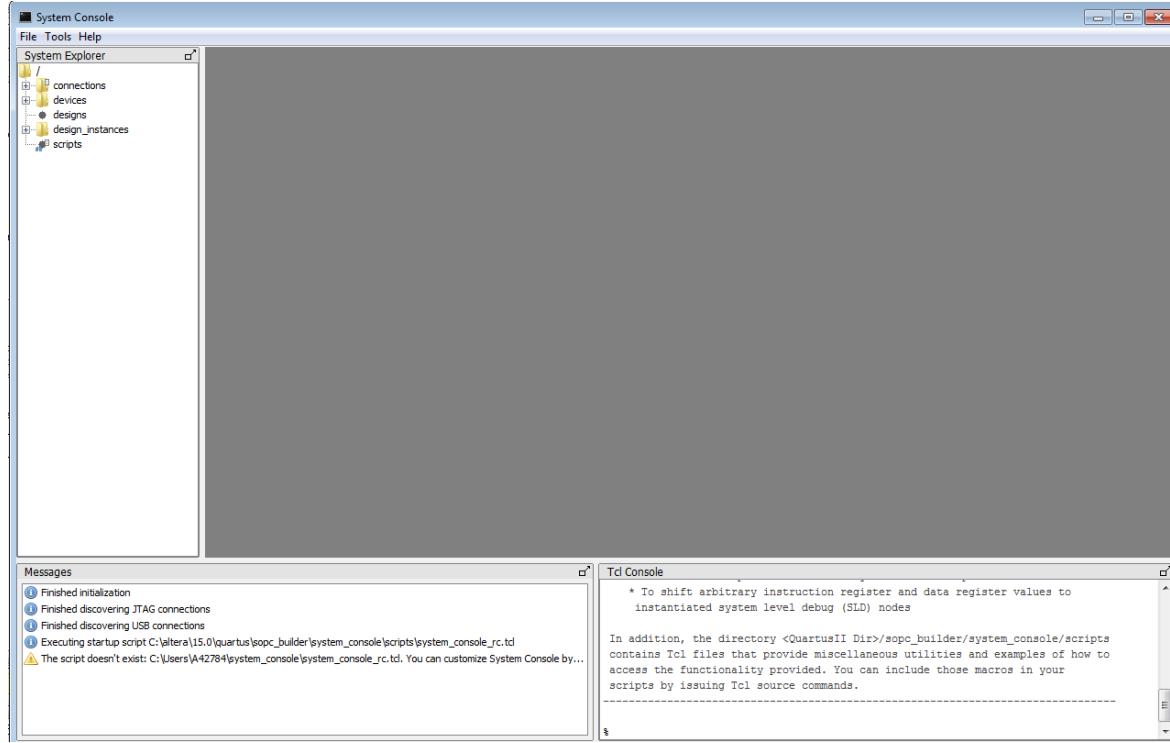
2.7.1.2 Open the `test_sys_top_qsys.qsys` file (you may need to browse to your top-level project folder)

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

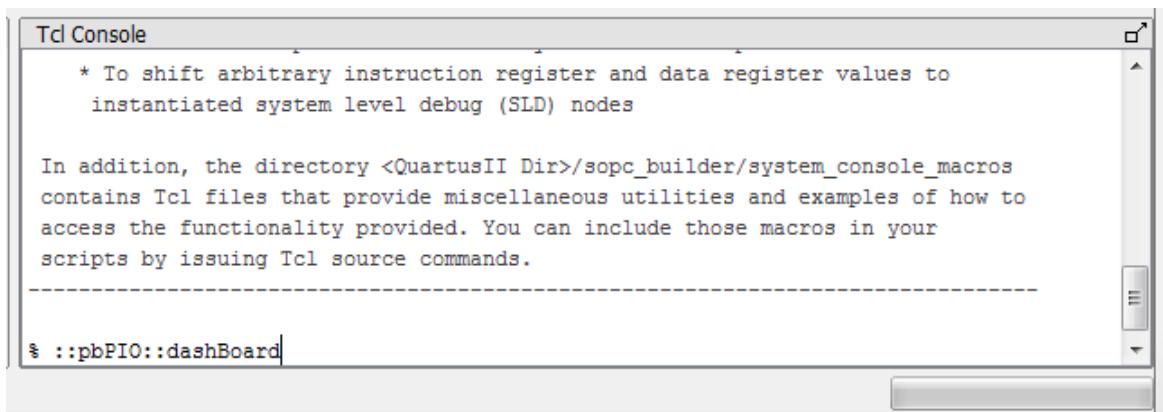
2.8.1 Push Button Dashboard

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!

Embedded Systems Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 3. EMBEDDED SYSTEMS LAB	81
3.1 Getting Started	81
3.1.1 Getting your DECA Kit	81
3.2 Examine the System Design	81
3.2.1 Examine the System Tool Flow	81
3.2.2 Examine the DECA Development Platform	82
3.3 Set up the Quartus II Project.....	83
3.3.1 Create a new Quartus II Project.....	83
3.4 Build the Hardware Design	87
3.4.1 Launch Qsys	87
3.4.2 Configure the Clock	88
3.4.3 Add an Avalon ALTPPLL for the processor and peripherals	89
3.4.4 Add a second Avalon ALTPPLL for the Internal ADC	93
3.4.5 Connect the incoming clock and reset to the PLL	96
3.4.6 Add a Nios II processor.....	98
3.4.7 Configure clock source for the Nios II processor	99
3.4.8 Add On-Chip Memory	100
3.4.9 Add an On-Chip Flash	103
3.4.10 Add an Internal MAX 10 ADC core	105
3.4.11 Add Avalon-MM Clock Crossing Bridge Peripheral for the “slow” peripherals.	106
3.4.12 Add the JTAG UART Peripheral	107
3.4.13 Add a Timer.....	109
3.4.14 Add a System ID Peripheral.....	111
3.4.15 Add PIO Peripheral for LEDs	112
3.4.16 Add PIO Peripheral for Pushbutton.....	113
3.4.17 Add I2C Peripherals for Capsense & the Relative Humidity and Temp Sensor HDC1000	115
3.4.18 Add a PIO module for the HDC1000 data ready signal.....	118
3.4.19 Add the PIO for the Capsense	119
3.4.20 Add the temperature sensor component	120
3.4.21 Resolve errors.....	120
3.4.22 Set Interrupt Priorities	121
3.4.23 Check the full system	122
3.4.24 Generate the Qsys System	124
3.4.25 Add the Qsys System to the Quartus Project	125
3.4.26 Compile the Quartus II project	126
3.4.27 Download the Configuration File to DECA.....	129
3.5 Create the Software Design	133
3.5.1 Start Nios II Software Build Tools for Eclipse	133
3.5.2 Create a New Software Project.....	135
3.5.3 Add Source Code to the Project	137

3.5.4	Configure the Board Support Package	138
3.5.5	Build the Software Project	140
3.5.6	Run the Application on the DECA board	141
3.5.7	Download the Executable to the DECA	141

LAB 3. EMBEDDED SYSTEMS LAB

Overview: This lab teaches you how to create an embedded system implemented in programmable logic. You will build a processor-based hardware system and run software on it. As the lab progresses, you will see how quick and easy it is to build entire systems using Altera's QSys tool to configure and integrate pre-verified IP blocks.

Lab Notes: Many of the names that the lab asks you to choose for files, components, and other objects in this exercise must be spelled exactly as directed. This nomenclature is necessary because the pre-written software application includes variables that use the names of the hardware peripherals. Naming the components differently can cause the software application to fail. There are also other similar dependencies within the project that require you to enter the correct names.

3.1 Getting Started

The first objective is to ensure that you have all of the necessary hardware items and software installed so that the lab can be completed successfully. Below is a list of items required to complete this lab:

- Arrow DECA Evaluation Kit (www.arrow.com/deca).
- USB cable
- Lab files
- Quartus II 15.0 Design Software
- Personal computer or laptop running Windows 7 with at least an Intel i3 core (or equivalent), 4 GB of RAM, and 12 GB of free hard disk space
- A desire to learn

3.1.1 Getting your DECA Kit

If you are attending a DECA Workshop, you should have received your DECA kit in the 3-in-1 evaluation kit bundle when you arrived at the workshop location. If you are working on this lab independently, a DECA kit can be purchased through your Arrow sales representative or at parts.arrow.com.



Make sure you have a USB cable to connect the on-board USB Blaster to your laptop. If you are attending the workshop, a USB cable should be included in your evaluation kit bundle.

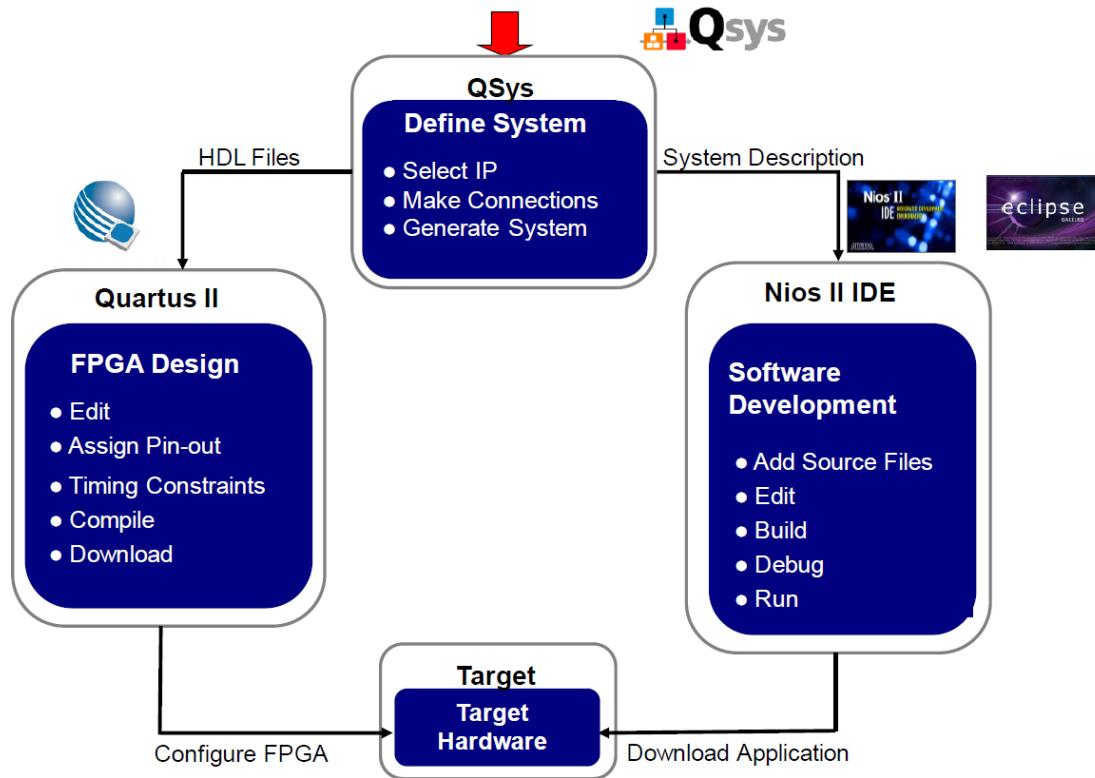
3.2 Examine the System Design

Overview: In this section, you will examine the design flow used in modern Altera FGPA designs.

3.2.1 Examine the System Tool Flow

Developing software for an Altera system on a programmable chip requires an understanding of the design flow between the Qsys system integration tool and the Nios II Embedded Development Suite (EDS). Typically, design

requirements begin with requirements and become inputs to system definitions. System definition is the first step in the design flow process. For this workshop, the design will be built and then the FPGA image will be downloaded into the dev board. The objective of the module is to review the development tools that will be used.



The above diagram shows the typical design flow for the system design. The system definition is done with Qsys. The Nios II IDE uses the system description to create a new project for the software application. The output of the FPGA design is a FPGA image that is used to configure the FPGA. The output of the software flow is an executable which runs on the Nios II processor.

3.2.2 Examine the DECA Development Platform

Examine the components on the DECA board. The development board provides a full system with the MAX10 at its heart including external memory, LEDs, sensors, buttons, and power supplies.



There are many components on the DECA board that can be used including the LEDs, capacitive push buttons, HDMI port, a MIPI interface and a full suite of sensors.

The completed system will include many components including the Nios II soft processor, JTAG UART, on-chip memory, PLLs, and an I2C interface.

The system that will be created in Qsys will use a library of re-usable IP blocks. Interconnect between components is automatically connected by Qsys. The system interconnect manages the dynamics bus-width matching, interrupt priorities, arbitration and address mapping. The processor that is used, Nios II, is a full featured processor that can run operating systems such as Linux.

The following modules will guide you through the process of building a basic embedded system.

3.3 Set up the Quartus II Project

In this module, you will create a Quartus II project for your embedded system design and create the software project to run on the Nios II processor.

3.3.1 Create a new Quartus II Project

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

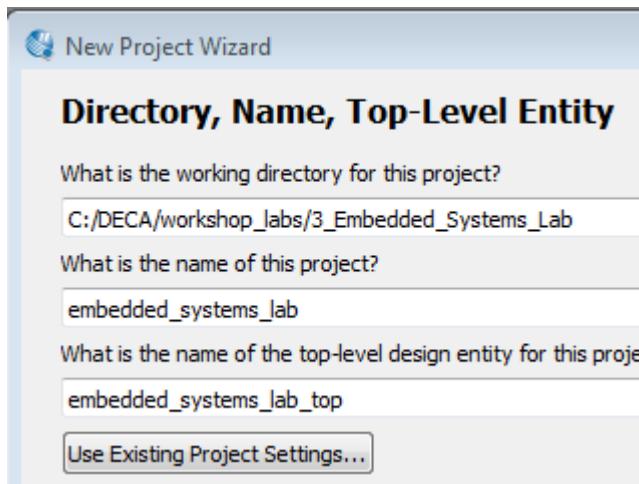
3.3.1.2 Configure the New Project Wizard directory, name, and top-level entity information.

Click on the  button and browse to the embedded systems lab folder (for example
c:\DECA\workshop_labs\3_EMBEDDED_Systems_Lab)

Specify the name of the project: **embedded_systems_lab**

Specify the name of the top level entity: **embedded_systems_lab_top**

(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.)



Click 

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

3.3.1.4 Add source files to the project

Click on the  button and browse into the **source** directory where you will locate the two provided design files: **embedded_systems_lab_top.v**, **embedded_systems_lab.sdc**, and select both files and add them to the files listing. Note: to see the sdc file, change the file type filter to "All Files (*.*)".

 Don't forget to click the Add button to add the files to the project.

New Project Wizard

Add Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name: ...

	Type	Library	Design Entry/Synthesis Tool	HDL Version
/embedded_systems_lab_top.v	Verilog HDL File			Default
/embedded_systems_lab.sdc	Synopsys Design Constraints File			

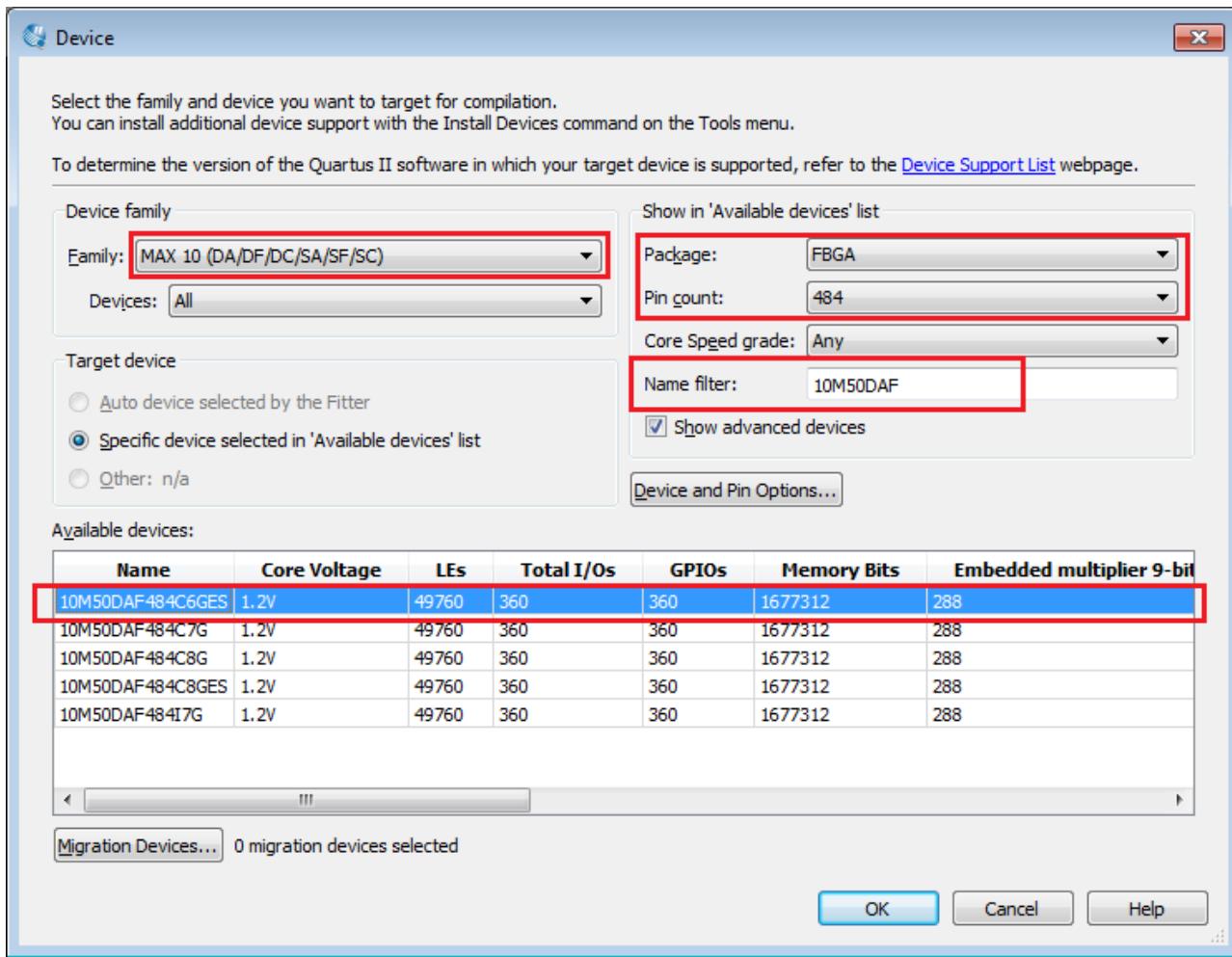
Add Add All
Remove Up
Down Properties

Click Next >

3.3.1.5 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**.



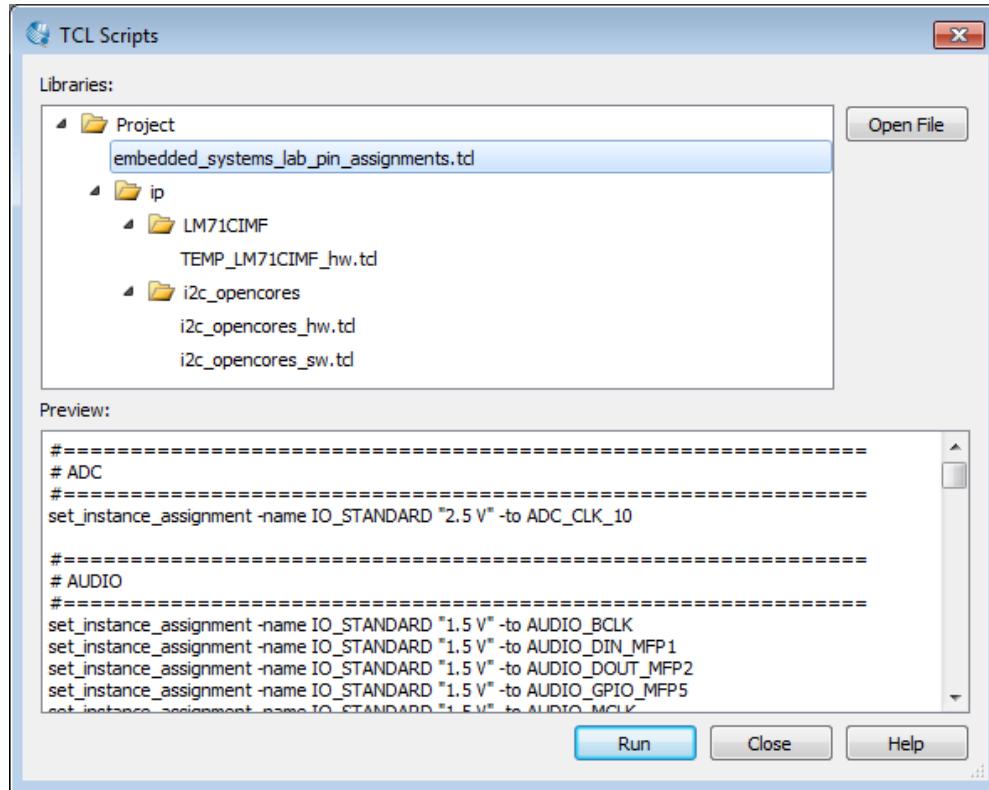
After making your selection, look at your kit and confirm that the part number marked on your device matches your selection. Click **Finish**

3.3.1.6 Execute Setup Script

For your convenience in this lab, the I/O pin constraints have been programmed into a Tcl script to set up the Quartus II project properly. Under the menu, select: **Tools → Tcl Scripts**

3.3.1.7 In the Tcl Scripts dialog box choose the `embedded_systems_lab_pin_assignments.tcl` script.

3.3.1.8 Click **Run**. (Make sure to click "Run" prior to clicking "Close".)



3.4 Build the Hardware Design

Overview: In this module, you will use the Qsys system integration tool to design your hardware system. You will add standard and custom components, make interface connections, assign clocks, set arbitration levels for interrupts, and generate the HDL for the system.

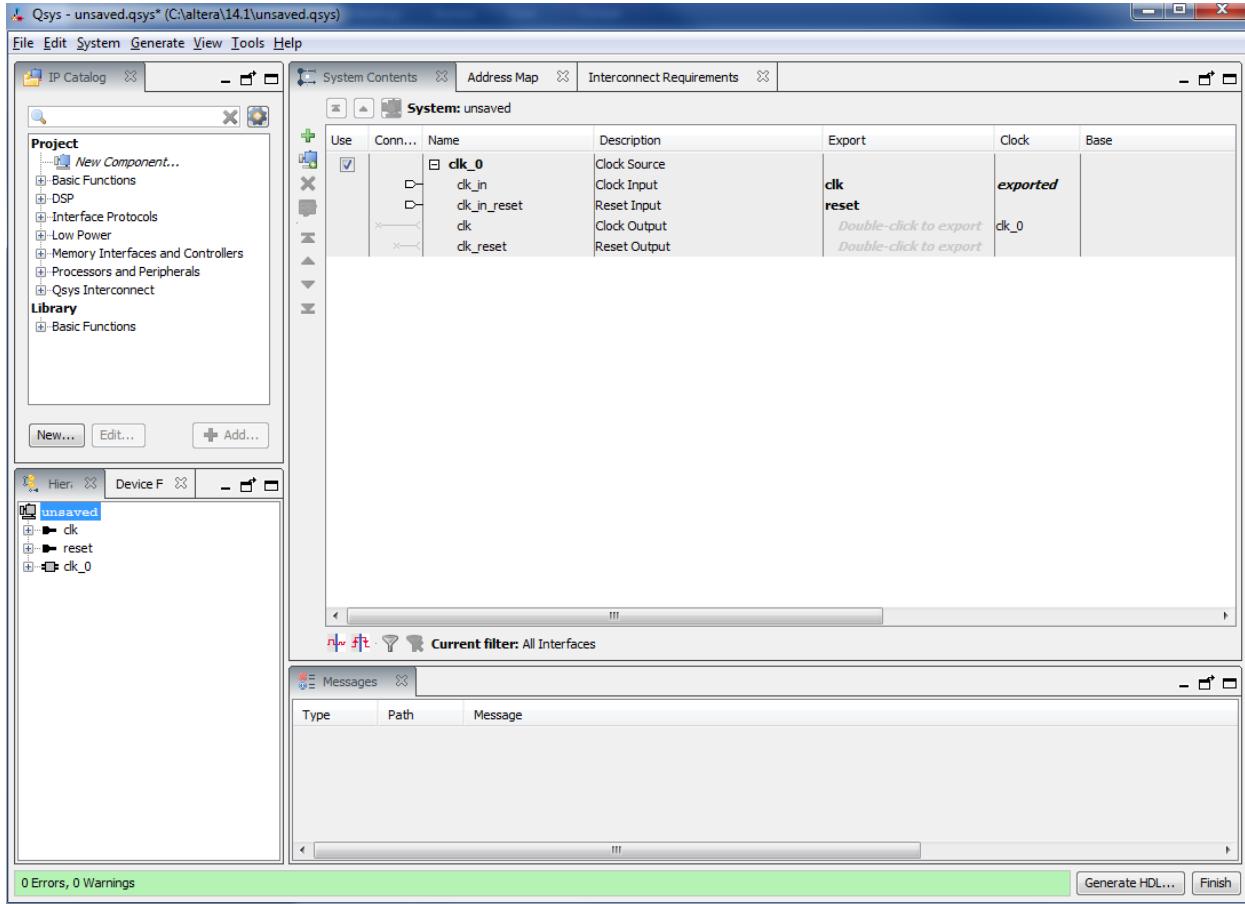
3.4.1 Launch Qsys

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...).

3.4.1.1 Open Qsys: From the Quartus II window: **Tools → Qsys**.

3.4.1.2 In the new Qsys window, you should see a single Clock Source component named **clk_0** in the System Contents tab. This tab shows all of the components currently in your system.



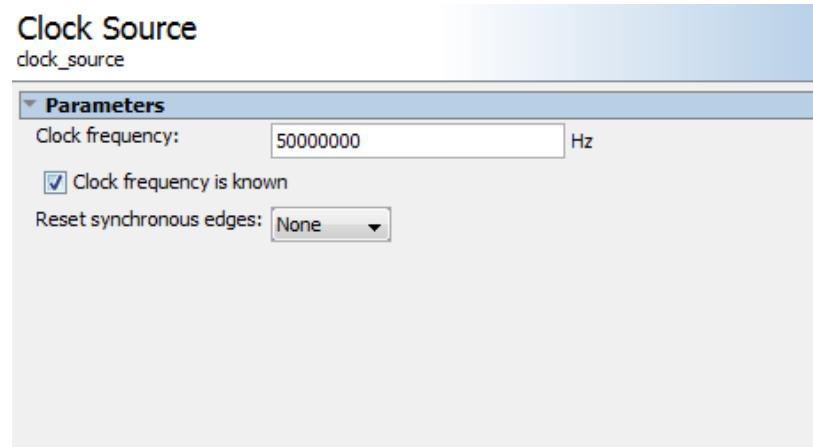
3.4.2 Configure the Clock

In this section, you will configure the clock input to your Qsys system. This clock will be fed to a PLL to provide additional frequencies.

3.4.2.1 Double click on the Clock Source component named clk_0. This will open the Parameter editor window which should look very familiar to the traditional megawizard windows.

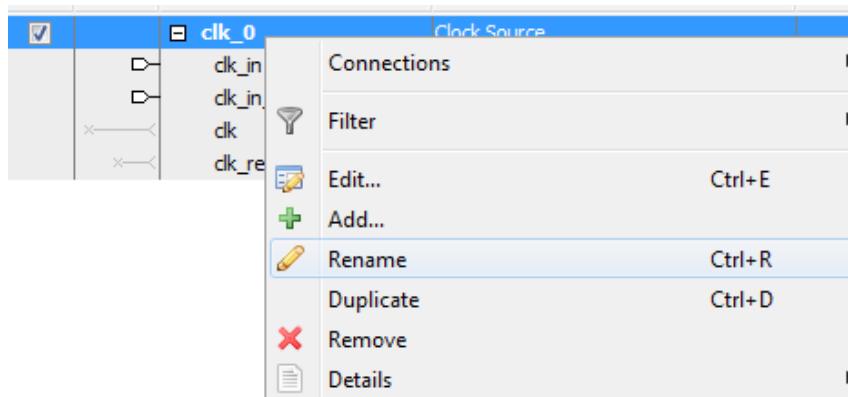
3.4.2.2 Change the clock frequency parameter to 50 MHz (50000000 Hz).

Ensure that the "Clock frequency is known" parameter is enabled.



Click the "X" on the Parameter tab to close the parameter window.

3.4.2.3 To rename the clock, right-click on the clock and select "Rename" or press **CTRL+R**. Rename the clock to **clk_50** and press Enter.

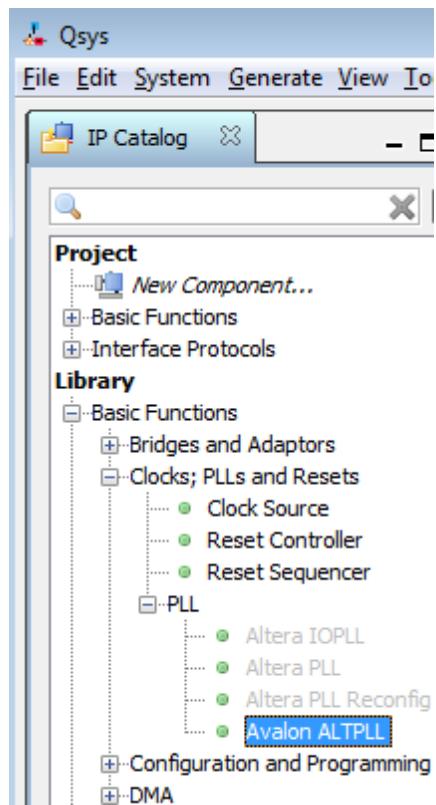


3.4.2.4 Save the Qsys system. Click **File → Save As** and name your Qsys system **deca_top.qsys**. This is the entity name by which you will be instantiating your Qsys system into your top-level file. Click Save.

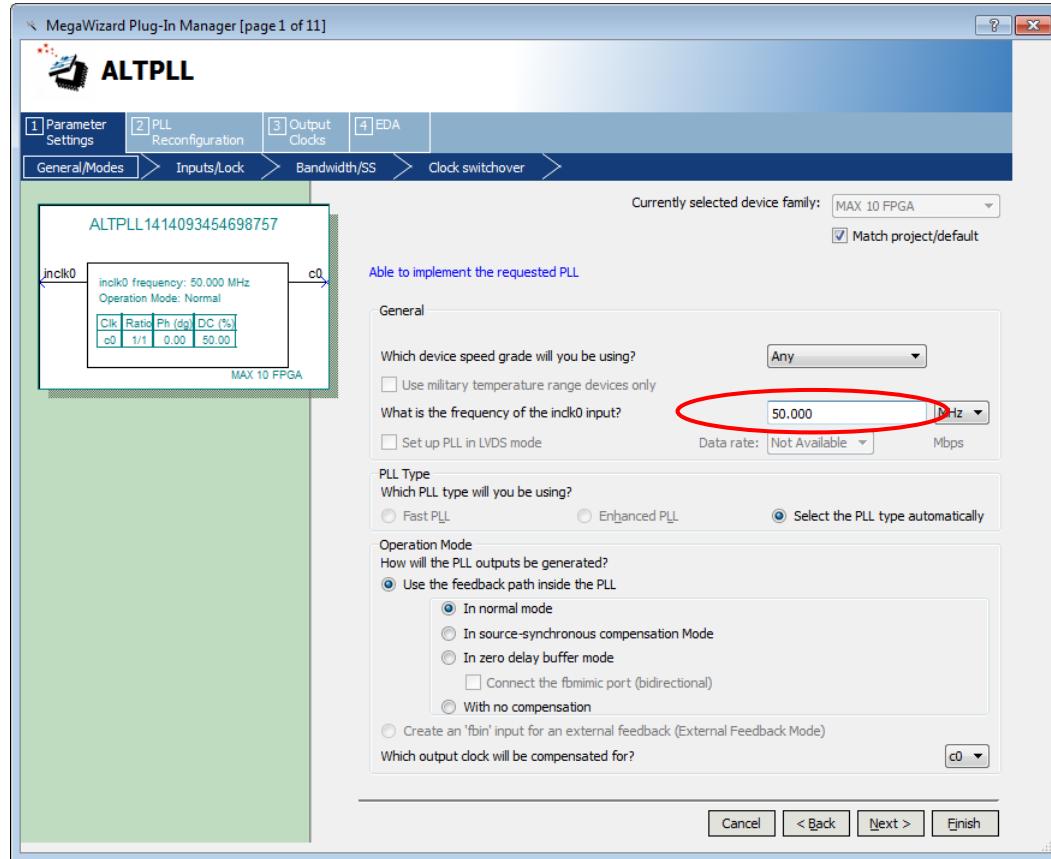
3.4.3 Add an Avalon ALTPPLL for the processor and peripherals

The Avalon ALTPPLL peripheral instantiates a PLL that will generate the clocks for the system.

3.4.3.1 From the IP Catalog pane, expand “Basic Functions,” then expand “Clocks; PLLs and Resets,” then expand “PLL” and double click on “Avalon ALTPLL.”



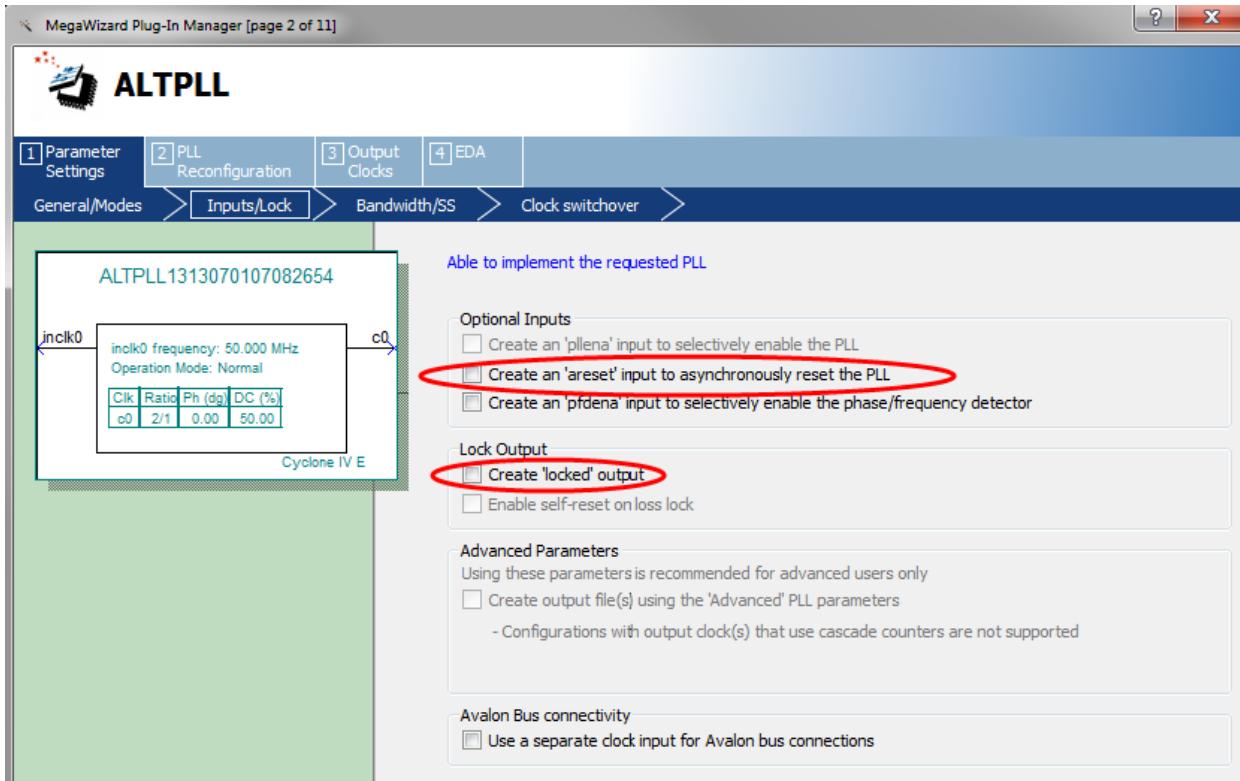
3.4.3.2 “General/Modes” tab (Page 1) of PLL MegaWizard. Change the frequency of the clock input to 50 MHz. This source is provided by the oscillator on the DECA board.



Click Next to move to the next tab of the wizard. (You may need to scroll down to see the Next button.)

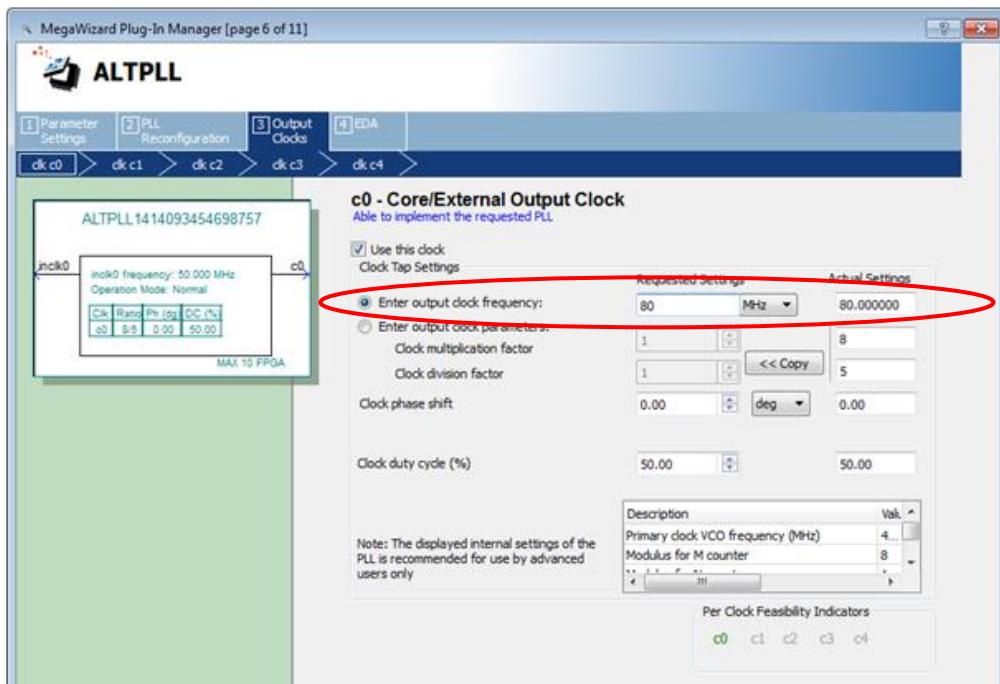
3.4.3.3 “Inputs/Lock” tab (Page 2): Uncheck both “Create an ‘areset’ input to asynchronously reset the PLL” and “Create ‘locked’ output” options.

Accept all other defaults.

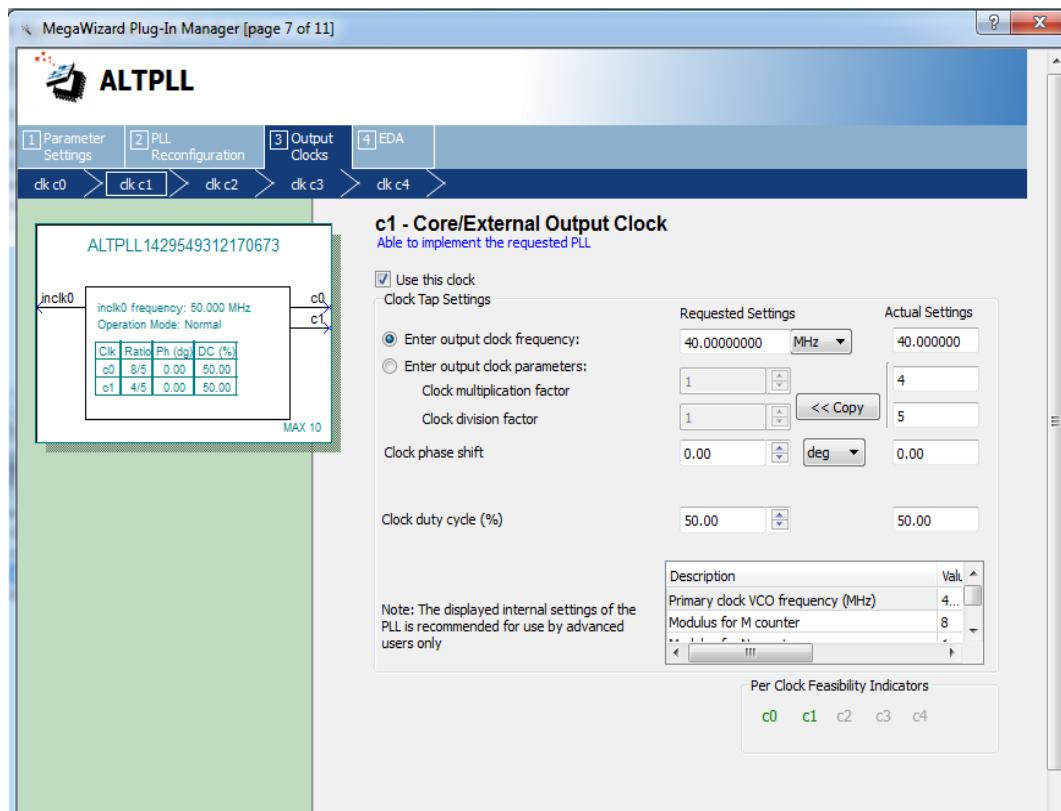


3.4.3.4 Pages 3-5: Accept all defaults and click next until you reach the Output Clocks tab.

3.4.3.5 On “c0 Core/External Output” (Page 6): Click “Enter output clock frequency”. Configure c0 as 80 MHz output. Click on the “Enter output clock frequency” button and enter 80 MHz. This clock will be used as the processor system clock, clocking the Nios II processor. Click Next.



3.4.3.6 “c1 Core/External Output” (Page 7): Click “Enter output clock frequency”. Configure c1 as 40 MHz output. Check the “Use this clock” button. Click on the “Enter output clock frequency” button and enter 40 MHz. This clock will be used to clock various peripherals in the system.



3.4.3.7 Click Finish. This will take you to the summary tab.

Click Finish again to close the Avalon ALTPLL MegaWizard.

3.4.3.8 A component entitled “altpll_0” should appear under Module Name. Rename the Avalon ALTPLL component from “altpll_0” to “**nios_p11**”. (You can right click to bring up a menu with a rename option.)

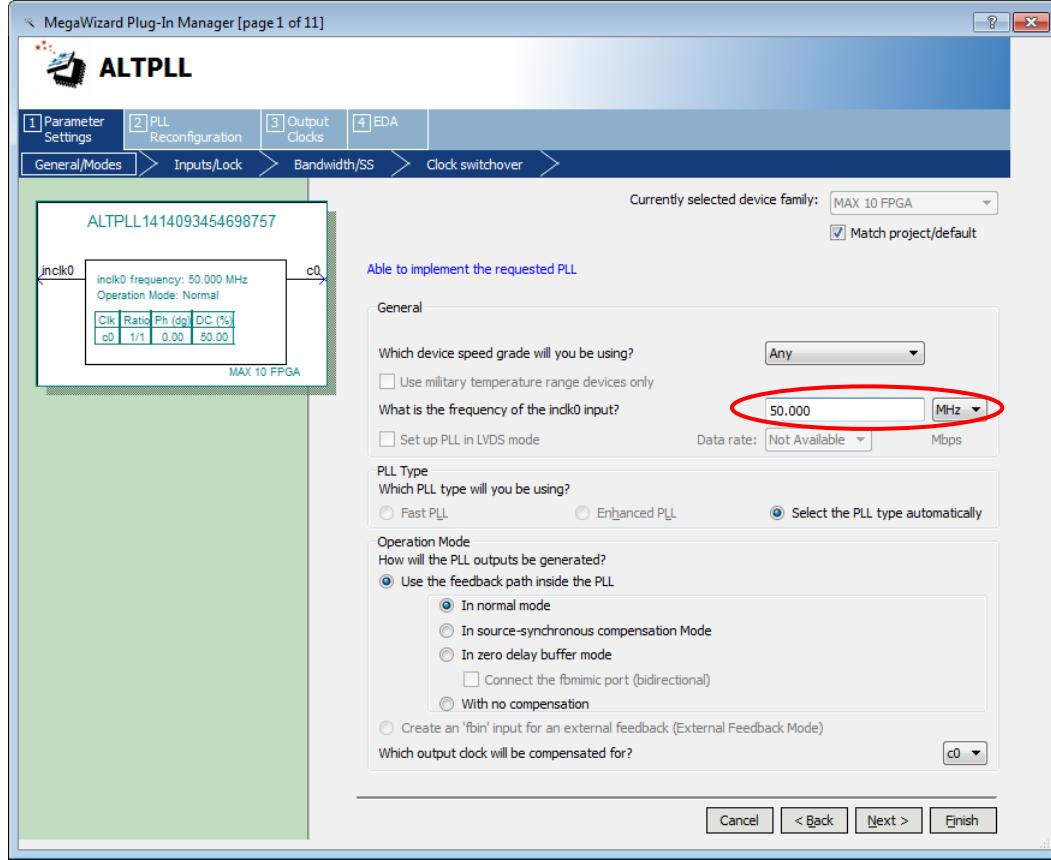
Some errors and warnings will appear in the bottom console indicating that various ports are not connected. Ignore these for now. We will address these connections in the upcoming steps.

3.4.4 Add a second Avalon ALTPLL for the Internal ADC

The internal ADC has specific requirements for its clock source, so we will use another PLL to generate a 10 MHz clock for the ADC.

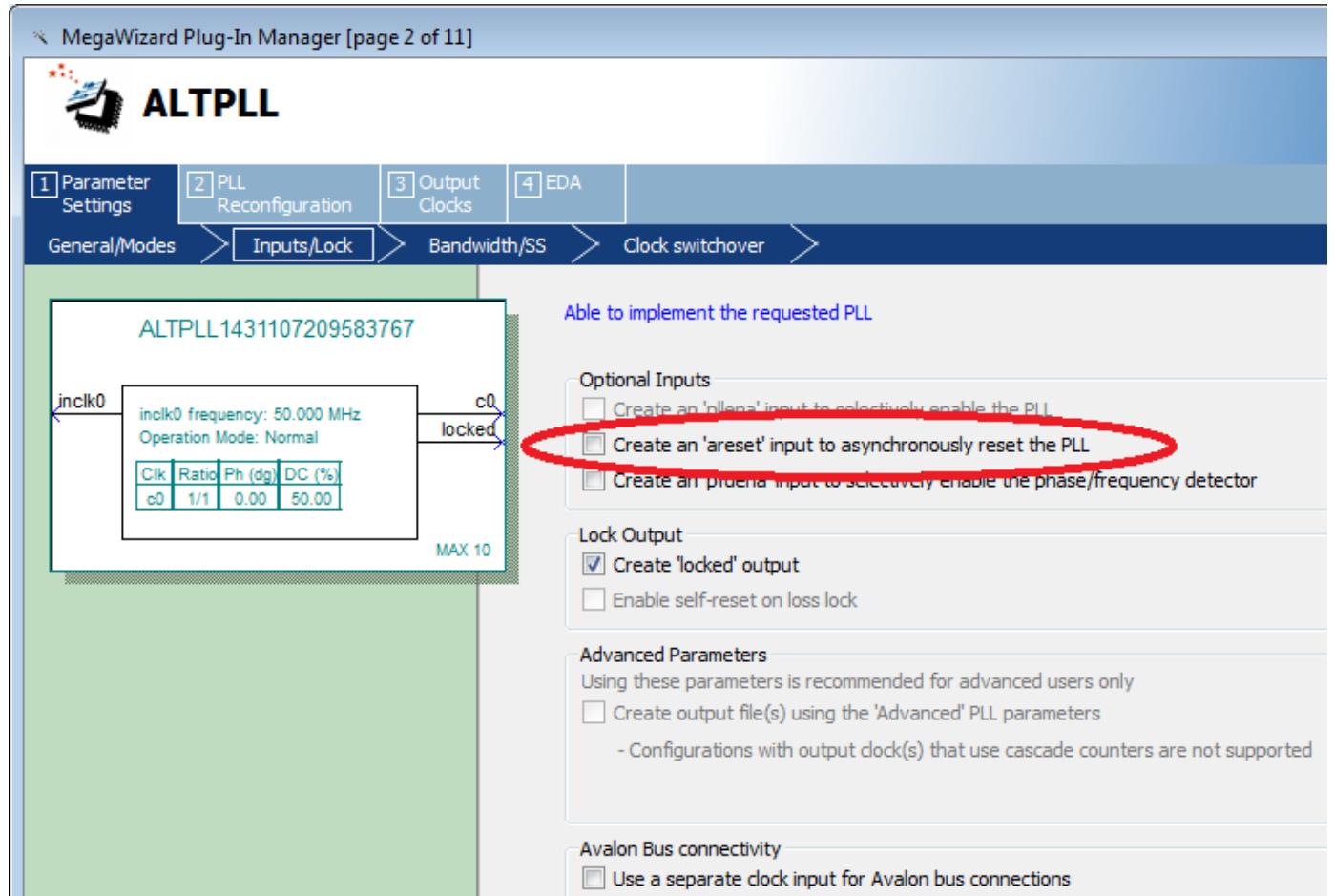
3.4.4.1 Once again, from the IP Catalog pane, expand “Basic Functions,” then expand “Clocks; PLLs and Resets,” then expand “PLL” and double click on “Avalon ALTPLL.”

3.4.4.2 “General/Modes” tab (Page 1) of PLL MegaWizard: Change the frequency of the clock input to 50 MHz. This source is provided by the oscillator on the DECA board.



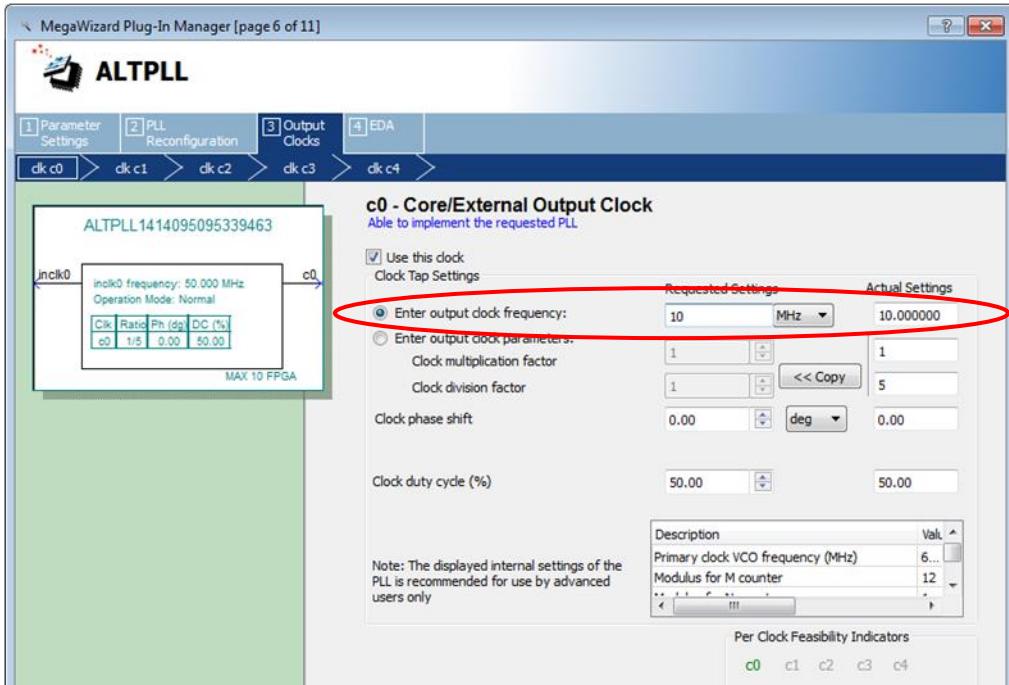
Click Next to move to the next tab of the wizard. (You may need to scroll down to see the Next button.)

- 3.4.4.3 “Inputs/Lock” tab (Page 2): Uncheck just “Create an ‘areset’ input to asynchronously reset the PLL”. Accept all other defaults.



Pages 3-5: Accept all defaults and click next until you reach the Output Clocks tab.

- 3.4.4.4 On “c0 Core/External Output” (Page 6): Click “Enter output clock frequency”. Configure c0 as 10 MHz output. Click on the “Enter output clock frequency” button and enter 10 MHz. This clock will be used as the ADC reference clock driving the ADC. Click Next.



Click Finish. This will take you to the summary tab.

Click Finish again to close the Avalon ALTPPLL MegaWizard.

3.4.4.5 A component entitled “altpll_0” should appear under Module Name. Rename the Avalon ALTPPLL component from “altpll_0” to “adc_pll”. (You can right click to bring up a menu with a rename option.)

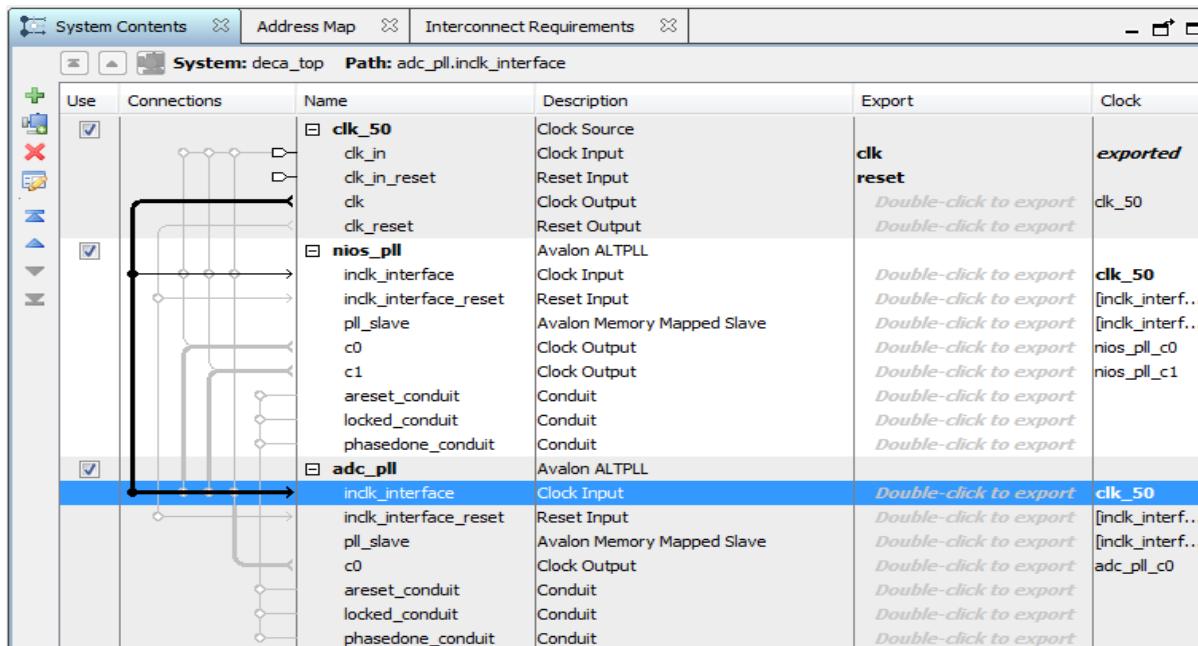
Some errors and warnings will appear in the bottom console indicating that various ports are not connected. Ignore these for now. We will address these connections in the upcoming steps.

3.4.5 Connect the incoming clock and reset to the PLL

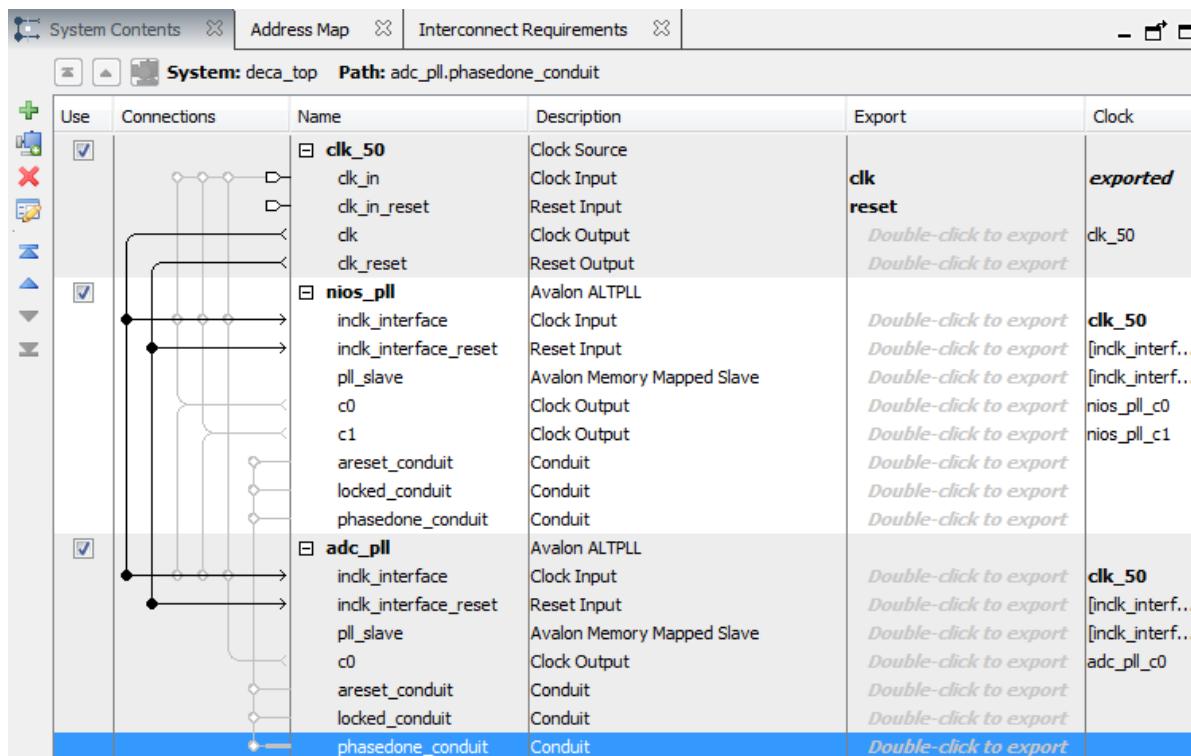
Qsys needs to know what clock and reset sources to use as the input to the PLL component. The clock and reset sources can come from an external source or from another component within the Qsys system. In our case, we will be connecting them to an external clock and reset.

Click on the "System Contents" tab to return to the view of the components in our system. At this point, there are three components, a "Clock Source" component that was in the system by default when Qsys first launched and the "Avalon ALTPPLL" component that we added in the first step. The Clock Source component is a Qsys component which brings in a clock and reset source from outside of the Qsys system. We will connect its nodes to the corresponding nodes on the Avalon PLL component.

In the "Connections" column, hover over the connections and you will then be able to fill in connection dots to make connections.



- 3.4.5.1 Connect the clk clock output port of the Clock Source to the inclk_interface of the nios_pll component and the inclk_interface of the adc_pll component. Similarly connect the clk_reset reset output port of the Clock Source to the inclk_interface_reset of the Avalon ALTPLL components. Your resulting connections should look as follows:

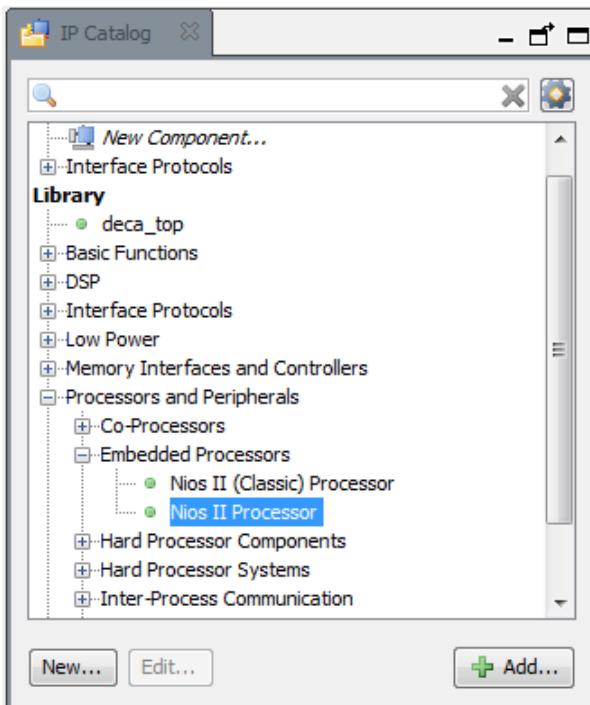


Click on **File → Save** and save your work periodically as you continue through the design.

3.4.6 Add a Nios II processor

A CPU is needed to run the software applications.

- 3.4.6.1 From the IP Catalog panel on the left side of the Qsys window, expand the menus for Processors and Peripherals → **Embedded Processors** and select the Nios II Processor.



Note that MAX10 devices do not support the Nios II (Classic) Processor. However, all code developed on the classic version is fully forward compatible.

- 3.4.6.2 Double click on the name or click "Add..." to add the component to the system. The Nios II parameter editor window will open.

- 3.4.6.3 In the Main tab, ensure that the "Nios II/e" option is selected.

- 3.4.6.4 The settings in the Vectors tab will be set in a later step so skip that for now.

Note that until these settings are applied, the following two errors in the Qsys window are expected.

Error: nios2_gen2_0: Reset slave is not specified. Please select the reset slave.

Error: nios2_gen2_0: Exception slave is not specified. Please select the exception slave

- 3.4.6.5 The settings in the other tabs are left as their defaults but feel free to explore the parameter editor and see what settings can be applied to the Nios II. Click Finish.

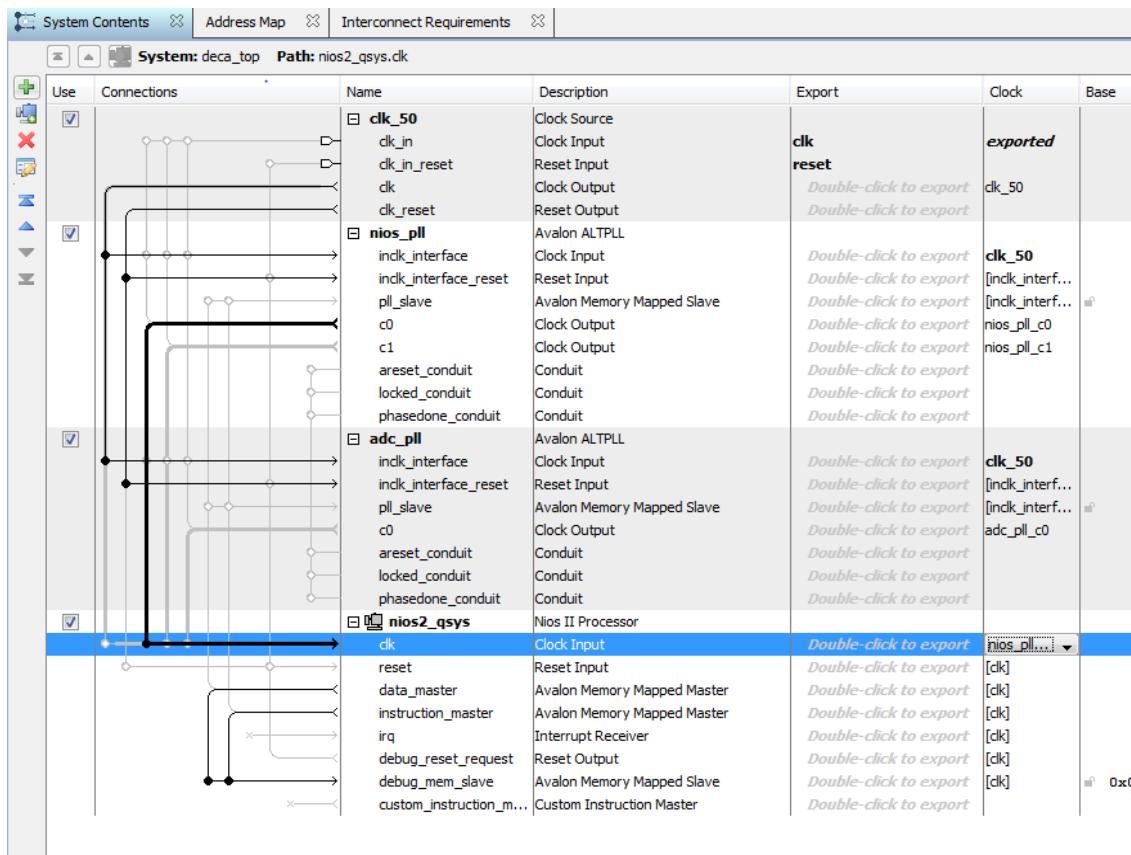
Note that there will be errors related to clocks as well. This will be resolved in a few steps.

3.4.6.6 Rename the Nios II to `nios2_qsys`.

3.4.7 Configure clock source for the Nios II processor

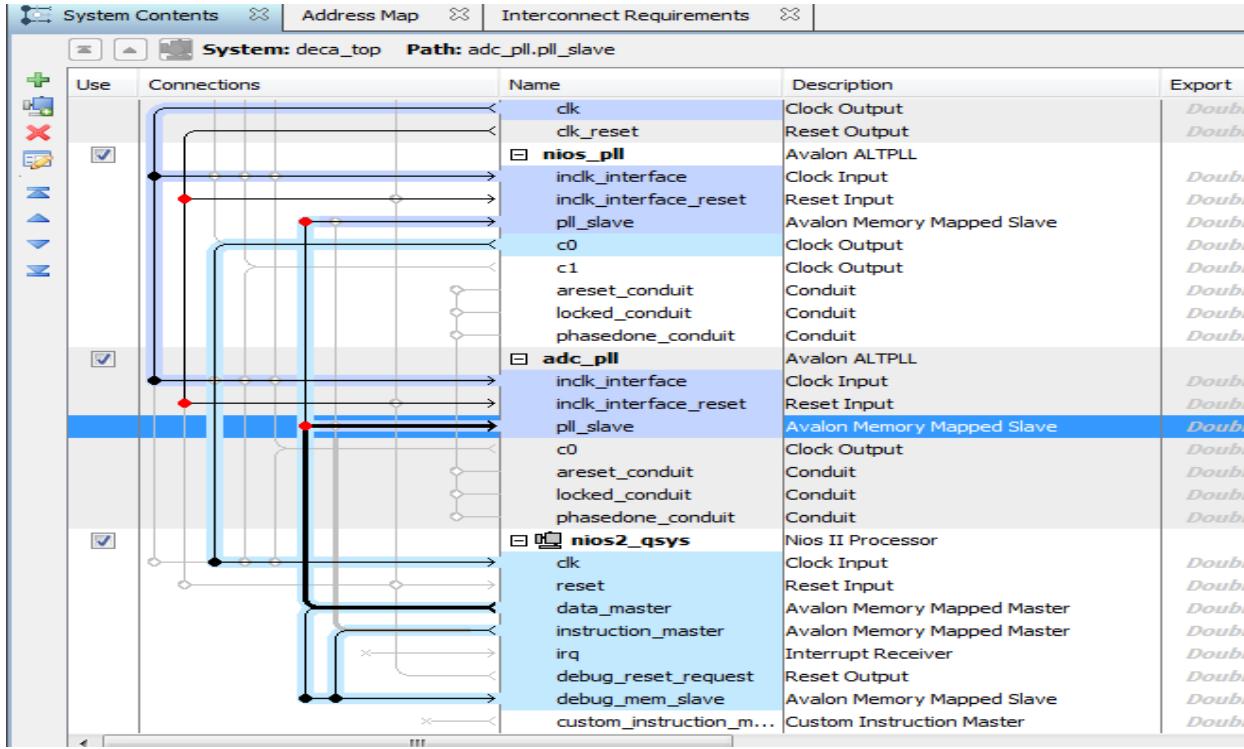
At this point there are 4 components in the system. From the drop-down list in the Clock column, double-check that the PLL is set up with the `clk_50` source. Note that we made this connection with the connection dots in an earlier step.

- 3.4.7.1 Change the clock setting for the `nios2_qsys` to be driven by the `nios_pll_c0` source. Notice that the connection in the "Connections" column is automatically made for you. Either method can be used to make clock connections in Qsys. Your system should now look as follows:



3.4.7.2 Connect the pll_slave of both the nios_pll and the adc_pll to the data_master of the nios2_qsys.

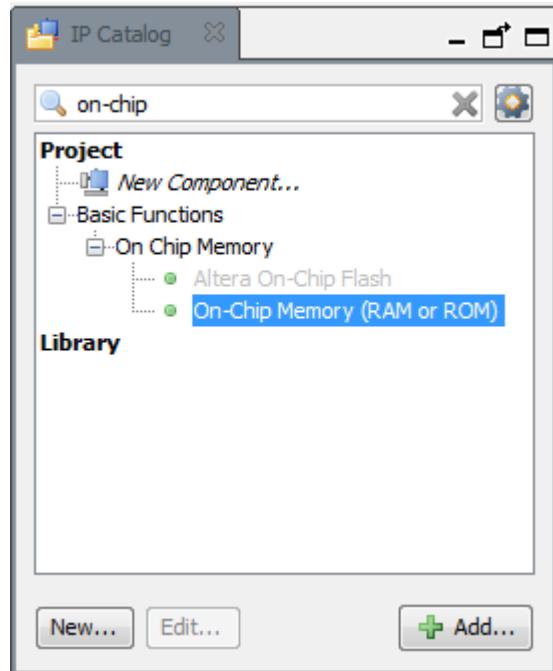
Your system should now look as follows:



3.4.8 Add On-Chip Memory

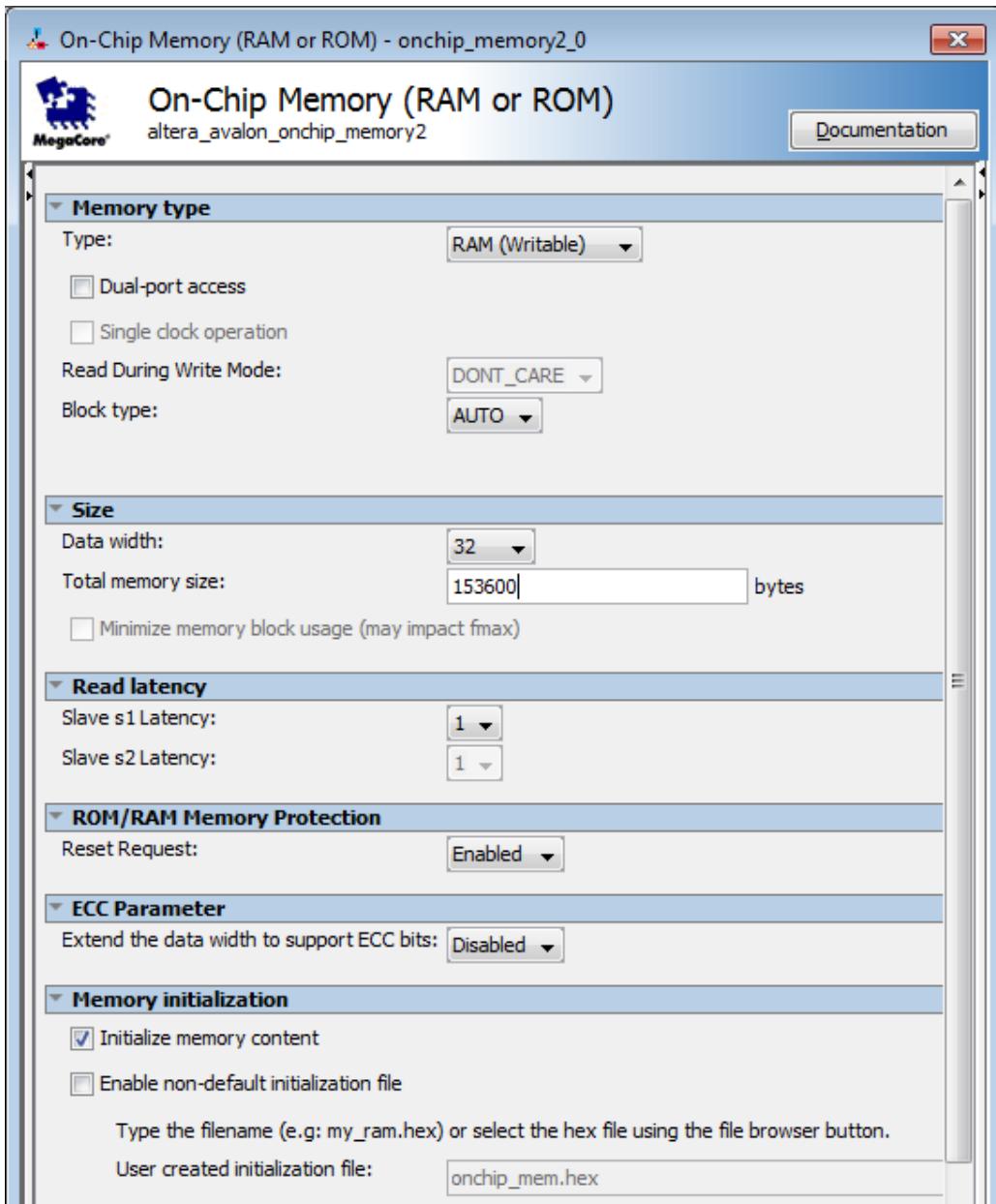
Altera FPGAs provide internal on-chip memory blocks that can be used to build up an internal RAM (or ROM) block of memory. In this lab, this provides the Nios II with access to very low-latency high speed memory for executable code and variable storage.

- 3.4.8.1 In the IP Catalog panel, type "on-chip" in the search bar. You should see the On-Chip Memory (RAM or ROM) appear under **Basic Functions → On Chip Memory**.



3.4.8.2 Double click the component or select it and click "Add..." to add it to the system. The On-Chip Memory parameter editor will open.

3.4.8.3 Change the total memory size parameter to 153600 bytes, or type 150k, and the field will update



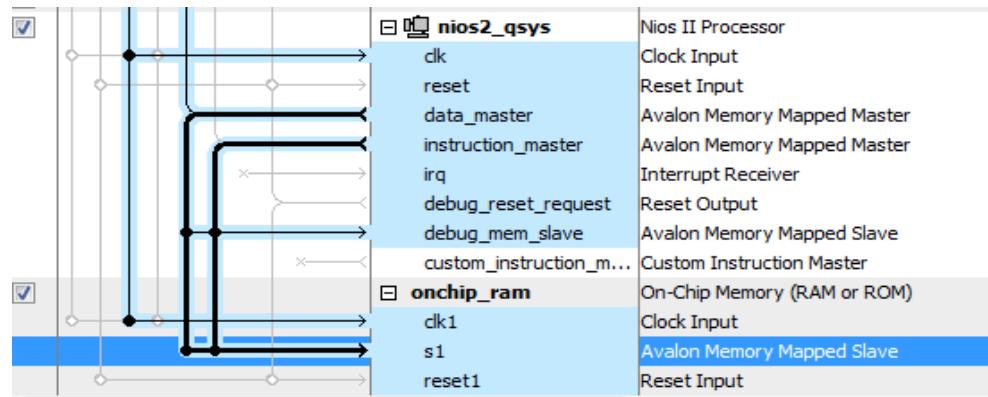
3.4.8.4 Accept the defaults for the remaining fields and click Finish to add the component to the system.

Don't worry about the errors; they will be corrected later in the lab.

3.4.8.5 Rename the component to **onchip_ram**.

3.4.8.6 Using the Clock column, change the clock Input of the onchip_ram to the nios_pll c0 clock source.

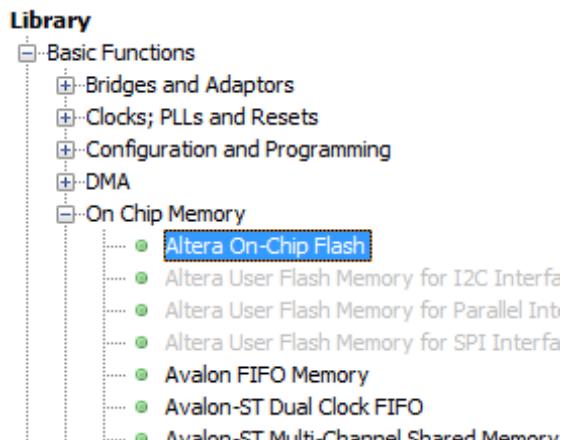
3.4.8.7 Using the Connections column, connect the s1 Avalon Memory Mapped Slave interface of the onchip_ram to the nios2_qsys.instruction_master and nios2_qsys.data_master.



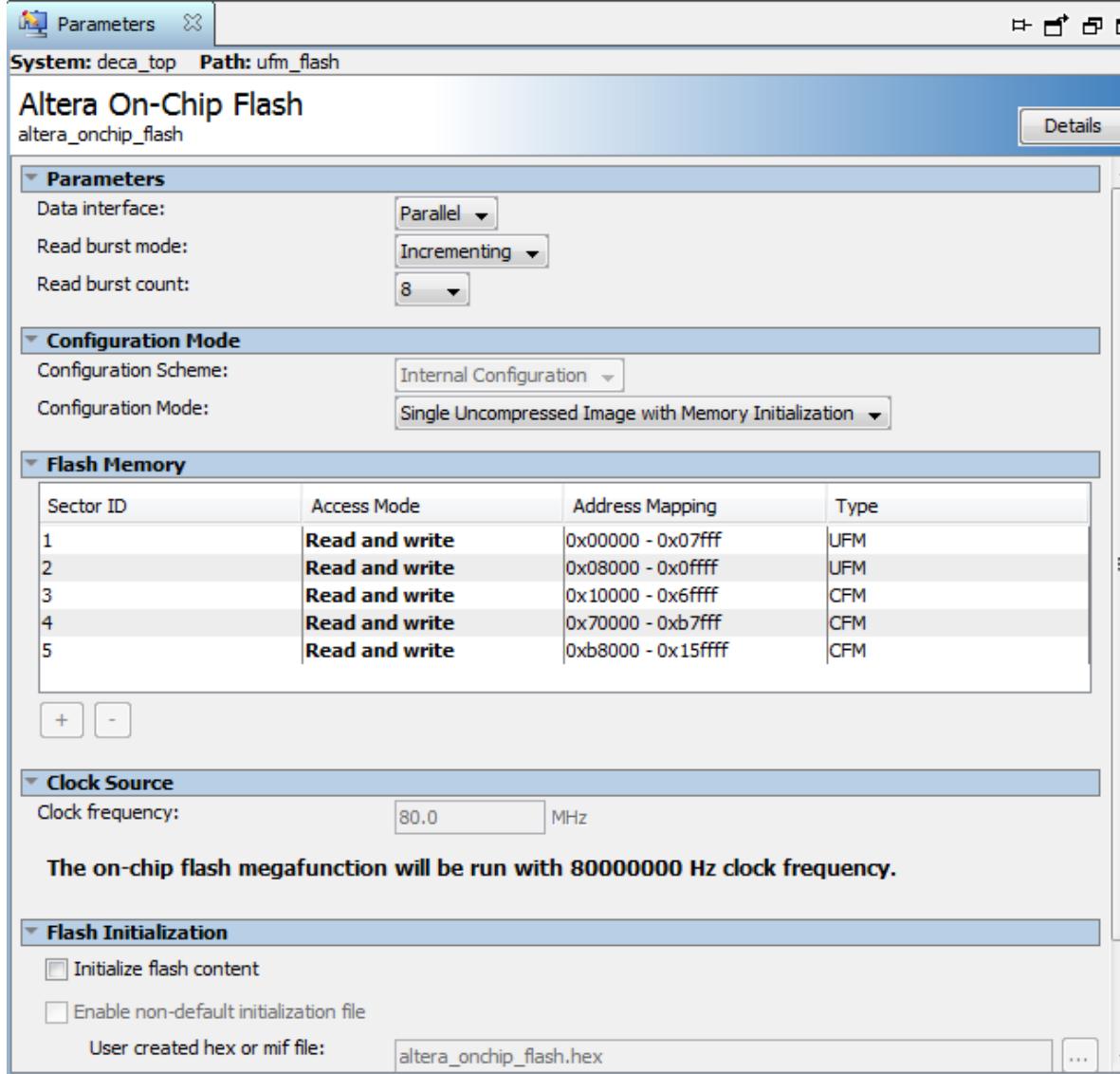
3.4.9 Add an On-Chip Flash

The MAX 10 FPGA contains on-chip flash which is used to store the FPGA configuration and can also be used to store Nios II code or other non-volatile data.

3.4.9.1 Expand Basic Functions. Expand On Chip Memory and double click on Altera On-Chip Flash.



- 3.4.9.2 Change the Configuration Mode to Single Uncompressed Image with Memory Initialization file and click Finish.



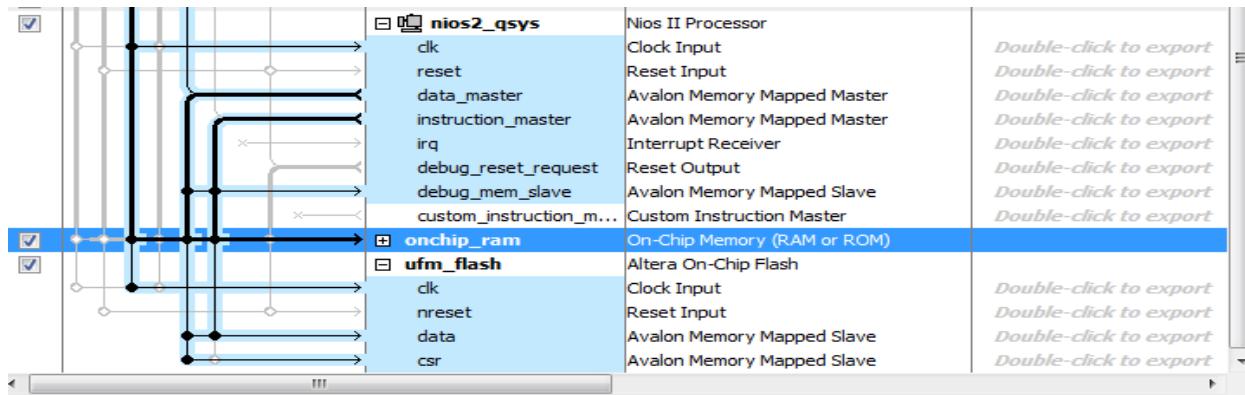
- 3.4.9.3 Rename the component to **ufm_flash**

- 3.4.9.4 In the clock column, select nios_pll c0 as the clock for the ufm_flash.

- 3.4.9.5 Connect the data to both nios2_qsys/data_master and nios2_qsys/instruction_master

- 3.4.9.6 Connect the csr (control and status registers) to the nios2_qsys.data_master only

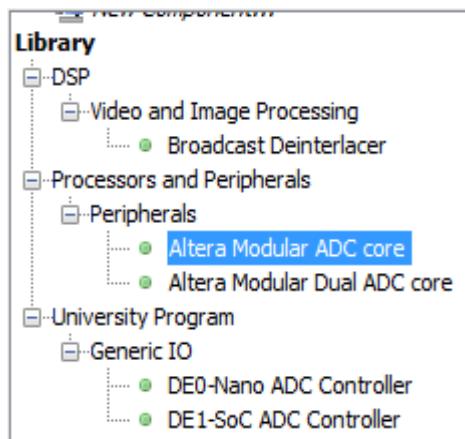
The system should look like this:



3.4.10 Add an Internal MAX 10 ADC core

The MAX 10 FPGA includes an analog block with ADC core.

3.4.10.1 Expand Processors and Peripherals → Peripherals then double click on Altera Modular ADC core.

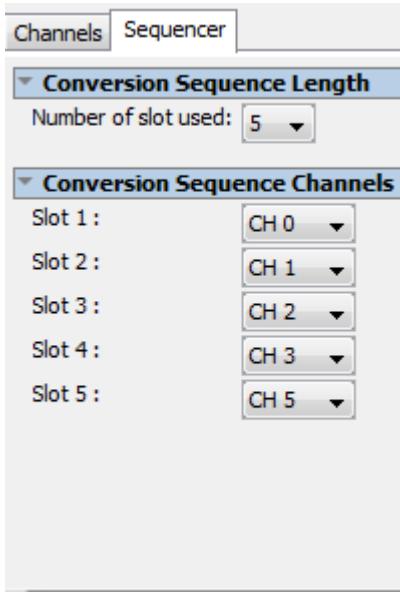


3.4.10.2 Leave the default settings on the General tab.

3.4.10.3 Click on the Channels tab and enable the following channels (0,1,2,3,5)

3.4.10.4 Click on the Sequencer tab and for the Number of slot used select 5.

Using the pull-down menus, arrange the channels into the sequencer slots as shown below.



Click Finish.

3.4.10.5 Rename the component to `modular_adc`.

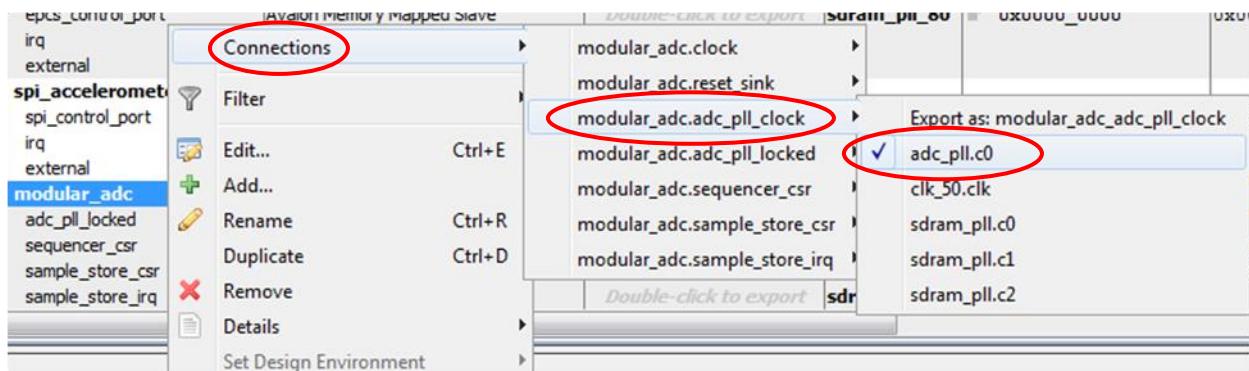
3.4.10.6 In the clock column, select `nios_pll_c0` as the clock for the `modular_adc.clock`

3.4.10.7 Connect the `sequencer_csr` to only the `nios2_cpu.data_master`

3.4.10.8 Connect the `sample_store_csr` to only the `nios2_cpu.data_master`

3.4.10.9 Connect the `irq` to the `nios2_qsys.irq`

3.4.10.10 The modular ADC core has some additional connections that need to be made. To make these connections, right click on the component and go to the “Connections” and then “modular_adc.adc_pll_clock” and then select “adc_pll.c0” to make this connection



3.4.10.11 Similarly, right click on the component again and go to the “Connections” and then “modular_adc.adc_pll_locked” and then select “adc_pll.locked_conduit” to make this connection.

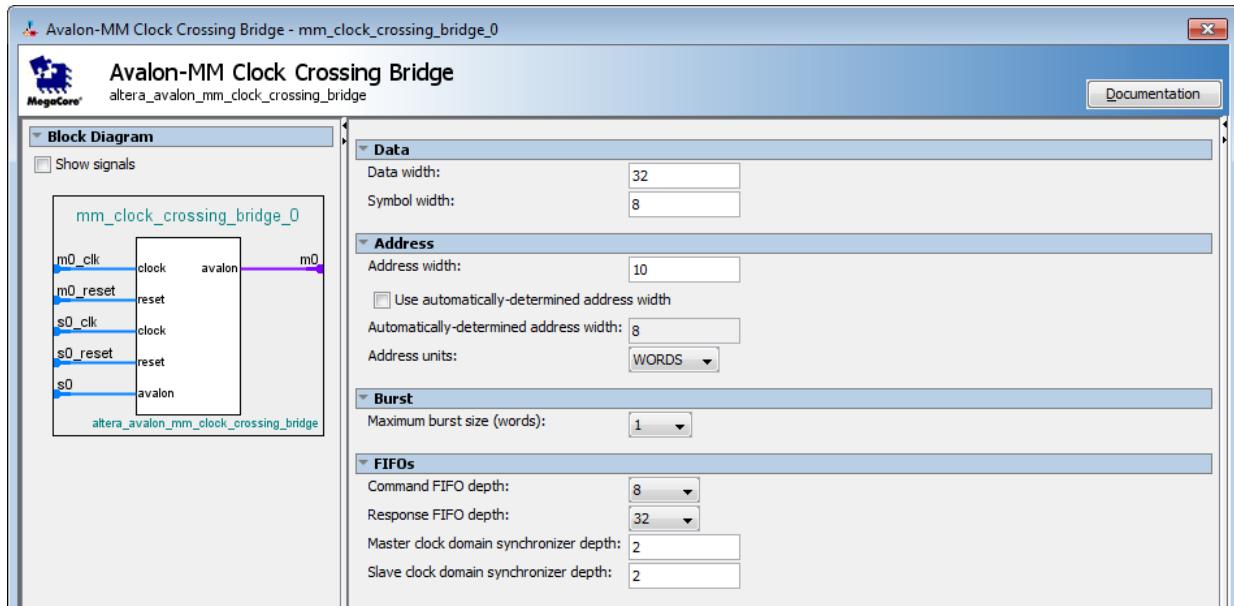
3.4.11 Add Avalon-MM Clock Crossing Bridge Peripheral for the “slow” peripherals.

We will place several “slow” peripherals in a separate clock domain from the Nios II processor. While Qsys can automatically handle crossing clock domains even if a bridge is not used, adding the bridge allows more control on crossing the clock domains. If the bridge were not explicitly added, Qsys would insert clock domain crossing logic for each peripheral. With the bridge, a single clock crossing bridge is built into the system for all of the slow peripherals.

3.4.11.1 From the IP Catalog menu, **Basic → Bridges and Adapters → Memory Mapped** and double click on Avalon-MM Clock Crossing Bridge.

3.4.11.2 Change the Address Units to WORDS.

3.4.11.3 Change the Command FIFO depth to 8 and the Response FIFO depth to 32.



Click Finish.

3.4.11.4 Right click on the Name field and choose Rename from the pop up menu. Name this bridge **slow_periph_bridge**.

3.4.11.5 Connect the nios2_qsys's data_master port to the s0 Avalon Memory Mapped Slave of this bridge. The m0 master port will be connected in the upcoming steps.

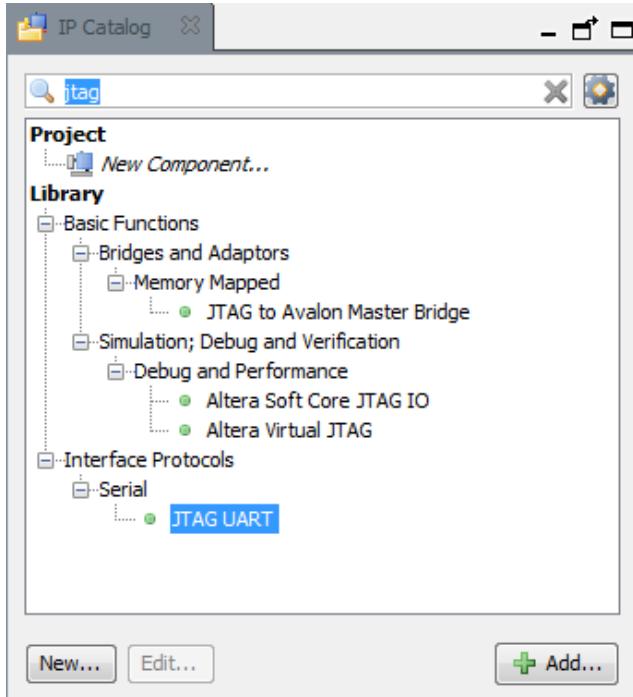
3.4.11.6 In the clock column, select nios_pll_c0 as the clock for the s0_clk

3.4.11.7 In the clock column, select nios_pll_c1 as the clock for the m0_clk

3.4.12 Add the JTAG UART Peripheral

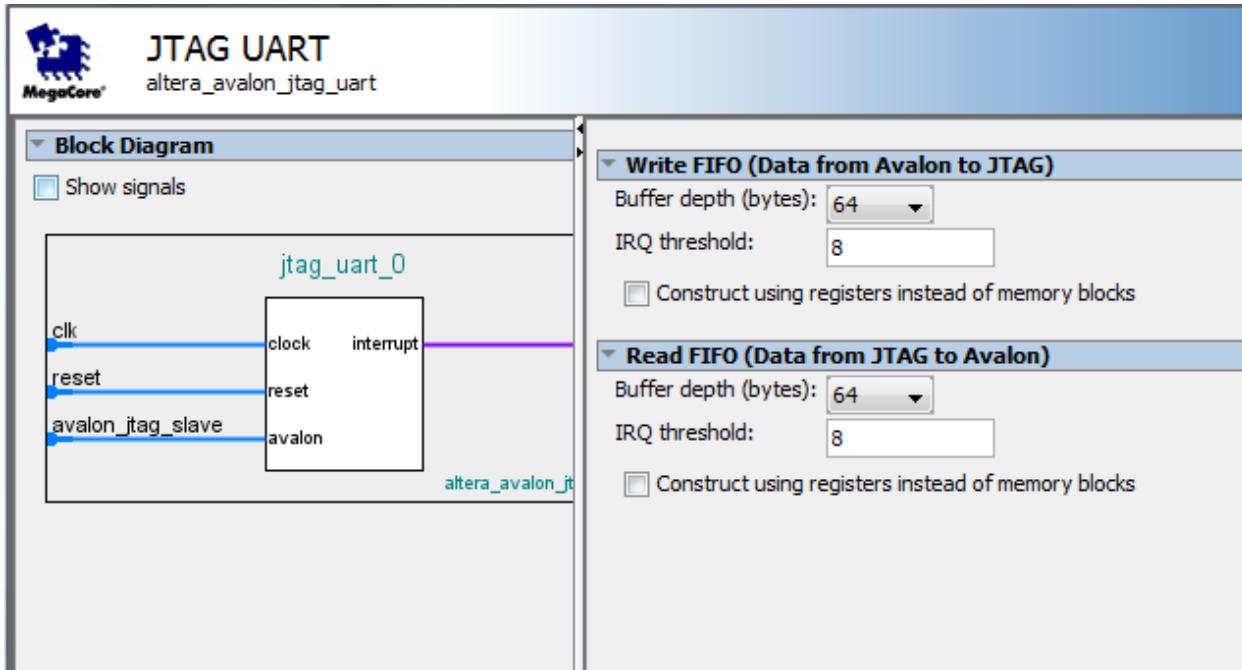
Many software developers like to have access to a debug serial port from the target to leverage `printf` debugging, input control, log status information, etc. The JTAG UART connects to Nios II processor to the debugger console in the Nios II IDE for easy debug and development using a console interface.

3.4.12.1 In the IP Catalog search bar, type **JTAG UART**. You should see the JTAG UART peripheral appear under: **Protocols → Serial**.



3.4.12.2 Double-click the component or select it and click "Add..." to add it to the system. The JTAG UART parameter editor will open.

3.4.12.3 Verify that the parameters for the Write and Read FIFO as the same as below.

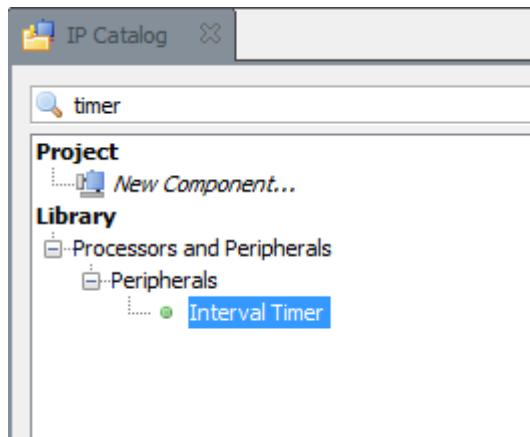


- 3.4.12.4 Click Finish to add the component to the system. Don't worry about the errors; they will be addressed later.
- 3.4.12.5 Rename the component `jtag_uart`.
- 3.4.12.6 Connect the `avalon_jtag_slave` port of the `jtag_uart` to the `data_master` port of the `nios2_qsys`.
- 3.4.12.7 In the clock column, select `nios_pll_c0` as the Clock Input.
- 3.4.12.8 Connect the `irq` port of the `jtag_uart` to the `irq` port of the `nios2_qsys` processor

3.4.13 Add a Timer

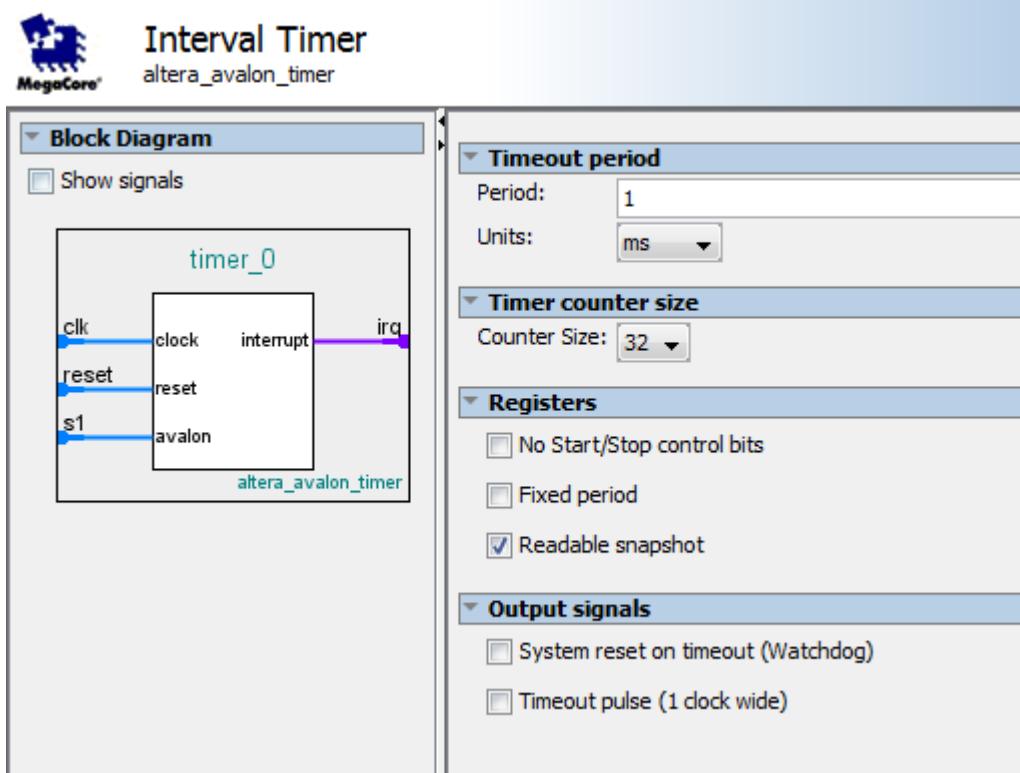
Many software applications make use of periodic interrupts to maintain timing requirements and prevent system lockups.

- 3.4.13.1 In the IP Catalog search bar, type `timer`. The Interval Timer component will appear under **Processors and Peripherals → Peripherals**.



3.4.13.2 Double-click the component or select it and click "Add..." to add it to the system. The Altera Interval Timer parameter editor will open.

3.4.13.3 Ensure the parameters are set to match the following such that the timer interval is 1 ms and the size is 32 bits.



3.4.13.4 Click Finish. Do not worry about the warnings/errors as they will be fixed once we connect the clock to the IP.

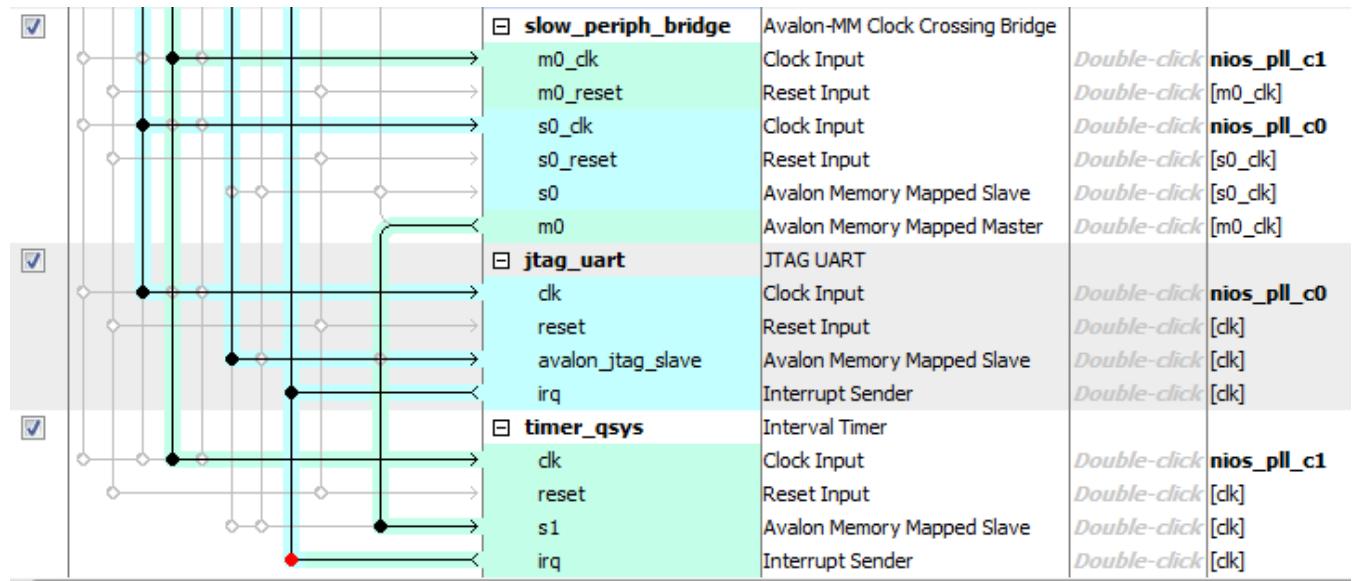
3.4.13.5 Rename the component **timer_qsys**.

3.4.13.6 Connect the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge. To do this you will need to hover your mouse over the connections area and pay attention to which line is the connection to the m0 master port of the bridge.

3.4.13.7 In the clock column, select nios_pll_c1 as the clock for the Clock Input.

3.4.13.8 Connect the irq port of the timer_qsys to the irq port of the nios2_qsys processor

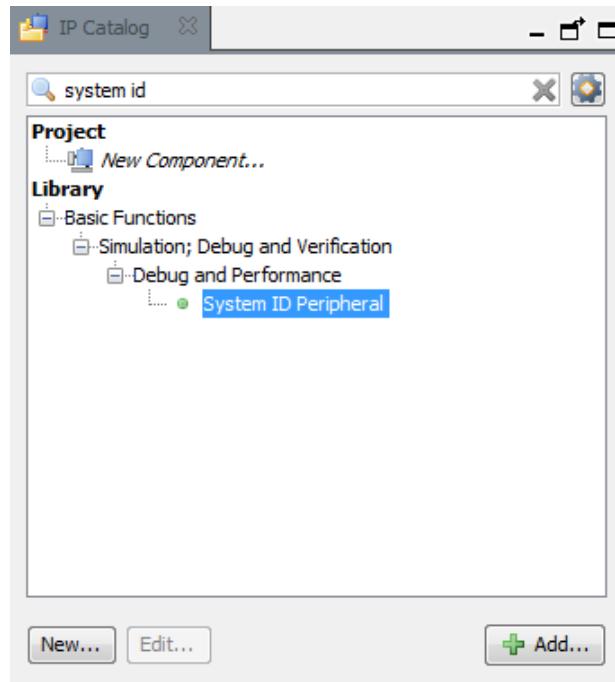
When you move your mouse cursor away, the connections should look as shown in the figure below. Double check these connections carefully:



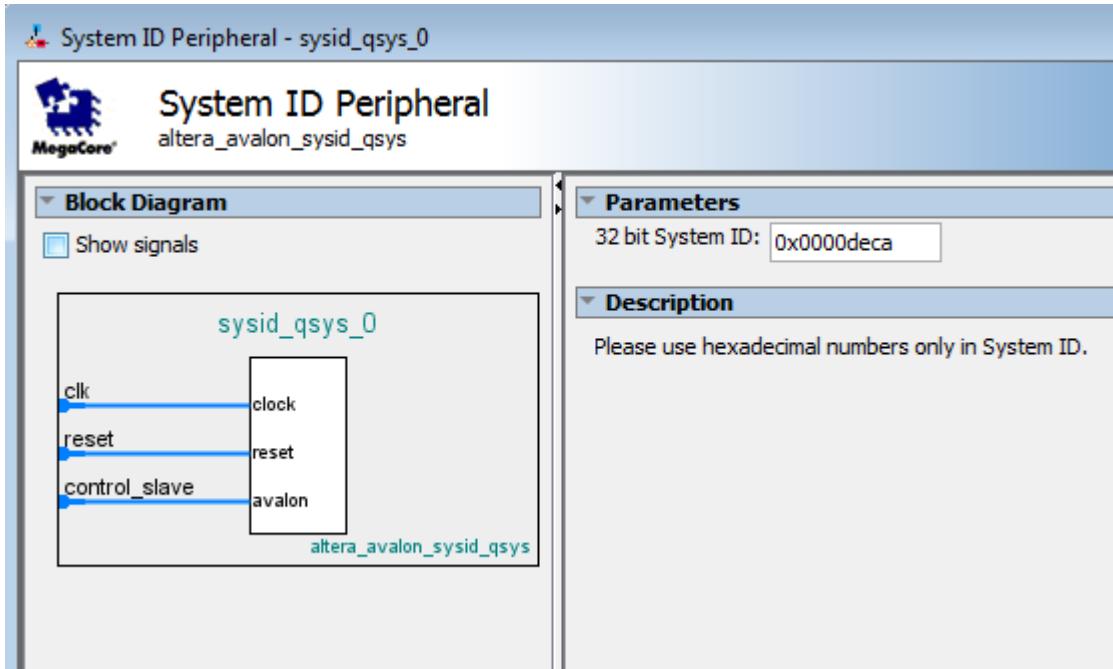
3.4.14 Add a System ID Peripheral

This is a VERY IMPORTANT peripheral to have in your system. It allows the Nios II development tools to validate that the software application is being built for the correct hardware system.

- 3.4.14.1 In the IP Catalog search bar, type **system id**. The System ID Peripheral will appear under **Basic Functions** → **Simulation; Debug and Verification** → **Debug and Performance**



3.4.14.2 Double-click the component or select it and click "Add..." to add it to the system. The System ID Peripheral parameter editor will open.



3.4.14.3 Edit the 32 bit System ID to any value you like, or use **0x0000deca**

3.4.14.4 Select Finish and ignore the errors.

3.4.14.5 Rename the component **sysid_qsys**.

3.4.14.6 Change the connection on the control_slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.

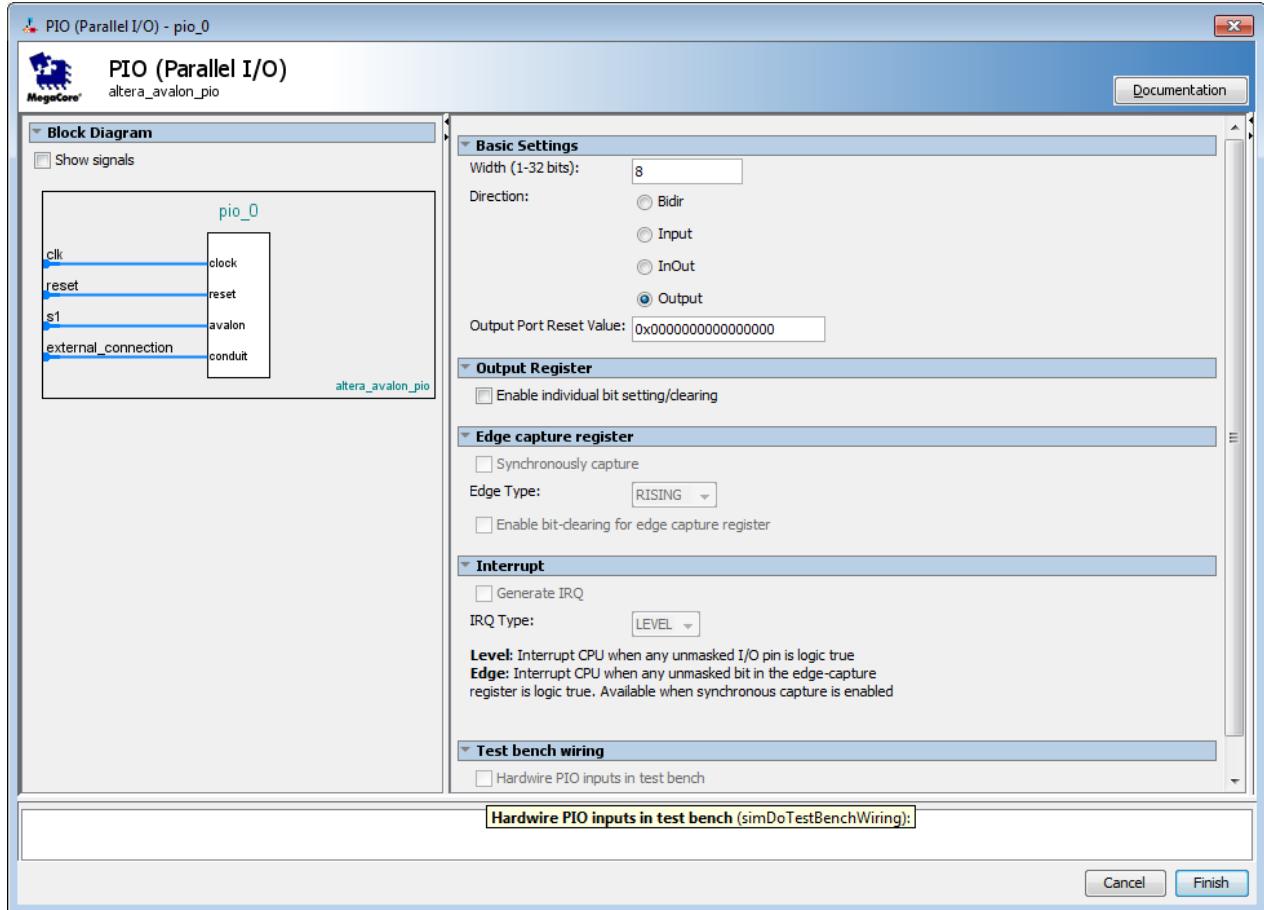
3.4.14.7 In the clock column, select nios_pll_c1 as the clock for the Clock Input

3.4.15 Add PIO Peripheral for LEDs

The DECA board has 8 LEDs on it. You can drive these LEDs with an output PIO peripheral.

3.4.15.1 From the IP Catalog menu, expand Peripherals, expand Microcontroller Peripherals and double click on PIO (Parallel I/O).

3.4.15.2 Set the Width to 8 bits. Ensure that the “Direction” is set to “Output”. Click Finish.



3.4.15.3 Rename the peripheral `led_pio`.

3.4.15.4 Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the `slow_periph_bridge`.

3.4.15.5 In the clock column, select `nios_pll_c1` as the clock for the Clock Input.

3.4.15.6 Finally, click in the "click to export" field next to the `external_connection` Conduit Endpoint and name: `led_pio_external`

3.4.16 Add PIO Peripheral for Pushbutton

The DECA board has two pushbuttons labeled KEY0, KEY1 connected to two of the FPGA I/O pins. You can use an input PIO peripheral to detect when any of these pushbuttons have been pressed and signal an interrupt to the processor. We will add a PIO peripheral for one of the buttons.

3.4.16.1 From the IP Catalog menu, expand Peripherals, expand Microcontroller Peripherals and double click on PIO (Parallel I/O).

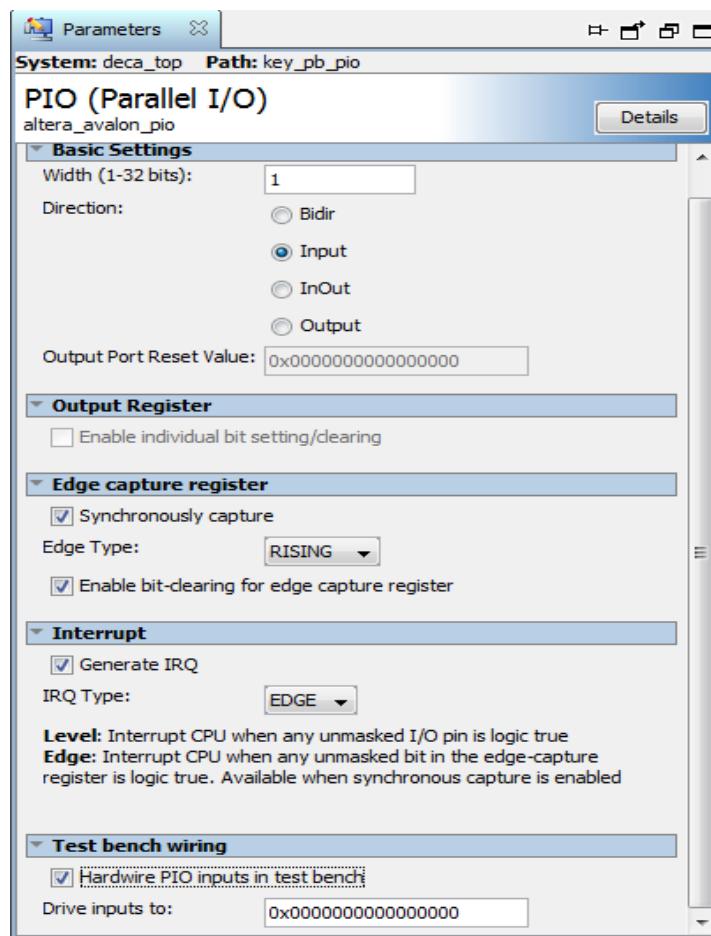
3.4.16.2 Set the "Width" to 1 bit. Set "Direction" to "Input ports only".

3.4.16.3 Check the Synchronously capture and Rising edge options in the Edge capture register section.

3.4.16.4 Check the Generate IRQ and Edge options in the Interrupt section:

3.4.16.5 Check the Hardwire PIO inputs in the test bench option and drive the inputs to 0x00000000.

Click Finish



3.4.16.6 Rename the peripheral **key_pb_pio**.

3.4.16.7 Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the **slow_periph_bridge**.

3.4.16.8 In the clock column, select **nios_pll_c1** as the clock for the Clock Input

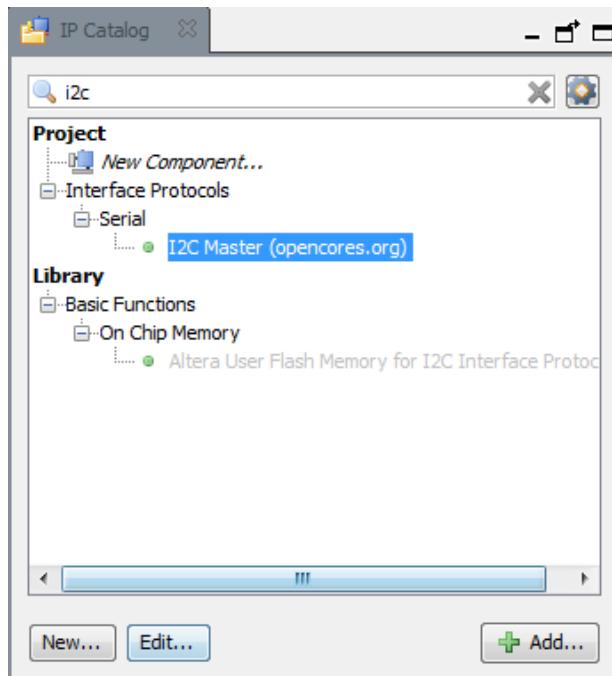
3.4.16.9 Connect the irq to the **nios2_qsys.irq**

3.4.16.10 Finally, click in the "click to export" field next to the **external_connection Conduit Endpoint** and name it: **key_pb_pio_external**

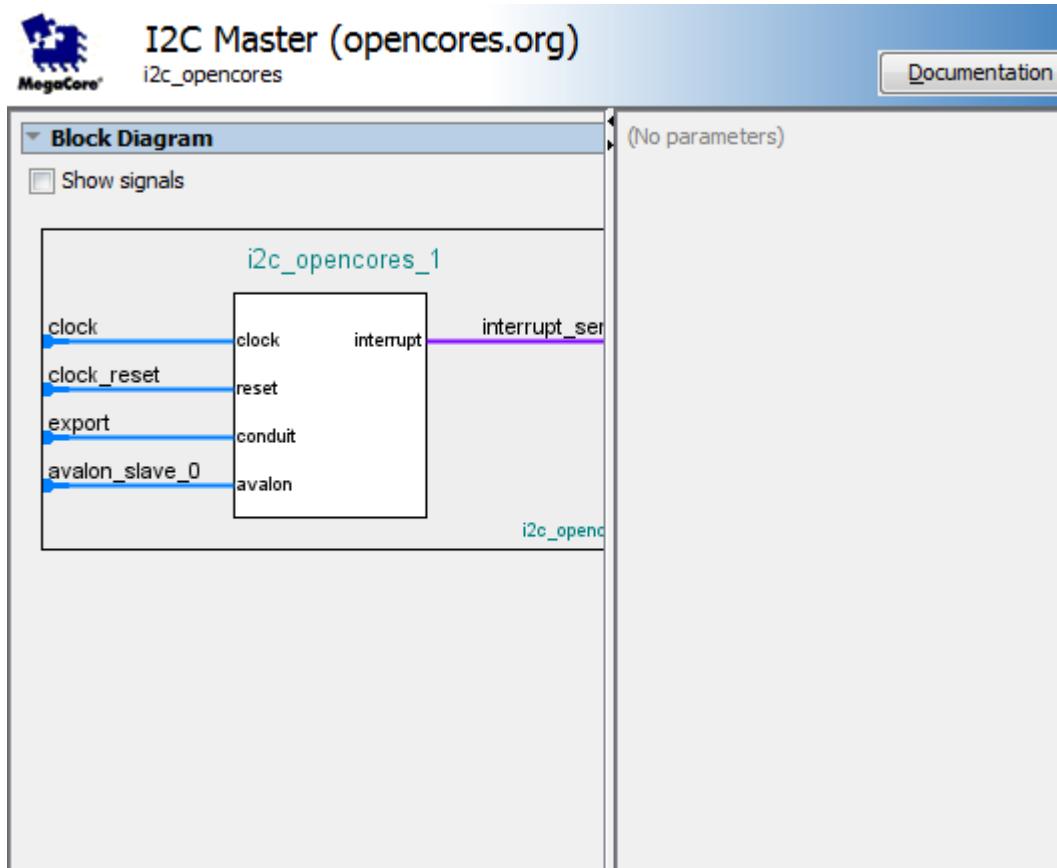
3.4.17 Add I2C Peripherals for Capsense & the Relative Humidity and Temp Sensor HDC1000

Since the Capsense and the HDC1000 sensors communicate with the host using an I2C bus, an I2C controller needs to be added to the design for each sensor. This is done for DECA because we wanted to isolate the I2C busses but in theory you could share one I2C master amongst all of the slaves if your board shares the SCL/SDA pins.

3.4.17.1 The lab files contain an I2C master IP core so there should be an I2C Master (opencores.org) in the IP catalog under **Project → Interface Protocols → Serial**.



3.4.17.2 Double-click the component or select it and click "Add..." to add it to the system. The I2C parameter editor will open.



3.4.17.3 There are no parameters to be modified in this IP so select Finish. Rename the component `capsense_i2c`.

3.4.17.4 Change the connection on the `avalon_slave_0` slave port of the peripheral to be connected to the m0 master port of the `slow_periph_bridge`.

3.4.17.5 In the clock column, select `nios_pll_c1` as the clock for the Clock Input.

3.4.17.6 The I2C pins need to be added as top-level ports to the Qsys system. Double-click the export column next to the `export` signal in the `capsense_i2c` component and type `capsense_i2c`.

<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>clk_50</code>	Clock Source	<i>exported</i>	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>nios_pll</code>	Avalon ALTPLL	<code>clk_50</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>adc_pll</code>	Avalon ALTPLL	<code>clk_50</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>nios2_qsys</code>	Nios II Processor	<code>nios_pll_c0</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>onchip_ram</code>	On-Chip Memory (RAM or ROM)	<code>nios_pll_c0</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>ufm_flash</code>	Altera On-Chip Flash	<code>nios_pll_c0</code>	multi
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>modular_adc</code>	Altera Modular ADC core	<code>multiple</code>	multi
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>timer_qsys</code>	Interval Timer	<code>nios_pll_c1</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>slow_periph_bridge</code>	Avalon-MM Clock Crossing Bridge	<code>multiple</code>	0x010
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>key_pb_pio</code>	PIO (Parallel I/O)	<code>nios_pll_c1</code>	0x000
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <code>capsense_i2c</code>	I2C Master (opencores.org)		
		clock	Clock Input	<i>Double-click to export</i> <code>nios_pll_c1</code>
		clock_reset	Reset Input	<i>Double-click to export</i> [clock]
		export	Conduit	capsense_i2c
		avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to export</i> [clock] 0x006
		interrupt_sender	Interrupt Sender	<i>Double-click to export</i> [clock]

3.4.17.7 Connect the irq to the nios2_qsys.irq

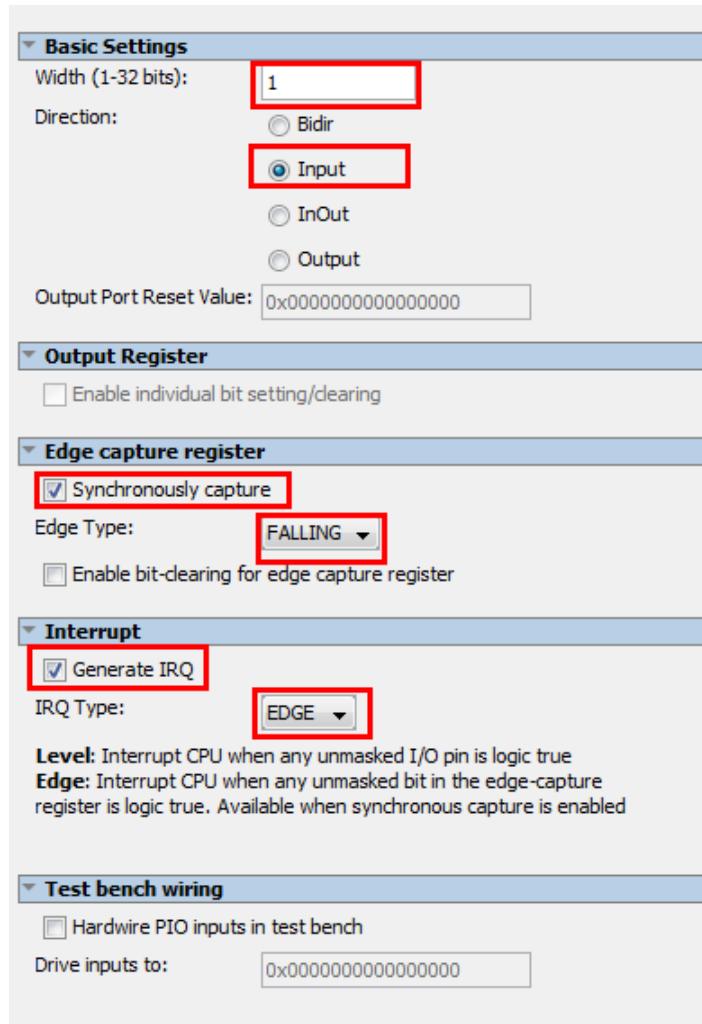
3.4.17.8 Repeat above steps for HDC1000 sensor and name the component `rh_temp_i2c` and export signal as `rh_temp_i2c`.

3.4.18 Add a PIO module for the HDC1000 data ready signal

3.4.18.1 From the IP Catalog menu, expand Peripherals, expand Microcontroller Peripherals and double click on PIO (Parallel I/O).

3.4.18.2 Configure the PIO peripheral as follows:

- “Width” set to 1 bit.
- Direction set to “Input”.
- Synchronously capture, Falling Edge
- Generate IRQ, Edge sensitive



Click Finish

3.4.18.3 Rename the peripheral **rh_temp_drdyn**.

3.4.18.4 Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.

3.4.18.5 In the clock column, select nios_pll_c1 as the clock for the Clock Input.

3.4.18.6 Connect the irq to the nios2_qsys.irq

3.4.18.7 Finally, click in the "click to export" field next to the external_connection Conduit Endpoint and name the conduit: **rh_temp_drdyn_external**

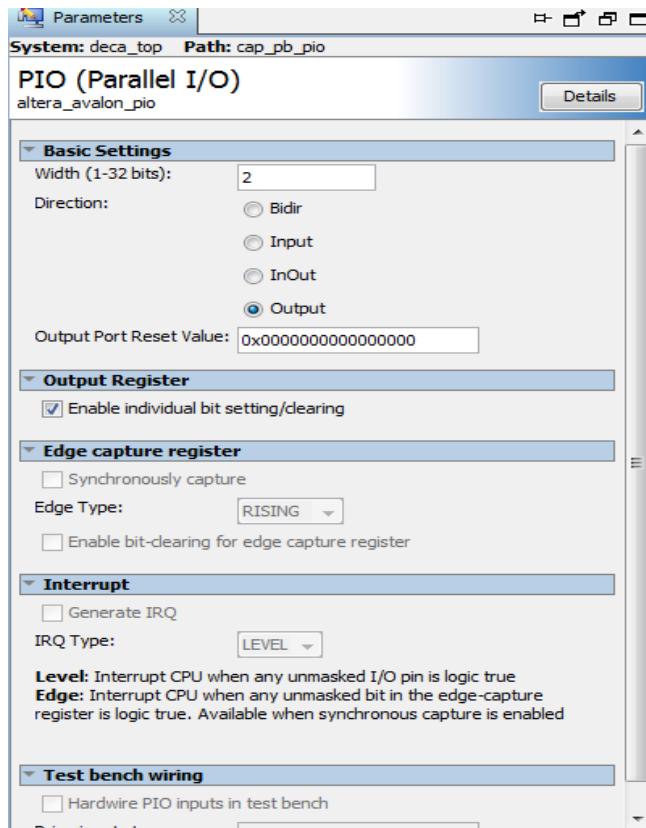
3.4.19 Add the PIO for the Capsense

This PIO will be used to display the result of a Capsense button press event, and will be connected to the LEDS on the board.

3.4.19.1 From the IP Catalog menu, expand Peripherals, expand Microcontroller Peripherals and double click on PIO (Parallel I/O).

3.4.19.2 Set the "Width" to 2 bits. Ensure that the "Direction" is set to "Output ports only".

Set the Enable individual bit setting/clearing option. Click Finish.



3.4.19.3 Rename the peripheral **cap_pb_pio**.

3.4.19.4 Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the **slow_periph_bridge**.

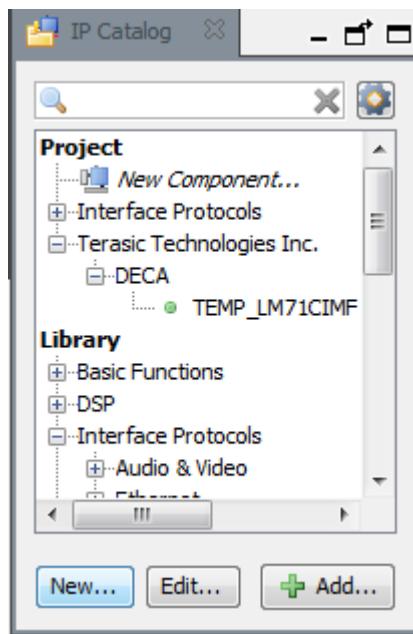
3.4.19.5 In the clock column, select **nios_pll_c1** as the clock for the Clock Input.

3.4.19.6 Connect the irq to the **nios2_qsys.irq**

3.4.19.7 Finally, click in the "click to export" field next to the **external_connection Conduit Endpoint** and name the conduit: **cap_pb_pio_external**

3.4.20 Add the temperature sensor component

3.4.20.1 In the IP Catalog, under Project, select **Terasic Technologies Inc** → **DECA** → **TEMP_LM71CIMF** and double click to add the component.



3.4.20.2 Connect the **clock_sink** to the **nios_pll_c1**, the **avalon_slave** to the **m0** of the **slow_periph_bridge** and the **conduit_end** should be exported out, you can call it the **lm71_external** port.

3.4.21 Resolve errors

At this point, Qsys will report a number of errors referencing unconnected clocks, unconnected resets, and unconnected Avalon interface because some of the components in your Qsys system are not fully connected. Once all of the interfaces are connected, these errors will disappear.

3.4.21.1 Assign Base Addresses

When the peripherals were added to the system, they were all given the default base address of 0x00000000, so the components now have overlapping addresses. Qsys will report this as an error. You can manually enter the base addresses in the Base column, or you can let Qsys automatically assign them. Automatically assign them now: select **System** → **Assign Base Addresses**.

3.4.21.2 Create Global Reset Network

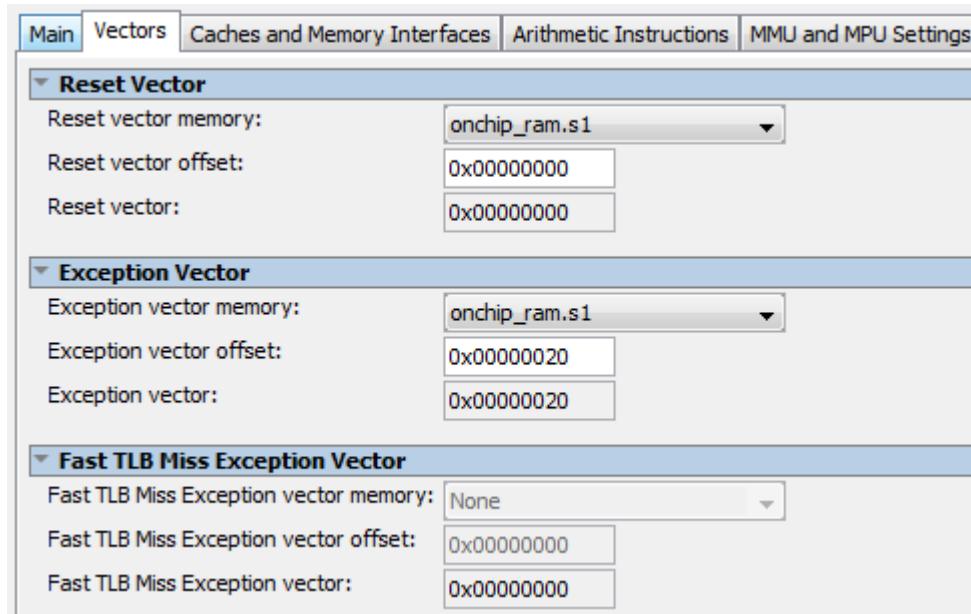
In some cases, the reset ports of the components may not have been connected. You can manually connect these ports, or you can let Qsys automatically connect them. Automatically connect them now: select **System → Create Global Reset Network**

3.4.21.3 Define the Nios II Reset and Exception Vectors

Like any processor, the Nios II requires memory locations to jump to in the event of a processor reset or exception within the execution of its code. The reset vector is the memory location to which the processor jumps on processor reset and the exception vector is the memory location to which the processor jumps on an exception. These are typically in non-volatile memory and can be at the same memory location.

3.4.21.4 To set these vectors, double-click on the Nios II component (nios2_qsys). The Nios II parameter editor will reopen.

3.4.21.5 Click on the Vectors tab and set both the reset vector memory and exception vector memory to be `onchip_ram.s1` from the pull-downs. The offset and vector values may be different than the image below but will be corrected in a following step.

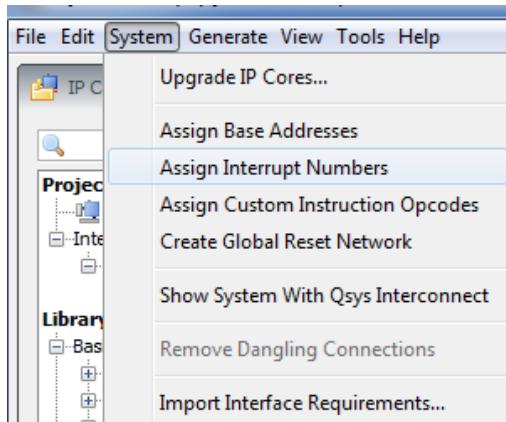


3.4.21.6 Review message window for remaining errors

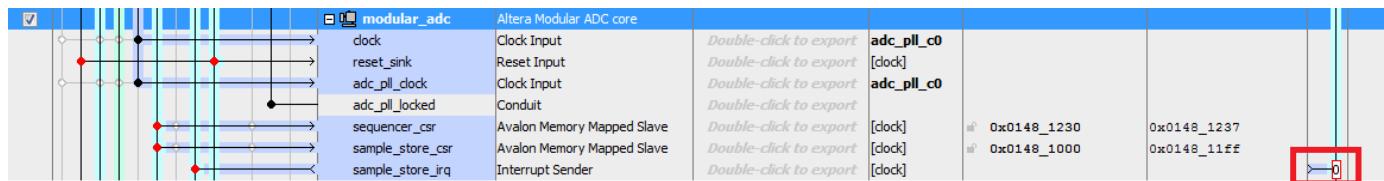
At this point, there should be no remaining errors in the Message window. Resolve any remaining errors.

3.4.22 Set Interrupt Priorities

The Nios II processor can process up to 32 independent interrupts (IRQs) from various parts of a system that can be assigned unique priorities. This system only has 7 interrupts and the priorities will need to be set manually although it can be done automatically by selecting **System → Assign Interrupt Numbers** from the Qsys menu as below.



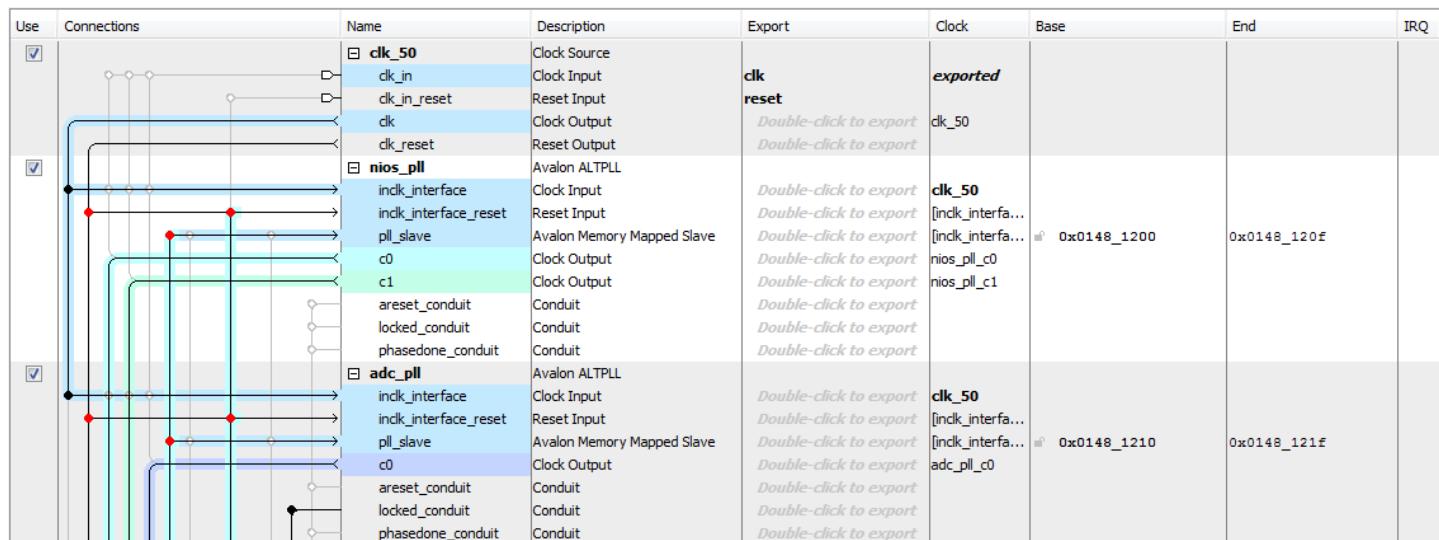
3.4.22.1 You can also manually set an IRQ priority in Qsys by double clicking the number in the IRQ column of the System Contents tab and entering the priority (priority 0 is the highest priority. For example, double click the number in the IRQ column to the right to the "irq" signal in the modular_adc module and type 0. This will give the adc component's interrupt the highest priority.

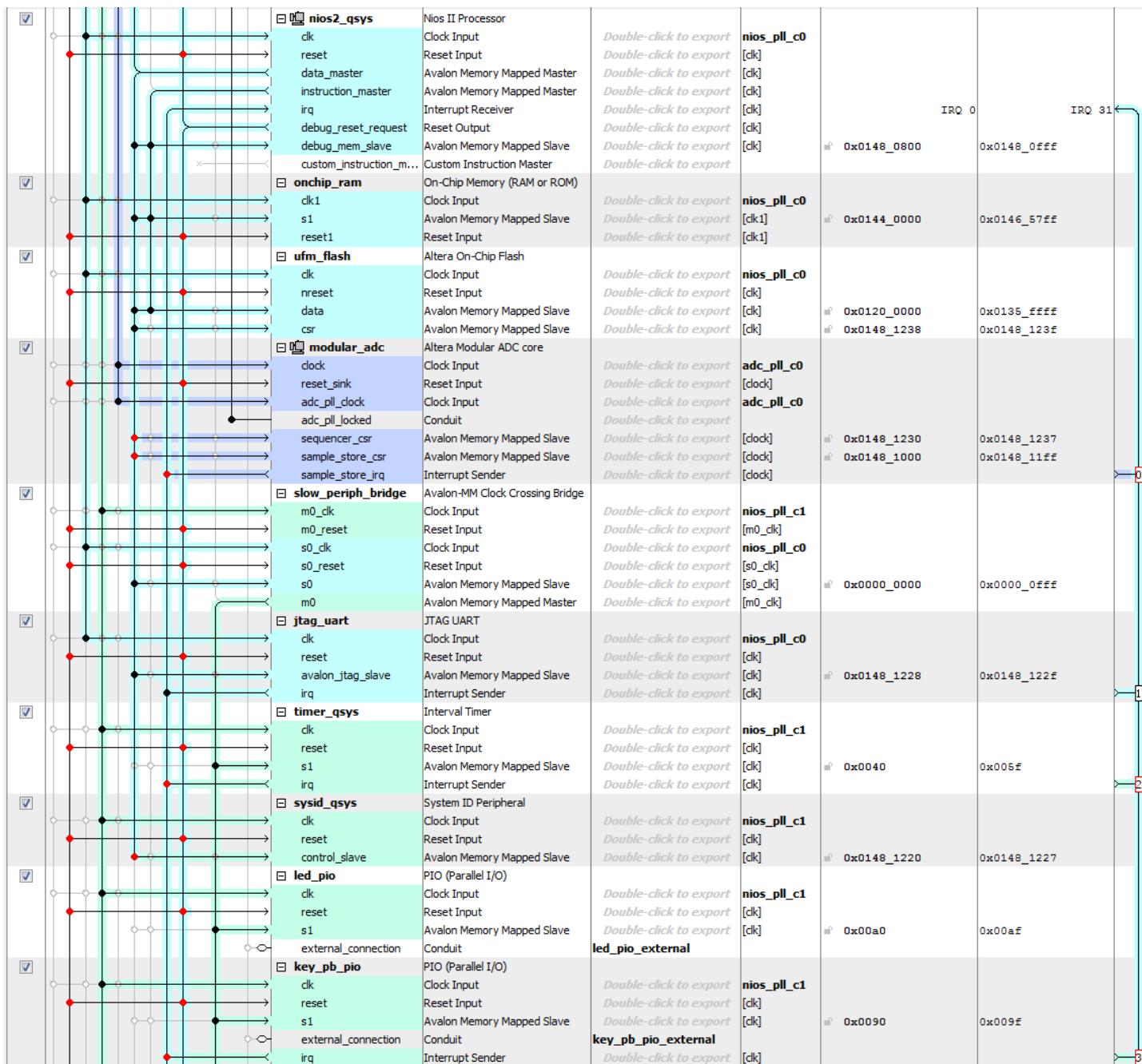


3.4.23 Check the full system

Below is a screenshot of the full Qsys system with all connections visible.

3.4.23.1 Confirm that your Qsys system matches the screenshot below.





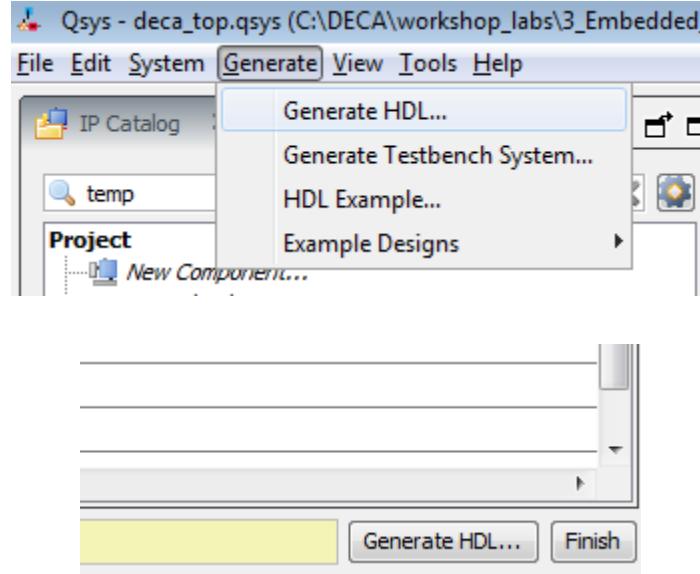


3.4.23.2 Make sure that there are no error messages in the Messages tab.

3.4.24 Generate the Qsys System

One of the great parts about Qsys is that it generates HDL (hardware description language) code from the created system so that the internals can be investigated for a better understanding. The next step is to generate the HDL from the system.

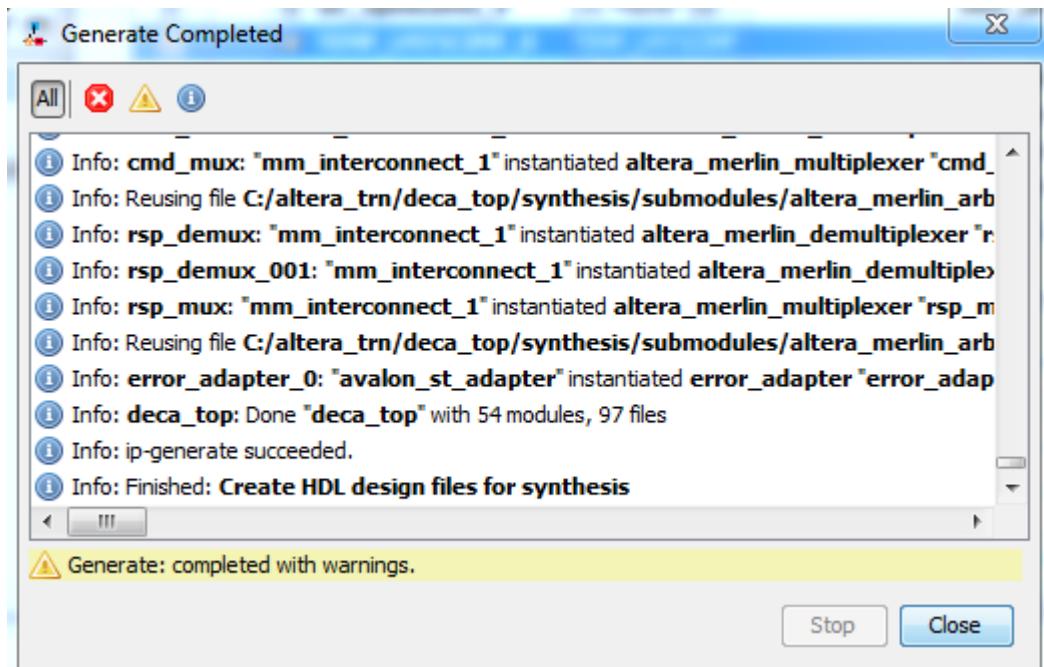
3.4.24.1 Select **Generate → Generate HDL...** from the Qsys menu or **alternatively** click the **Generate HDL...** button on the bottom right of the Qsys window.



3.4.24.2 The Generate window will appear. Select "Verilog" as the synthesis language and "None" from the simulation model dropdown. VHDL can be used but the top level file in this lab is in Verilog. The generated

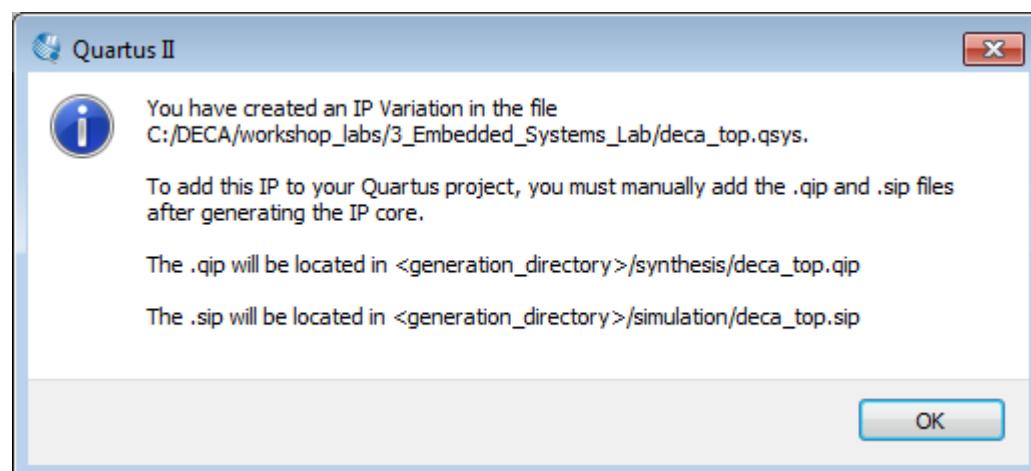
HDL files will appear in the directory pointed to by the file path specified under the Output Directory section. Leave this as the default.

3.4.24.3 Click Generate. Qsys will generate the necessary HDL for synthesis. When the generate process completes (with warnings), click Close.



3.4.25 Add the Qsys System to the Quartus Project

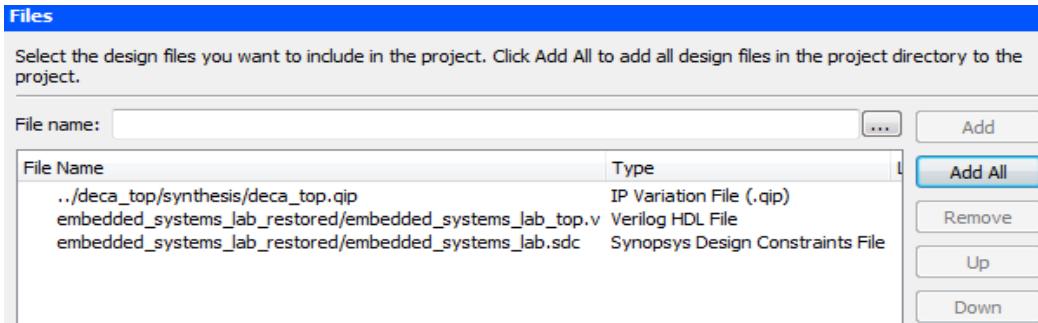
The system created in Qsys now needs to be added to your Quartus project so that it can be instantiated in the top-level design file. You can think of the Qsys system as a module or component as you would in any other FPGA design. Qsys generates IP "pointer" files for both synthesis (.qip) and simulation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



3.4.25.1 Open the project files manager: **Project → Add/Remove Files in Project** from the Quartus II menu.

3.4.25.2 Click the  and browse to the synthesis directory noted above (it should be `<project_directory>/deca_top/synthesis/`) and select `deca_top.qip`.

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

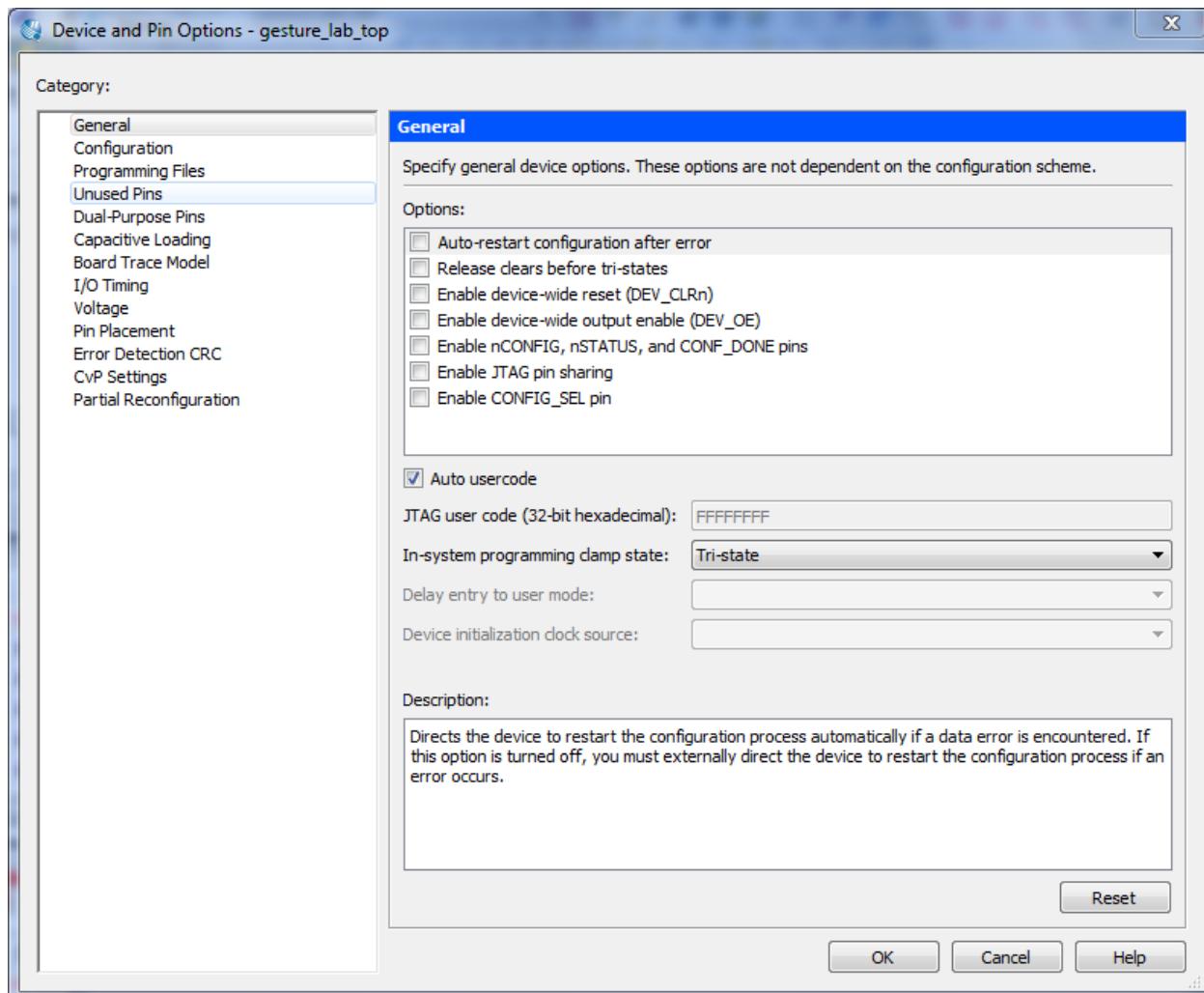


3.4.26 Compile the Quartus II project

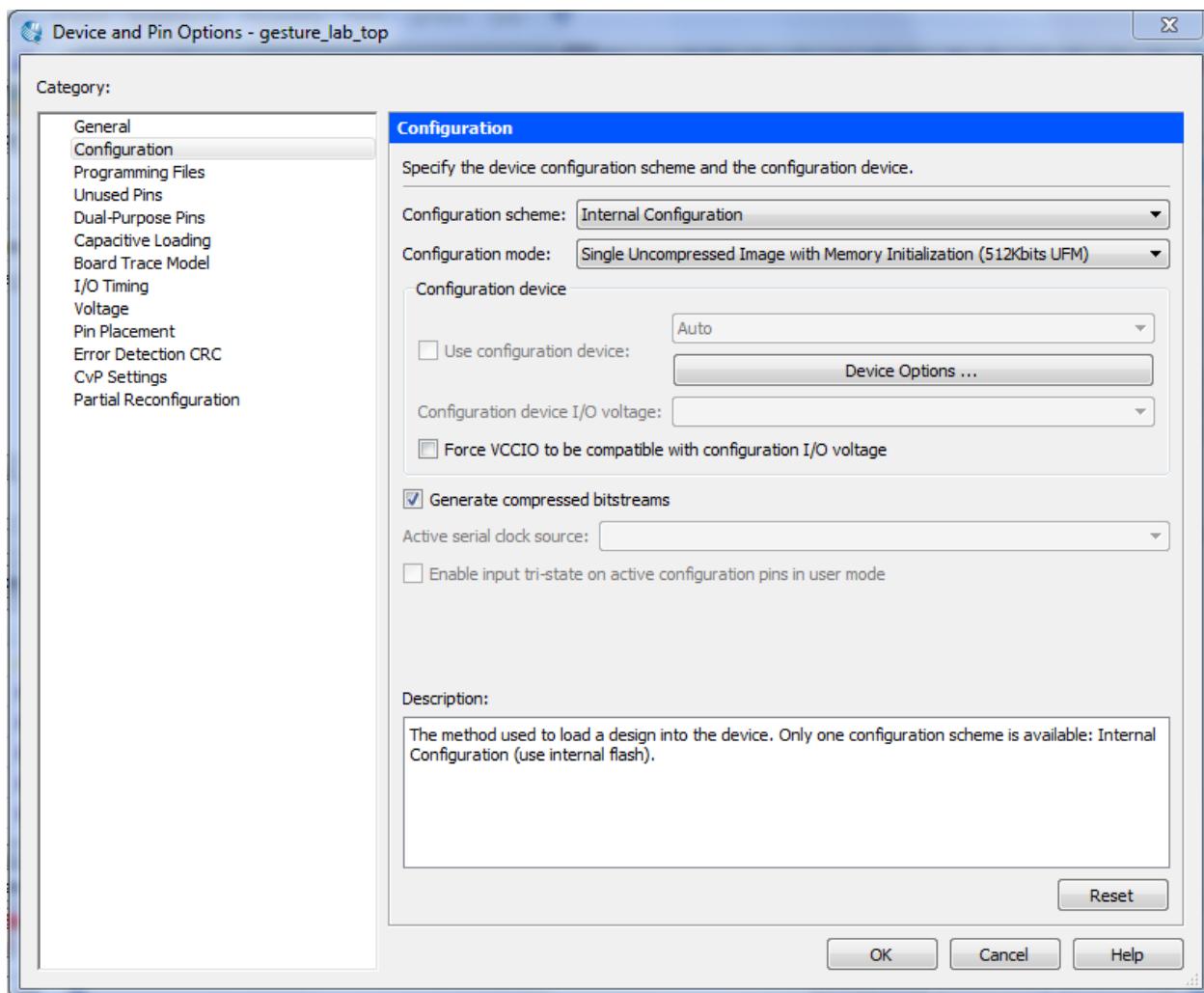
With the hardware design complete, a few device settings need to be changed before the project can be compiled to create a configuration file.

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

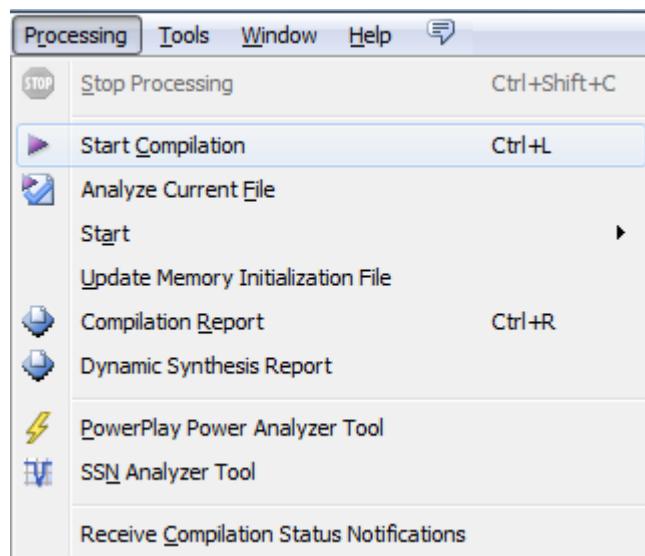
3.4.26.2 Unselect all of the checkboxes in the Options box in the General category so that they match the following.
This will allow the fitter to appropriately select the correct VCCIO for the banks that contain the device configuration pins.



3.4.26.3 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.



3.4.26.4 Kick off the compilation by selecting **Processing → Start Compilation** or double-click Compile Design in the Task window.



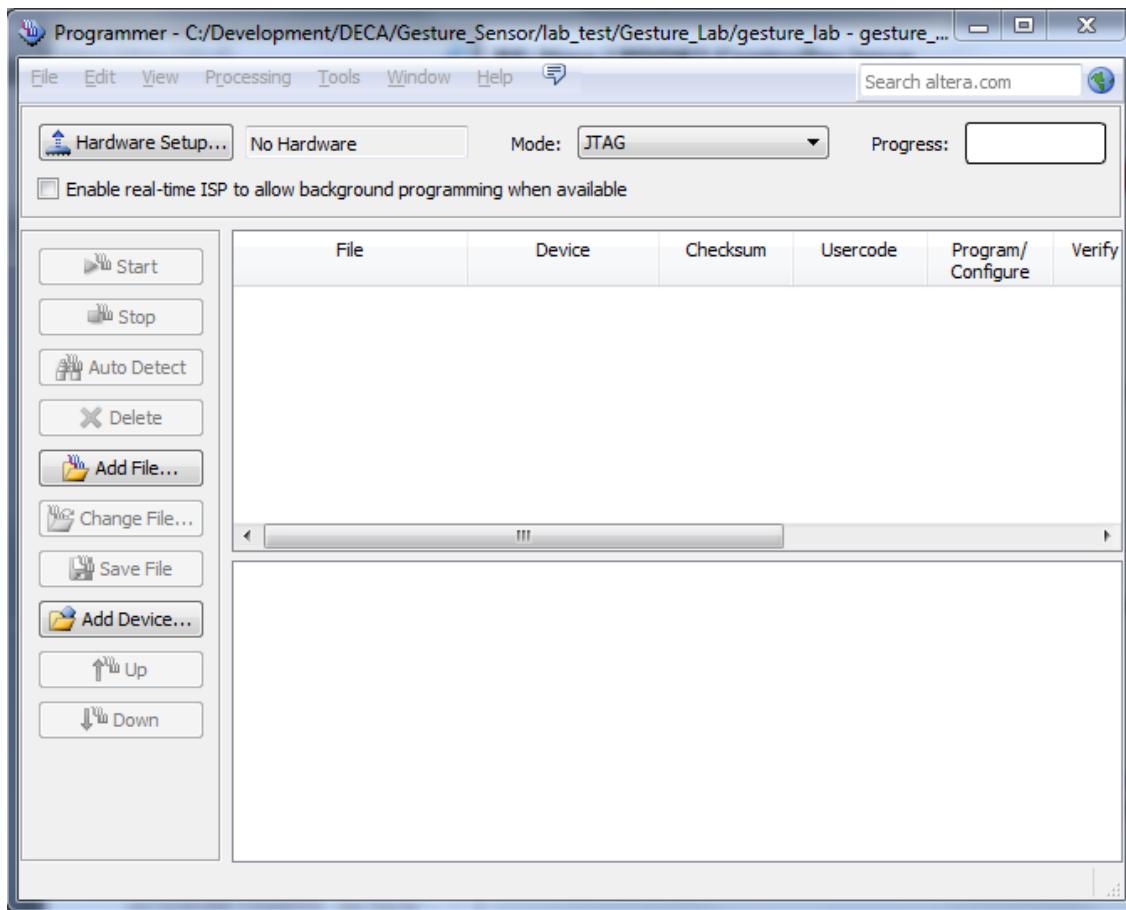
3.4.26.5 After a few minutes, the compilation should complete with no errors.

Tasks		
Flow:	Task	
Compiling	Compile Design	00:02:53
	Analysis & Synthesis	00:01:09
	Fitter (Place & Route)	00:01:03
	Assembler (Generate programming files)	00:00:09
	TimeQuest Timing Analysis	00:00:26
	EDA Netlist Writer	00:00:06
	Program Device (Open Programmer)	

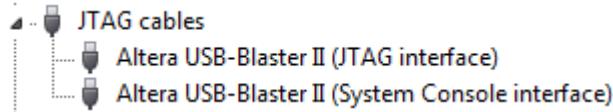
3.4.27 Download the Configuration File to DECA

Now that the hardware design is complete and has been converted into a configuration file, the DECA needs to be programmed.

3.4.27.1 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.



3.4.27.2 Connect your DECA board to your PC using a USB cable. Be sure to connect it to the mini-USB connector labeled **USB2_J10** (on the bottom right of the board). Since the USB Blaster II driver software should already be installed, the Windows' Device Manager should display two entries under "JTAG Cables".

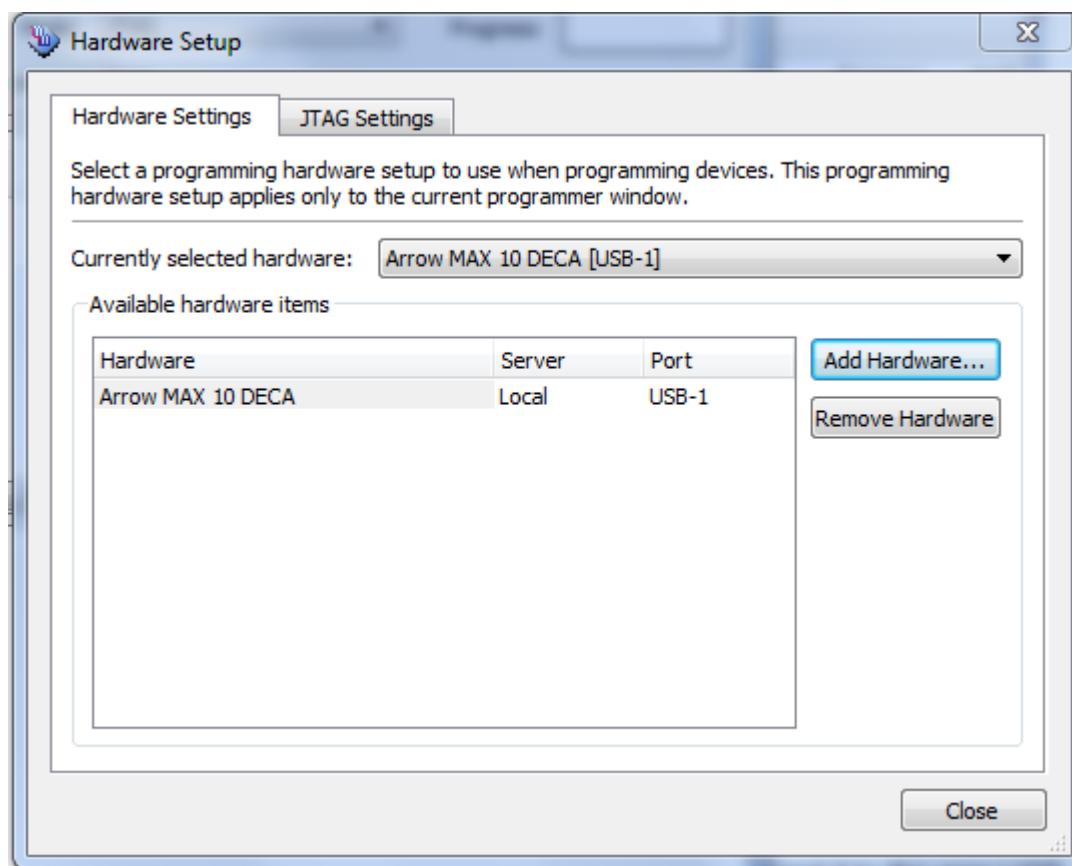


You should see a few LEDs light up on your DECA including the blue LED labeled **3.3V** and the green LED labeled **CONF_D**.

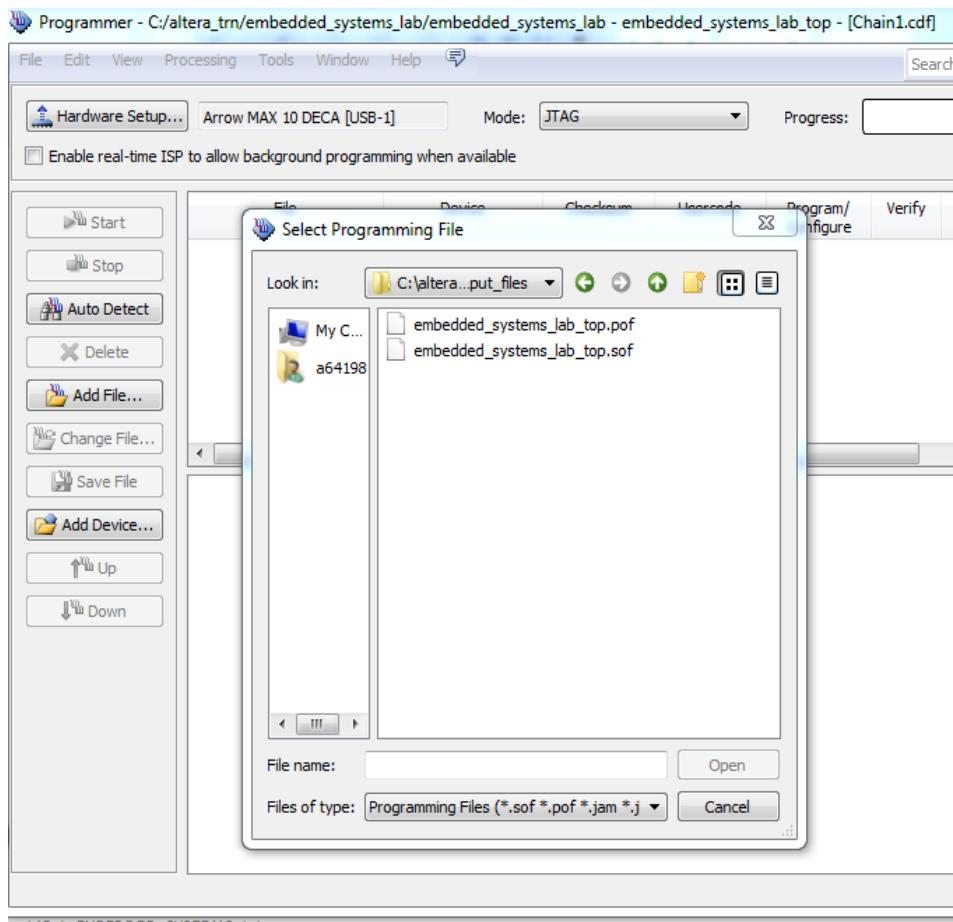


If the Device Manager shows an unconfigured USB Blaster, if Windows tries to look for drivers, or if the LEDs on the DECA do not light up, ask your workshop trainer for help.

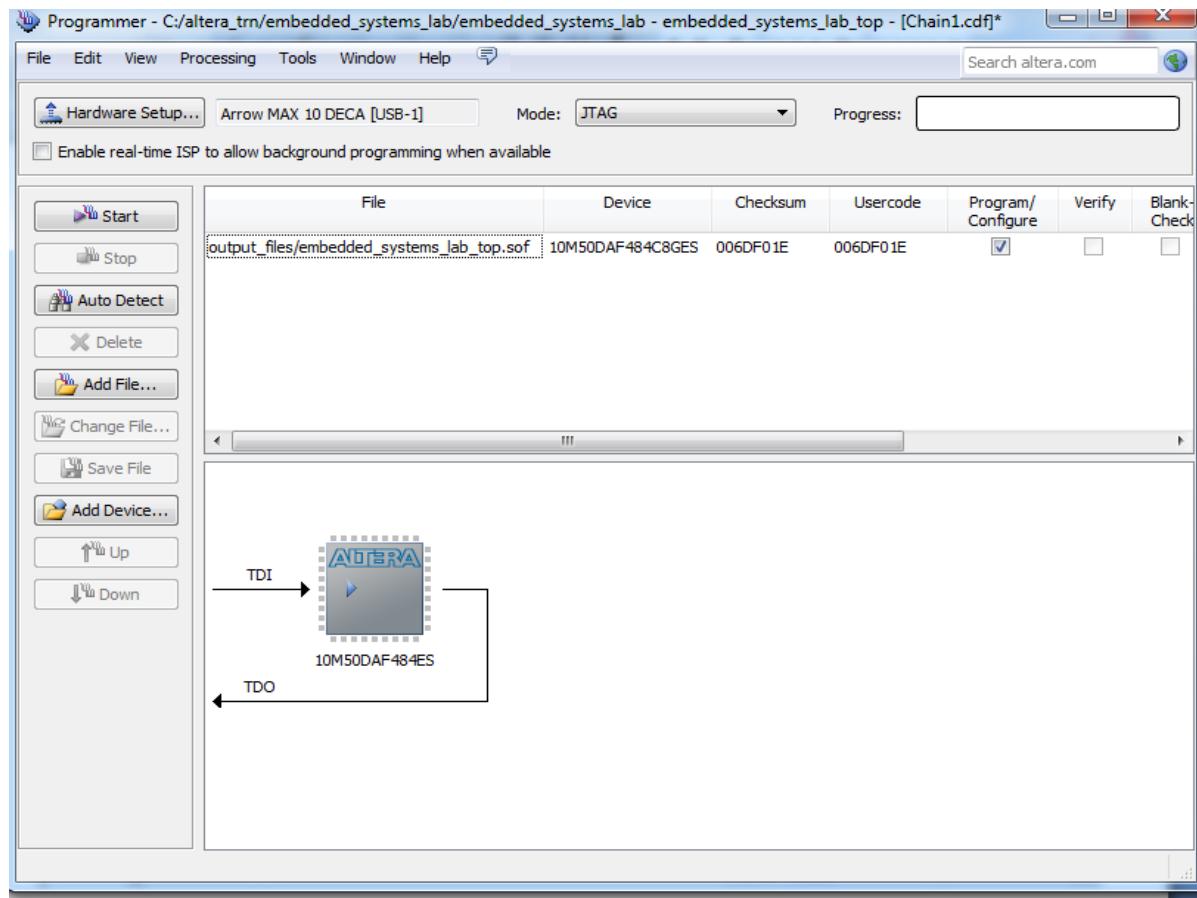
- 3.4.27.3 In the Programmer window, click Hardware Setup and double-click the Arrow MAX10 DECA entry in the Hardware pane. The Currently Selected Hardware drop-down should now show Arrow MAX10 DECA [USB-1]. Depending on your PC, the USB port number may be different. Click Close.



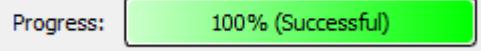
3.4.27.4 Click "Add File..." and navigate to <project_directory>/output_files/ in your compilation directory. Open the `embedded_systems_lab_top.sof` file.



3.4.27.5 Make sure that the Programmer shows the correct file and the correct part in the JTAG chain as below.



3.4.27.6 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).

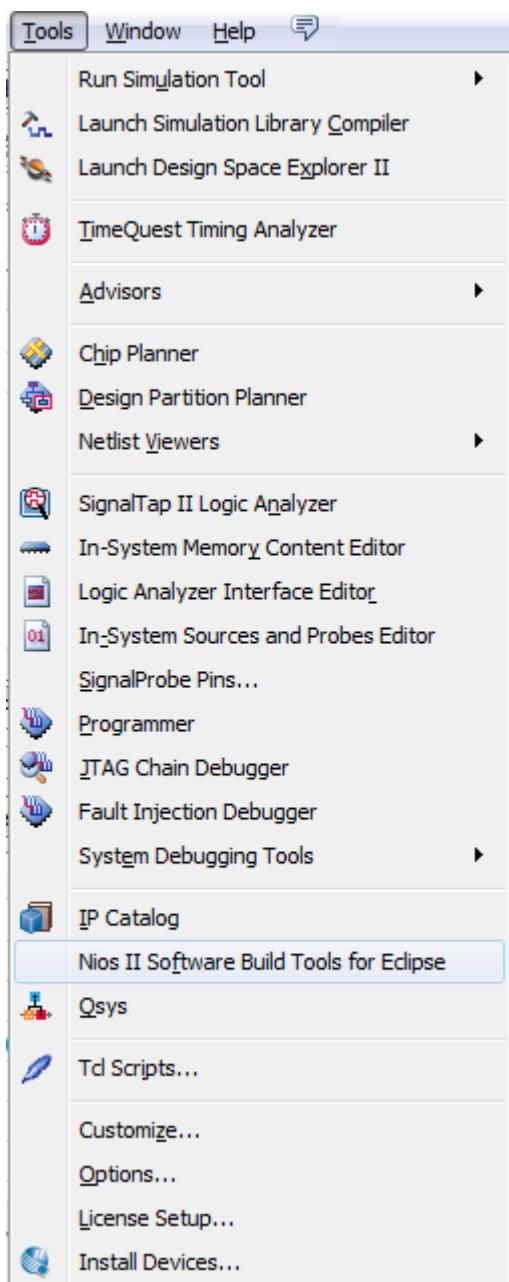


3.5 Create the Software Design

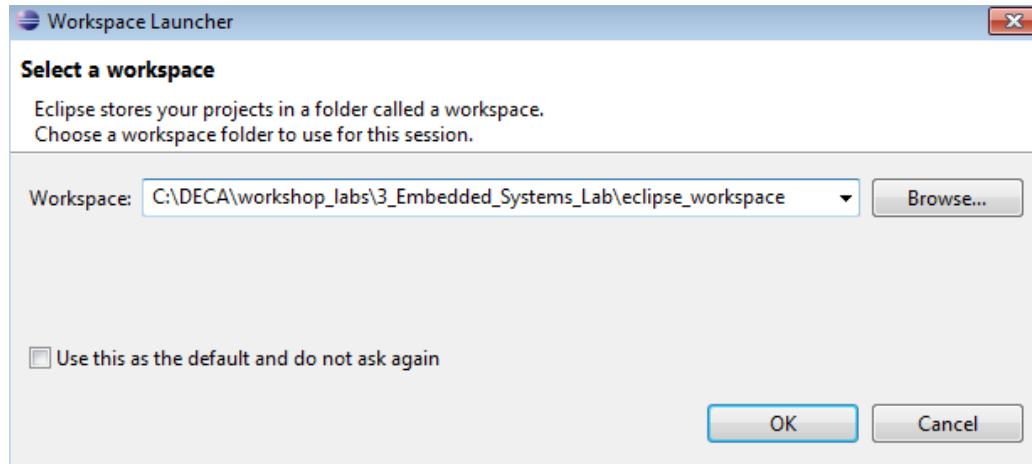
Overview: In this section, you will use the Nios II Software Build Tools (SBT) for Eclipse to quickly create a board support package (BSP) and a C software application to run on the Nios II processor you implemented in the last section.

3.5.1 Start Nios II Software Build Tools for Eclipse

3.5.1.1 From the main Quartus window, start the SBT from **Tools → Nios II Software Build Tools for Eclipse.**



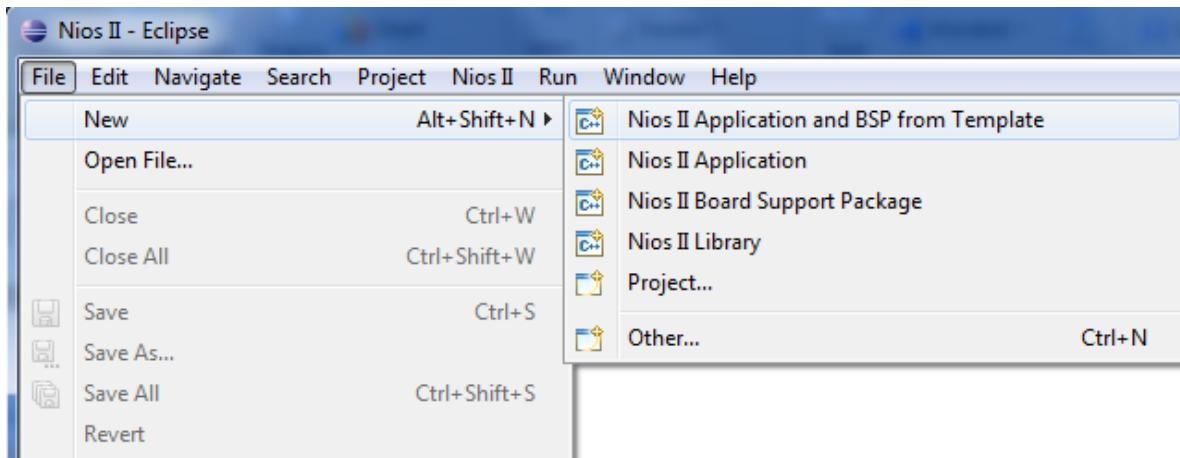
- 3.5.1.2 The Eclipse Workspace Launcher will open. Click "Browse..." and create a new folder titled **eclipse_workspace** in your lab directory to use as the software directory for the project. Click "OK".



3.5.2 Create a New Software Project

Now that Eclipse has a workspace, a new software application project and BSP can be created for your hardware system.

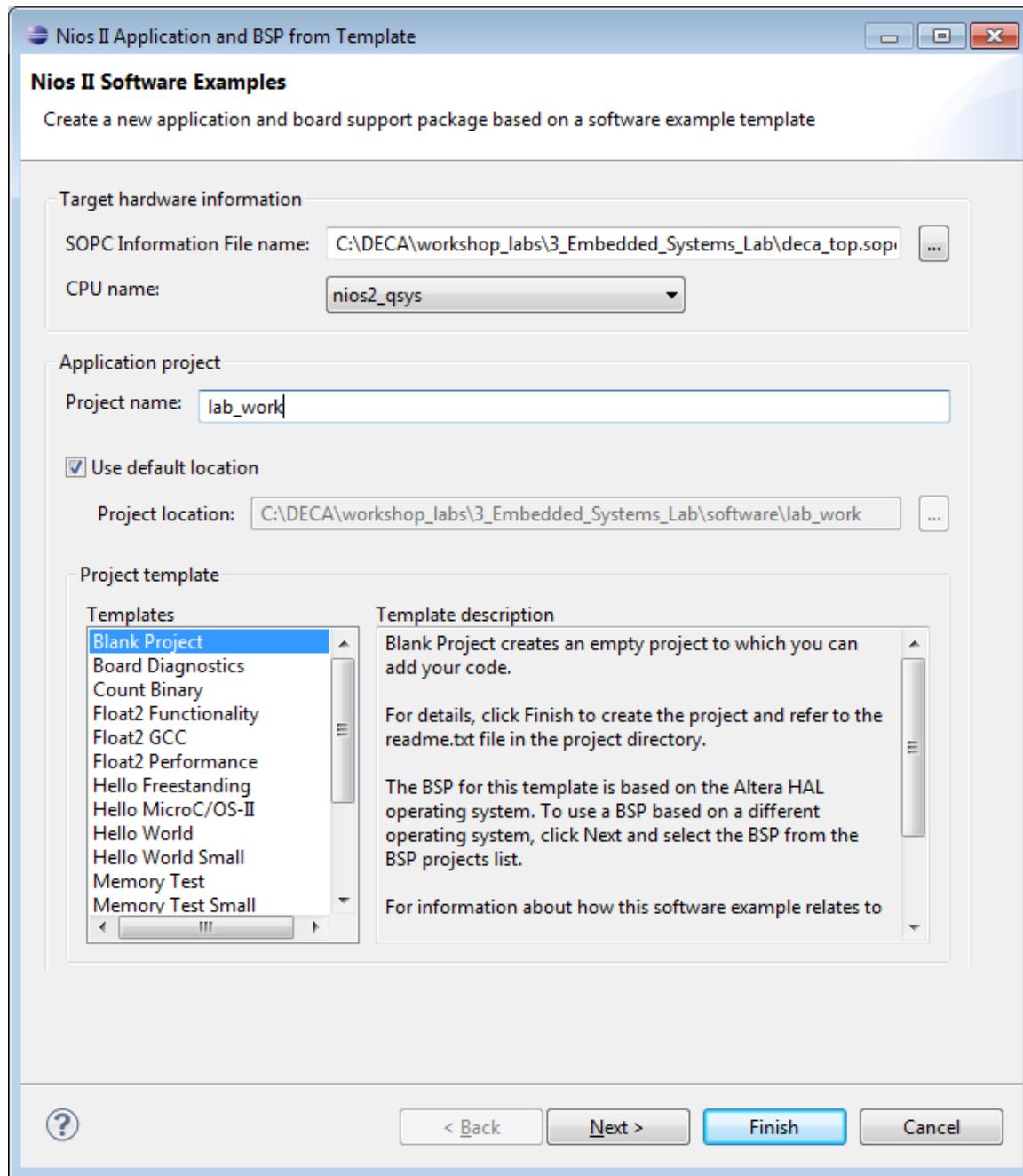
- 3.5.2.1 Once Eclipse opens to the workbench in the Nios II perspective, select **File → New → Nios II Application and BSP from Template** as shown below. This is an easy way to create a BSP and application together in a few easy steps.



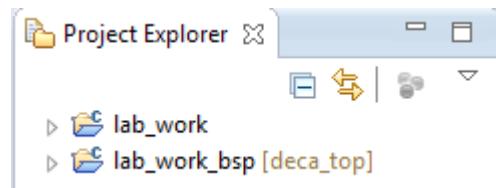
The BSP uses the Qsys-generated .sopcinfo file to import necessary settings from the hardware project to the software project so that your application can run on the Nios II processor. It allows Eclipse to build the system library drivers and generate system-specific macros for the custom Qsys system with the Nios II processor.

3.5.2.2 Click "..." to select `deca_top.sopcinfo` from your project directory and call the project `lab_work`. Make sure you select **Blank Project** from the Templates section as the software source files will be added in a later step. Make sure the settings match the screenshot below and select "Finish".

Note: your file path may be different than the one shown.



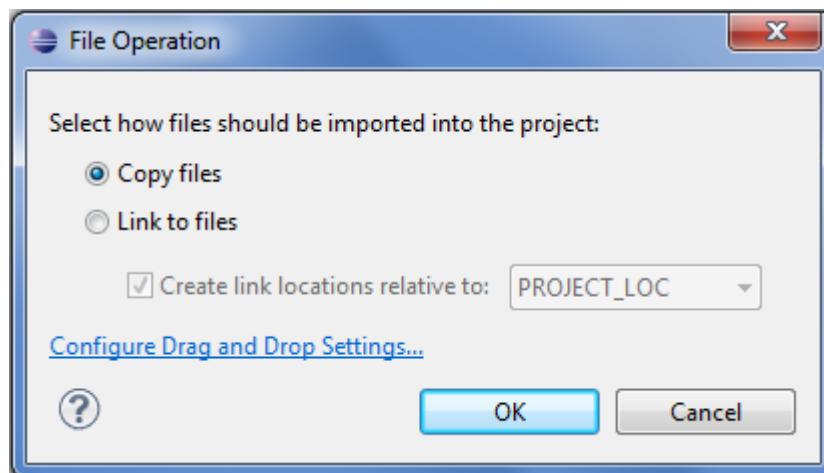
- 3.5.2.3 Eclipse will create two directories in the workspace; one for the application project and one for the BSP. The application directory (lab_work) is currently empty while the BSP directory (lab_work_bsp) contains software drivers, a system.h header file, initialization source code and other software infrastructure.



3.5.3 Add Source Code to the Project

The C source files and accompanying header files have been provided for you in this lab. All that needs to be done is to copy them to your workspace.

- 3.5.3.1 From Windows Explorer, navigate to your main project directory and into the folder <project_directory>/sw_src/. There you should find a number of C source and header files. We can start with the hello_led.c which you will copy to this project.
- 3.5.3.2 Select the hello_led.c and drag and drop it into the `lab_work` directory in Eclipse. Select the "Copy files" option in the pop-up and click "OK".



Since we are copying the files instead of linking to them, any changes that you would want to make to the source files need to be made to the versions inside the `lab_work` directory. Otherwise, the changes will not be compiled.

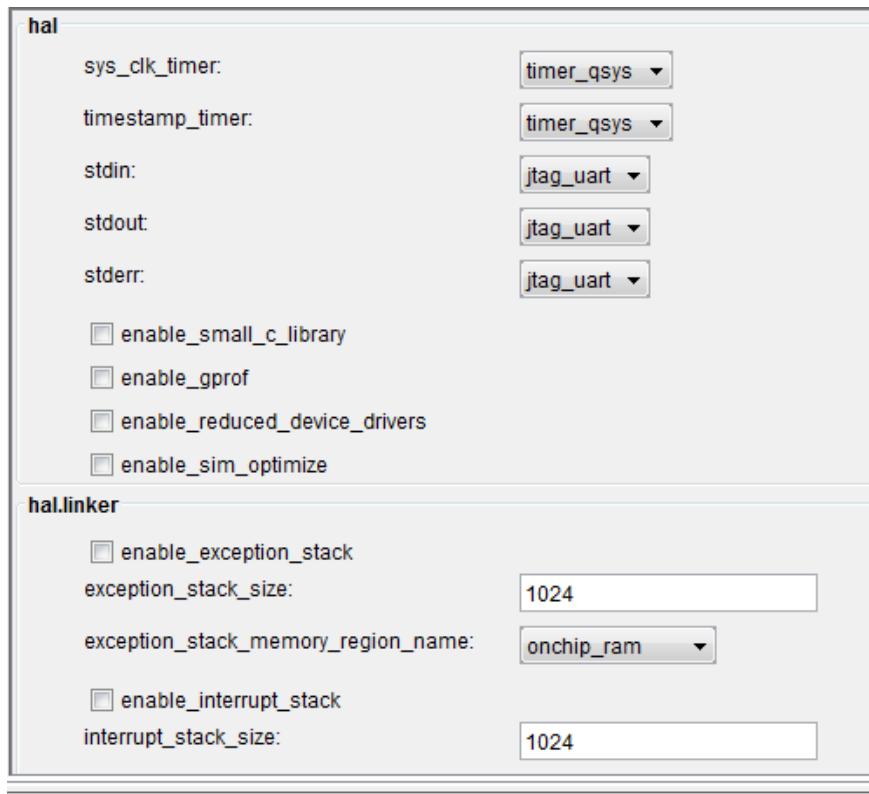
You should now see the new file appear under the `lab_work` project in the Project Explorer.

- 3.5.3.3 In some cases, the familiar windows ‘do not enter’ symbol appears indicating you cannot add files using the previous method. In this case, you can copy files using Windows Explorer. Copy files from the `sw_src` folder into the `lab_work` folder. In Eclipse, you need to right-click your `lab_work` project and click Refresh
- 3.5.3.4 Using this method, the C-source files added to the project may not automatically be added to the Makefile. You will notice a white dot or green dot beside the source files. For the C-files, you need to make this dot green by right-clicking each .c file and selecting: **Add to Nios II Build**

3.5.4 Configure the Board Support Package

The Board Support Package specifies the properties of the software system and needs to be configured for the software to execute correctly. These properties include setting the stdin, stdout and stderr interfaces, memory allocation for the heap and stack, drivers, and whether an operating system will be used.

- 3.5.4.1 Right-click on the `lab_work_bsp` project and select **Nios II → BSP Editor** from the pop-up menu.
- 3.5.4.2 The Nios II BSP Editor will open. In the Common settings under the Main tab, ensure that the settings are configured as follows.



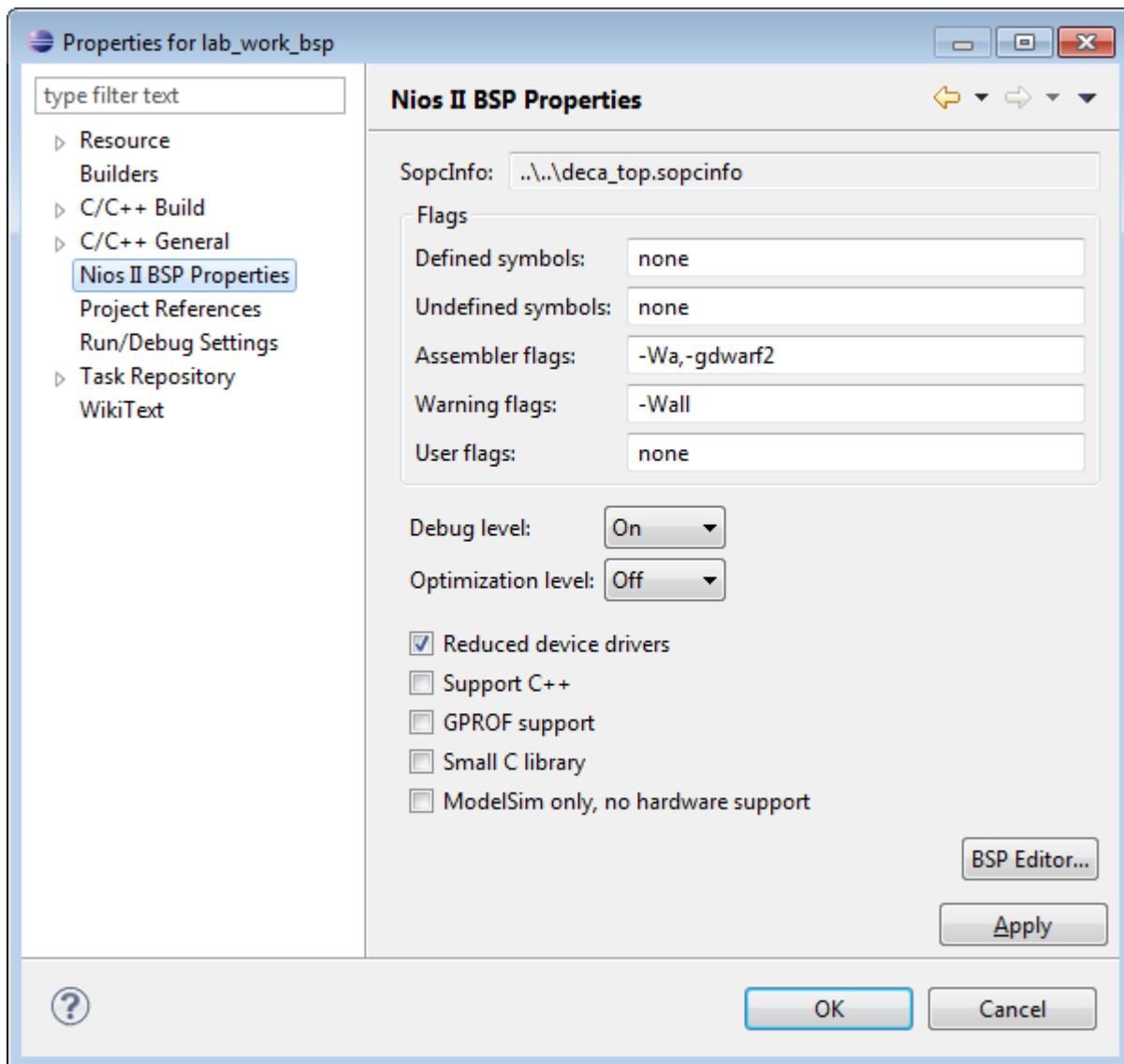
Notice that since there is no operating system in this lab, the stdout, stdin, and stderr messages are reported through the JTAG UART which you will be able to see in the Nios II Console in Eclipse. On-chip memory will be used for processor code storage, data storage, the exception, and interrupt stack.

Feel free to explore the BSP editor. The Drivers tab gives the user control over what drivers are built into the BSP. The Linker Script tab provides a mechanism to adjust what memory regions are utilized for certain purposes. We only

have one memory in this system but for systems with multiple memory locations (i.e. DDR3, flash, and on-chip ram), this is particularly useful.

- 3.5.4.3 Click the "Generate" button to update the BSP and select **File → Exit** to close it once the process is complete.
- 3.5.4.4 There are a few more BSP settings to edit. Right-click on the `lab_work_bsp` project and select Properties from the pop-up menu.
- 3.5.4.5 In the Properties window, select the Nios II BSP Properties tab. It may take a moment to load the settings.
- 3.5.4.6 To keep the software footprint small, enable the "Reduced device drivers" option. As there is no C++ code, disable the "Support C++" option.

The BSP Properties should match the following.

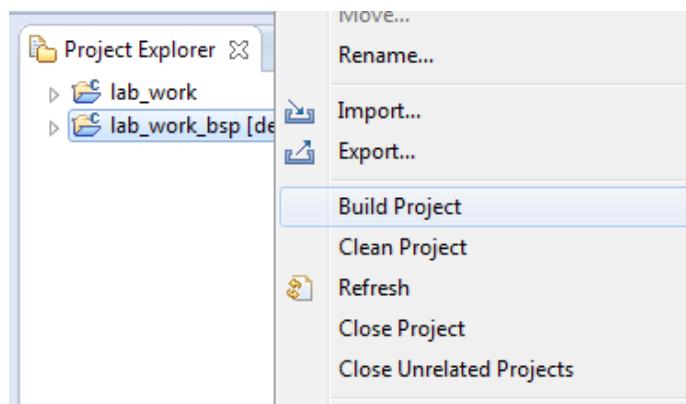


3.5.4.7 Select "Apply" and click "OK" to exit the Properties window.

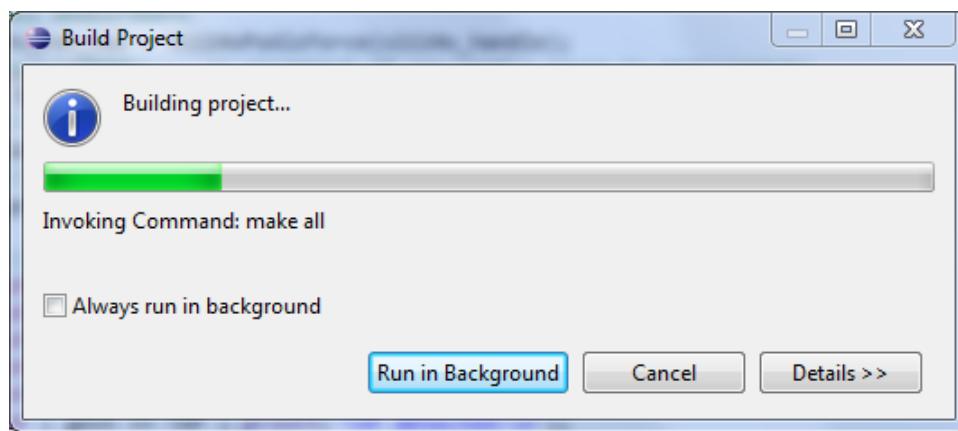
3.5.5 Build the Software Project

With all of the appropriate settings configured, you can now build the BSP and software project using the next two steps to produce an executable and linked format (.elf) file to run on the DECA board.

3.5.5.1 Right-click on the `lab_work_bsp` project and select Build Project from the pop-up menu to build the BSP.



You can have the build process run in the background by from the pop-up window if you wish. You can observe the process commands in the Console window.



3.5.5.2 Repeat this procedure for the application. Right-click on the `lab_work` project and select Build Project from the pop-up menu.

```

CDT Build Console [lab_work]
Info:          8b Kbytes tree for stack + heap.
Info: Creating lab_work.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab_work.elf >lab_work.objdump
[lab_work build complete]

09:20:17 Build Finished (took 7s.639ms)

```

3.5.6 Run the Application on the DECA board

Overview: Now that you have an executable, you can download the application to the on-chip memory in the MAX10 and the Nios II processor will execute it. The output will be the lower LED's shifting.

3.5.7 Download the Executable to the DECA

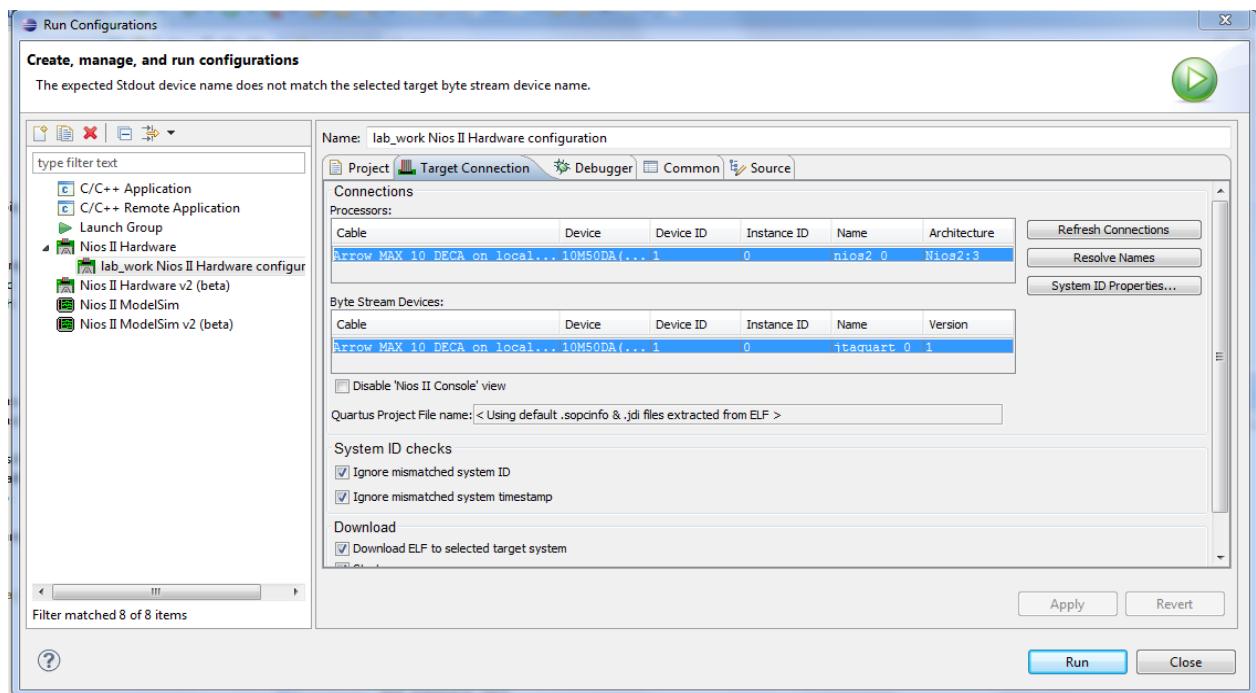
First, a target configuration will need to be established with the DECA so that Eclipse can download the code and communicate with the board.

3.5.7.1 Right-click on the `lab_work` software project and select **Run As → Nios II Hardware**.

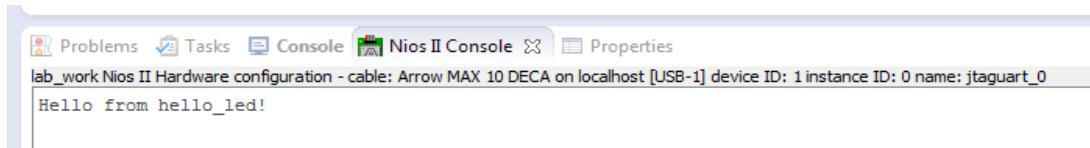
This will rebuild the software project to create an up-to-date executable and download the code into memory on the DECA. The debugger then resets the Nios II and it begins executing the code.



Note: If a Run Configuration dialogue appears, you may need to click the Target Connection tab and scroll to the right. Click "Refresh Connections" and the appropriate connection to the DECA should appear as below. Then click "Run".



3.5.7.2 After a few seconds, the Nios II Console should open at the bottom of Eclipse:



Once you have the `hello_led` application running on the Nios II processor, can try out other applications available in the source folder including humidity, temperature, capsense etc. to understand the full potential of the DECA platform.

There are two methods to try out other applications. Here are some optional steps:

- 3.5.7.3 Update the existing **lab_work** folder c-source files by removing **hello_led.c** and replacing it with the source files found in one of the other applications located in the original **sw_src** folder, or
- 3.5.7.4 Alternatively, create a new software project and repeat the steps used to create the original software project but choose one of the other applications found in the **sw_src** folder.

**CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED THE
EMBEDDED SYSTEMS LAB!**

Gesture Sensor Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 4. GESTURE SENSOR LAB	144
4.1 Getting Started	144
4.2 Examine the System Design	145
4.2.1 Examine the System Tool Flow	145
4.2.2 Examine the DECA Development Platform	146
4.3 Set Up the Quartus II Project	147
4.3.1 Extract the lab files.....	148
4.3.2 Create a new Quartus II Project.....	148
4.4 Build the Hardware Design	150
4.4.1 Launch Qsys	150
4.4.2 Configure the Clock	151
4.4.3 Add a Nios II Processor	152
4.4.4 Add On-Chip Memory	154
4.4.5 Add the JTAG UART Peripheral	155
4.4.6 Add a PLL	157
4.4.7 Add a Timer.....	161
4.4.8 Add a System ID Peripheral.....	162
4.4.9 Add an Interrupt Pin	163
4.4.10 Add an I2C Peripheral.....	165
4.4.11 Connect the Qsys system and remove errors	167
4.4.12 Set Interrupt Priorities	170
4.4.13 Define the Nios II Reset and Exception Vectors.....	171
4.4.14 Resolve Memory Addresses	172
4.4.15 Check the full system	173
4.4.16 Generate the Qsys System.....	175
4.4.17 Add the Qsys System to the Quartus Project	176
4.4.18 Modify the Top-Level Design File	177
4.4.19 Import Pin Assignments	179
4.4.20 Compile the Quartus II project	179
4.4.21 Download the Configuration File to DECA.....	182
4.5 Create the Software Design	185
4.5.1 Start Nios II Software Build Tools for Eclipse	186
4.5.2 Create a New Software Project.....	187
4.5.3 Add Source Code to the Project	189
4.5.4 Configure the Board Support Package	190
4.5.5 Configure the Application Project.....	194
4.5.6 Build the Software Project.....	195
4.6 Run the Application on the DECA board.....	197
4.6.1 Download the Executable to the DECA	197

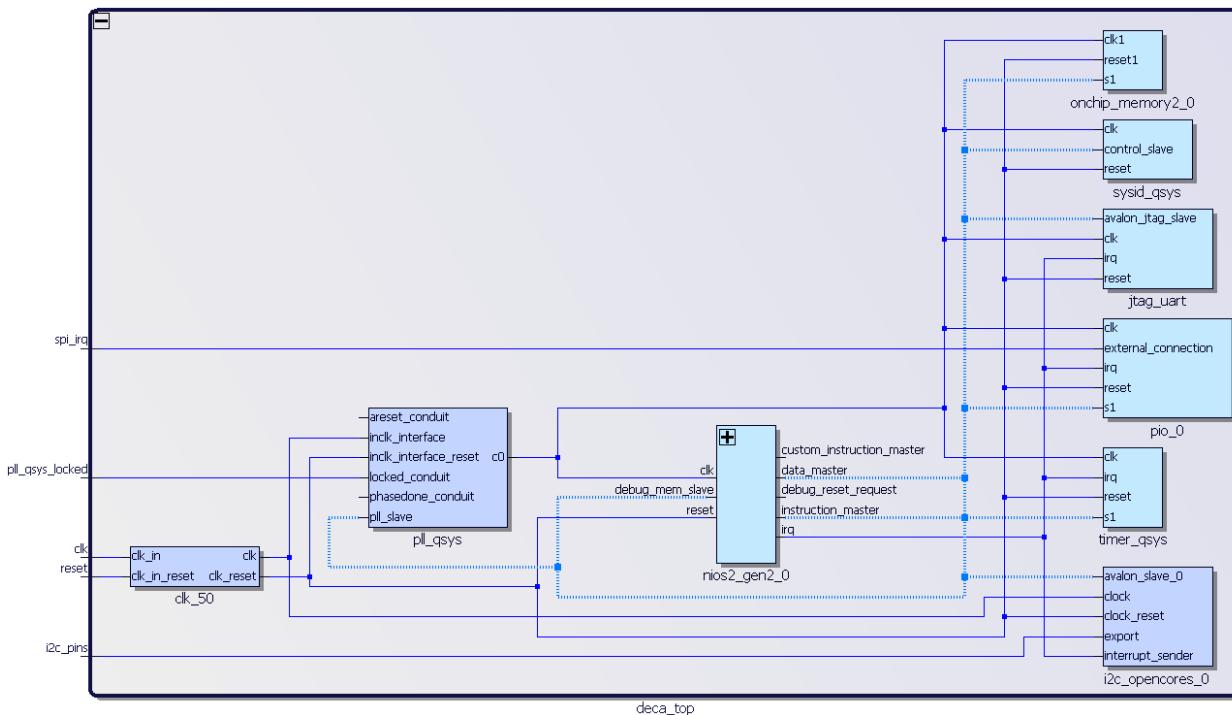
LAB 4. GESTURE SENSOR LAB

Overview: In this lab, you will create a Qsys design to interface with the Silicon Labs Si1143 Gesture Sensor IC. After configuring your DECA board with the design, you will run a C program on the embedded Nios II soft processor to decode hand gestures made over the Si1143

This lab will provide a working knowledge of a basic FPGA design flow including an embedded processing component. At the end of this lab you should be able to:

- Build a complete FPGA design including a Nios II soft processor
- Configure the target FPGA using the Quartus II Programmer
- Create and run a software application on the Nios II processor in the target FPGA

Below is a schematic view of the embedded system you will create in this lab.



4.1 Getting Started

Overview: The first objective is to ensure that you have all of the necessary hardware items and software installed so that the lab can be completed successfully.

Below is a list of items required to complete this lab:

- Arrow DECA Evaluation Kit
- USB cable
- Lab files
- Quartus II 15.0 Design Software
- Personal computer or laptop running Windows 7 with at least an Intel i3 core (or equivalent), 4 GB of RAM, and 12 GB of free hard disk space
- A desire to learn

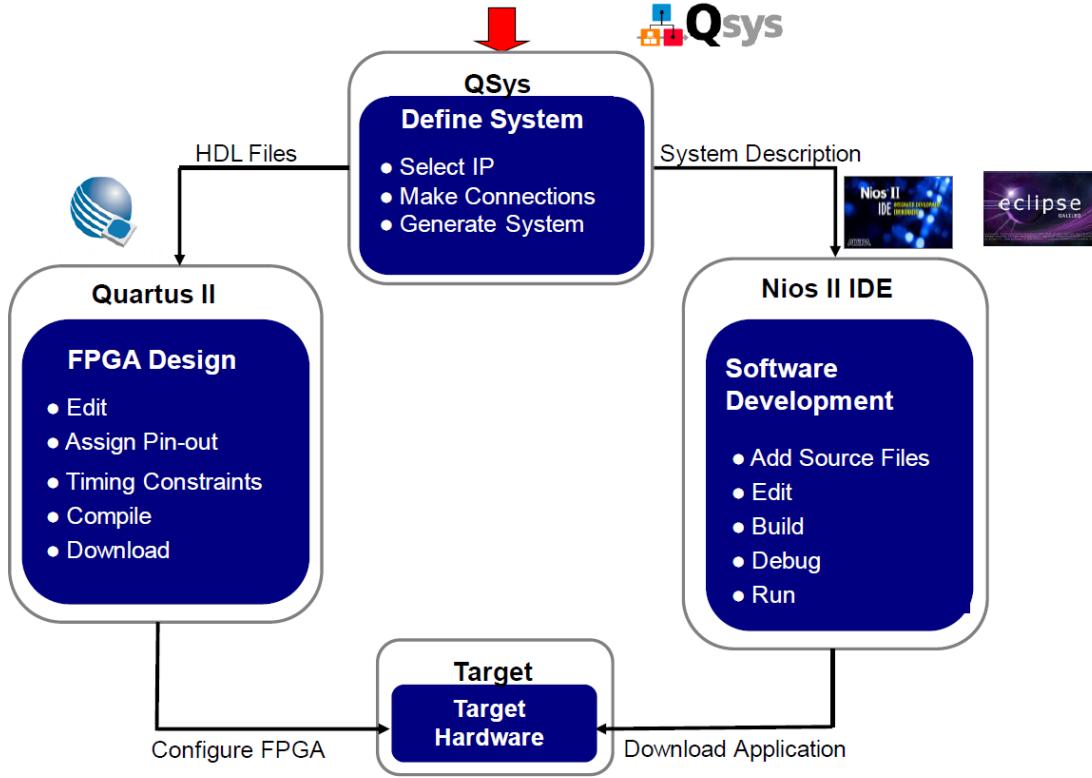
If you are missing one of these items, please let your instructor know. Instructions for how to download Quartus can be found in the Appendix.

4.2 Examine the System Design

Overview: In this section, you will examine the design flow used in modern Altera FGPA designs.

4.2.1 Examine the System Tool Flow

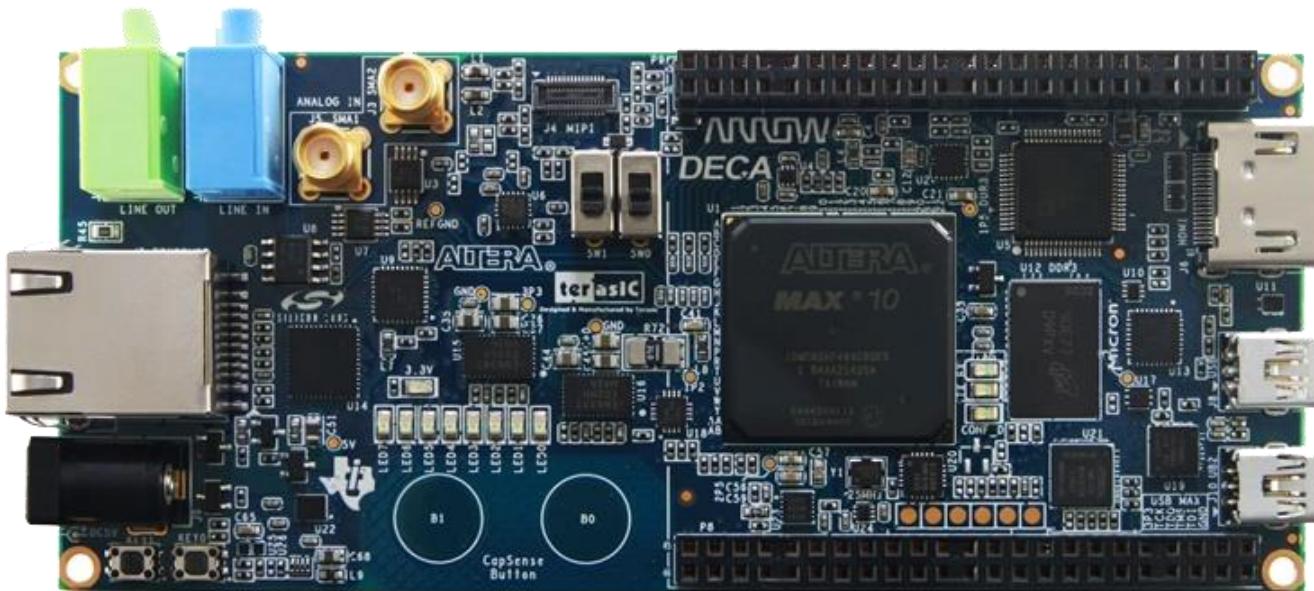
Developing software for an Altera system on a programmable chip requires an understanding of the design flow between the Qsys system integration tool and the Nios II Embedded Development Suite (EDS). Typically, design requirements begin with requirements and become inputs to system definitions. System definition is the first step in the design flow process. For this workshop, the design will be built and then the FPGA image will be downloaded into the dev board. The objective of the module is to review the development tools that will be used.



The above diagram shows the typical design flow for the system design. The system definition is done with Qsys. The Nios II IDE uses the system description to create a new project for the software application. The output of the FPGA design is a FPGA image that is used to configure the FPGA. The output of the software flow is an executable which runs on the Nios II processor.

4.2.2 Examine the DECA Development Platform

Examine the components on the DECA board. The development board provides a full system with the MAX10 at its heart including external memory, LEDs, sensors, buttons, and power supplies.



There are many components on the DECA board that can be used including the LEDs, capacitive push buttons, HDMI port, a MIPI interface and a full suite of sensors.

The completed system will include many components including the Nios II soft processor, JTAG UART, on-chip memory, PLLs, and an I2C interface.

The system that will be created in Qsys will use a library of re-usable IP blocks. Interconnect between components is automatically connected by Qsys. The system interconnect manages the dynamics bus-width matching, interrupt priorities, arbitration and address mapping. The processor that is used, Nios II, is a full featured processor that can run operating systems such as Linux.

The following modules will guide you through the process of building a basic embedded system.

4.3 Set Up the Quartus II Project

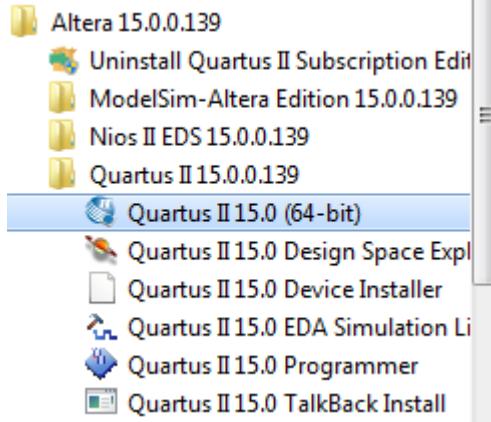
In this module, you will create a Quartus II project for your gesture sensor design.

4.3.1 Extract the lab files

- 4.3.1.1 Create a new directory on your computer and ensure that there are no spaces in the directory path. This will cause problems with the software tools. Call this new directory `gesture_lab`.
- 4.3.1.2 Extract the workshop files into this new directory.

4.3.2 Create a new Quartus II Project

- 4.3.2.1 Launch Quartus II 15.0 (64-bit) from the Start menu.



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

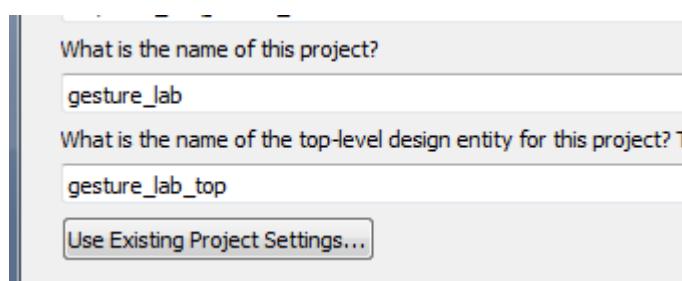
- 4.3.2.3 Configure the New Project Wizard directory, name, and top-level entity information.

Click on the button and browse to: C:\DECA\workshop_labs\4_Gesture_Sensor_Lab

Specify the name of the project: `gesture_lab`

Specify the name of the top level entity: `gesture_lab_top`

(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.)



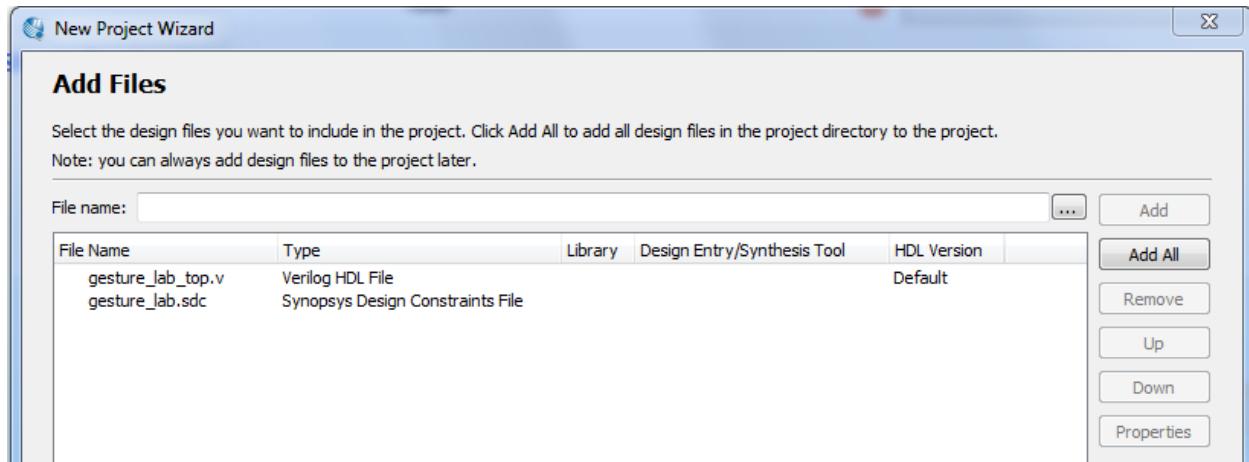
Click

4.3.2.4 On the Project Type page, select "Empty Project" and click **Next >**

4.3.2.5 Add source files to the project

Click on the  button and browse to the project's top-level directory (generally the default folder) where you will locate the two provided design files: `gesture_lab_top.v`, and `gesture_lab.sdc` and select both files and add them to the files listing. Note: may need to change the file type filter to "All Files (*.*)".

Don't forget to click the Add button to add the files to the project.

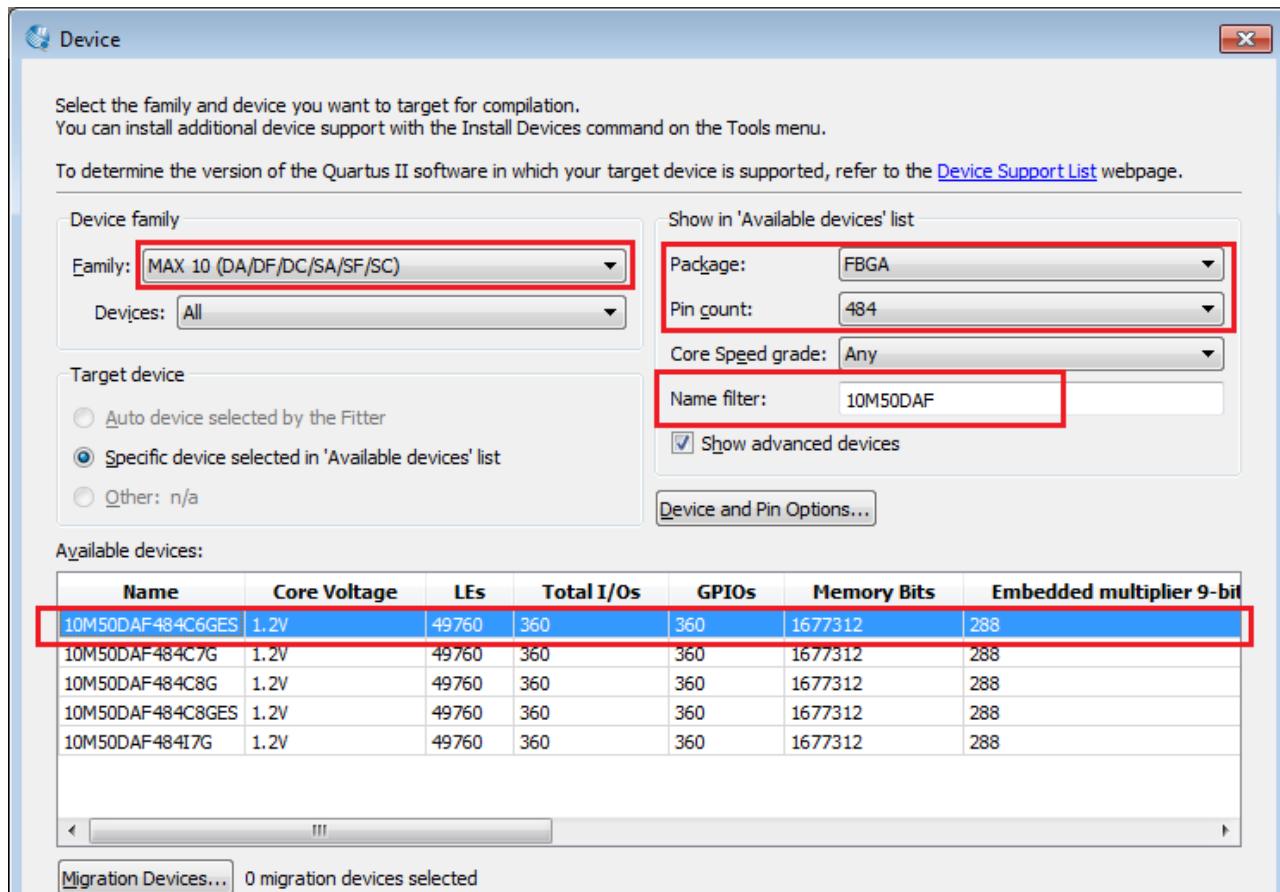


Click **Next >**

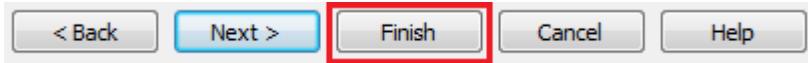
4.3.2.6 Specify Family and Device Settings

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**.



4.3.2.7 Click Finish to close the New Project Wizard



4.4 Build the Hardware Design

Overview: In this module, you will use the Qsys system integration tool to design your hardware system. You will add standard and custom components, make interface connections, assign clocks, set arbitration levels for interrupts, and generate the HDL for the system.

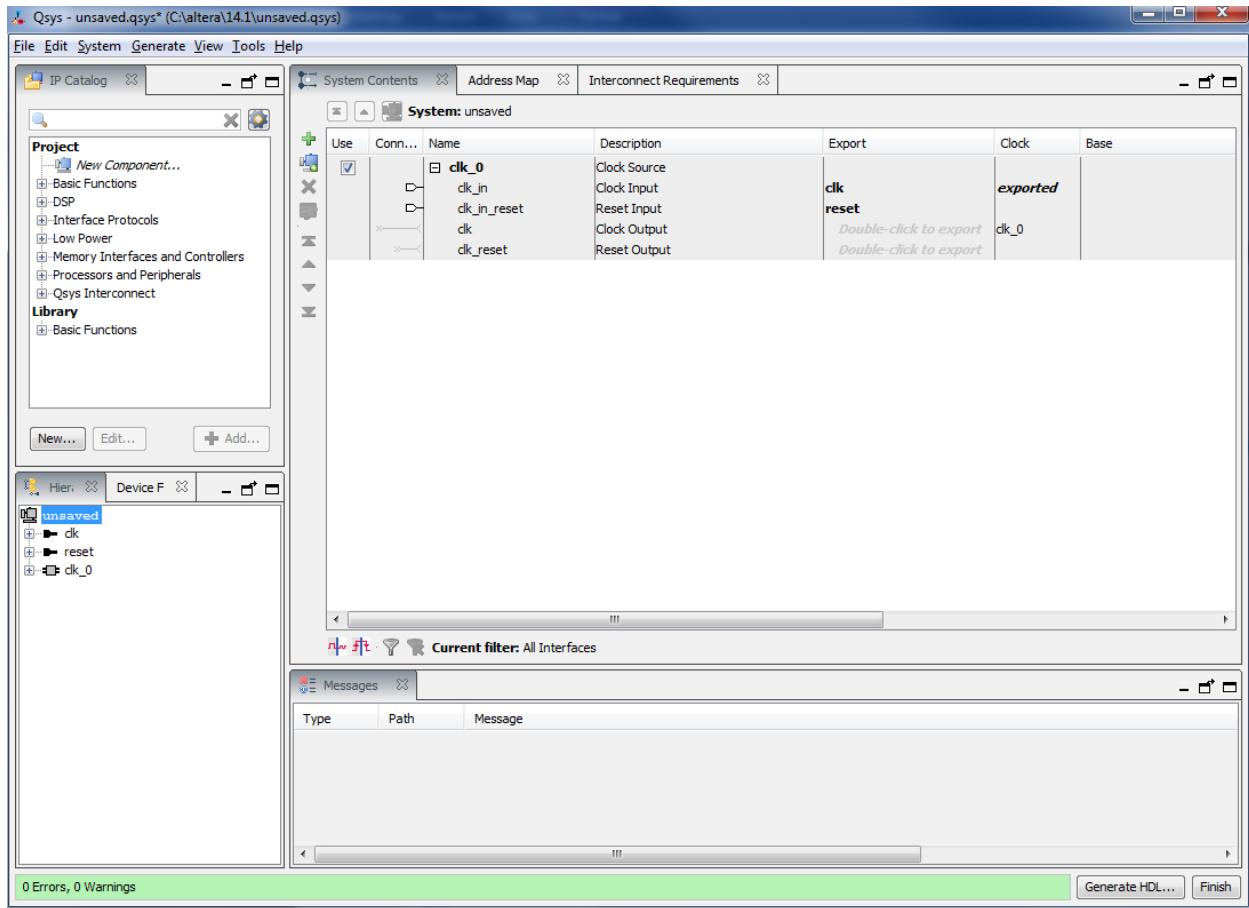
4.4.1 Launch Qsys

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...).

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

4.4.1.2 In the new Qsys window, you should see a single Clock Source component named `clk_0` in the System Contents tab. This tab shows all of the components currently in your system.



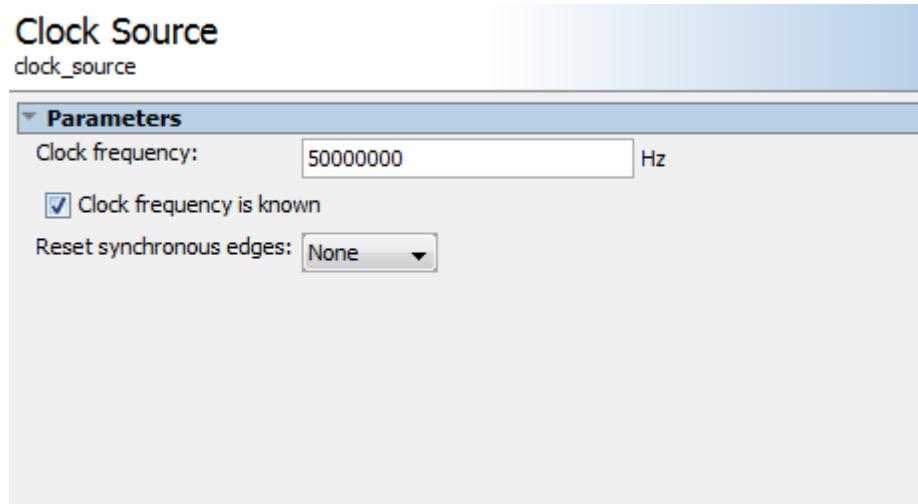
4.4.2 Configure the Clock

In this section, you will configure the clock input to your Qsys system. This clock will be fed to a PLL to provide additional frequencies.

4.4.2.1 Double click on the Clock Source component named clk_0. This will open the Parameter editor window which should look very familiar to the traditional megawizard windows.

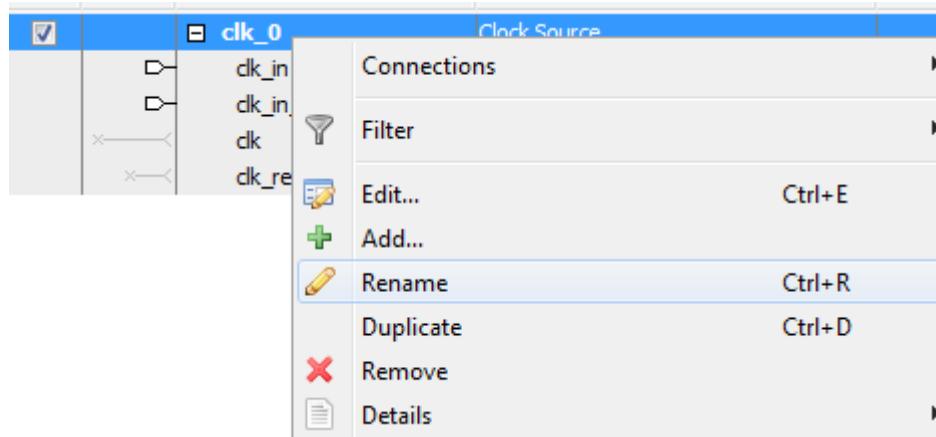
4.4.2.2 Change the clock frequency parameter to 50 **MHz** (50000000 Hz).

Ensure that the "Clock frequency is known" parameter is enabled.



Click the "X" on the Parameter tab to close the parameter window.

4.4.2.3 To rename the clock, right-click on the clock and select "Rename" or press **CTRL+R**. Rename the clock to **clk_50** and press Enter.

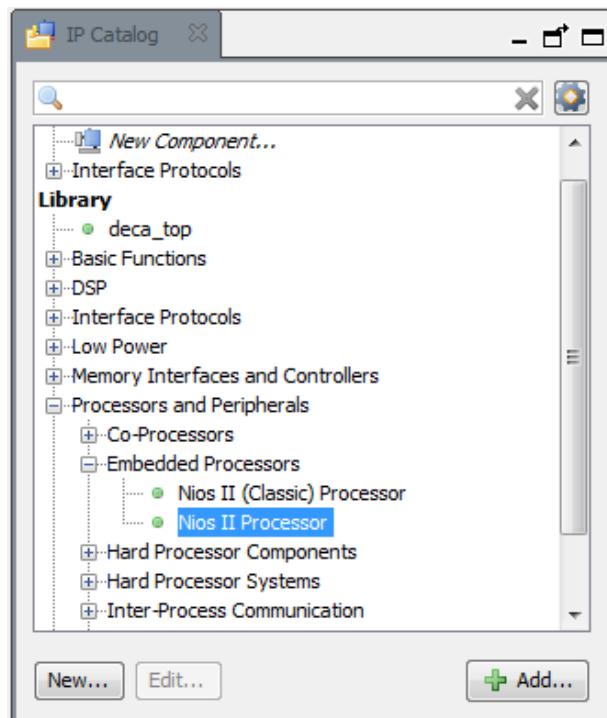


4.4.2.4 Save the Qsys system. Click **File → Save As** and name your Qsys system **deca_top.qsys**. This is the entity name by which you will be instantiating your Qsys system into your top-level file. Click Save.

4.4.3 Add a Nios II Processor

Since we are using a software algorithm to decode the raw data coming from the Si1143 Gesture Sensor IC, we need to add a Nios II Soft Processor to our system on which the software can run. We will connect the Nios II processor to a number of peripherals later in this module.

- 4.4.3.1 From the IP Catalog panel on the left side of the Qsys window, expand the menus for **Processors and Peripherals** → **Embedded Processors** and select the Nios II Processor.





Note that MAX10 devices do not support the Nios II (Classic) Processor. However, all code developed on the classic version is fully forward compatible.

- 4.4.3.2 Double click on the name or click "Add..." to add the component to the system. The Nios II parameter editor window will open.

- 4.4.3.3 In the Main tab, ensure that the "Nios II/e" option is selected.

- 4.4.3.4 The settings in the Vectors tab will be set in a later step so skip that for now.

Note that until these settings are applied, the following two errors in the Qsys window are expected.

Error: nios2_gen2_0: Reset slave is not specified. Please select the reset slave.

Error: nios2_gen2_0: Exception slave is not specified. Please select the exception slave

- 4.4.3.5 The settings in the other tabs are left as their defaults but feel free to explore the parameter editor and see what settings can be applied to the Nios II. Click Finish.

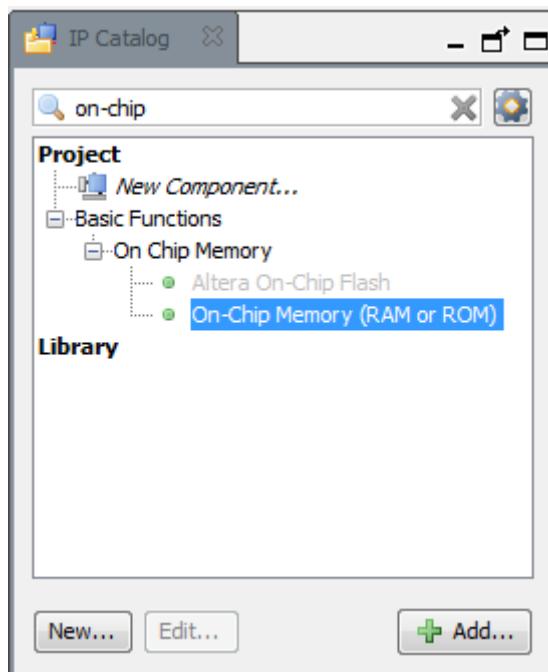
Note that there will be errors related to clocks as well. This will be resolved in a few steps.

- 4.4.3.6 Rename the Nios II to **nios2_qsys**.

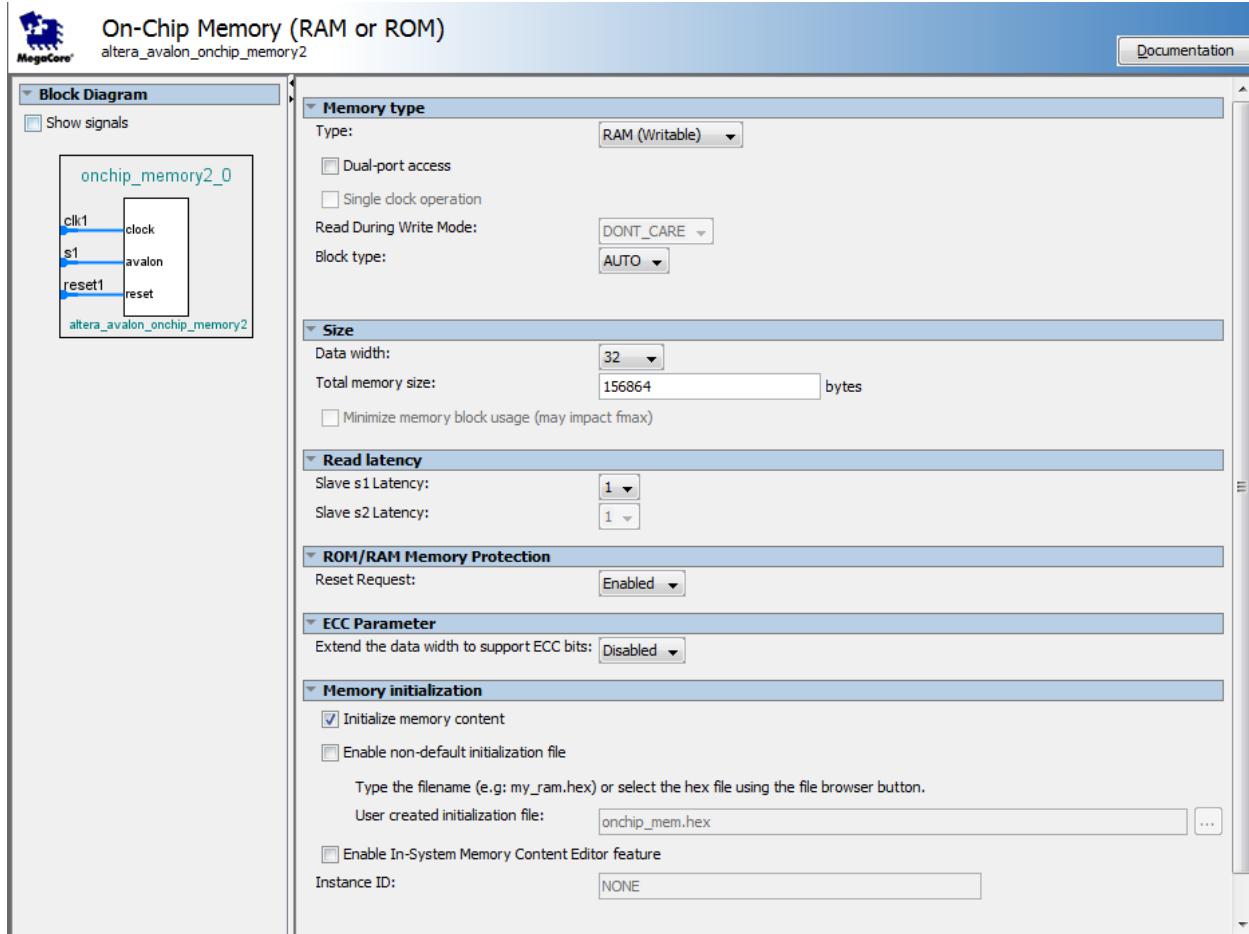
4.4.4 Add On-Chip Memory

Altera FPGAs provide internal on-chip memory blocks that can be used to build up an internal RAM (or ROM) block of memory. In this lab, this provides the Nios II with access to very low-latency high speed memory for executable code and variable storage.

- 4.4.4.1 In the IP Catalog panel, type "on-chip" in the search bar. You should see the On-Chip Memory (RAM or ROM) appear under **Basic Functions → On Chip Memory**.



- 4.4.4.2 Double click the component or select it and click "Add..." to add it to the system. The On-Chip Memory parameter editor will open.
- 4.4.4.3 Change the total memory size parameter to **156864** bytes. This allows enough storage for the application code and scratch pad memory for the processor.

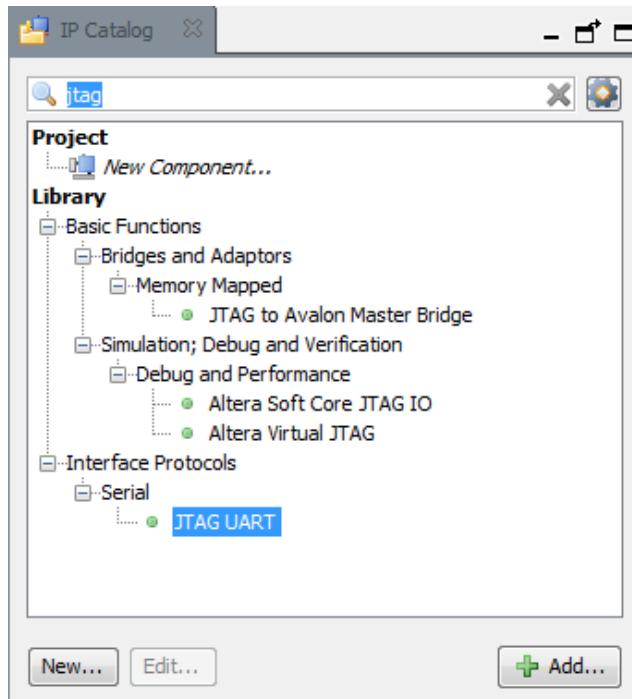


- 4.4.4.4 Click Finish to add the component to the system. Don't worry about the errors; they will be corrected later in the lab.
- 4.4.4.5 Rename the component to **onchip_ram**.

4.4.5 Add the JTAG UART Peripheral

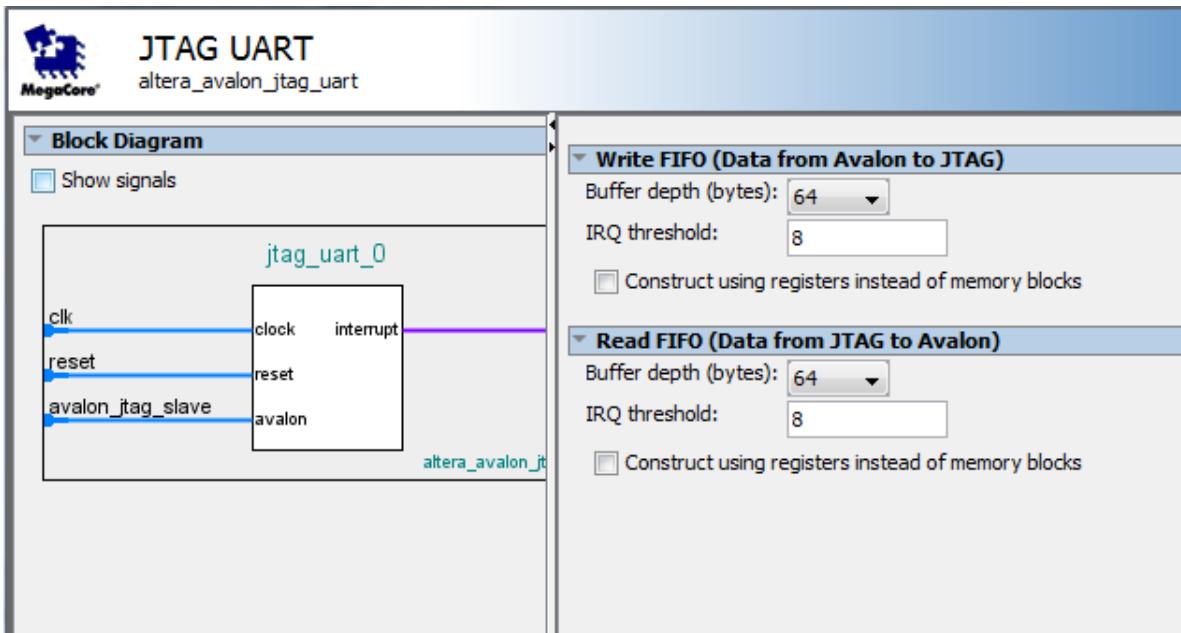
Many software developers like to have access to a debug serial port from the target to leverage `printf` debugging, input control, log status information, etc. The JTAG UART connects to Nios II processor to the debugger console in the Nios II IDE for easy debug and development using a console interface.

- 4.4.5.1 In the IP Catalog search bar, type **JTAG UART**. You should see the JTAG UART peripheral appear under Interface Protocols → Serial.



- 4.4.5.2 Double-click the component or select it and click "Add..." to add it to the system. The JTAG UART parameter editor will open.

- 4.4.5.3 Verify that the parameters for the Write and Read FIFO as the same as below.



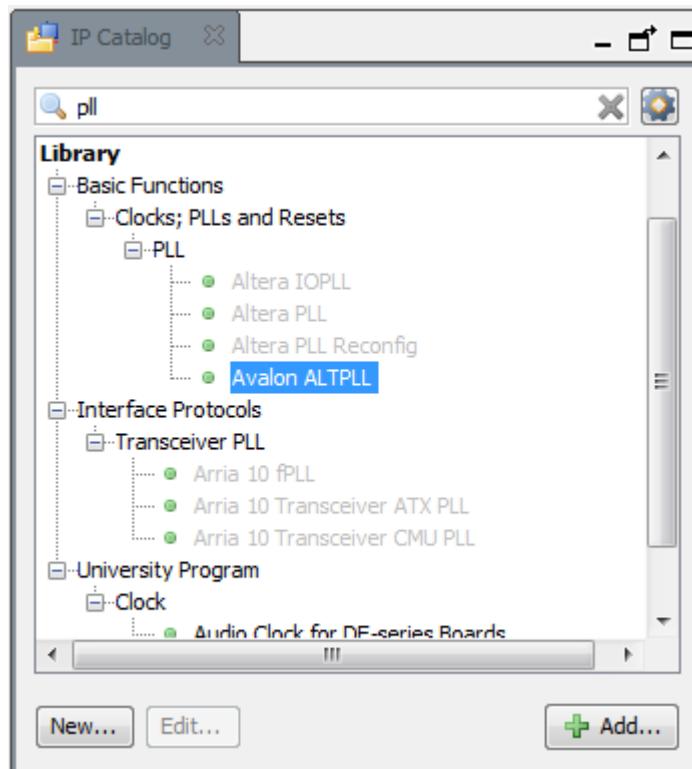
4.4.5.4 Click Finish to add the component to the system. Don't worry about the errors; they will be addressed later.

4.4.5.5 Rename the component `jtag_uart`.

4.4.6 Add a PLL

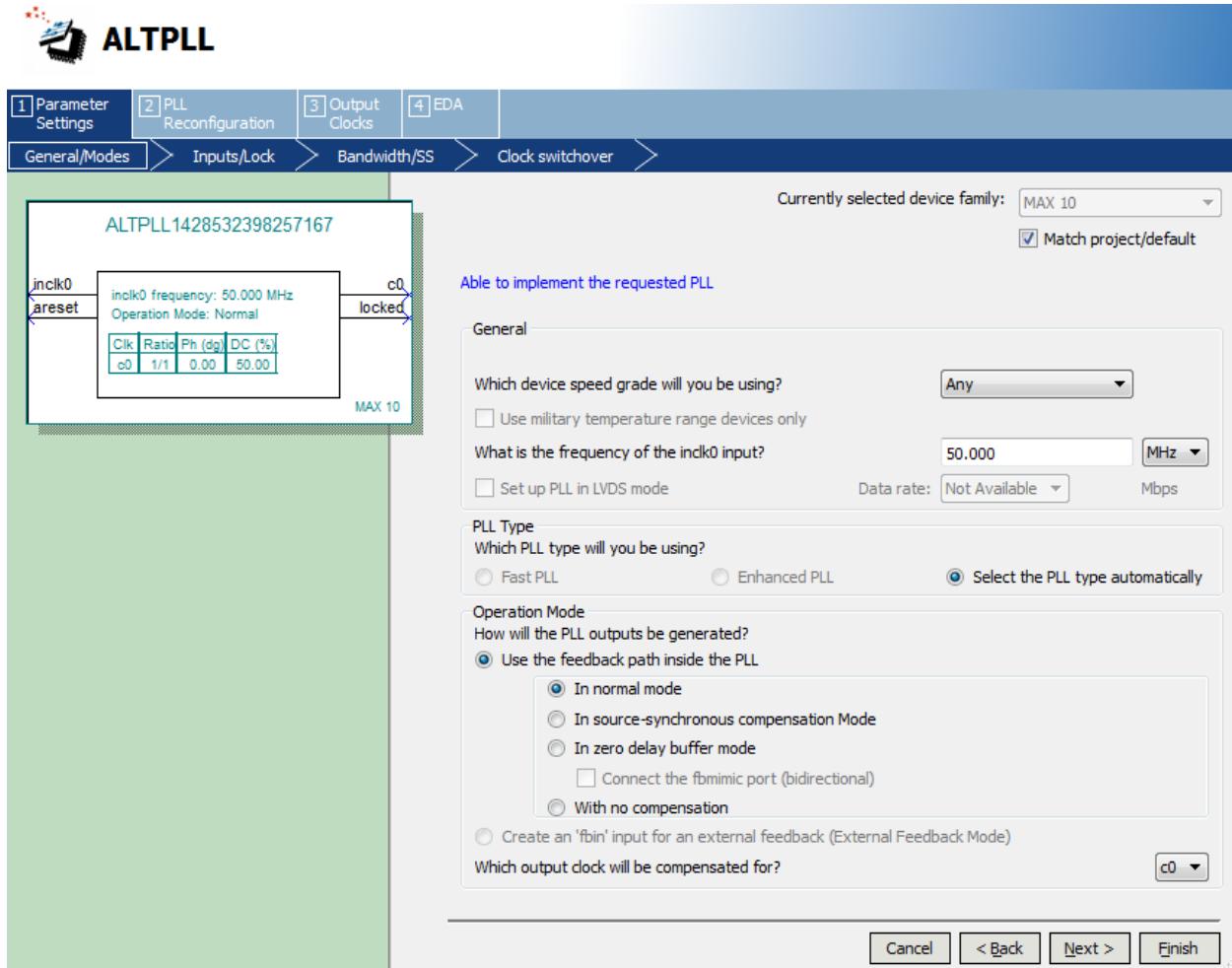
A PLL will generate a system clock for the Qsys system.

4.4.6.1 In the IP Catalog search bar, type `PLL`. The Avalon ALTPPLL peripheral will appear under **Basic Functions** → **Clocks; PLLs and Resets**.

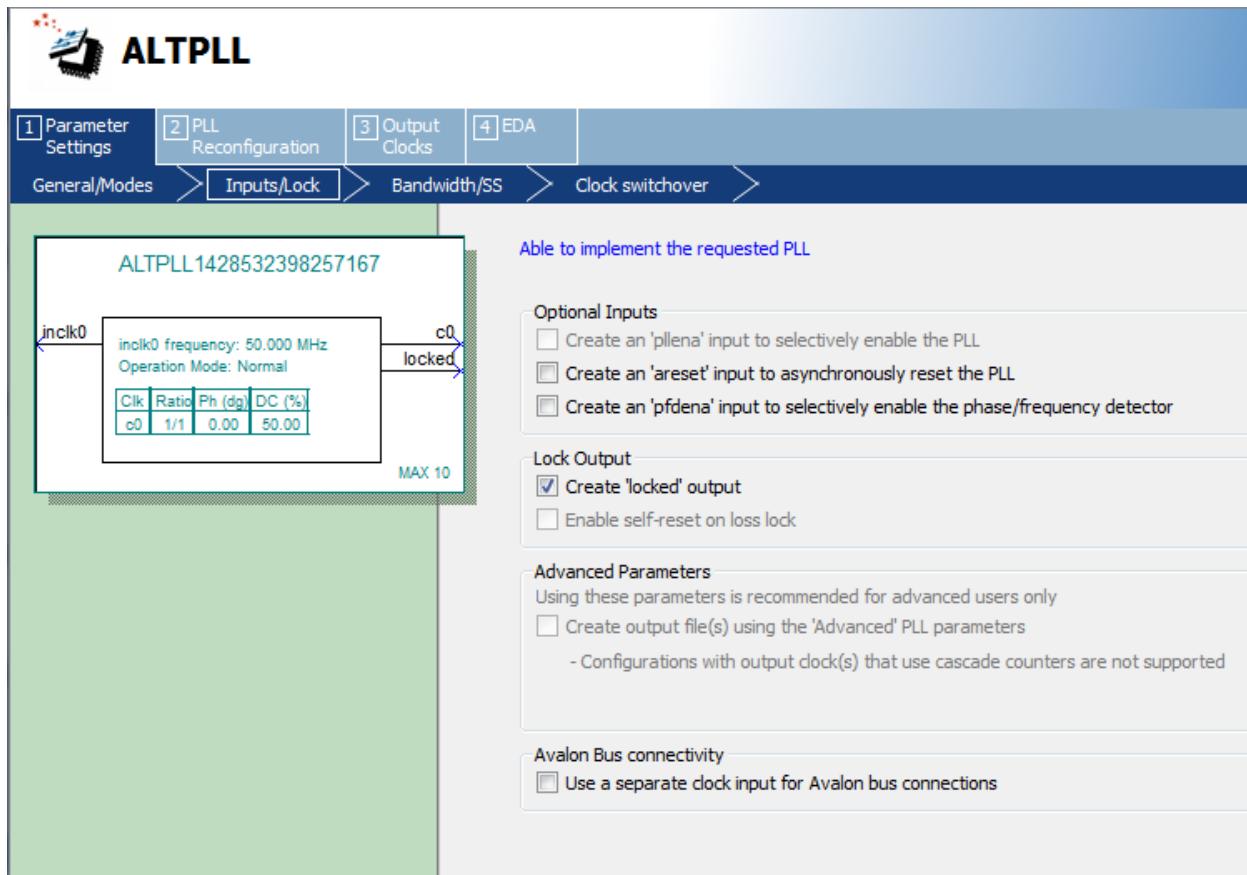


4.4.6.2 Double-click the component or select it and click "Add..." to add it to the system. The ALTPPLL parameter editor will open.

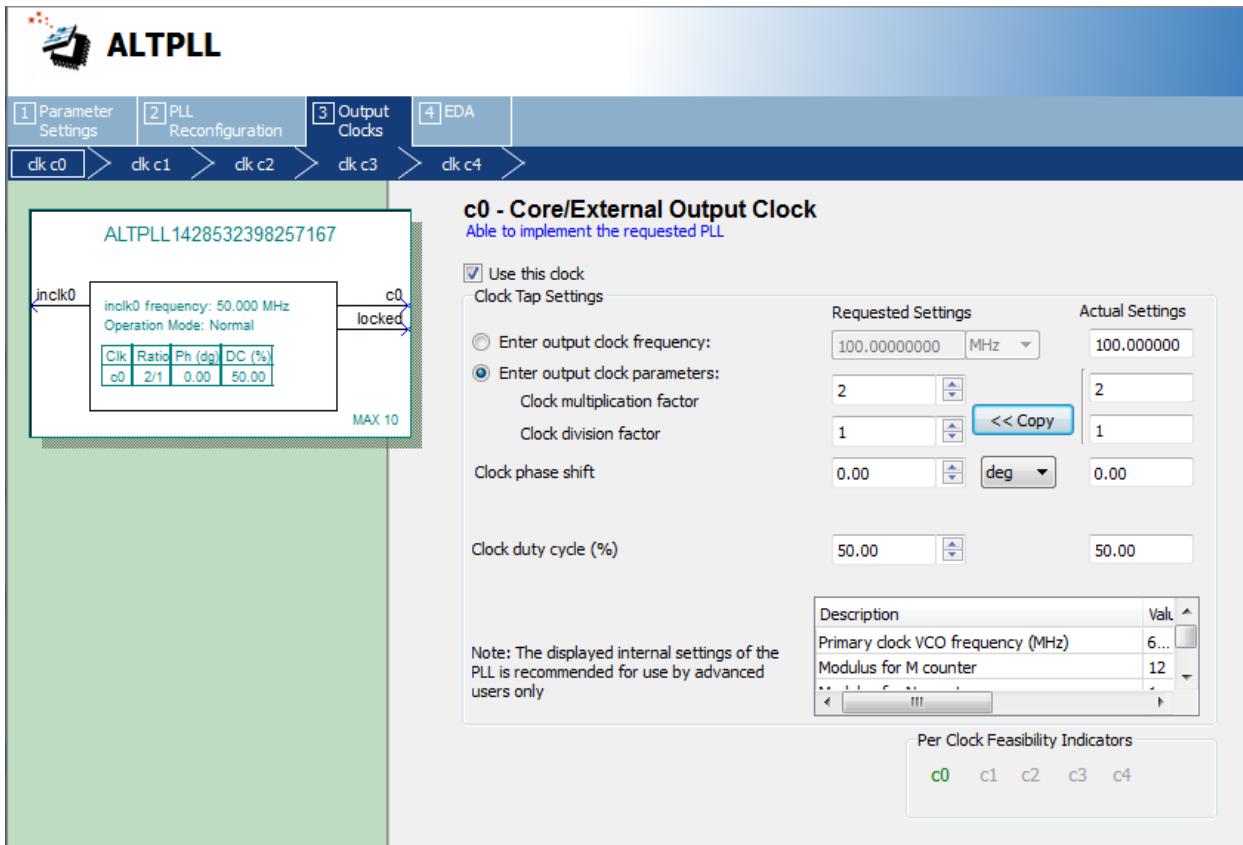
4.4.6.3 In the **Parameter Settings → General/Modes** tab, change the input clock frequency to 50 MHz (this is the maximum frequency of the oscillator on the DECA board).



4.4.6.4 On the **Parameter Settings → Inputs/Lock** tab, ensure that only the "Create 'locked' output" box is checked.



- 4.4.6.5 On the Output Clocks/clk c0 tab, make sure the clock is enabled and ensure that the output frequency is **100 MHz** either by specifying the multiplication and division factors or the actual frequency. Make sure no other clocks are enabled as they are not needed.



- 4.4.6.6 Click Finish. The errors will be addressed later.

- 4.4.6.7 Rename the PLL as **pll_qsys**.

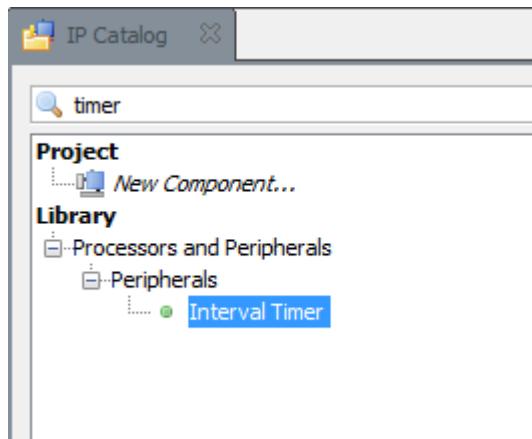
- 4.4.6.8 Since the locked output was enabled, added as an input port to the Qsys system so that it can be connected to an external pin. This will enable a port to connect the output to user logic in the top-level design. Double-click the export column next to the "external_connection" signal and type **pll_qsys_locked**. The PLL component should now look like:



4.4.7 Add a Timer

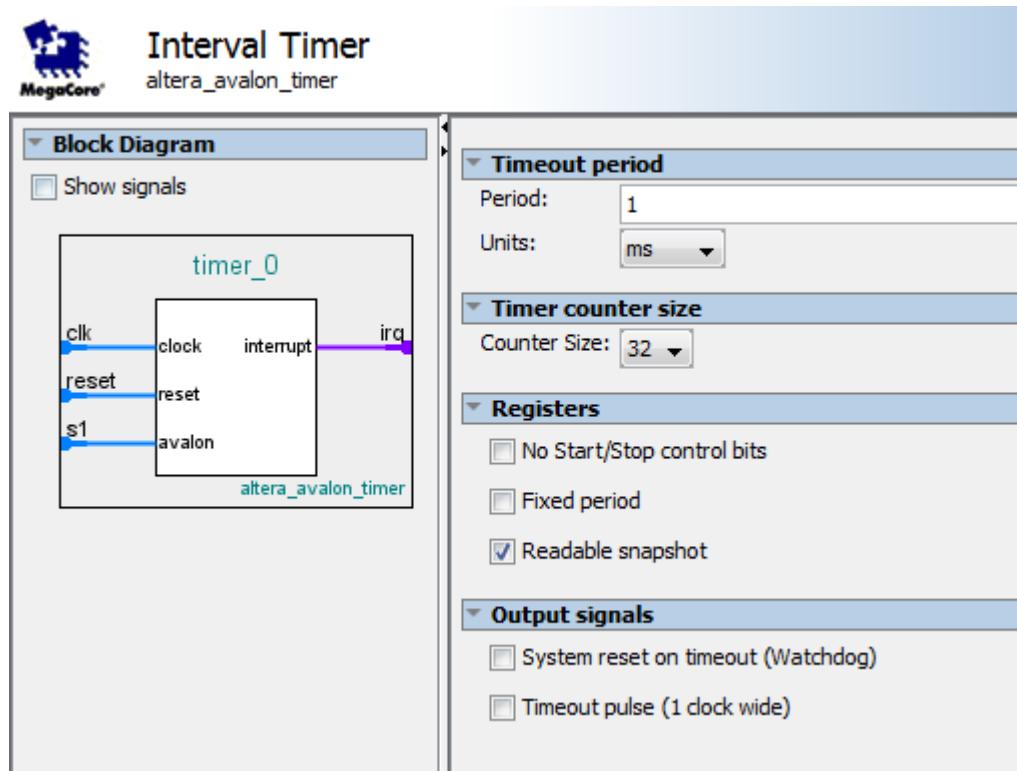
Many software applications make use of periodic interrupts to maintain timing requirements and prevent system lockups.

- 4.4.7.1 In the IP Catalog search bar, type **timer**. The Interval Timer component will appear under **Processors and Peripherals → Peripherals**.



- 4.4.7.2 Double-click the component or select it and click "Add..." to add it to the system. The Altera Interval Timer parameter editor will open.

- 4.4.7.3 Ensure the parameters are set to match the following such that the timer interval is 1 ms and the size is 32 bits.

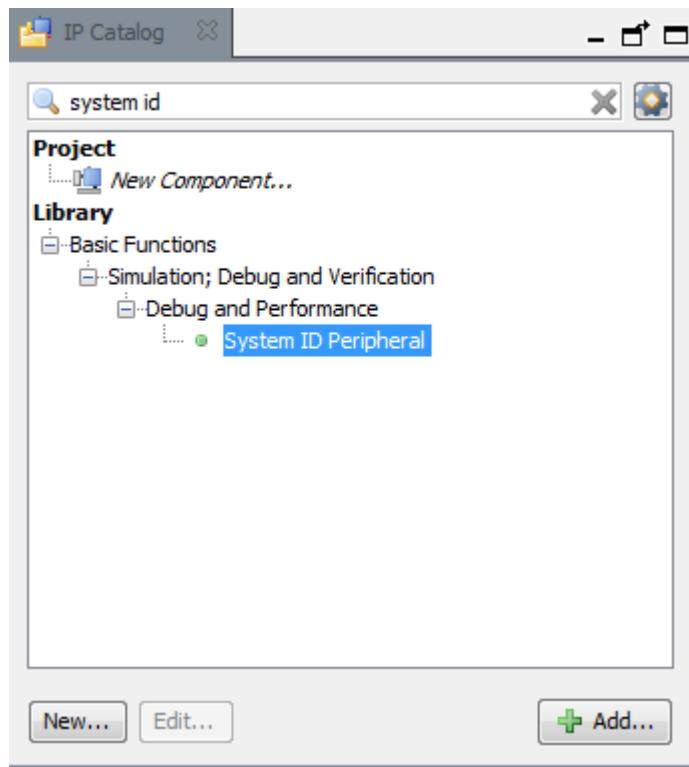


4.4.7.4 Click Finish. Do not worry about the warnings/errors as they will be fixed once we connect the clock to the IP.

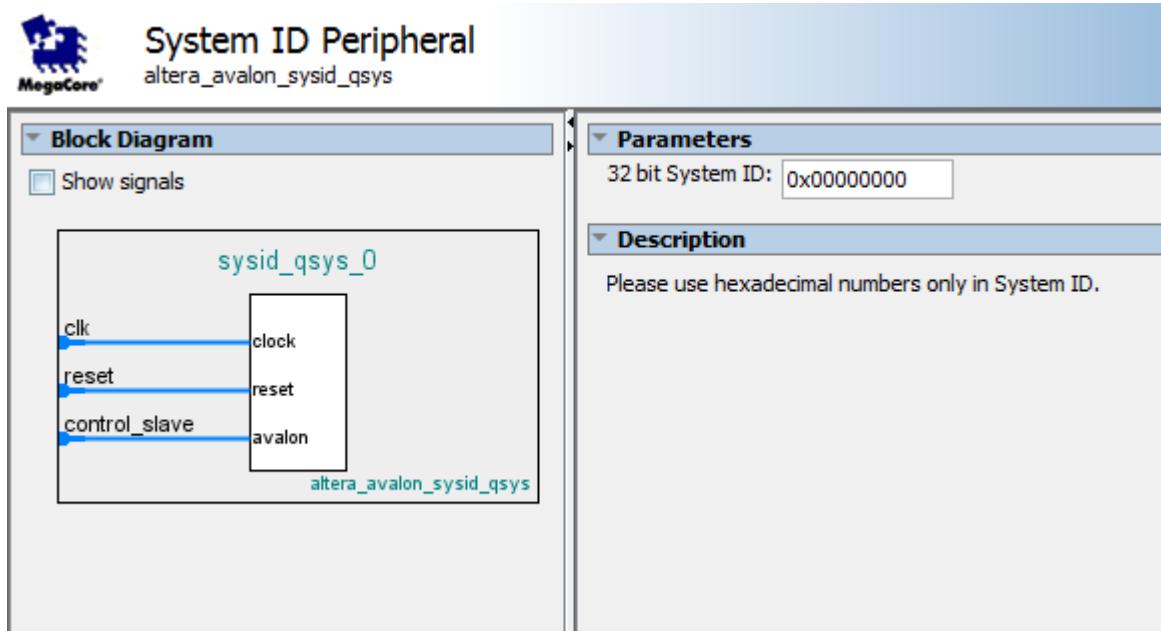
4.4.7.5 Rename the component **timer_qsys**.

4.4.8 Add a System ID Peripheral

4.4.8.1 In the IP Catalog search bar, type **system id**. The System ID Peripheral will appear under **Processors and Peripherals → Debug and Performance**.



- 4.4.8.2 Double-click the component or select it and click "Add..." to add it to the system. The System ID Peripheral parameter editor will open.



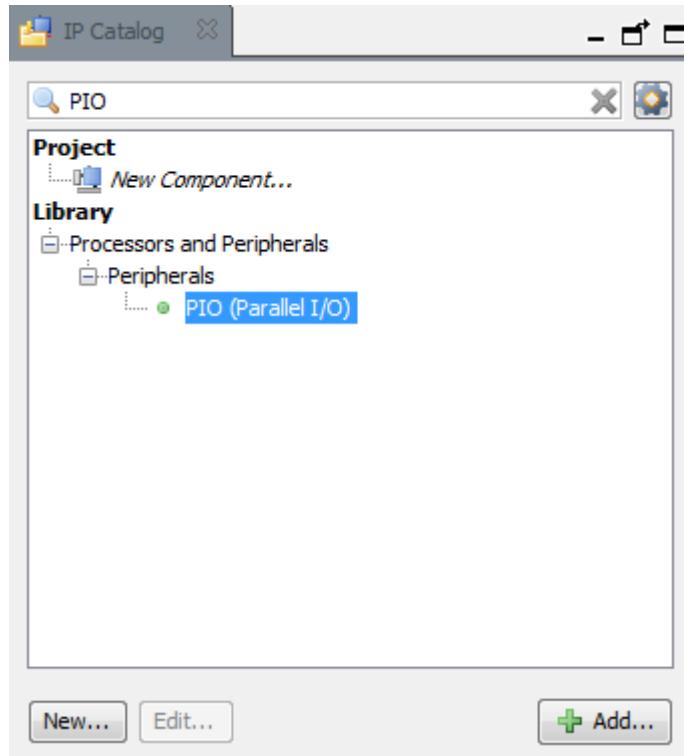
- 4.4.8.3 Select Finish and ignore the errors.

- 4.4.8.4 Rename the component **sysid_qsys**.

4.4.9 Add an Interrupt Pin

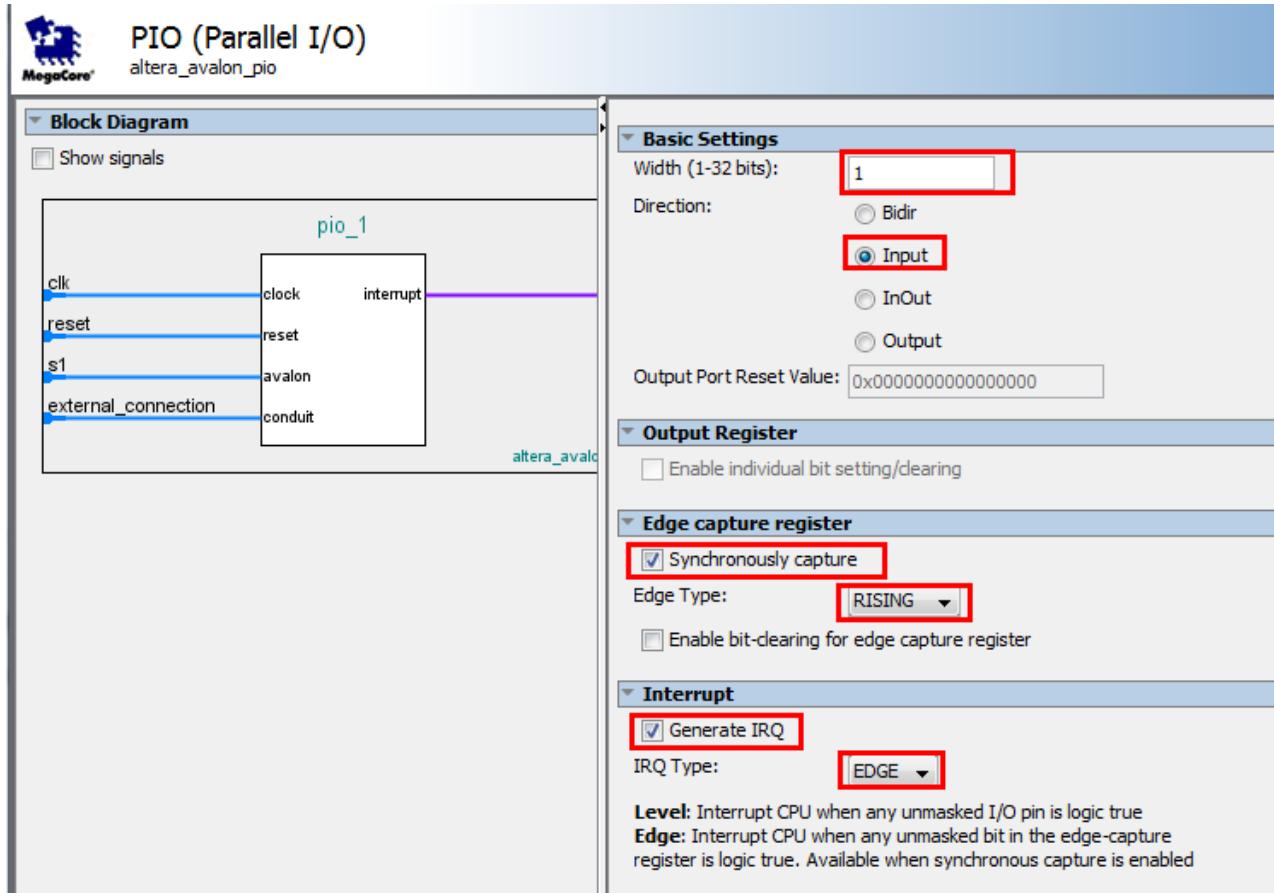
The Silicon Labs Si1143 Gesture Sensor has an interrupt pin to interrupt the host for a number of things, including when a measurement has been made or when the measurement exceeds a programmable threshold. Since the I²C bus has no interrupts, a parallel IO (PIO) pin needs to be added as an input to the Nios II processor.

4.4.9.1 In the IP Catalog search bar, type **PIO**. The PIO (Parallel I/O) component will appear under **Processors and Peripherals → Debug and Performance**.



4.4.9.2 Double-click the component or select it and click "Add..." to add it to the system. The PIO parameter editor will open.

4.4.9.3 Since only one input pin is required, configure the IP to have a width of 1 with "Input" as the direction. Ensure that the other settings match the following.



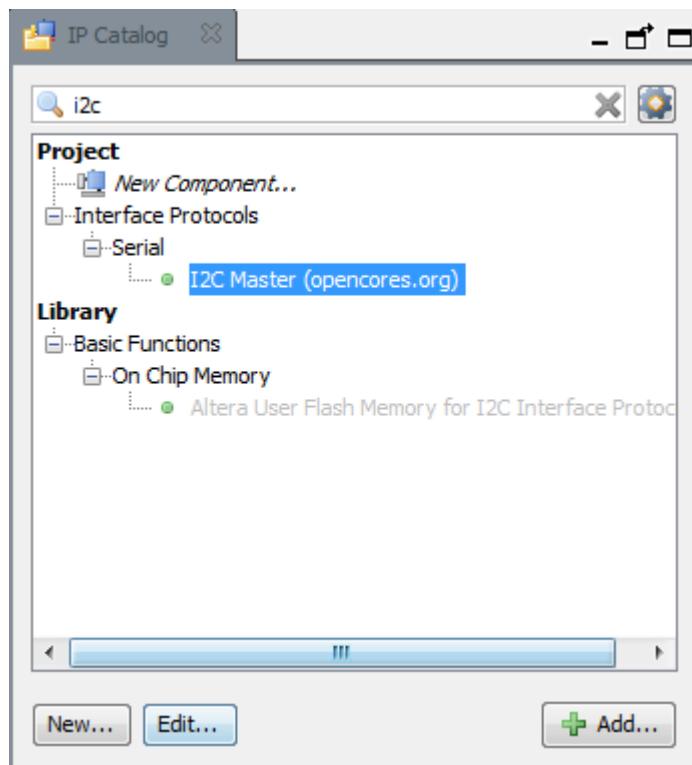
4.4.9.4 Select Finish. Rename the IP `pio_0` if it isn't already.

4.4.9.5 The interrupt pin needs to be added as an input port to the Qsys system so that it can be connected to an external pin. Double-click the export column next to the "external_connection" signal and type `i2c_irq`.

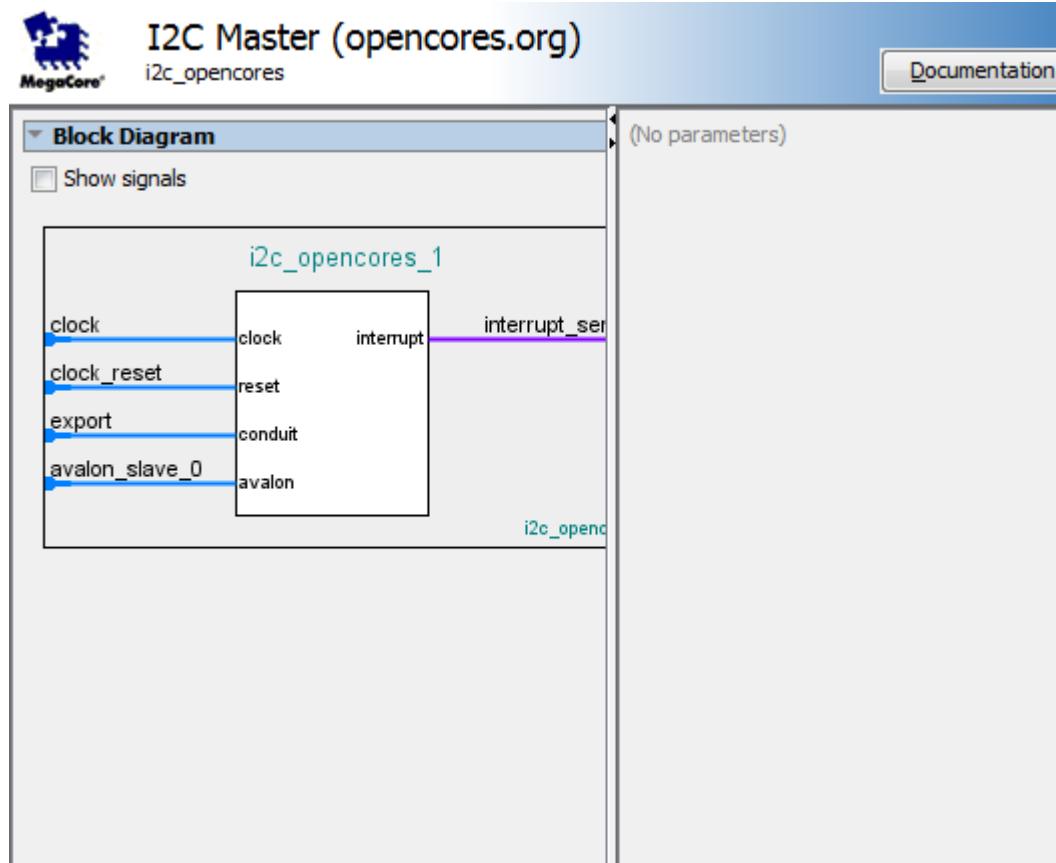
4.4.10 Add an I2C Peripheral

Since the Silicon Labs Si1143 Gesture Sensor communicates with the host using an I2C bus, an I2C controller needs to be added to the design.

4.4.10.1 The lab files contain an I2C master IP core so there should be an I2C Master (opencores.org) in the IP catalog under **Project → Interface Protocols → Serial**.



- 4.4.10.2 Double-click the component or select it and click "Add..." to add it to the system. The I2C parameter editor will open.



- 4.4.10.3 There are no parameters to be modified in this IP so select Finish. Rename the component `i2c_opencores_0` if it isn't already.

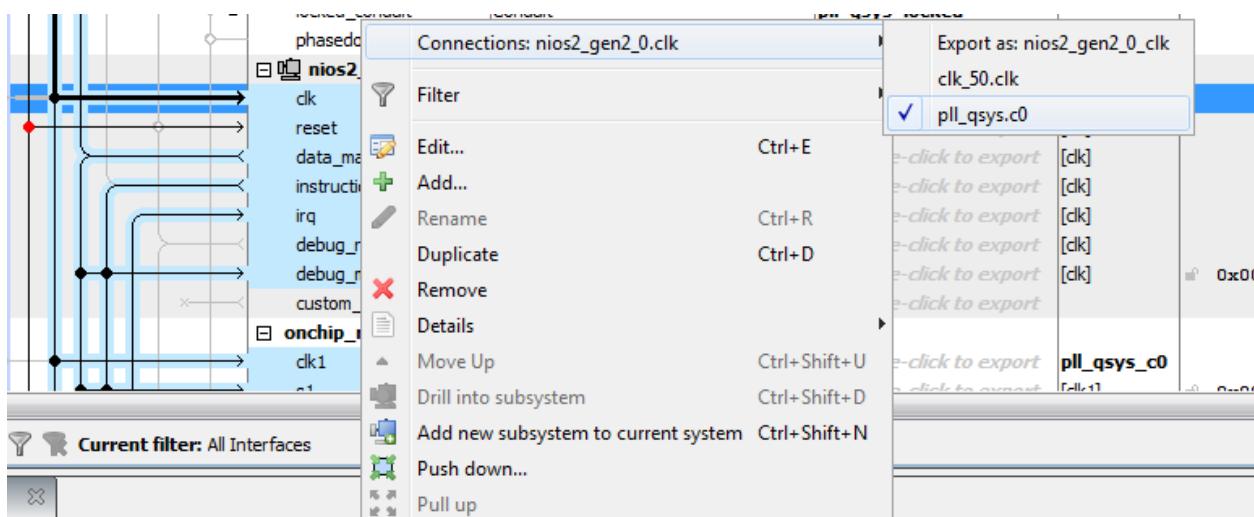
- 4.4.10.4 The I2C pins need to be added as top-level ports to the Qsys system. Double-click the export column next to the `export` signal in the `i2c_opencores_0` component and type `i2c_pins`.

4.4.11 Connect the Qsys system and remove errors

At this point, Qsys will report a number of errors referencing unconnected clocks, unconnected resets, and unconnected Avalon interface because none of the components in your Qsys system are connected. Once the interfaces are connected, many of these errors will disappear. These connections can be made in any order but for simplicity's sake, the connections will be made from top to bottom.

Connections can be made in a number of ways; either by clicking the connection nodes in the Connections column of the System Contents tab or by right clicking the signals and selecting the appropriate connection. The latter method is used in the following sections.

4.4.11.1 Right click the clk signal in the Nios II Processor, highlight "Connections: nios2_qsys.clk" and select "pll_qsys.c0". This will connect the clk input on the Nios II processor to the c0 output of the PLL.





Signal names are referred to by the following convention: <component_name.signal_name> where "component_name" refers to the name of the Qsys component in the Name column of the System Contents tab and "signal_name" refers to the signal or interface within the component.

The small arrows in the connections column denote whether the interface or signal is a source or a sink.

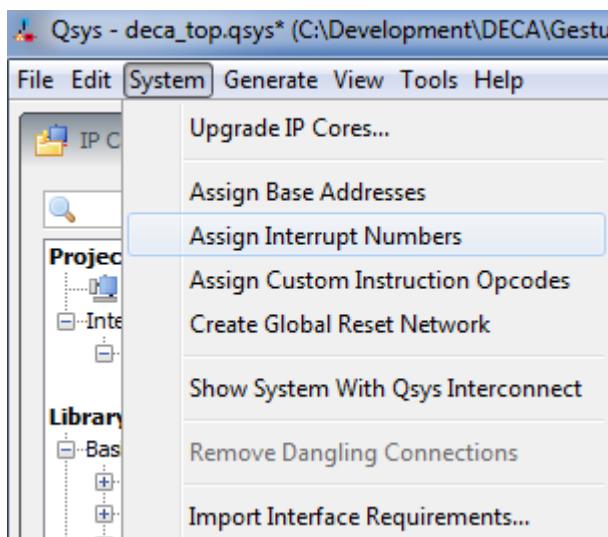
- 4.4.11.2 Repeat the same steps of connecting signals following the table below. Interfaces might be connected to more than one component and the order of the connections does not matter. For simplicity, the table lists the connections starting with the source component at the top of the Qsys system.

Source Signal	Sink Signal
clk_50.clk	i2c_opencores_0.clock
	pll_qsys.inclk_interface
clk_50.clk_reset	i2c_opencores_0.clock_reset
	jtag_uart.reset
	nios2_qsys.reset
	onchip_ram.reset
	pio_0.reset
	pll_qsys.inclk_interface_reset
	sysid_qsys.reset
	timer_qsys.reset
nios2_qsys.data_master	i2c_opencores_0.avalon_slave_0
	jtag_uart.avalong_jtag_slave
	nios2_qsys.debug_mem_slave
	onchip_ram.s1
	pio_0.s1
	pll_qsys pll_slave
	sysid_qsys.control_slave
	timer_qsys.s1
nios2_qsys.instruction_master	nios2_qsys.debug_mem_slave
	onchip_ram.s1
timer_qsys.irq	nios2_qsys.irq

pll_qsys.co	jtag_uart.clk
	nios2_qsys.clk
	onchip_ram.clk1
	pio_0.clk
	sysid_qsys.clk
	timer_qsys.clk
i2c_opencores_0.interrupt_sender	nios2_qsys.irq
jtag_uart.irq	nios2_qsys.irq
pio_0.irq	nios2_qsys.irq

4.4.12 Set Interrupt Priorities

The Nios II processor can process up to 32 independent interrupts (IRQs) from various parts of a system that can be assigned unique priorities. This system only has 4 interrupts and the priorities will need to be set manually although it can be done automatically by selecting **System → Assign Interrupt Numbers** from the Qsys menu as below.



- 4.4.12.1 Set an IRQ priority in Qsys by double clicking the number in the IRQ column of the System Contents tab and entering the priority (priority 0 is the highest priority. For example, double click the number in the IRQ column to the right to the "irq" signal in the timer_qsys module and type 0. This will give the timer component's interrupt the highest priority.

Name	Description	Export	Clock	Base	End	IRQ
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]		IRQ 0	IRQ 31
irq	Interrupt Receiver	Double-click to export	[clk]			
debug_reset_request	Reset Output	Double-click to export	[clk]			
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0008_0800	0x0008_0FFF	
custom_instruction_m...	Custom Instruction Master	Double-click to export				
timer_qsys	Interval Timer		pll_qsys_c0			
clk	Clock Input	Double-click to export	[clk]			
reset	Reset Input	Double-click to export	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0008_1020	0x0008_103F	
irq	Interrupt Sender	Double-click to export	[clk]			0

- 4.4.12.2 Assign the IRQ priorities to the other interrupt senders according to the following table:

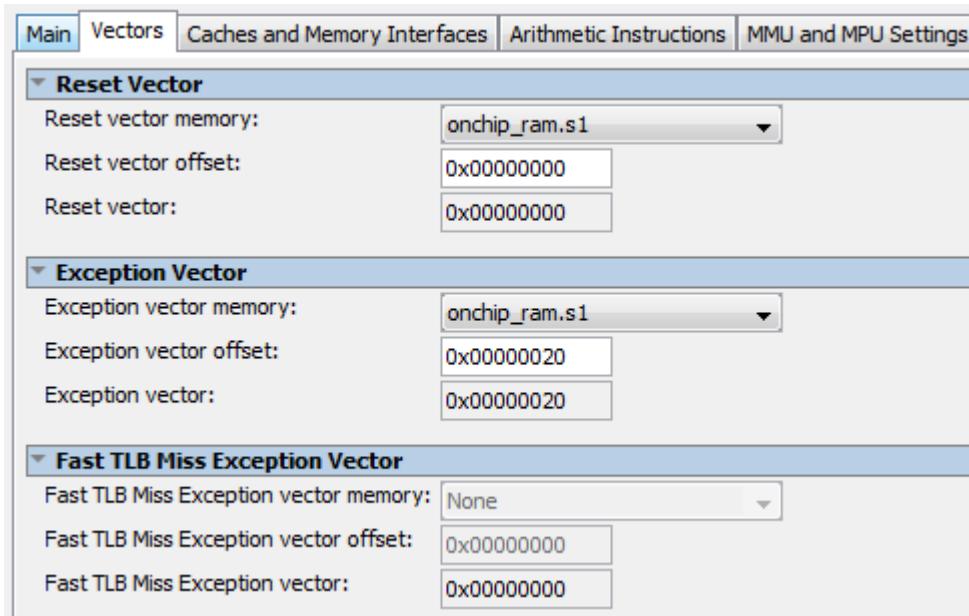
IRQ	Priority
jtag_uart.irq	3
timer_qsys.irq	0
pio_0.irq	1
i2c_opencores_0.interrupt_sender	2

4.4.13 Define the Nios II Reset and Exception Vectors

Like any processor, the Nios II requires memory locations to jump to in the event of a processor reset or exception within the execution of its code. The reset vector is the memory location to which the processor jumps on processor reset and the exception vector is the memory location to which the processor jumps on an exception. These are typically in non-volatile memory and can be at the same memory location.

4.4.13.1 To set these vectors, double-click on the Nios II component (nios2_qsys). The Nios II parameter editor will reopen.

4.4.13.2 Click on the Vectors tab and set both the reset vector memory and exception vector memory to be **onchip_ram.s1** from the pull-downs. The offset and vector values may be different than the image below but will be corrected in a following step.

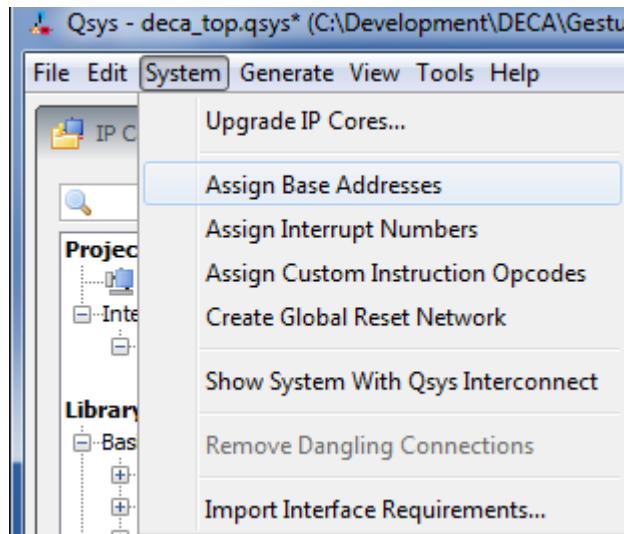


4.4.13.3 Close the parameter editor to close the window. Remaining errors will be resolved in the following steps.

4.4.14 Resolve Memory Addresses

All of the components in a Qsys system are assigned a memory address so that other components can access them through the Avalon Memory-Mapped interconnect that Qsys generates. Qsys automatically assigns the base address of each component to 0x000000 which causes the memory overlap errors. Assigning a unique base address to each component resolves these errors.

There are two ways to assign base addresses to your Qsys system: enter them manually or have Qsys automatically assign them. For this lab, we will manually assign them but if you were defining your own system and wanted to do it automatically, click on **System → Assign Base Addresses** from the Qsys menu as below.



- 4.4.14.1 Double-click on the hex address in the Base column of the System Contents tab next to the debug_mem_slave signal of the Nios II Processor component and enter 0x0008_0800 to change the base address of the component.

<input checked="" type="checkbox"/> nios2_gen2_0	Nios II Processor			
→ dk	Clock Input	Double-click to export	pll_qsys_c0	
→ reset	Reset Input	Double-click to export	[dk]	
→ data_master	Avalon Memory Mapped Master	Double-click to export	[dk]	
→ instruction_master	Avalon Memory Mapped Master	Double-click to export	[dk]	
→ irq	Interrupt Receiver	Double-click to export	[dk]	IRQ 0
→ debug_reset_request	Reset Output	Double-click to export	[dk]	
→ debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[dk]	0x0008_0800
→ custom_instruction_m...	Custom Instruction Master	Double-click to export		

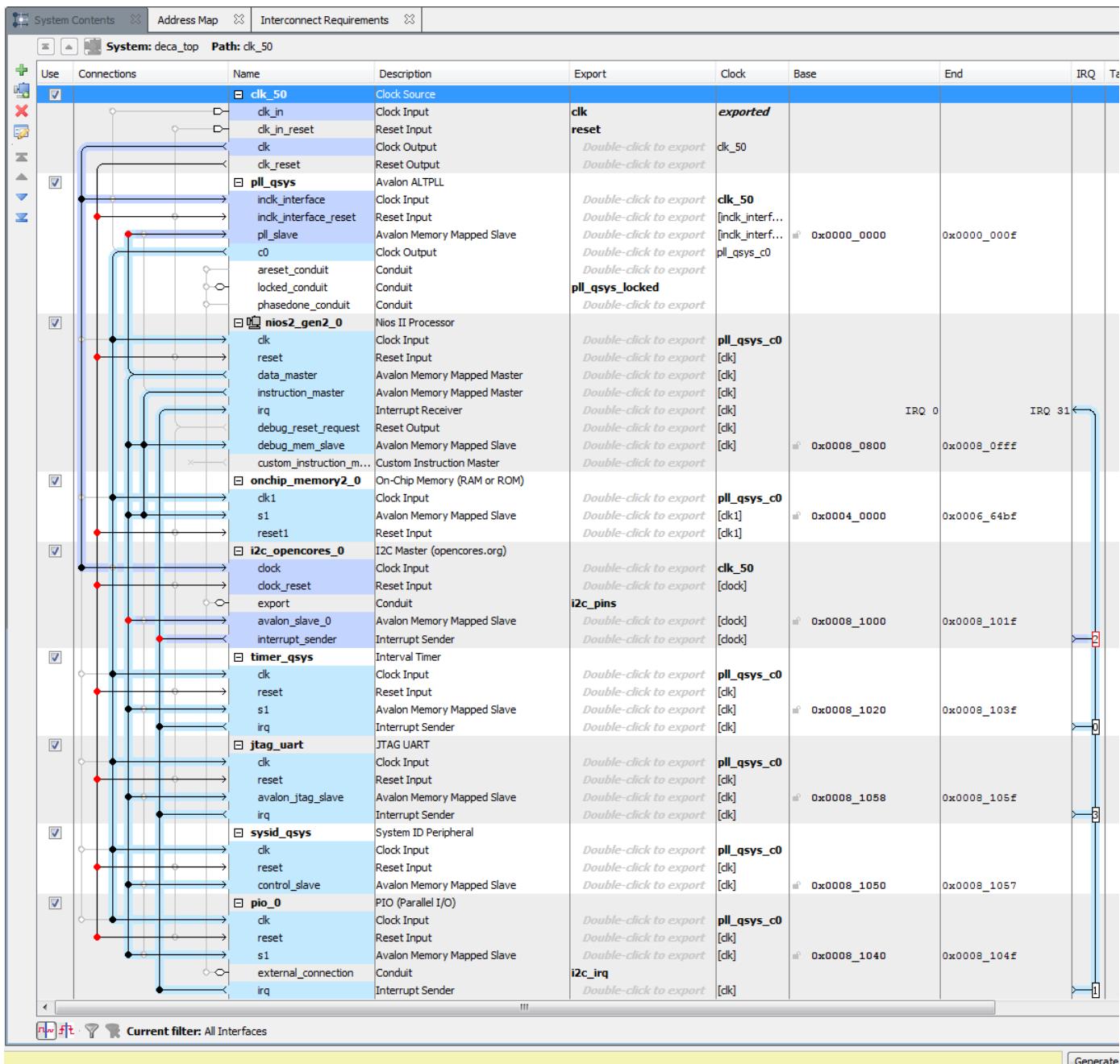
- 4.4.14.2 Repeat this process for each component until the base addresses match the table below.

Component	Base Address
nios2_qsys	0x0008_0800
timer_qsys	0x0008_1020
pll_qsys	0x0000_0000
onchip_ram	0x0004_0000
i2c_opencores_0	0x0008_1000
jtag_uart	0x0008_1058
sysid_qsys	0x0008_1050
pio_0	0x0008_1040

4.4.15 Check the full system

Below is a screenshot of the full Qsys system with all connections visible.

4.4.15.1 Double check that your Qsys system matches the screenshot below. The order of the components is not important but ensure that the connections are the same.



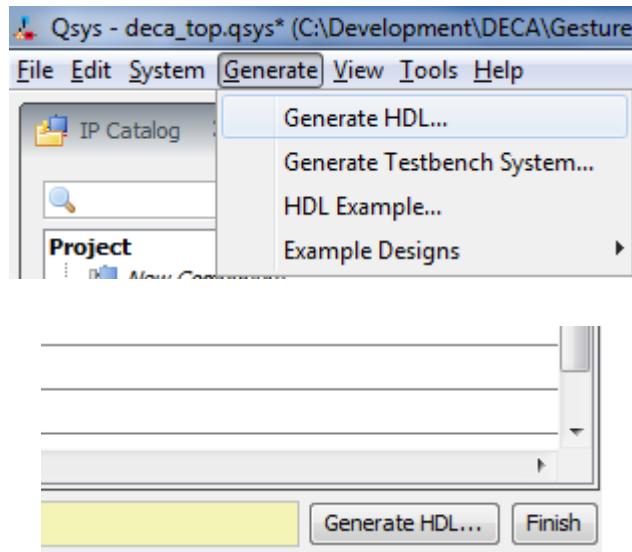
4.4.15.2 Make sure that there are no error messages in the Messages tab. There should only be the two warnings shown below and info messages. If you do have an error, refer to the error message to resolve it.

	2 Warnings	
	deca_top pll_qsys	pll_qsys.areset_conduit must be exported, or connected to a matching conduit.
	deca_top pll_qsys	pll_qsys.phasedone_conduit must be exported, or connected to a matching conduit.

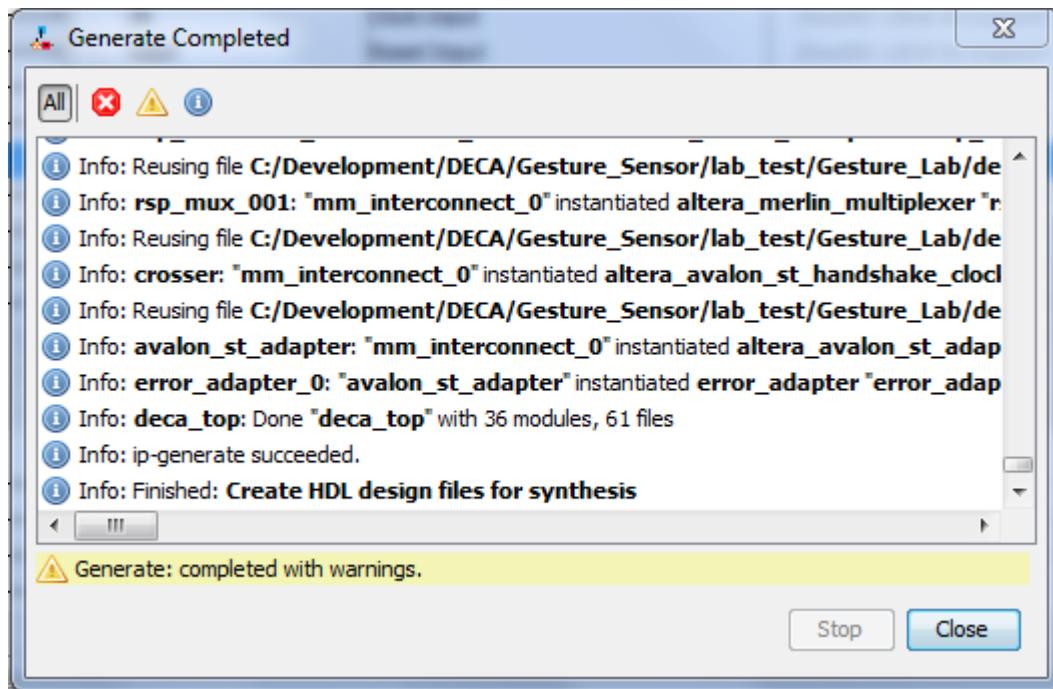
4.4.16 Generate the Qsys System

One of the great parts about Qsys is that it generates HDL (hardware description language) code from the created system so that the internals can be investigated for a better understanding. The next step is to generate the HDL from the system.

- 4.4.16.1 Select **Generate → Generate HDL...** from the Qsys menu or click the Generate HDL... button on the bottom right of the Qsys window.



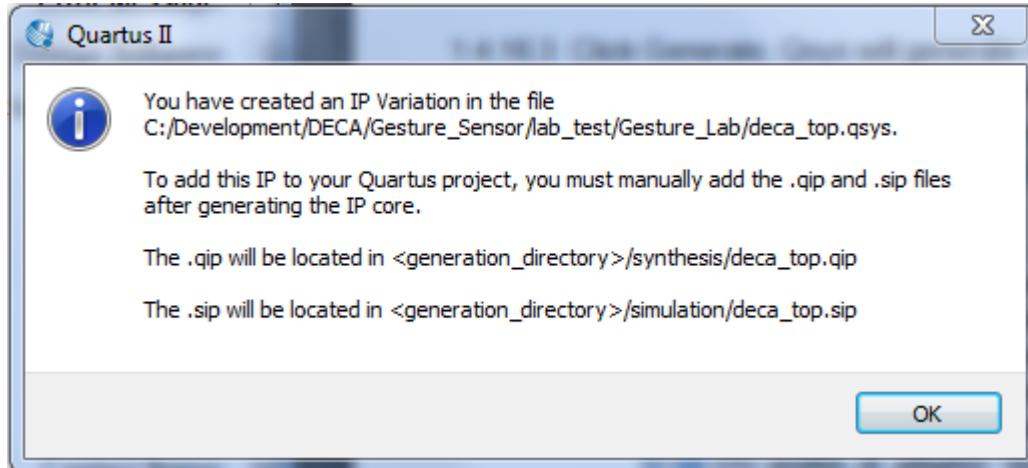
- 4.4.16.2 The Generate window will appear. Select "Verilog" as the synthesis language and "None" from the simulation model dropdown. VHDL can be used but the top level file in this lab is in Verilog. The generated HDL files will appear in the directory pointed to by the file path specified under the Output Directory section. Leave this as the default.
- 4.4.16.3 Click Generate. Qsys will generate the necessary HDL for synthesis. When the generate process completes (with warnings), click Close.



4.4.17 Add the Qsys System to the Quartus 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.

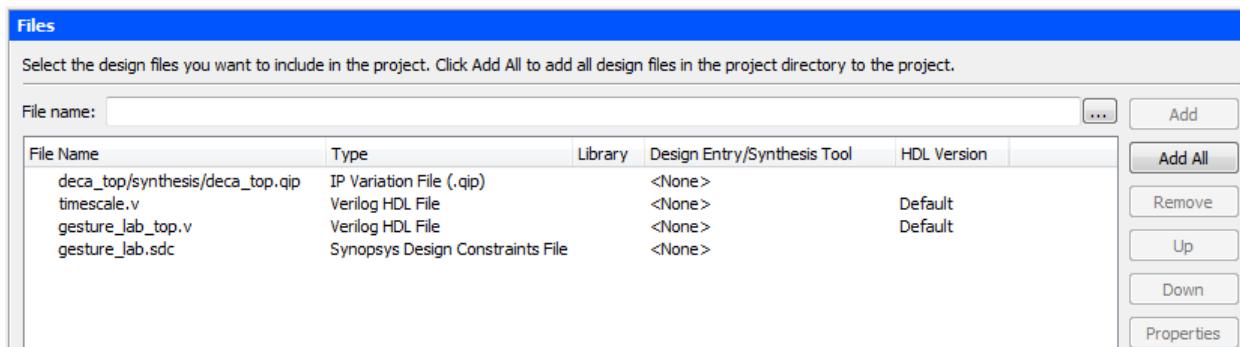
- 4.4.17.1 Click the "X" at the top right of the Qsys window to close Qsys. A Quartus II message window should appear advising you to add the Qsys system to your project. Note the file paths for the .qip and .sip files.



- 4.4.17.2 Click "OK" and select **Project → Add/Remove Files in Project** from the Quartus II menu.

- 4.4.17.3 Click the "..." and browse to the synthesis directory noted above (it should be `<project_directory>/deca_top/synthesis/`) and select `deca_top.qip`.

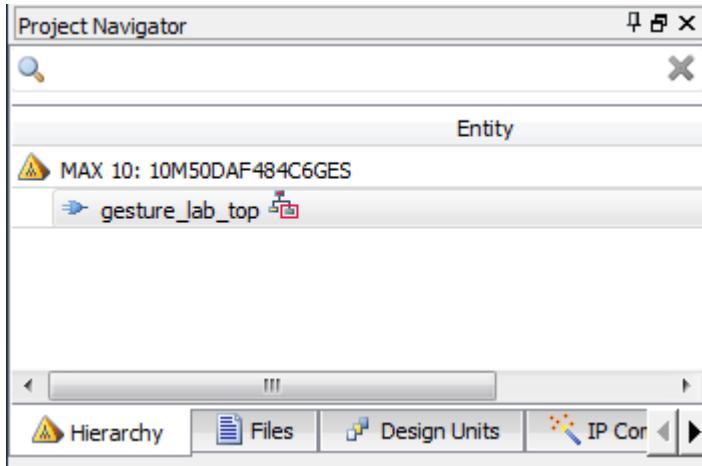
- 4.4.17.4 Click "Add" to add the .qip file to the project. Click "Apply" and "OK".



4.4.18 Modify the Top-Level Design File

The top-level design file has already been created for you but a few modifications need to be made to compile the design.

- 4.4.18.1 Double-click the top-level design entity, `gesture_lab_top`, in the Hierarchy tab of the Project Navigator window. The file `gesture_lab_top.v` that you added to the project in step 4.3.2.5 should open in the main window.



- 4.4.18.2 Take a look at the top level file observing the ports, wire declarations and the structural coding. Examine the instantiation of the Qsys system on line 198 of the file. Notice that the signals that were exported from within Qsys are listed as ports to the Qsys system.

- 4.4.18.3 Uncomment the four `assign` statements on lines 187-190 and add the signal names for the I2C signals and the PLL locked signal. Pay close attention to the polarity of the signals.

```

181 //=====
182 // Structural coding
183 //=====
184
185     // add LEDs to monitor important signals
186     // such as the I2C bus, the interrupts and PLL outputs
187     //assign LED[ 0 ] = SIGNAL_NAME;
188     //assign LED[ 1 ] = SIGNAL_NAME;
189     //assign LED[ 2 ] = SIGNAL_NAME;
190     //assign LED[ 3 ] = SIGNAL_NAME;
191
192
193     assign LED[ 4 ] = 1;
194     assign LED[ 5 ] = 1;
195     assign LED[ 6 ] = 1;
196     assign LED[ 7 ] = reset_n;
197
198     deca_top u0 (
199         .clk_clk             ( MAX10_CLK1_50 ),           // clk.clk
200         .pll_qsys_locked_export ( pll_qsys_locked ),      // pll_qsys_locked.export
201         .reset_reset_n       ( reset_n ),                 // reset.reset_n
202         .i2c_irq_export      ( LIGHT_INT ),               // spi_irq.export
203         .i2c_pins_scl_pad_io ( LIGHT_I2C_SCL ),          // i2c_pins.scl_pad_io
204         .i2c_pins_sda_pad_io ( LIGHT_I2C_SDA )           // .sda_pad_io
205     );
206

```

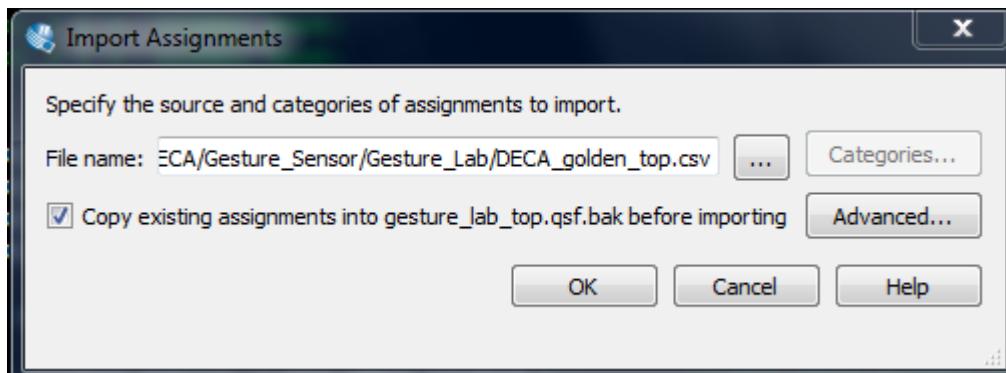
4.4.18.4 Click **File → Save All** from the Quartus II menu.

4.4.19 Import Pin Assignments

Normally, the pin assignments for a design would need to be entered manually depending on the board-level implementation. Since the pins on the DECA board are fixed, we can import the pin assignments automatically.

4.4.19.1 Import the pin assignments from **Assignments → Import Assignments**.

4.4.19.2 Click the "..." and select `DECA_golden_top.csv`. Click OK. Quartus II should report 563 assignments imported.



Type	ID	Message
Info	140120	Import completed. 563 assignments were written (out of 563 read).

4.4.19.3 Open the Pin Planner from **Assignments → Pin Planner**. Take a look at the pin assignments particularly at the signals beginning with `LIGHT`. These are the I2C signals to the Si1143 sensor.

?	LIGHT_I2C_SCL	Unknown	PIN_Y8	3	B3_N0	3.3-V LVTTL	8mA (default)
?	LIGHT_I2C_SDA	Unknown	PIN_AA8	3	B3_N0	3.3-V LVTTL	8mA (default)
?	LIGHT_INT	Unknown	PIN_AA9	3	B3_N0	3.3-V LVTTL	8mA (default)

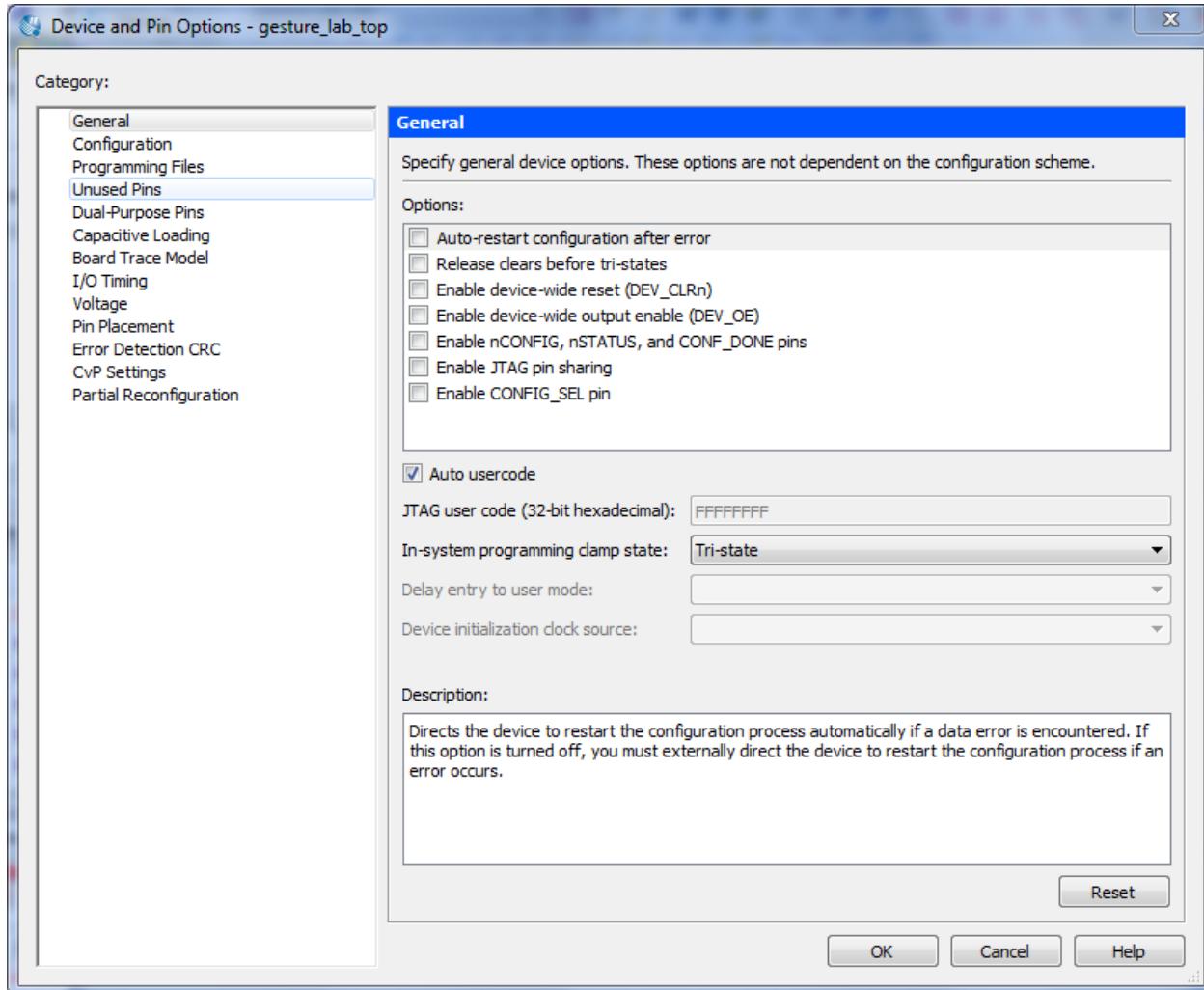
4.4.19.4 Click the "X" at the top right to close the Pin Planner.

4.4.20 Compile the Quartus II project

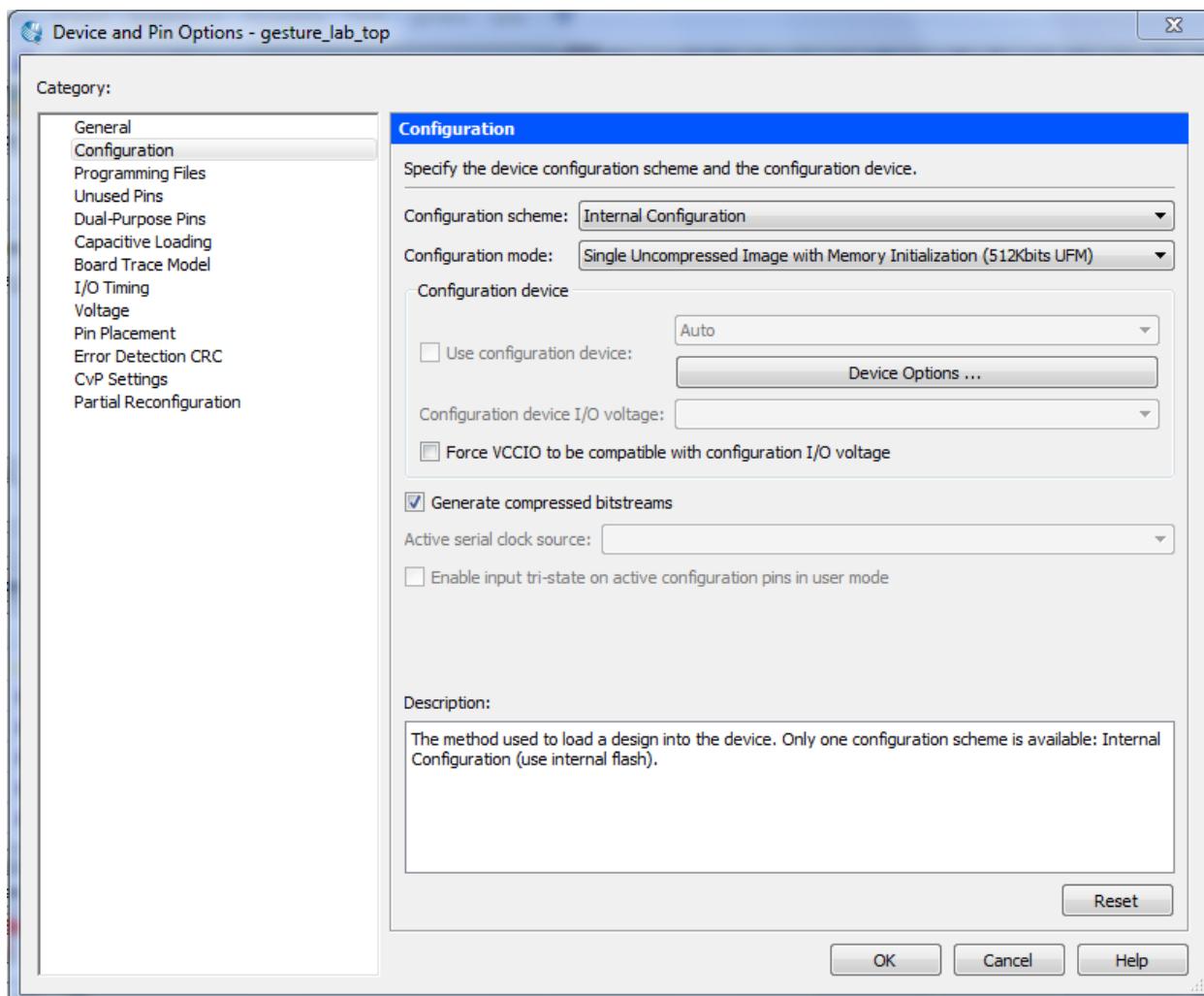
With the hardware design complete, a few device settings need to be changed and the project can be compiled to create a configuration file.

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

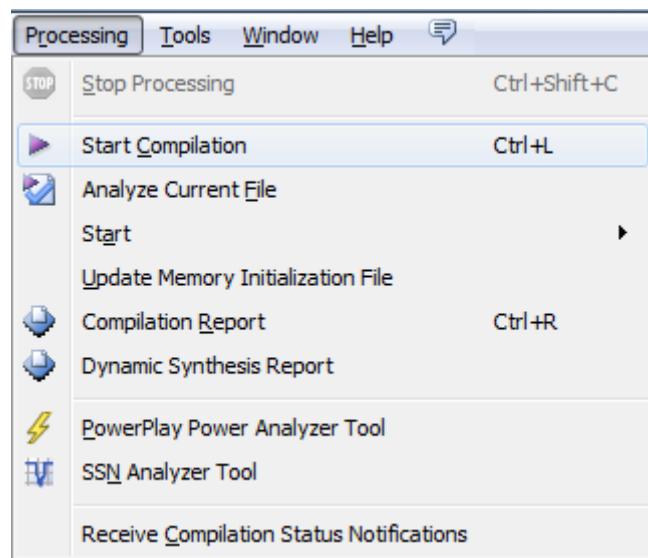
4.4.20.2 Unselect all of the checkboxes in the Options box in the General category so that they match the following. This will allow the fitter to appropriately select the correct VCCIO for the banks that contain the device configuration pins.



4.4.20.3 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.



4.4.20.4 Kick off the compilation by selecting **Processing → Start Compilation** or double-click Compile Design in the Task window.



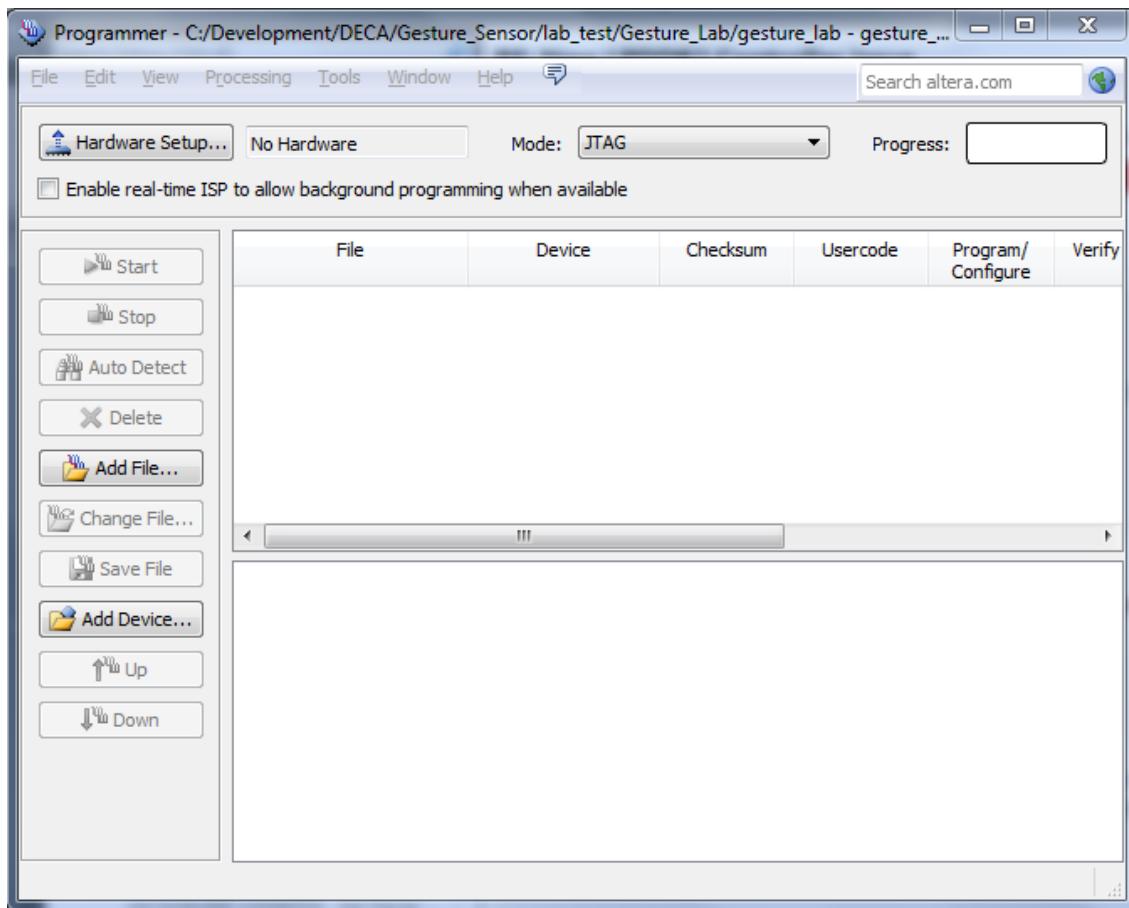
4.4.20.5 After a few minutes, the compilation should complete with no errors.

Tasks		
Flow:	Compilation	Customize...
	Task	⌚
✓	▶ ▶ Compile Design	00:02:53
✓	▶ ▶ Analysis & Synthesis	00:01:09
✓	▶ ▶ Fitter (Place & Route)	00:01:03
✓	▶ ▶ Assembler (Generate programming files)	00:00:09
✓	▶ ▶ TimeQuest Timing Analysis	00:00:26
✓	▶ ▶ EDA Netlist Writer	00:00:06
	▶ Program Device (Open Programmer)	

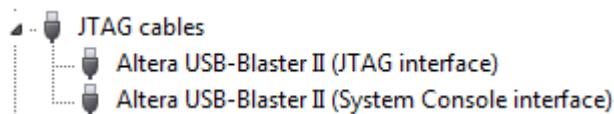
4.4.21 Download the Configuration File to DECA

Now that the hardware design is complete and has been converted into a configuration file, the DECA needs to be programmed.

- 4.4.21.1 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.



- 4.4.21.2 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 Windows' Device Manager should display two entries under "JTAG Cables".

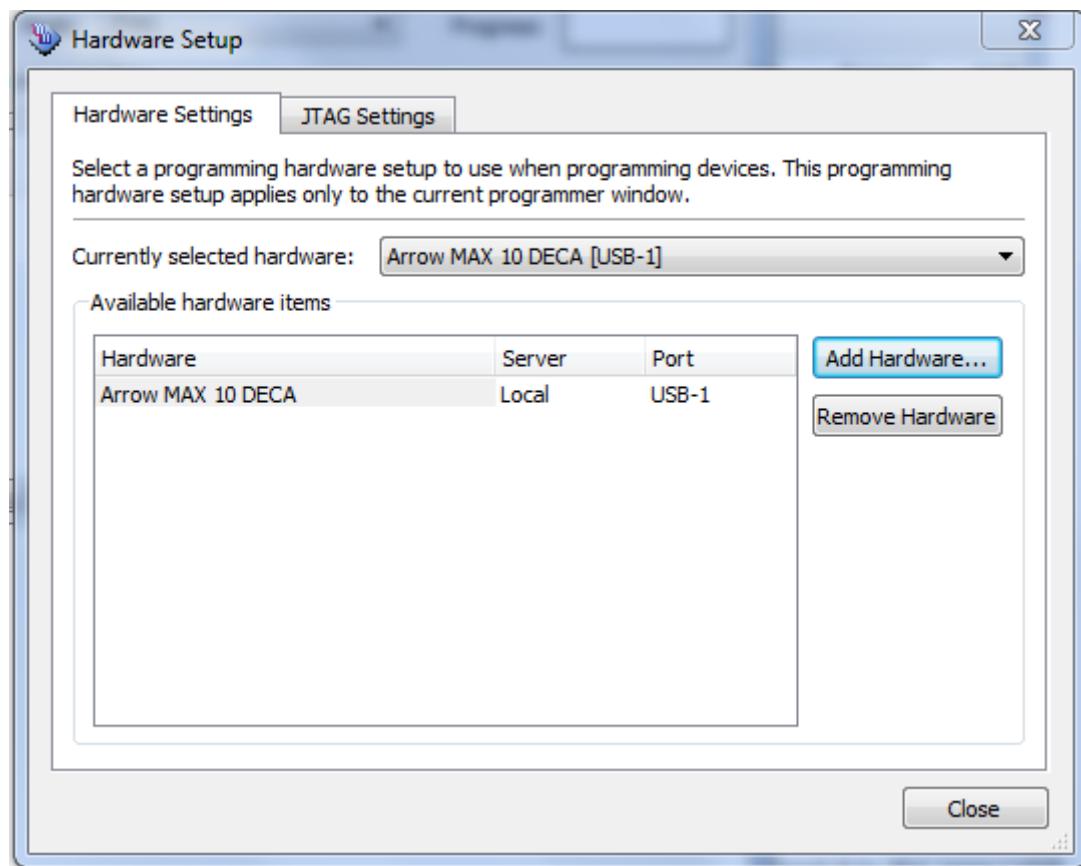


You should see a few LEDs light up on your DECA including the blue LED labeled **3.3V** and the green LED labeled **CONF_D**.

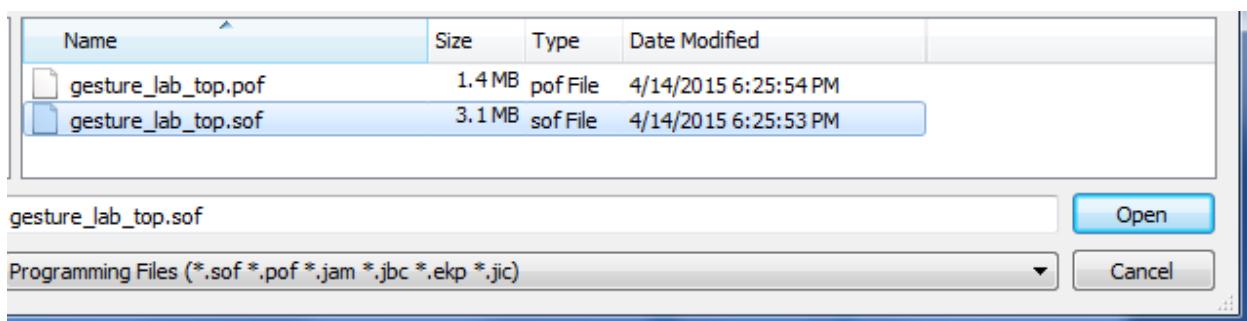


If the Device Manager shows an unconfigured USB Blaster, if Windows tries to look for drivers, or if the LEDs on the DECA do not light up, ask your workshop trainer for help.

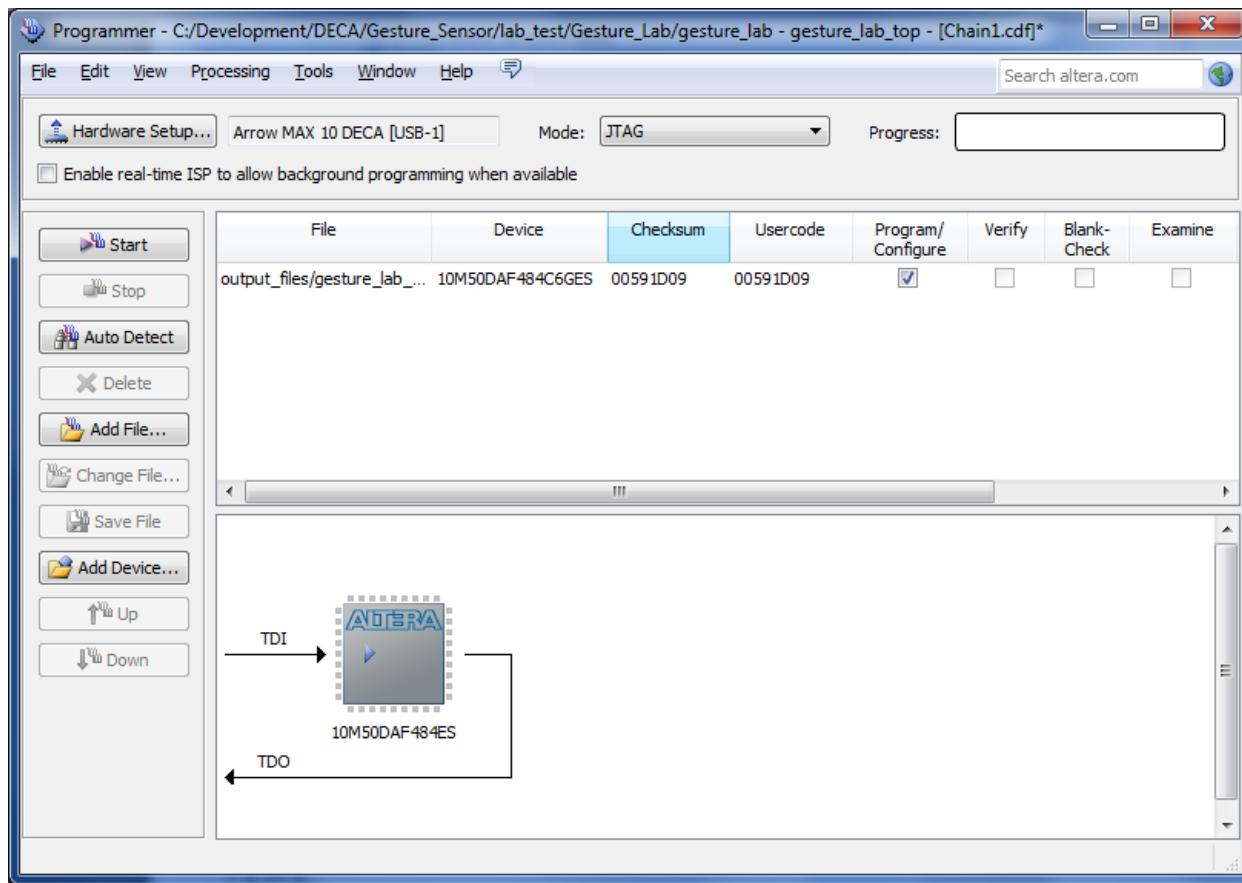
- 4.4.21.3 In the Programmer window, click Hardware Setup and double-click the Arrow MAX10 DECA entry in the Hardware pane. The Currently Selected Hardware drop-down should now show Arrow MAX10 DECA [USB-1]. Depending on your PC, the USB port number may be different. Click Close.



- 4.4.21.4 Click "Add File..." and navigate to <project_directory>/output_files/ in your compilation directory. Open the **gesture_lab_top.sof** file.



4.4.21.5 Make sure that the Programmer shows the correct file and the correct part in the JTAG chain as below.



4.4.21.6 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).

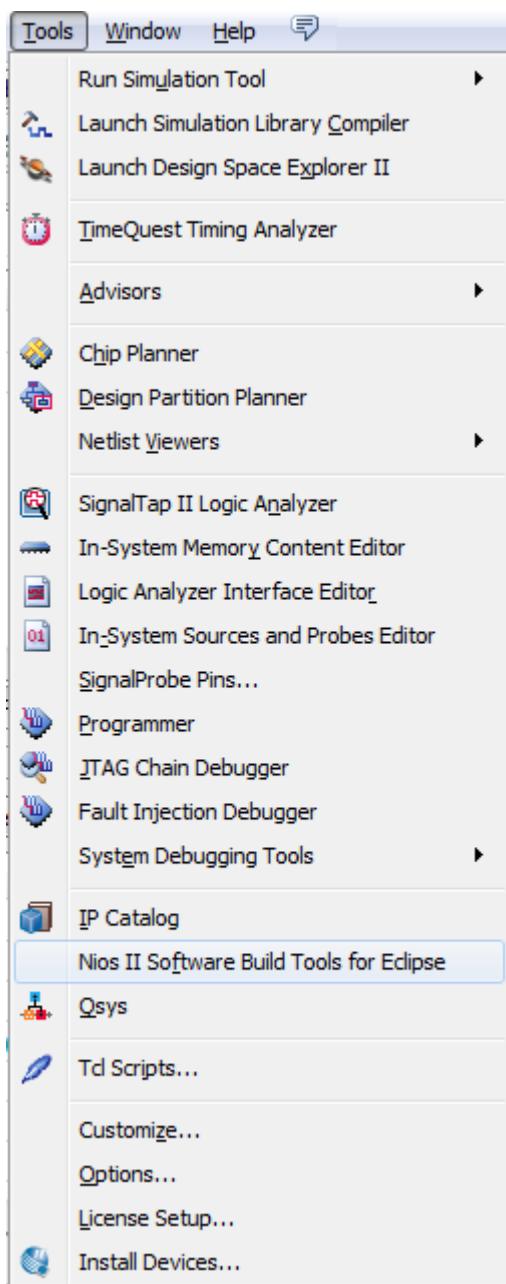
Progress: **100% (Successful)**

4.5 Create the Software Design

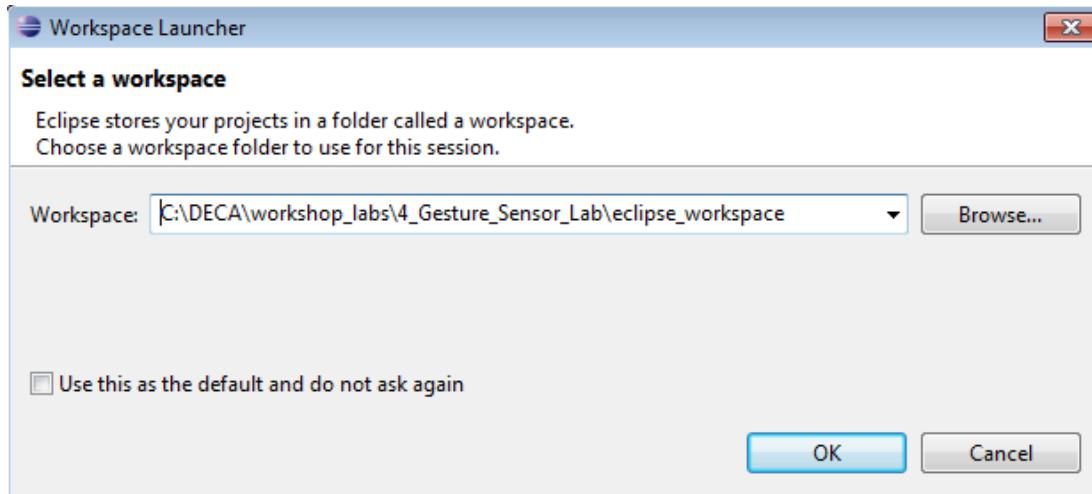
Overview: In this section, you will use the Nios II Software Build Tools (SBT) for Eclipse to quickly create a board support package (BSP) and a C software application to run on the Nios II processor you implemented in the last section. After creating and compiling the project, you will run it on the Nios II and interact with the Silicon Labs Si1143 Gesture IC by waving your hand across the back of the DECA.

4.5.1 Start Nios II Software Build Tools for Eclipse

4.5.1.1 From the main Quartus window, start the SBT from **Tools → Nios II Software Build Tools for Eclipse**.



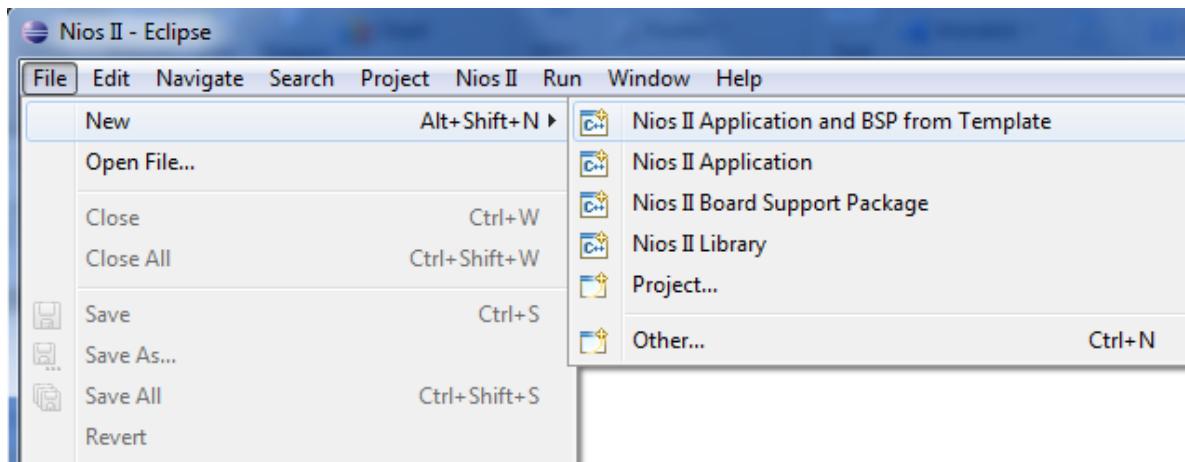
- 4.5.1.2 The Eclipse Workspace Launcher will open. Click "Browse..." and create a new folder titled **eclipse_workspace** in your lab directory to use as the software directory for the project. Click "OK".



4.5.2 Create a New Software Project

Now that Eclipse has a workspace, a new software application project and BSP can be created for your hardware system.

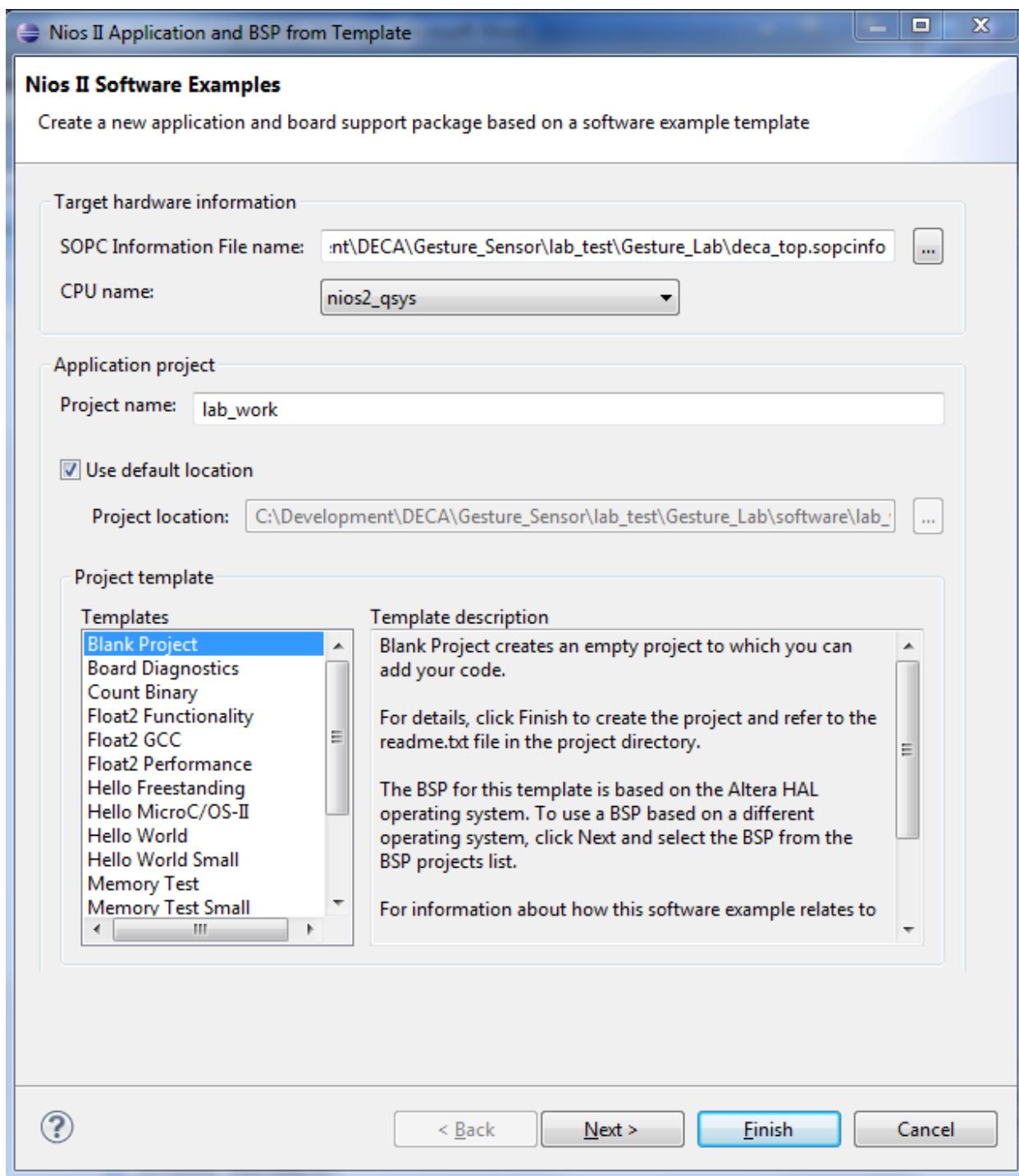
- 4.5.2.1 Once Eclipse opens to the workbench in the Nios II perspective, select **File → New → Nios II Application and BSP from Template** as shown below. This is an easy way to create a BSP and application together in a few easy steps.



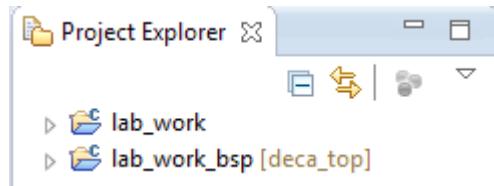
The BSP uses the Qsys-generated .sopcinfo file to import necessary settings from the hardware project to the software project so that your application can run on the Nios II processor. It allows Eclipse to build the system library drivers and generate system-specific macros for the custom Qsys system with the Nios II processor.

- 4.5.2.2 Click "..." to select **deca_top.sopcinfo** from your project directory and call the project **lab_work**. Make sure you select **Blank Project** from the Templates section as the software source files will be added in a later step. Make sure the settings match the screenshot below and select "Finish".

Note: your file path may be different than the one shown.



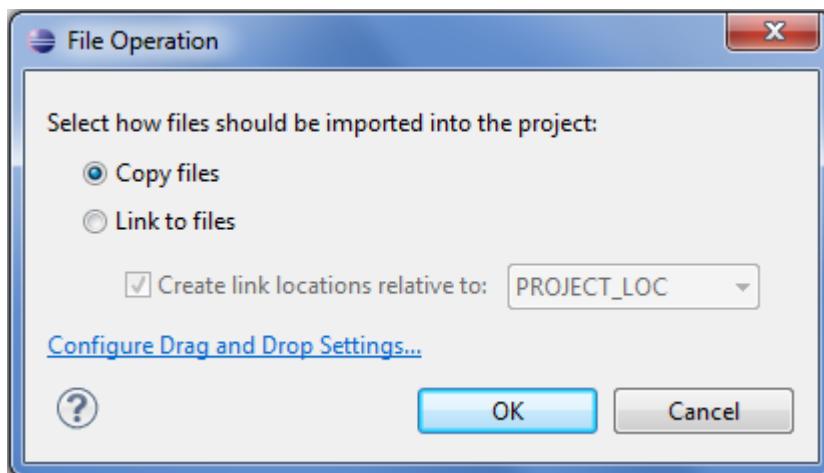
- 4.5.2.3 Eclipse will create two directories in the workspace; one for the application project and one for the BSP. The application directory (`lab_work`) is currently empty while the BSP directory (`lab_work_bsp`) contains software drivers, a `system.h` header file, initialization source code and other software infrastructure.



4.5.3 Add Source Code to the Project

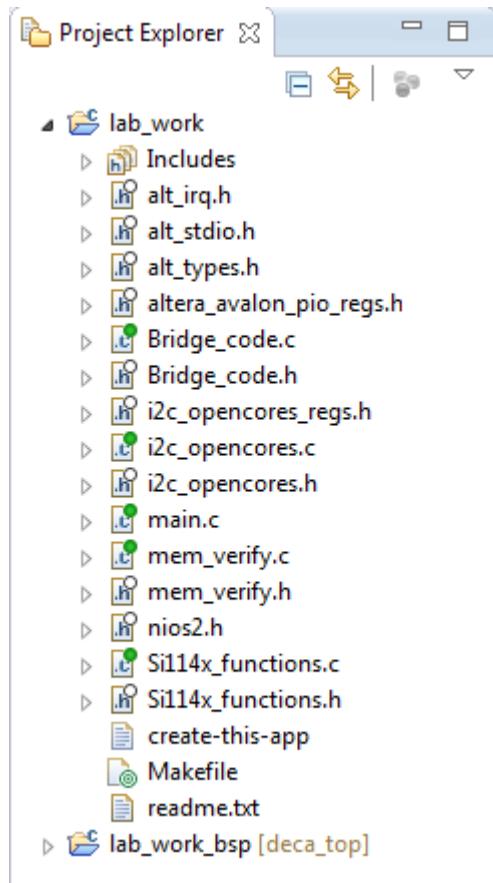
The C source files and accompanying header files have been provided for you in this lab. All that needs to be done is to copy them to your workspace.

- 4.5.3.1 From Windows Explorer, navigate to your main project directory and into the folder `<project_directory>/sw_src/`. There you should find a number of C source and header files. All of these need to be copied to the project.
- 4.5.3.2 Select all of the files and drag and drop them into the `lab_work` directory in Eclipse. Select the "Copy files" option in the pop-up and click "OK".



Since we are copying the files instead of linking to them, any changes that you would want to make to the source files need to be made to the versions inside the `lab_work` directory. Otherwise, the changes will not be compiled.

You should now see all of the files appear under the `lab_work` project in the Project Explorer.

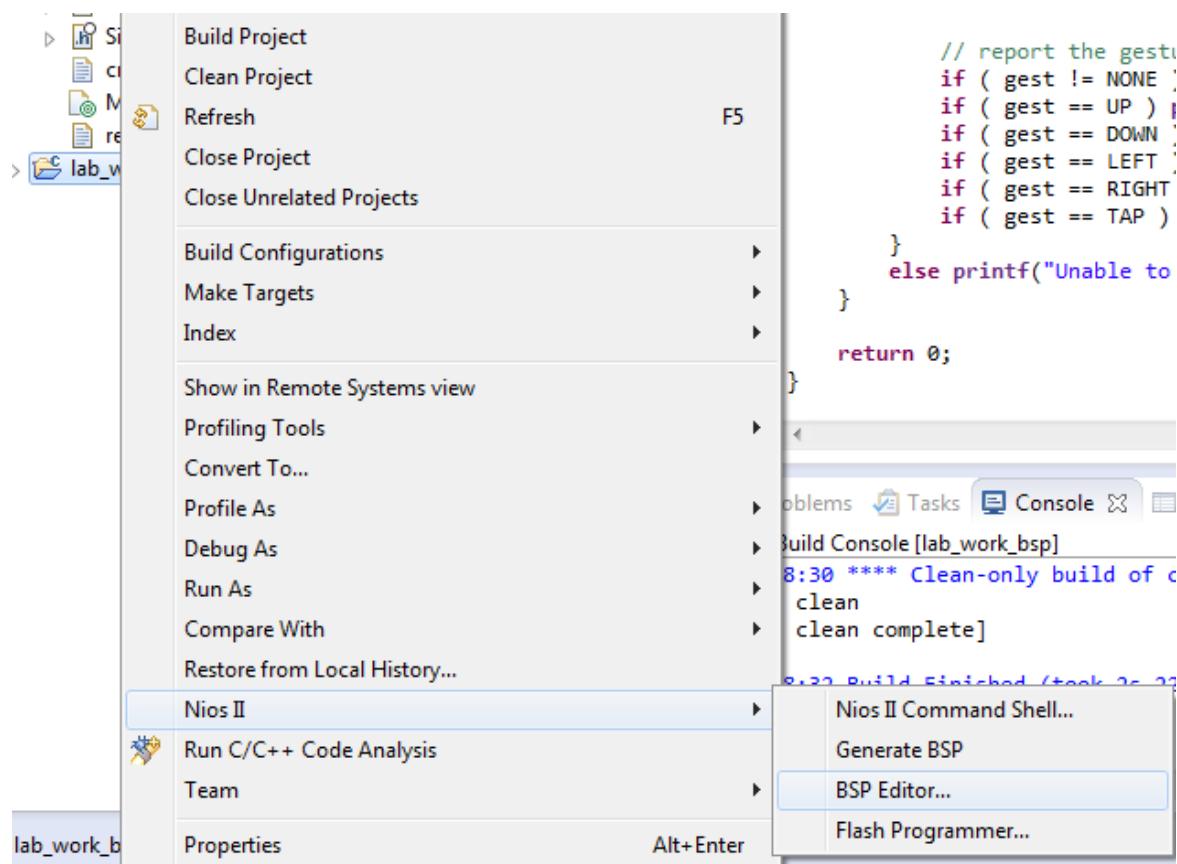


- 4.5.3.3 Double-click the main.c file and examine the `main` function starting on line 23. Try to follow the flow of the code to understand how the program executes and decodes the data from the sensor.

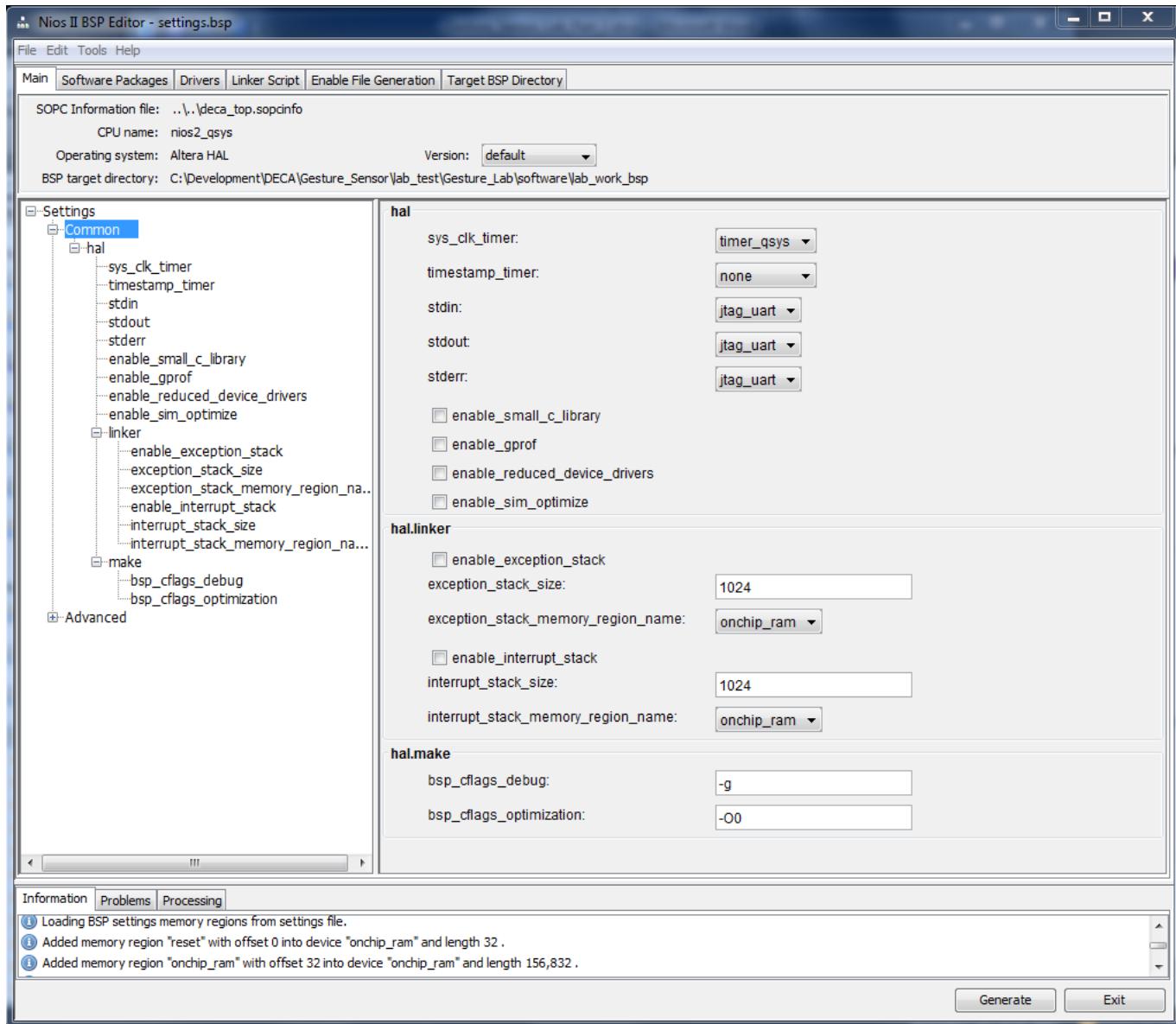
4.5.4 Configure the Board Support Package

The Board Support Package specifies the properties of the software system and needs to be configured for the software to execute correctly. These properties include setting the stdin, stdout and stderr interfaces, memory allocation for the heap and stack, drivers, and whether an operating system will be used.

- 4.5.4.1 Right-click on the **lab_work_bsp** project and select **Nios II → BSP Editor** from the pop-up menu.



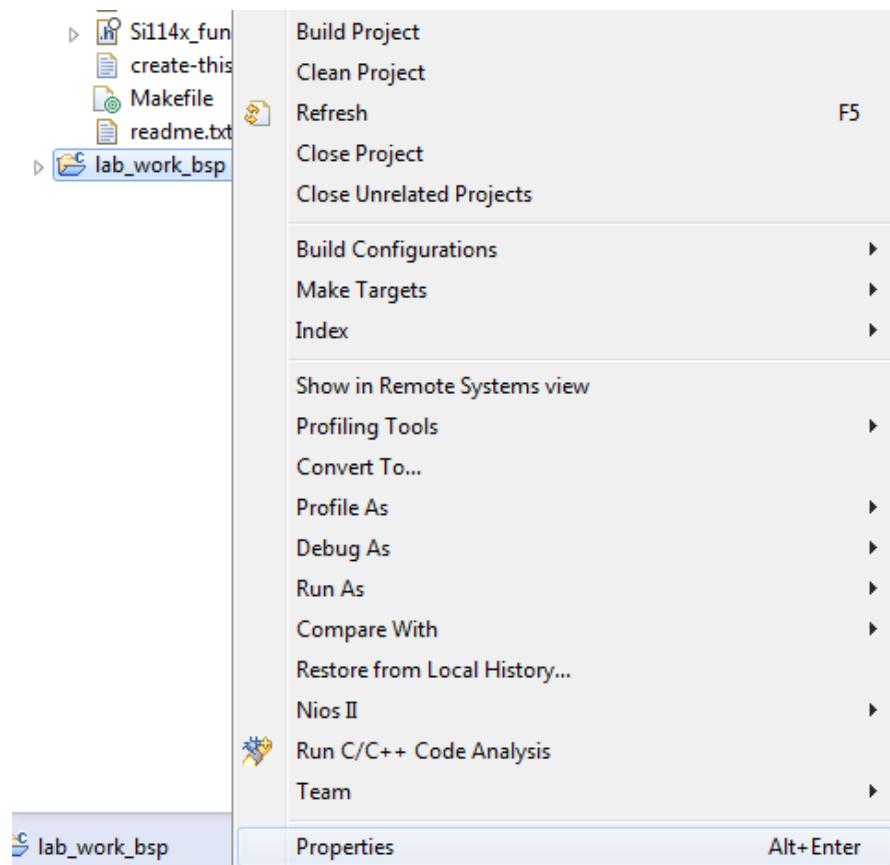
4.5.4.2 The Nios II BSP Editor will open. In the Common settings under the Main tab, ensure that the settings are configured as follows.



Notice that since there is no operating system in this lab, the stdout, stdin, and stderr messages are reported through the JTAG UART which you will be able to see from Eclipse. On-chip memory will be used for the exception and interrupt stack registers.

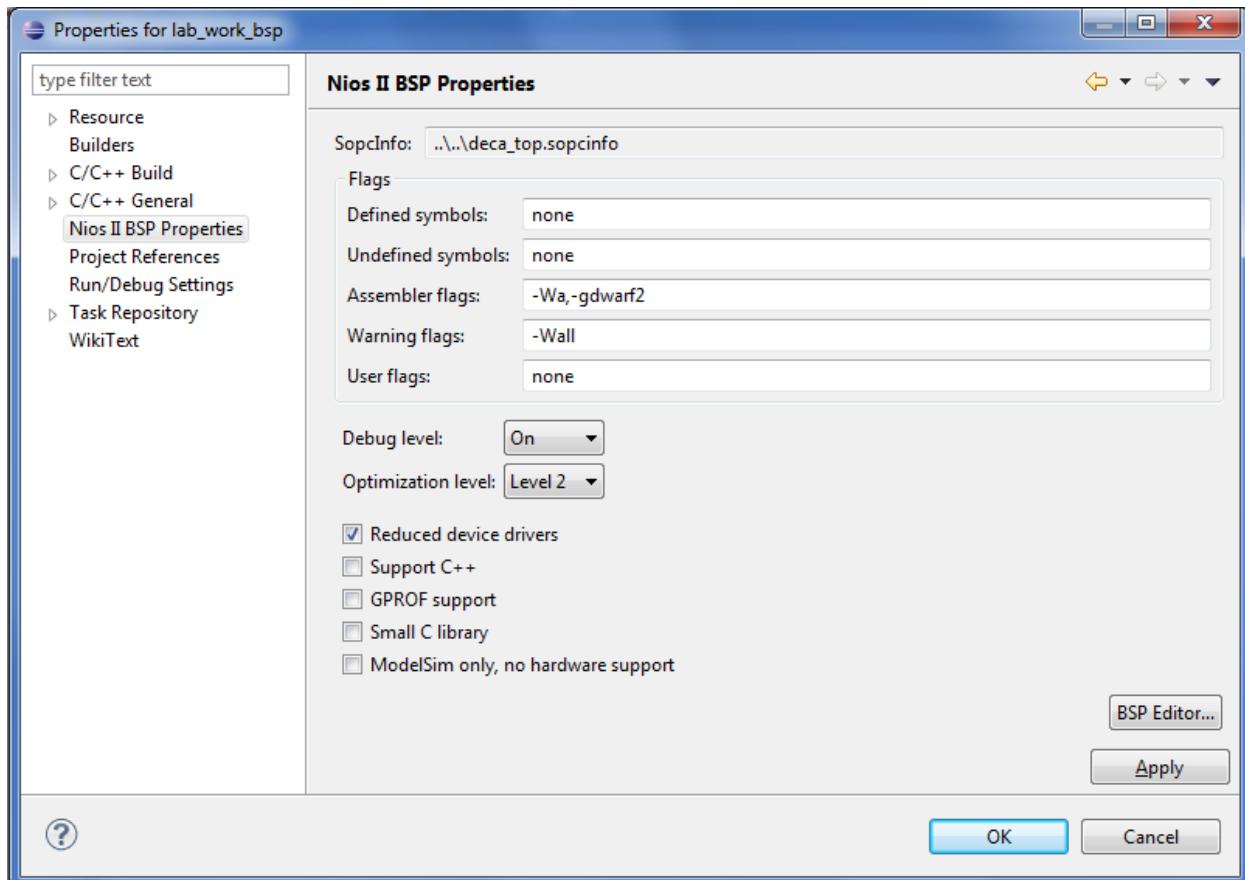
Feel free to explore the BSP editor. The Drivers tab gives the user control over what drivers are built into the BSP. The Linker Script tab provides a mechanism to adjust what memory regions are utilized for certain purposes. We only have one memory in this system but for systems with multiple memory locations (i.e. DDR3, flash, and on-chip ram), this is particularly useful.

- 4.5.4.3 Click the "Generate" button to update the BSP and select **File → Exit** to close it once the process is complete.
- 4.5.4.4 There are a few more BSP settings to edit. Right-click on the **lab_work_bsp** project and select Properties from the pop-up menu.



- 4.5.4.5 In the Properties window, select the Nios II BSP Properties tab. It may take a moment to load the settings.
- 4.5.4.6 Set the Optimization level setting to **Level 2**. This setting allows the user to tune the degree to which the compiler optimizes the code allowing for control between code size and performance.
- 4.5.4.7 To keep the software footprint small, enable the "Reduced device drivers" option. As there is no C++ code, disable the "Support C++" option.

The BSP Properties should match the following.



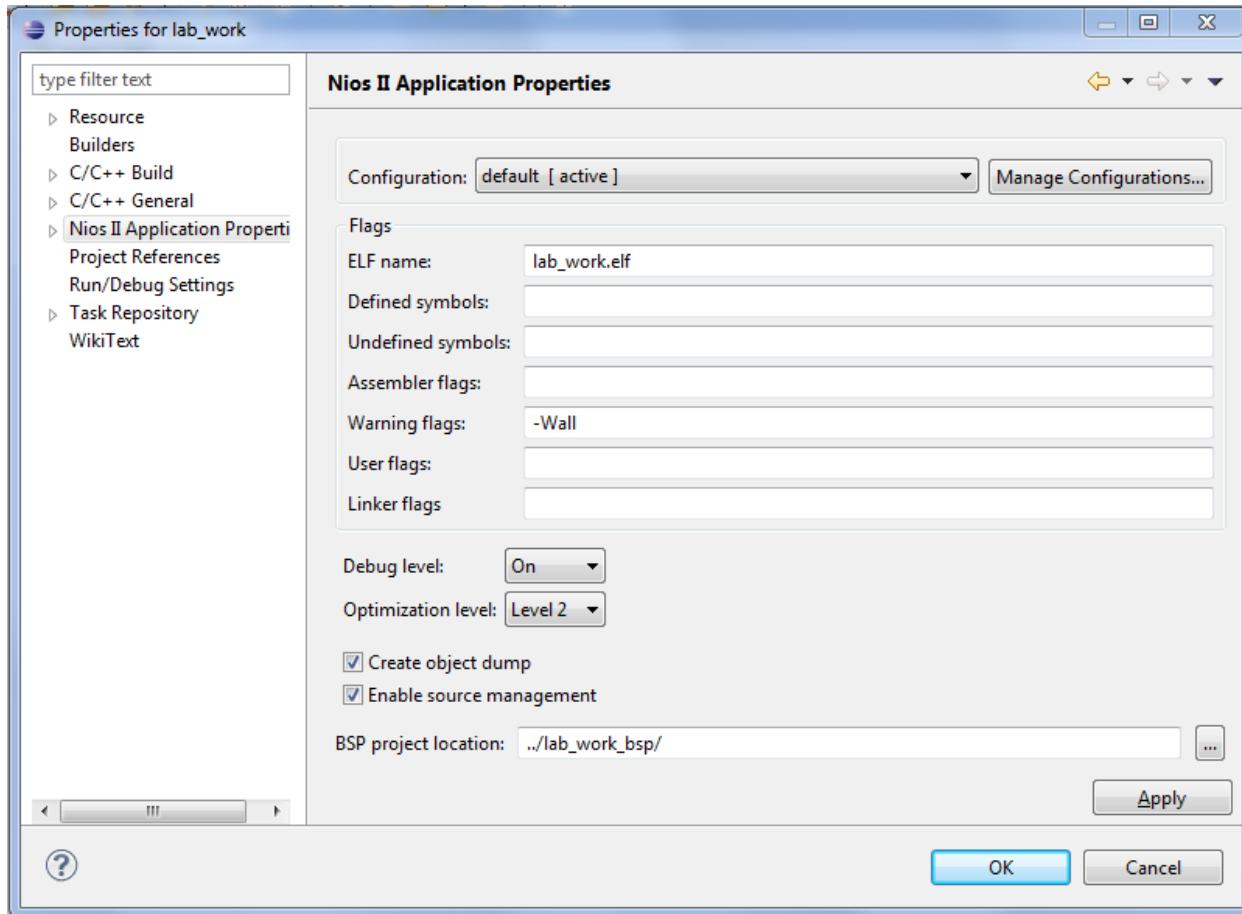
- 4.5.4.8 Select "Apply" and click "OK" to exit the Properties window.

4.5.5 Configure the Application Project

The software application settings will need to match those of the BSP for the code to execute properly.

4.5.5.1 Right-click on the **lab_work** software project and select properties from the pop-up menu as you did with the BSP.

4.5.5.2 Select the Nios II Application Properties tab and change the Optimization level to **Level 2** as shown below.

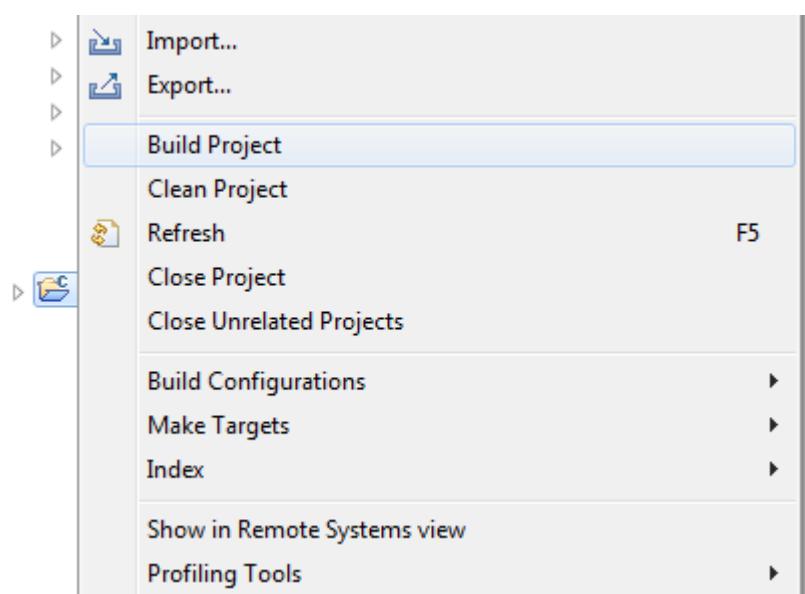


4.5.5.3 Select "Apply" and click "OK" to exit the Properties window.

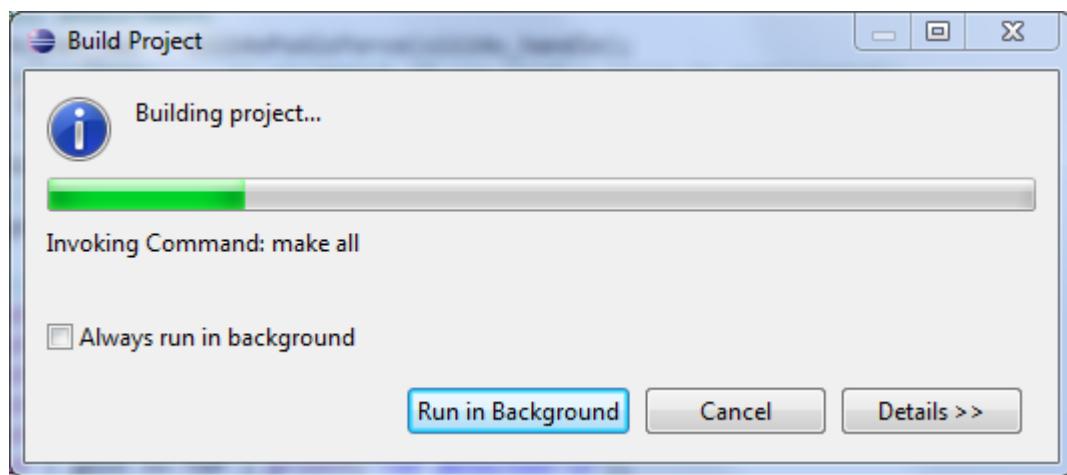
4.5.6 Build the Software Project

With all of the appropriate settings configured, you can now build the BSP and software project using the next two steps to produce an executable and linked format (.elf) file to run on the DECA board.

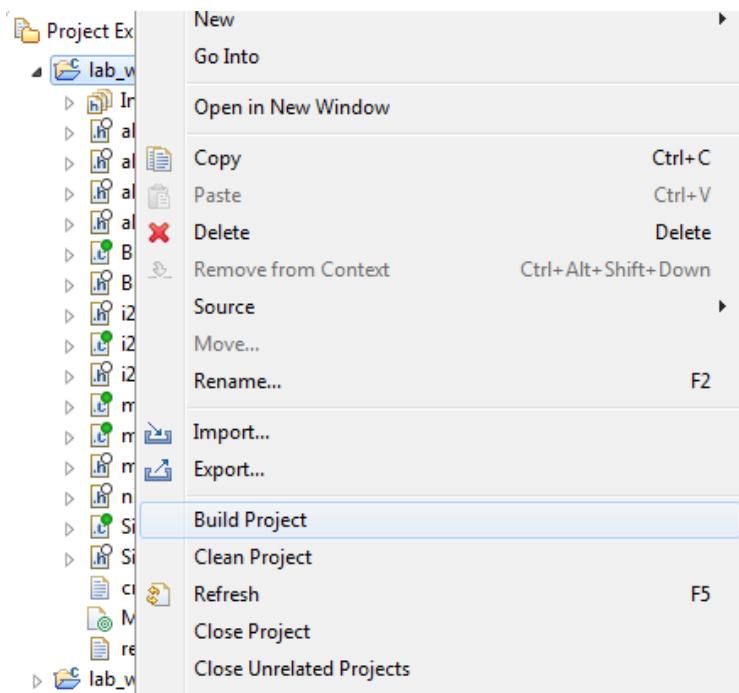
4.5.6.1 Right-click on the **lab_work_bsp** project and select Build Project from the pop-up menu to build the BSP.



You can have the build process run in the background by from the pop-up window if you wish. You can observe the process commands in the Console window.



- 4.5.6.2 Right-click on the **lab_work** project and select Build Project from the pop-up menu. Again, you have the process run in the background if you wish.



Assuming there were no errors, you have successfully compiled a Nios II software application

```

CDT Build Console [lab_work]
INFO:          86 Kbytes tree for stack + heap.
Info: Creating lab_work.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab_work.elf >lab_work.objdump
[lab_work build complete]

09:20:17 Build Finished (took 7s.639ms)

```

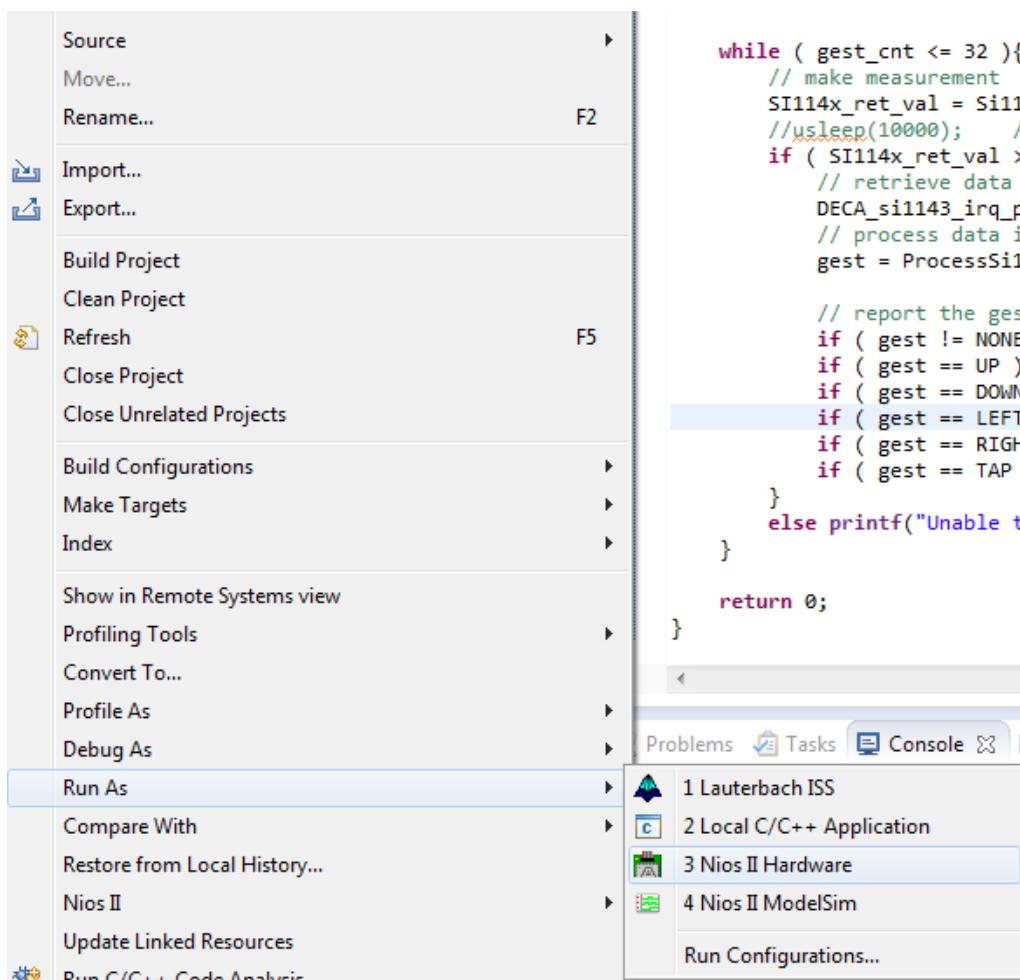
4.6 Run the Application on the DECA board

Overview: Now that you have an executable, you can download the application to the on-chip memory in the MAX10 and the Nios II processor will execute it. You will then be able to wave your hand in front of the Si1143 Gesture sensor and see your motions recorded!

4.6.1 Download the Executable to the DECA

First, a target configuration will need to be established with the DECA so that Eclipse can download the code and communicate with the board.

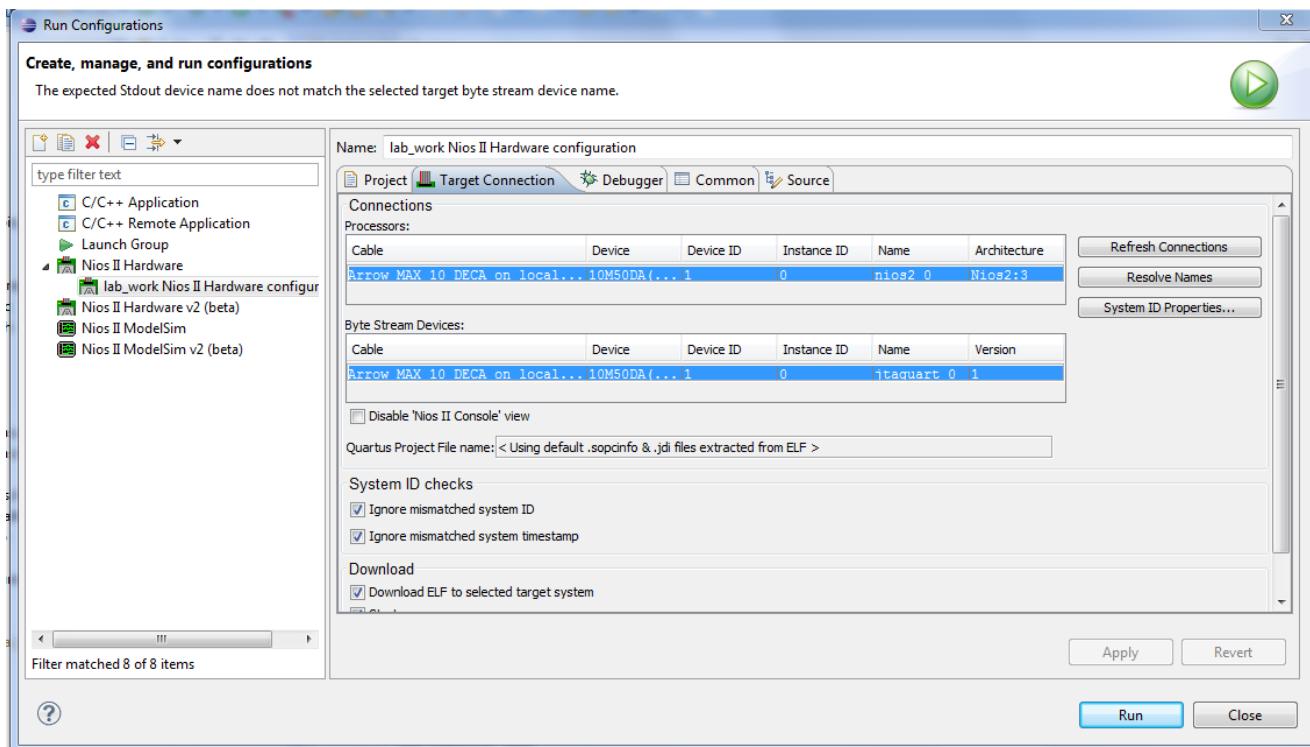
4.6.1.1 Right-click on the `lab_work` software project and select **Run As → Nios II Hardware.**



This will rebuild the software project to create an up-to-date executable and download the code into memory on the DECA. The debugger then resets the Nios II and it begins executing the code.



Note: If a Run Configuration dialogue appears, you may need to click the Target Connection tab and scroll to the right. Click "Refresh Connections" and the appropriate connection to the DECA should appear as below. Then click "Run".



- 4.6.1.2 After a few seconds, the Nios II Console should open at the bottom of Eclipse and the following initialization data should appear.

```

Problems Tasks Console Nios II Console Properties
lab_work Nios II Hardware configuration - cable: Arrow MAX 10 DECA on localhost [USB-1] de

Gesture Sensor program starting... Please wait...

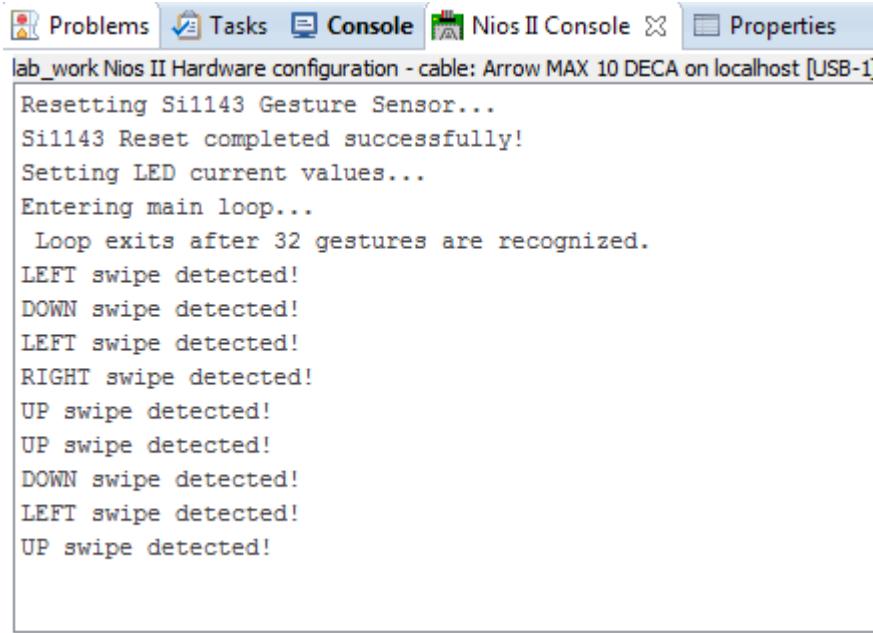
Initializing I2C bus...
    Initializing I2C at 0x81000,
        with clock speed 0x2faf080
        and SCL speed 0x30d40
        and prescale 0x31
Resetting Si1143 Gesture Sensor...
Si1143 Reset completed successfully!
Setting LED current values...
Entering main loop...
Loop exits after 32 gestures are recognized.

```

The DECA is now ready to record your gestures.

4.6.1.3 Flip the board over so that the MAX10 is facing down. The gesture sensor is configured such that what it sees as "down" is in the same direct as the USB jacks. Left, right and up are then relative to that.

4.6.1.4 Move your hand across the back of the DECA 2-3 inches away from the board at 90 degree angles to the edges of the board. Each "swipe" should last about 0.5 seconds. The console should report recognized gestures!



The screenshot shows a software interface with a top navigation bar containing 'Problems', 'Tasks', 'Console' (which is highlighted in blue), 'Nios II Console', and 'Properties'. Below the navigation bar, the title bar reads 'lab_work Nios II Hardware configuration - cable: Arrow MAX 10 DECA on localhost [USB-1]'. The main area is a terminal window displaying the following text:

```
Resetting Si1143 Gesture Sensor...
Si1143 Reset completed successfully!
Setting LED current values...
Entering main loop...
Loop exits after 32 gestures are recognized.
LEFT swipe detected!
DOWN swipe detected!
LEFT swipe detected!
RIGHT swipe detected!
UP swipe detected!
UP swipe detected!
DOWN swipe detected!
LEFT swipe detected!
UP swipe detected!
```

Congratulations! you have completed the Gesture Sensor Lab!

Max10 ADC Data Capture Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 5. MAX10 ADC DATA CAPTURE	202
5.1 Getting Started	203
5.1.1 Prepare the analog Signal Source	203
5.2 ADC Core overview.....	203
5.3 Create a Simple ADC system; no Nios required.....	205
5.3.1 Open the Quartus Project	205
5.3.2 Build the Qsys system	206
5.3.3 Complete the Qsys system	213
5.3.4 Verify your system.....	213
5.3.5 Generate the Qsys System.....	214
5.3.6 Compile the design in Quartus.....	216
5.3.7 Connect the Hardware	216
5.3.8 Test the Hardware	217
5.3.9 Verify operation using the ADC Toolkit.....	219
5.3.10 LED Volume Meter design	223
5.3.11 Debug the design using Signal Tap II	223
5.4 Create a software acquisition system, using Nios	226
5.4.1 Open the ADC Nios Lab project.....	227
5.4.2 Modify the Qsys system.....	227
5.4.3 Compile the project in Quartus	230
5.4.4 Configure the FPGA.....	230
5.4.5 Create the Nios software application	230
5.4.6 Configure the target hardware	235
5.4.7 Debug the application	236
5.4.8 Other [possibly] helpful information.....	239

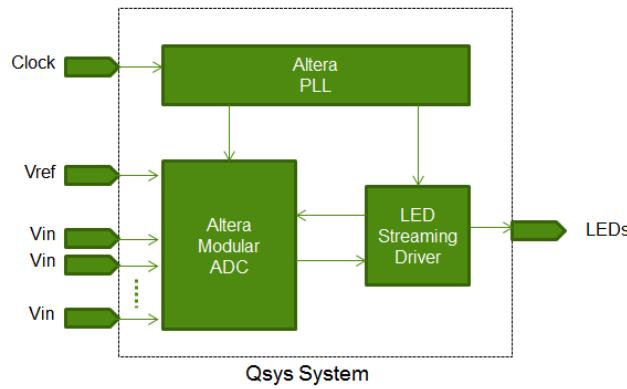
LAB 5. MAX10 ADC DATA CAPTURE

Overview: The purpose of this lab is to learn about the basic architecture and configuration options for the MAX 10 ADC.

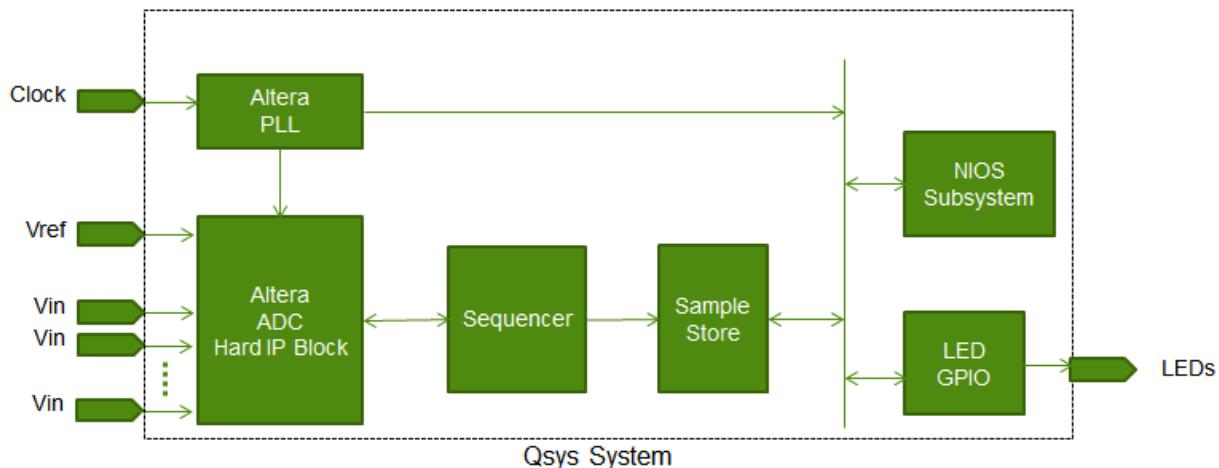
In this lab you will implement the MAX 10 ADC hard IP in a design. In the first part of the lab, you will implement a simple hardware-only streaming design. This design samples an incoming analog waveform on the LINE_IN connector, and displays the amplitude of that signal using the LEDs on the DECA kit. You will also observe the data with the ADC toolkit and with the SignalTap Logic analyzer.

For the second part of the lab, you will replace the simple hardware driver with a Nios processor system. This Nios based system will perform the same function as the hardware-only lab, but does it with software running in the Nios processor. SignalTap is also used to observe the system operation.

Simple Hardware-Only Streaming Lab Block Diagram::



Nios Processor Lab Block Diagram::



5.1 Getting Started

Overview: The first objective is to ensure that you have all of the necessary hardware items and software installed so that the lab can be completed successfully.

Below is a list of items required to complete this lab:

- Arrow DECA Evaluation Kit
- USB cable
- 2.5mm Audio cable
- Analog Signal Generator (Smart phone with Waveform App will work)
- Lab files
- Quartus II 15.0 Design Software
- Personal computer or laptop running Windows 7 with at least an Intel i3 core (or equivalent), 4 GB of RAM, and 12 GB of free hard disk space
- A desire to learn

If you are missing one of these items, please your instructor know. Instructions for how to download Quartus can be found in the Appendix.

5.1.1 Prepare the analog Signal Source

This lab requires an analog signal source. A smart phone with a Waveform generator app will work, although you should not expect good results for the linearity measurement portion of the Lab. If you are using a smart phone for this lab, please prepare it now by downloading and installing an appropriate application. For best results, choose an app that can generate sinewaves, and permits manually entering the desired frequency. **Waveform Generator Lite** and **Frequency Sound Generator** are two possible options.



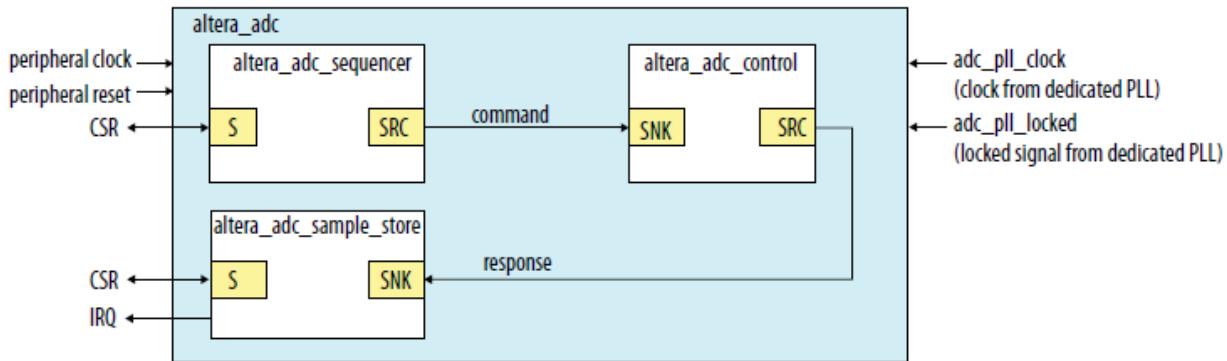
If your laptop has a headphone jack, you can use it to generate a sine wave from: <http://onlinetonegenerator.com/>

5.2 ADC Core overview

Altera supports four configuration variants of the ADC core:

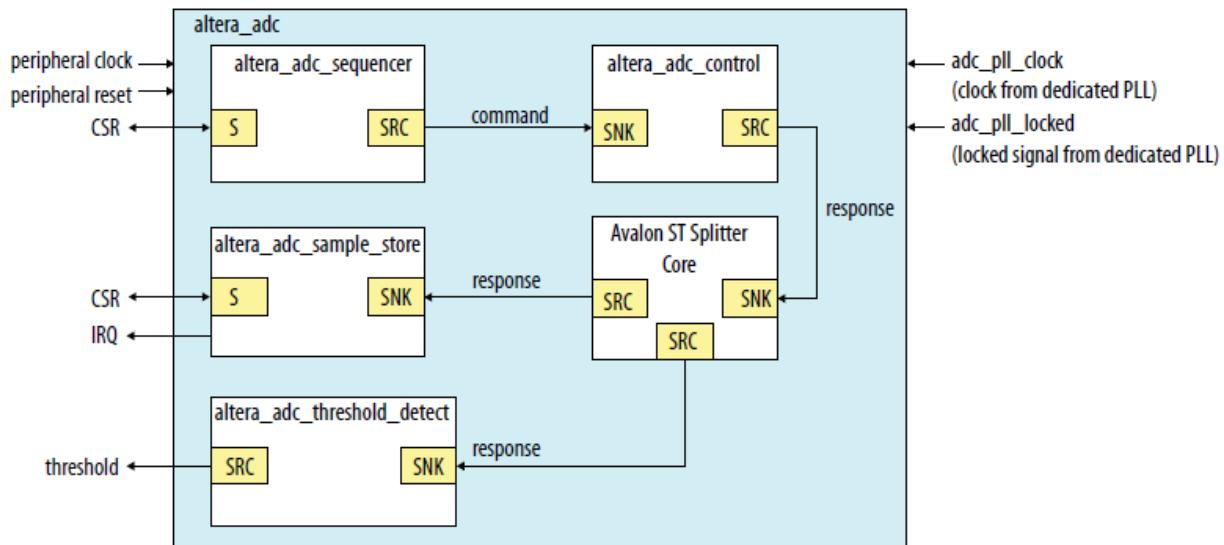
- **Configuration 1: Standard Sequencer with Avalon-MM Sample Storage**

In this configuration variant, you can use the standard sequencer micro core with internal on-chip RAM for storing ADC samples.



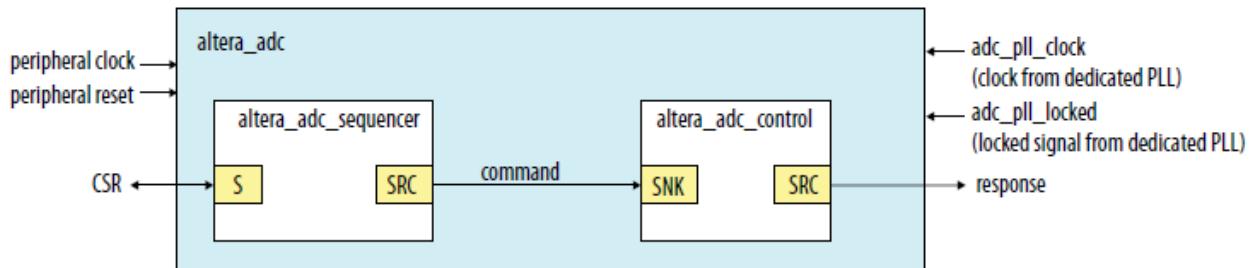
- **Configuration 2: Standard Sequencer with Avalon-MM Sample Storage and Threshold Violation Detection**

In this configuration variant, you can use the standard sequencer micro core with internal on-chip RAM for storing ADC samples with the additional capability of detecting threshold violation.



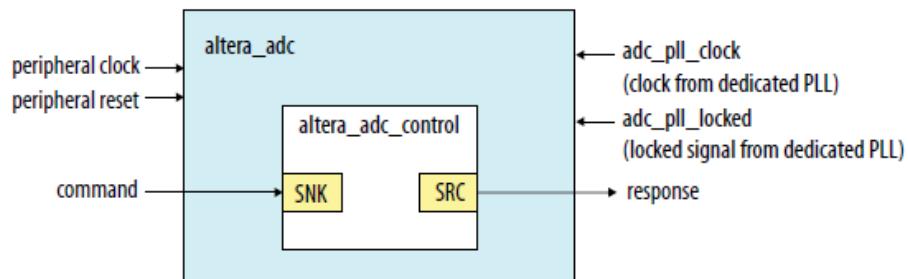
- **Configuration 3:** Standard Sequencer with External Sample Storage

In this configuration variant, you can use the standard sequencer micro core and store the ADC samples in external storage.



- **Configuration 4:** ADC Control Core Only

In this configuration variant, the Altera Modular ADC generates only the ADC control core.



5.3 Create a Simple ADC system; no Nios required

For the first part of this lab, we will be building a **Configuration 3** variant of the ADC core. In this variant, the data streams out of the ADC, and the user application handles the storing of data samples. For this lab, we will simply reformat the data with a simple Verilog application, and display it on the LEDs in real time to make a digital volume meter, without buffering.

5.3.1 Open the Quartus Project

In this section you will use an existing Quartus project, and build a Qsys system that contains a Modular ADC. The top level Verilog design file along with pin assignments has been constructed for you.

5.3.1.1 Launch Quartus, version 15.0

5.3.1.2 Open the Quartus project for this lab: **File → Open Project**

5.3.1.3 Browse to <location of lab>/5_ADC_Lab/ADC_Streaming_Lab/ADC_Streaming_Lab.qpf and Click Open

5.3.2 Build the Qsys system

5.3.2.1 Launch Qsys: Click **Tools → Qsys** (or use the Qsys tool bar icon )

When Qsys opens, you should have an almost empty system, with the exception of one **Clock Source** component. Next, you will add several more components to this system.

5.3.2.2 Save this Qsys file as **adc_qsys.qsys**

5.3.2.3 Add Clock Source Component

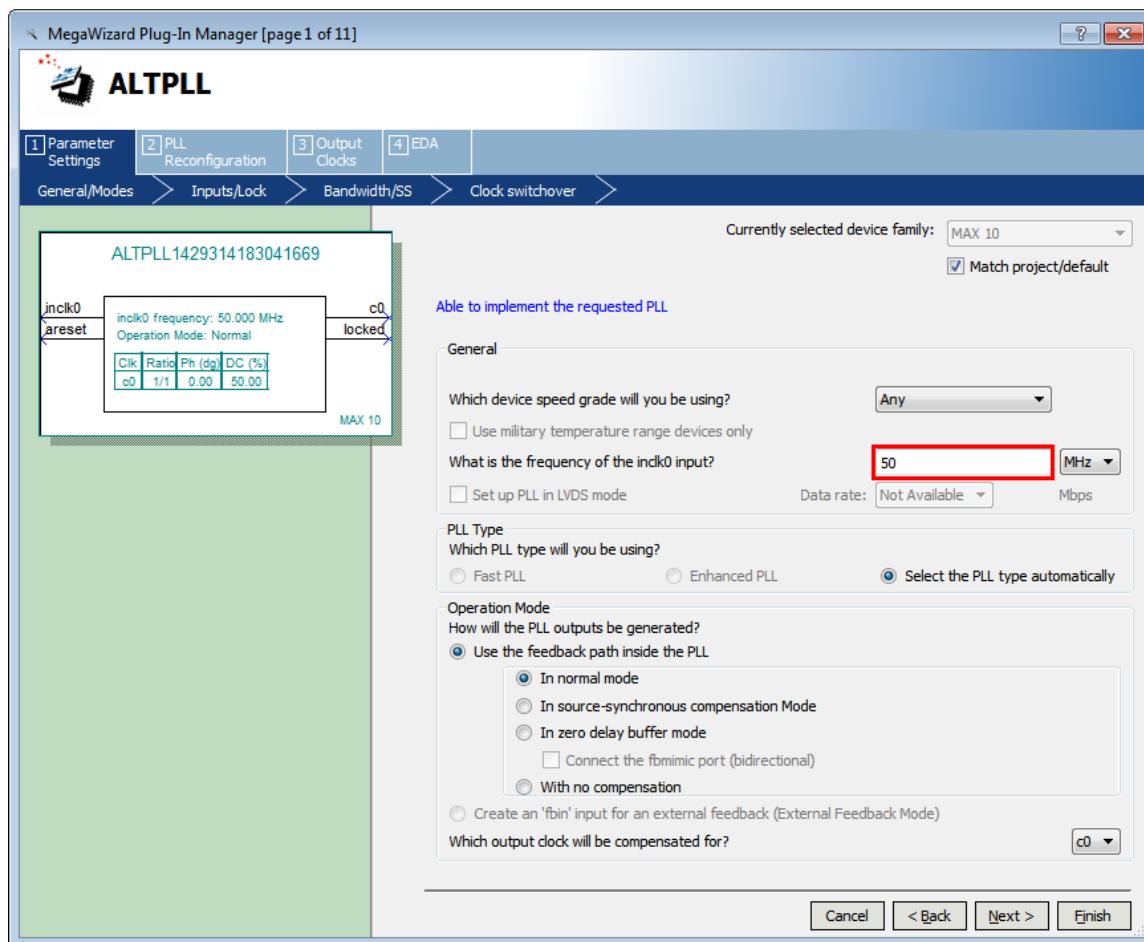
This component should already be present as a default in all new systems. If not, you can add it using the IP catalog by searching for Clock Source. The input frequency to the system is 50MHz

5.3.2.4 Add Avalon ALTPLL

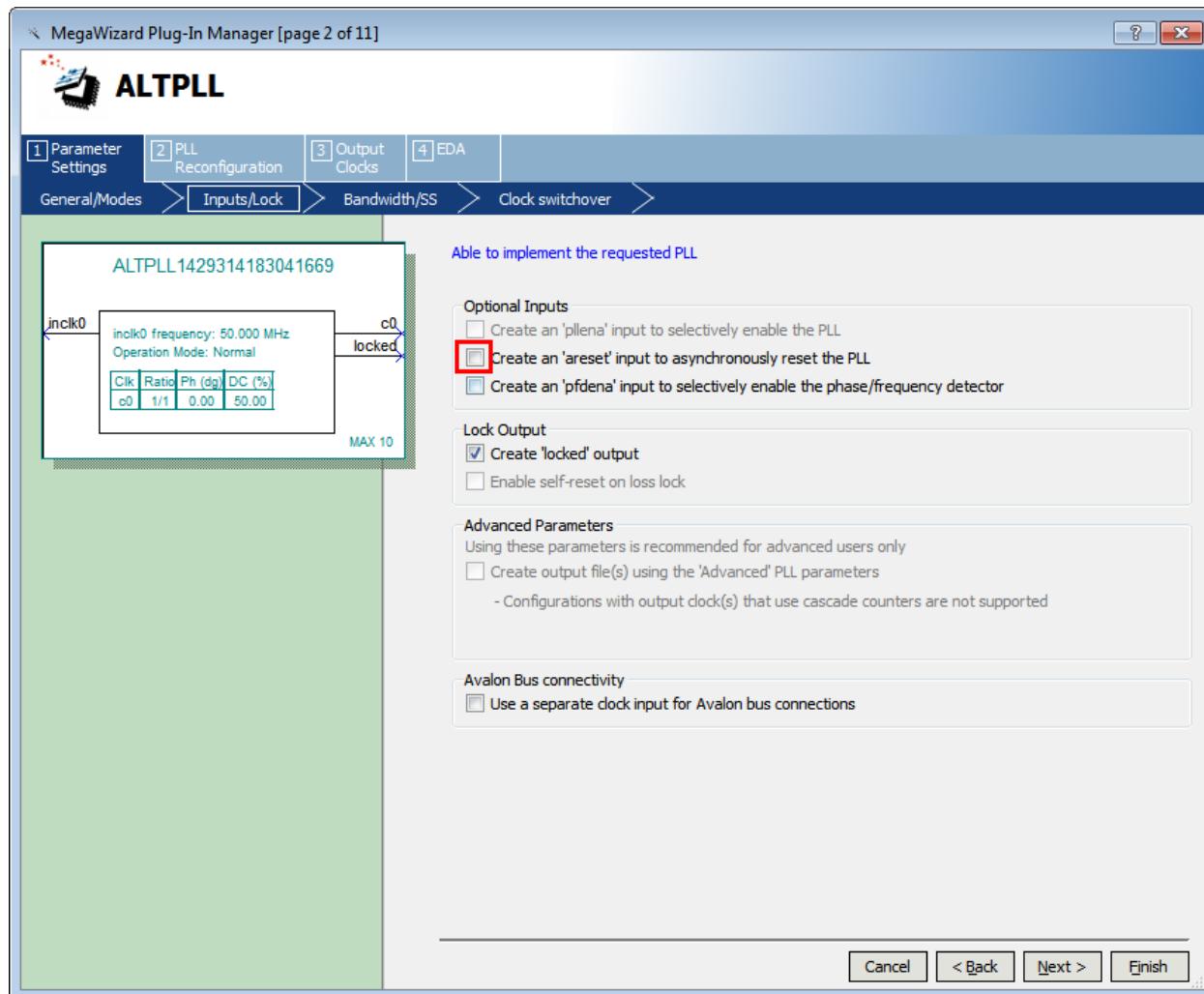
Add an Avalon PLL. This PLL will receive the board clock at 50 MHZ, and generate a 10MHZ clock for the ADC, and a 50MHz clock for the user logic.

IP Catalog → Basic Functions → Clocks; Pls and Resets → PLL → Avalon ALTPLL

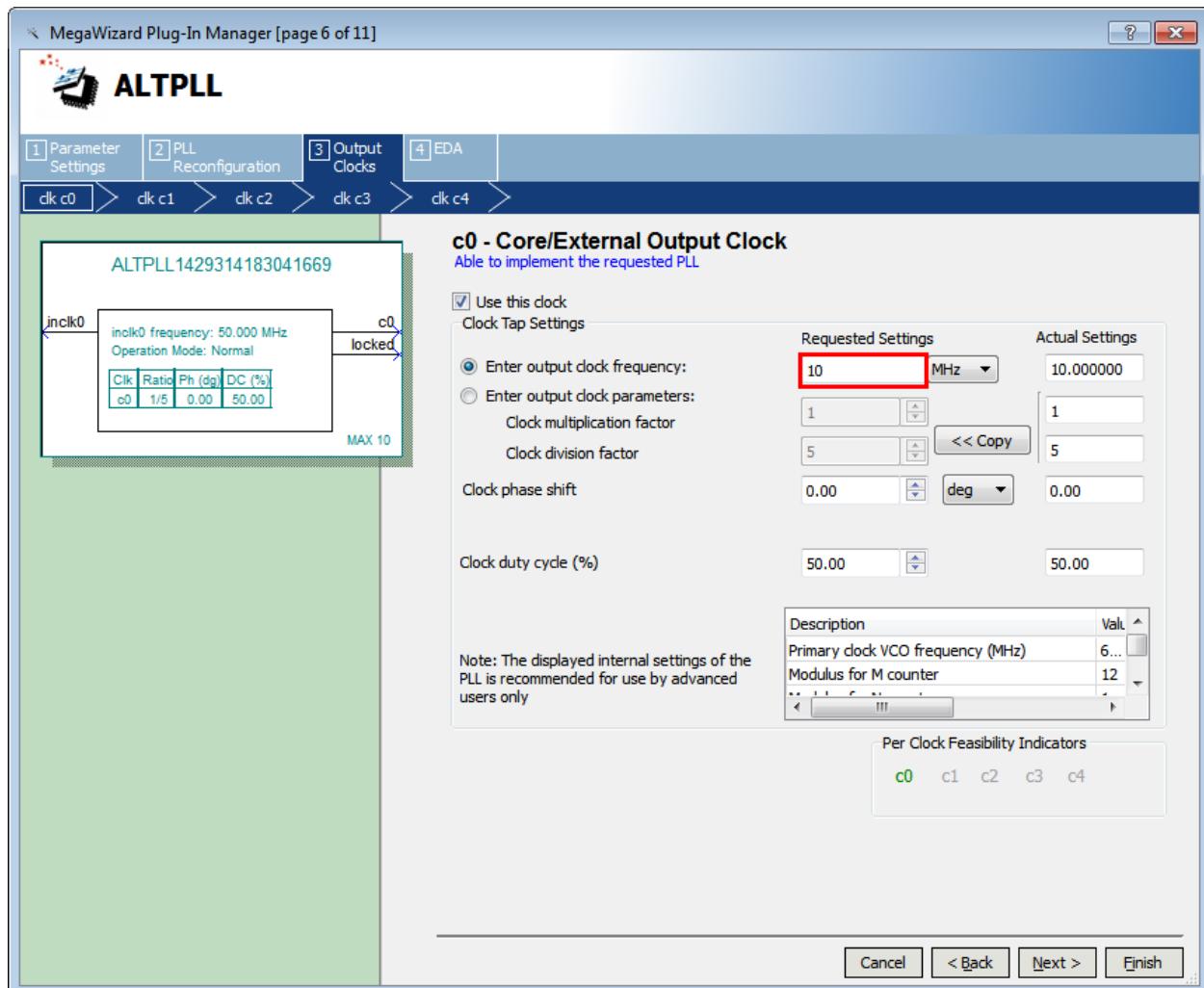
Set the input frequency to 50MHz (MegaWizard Plug-In Manager [Page 1 of 11])



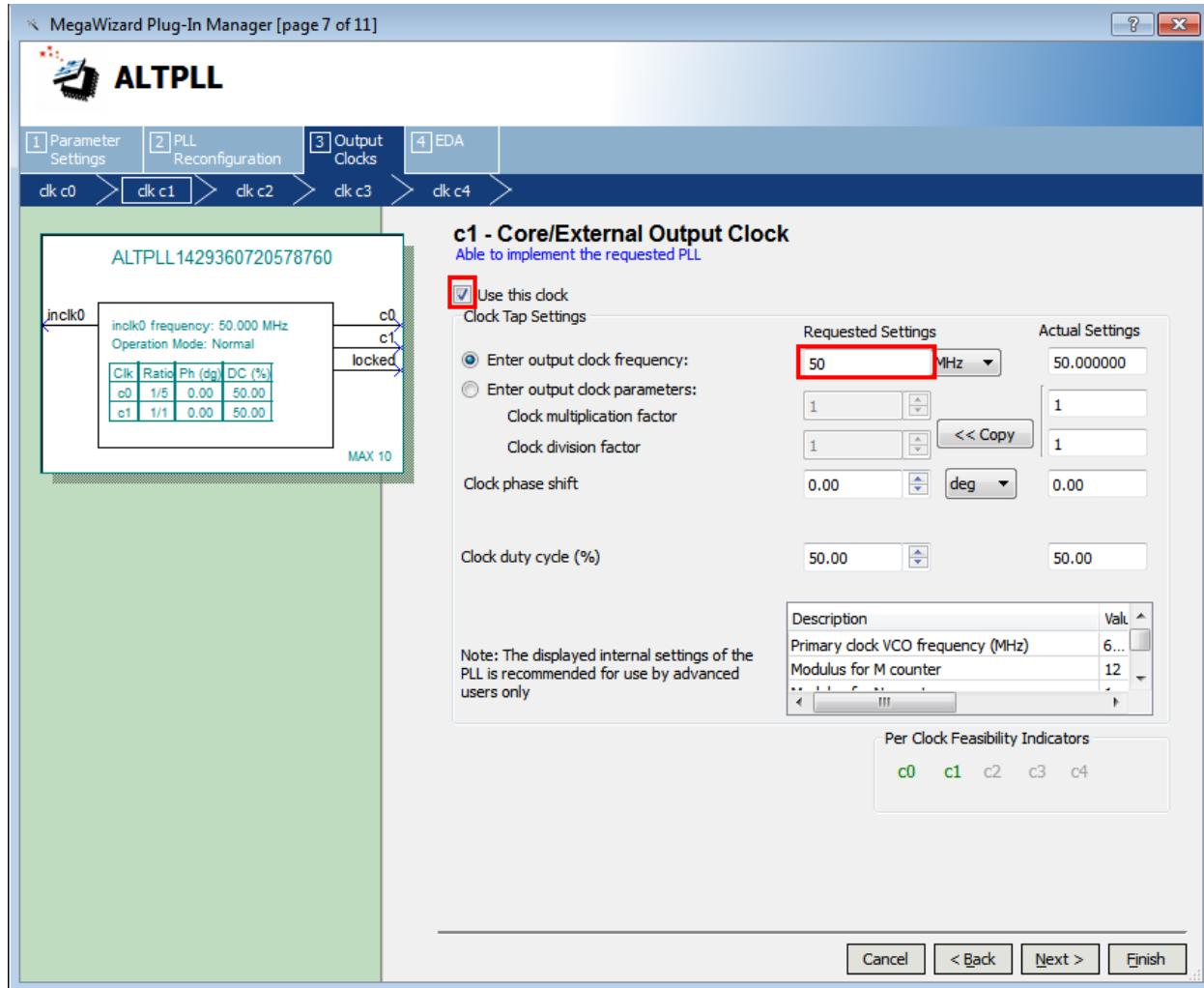
Uncheck the box for Create an 'areset' input (MegaWizard Plug-In Manager [Page 2 of 11])



Set the output frequency of clk c0 to 10 MHz (MegaWizard Plug-In Manager [page 6 of 11])



Set the output frequency of clk c1 to 50 MHz (MegaWizard Plug-In Manager [page 7 of 11])

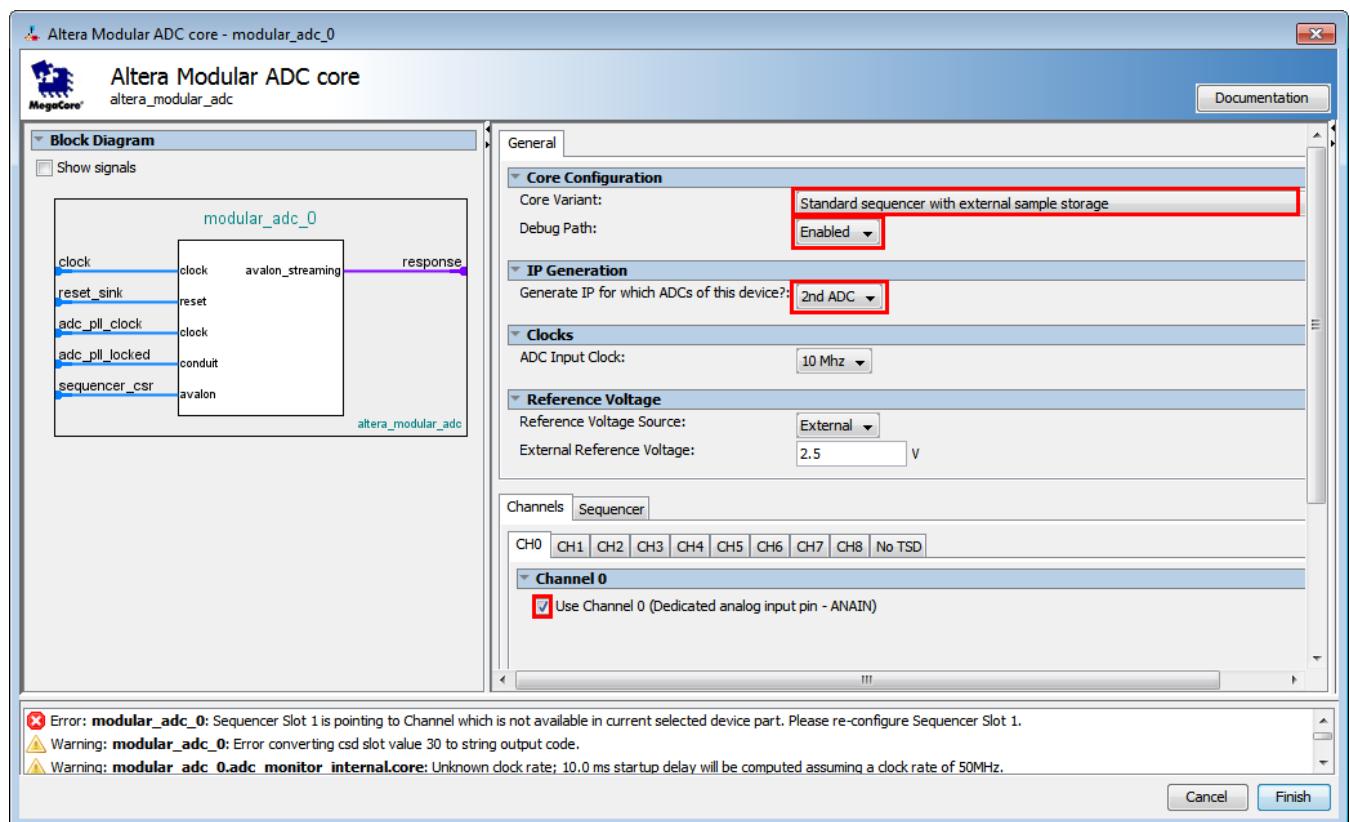


Click Finish, and then Finish again to complete the PLL configuration.

5.3.2.5 Add the Modular ADC Core from the IP Catalog.

Locate the ADC core under Processors and **Peripherals** → **Peripherals** → **Altera Modular ADC Core**, or simply type ADC in the search box. Click Add

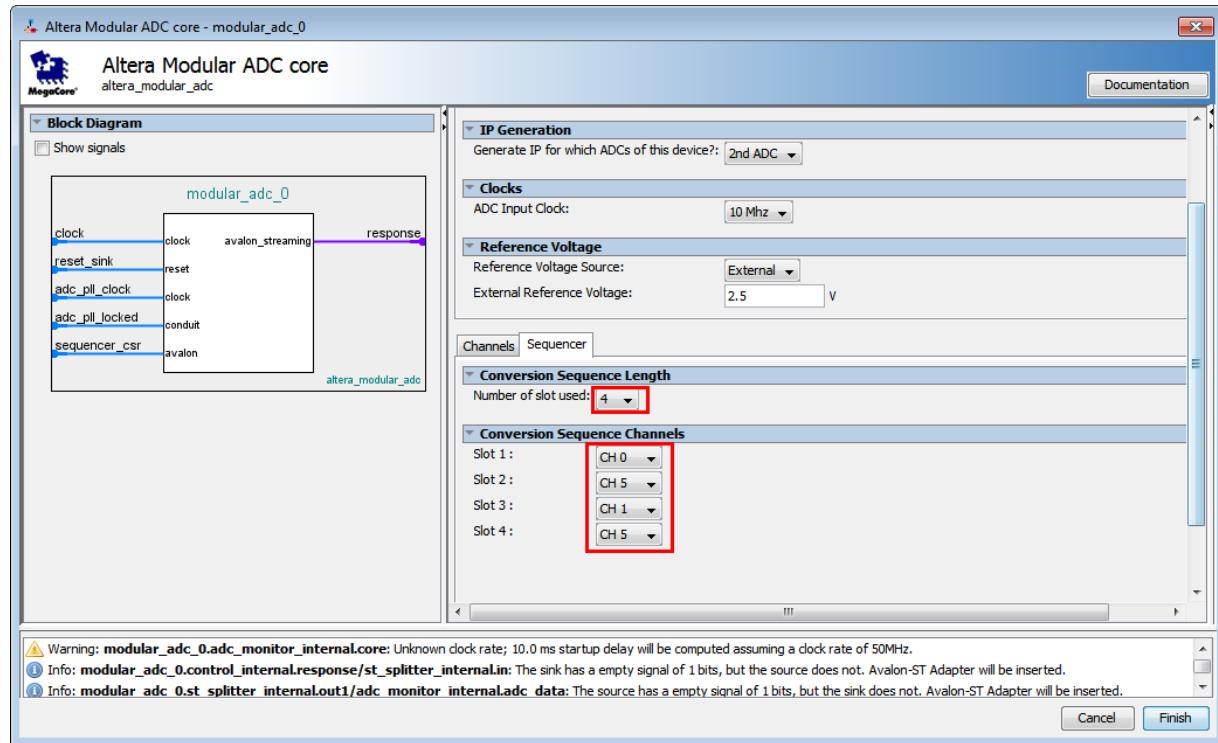
- Set the Core Variant to **Standard sequencer with external sample storage**
- Set the IP for **2nd ADC**. The line input is connected to Channel 5 of ADC 2. Note all other inputs on the DECA board are fed to ADC 1
- On the Channels Tab, check the boxes to use **Channel 0, Channel 1, and Channel 5**. The Audio signal is connected to ADC 2, Channel 5. To demonstrate the sequencer, we are also enabling channel 0 and 1, but there will be no input signal on these channels.



- On the Sequencer Tab, select the number of slots to use as 4
- On the Sequencer tab, set the conversion sequence as follows:

Slot 1	CH 0
Slot 2	CH 5
Slot 3	CH 1
Slot 4	CH 5

- Verify your configuration matches the screenshot below, and click Finish



5.3.2.6 Add the LED_Streaming_Driver.

This is a custom component that is located in the DECA Labs folder of the IP Catalog. There are no configurable options for this component, so just click Finish

If you want to inspect the verilog code for this component, it is located in the IP directory. In summary, this component has three interfaces:

- An Avalon memory mapped master for writing commands to the ADC sequencer control and status register. After coming out of reset, the LED Driver performs a single write to the ADC sequencer to put the ADC in continuous acquisition run mode.
- An Avalon streaming sink to receive the data from the ADC "response" streaming source. The LED driver receives the data and channel information, so it can ignore other channels, if they are enabled. The Audio LINE_IN signal is on channel 5.
- An exported conduit that drives the board LEDs. The value output is based on the analog level received from the ADC

5.3.3 Complete the Qsys system

At this time, you should have four components in the system, with 7 Errors in the messages tab. Now we are going to connect the components together.

5.3.3.1 Connect the clock, data, and IRQ connections.

Using the screenshot below, connect the Qsys components. Pay careful attention to the clock connection to the ADC core. Note in the screen shot, the "Show clock domains in the system table" is enabled. This option displays each clock domain in a different color, and can be helpful in identifying unintentional clock domain crossings.

5.3.3.2 Name the Qsys components.

Right click on the component name (or use keyboard shortcut Ctrl-R). The names of the components must be entered exactly as shown if you want to use the SignalTap portion of this lab. The names should be:

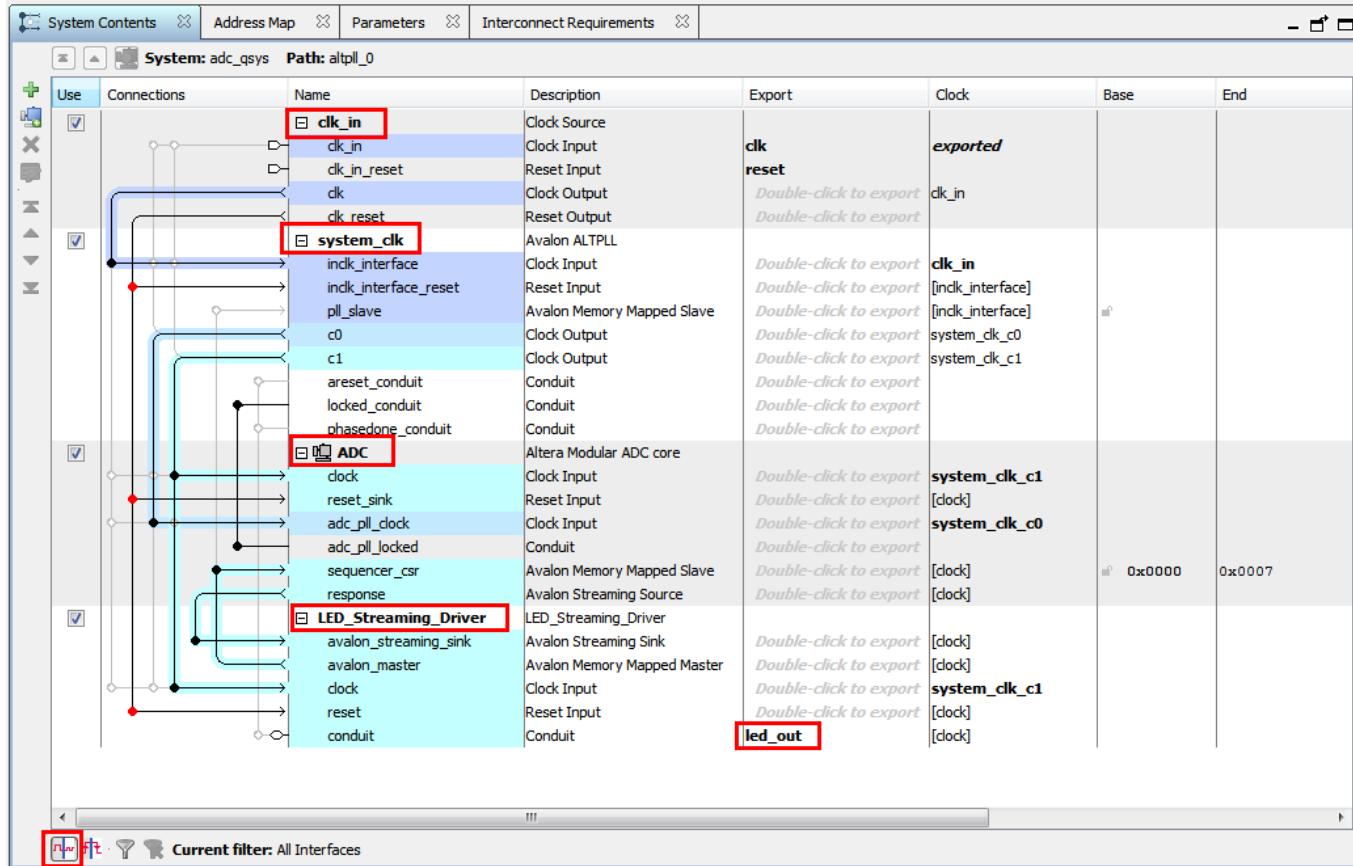
- `clk_in`
- `system_clk`
- `ADC`
- `LED_Streaming_Driver`

5.3.3.3 Export the LED bus.

Double click in the export column to export the conduit out of the LED driver. This will connect to the LED pins at the top level. Name this conduit `led_out`

5.3.4 Verify your system

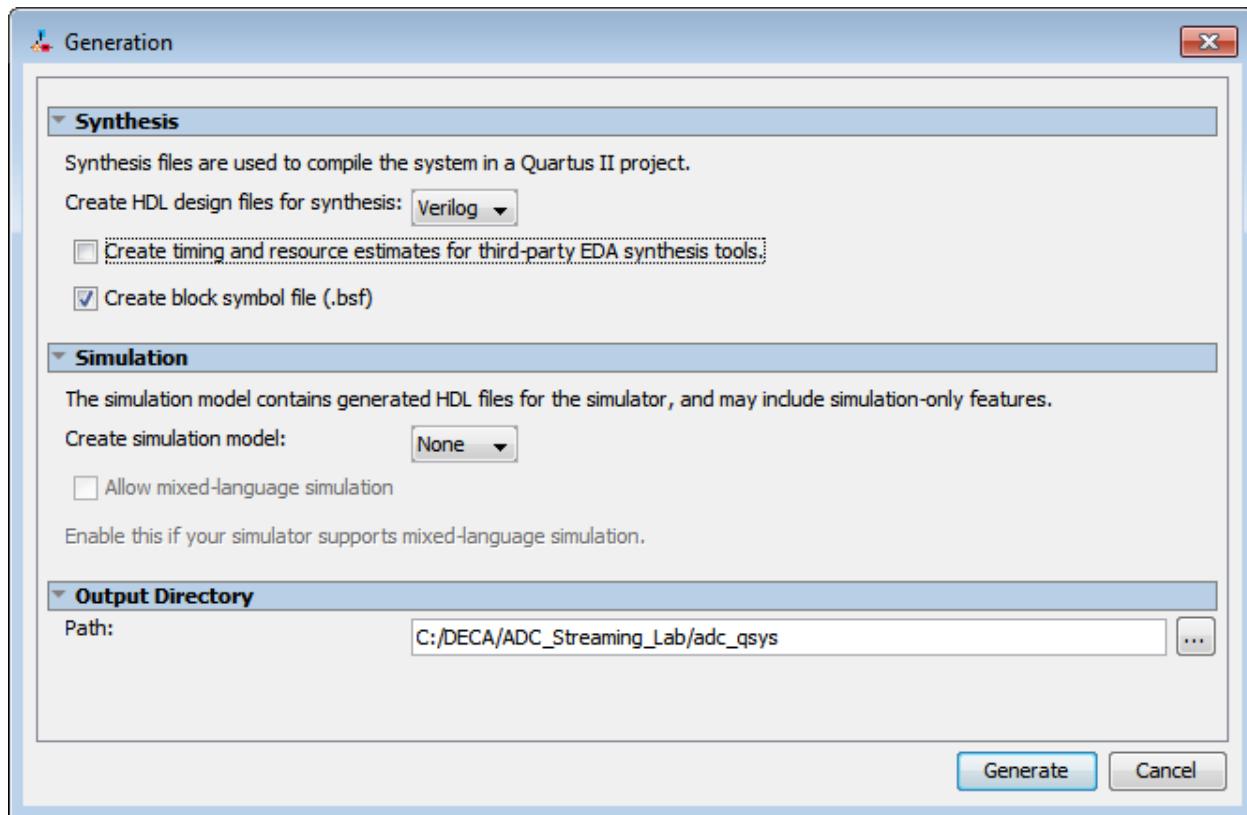
When complete, your system should look like the screenshot below: You will still have 3 warnings about ports that have been left unconnected. You can safely ignore these warnings.



5.3.5 Generate the Qsys System

If your system matches, click Generate HDL to generate the system. Choose Verilog as the language for synthesis.

Set the output Directory path to <project Directory>/adc_qsys and click generate.



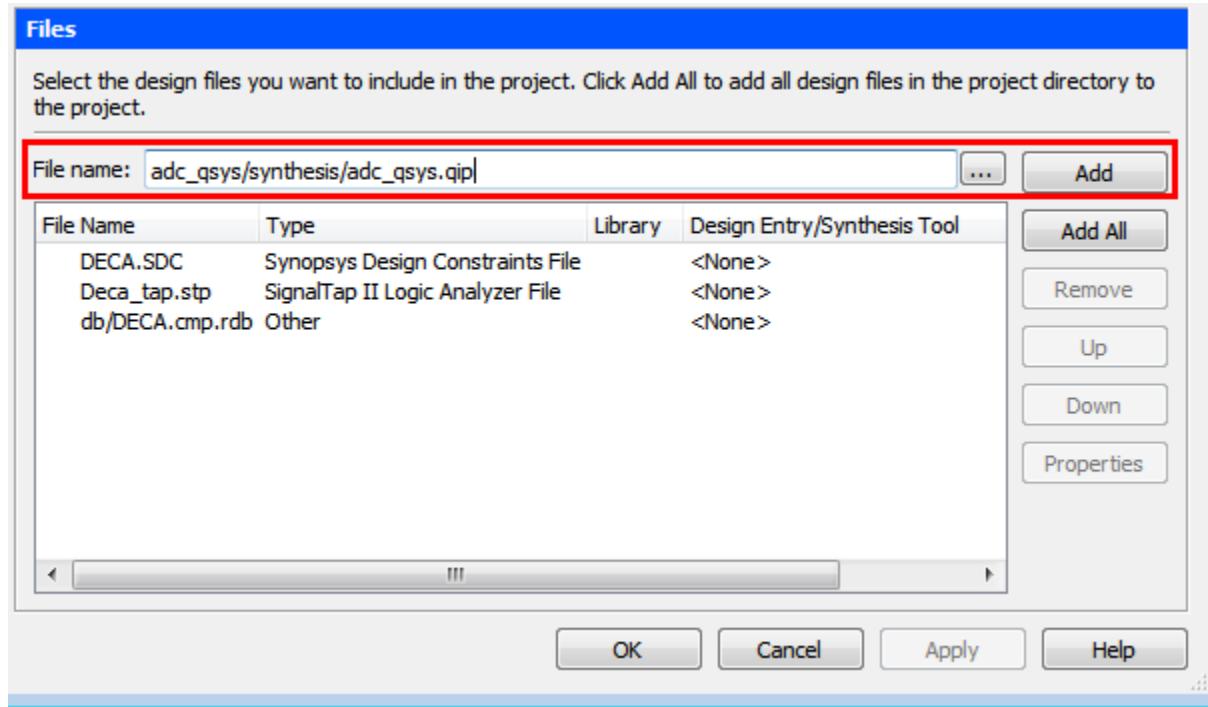
The Generation process window will appear, and the progress will be displayed; generation takes a few minutes for this system. When complete, click Close.

5.3.6 Compile the design in Quartus

5.3.6.1 Add the Quartus IP file (.qip) file to the project: **Project → Add/Remove Files to Project**

Note: When you finished generating the Qsys system, you may have seen a dialog box that reminded you to add the Quartus IP file to the project. That is what we are doing now.

Click on the  button to browse to the `adc_qsys/synthesis/adc_qsys.qip` file. Don't forget to click **Add** after selecting the file. Click OK.



5.3.6.2 Start the Compile: **Processing → Start Compilation** (or click the compile icon on the toolbar)

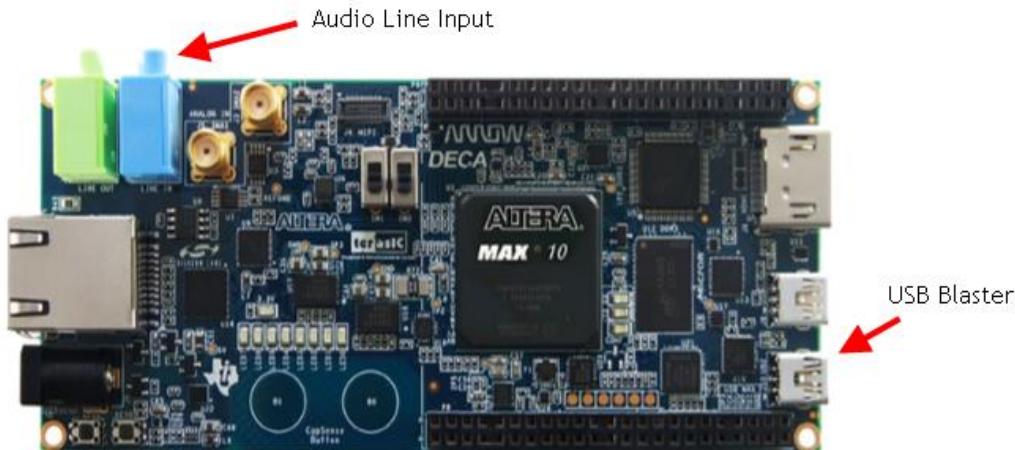
5.3.7 Connect the Hardware

5.3.7.1 Connect the Audio Cable

Connect the 2.5mm cable from the signal generator (smart phone) to the Line in jack of the DECA board. This is J2 on the DECA board (blue jack).

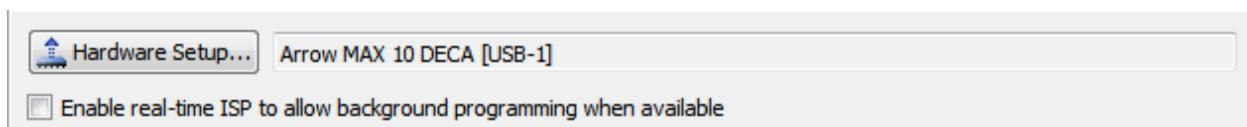
5.3.7.2 Connect the USB Cable

Plug the USB cable into the on-board USB blaster II. This is J10 on the board



5.3.8 Test the Hardware

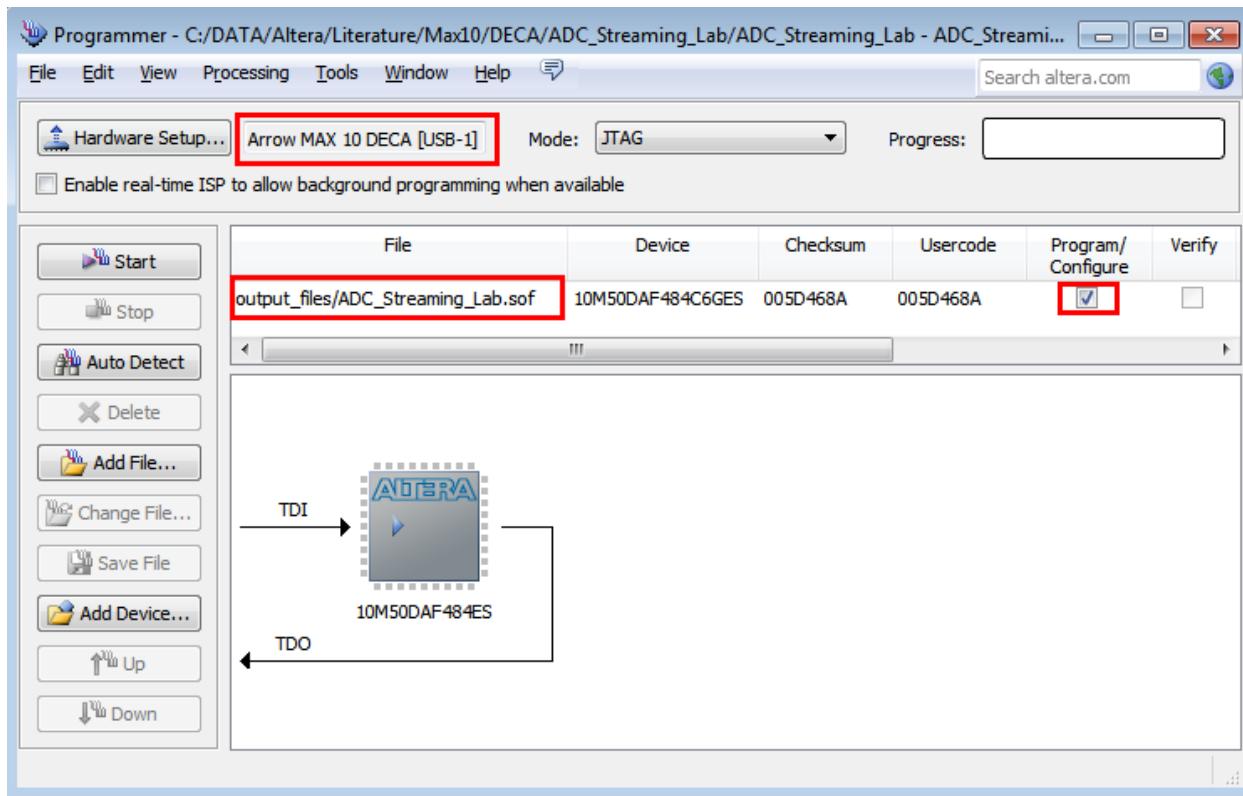
5.3.8.1 Open the Quartus Programmer: **Tools → Programmer**, or click the programmer icon  in the toolbar
 Confirm the Programming hardware: Arrow MAX 10 DECA should appear in the programming hardware window. If it says “No Hardware”, the programmer hardware will need to be configured. See the Appendix, **Chapter 9.3 Configure the Quartus the programmer** for detailed instructions.



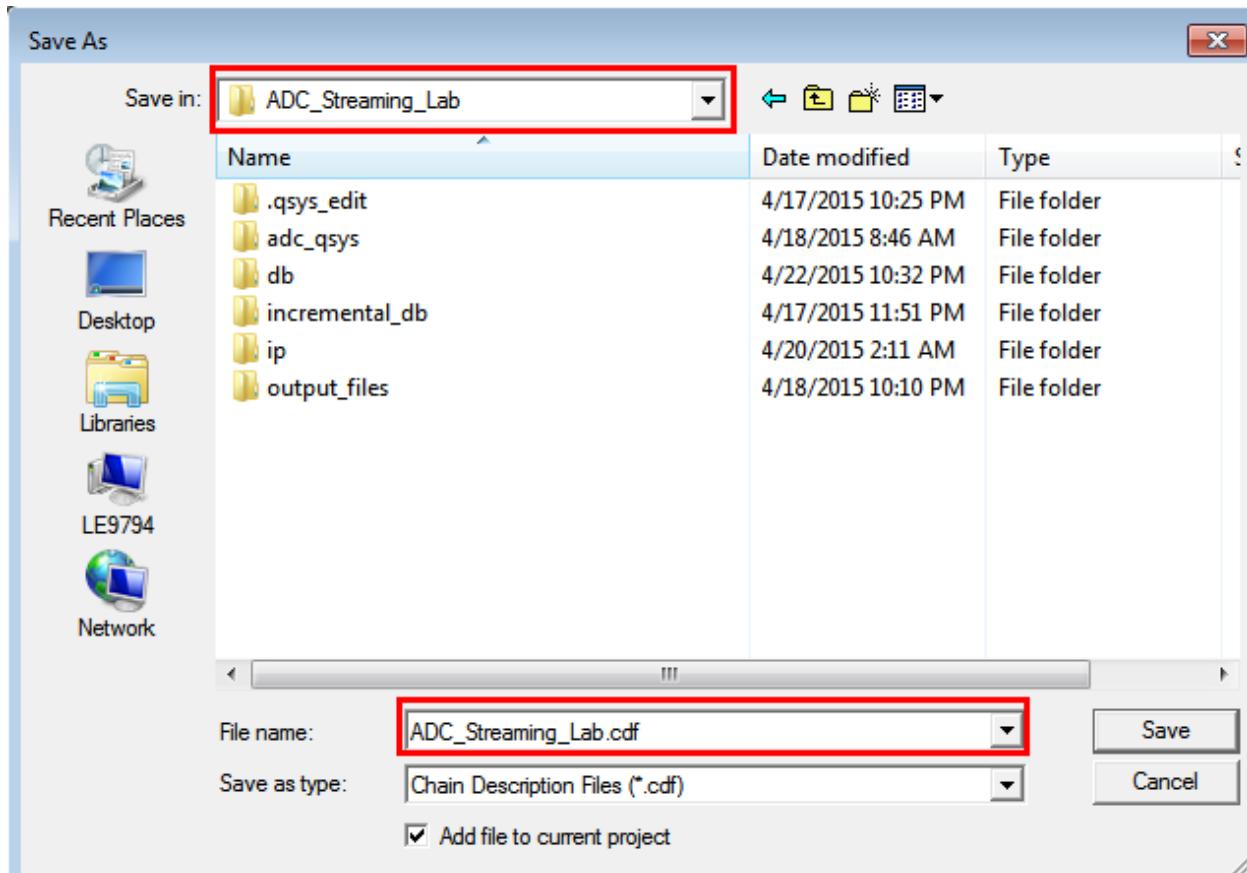
5.3.8.2 Add the programming file: click Add file, and browse to the **ADC_Streaming_Lab.sof** file in the **output_files** directory.

5.3.8.3 Verify the Program / Configure box is checked.

If your screen matches the one below, **press the Start button**. Programming is very fast (~2 seconds). Confirm the Progress is 100% (Successful)



- 5.3.8.4 Save the programming configuration: **File → Save As**. If you name the configuration file with the same name as the project, it will automatically open when you open the programmer. Browse to the project directory, name the file: **ADC_Streaming_Lab.cdf**, and click Save to continue.



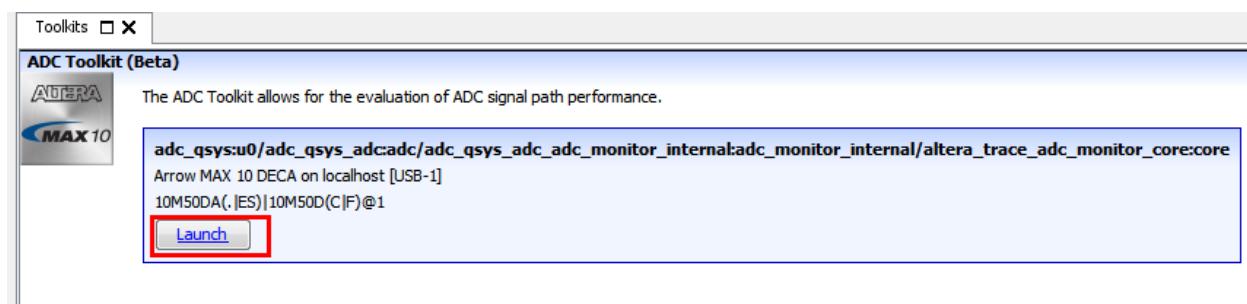
5.3.9 Verify operation using the ADC Toolkit

- 5.3.9.1 Open system console: in Quartus, **Tools → System Debugging → System Console**. Or alternatively, from Qsys: **Tools → System Console**

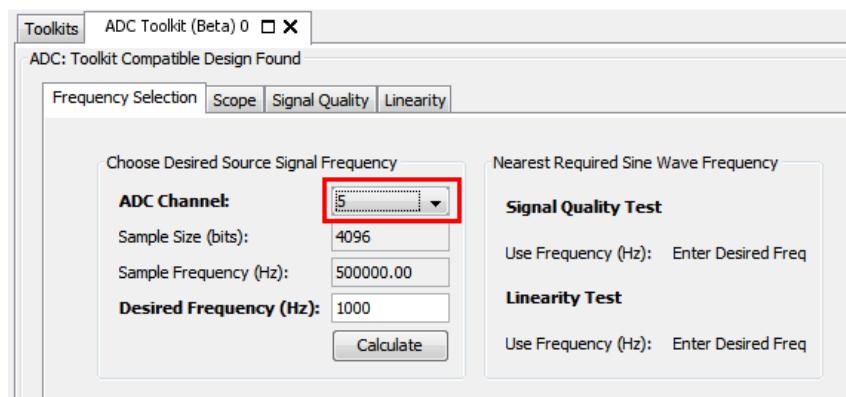
- 5.3.9.2 Launch the ADC tool kit by pressing the Launch button.



NOTE: if Launch does not become available by default you will need to click “Load Design” and point to the SOF file that is configured into the device. System Console will use SOF file data to match with JTAG debug target in the FPGA



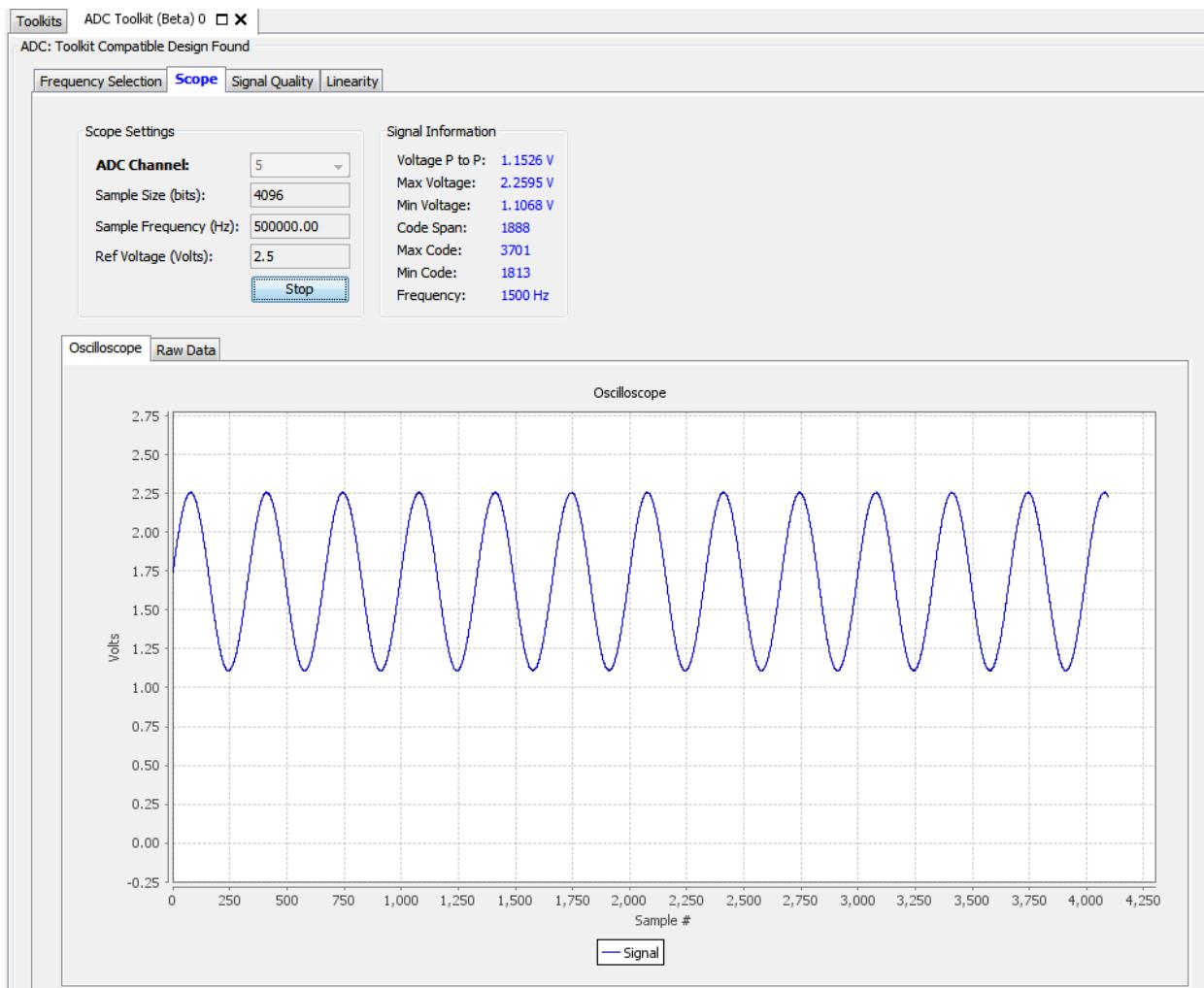
- 5.3.9.3 In the Frequency Select tab, select the ADC channel to evaluate. The Line-In jack is connected to channel 5, so select channel 5. Press Calculate. This will tell you the frequencies required for the Signal Quality Test and the Linearity Test.



5.3.9.4 Go to the **Scope** tab.

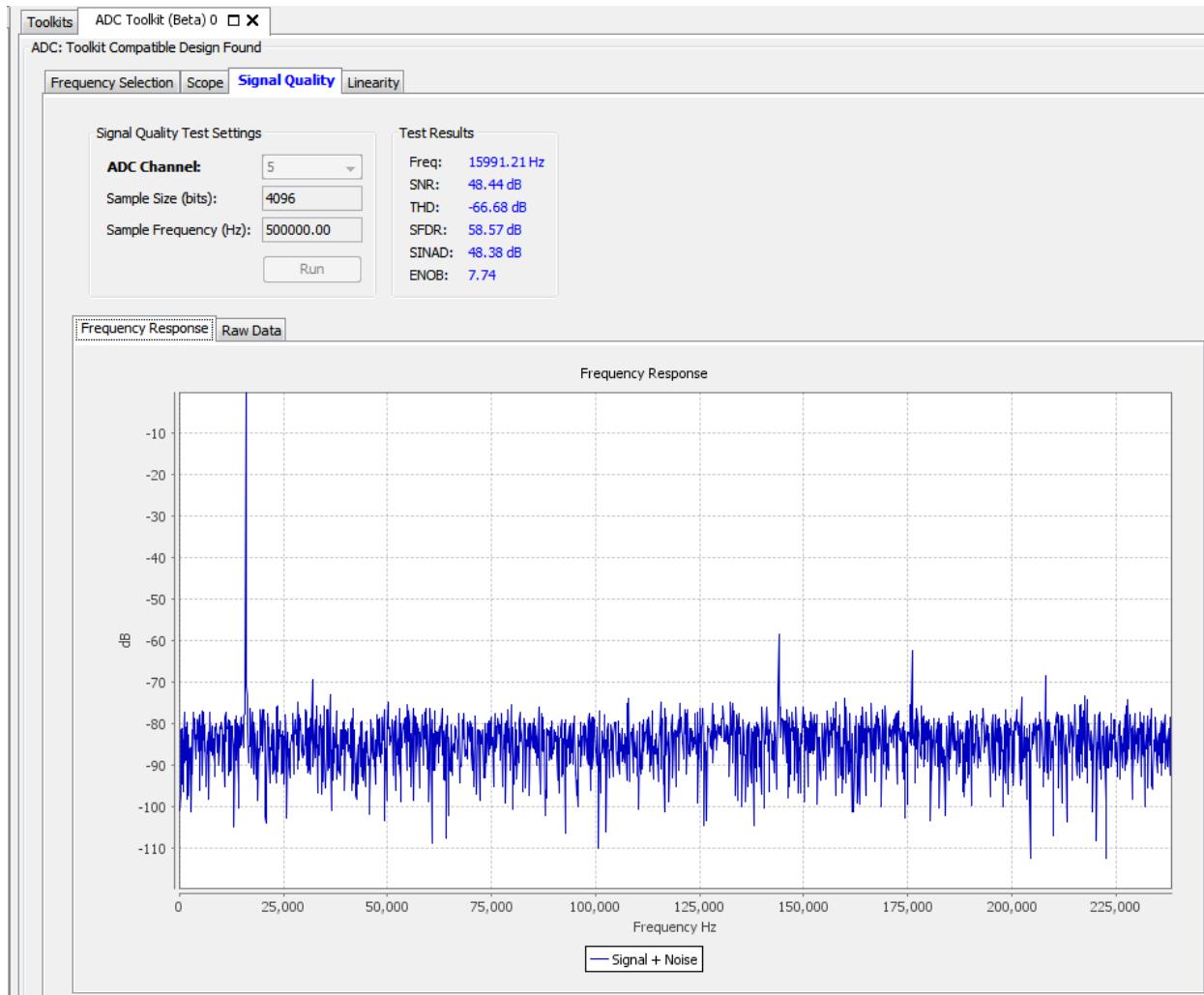
5.3.9.5 Confirm that ADC Channel 5 is selected. Click Run to begin acquiring data from the ADC

5.3.9.6 Start the Signal Generator (or Waveform Generator app on your smart phone). Select a waveform to output, and you should see the waveform appear on the Oscilloscope window.

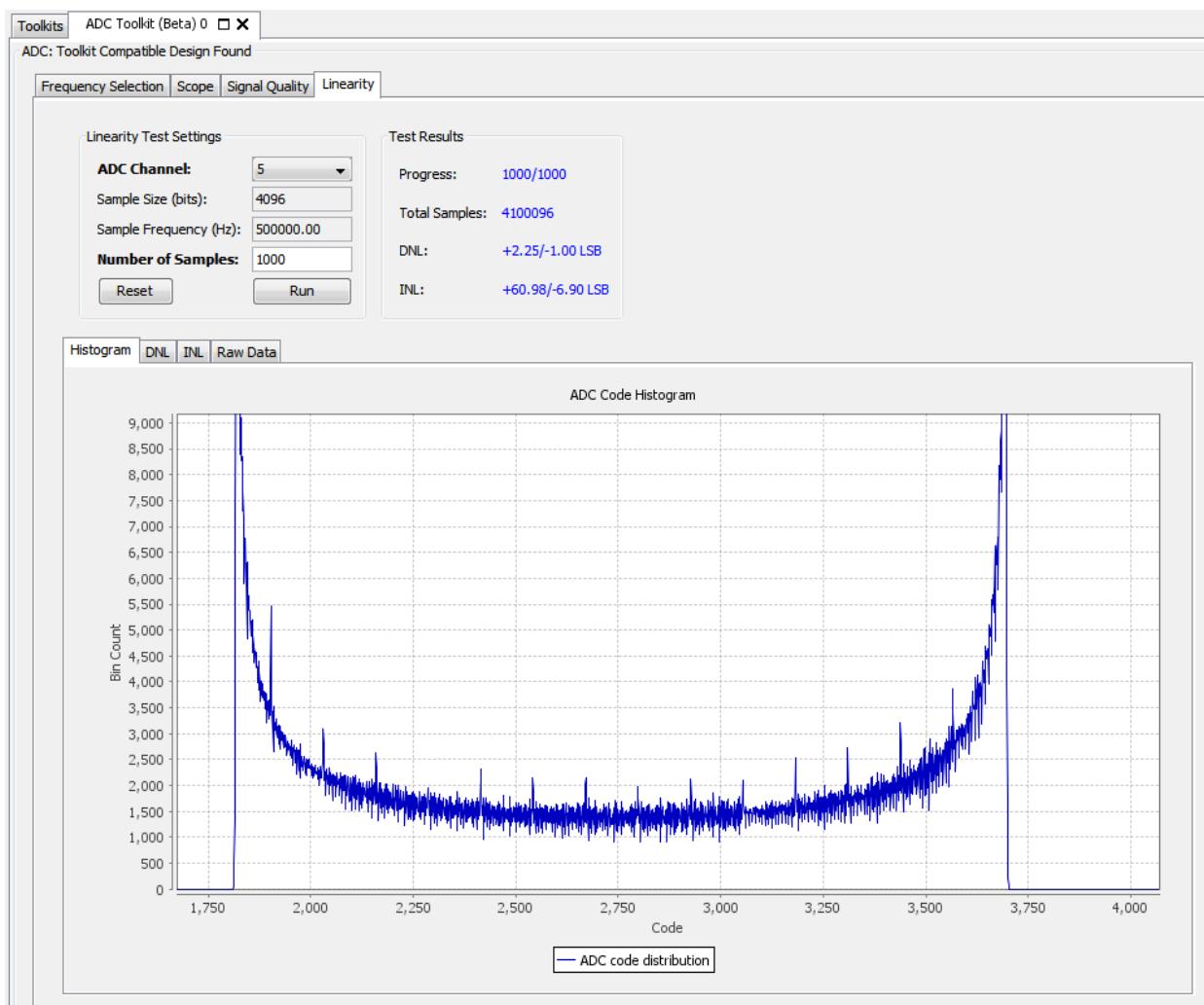


5.3.9.7 Click Stop to move to the next section

5.3.9.8 Switch to the Signal Quality tab. Click Run to perform a Frequency Response Measurement: To zoom, click and drag a portion of the graph, i.e. grab the top left and drag to the region edge on the right to zoom. If you want to see the raw data that was measured for this plot, click on the Raw Data tab. Note for these measurements, it is critical that the frequency output from the signal source matches exactly the frequency calculated by the ADC toolkit. Since this is difficult without lab quality equipment, datasheet specifications should not be expected.



- 5.3.9.9 Switch to the Linearity tab. If you have a sine wave running, you will see a bathtub curve, showing the number of samples collected at each ADC code. For an ideal sinewave, this will be a smooth curve. The smart phone is not an ideal source.



5.3.10 LED Volume Meter design

The design we built contains a streaming LED driver, which decodes the ADC value into an LED bar graph in real time. As you adjust the amplitude on your waveform generator, you may have noticed the LED bar graph changing in amplitude. Adjust the volume now to confirm the functionality. In the design, we have two LEDs on as the "off" condition, so even with no sound, you should see two LEDs.

- 5.3.10.1 If you are using a smart phone as your signal source, switch to a music playing app, and you will see the LEDs respond with a simple "VU meter" display.

5.3.11 Debug the design using Signal Tap II

Max 10 is the first CPLD from Altera to support SignalTap, Altera's embedded logic Analyzer. As part of the lab you opened, a completed SignalTap system was included. In this section we will explore some of the basic SignalTap capabilities.

5.3.11.1 Open SignalTap: Tools → SignalTap II Logic Analyzer

If you have already programmed the device, the system status should be "Ready to Acquire". If not, program the device using the Quartus programmer, or use the programmer built into SignalTap.



NOTE: In some cases, Quartus II Web-Edition will throw an error when launching SignalTap. Altera requires that you enable the TalkBack feature in order to use the SignalTap II Logic Analyzer.

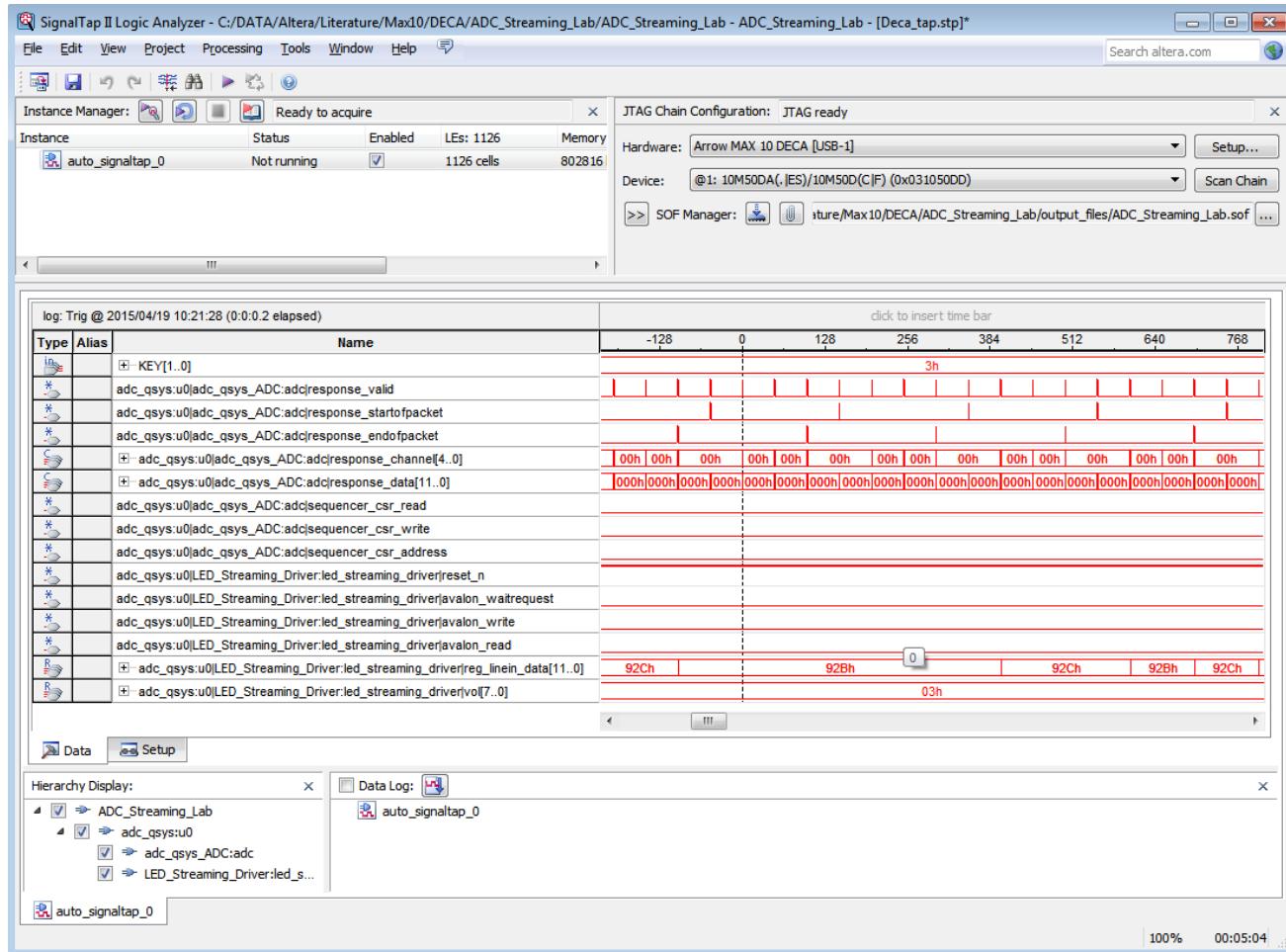
If so, follow these steps:

Go to the menu: **Tools → Options** and select the Category: Internet Connectivity. In the right-hand pane, click the TalkBack Options... button then check the **Enable sending TalkBack data to Altera** checkbox.

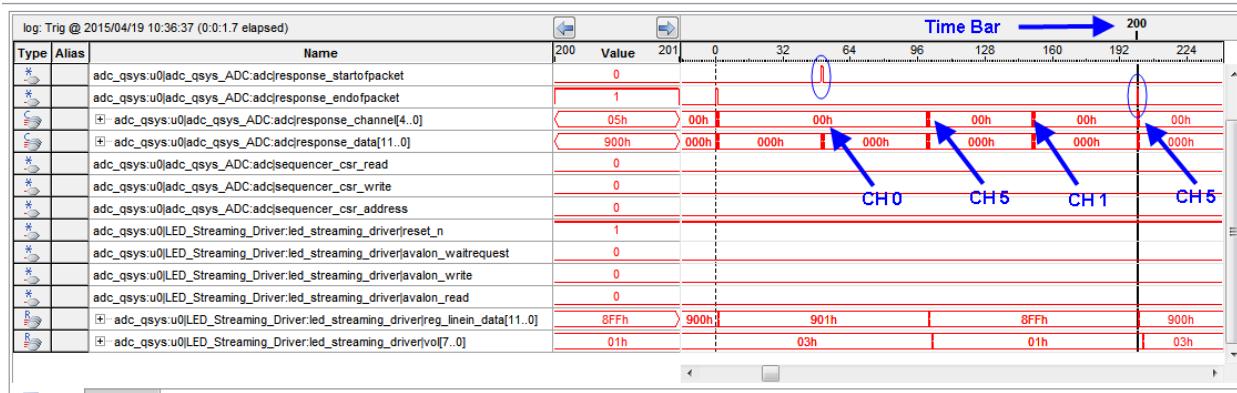
Click OK twice to close both the Quartus II TalkBack dialog box as well as the Options dialog box.

5.3.11.2 Start the acquisition: Processing → Run Analysis, or by clicking on the Run Analysis icon in the toolbar.

The SignalTap system included with the lab has been configured to trigger on a valid Channel 5 data response from the ADC. If your ADC is running correctly, the SignalTap system will trigger instantly, and display the captured data. The result should look something like this.



5.3.11.3 Observe the ADC sequence: response_startofpacket is asserted to indicate the start of a sequence, and the response_channel bus provides the channel number. Insert a time bar by clicking above the timeline to see the data values on the bus without having to zoom in.



5.3.11.4 Change the trigger condition. Most changes to SignalTap require a full recompile of the system. Some changes, like the trigger condition, can be made without recompiling. Let's change the trigger to look for a specific data value. Go to the Setup Tab, and enter E00 as the trigger condition for the reg_linein_data.

If you want to ensure you don't accidentally make a change that requires a full recompile, set the Lock mode to "Allow trigger condition changes only".

trigger: 2015/04/19 10:28:44 #0			Lock mode:	Allow all changes	
Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
*		adc_qsys:u0 adc_qsys_ADC:adc response_startofpacket	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 adc_qsys_ADC:adc response_endofpacket	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
C		+ adc_qsys:u0 adc_qsys_ADC:adc response_channel[4..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	05h
C		+ adc_qsys:u0 adc_qsys_ADC:adc response_data[11..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXh
*		adc_qsys:u0 adc_qsys_ADC:adc sequencer_csr_read	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 adc_qsys_ADC:adc sequencer_csr_write	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 adc_qsys_ADC:adc sequencer_csr_address	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver reset_n	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver avalon_waitrequest	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver avalon_write	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver avalon_read	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		+ adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver req_linein_data[11..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	E00h
R		+ adc_qsys:u0 LED_Streaming_Driver:led_streaming_driver vol[7..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXh

5.3.11.5 Run Analysis again

NOTE: if the Run Analysis option is greyed out, confirm that you have an instance selected in the Instance manager window.



This time, the system may not trigger immediately, depending on the input level from your signal source. Turn the volume up, and the system should trigger. If not, try a lower trigger value, like 900h

That completes this portion of the Lab.

5.4 Create a software acquisition system, using Nios

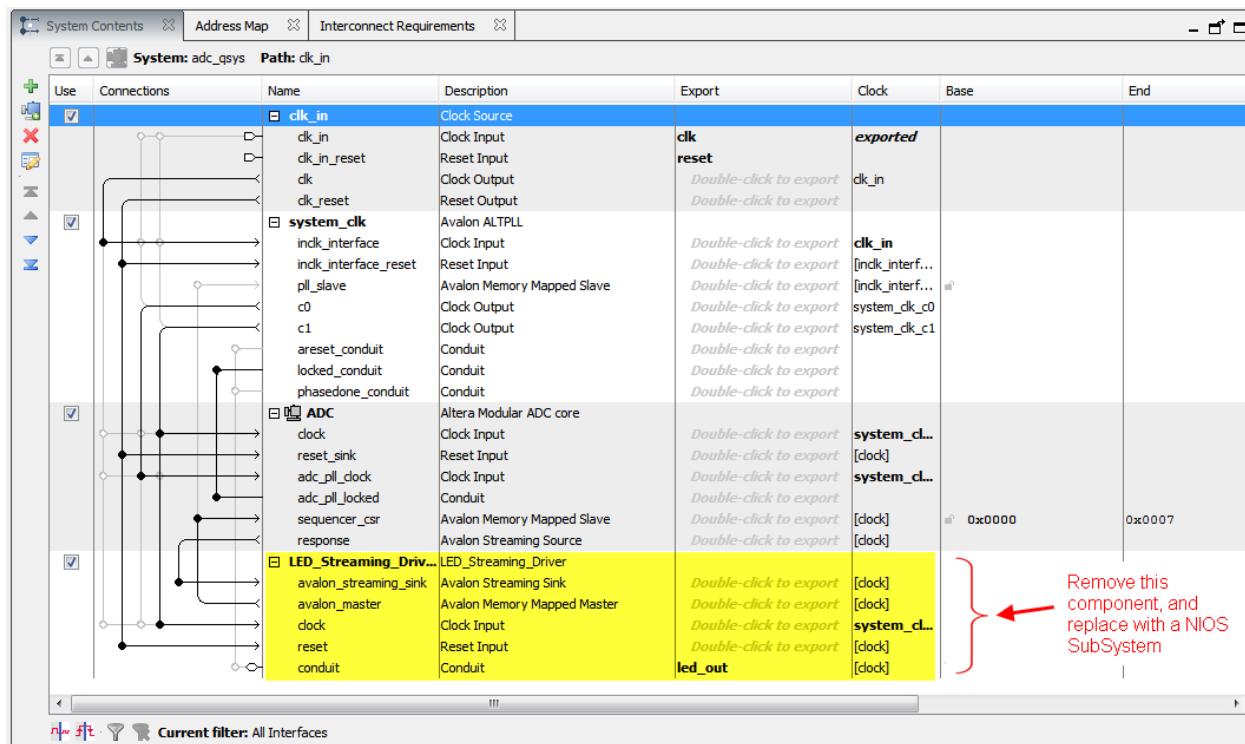
In this section of the lab, you will use a Nios processor to read data from the modular ADC, and send that data to a Parallel IO peripheral that drives the LEDs. The overall function of this system is similar to the streaming system, but uses a Nios instead of dedicated hardware.

To convert the system, you will remove the Streaming LED controller and replace it with a complete Nios processor Subsystem. Since Nios will require a memory mapped interface vs. a Streaming interface, you will also reconfigure the Modular ADC core to output the data in a memory mapped format.



A second set of files have been provided as a starting point for this section. You could start with the files created during the streaming lab, but the SignalTap signals are different, so it will be much easier to start with the provided lab files.

As a reminder, here is what the system looked like in the first section of this lab:



5.4.1 Open the ADC Nios Lab project

5.4.1.1 Close the Streaming Lab Project (if necessary), closing all open Quartus windows, including Qsys, SignalTap, programmer, etc.

5.4.1.2 Open the ADC Nios Lab Project: **File → Open Project** and browse to `<location of lab>\workshop_labs\5_ADC_Lab\ADC_Nios_Lab\ADC_Nios_Lab.qpf`. Click Open

5.4.2 Modify the Qsys system

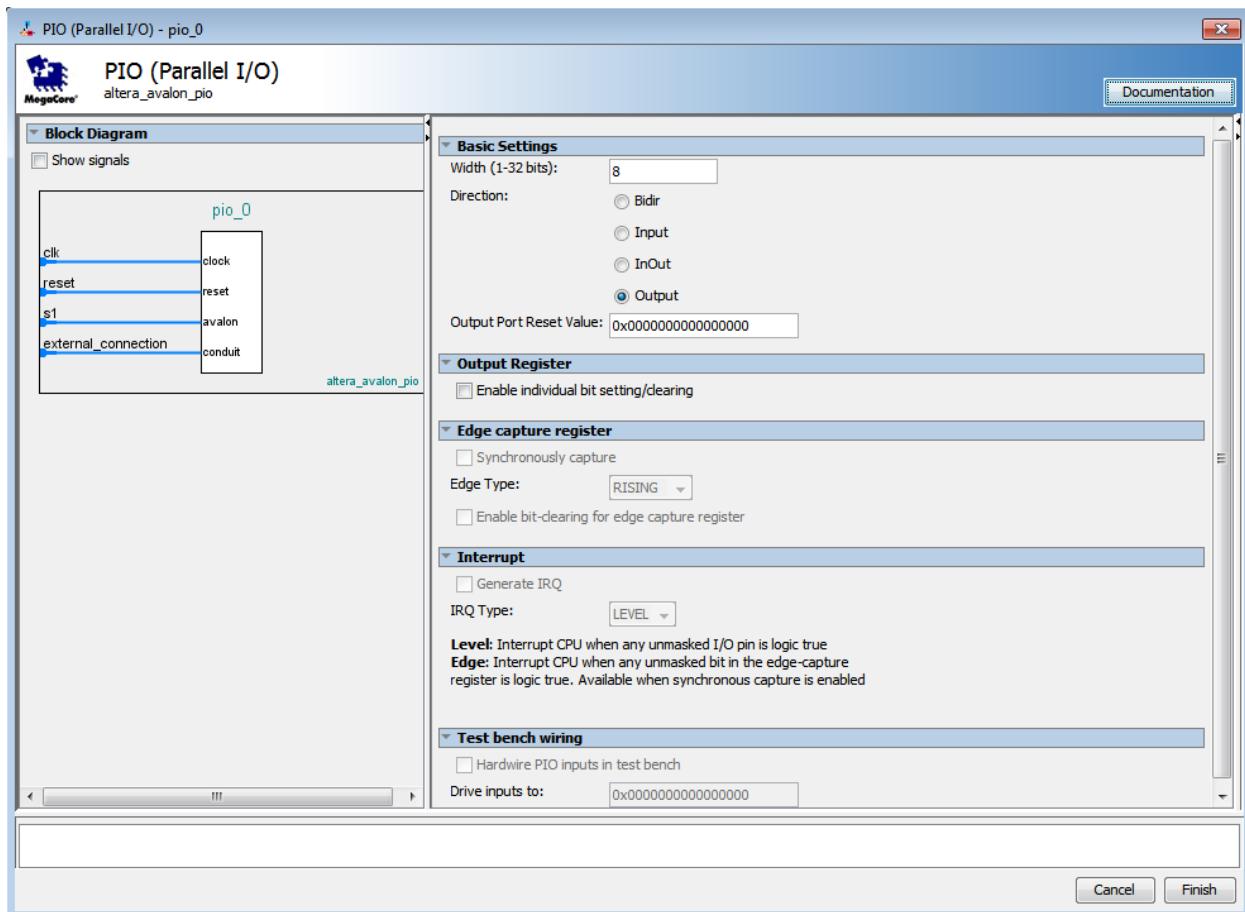
5.4.2.1 Launch Qsys: Click **Tools → Qsys** (or use the Qsys tool bar icon 

5.4.2.2 Open the Qsys system `<project directory>\adc_qsys.qsys`

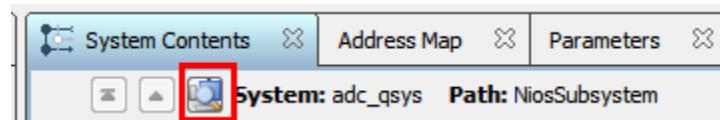
5.4.2.3 Delete the LED_Streaming_Driver. Click on the module and press Delete (Or alternatively, right click, and select Delete)

5.4.2.4 Add a Parallel Input/output component: **Processor and Peripherals → Peripherals** (or search for PIO in the IP catalog). This component will connect to the LEDs, so they can be written to from Nios

The default settings are correct. Click Finish



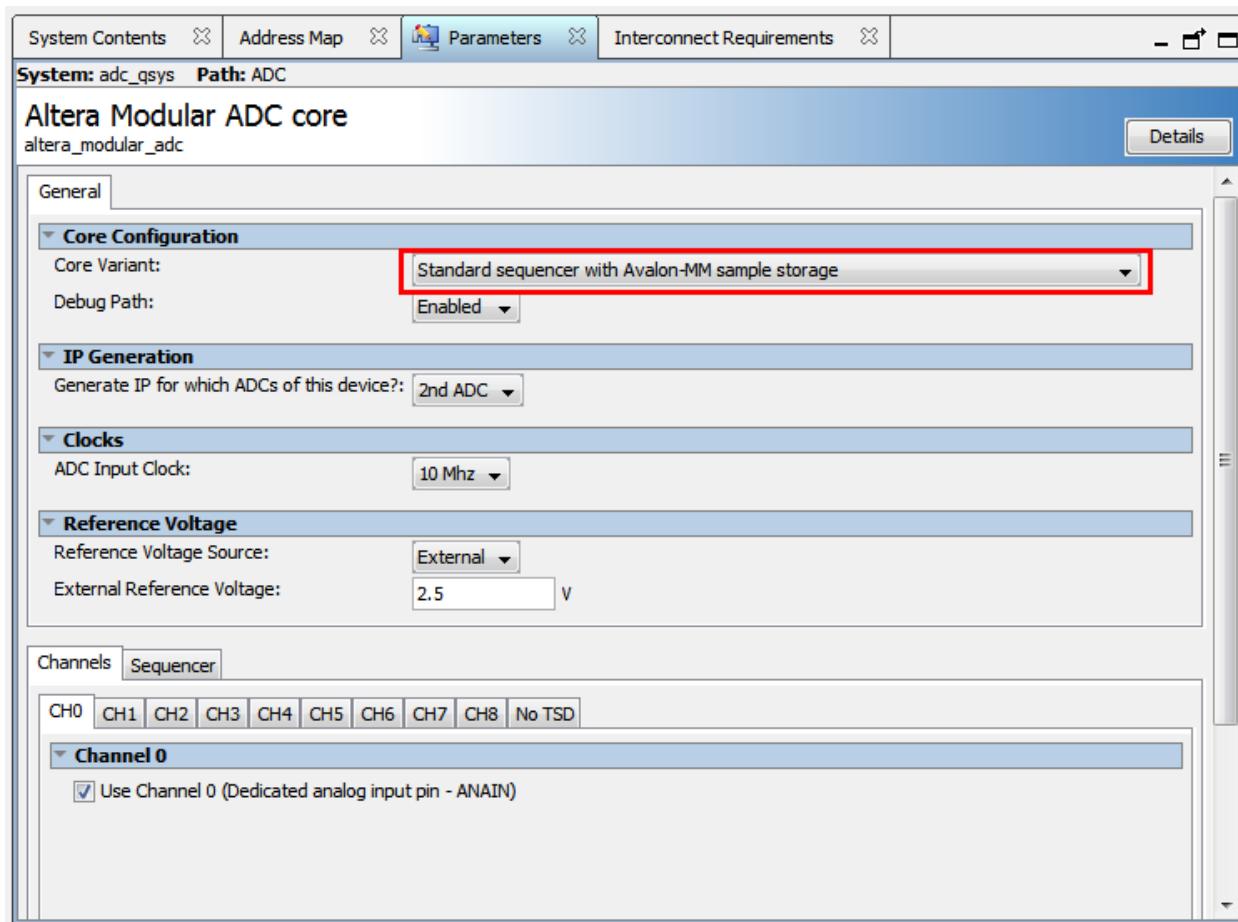
- 5.4.2.5 Add the Nios Subsystem: This is a custom system created for you, for this lab. It should be located in the DECA_Labs folder. There are no configuration options, so just click Finish to add it to the system. If you want to inspect this subsystem, right click on the module after it has been added, and select Drill into subsystem. You can also use the System navigation buttons to move up and down the hierarchy of a Qsys system. Return to the top level of the hierarchy when complete.



- 5.4.2.6 Rename the Nios subsystem to **NiosSubsystem**

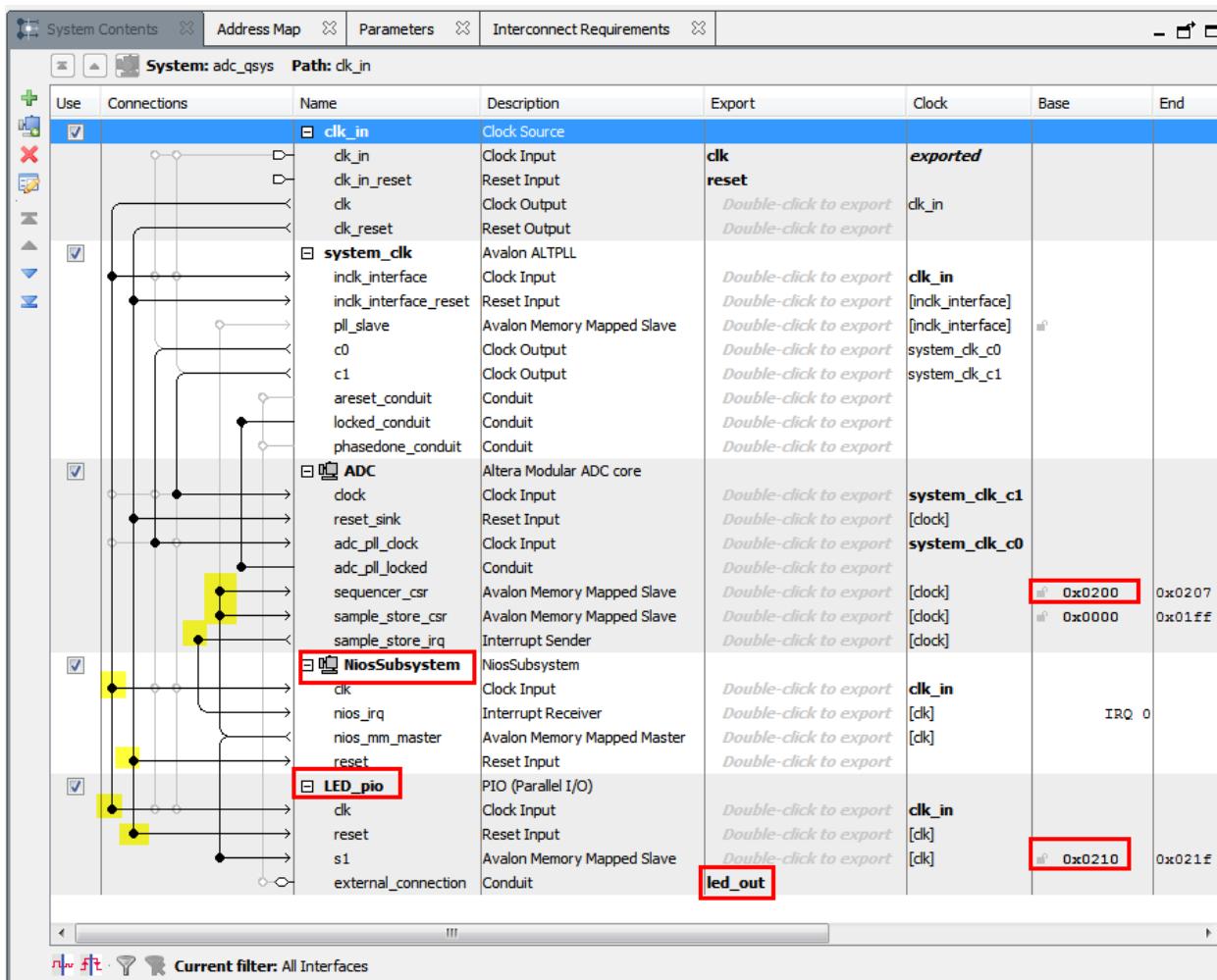
- 5.4.2.7 Reconfigure the ADC Core. Double click on the Altera ADC Modular Core to open the Parameters Tab. Change the Core Variant to **Standard sequencer with Avalon-MM sample storage**.

Changing to this variant has several effects. One of which is that the streaming interface is no longer available. Instead, the streaming interface is connected internally inside the ADC core to an on-chip RAM. This RAM is accessible through an Avalon memory mapped interface that now appears on the ADC core in Qsys.



5.4.2.8 Connect the Qsys components. Connect the clock, reset, and memory mapped interfaces as shown in the screenshot below:

5.4.2.9 Fix the Address conflict. There will be a memory conflict in your system (the sample_store_csr, the sequencer_csr, and the LED_pio are all at address 0x000). Fix this manually by setting the sequencer_csr Base Address to 0x0200, and the LED_pio to 0x0210. **Alternatively**, you can let Qsys fix it automatically: **System→Assign Base Addresses**.



5.4.2.10 Generate the system. You should have 3 warnings; these can be ignored. Click Generate HDL to open the Generate Dialog box, and click Generate to begin the generation process.

5.4.2.11 If there are no errors, click Close to close the Qsys Generation status dialog box.

5.4.2.12 Close Qsys and return to Quartus. If you get a message reminding you to add the qip, you can ignore this message, as the qip was added for you in this project.

5.4.3 Compile the project in Quartus

5.4.3.1 Start the Compile: **Processing → Start Compilation** (or click the compile icon  on the toolbar)

Confirm the compile completed successfully.

5.4.4 Configure the FPGA

5.4.4.1 Open the Quartus II Programmer via **Tools → Programmer**

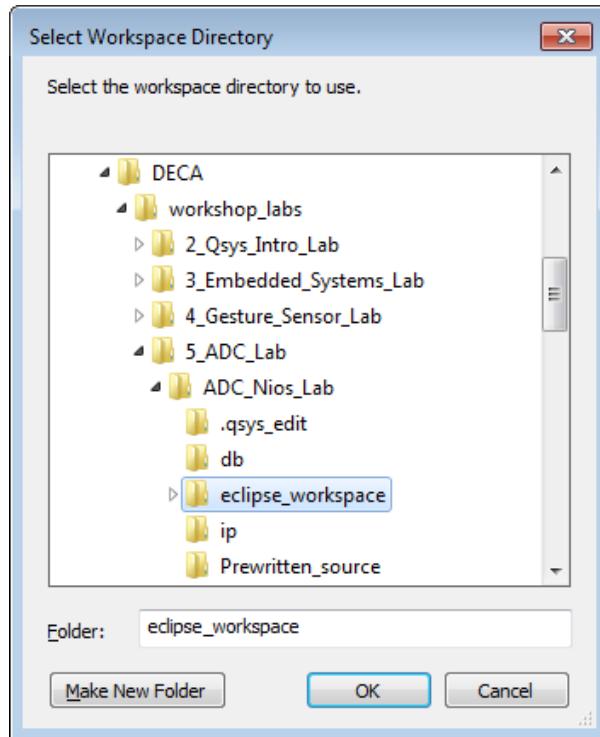
5.4.4.2 The Programmer window will open with a predefined Chain-Description File (**ADC_Nios_Lab.cdf**) that specifies the programming file **output_files/ADC_Nios_Lab.sof**

5.4.4.3 Click Start to begin configure the FPGA.

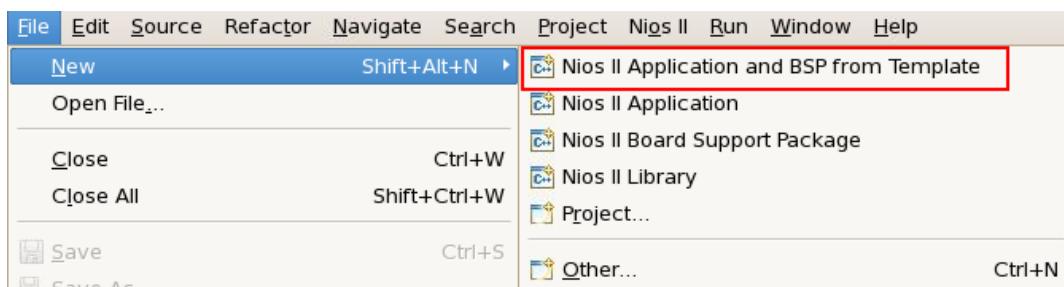
5.4.5 Create the Nios software application

5.4.5.1 Open Eclipse: From Quartus, **Tools → Nios II Software Build Tools for Eclipse**

5.4.5.2 Select a workspace: It is recommended that you create a workspace named **eclipse_workspace** in the same directory as the Quartus project files. Browse to the project folder and click Make New Folder. Rename the folder to **eclipse_workspace**. Click OK to continue

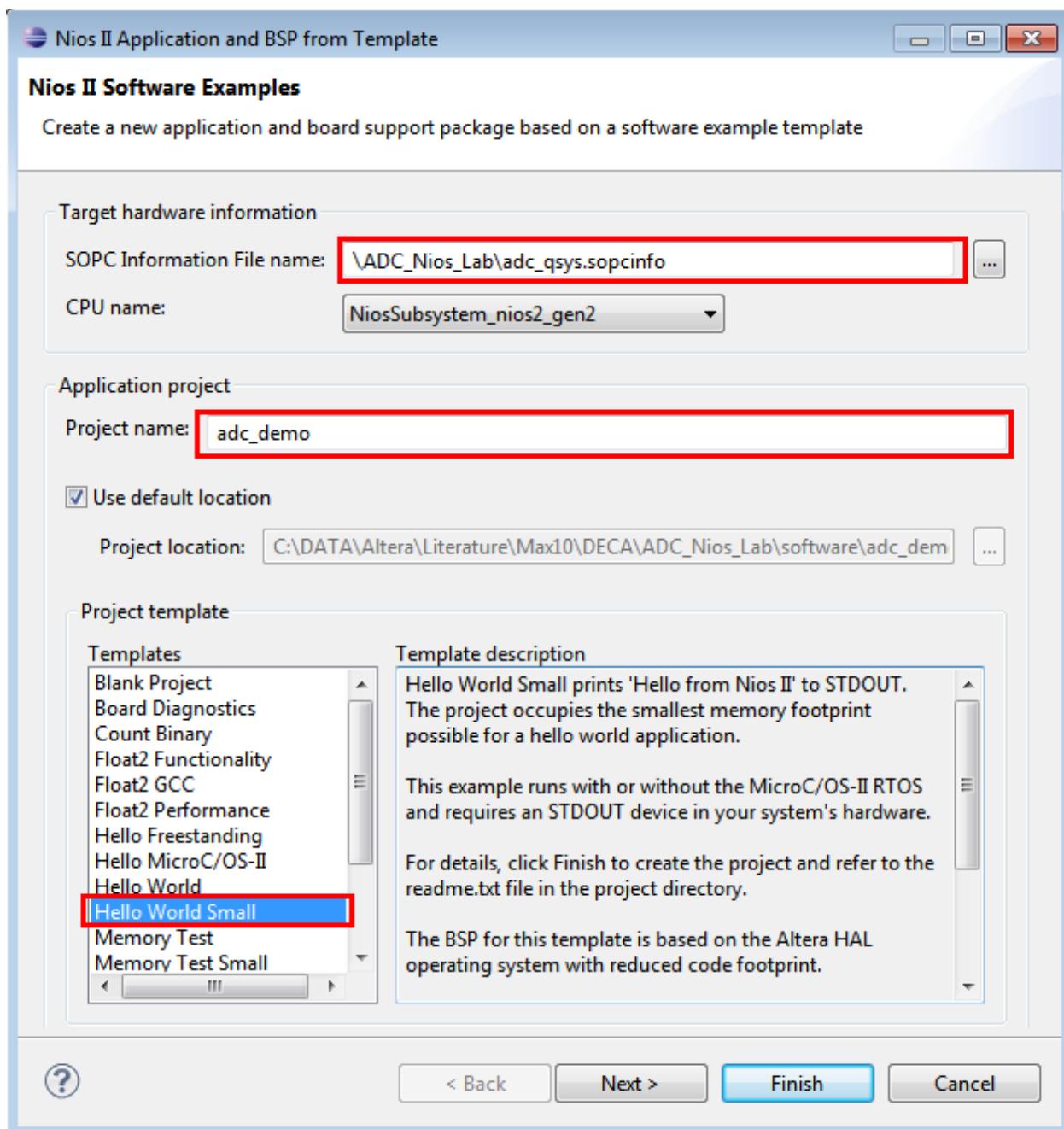


5.4.5.3 Create the project: **File → New → Nios II Application and BSP from Template**



5.4.5.4 Configure the new project. When the Template dialog appears, enter these values:

- Browse to <project directory>\ADC_Nios_Lab\adc_qsys.sopcinfo
- Name the project **adc_demo**
- Select Hello World Small as the Project Template
- Click Finish

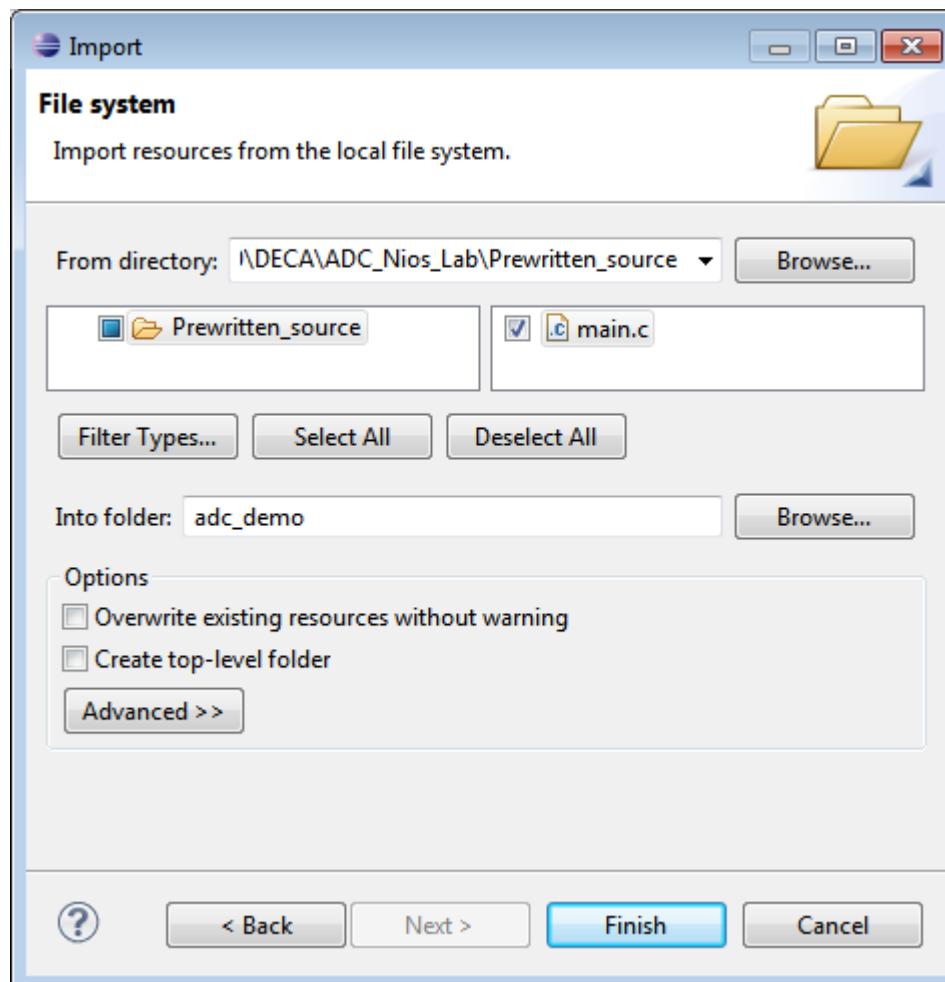


5.4.5.5 Import the pre-written C-Code for this lab: **File → Import → General → File System**. Click Next

For the "From directory", Browse to <project directory>ADC_Nios_Lab\Prewritten_source and check on **main.c** file

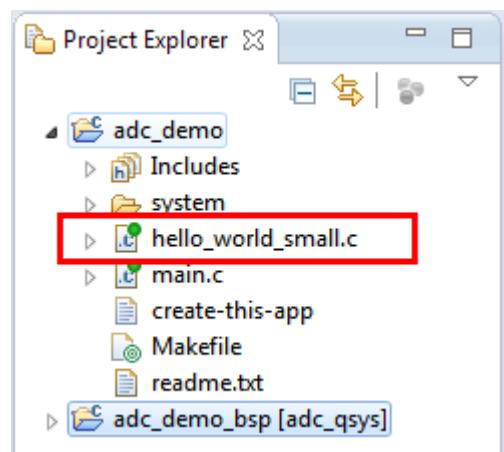
Browse to **adc_demo** for the "Into Folder".

Click Finish.



Alternatively, to import a file into the workspace, you can simply drag it in from Windows Explorer.

5.4.5.6 Delete the hello_world_small.c from Project



5.4.5.7 Build the project : **Project → Build All** (or click the Build All icon  on the toolbar)

5.4.6 Configure the target hardware

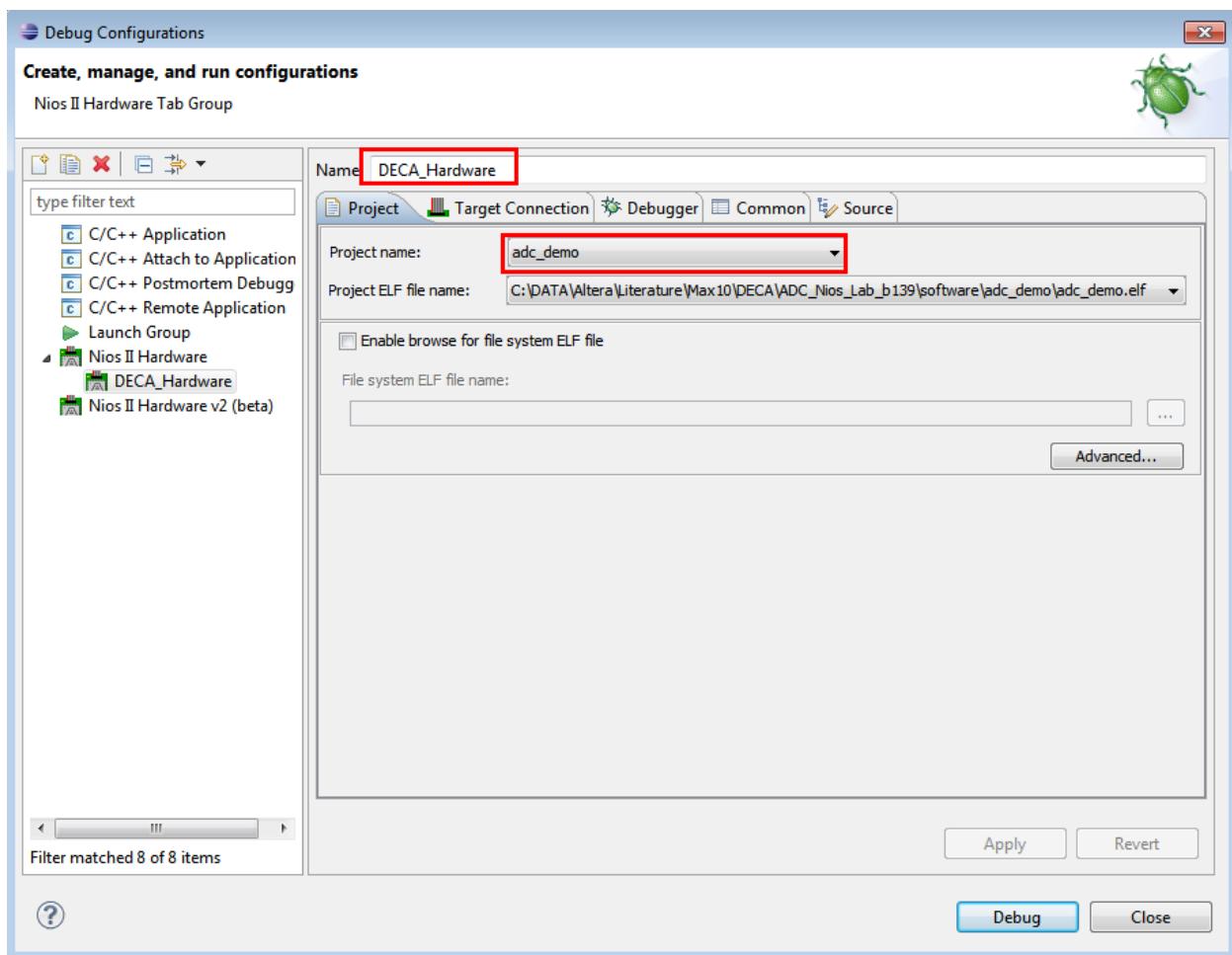
5.4.6.1 Open the debug configuration dialog: **Run → Debug Configurations**

5.4.6.2 Double Click on Nios II Hardware to create a new Nios II launch configuration

5.4.6.3 Name this configuration: **DECA_hardware**

5.4.6.4 Select **adc_demo** as the Project name. The ELF should automatically populate if the compile was successful.

5.4.6.5 The result should look like this:



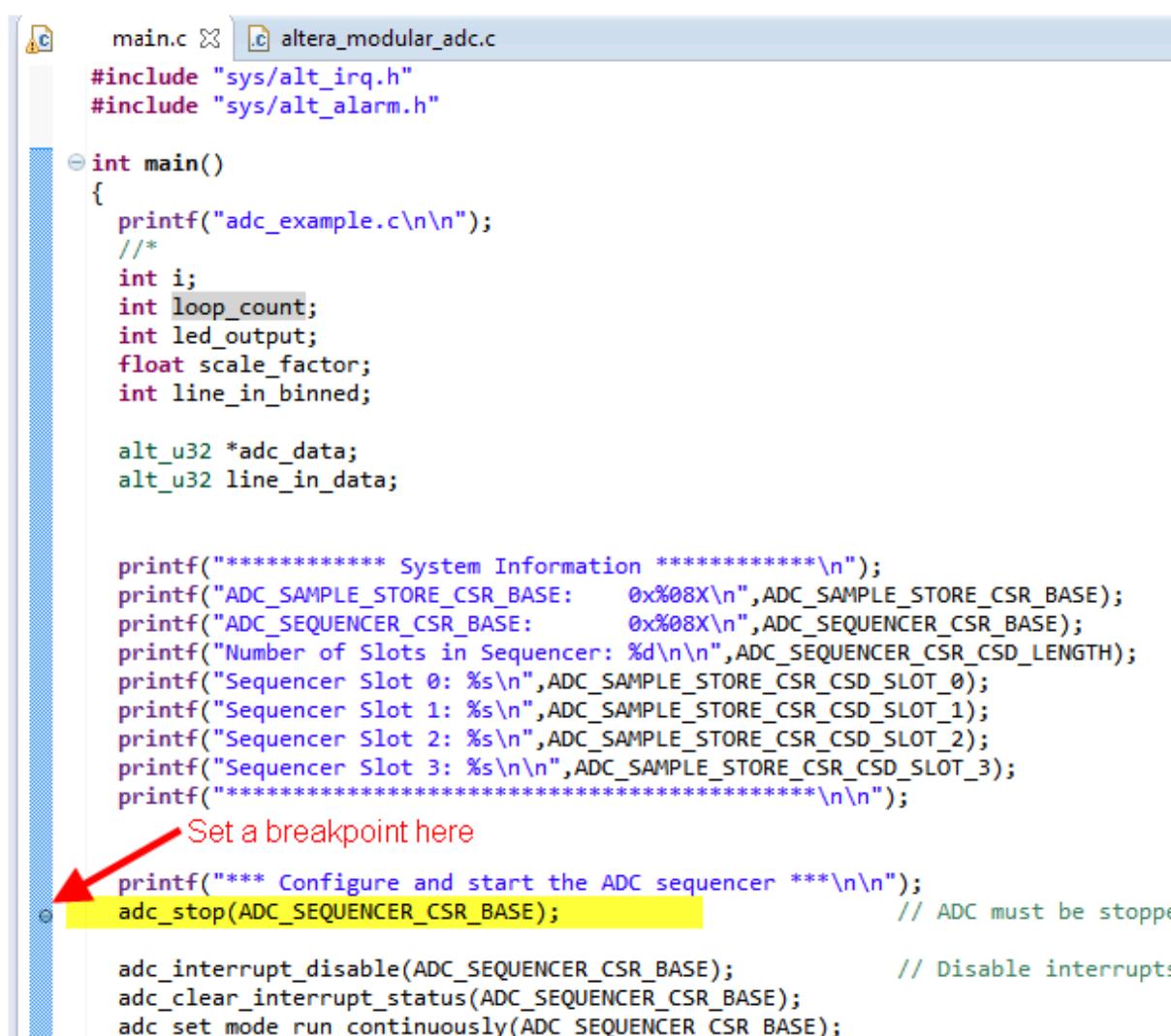
5.4.6.6 Click Debug to begin debugging the application.

5.4.7 Debug the application

When you begin the debug session, the debugger downloads the code to the on-chip RAM, and begins executing the code. A breakpoint is automatically set at the entry point to main(), so the processor should be paused, waiting for you to resume.

5.4.7.1 Set a breakpoint at line 45 by double clicking in the blue margin to the left of the code.

This will stop the processor just before configuring the ADC core.



```

main.c  altera_modular_adc.c

#include "sys/alt_irq.h"
#include "sys/alt_alarm.h"

int main()
{
    printf("adc_example.c\n\n");
    /*
    int i;
    int loop_count;
    int led_output;
    float scale_factor;
    int line_in_binned;

    alt_u32 *adc_data;
    alt_u32 line_in_data;

    **** System Information *****
    printf("ADC_SAMPLE_STORE_CSR_BASE: 0x%08X\n", ADC_SAMPLE_STORE_CSR_BASE);
    printf("ADC_SEQUENCER_CSR_BASE: 0x%08X\n", ADC_SEQUENCER_CSR_BASE);
    printf("Number of Slots in Sequencer: %d\n\n", ADC_SEQUENCER_CSR_CSD_LENGTH);
    printf("Sequencer Slot 0: %s\n", ADC_SAMPLE_STORE_CSR_CSD_SLOT_0);
    printf("Sequencer Slot 1: %s\n", ADC_SAMPLE_STORE_CSR_CSD_SLOT_1);
    printf("Sequencer Slot 2: %s\n", ADC_SAMPLE_STORE_CSR_CSD_SLOT_2);
    printf("Sequencer Slot 3: %s\n", ADC_SAMPLE_STORE_CSR_CSD_SLOT_3);
    printf("*****\n\n");

    Set a breakpoint here

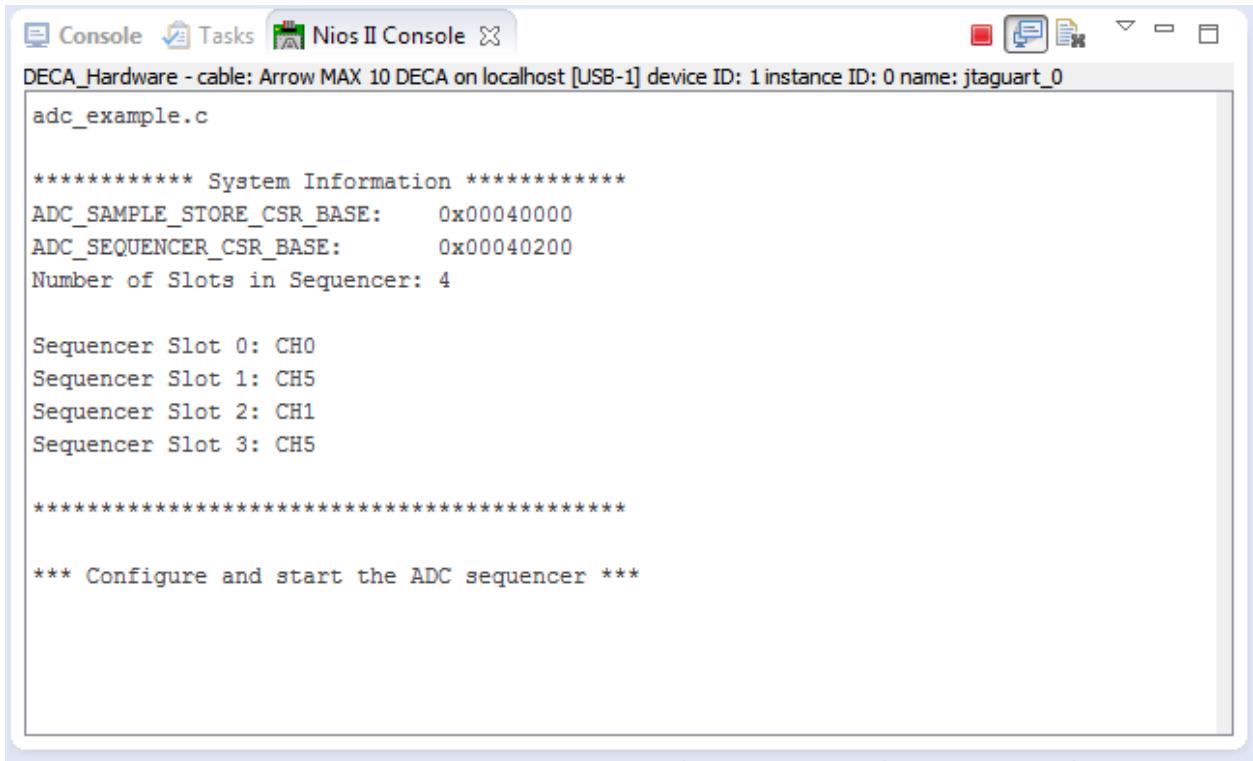
    printf("**** Configure and start the ADC sequencer ****\n\n");
    adc_stop(ADC_SEQUENCER_CSR_BASE); // ADC must be stopped

    adc_interrupt_disable(ADC_SEQUENCER_CSR_BASE); // Disable interrupts
    adc_clear_interrupt_status(ADC_SEQUENCER_CSR_BASE);
    adc_set_mode_run_continuously(ADC_SEQUENCER_CSR_BASE);
}

```

5.4.7.2 Resume the processor: **Run → Resume**, or press the Resume icon 

5.4.7.3 Observe the console output. Notice the slot information matches the configuration we entered in the modular ADC core. This information was automatically passed from the hardware configuration in Qsys to the software via the .sopcinfo file that was used to create the application.



The screenshot shows a computer window titled "Nios II Console". The title bar also includes "Console", "Tasks", and "DECA_Hardware - cable: Arrow MAX 10 DECA on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0". The main window displays the following text:

```
DECA_Hardware - cable: Arrow MAX 10 DECA on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0
adc_example.c

***** System Information *****
ADC_SAMPLE_STORE_CSR_BASE:      0x00040000
ADC_SEQUENCER_CSR_BASE:         0x00040200
Number of Slots in Sequencer: 4

Sequencer Slot 0: CH0
Sequencer Slot 1: CH5
Sequencer Slot 2: CH1
Sequencer Slot 3: CH5

*****
*** Configure and start the ADC sequencer ***
```

5.4.7.4 Open SignalTap Analyzer: From Quartus, **Tools → SignalTap II Logic Analyzer**.

The Deca_tap_nios.stp system should open.

5.4.7.5 Start the Analyzer in repetitive trigger mode: **Processing → Autorun Analysis**, or press the autorun icon .

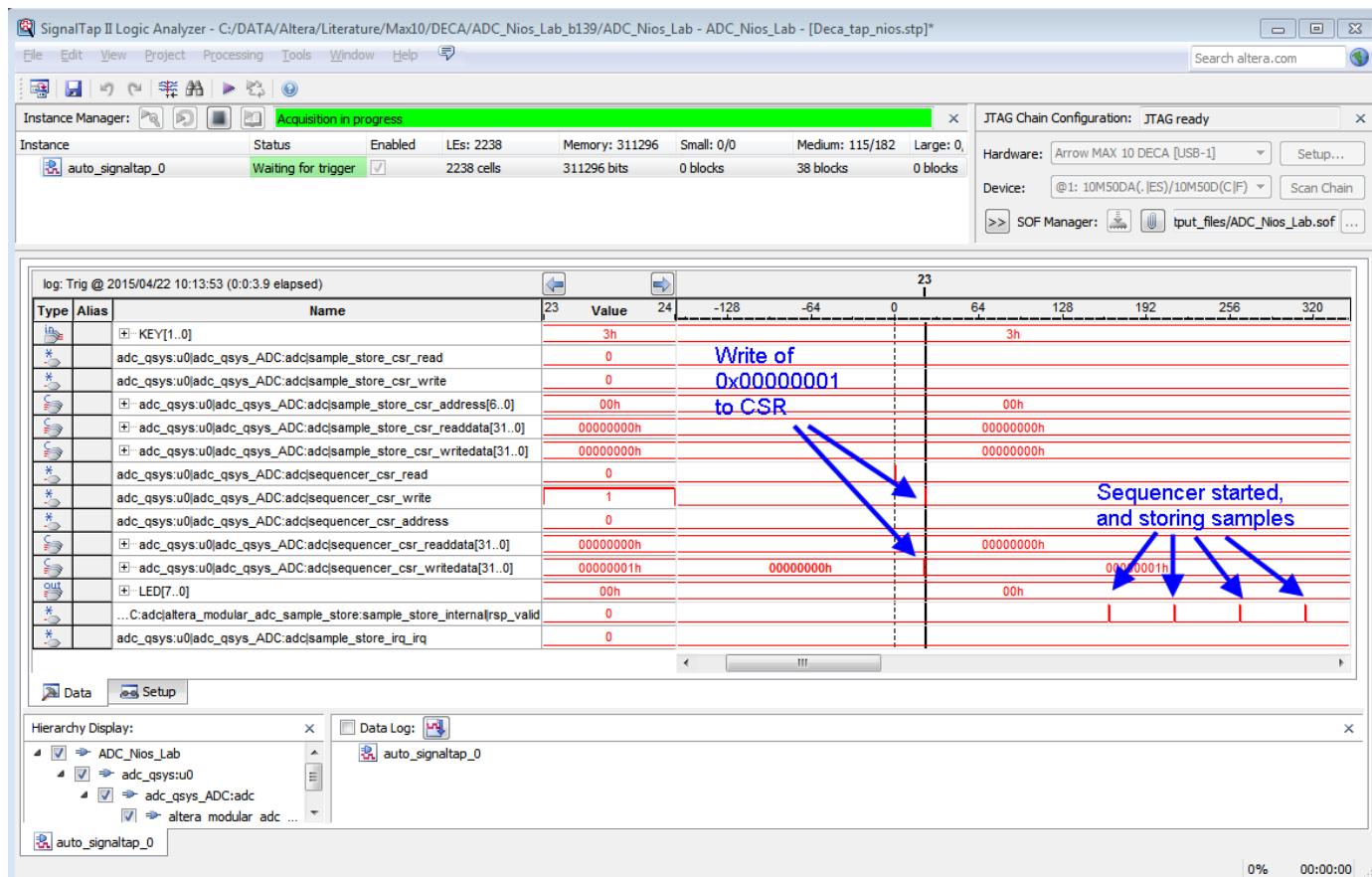
The trigger condition has been configured to trigger on reads OR writes to the ADC (see this on the Setup tab), so every time Nios accesses the ADC, the Logic Analyzer should trigger, making it easy to understand how the system functions.

5.4.7.6 In the debugger, step through the code: **Run → Step Over**, or by pressing the Step Over icon .

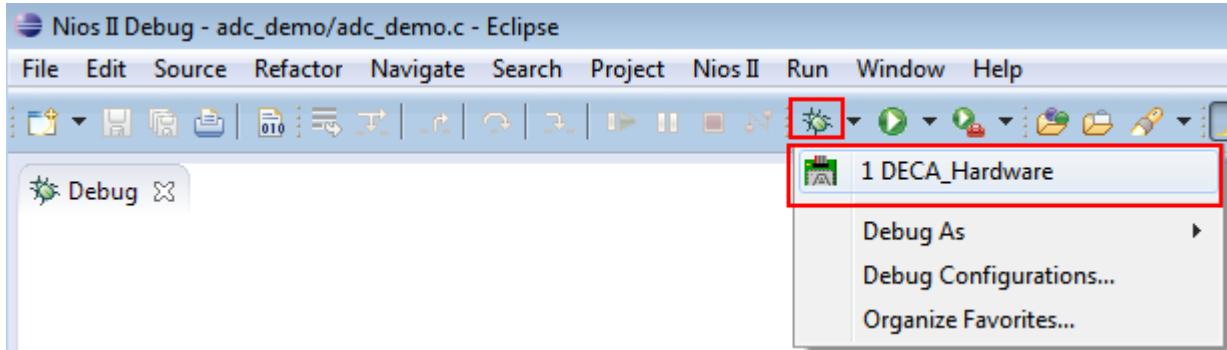
After each step, the SignalTap system should automatically trigger, and display the waveforms. See the screenshot below for the result of stepping over the **adc_start(ADC_SEQUENCER_CSR_BASE)** instruction.

5.4.7.7 Resume the code by pressing the resume icon .

This will cause the program to run freely until the next breakpoint. Since there are no more breakpoints set, the system will perform 100,000 reads, and then complete.



Rerun the application, if desired. To restart the code, click on the DECA_Hardware option under the Debug Icon:



5.4.7.9 Congratulations. You have completed the ADC capture lab! Feel free to experiment with changes to the c-code and the SignalTap analysis system as time permits. See the following section for additional reference information.

5.4.8 Other [possibly] helpful information

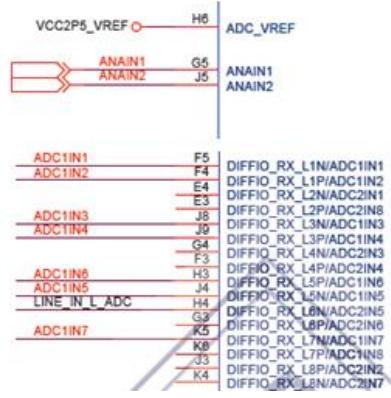
5.4.8.1 How big is my code?

If you plan to run Nios from a Max10 device without external RAM, on-chip RAM will often be the limiting factor in your design. So the question arises, how can I easily find out the size of my code? Use the nios2-stackreport command, from a Nios command shell.

```
a30815@LE9794 /cygdrive/c/altera/15.0.0.139
$ nios2-stackreport C:/DATA/Altera/Literature/Max10/DECA/ADC_Nios_Lab_b139/software/adc_demo/adc_demo.elf
Info: <C:/DATA/Altera/Literature/Max10/DECA/ADC_Nios_Lab_b139/software/adc_demo/adc_demo.elf> 10228 Bytes program size (code + initialized data).
Info:
        21 KBytes free for stack + heap.
a30815@LE9794 /cygdrive/c/altera/15.0.0.139
$
```

5.4.8.2 What other signals are connected to the ADC on the DECA board?

The DECA schematics are included in the lab files. For convenience, here is a screenshot of the relevant section. ANAIN1 and ANAIN2 come from the SMA connectors, Line_IN_L is connected to the left channel of the Line In jack, and the rest are connected to the 46 pin headers.



That completes this lab section.

USB 2.0 to SDHC Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 6. USB 2.0 TO SDHC LAB	242
6.1 Getting Started	242
6.1.1 Getting your DECA Kit	243
6.1.2 Installing the Altera Complete Design Software v15.0.....	243
6.1.3 Licensing the USB and SDHC IP cores	243
6.2 Review the System Design Flow	244
6.2.1 Examine the System Tool Flow	244
6.3 Review and prepare the DECA Development Platform	245
6.3.1 Insert the SD Card	246
6.4 Compiling the Quartus II Project	246
6.4.1 Open the Project in Quartus II	246
6.4.2 Review the Qsys design	249
6.4.3 Generate the Qsys HDL Files for the project.....	250
6.5 Download the Configuration File to DECA Board.	256
6.5.1 Open the Programmer Tool	256
6.5.2 Create the Software Design.....	259
6.5.3 Start Nios II Software Build Tools for Eclipse	259
6.5.4 Eclipse Opens with a Blank Project	261
6.5.5 Open the USB2.0 Project Software Project ...	261

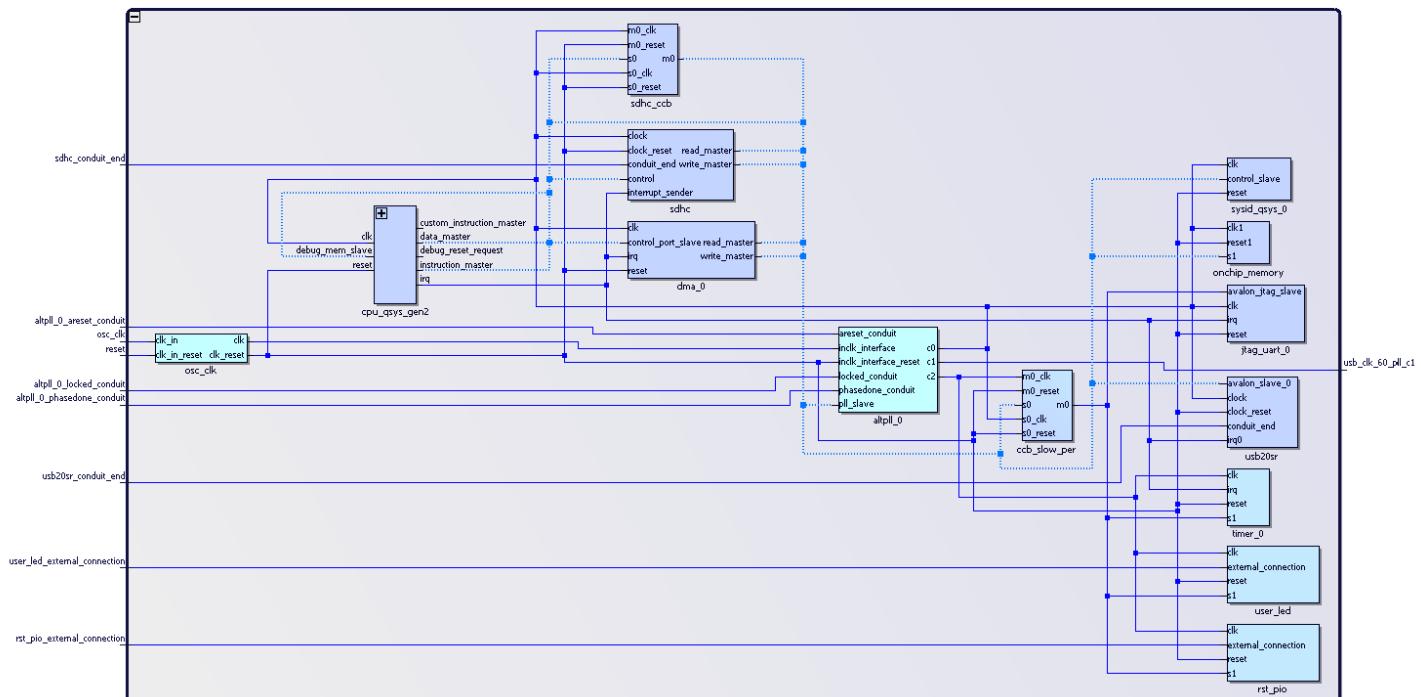
LAB 6. USB 2.0 TO SDHC LAB

Overview: In this lab, you will compile a Qsys based design that allows your laptop to browse the SD card on your DECA board, essentially turning the DECA board into a mass storage device. After configuring your DECA board with this design, you will run a C code executable on the Nios II processor to control the data flow between the SD Card and the USB 2.0 port on your laptop.

Since this lab is complete, there will be no editing. This exercise will provide a working knowledge of an FPGA design flow which also includes an embedded processing component, the Nios II soft core processor. At the end of this lab you should be able to:

- Understand the complete Embedded FPGA design flow
- Configure the target FPGA on the DECA Board using the Quartus II Programmer
- Create and run a software application on the Nios II processor

Below is a schematic view of the embedded system you will create in this lab.



6.1 Getting Started

Overview: Please ensure that you have the necessary hardware and the software installed so that this lab can be completed successfully.

Below is a list of items required to complete this lab:

- Arrow DECA Evaluation Kit
- Two USB cables
- Lab Design files
- Quartus II v15.0 Design Software
- Personal computer or laptop running Windows 7 with at least an Intel i3 core (or equivalent), 4 GB of RAM, and 12 GB of free hard disk space

6.1.1 Getting your DECA Kit

If you are attending a DECA Workshop, you will receive your DECA kit in the 3-in-1 evaluation kit bundle when you arrive at the workshop location. If you are working on this lab independently, a DECA kit can be purchased through your Arrow sales representative or at parts.arrow.com.



Make sure you have a USB cable to connect the on-board USB Blaster to your laptop. If you are attending the workshop, USB cables are included in your evaluation kit bundle.

6.1.2 Installing the Altera Complete Design Software v15.0

This Lab assumes you have Quartus II v15.0 installed on your computer prior to the workshop date as there is not enough time to download and install the software during the workshop. It is recommended that you complete the Introductory Labs before you do the USB2.0 to SDHC Lab.

6.1.3 Licensing the USB and SDHC IP cores

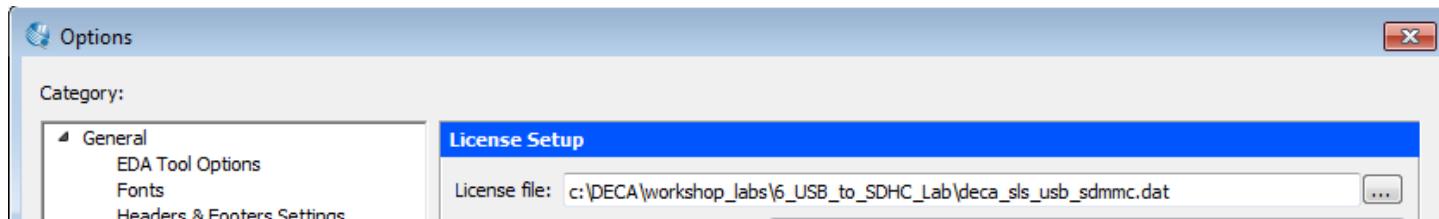
You will require a license in order to compile the FPGA design for this lab. The license is provided with the lab files. Here are the steps required to configure the license in Quartus

6.1.3.1 In the Quartus II menu, select **Tools → License Setup**

6.1.3.2 In the License Setup section browse to your license located in:

`c:\DECA\workshop_labs\6_USB_to_SDHC_Lab\deca_sls_usb_sdmmc.dat`

6.1.3.3 Your screen should now look like this:





Note that if you already have a license file in this field, you can append the new license file using a semi-colon (;), such as:

```
c:\existing_license.dat;c:\DECA\workshop_labs\...\deca_sls_usb_sdmmc.dat
```

- 6.1.3.4 There are several different ways to 'refresh' the License Setup dialog box. Generally, select OK, to close the dialog box, then re-open it via: **Tools → License Setup**
- 6.1.3.5 Review the Licensed AMPP/MegaCore functions section and scroll to the bottom of the list to see your SLS licenses:

Licensed AMPP/MegaCore functions:	
Vendor	Product
System Level Solutions, Inc (5750)	0C00
System Level Solutions, Inc (5750)	0A04

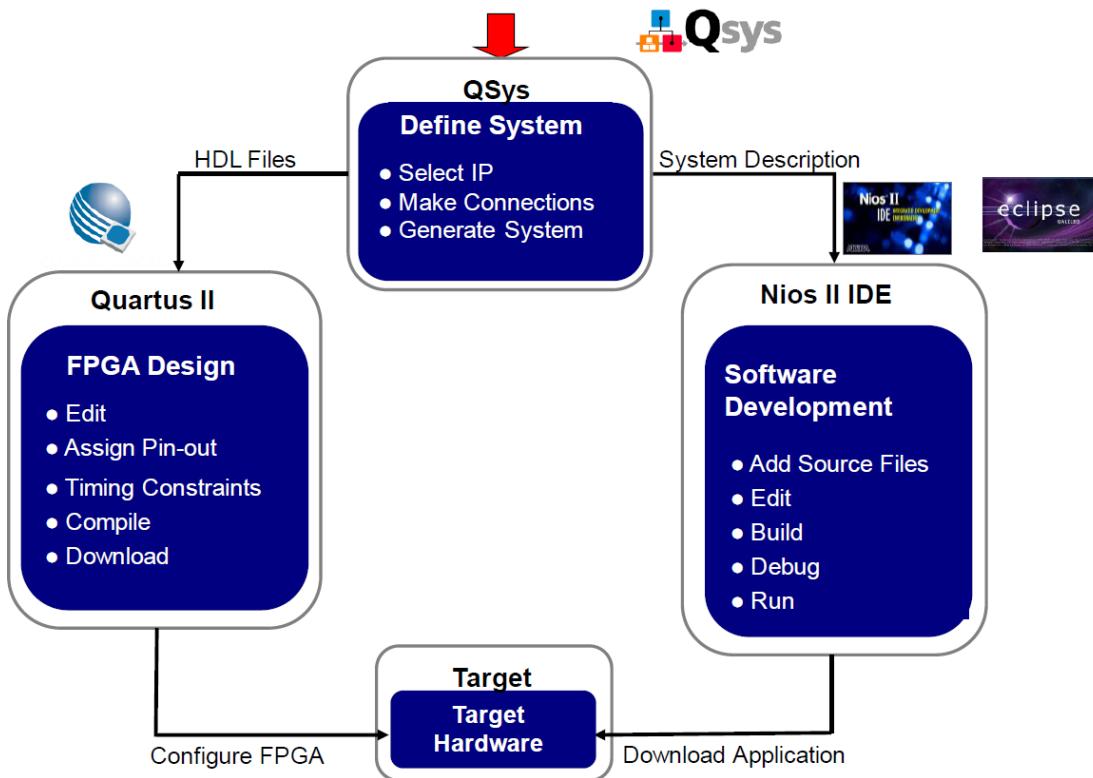
6.2 Review the System Design Flow

Overview: In this section, you will review the design flow needed to do an Altera FPGA Embedded design.

6.2.1 Examine the System Tool Flow

Developing software for an Altera FPGA requires an understanding of the design flow between the Qsys System Integration tool and the Nios II Embedded Design Suite (EDS). For this workshop, the design will be created within Qsys, RTL will be exported from Qsys, and Quartus II will compile a program that can be downloaded to the MAX 10 FPGA on the DECA Board. The software for the Nios II will be developed using the Nios II 15.0 Software Build Tools for Eclipse.

The objective of this lab is to experience the complete development tool flow required to design a complete embedded system.



The above diagram shows the typical design flow for an embedded system-level design. There are two separate flows: hardware and software.

Within Quartus II, the hardware flow takes the Qsys generated RTL and places and routes the logic in the FPGA fabric. The output of Quartus is a hardware image that can be downloaded to the FPGA.

For the software flow, the Qsys system description is imported into the Nios II EDS so that a software application can be built to match the hardware that was just created. The output of the Nios II EDS is a software executable that can be run on the Nios II processor.

6.3 Review and prepare the DECA Development Platform

Review the components on the DECA board. This development board provides a full system built around the MAX10, including external memory, LEDs, sensors, buttons, and power supplies.



There are many components on the DECA board including the LEDs, capacitive push buttons, USB, Ethernet, HDMI and MIPI Interfaces.

This design will use on chip memory, PLLs, Nios II soft processor, and third party USB and SDHC IP provided by SLS.

6.3.1 Insert the SD Card

Plug the Micro SD card into the SD card slot on the back of the board



Note: This step must be completed before plugging in the USB cable, or Nios will not properly initialize the SD card during the boot process.

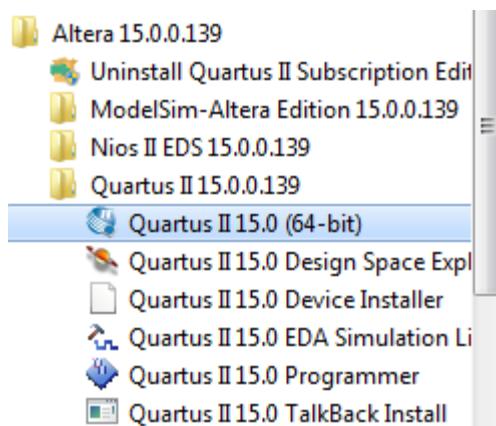
6.4 Compiling the Quartus II Project

In this module, you will use a project that has already been created. You will run this project through the design tools including Quartus II, Qsys and Nios II EDS in order to download it to the DECA board. When you run the Nios II executable you will be able to browse to the DECA SD card from your laptop, mimicking a USB drive.

6.4.1 Open the Project in Quartus II

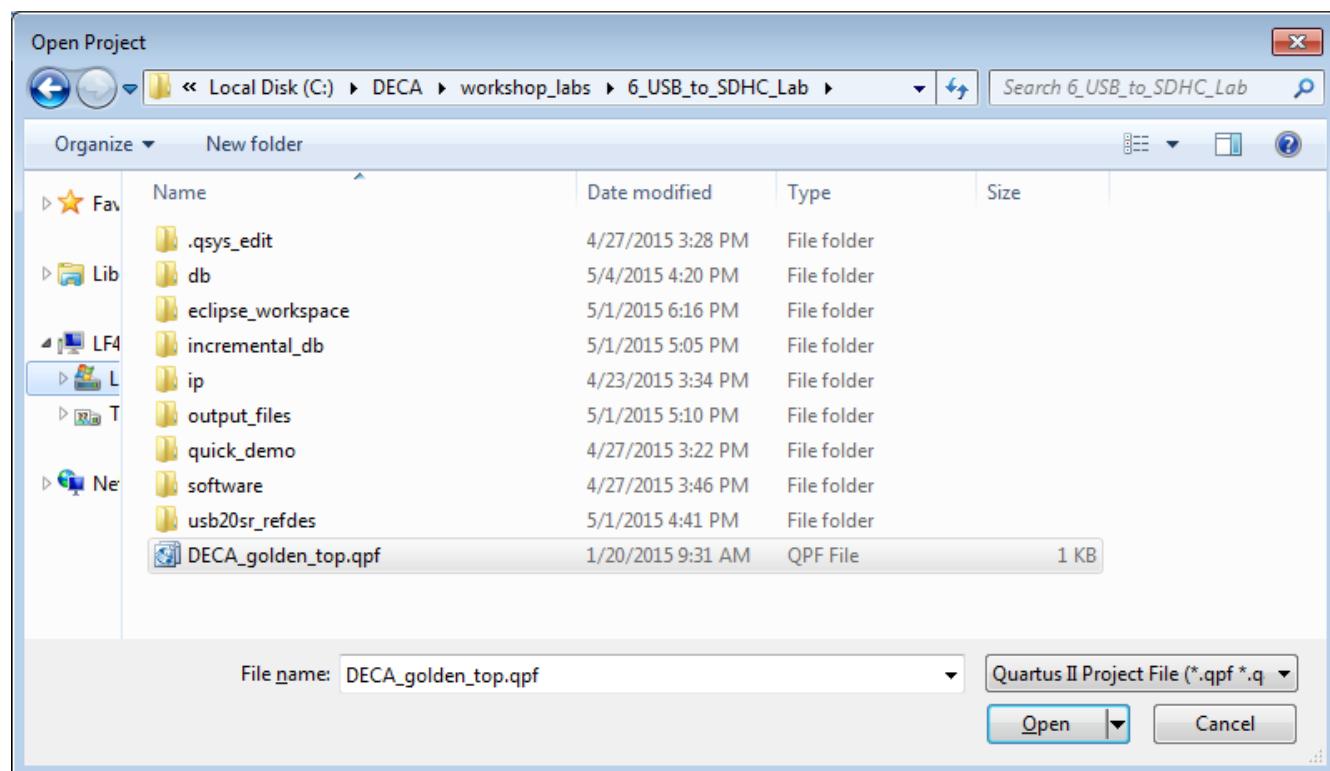
- 6.4.1.1 The Quartus II project resides in the default location of:
c:\DECA\workshop_labs\6_USB_to_SDHC_Lab

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

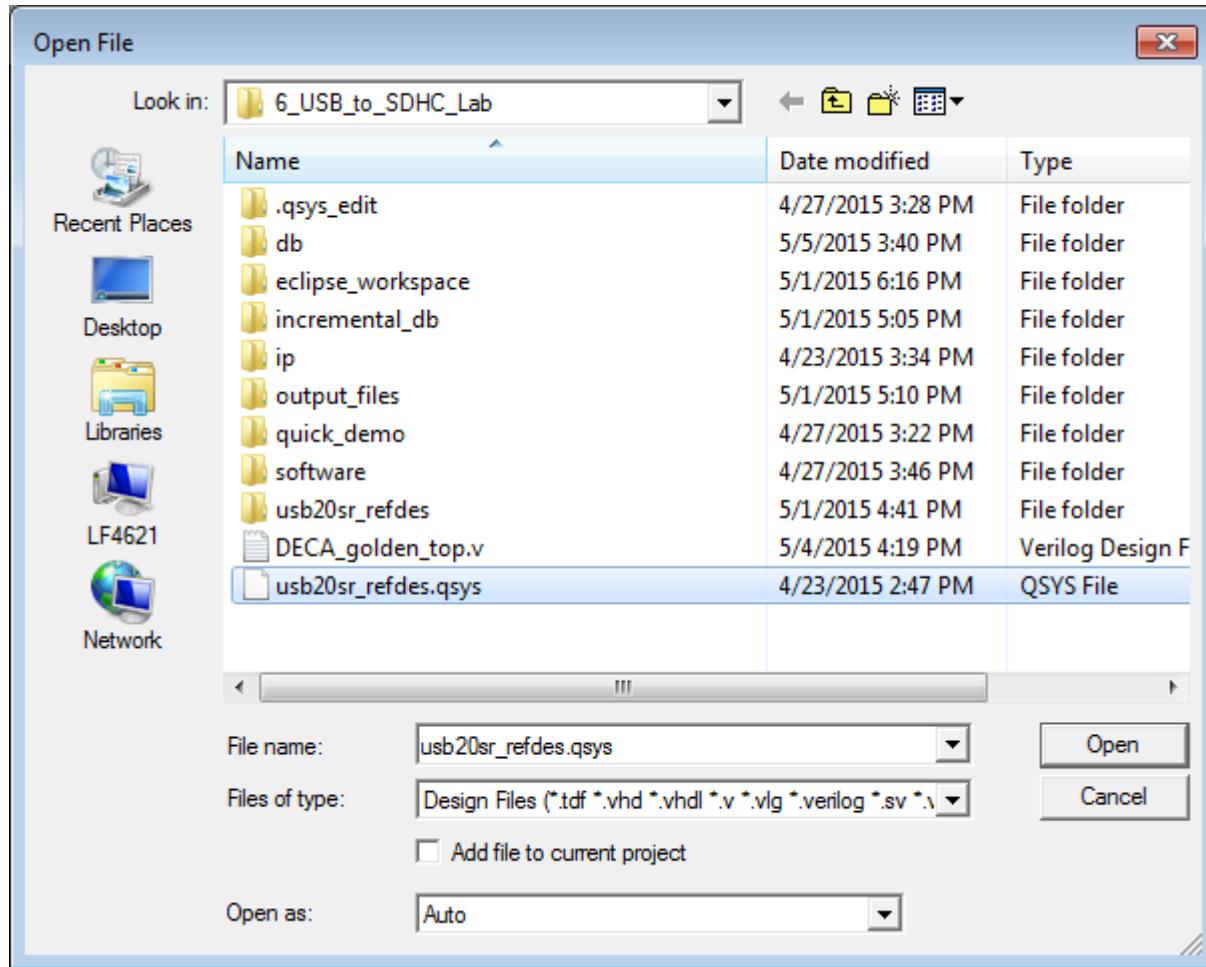


- 6.4.1.3 Open the project via the Quartus II menu: Click **File → Open Project**

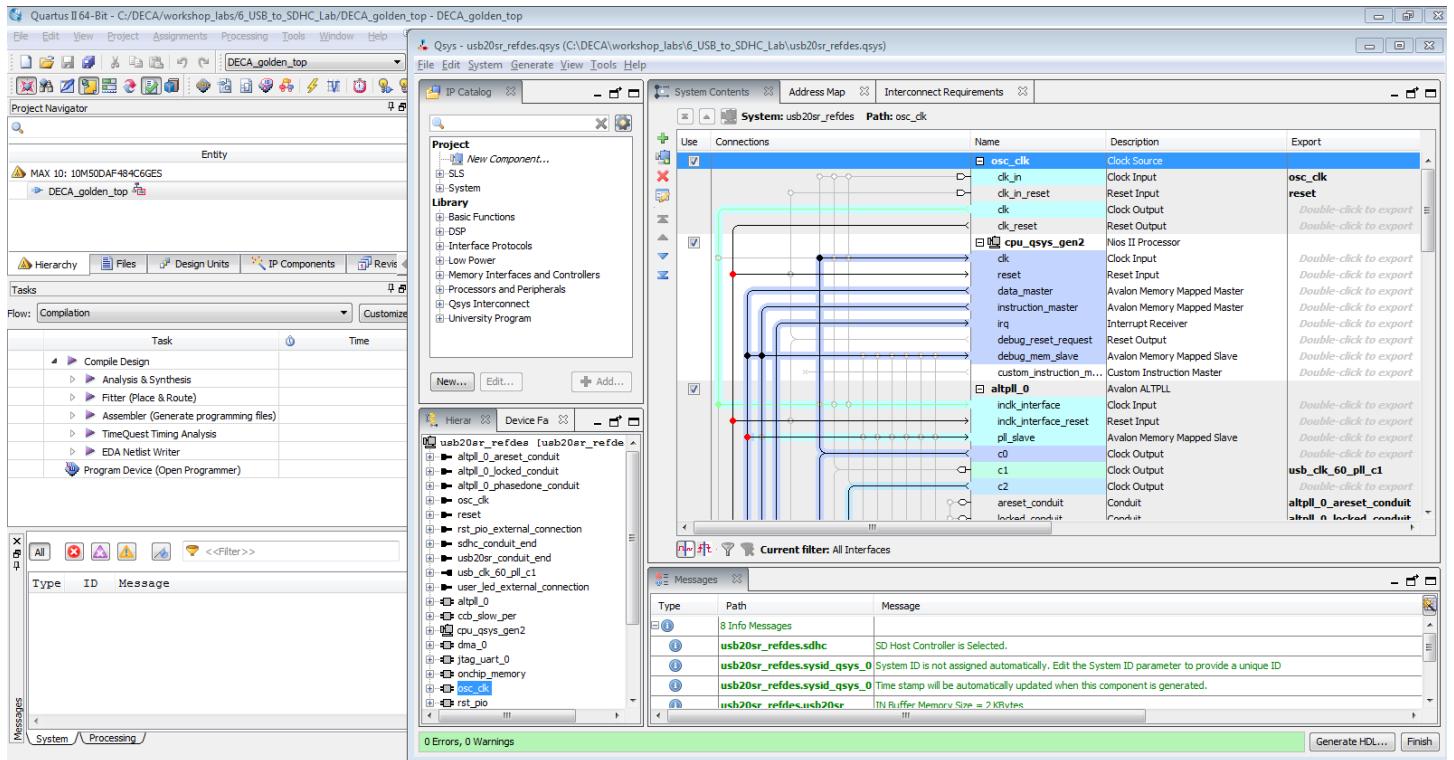
- 6.4.1.4 Browse to **c:\DECA\workshop_labs\6_USB_to_SDHC_Lab** then open **DECA_golden_top.qpf**



6.4.1.5 After the Project opens open the Qsys file via: **File → Open** and open the Qsys file **usb20sr_refdes.qsys** as shown:



6.4.1.6 You should now have both Quartus II Project and the Qsys Project open as shown



6.4.2 Review the Qsys design

6.4.2.1 Nios II Processor Data Master Connection

The Nios II processor has two masters as part of its Harvard architecture. The instruction master connects to the on-chip ram where it executes the firmware. The data master connects to the Avalon MM slaves. The Nios II processor uses this master to talk to all of the peripherals. Note that there are several other masters in this design

6.4.2.2 DMA Avalon Master for USB

The connections of the USB IP core (usb20sr) are connected to the dma_0 peripheral. This DMA engine is configured by the Nios II processor to move data in and out of the USB IP to and from the on-chip memory.

6.4.2.3 SDHC Avalon MM Interface

The SDHC IP has an Avalon MM Slave that the Nios II processor sets up the parameters required to configure the SDHC peripheral. The SDHC IP also has two Avalon MM masters that is essentially a custom DMA engine built within the IP. These masters move data to/from the on-chip memory.

6.4.2.4 Clock Domains

You will notice that there are several different clock domains in this design. There is a fast and slow clock domain allowing the design to run the high-speed peripherals on the fast clock and the slow-speed peripherals on the slow clock. This enables the high-speed clock to run faster due to a lower fan-out (number of endpoints the net has to reach).

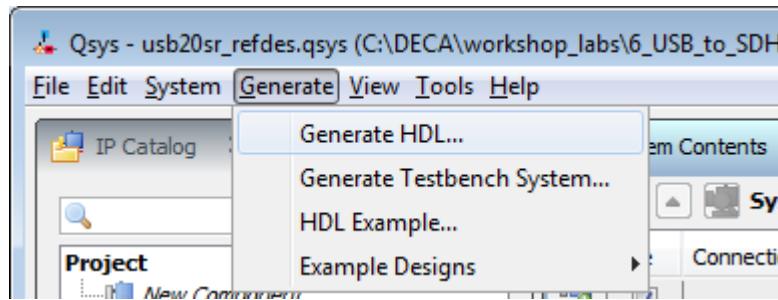
6.4.2.5 Browse the system

Browse around the rest of the system to review the dataflow for a better understanding of how the system is connected.

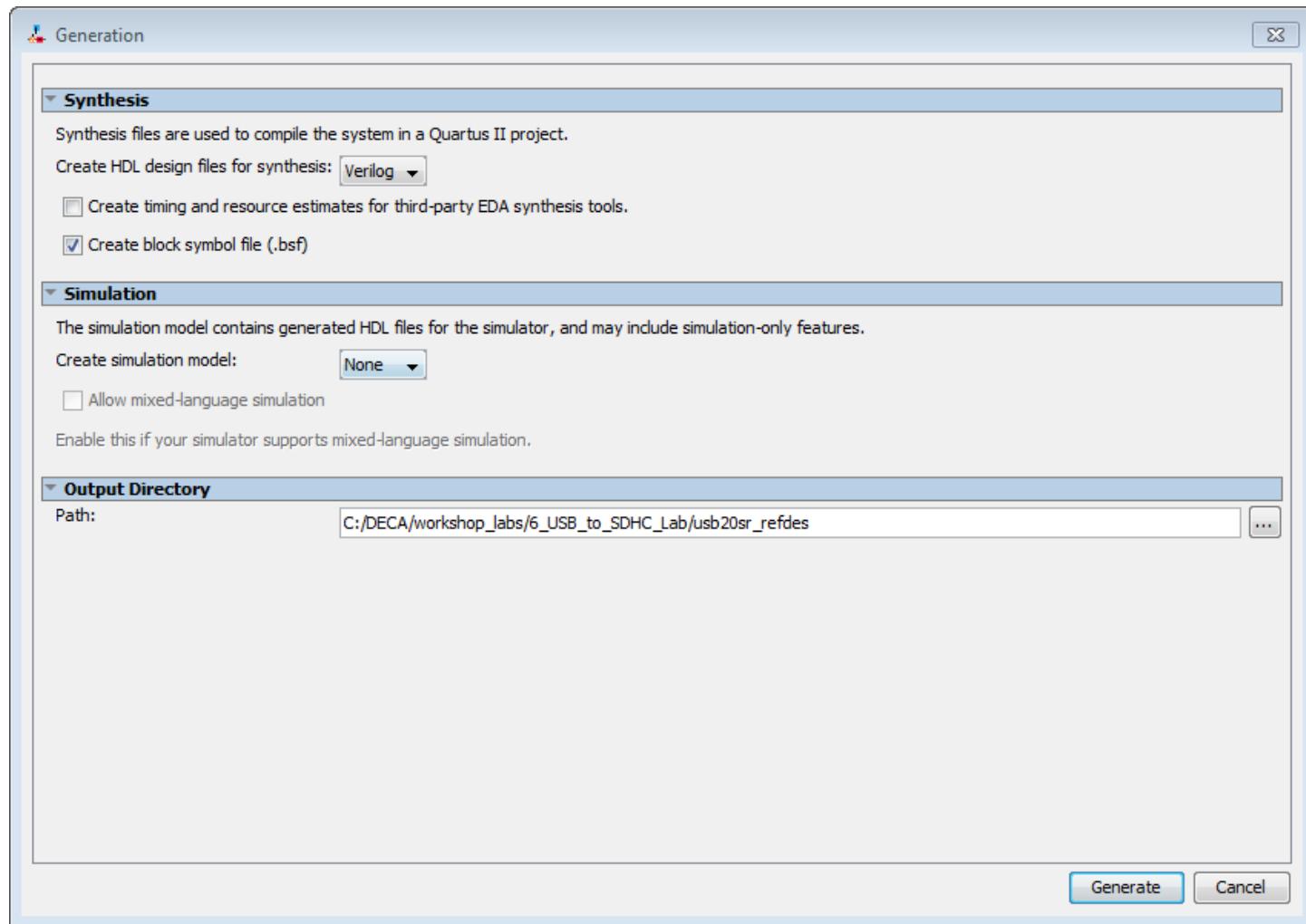
6.4.3 Generate the Qsys HDL Files for the project

Qsys generates HDL (hardware description language) code so that Quartus can compile this into an FPGA architecture.

In the Qsys tool window Select **Generate → Generate HDL...** from the Qsys menu or click the Generate button in the bottom right corner of the Qsys window

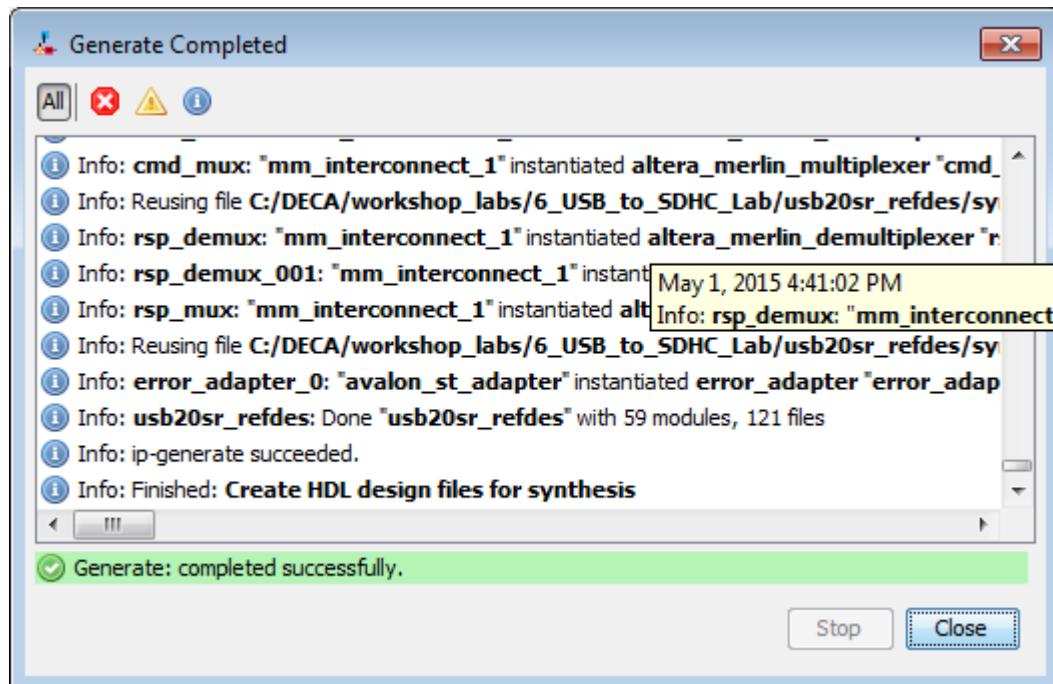


6.4.3.1 Select Output files you prefer or use the defaults as shown and click Generate

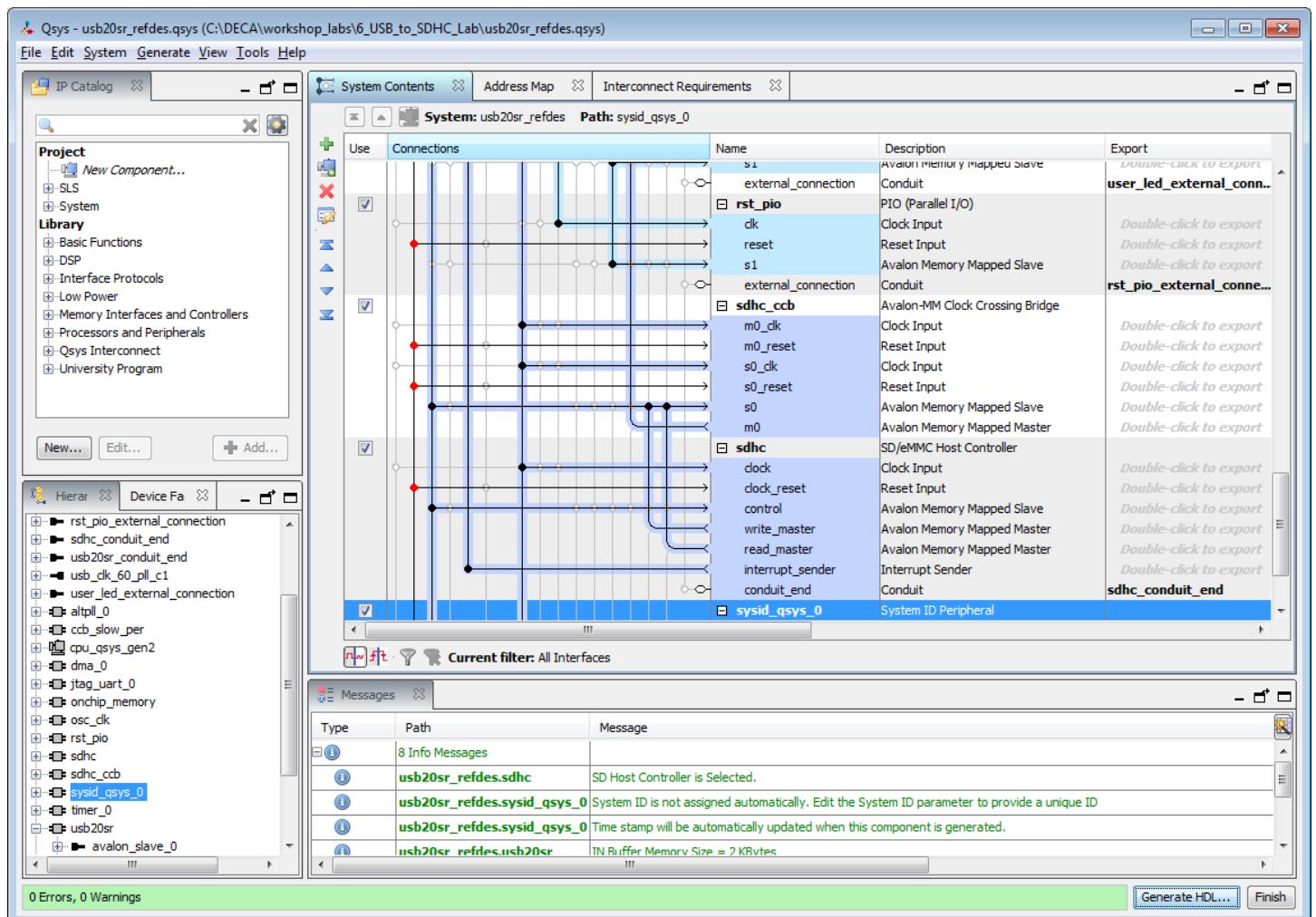


6.4.3.2 Finish Qsys HDL File Generation

When Qsys finishes generating the HDL files you will see the Generation Complete message box. Click Close.

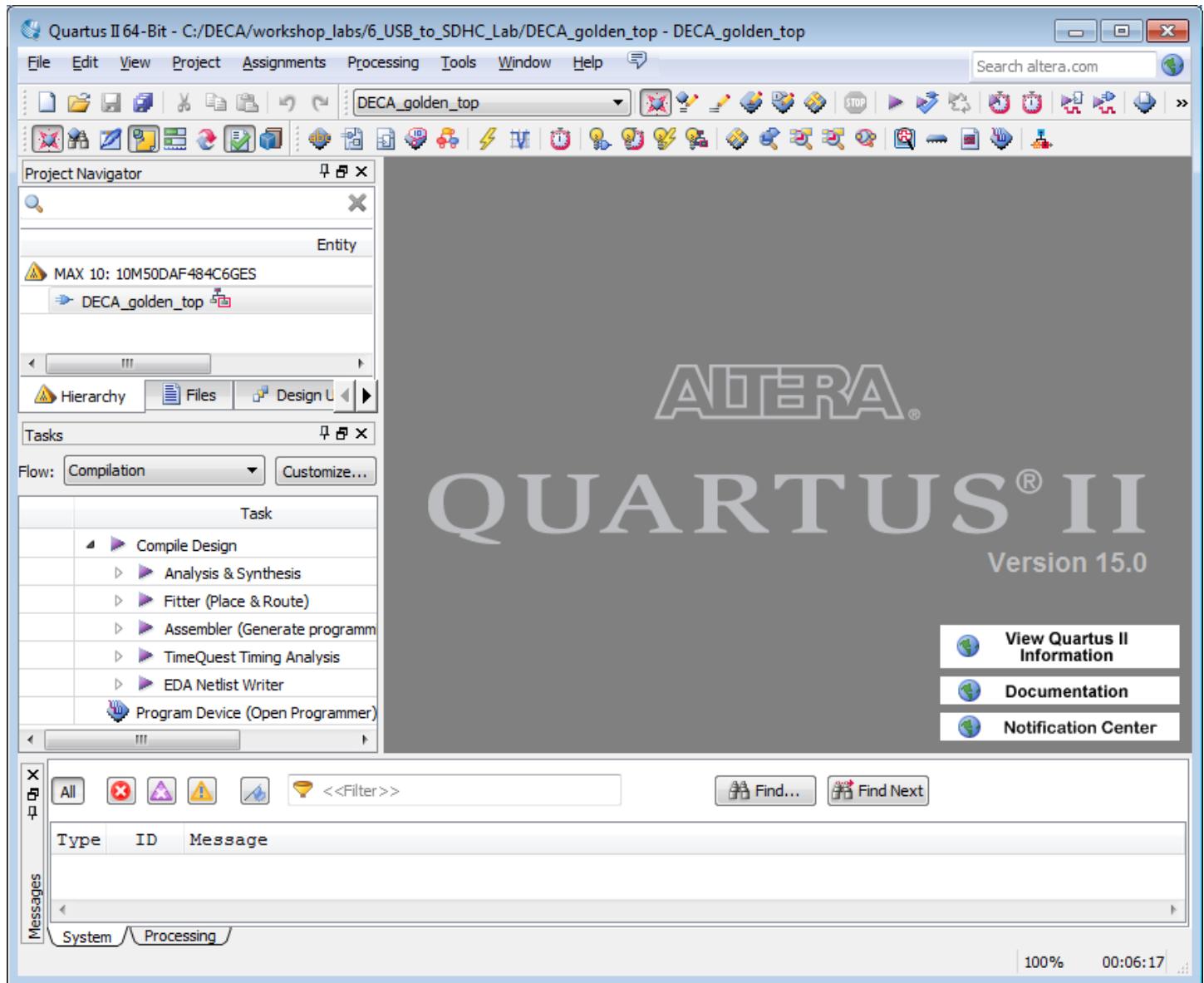


6.4.3.3 Finish Qsys by Clicking on Finish in bottom right of Qsys Tool window



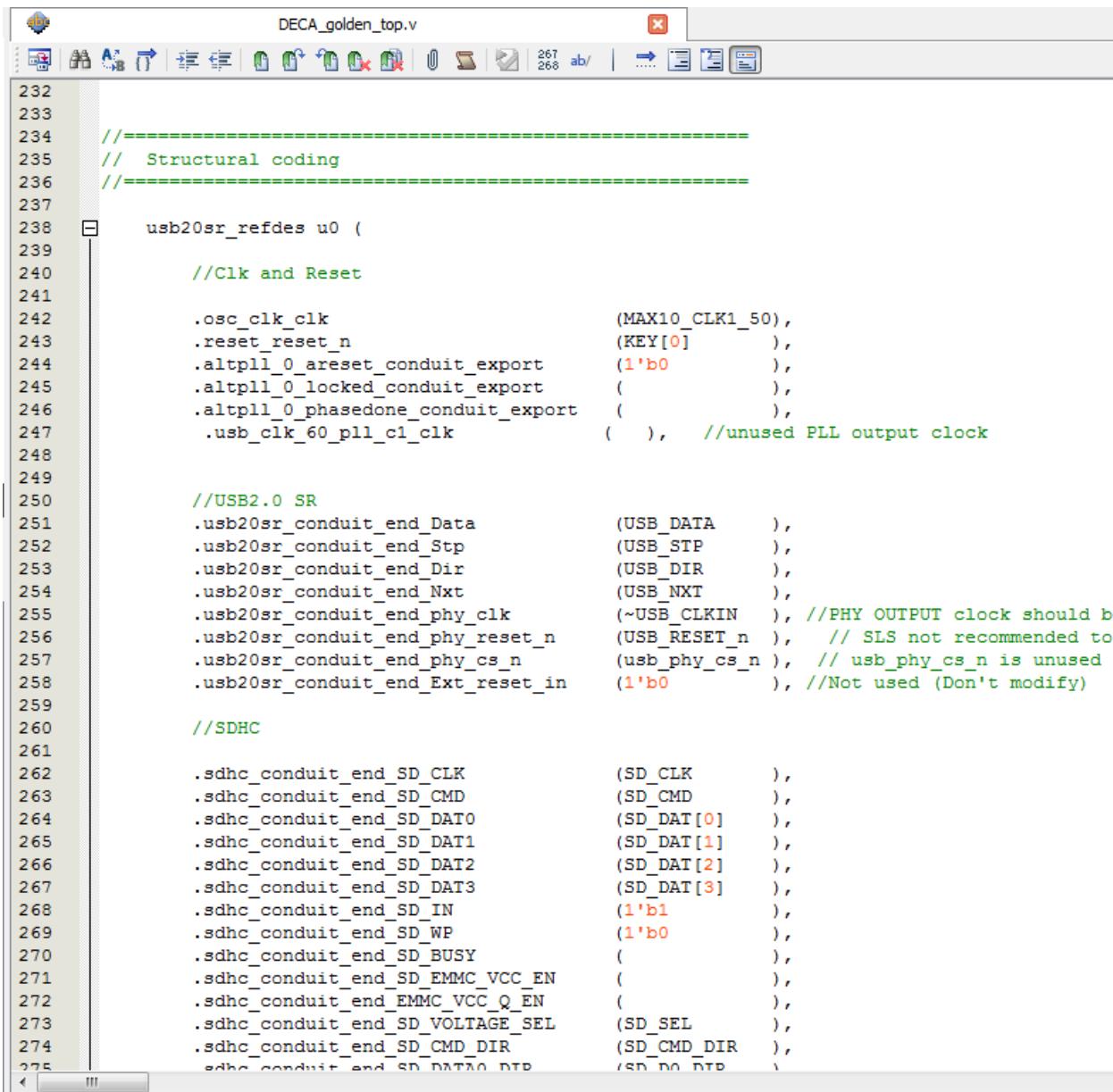
6.4.3.4 Compiling the Design in Quartus

Before compiling the design you can open the top-level design file by double-clicking **DECA_golden_top** in the Project navigator:



6.4.3.5 Explore the section where the Qsys system has been instantiated (`usb20sr_refdes`)

Here you can see all the ports of the Qsys system, and the connections that were made to the pins of the device.



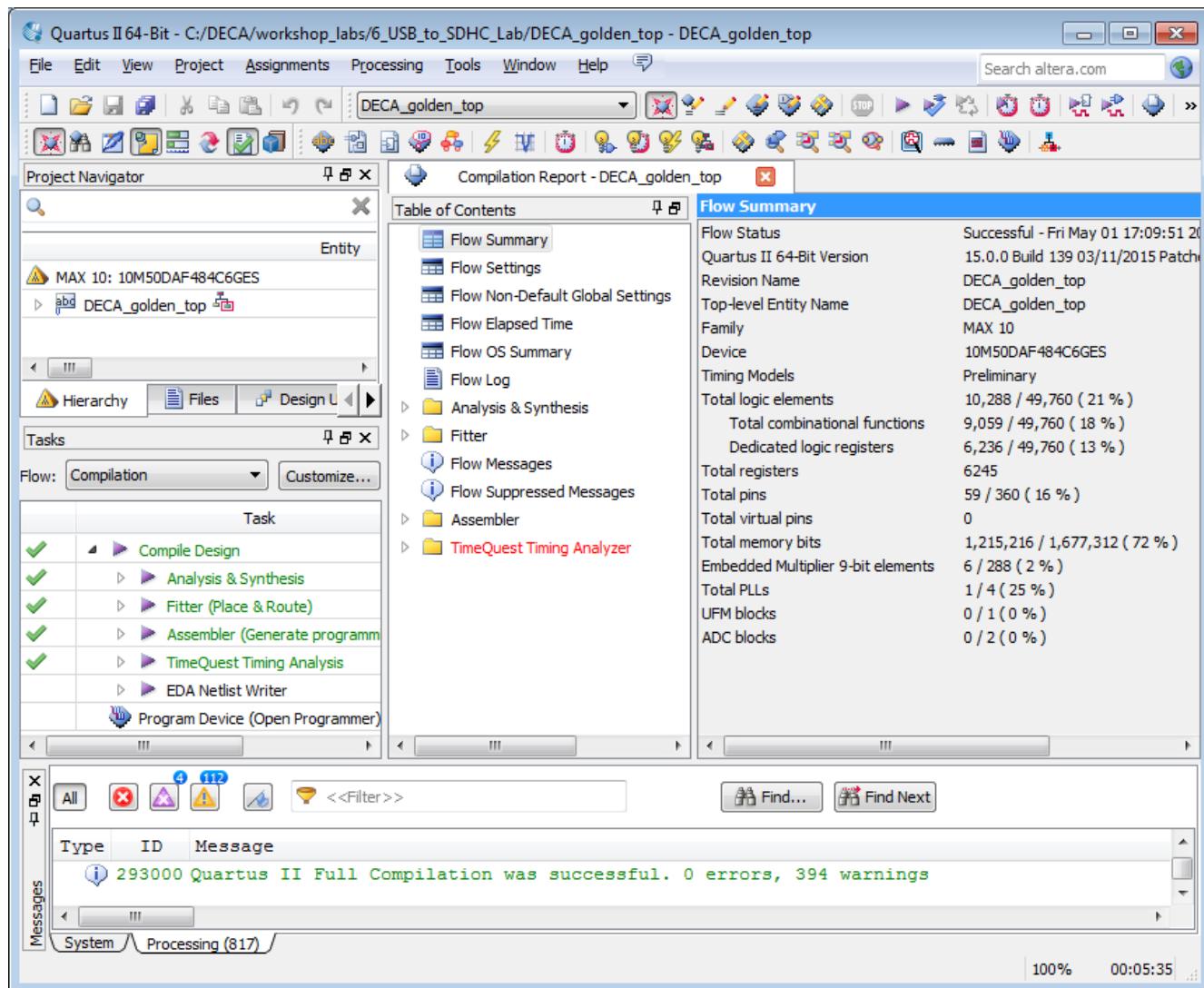
```

232
233
234 //=====
235 // Structural coding
236 //=====
237
238     usb20sr_refdes u0 (
239
240         //Clk and Reset
241
242         .osc_clk_clk          (MAX10_CLK1_50),
243         .reset_reset_n        (KEY[0]      ),
244         .altpll_0_areset_conduit_export  ('1'b0      ),
245         .altpll_0_locked_conduit_export ('1'b0      ),
246         .altpll_0_phasedone_conduit_export ('1'b0      ),
247         .usb_clk_60_pll_c1_clk    ('1'b0      ), //unused PLL output clock
248
249
250 //USB2.0 SR
251     .usb20sr_conduit_end_Data   (USB_DATA      ),
252     .usb20sr_conduit_end_Stp   (USB_STP       ),
253     .usb20sr_conduit_end_Dir   (USB_DIR       ),
254     .usb20sr_conduit_end_Nxt   (USB_NXT       ),
255     .usb20sr_conduit_end_phy_clk (~USB_CLKIN   ), //PHY OUTPUT clock should be
256     .usb20sr_conduit_end_phy_reset_n (USB_RESET_n ), // SLS not recommended to
257     .usb20sr_conduit_end_phy_cs_n (usb_phy_cs_n ), // usb_phy_cs_n is unused
258     .usb20sr_conduit_end_Ext_reset_in ('1'b0      ), //Not used (Don't modify)
259
260
261 //SDHC
262
263     .sdhc_conduit_end_SD_CLK   (SD_CLK       ),
264     .sdhc_conduit_end_SD_CMD   (SD_CMD       ),
265     .sdhc_conduit_end_SD_DATO  (SD_DAT[0]    ),
266     .sdhc_conduit_end_SD_DAT1  (SD_DAT[1]    ),
267     .sdhc_conduit_end_SD_DAT2  (SD_DAT[2]    ),
268     .sdhc_conduit_end_SD_DAT3  (SD_DAT[3]    ),
269     .sdhc_conduit_end_SD_IN    ('1'b1      ),
270     .sdhc_conduit_end_SD_WP    ('1'b0      ),
271     .sdhc_conduit_end_SD_BUSY   ('1'b0      ),
272     .sdhc_conduit_end_SD_EMMC_VCC_EN ('1'b0      ),
273     .sdhc_conduit_end_SD_VOLTAGE_SEL (SD_SEL     ),
274     .sdhc_conduit_end_SD_CMD_DIR ('1'b0      ),
275     .sdhc_conduit_end_SD_DATA0_N ('1'b0      )

```

6.4.3.6 Compile the design: Processing → Start Compilation

6.4.3.7 When compilation completes you should see the following in the Quartus Tool window.



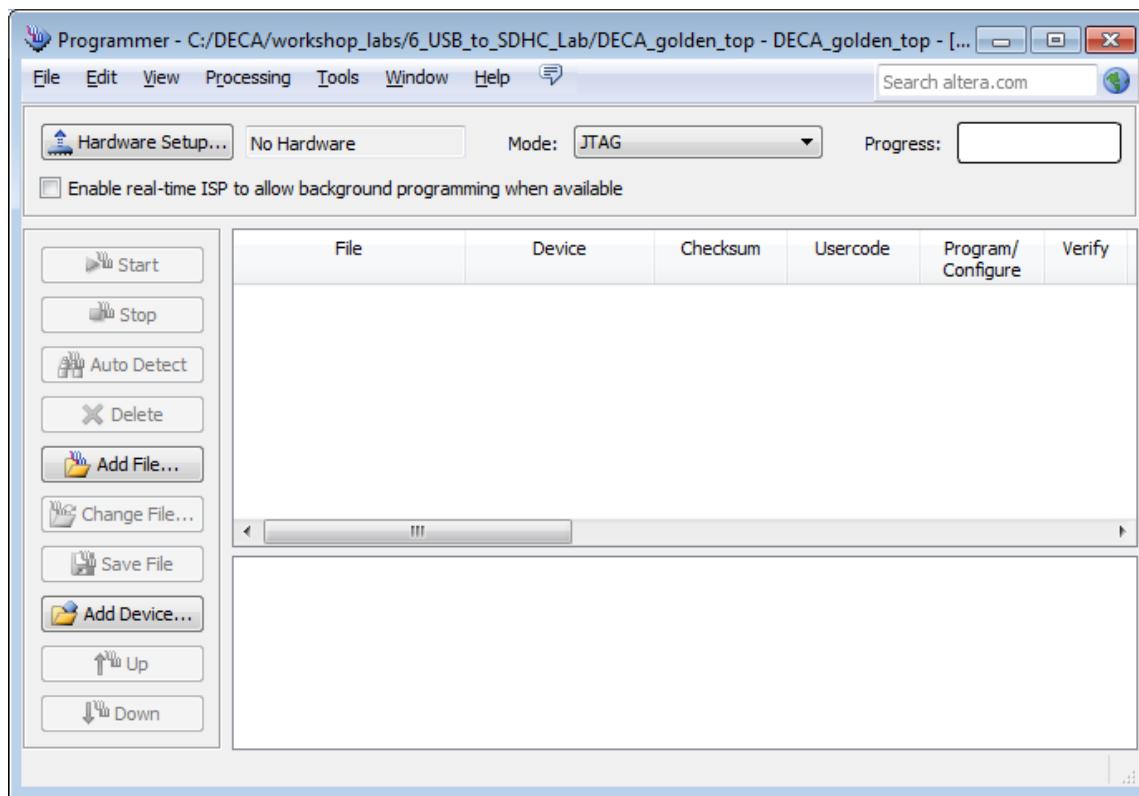
6.5 Download the Configuration File to DECA Board.

Overview: Once the design has compiled successfully, you can download the generated .sof file to the MAX 10.

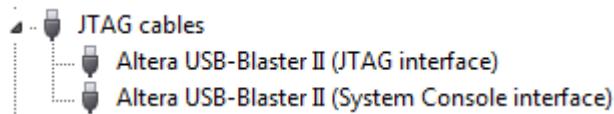
6.5.1 Open the Programmer Tool

6.5.1.1 From the Tools menu in the main Quartus II window, select **Tools → Programmer**.

6.5.1.2 Since the DECA isn't connected yet, the Programmer should show a blank configuration.



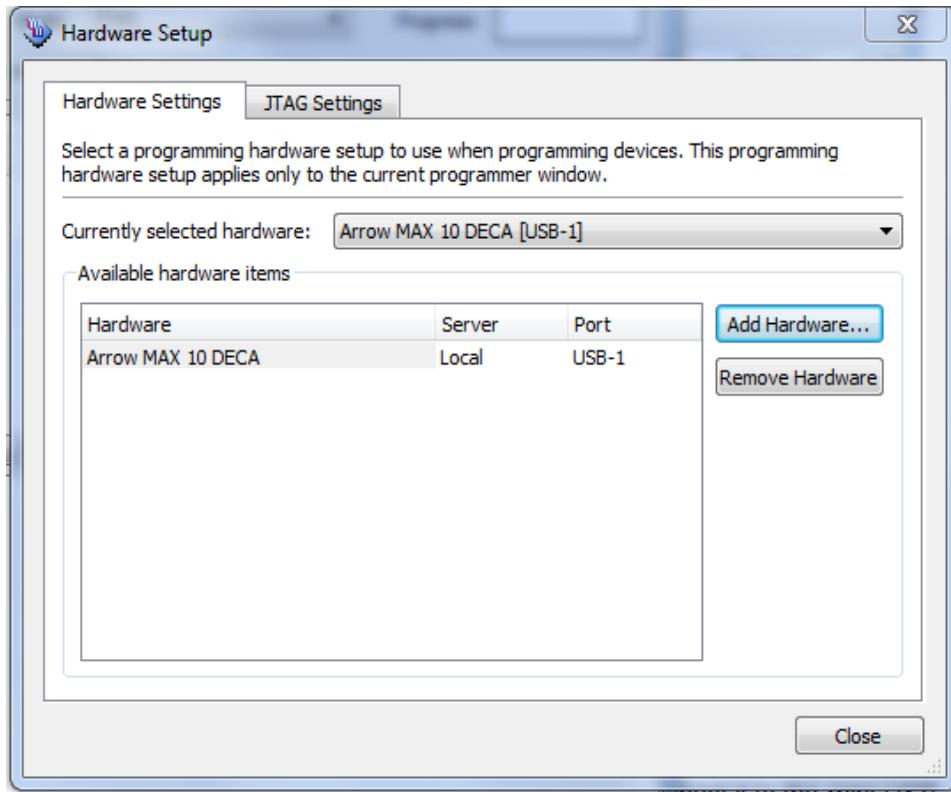
6.5.1.3 Connect your DECA board to your PC using a USB cable. Be sure to connect it to the mini-USB connector labeled **USB2 J10** (on the bottom right of the board). Since the USB Blaster II driver software should already be installed, the Window's Device Manager should display two entries under "JTAG Cables".



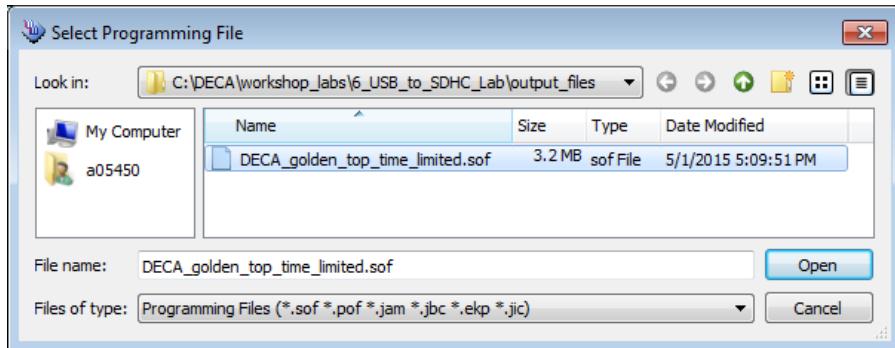
You should see the LEDs light up on your DECA including the blue LEDs labeled **3 . 3V** and the green LED labeled **CONF_D**.

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.

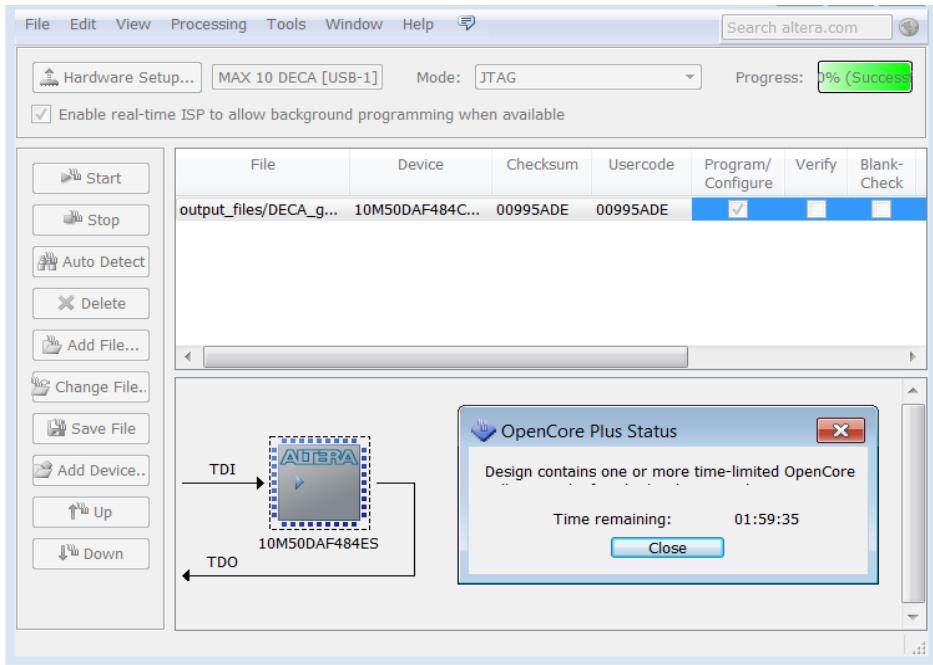
- 6.5.1.4 In the Programmer window, click Hardware Setup on top left. When the pop up below appears double-click the Arrow MAX10 DECA entry in the Hardware pane. This option will appear in the Programmer Tool window. Depending on your PC, the USB port number may be different. Click Close.



- 6.5.1.5 Click "Add File..." and browse to C:\DECA\workshop_labs\6_USB_to_SDHC_Lab\output_files in your compilation directory. Open the DECA_golden_top_time_limited.sof file. A pop up will appear mentioning that the IP in this design is time limited, click Open.



- 6.5.1.6 Make sure that the Programmer shows the correct file and the correct part in the JTAG chain as below.



- 6.5.1.7 Make sure the Program/Configure checkbox is checked and click Start to program the DECA board. You should see the CONF_D LED turn on to indicate that the configuration is complete and the Progress bar should reach 100% (Successful).



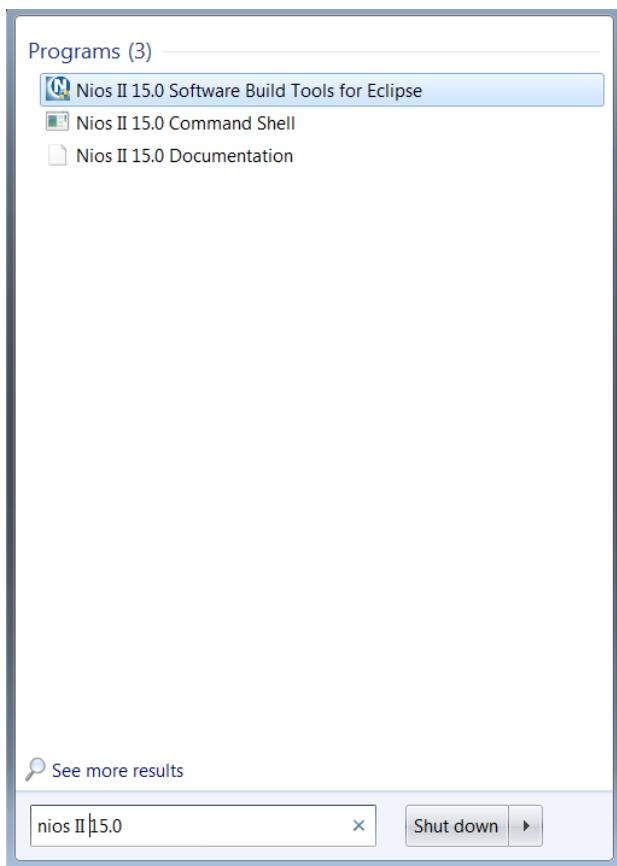
Note: There is an OpenCore Plus Status dialog box that pops up. Because the SLS USB and SDHC IP is licensed for trial purposes only, this IP will run for approximately two hours before it times out.

6.5.2 Create the Software Design

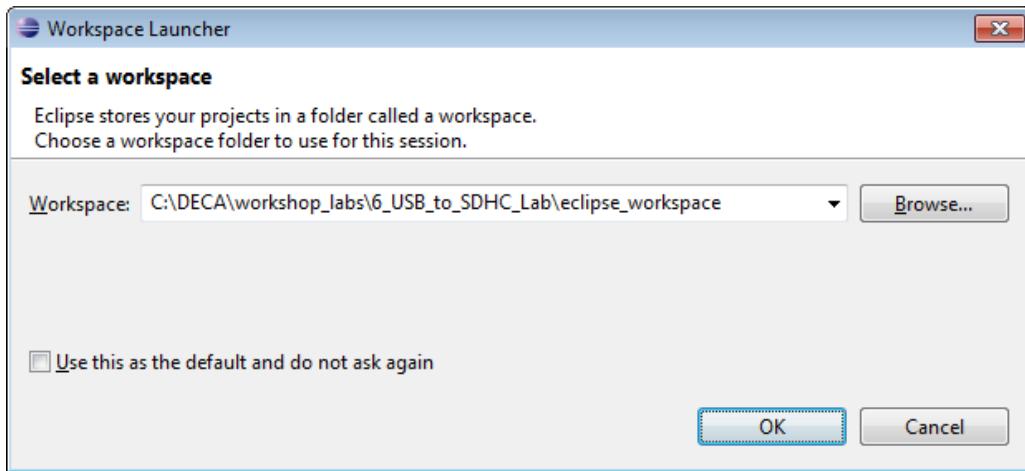
In this section, you will use the Nios II Software Build Tools (SBT) for Eclipse to create a board support package (BSP) and a C software application to run on the Nios II processor you implemented in the last section. After creating and compiling the project, you will run the executable on the Nios II enabling you browse files on the DECA Board SD Card.

6.5.3 Start Nios II Software Build Tools for Eclipse

- 6.5.3.1 From Windows, Select Start and type in "Nios II 15.0" and select Nios II 15.0 Software Build Tools for Eclipse.

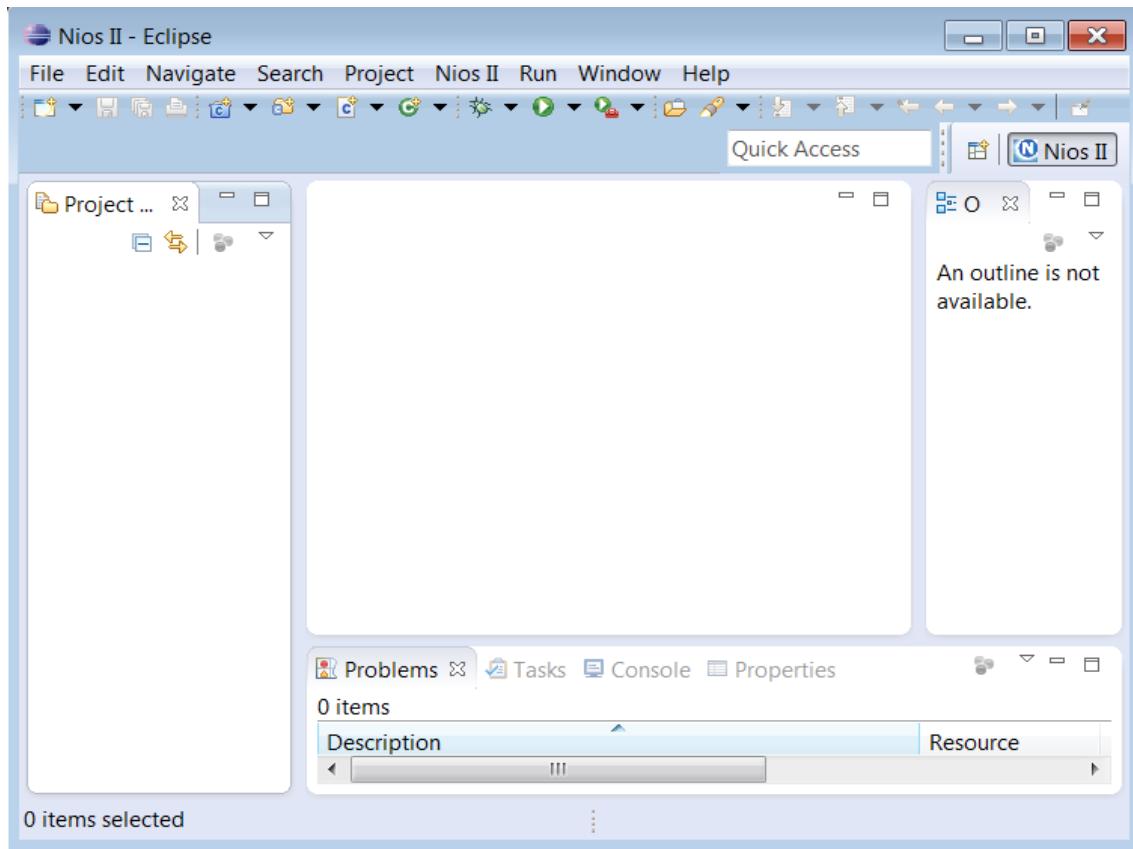


- 6.5.3.2 The Eclipse Workspace Launcher will open. Click "Browse..." and browse to the C:\DECA\workshop_labs\6_USB_to_SDHC_Lab folder that you created in the last section. Click "OK".
- 6.5.3.3 Add `eclipse_workspace` after the path in the window below, for a full path of:
C:\DECA\workshop_labs\6_USB_to_SDHC_Lab\eclipse_workspace



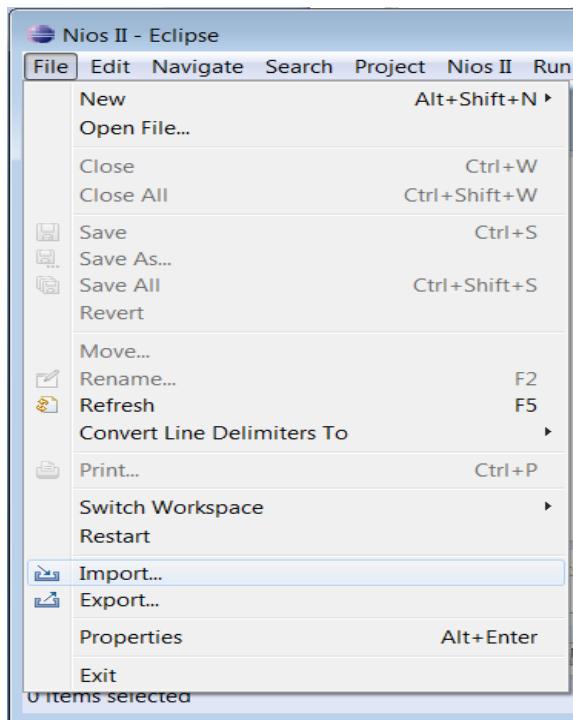
6.5.4 Eclipse Opens with a Blank Project

Now that Eclipse has a workspace, a new software application project and BSP can be created for your hardware system.

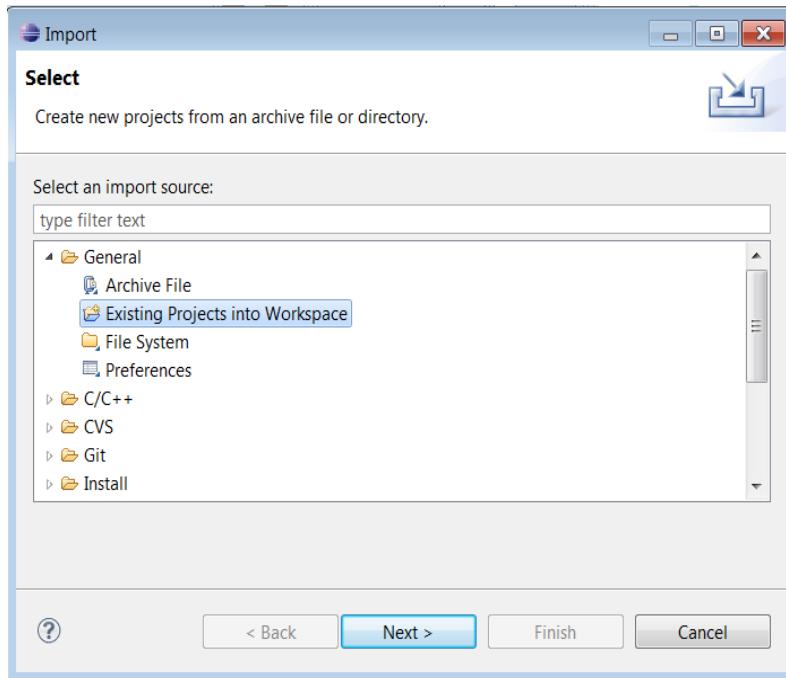


6.5.5 Open the USB2.0 Project Software Project

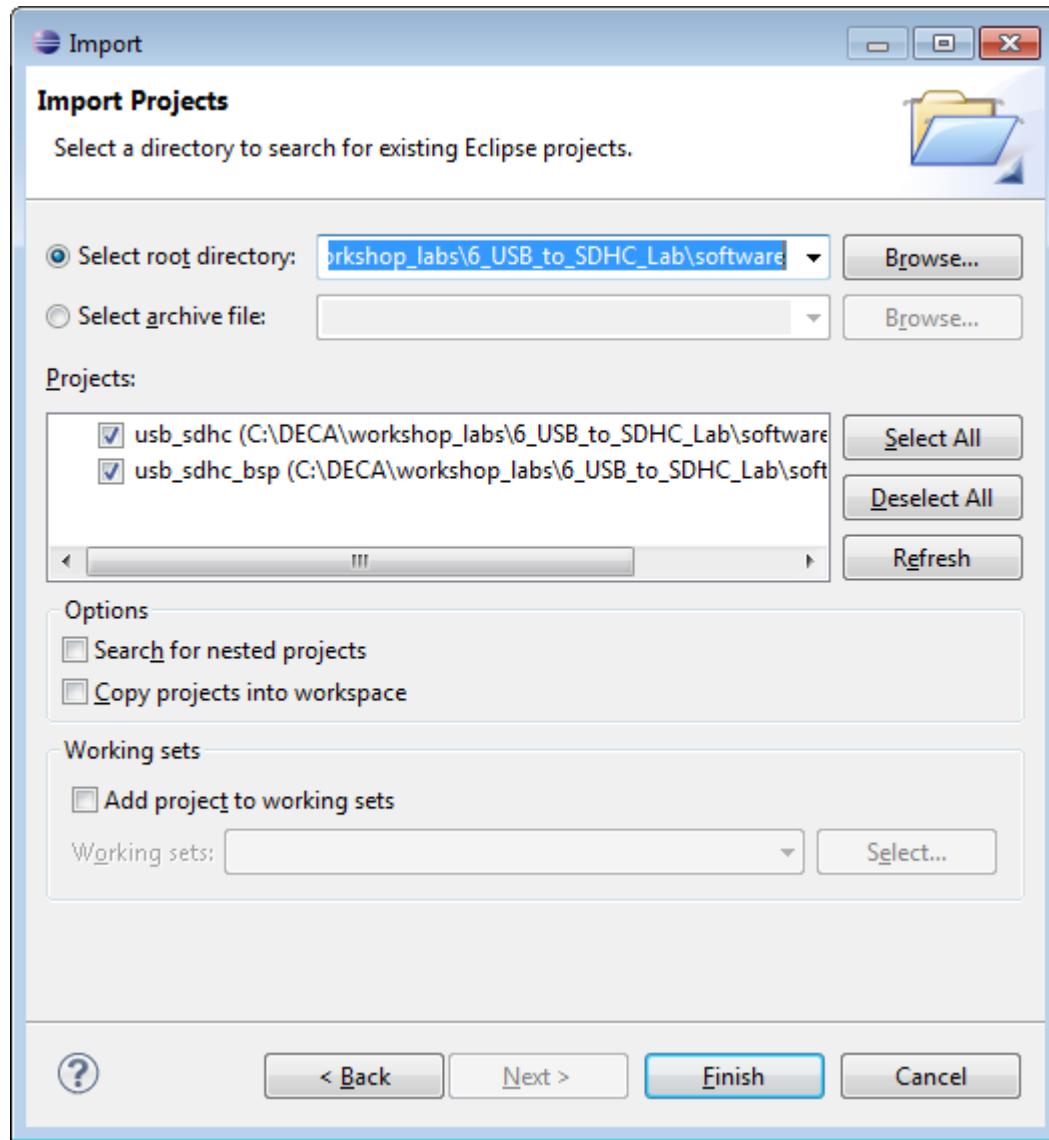
You can now load the design files as follows, select **File → Import**



6.5.5.1 The Import dialog box appears. Select **General → Existing Projects into Workspace** and Click Next

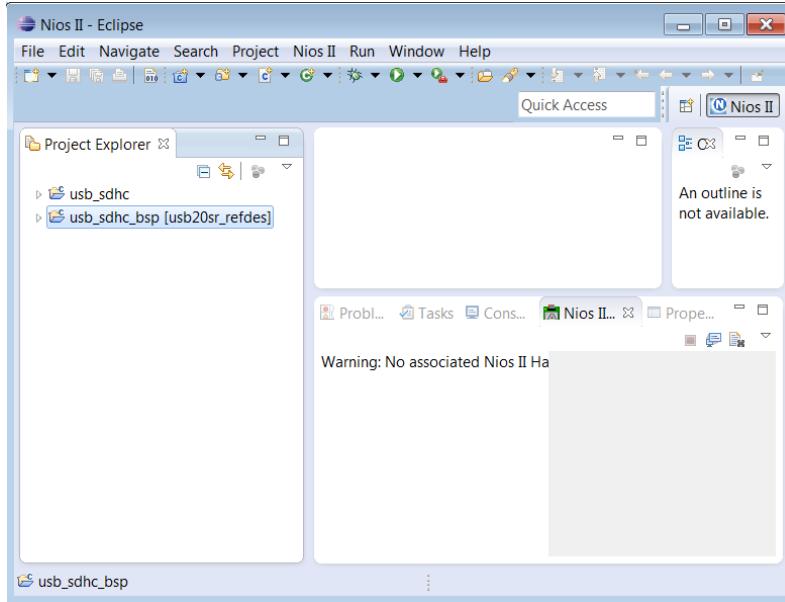


- 6.5.5.2 Browse to C:\DECA\workshop_labs\6_USB_to_SDHC_Lab and check the settings match the screenshot below. Select "Finish".

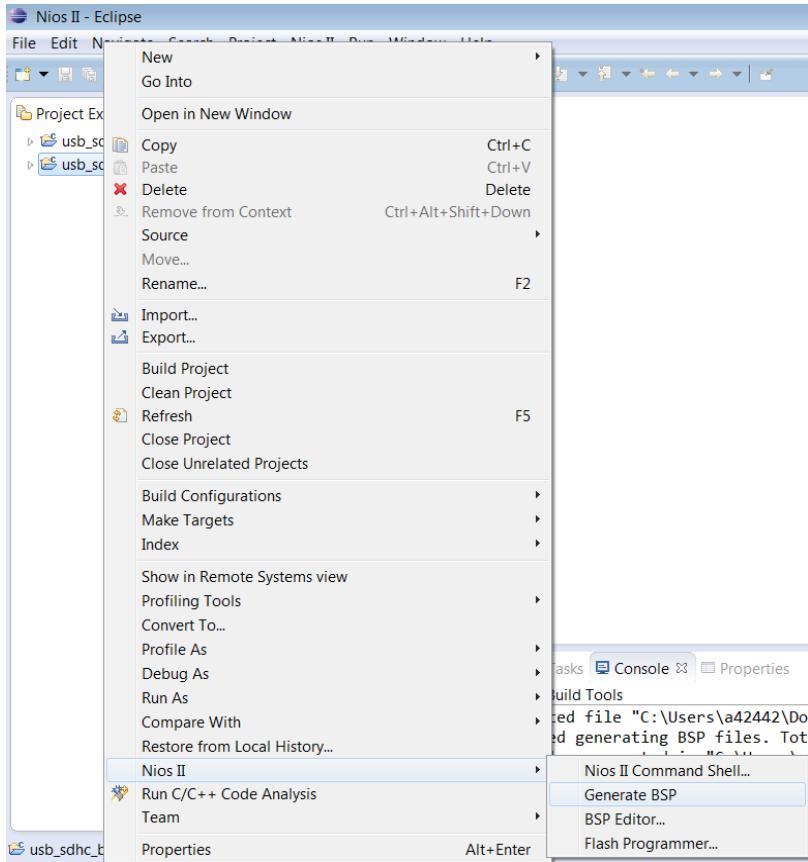


Now that the software project has been created, there are two steps involved in creating the executable file we need to run. The first is to generate the BSP, the second is to build (make) the project.

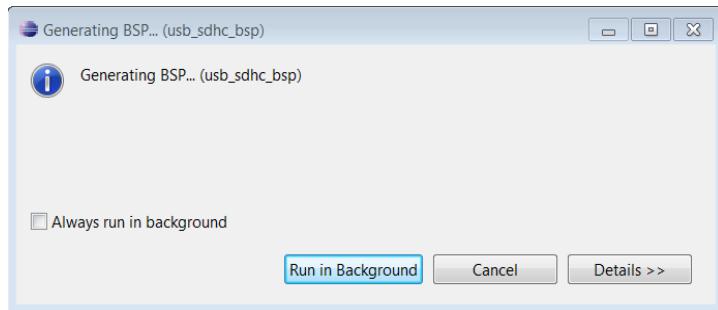
6.5.5.3 To generate the BSP, Right click the folder: **usb_sdhc_bsp** in Eclipse as shown:



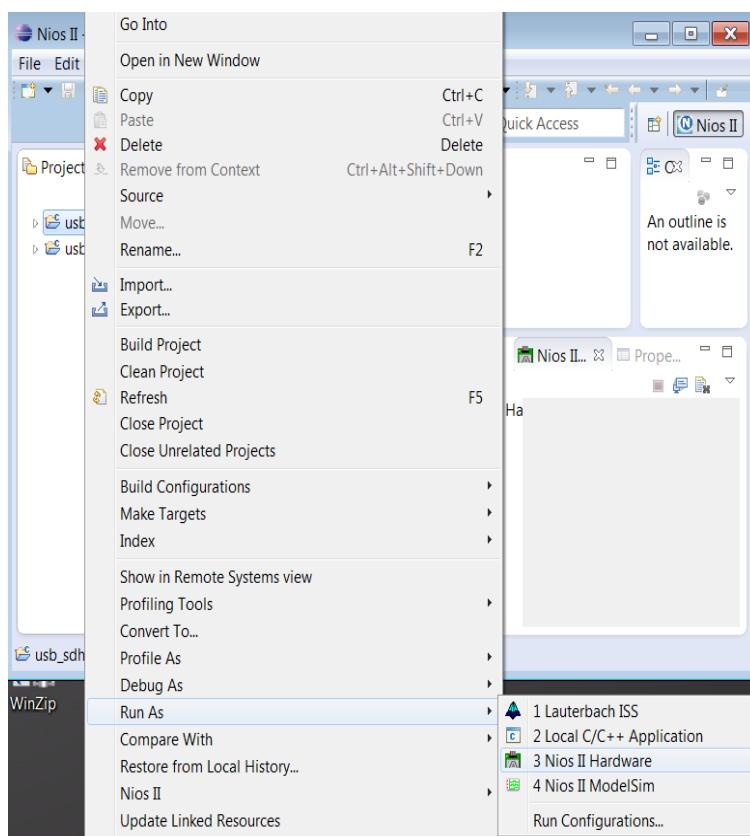
6.5.5.4 In the context menu that pops up, select **Nios II → Generate BSP**



6.5.5.5 The Generate BSP Dialog box will appear for a brief period



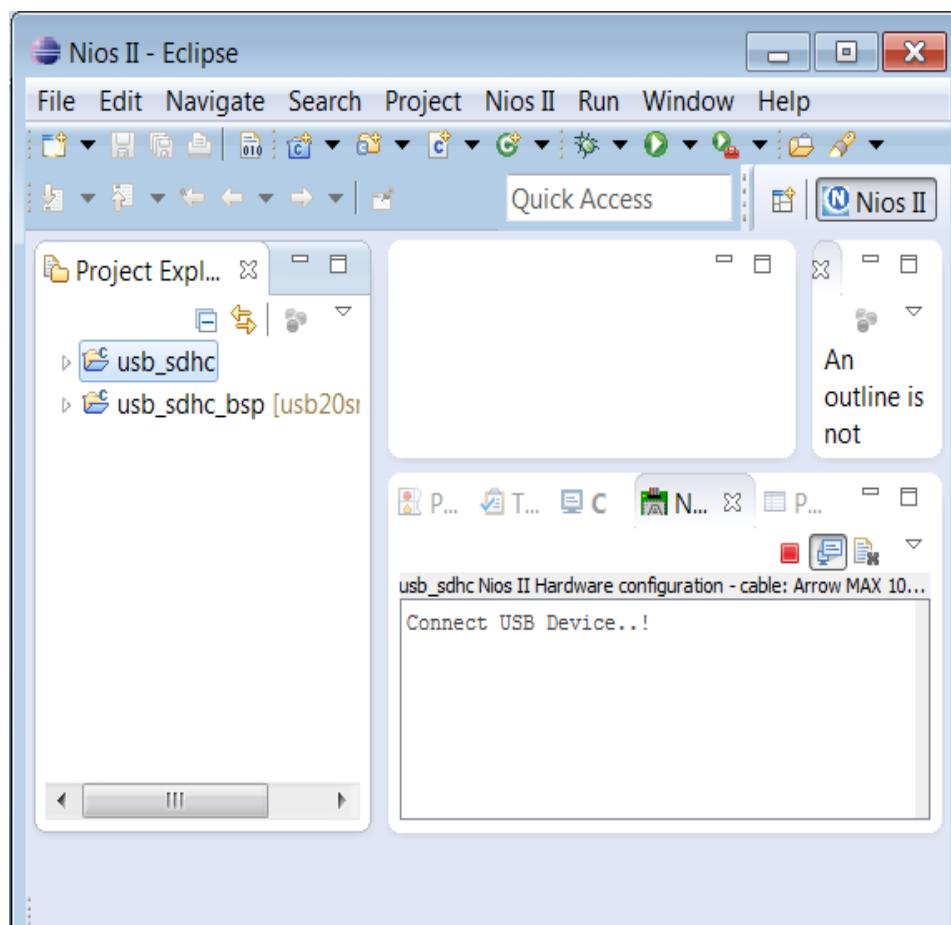
6.5.5.6 Build the executable code for the Nios II and Run it. Select usb_sdhc in Eclipse, right click and Select Run As → Nios II Hardware



This step will first build the software, then launch the Nios II download engine to download the ELF (Executable Linker File) to the Nios II processor's on-chip ram where execution begins.

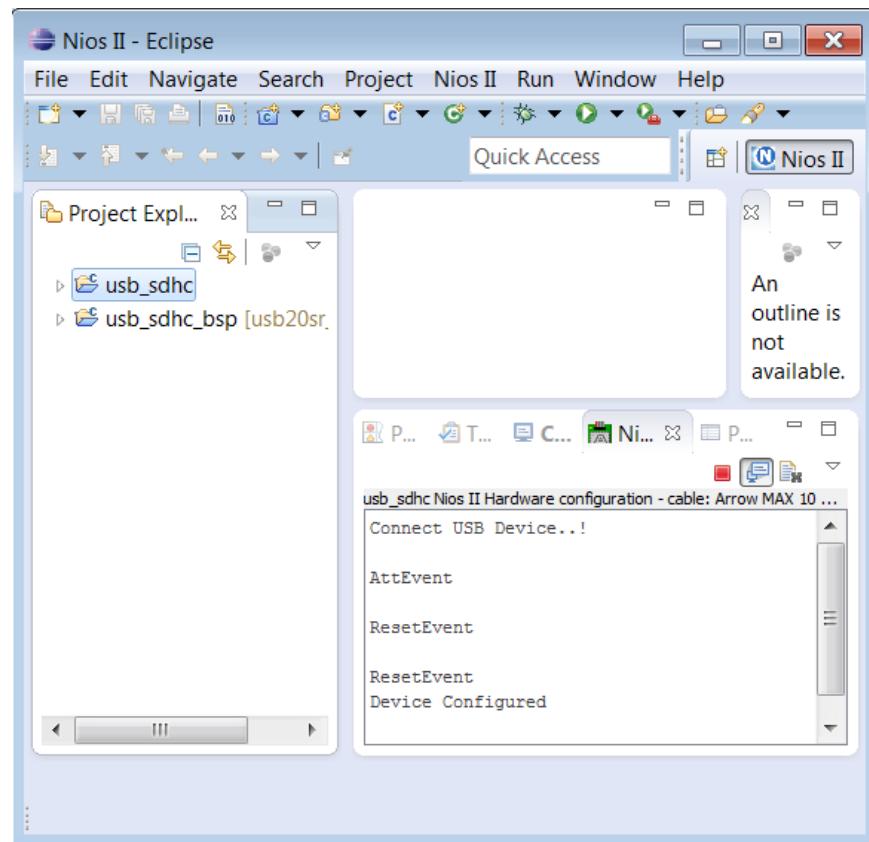
If communication to the Nios II processor hasn't been established (in some cases), you may need to open the Run Configurations Dialog box via the menu: **Run → Run Configurations...**. Go to the Target Connections tab and click Refresh Connections

6.5.5.7 You should now see the following message in the Eclipse console.

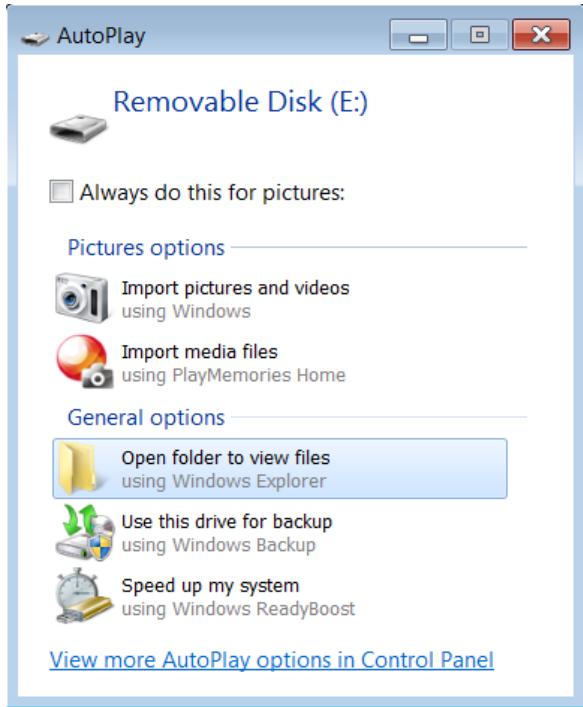


6.5.5.8 Connect your second USB Cable to the USB connector (J8 USB) and connect this to your computer

6.5.5.9 You should now see the following message in the Eclipse console. At this point you now have access to the SD Card on the DECA board as signified by the browser pop up.



6.5.5.10 Select "Open Folder to View files"



6.5.5.11 The DECA board is now setup as a USB mass storage device where you can move files or view files on the SDHC memory card.

6.5.5.12 Open the video on the SDHDC card located in **E:\Altera\Benefits of Dual Configuration Flash based FPGAs.mp4**

6.5.5.13 Feel free to explore at your own leisure.

Congratulations! you have completed the USB2.0 to SDHC Lab!

BLE / WIFI Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 7. INTERACT WITH DECA USING BLE AND WI-FI.....	270
7.1 System Overview and Architecture.....	270
7.1.1 Wi-Fi Subsystem	270
7.1.2 Bluetooth® Subsystem	271
7.2 Set up the DECA with the BLE/Wi-Fi Cape	272
7.2.1 Attach the BLE/Wi-Fi cape to the DECA.....	272
7.2.2 Open the Quartus II Project	272
7.2.3 Download the hardware configuration file (.sof) to the MAX10	273
7.2.4 Download the software executable (.elf) to the Nios II Soft Processor.....	275
7.2.5 Interact with the DECA board over Wi-Fi.....	278
7.2.6 Interact with the DECA board over Bluetooth® LE	282

LAB 7. INTERACT WITH DECA USING BLE AND WI-FI

Overview: In this lab, you will interact with the DECA platform from your Android or iOS smartphone over both Bluetooth® and Wi-Fi using the BeagleBone-compatible Wi-Fi cape from Dallas Logic. You'll be able to visit a webpage hosted on the DECA to see sensor data and change LEDs by connecting to the DECA as a Wi-Fi internet access point. You'll also be able to use a BLE app for iOS and Android to look at this data over Bluetooth® Low Energy.

MAX 10 FPGAs revolutionize non-volatile integration by delivering advanced processing capabilities in a low-cost, instant-on, small form factor programmable logic device. The devices also include full-featured FPGA capabilities such as digital signal processing, analog functionality, Nios II embedded processor support and memory controllers.

The DECA includes a variety of peripherals connected to the FPGA device, such as 4Gb DDR3L, MIPI camera interface, 10/100 Ethernet, temperature sensor, ambient light and gesture sensors, LEDs, capsense buttons and a BeagleBone compatible header.



Before continuing with this Tutorial, ensure that the Altera tools and drivers have been installed. Please refer to Lab 1 for instructions.

7.1 System Overview and Architecture

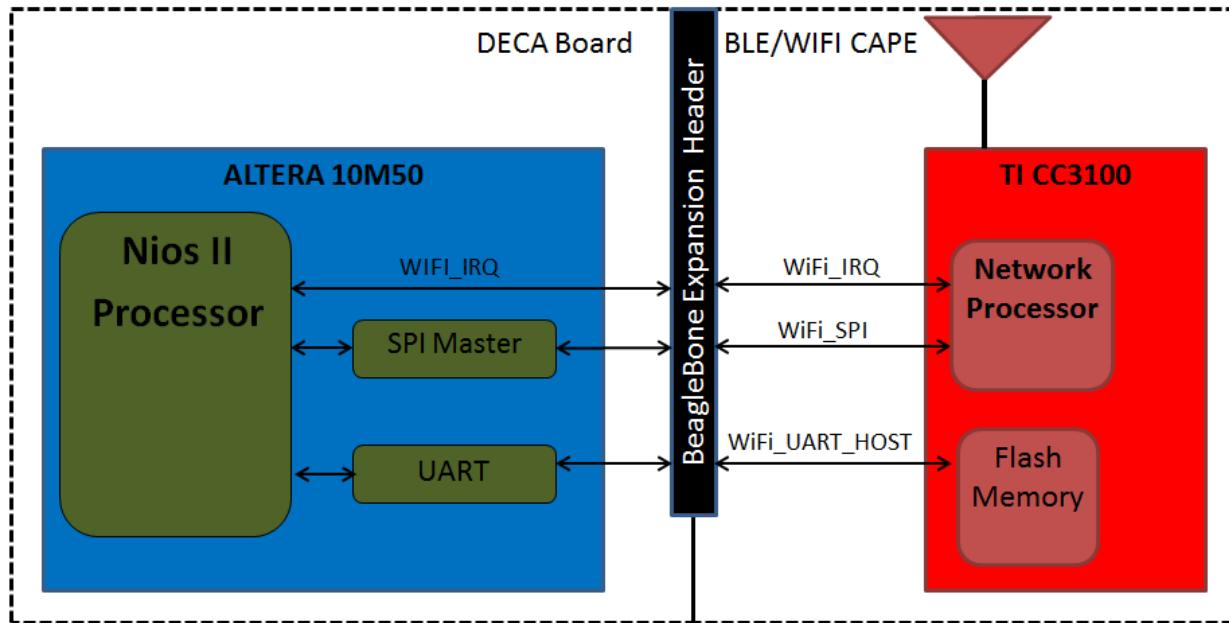
In this section, you will learn about how the MAX10 on the DECA interacts with the TI CC3100 Wi-Fi Network Processor and the CC2650 BLE chip on the BLE/Wi-Fi cape to create a wireless access point and a pairable Bluetooth® device. The Wi-Fi/BLE cape from Dallas Logic connects to the DECA board through the BeagleBone-compatible header giving the DECA board wireless capabilities.

7.1.1 Wi-Fi Subsystem

The Nios II processor communicates with the TI CC3100 Wi-Fi Network Processor over a SPI interface. The Nios II connects to a SPI master controller which manages data transfers to and from the CC3100. A parallel IO is used to implement the IRQ signal from the CC3100. Additionally, a UART interface is implemented to provide access to the on-chip flash memory on the CC3100. If the user wanted to change the website hosted on the CC3100 (as html code), the user would re-write the flash through this interface.

The CC3100 is a complete network processing solution on a single chip. The dedicated ARM MCU completely offloads Wi-Fi and Internet protocols from the host by implementing the entire TCP/IP stack, crypto and security engines, and an 802.11 b/g/n radio. The CC3100 has a dedicated on-board antenna which it uses to broadcast its wireless signal. A power management subsystem includes DC-DC converters and enables low power consumption modes such as hibernation drawing only 4 μ A of power.

7.1.1.1 Below is the Wi-Fi subsystem block diagram.

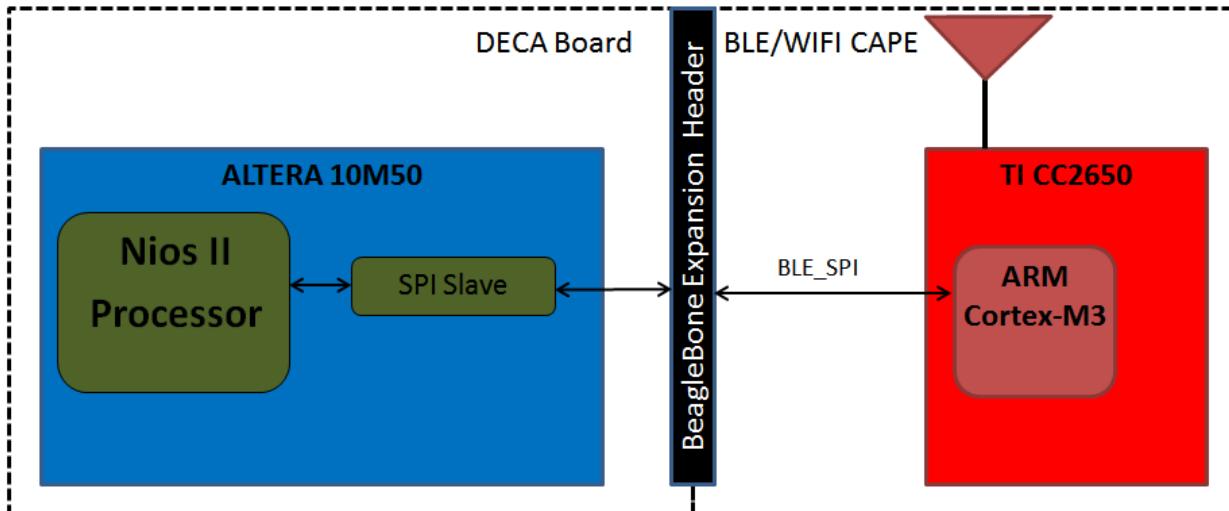


7.1.2 Bluetooth® Subsystem

The Nios II processor communicates with the CC2650 BLE MCU over a SPI interface. The Nios II connects to a SPI slave peripheral and receives data and requests from the CC2650 SPI master.

The CC2650 contains a 32-bit ARM Cortex-M3 processor to implement the Bluetooth® Low-Energy (Bluetooth Smart®) stack and the 802.15.4 MAC which are both embedded into on-chip ROM. The M3 processor manages all data transfers between the connected Bluetooth device and the Nios II processor, abstracting the BLE protocol away from the Nios II host. The CC2650 has a dedicated on-board antenna to send and receive data over the Bluetooth link.

7.1.2.1 Below is the Bluetooth® subsystem block diagram.



7.2 Set up the DECA with the BLE/Wi-Fi Cape

In this section, you will connect the BLE/Wi-Fi cape from Dallas Logic to the DECA board using the BeagleBone compatible header. Then you will download the hardware and software programming files to the board to get it ready to interact with.

7.2.1 Attach the BLE/Wi-Fi cape to the DECA

7.2.1.1 Place the DECA board on the table with the blue/green audio jacks pointing toward you.

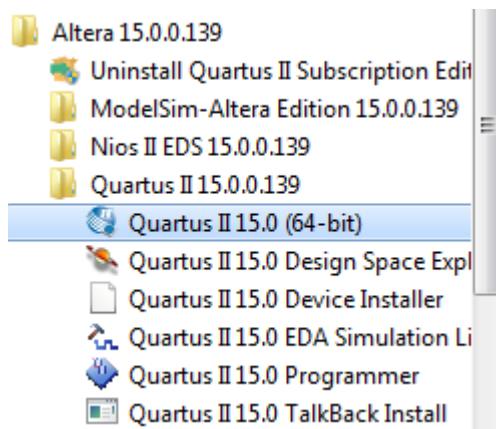
7.2.1.2 Align the BLE/Wi-Fi cape so that the Arrow logo is in the readable orientation and press the male header pins on the cape firmly into the female header socket on the DECA. The final assembly should match the image below. Ensure power is OFF.

Note: If you connect the cape incorrectly, you can permanently damage the hardware!
Ensure that the board matches the picture below



7.2.2 Open the Quartus II Project

7.2.2.1 Launch Quartus II 15.0 (64-bit) from the Start menu.



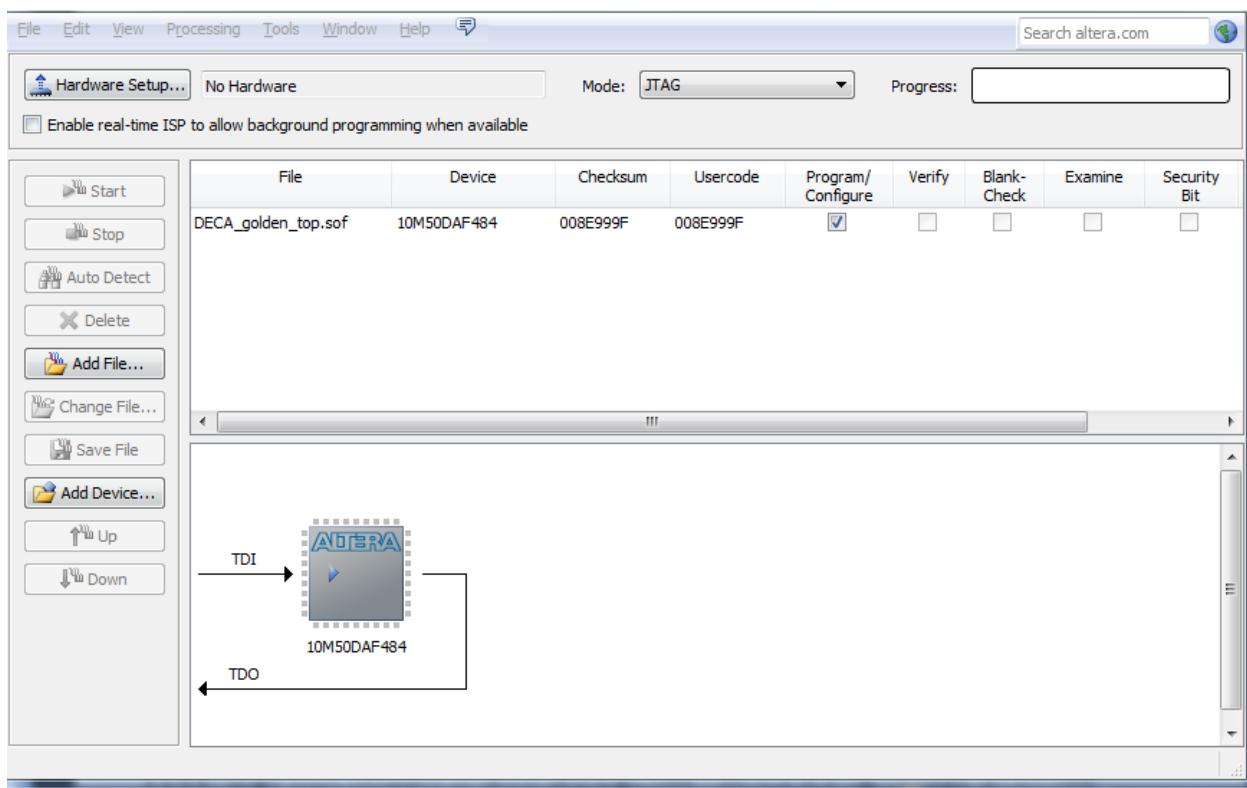
7.2.2.2 Open the workshop project from **File → Open Project**.

7.2.2.3 Browse to the directory where you extracted the lab files (for example `C:\DECA\workshop_labs\7_BLE_WIFI_Lab`) and open `DECA_golden_top.qpf`.

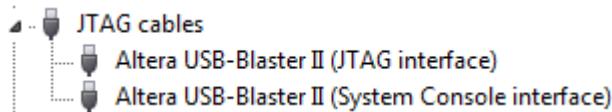
Feel free to explore the hardware source files, namely the top level file `deca_golden_top.v`. In this file, you'll see the top-level ports which are connected to pins on the DECA board, reg/wire declarations and some structural coding including instantiations for two components. One of these, named `RF0002_BASE`, contains a Nios II soft core processor that will be running the software code.

7.2.3 Download the hardware configuration file (.sof) to the MAX10

7.2.3.1 Open the Quartus II Programmer from **Tools → Programmer** or double-click on Program Device (Open Programmer) from the Tasks pane. The Programmer should open with a pre-defined configuration showing the appropriate device and programming file selected.



7.2.3.2 In the same orientation as above, plug in the USB cable included with your kit to the top USB connector labeled **J10 USB2**. Since the USB Blaster II driver software should already be installed, the Windows's Device Manager should display two entries under "JTAG Cables".

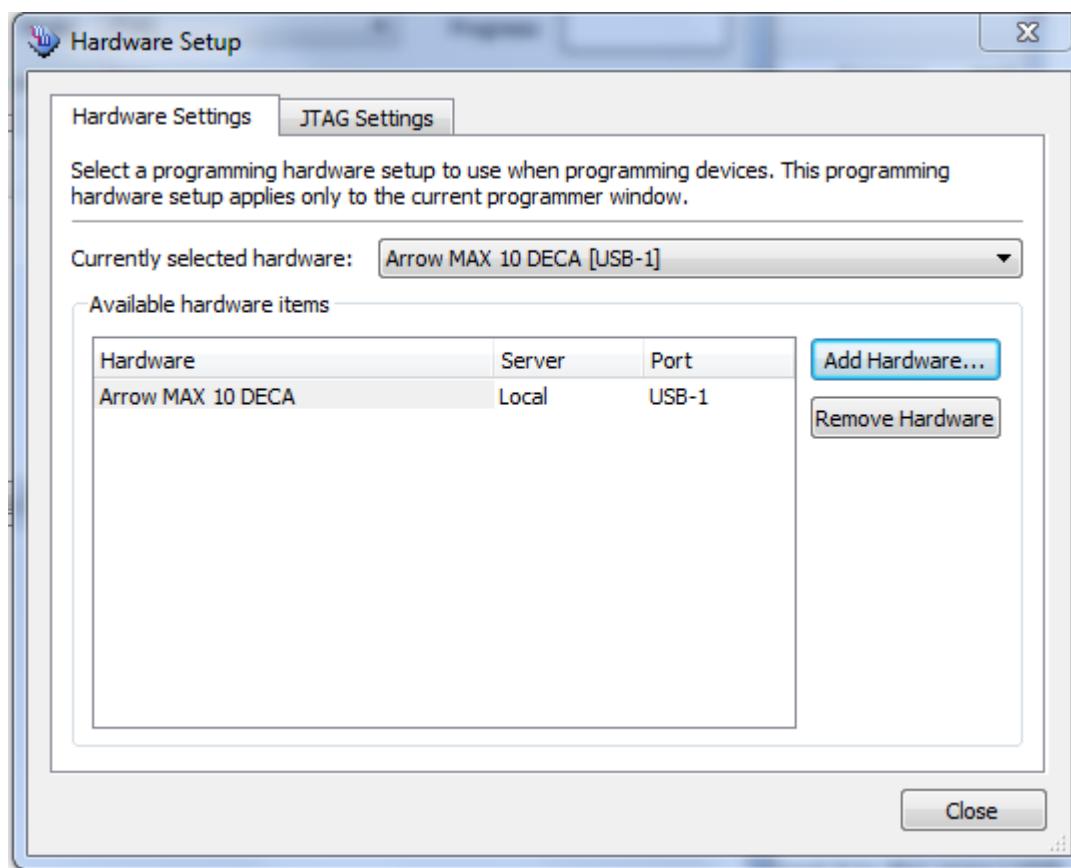


You should see a few LEDs light up on your DECA including the blue LED labeled **3.3V** and the green LED labeled **CONF_D**.



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.

- 7.2.3.3 Click the Hardware Setup... button in the upper left of the Programmer window. 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.



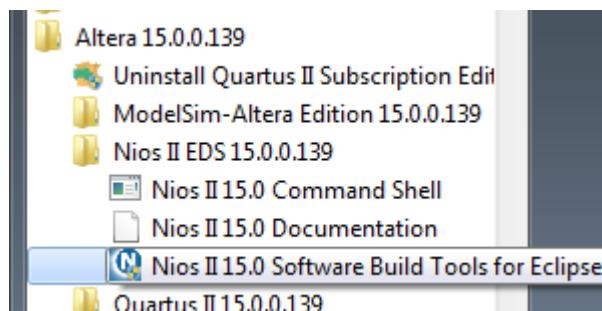
- 7.2.3.4 Make sure the Program/Configure checkbox is checked and click Start to program the DECA. You should see the CONF_D LED illuminate to indicate that the configuration is complete and the Progress bar in the Quartus II programmer should reach 100% (Successful). Additionally, all 8 blue LEDs should be on.

Progress: 100% (Successful)

7.2.4 Download the software executable (.elf) to the Nios II Soft Processor

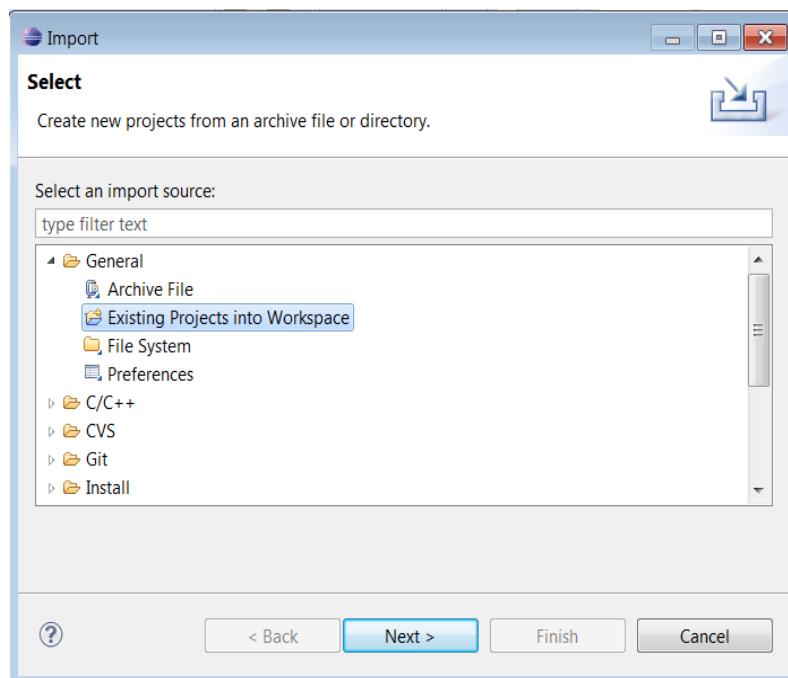
Now that the MAX10 has been programmed with the necessary hardware configuration, the software executable needs to be downloaded to the memory for the Nios II processor to begin executing it.

7.2.4.1 Launch Nios II 15.0 Software Build Tools

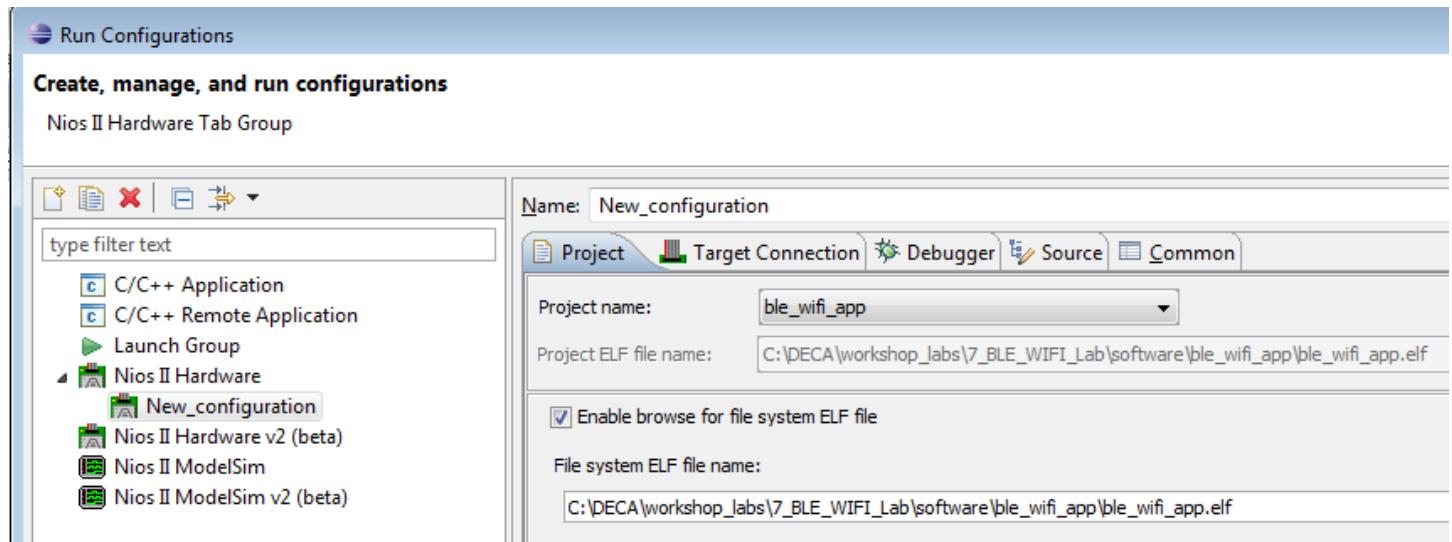


7.2.4.2 Import both the application and the bsp project folders using the menu: **File → Import**

7.2.4.3 The Import dialog box appears. Select **General → Existing Projects into Workspace** and Click Next



- 7.2.4.4 Browse to `C:\DECA\workshop_labs\7_BLE_WIFI_Lab\software` folder and select both `ble_wifi_app` and `ble_wifi_bsp` projects. Select "Finish".
- 7.2.4.5 For this step, we will not build the software project but merely just run the existing elf (executable linked format) file.
- 7.2.4.6 In the menu: **Run → Run Configurations...** Click **Nios II Hardware** along the left pane, and press the new button ()
- 7.2.4.7 A new configuration will appear. In the Project tab on the right-hand pane, check the Enable browse for system ELF file and browse to the `ble_wifi_app.elf` file as shown:



7.2.4.8 In the Target Connection tab, click the Refresh Connections if the Run button has been grayed out.

7.2.4.9 Click Run to launch the ble_wifi_app.

When the ble_wifi_app launches, it will run through several lines of code before stopping and asking the user for some input as shown

```
Problems Tasks Console Nios II Console X Properties
New_configuration - cable: Arrow MAX 10 DECA on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0
1
2
[NETAPP EVENT] IP Acquire
3
4
5
6
7
Host Driver Version: 1.0.0.1
Build Version 2.0.7.0.31.0.0.4.1.1.5.3.3
8
9
a
b
c
d
e
f
10
11
12
Device is configured in default state
13
14
15
16
17
18
[NETAPP EVENT] IP Acquire
Please input the SSID name for AP mode:
```

7.2.4.10 Create a custom SSID for the WIFI access point. Enter any name you wish, up to 32 characters long, then press enter

7.2.4.11 Enter what type of encryption you want for this session. Press 1 for Open encryption

7.2.5 Interact with the DECA board over Wi-Fi

The TI CC3100 on the BLE/Wi-Fi cape has now been configured as a Wi-Fi access point (AP). In this section, you will connect to the AP and send/receive data to the DECA board over the link.

7.2.5.1 From your smartphone or laptop, connect to SSID you created just as you would with any other wireless network. The access point has no security so your device should have no trouble connecting.

The Nios II Console should report a few more lines of information showing that the AP lease has been acquired and a client is connected.

```
[NETAPP EVENT] Lease Aquire  
1a  
1b  
Configured CC3100 in AP mode, Restarting CC3100 in AP mode  
1c  
1d  
1e  
1f  
[NETAPP EVENT] IP Acquire  
Connect client to AP test  
  
20  
21  
[NETAPP EVENT] Lease Aquire  
Client connected  
22  
23  
  
Domain name = mysimplelink.net 24  
Device URN = mysimplelink
```

7.2.5.2 In your web browser, type the domain name given by the terminal into the URL field i.e.

www.mysimplelink.net. You should see the following web page showing status information for the Wi-Fi access point. Feel free to scroll down and explore the Status page.

The screenshot shows a web-based status interface for a DallasLogic DECA module. At the top, there are logos for DallasLogic, Texas Instruments, and the DECA module itself. Below the header, a navigation menu includes links for Status, Accelerometer, Temperature & Humidity, and LED & Push Button. The main content area is titled "Status" and contains three sections: "Device", "Station (and P2P client)", and "Access Point (and P2P Go)".

Device	
Device Name:	mysimplelink
Device Mode:	Access Point
MAC Address:	D0:5F:B8:4D:64:22

Station (and P2P client)	
DHCP State:	Enabled
IP Address:	0.0.0.0
Subnet Mask:	0.0.0.0
Default Gateway:	0.0.0.0
DNS server:	0.0.0.0

Access Point (and P2P Go)	
Channel No:	6
SSID:	DallasLogicWIFI
Security Type:	Open

- 7.2.5.3 Click the link for the accelerometer, temperature & humidity data. The new page displays instantaneous accelerometer, temperature and humidity data from the various sensors on the board. A live feed of this data can be enabled by clicking the "Start Continuous" button on the page.

The screenshot shows a web browser window titled "SimpleLink(TM) - CC31xx". The address bar contains "www.mysimplelink.net". Below the address bar, there are several bookmarks: "Apps", "MAX10 Campaign L...", "Journey Program - ...", "Google", and "Other bookmarks". The main content area displays logos for Dallas Logic, Arrow, and Texas Instruments. Below these logos is a navigation menu with three items: "Status", "Accerometer, Temperature & Humidity", and "LED & Push Button". Underneath the menu, the text "WIFI BLE Info." is centered. To its right is a "Start Continuous" button. The main data section is titled "Temperature:" and shows the value "40.22407531738281" in a text input field. Below it are four more text input fields: "Humidity:" with value "18.3837890625", "X-axis:" with value "1981", "Y-Axis:" with value "2010", and "Z-Axis:" with value "1269".

Tilt the DECA board in various directions to see if you can determine the X, Y and Z orientations of the board.

- 7.2.5.4 Click the link for the LED & Push Button data. The new page displays the status of the LEDs on the DECA
Note that the Push Button portion is currently not implemented.



NOTE: Pushbutton 0 [Key 0] is connected to the system reset. Pushing this button will cause Nios to reset. If you accidentally reset the system, restart the application (Run → Run Configuration)



- [Status](#)
- [Accerometer, Temperature & Humidity](#)
- [LED & Push Button](#)

LED & Push Button

LED0	LED1	LED2	LED3	LED4	LED5	LED6	LED7
Turn On	Turn On	Turn Off	Turn Off	Turn On	Turn Off	Turn On	Turn On

Push Button Time:

Click a few LED buttons in the webpage and observe the corresponding LEDs illuminating on the DECA board.

Notice that the Nios II terminal displays an [HTTP EVENT] entry for each action you make in the webpage.

```
dc
[HTTP EVENT] event
dd
[HTTP EVENT] event
de
[HTTP EVENT] event
df
[HTTP EVENT] event
e0
[HTTP EVENT] event
e1
[HTTP EVENT] event
```

Congratulations! You have completed the WiFi portion of this lab!

7.2.6 Interact with the DECA board over Bluetooth® LE

The CC2650 fully implements the Bluetooth® Low Energy stack and is configured to appear as a variable device to other Bluetooth capable devices. In this section, you will pair your smartphone with the DECA and interact with the sensors over the Bluetooth link.



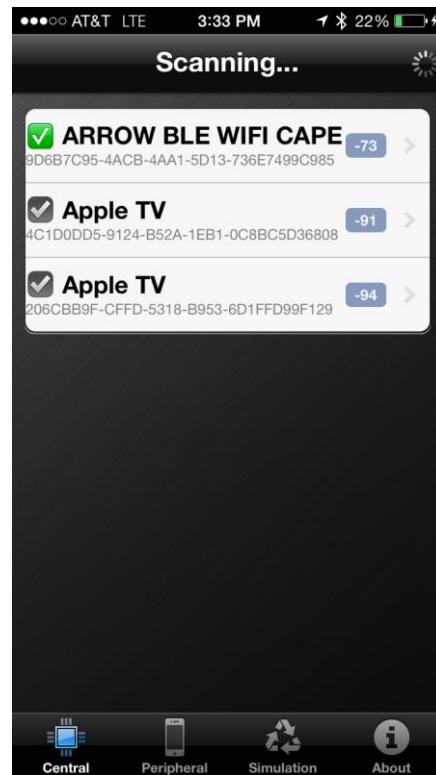
If you have an older smartphone, your device may not support Bluetooth Low Energy. Please ask your workshop trainer for help. For iOS users, continue on to the next step. For Android users, please skip to section 7.2.6.7.

7.2.6.1 Open the Bluetooth settings on your iOS device and ensure that the device's Bluetooth function is ON.

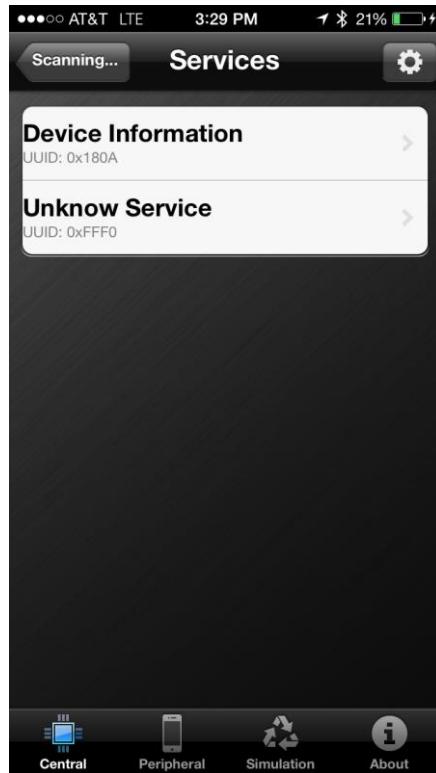
7.2.6.2 Open the App store and search for **BLE Utility**. Install the app below on your device.



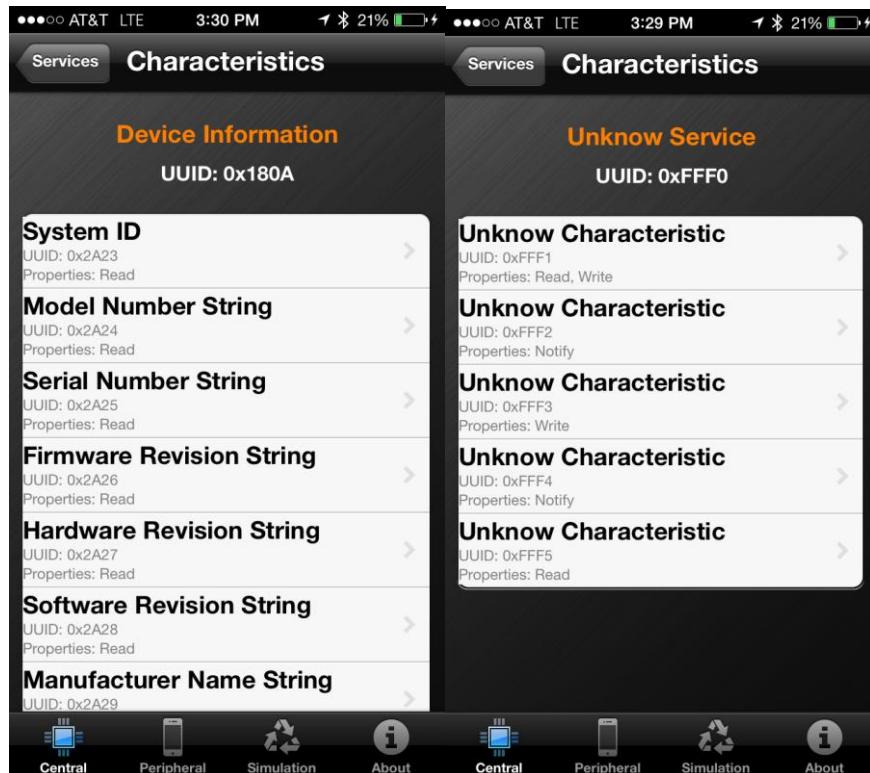
7.2.6.3 After the installation is complete, open the app. After the application opens and scans, you should see at least one entry for ARROW BLE WIFI CAPE. Note the negative number in the blue box. This is the device's Received Signal Strength Indicator (RSSI) number. It measures the strength of the Bluetooth device's signal in dB.



7.2.6.4 If there are multiple entries, move your iOS device very close to your DECA board. You should observe the RSSI value of one of the entries approach about -30. Select this device to pair with it.

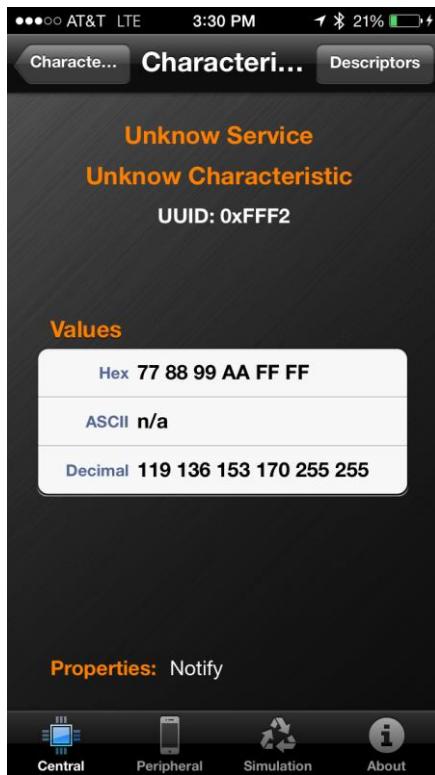


The CC2650 has been configured to provide two data "services". One provides Device Information such as system ID, hardware revision, and serial number; each of these fields is called a "characteristic". The other, titled "Unknown Service", allows the BLE Utility app to read temperature, humidity, and accelerometer data from the DECA board.



Feel free to explore the Device Characteristics service to see what kind of information can be learned.

7.2.6.5 From the "Unknown Service" select the Unknown Characteristic with UUID of 0xFFFF2. This characteristic displays the raw temperature and humidity data from the DECA board.

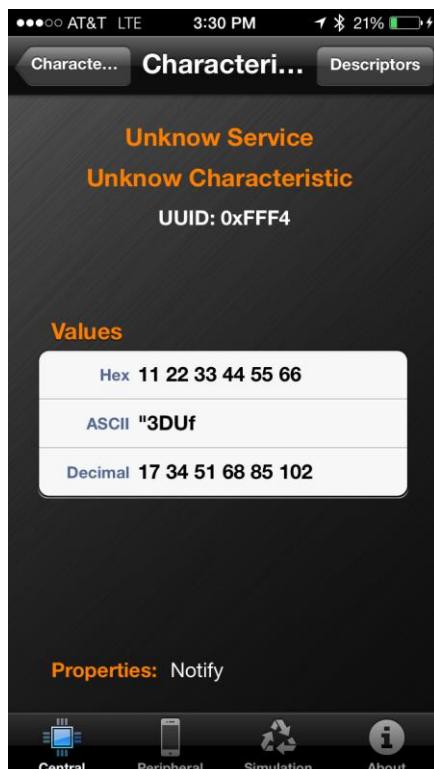


You should see the hexadecimal value fluctuating every few seconds. The format of the data is as follows:

temperature data (2 bytes), Humidity data (2 bytes), 0x0000

Wrap your hands around the DECA board and blow hot air on the board and observe how these values change.

- 7.2.6.6 Go back to the "Unknown Service" and select the Unknown Characteristic with UUID of 0xFFFF4. This characteristic displays the raw accelerometer data from the DECA board.



Again, you should see the hexadecimal value fluctuating. The format for the accelerometer data is as follows:

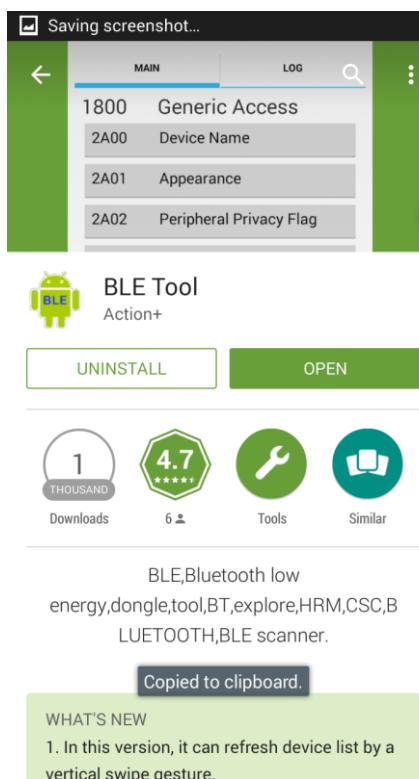
x-data (2 bytes), y-data (2 bytes), z-data (two bytes)

Tilt the DECA around and watch the hex value change. See if you can determine the nominal orientation of the accelerometer!

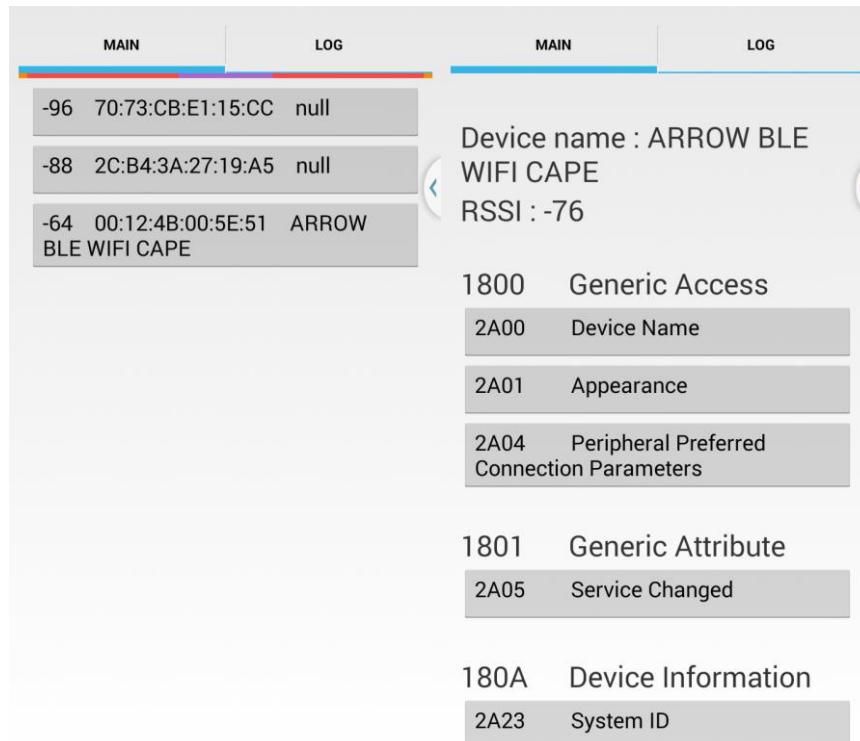
Congratulations! You have completed the BLE portion of this lab!

7.2.6.7 Open the Bluetooth settings on your Android device and ensure that the device's Bluetooth function is ON.

7.2.6.8 Open the Play Store and search for **BLE Tool**. Install the app below.

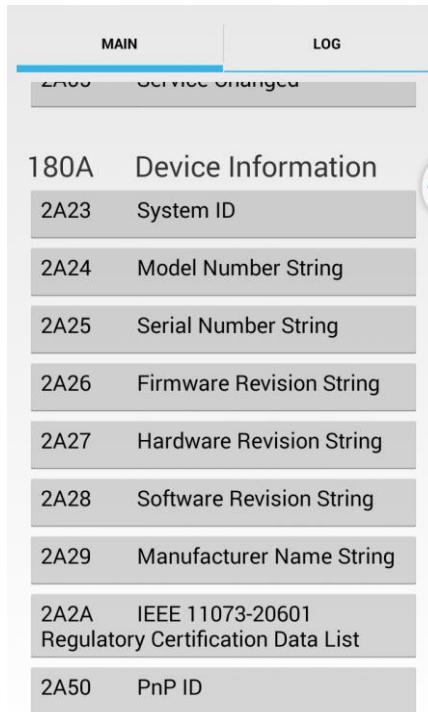


7.2.6.9 After the installation is complete, open the app. After the application opens and scans, you should see at least one entry for ARROW BLE WIFI CAPE. Note the negative number on the left side of each entry. This is the device's Received Signal Strength Indicator (RSSI) number. It measures the strength of the Bluetooth device's signal in dB.

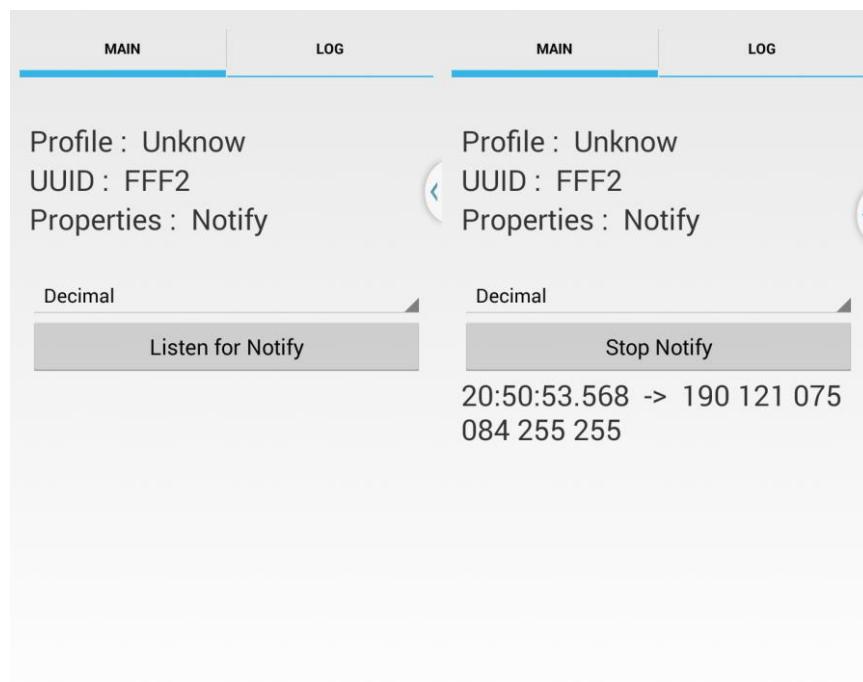


7.2.6.10 If there are multiple entries, move your Android device very close to your DECA board. You should observe the RSSI value of one of the entries approach about -30. Select this device to pair with it.

7.2.6.11 The CC2650 has been configured to provide two data "services". One provides Device Information such as system ID, hardware revision, and serial number; each of these fields is called a "characteristic". Feel free to open some characteristics and browse the information available. The other, titled "Unknown Service", allows the BLE Utility app to read temperature, humidity, and accelerometer data from the DECA board.



7.2.6.12 From the "Unknown Service" section select the Unknown Characteristic with UUID of 0xFFFF2. Click the Listen for Notify button. This characteristic displays the raw temperature and humidity data from the DECA board.

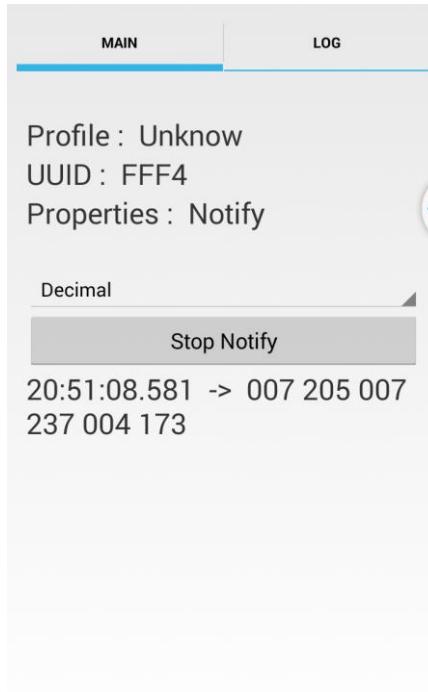


You should see the hexadecimal value fluctuating every few seconds. The format of the data is as follows:

Timestamp → temperature data (2 bytes), Humidity data (2 bytes), 0xFFFF

Wrap your hands around the DECA board and blow hot air on the board and observe how these values change.

7.2.6.13 Go back to the "Unknown Service" and select the Unknown Characteristic with UUID of 0xFFFF4. This characteristic displays the raw accelerometer data from the DECA board.



Again, you should see the hexadecimal value fluctuating. The format for the accelerometer data is as follows:

Timestamp -> x-data (2 bytes), y-data (2 bytes), z-data (two bytes)

Tilt the DECA around and watch the hex value change. See if you can determine the nominal orientation of the accelerometer!

CONGRATULATIONS! YOU HAVE COMPLETED THE BLE/WIFI CONNECTIVITY LAB!

MIPI to HDMI Lab

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 8. MIPI TO HDMI LAB	294
8.1 Set up the Quartus II project	294
8.1.1 Open the project file	294
8.1.2 Licensing the MIPI IP cores	294
8.2 Build the design.....	294
8.3 Open an existing Qsys file	295
8.4 Add some additional VIP blocks for Picture-in-Picture	296
8.4.1 Add the Mixer II IP	296
8.4.2 Add the TPG (Test Pattern Generator)	299
8.4.3 Modify the hdmi_cvo	299
8.4.4 Update the PLL settings.....	301
8.4.5 Connect the system	301
8.4.6 Resolve errors.....	302
8.4.7 Generate your Qsys system	303
8.4.8 Compile the design	303
8.4.9 Configure the FPGA.....	304
8.4.10 Verify your output.....	304

LAB 8. MIPI TO HDMI LAB

Overview: The goal of this workshop is to make use of the 8M-pixel camera and interact with Altera's Video and Image Processing suite to build, run and test an image video design.

8.1 Set up the Quartus II project

8.1.1 Open the project file

Browse to `C:\DECA\workshop_labs\8_MIPI_to_HDMI_Terasic` and open `mipi_to_hdmi_terasic.qpf`.

8.1.2 Licensing the MIPI IP cores

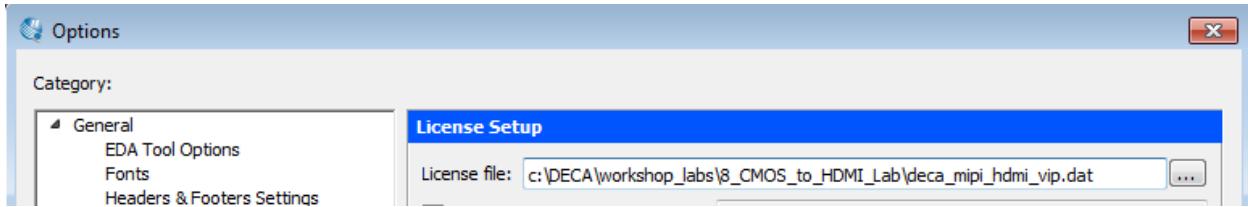
You will require a license in order to compile the FPGA design for this lab. The license is provided with the lab files. Here are the steps required to configure the license in Quartus

8.1.2.1 In the Quartus II menu, select **Tools → License Setup**

8.1.2.2 In the License Setup section browse to your license located in:

`c:\DECA\workshop_labs\8_CMOS_to_HDMI_Lab\deca_mipi_hdmi_vip.dat`

8.1.2.3 Your screen should now look like this:



Note that if you already have a license file in this field, you can append the new license file using a semi-colon (;), such as:

`c:\existing_license.dat;c:\DECA\workshop_labs\... \ deca_mipi_hdmi_vip.dat`

8.1.2.4 There are several different ways to 'refresh' the License Setup dialog box. Generally, select OK, to close the dialog box, then re-open it via: **Tools → License Setup**

8.1.2.5 Review the Licensed AMPP/MegaCore functions section and scroll to the bottom of the list to confirm the IP license was applied.

Your Quartus II project is set up. You are ready to start building your system

8.2 Build the design

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

8.3 Open an existing Qsys file

8.3.1.1 Launch Qsys from either the toolbar or the menu: **Tools → Qsys**

8.3.1.2 Choose **mipi_vip.qsys** and select Open.

Here you will find various components in this Qsys design. This design runs as-is. The default configuration looks like this functional diagram

Default Video Pipeline

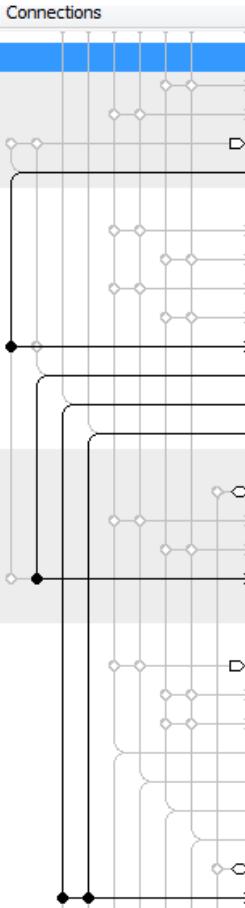


The blocks here are connected with a streaming protocol which is ideal for bursty applications including video and image processing. Here you will see that data is sent from a source (SRC) to a sink (SNK). The sink has the ability to apply backpressure in the event the IP is not ready to consume the data.

As shown in the diagram, the data is brought into the system, fed through a frame buffer, and driven out of the system to an HDMI transmitter. A brief description of these blocks follows:

- **terasic_bayer2rgb** – a piece of encrypted IP from Terasic that translates the CMOS camera input into a format the rest of Altera's Video and Image Processing (VIP) blocks can utilize.
- **frame_buffer** – an IP block from Altera's VIP library that writes the input data to DDR3 and reads out the output data using a ping-pong buffer approach (write to one page, read from another page, then swap)
- **hdmi_cvo** – another VIP block that takes the streaming data it receives and outputs it in a format that the HDMI transmitter can understand (parallel video, data enable, hsync, and vsync)

8.3.1.3 Review the connection diagram in Qsys and verify that it matches the functional view shown above:

Use	Connections	Name	Description	Export	Clock
		terasic_bayer2rgb	terasic_bayer2rgb		
		reset	Reset Input	<i>Double-click to export</i>	[clock]
		clock	Clock Input	<i>Double-click to export</i>	altpll_sys_c1
		din	Avalon Streaming Sink	terasic_bayer2rgb_din	[clock]
		dout	Avalon Streaming Source	<i>Double-click to export</i>	[clock]
		frame_buffer	Frame Buffer II (4K Ready)		
		main_clock	Clock Input	<i>Double-click to export</i>	altpll_sys_c1
		main_reset	Reset Input	<i>Double-click to export</i>	[main_clock]
		mem_clock	Clock Input	<i>Double-click to export</i>	altpll_sys_c1
		mem_reset	Reset Input	<i>Double-click to export</i>	[mem_clock]
		din	Avalon Streaming Sink	<i>Double-click to export</i>	[main_clock]
		dout	Avalon Streaming Source	<i>Double-click to export</i>	[main_clock]
		mem_master_wr	Avalon Memory Mapped Master	<i>Double-click to export</i>	[mem_clock]
		mem_master_rd	Avalon Memory Mapped Master	<i>Double-click to export</i>	[mem_clock]
		hdmi_cvo	Clocked Video Output II (4K Ready)		
		clocked_video	Conduit		
		main_clock	Clock Input	<i>Double-click to export</i>	altpll_sys_c1
		main_reset	Reset Input	<i>Double-click to export</i>	[main_clock]
		din	Avalon Streaming Sink	<i>Double-click to export</i>	[main_clock]
		status_update_irq	Interrupt Sender	<i>Double-click to export</i>	[main_clock]
		mem_if_ddr3_emif	DDR3 SDRAM Controller with UniPHY		
		pll_ref_clk	Clock Input	<i>Double-click to export</i>	
		global_reset	Reset Input	<i>Double-click to export</i>	
		soft_reset	Reset Input	<i>Double-click to export</i>	
		afi_clk	Clock Output	<i>Double-click to export</i>	
		afi_half_clk	Clock Output	<i>Double-click to export</i>	
		afi_reset	Reset Output	<i>Double-click to export</i>	
		afi_reset_export	Reset Output	<i>Double-click to export</i>	
		memory	Conduit	<i>Double-click to export</i>	
		avl	Avalon Memory Mapped Slave	<i>Double-click to export</i>	mem_if_ddr3_emif_afi_clk

8.4 Add some additional VIP blocks for Picture-in-Picture

These next steps modify Qsys system shown above to support picture-in-picture using a test pattern generator and a mixer. If you are not interested in this piece of the lab, you can proceed to the next section where you generate the Qsys system

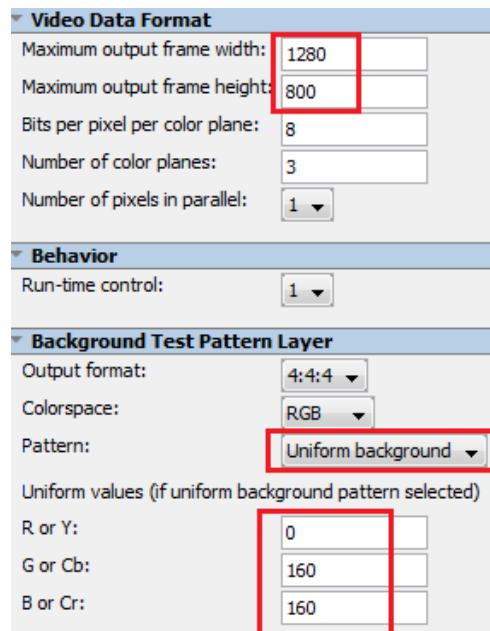
8.4.1 Add the Mixer II IP

The Mixer II IP is used to receive up to four inputs and overlay them to drive out to one display. We will use the mixer to center the camera video while the test pattern will be overlaid on top of the video. The mixer has the ability to move the images around on the screen using a register interface

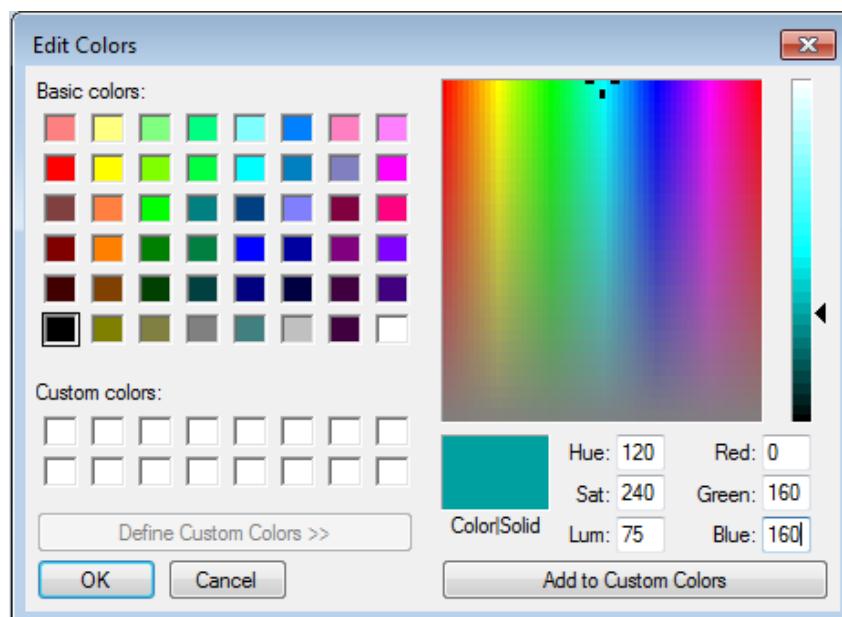
8.4.1.1 Add the Mixer II component in Qsys by typing **mixer** in the IP Catalog window

8.4.1.2 Double click the component titled: **Mixer II (4K Ready)**

8.4.1.3 Change the Video Data Format to match the following:



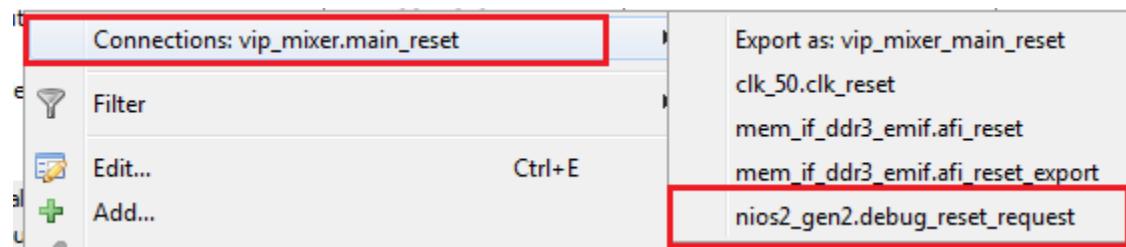
Note: the uniform background was arbitrarily chosen. Feel free to choose any other RGB color value. You can use Microsoft's Paint program, Edit Colors option to help choose a color.



- 8.4.1.4 Click Finish to close the Mixer II parameter editor.
- 8.4.1.5 Rename the component to **vip_mixer** (Right-click the component and select Rename)
- 8.4.1.6 Connect the clock to our video pipeline clock: **altpll_sys_c1**. You can do this by using the drop-down menu in the Clock column

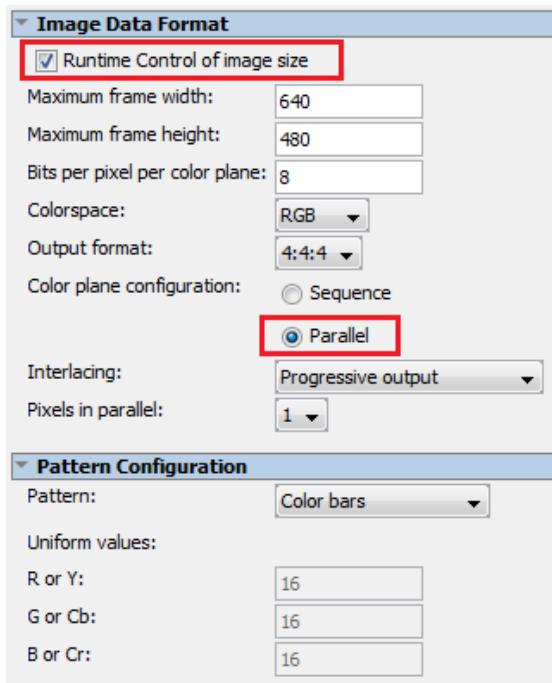
Name	Description	Export	Clock
external_connection	Conduit	ddr3_status_external_c...	
vip_mixer	Mixer II (4K Ready)		
main_clock	Clock Input	Double-click to export	altpll_sys_c1
main_reset	Reset Input	Double-click to export	altpll_mipi_c0
din0	Avalon Streaming Sink	Double-click to export	altpll_sys_c0
din1	Avalon Streaming Sink	Double-click to export	altpll_sys_c1
din2	Avalon Streaming Sink	Double-click to export	altpll_sys_c2
din3	Avalon Streaming Sink	Double-click to export	clk_50

- 8.4.1.7 Connect the reset to **nios2_gen2.debug_reset_request**. You can do this by right-clicking the reset signal then selecting Connections: **vip_mixer.main_reset** → **nios2_gen2.debug_reset_request**



8.4.2 Add the TPG (Test Pattern Generator)

- 8.4.2.1 Add the Test Pattern Generator II (4K Ready) component in Qsys by typing **tpg** in the IP Catalog window
- 8.4.2.2 Double click the component titled: **Test Pattern Generator II (4K Ready)**
- 8.4.2.3 Enable the Runtime Control of image size and switch the Color plane configuration to Parallel
- 8.4.2.4 The IP should be configured as:



- 8.4.2.5 Click Finish to add the TPG to your system
- 8.4.2.6 Rename the component to: **tpg**
- 8.4.2.7 Connect the clock and reset to match the **vip_mixer** component

8.4.3 Modify the **hdmi_cvo**

The Clocked Video Output resolution needs to be updated to support 1280 x 800 (the same resolution that the **vip_mixer** is using. The sync parameters change based on this new resolution. There are many useful references for determining the sync pulse widths. One of these resources can be found here:

<http://tinyvga.com/vga-timing/1280x800@60Hz>

8.4.3.1 Double-click the hdmi_cvo (or right-click and choose Edit) then reconfigure the IP to these parameters

Image Data Format

Image width / Active pixels:	1280	pixels
Image height / Active lines:	800	lines
Bits per pixel per color plane:	8	bits
Number of color planes:	3	
Color plane transmission format:	<input type="radio"/> Sequence <input checked="" type="radio"/> Parallel	
<input type="checkbox"/> Allow output of channels in sequence		
Number of pixels in Parallel:	1	<input type="button" value="▼"/>
<input type="checkbox"/> Interlaced video		

Syncs Configuration

Syncs signals:	<input type="radio"/> Embedded in video <input checked="" type="radio"/> On separate wires
Active picture line:	0

Frame / Field 1 Parameters

Ancillary packet insertion line:	0
----------------------------------	---

Embedded Syncs Only - Frame / Field 1

Separate Syncs Only - Frame / Field 1

Horizontal sync:	136	pixels
Horizontal front porch:	64	pixels
Horizontal back porch:	200	pixels
Vertical sync:	3	lines
Vertical front porch:	1	lines
Vertical back porch:	24	lines

Interlaced and Field 0 Parameters

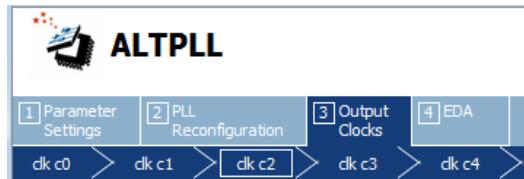
General Parameters

Pixel fifo size:	1280	pixels
Fifo level at which to start output:	1279	pixels
<input type="checkbox"/> Video in and out use the same clock		
<input type="checkbox"/> Use control port		
<input type="checkbox"/> Accept synchronization outputs		
Runtime configurable video modes:	1	modes
Width of vid_std bus:	1	bits

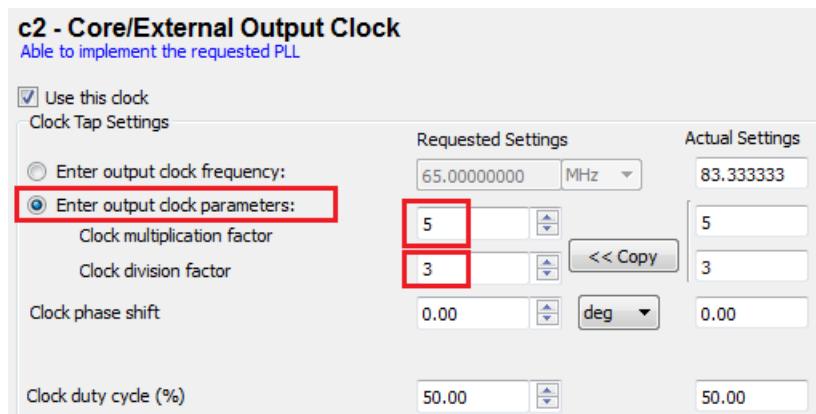
8.4.4 Update the PLL settings

When the output resolution is increased, the pixel clock also has to be increased. The pixel clock for 1280 x 800 is 83.46 MHz. This frequency cannot be achieved with our current PLL settings (due to the other clocks in the system) however we can get very close (83.33 MHz).

- 8.4.4.1 Find the altpll_sys component and double click it. A right-hand pane opens.
- 8.4.4.2 Click the Edit Parameters... button to edit the PLL settings
- 8.4.4.3 Using the tabs along the top of the PLL MegaWizard, jump to section **[3] Output Clocks**, then the sub-section **clk c2**



- 8.4.4.4 Update the parameters shown here to obtain a frequency of 83.3333 MHz



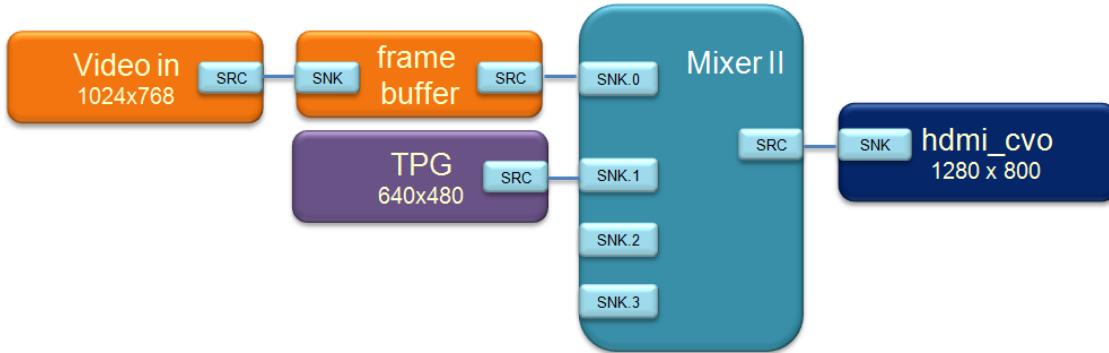
- 8.4.4.5 Click Finish twice to close the PLL MegaWizard.

- 8.4.4.6 You can also close the PLL Parameter editor window

8.4.5 Connect the system

Now we will remap our video pipeline to match something like this:

Modified Video Pipeline



- 8.4.5.1 Connect these following interfaces. There are several different methods including the connection matrix on the left of Qsys, or using the right-click method outlined above:

<code>vip_mixer.dout</code>	<code>hdmi_cvo.din</code>
<code>vip_mixer.control</code>	<code>nios2_gen2.data_master</code>
<code>tpg.control</code>	<code>nios2_gen2.data_master</code>
<code>tpg.dout</code>	<code>vip_mixer.din1</code>
<code>frame_buffer.dout</code>	<code>vip_mixer.din0</code>

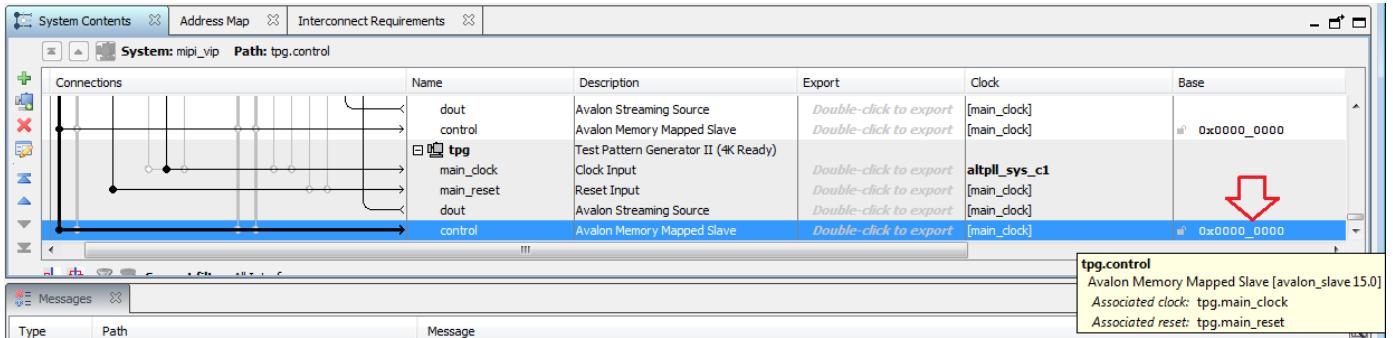
8.4.6 Resolve errors

If the previous connections were done correctly, you will likely have several error messages:

Type	Path	Message
✖ 4 Errors		
✖	<code>mipi_vip.nios2_gen2.data_master</code>	<code>tpg.control (0x0..0x1f) overlaps ddr3_status.s1 (0x0..0xf)</code>
✖	<code>mipi_vip.nios2_gen2.data_master</code>	<code>vip_mixer.control (0x0..0xff) overlaps tpg.control (0x0..0x1f)</code>
✖	<code>mipi_vip.nios2_gen2.instruction_master</code>	<code>tpg.control (0x0..0x1f) overlaps ddr3_status.s1 (0x0..0xf)</code>
✖	<code>mipi_vip.nios2_gen2.instruction_master</code>	<code>vip_mixer.control (0x0..0xff) overlaps tpg.control (0x0..0x1f)</code>
⚠ 14 Warnings		

These messages are all related. Qsys is throwing an error because the Avalon bus masters have overlapping peripherals that share memory regions. Naturally, this causes an addressing error.

- 8.4.6.1 To resolve this error, set the base address of the **tpg** to: **0x100**. You can edit the address by double-clicking the base address of the tpg's control port as shown by the arrow:



- 8.4.6.2 Verify there are no other red error messages.

- 8.4.6.3 You will find several other warning messages in this Qsys design mainly because unnecessary ports are not brought out of the Qsys system. You can safely ignore these warnings.

8.4.7 Generate your Qsys system

- 8.4.7.1 In the bottom right-hand corner of Qsys, select Generate HDL... then click Generate in the dialog box that opens.

Qsys will begin generating the design files necessary to implement the system that is described within Qsys. This process will take several minutes to complete.

- 8.4.7.2 After generation completes, click Close to leave the Qsys generation dialog box

- 8.4.7.3 Switch back to Quartus leaving Qsys open.

8.4.8 Compile the design

- 8.4.8.1 In the Quartus II menu, select: **Processing → Start Compilation**

This compile takes roughly 10 minutes to complete

8.4.9 Configure the FPGA

8.4.9.1 Prior to powering on the board:

- Connect the MIPI camera to your DECA board
- Set SW0 and SW1 both to their default position of “00”

8.4.9.2 Now you are ready to connect the USB cable from **UB2 J10** on the DECA board to your computer.

8.4.9.3 Open the Quartus II Programmer via: **Tools → Programmer**

A chain description file opens with the programming file already specified.

8.4.9.4 Press Start to begin configuration

8.4.10 Verify your output

8.4.10.1 After downloading the design to your DECA board, you will find that the majority of the LED’s are on.

8.4.10.2 Connect your DECA board to an HDMI monitor or projector

By default, the mixer’s outputs are disabled so we are expecting a blank screen at this time. Generally, software is written to update the registers on both the Mixer II and the TPG. In our case, we are going to use System Console to update these registers

8.4.10.3 To launch System Console, return back to Qsys

8.4.10.4 In the Qsys menu, select: **Tools → System Console**

8.4.10.5 Once System Console loads, you can launch a prewritten TCL script to enable the mixer and TPG.

8.4.10.6 Type **source vippmix.tcl** at the Tcl Console prompt (located in the lower right-hand window)

8.4.10.7 You should see two images appear. The larger camera input and a smaller TPG. You can move either of these images around by adjusting the x & y coordinates. You can also adjust the size of the TPG. More information can be found in the vippmix.tcl file provided.

Here are some useful commands you can use to experiment with:

8.4.10.8 Move the TPG to coordinates: 100, 100: **master_write_32 \$nm \$frm1x 100 100**

8.4.10.9 Move the TPG to coordinates: 0, 0: **master_write_32 \$nm \$frm1x 0 0**

8.4.10.10 Resize the TPG to 640x480: **master_write_32 \$nm \$tpg 1 0 0 640 480**

8.4.10.11 Turn off the TPG if you no longer want to use it: **master_write_32 \$nm \$frm1 0**

Note moving the TPG beyond the limit of 1280 x 800 will cause the mixer to crash. This can be accidentally done if you increase the size of the TPG too large or moving the starting corner too far over or down.

CONGRATULATIONS! YOU HAVE COMPLETED THE MIPI TO HDMI LAB

Appendix

For the MAX® 10 DECA FPGA Evaluation Kit

Version 15.0

6/07/2015

TABLE OF CONTENTS

LAB 10. APPENDIX.....	308
10.1.1 Getting your DECA Kit	308
10.1.2 Installing the Altera Complete Design Software v15.0.....	308
10.1.3 Configure the Quartus the programmer.....	312

LAB 9. APPENDIX

9.1 Getting your DECA Kit

If you are attending a DECA Workshop, you should have received your DECA kit in the 3-in-1 evaluation kit bundle when you arrived at the workshop location. If you are working on this lab independently, a DECA kit can be purchased through your Arrow sales representative or at parts.arrow.com.



Make sure you have a USB cable to connect the on-board USB Blaster to your laptop. If you are attending the workshop, a USB cable should be included in your evaluation kit bundle.

9.2 Installing the Altera Complete Design Software v15.0

You will need to have Quartus II 15.0 installed on the computer with which you wish to complete the lab prior to the workshop date as there is not enough time to download and install the software during the workshop.

If you are completing this lab independently, the Quartus II 15.0 Web Version software can be installed for free by following the steps below.

- 9.2.1.1 Go to the Altera Download Center [here](#). You will need to log in to a myAltera account to download the software. You can log into an existing account or create a new account.

 **myAltera Sign In**

[? myAltera Account Help](#)
 [Terms and Conditions](#)



User Name

Password

[Forgot Your User Name or Password?](#)

Remember me

Don't have an account?

[Create Your myAltera Account](#)
Your myAltera account allows you to file a service request, register for a class, download software, and more.

Enter your email address

Note: If your email address already exists in our system we will retrieve the associated information.

308

Max10 DECA Workshop Manual

- 9.2.1.2 Select 15.0 as the release and make sure you have selected the two files shown below. The Quartus II Software and MAX10 FPGA device support are required for this lab. You can download all of the tools and device support packs if you wish but it will take longer.

Select All

Quartus II Web Edition (Free)

Quartus II Software (includes Nios II EDS)
Size: 1.2 GB MD5: FB732633ECB57BE1A73BE45D172918AF  UPDATE

ModelSim-Altera Edition (includes Starter Edition)
Size: 1.1 GB MD5: C931A4F7F9B4306DD8E8248607993C7C

Devices

You must install device support for at least one device family to use the Quartus II software.

Arria II device support
Size: 497.7 MB MD5: B329C8FCC2E1315BOE36C11AD41A23F7

Cyclone IV device support
Size: 462.7 MB MD5: 599819EBE4DDBFA0B622505B22432E86

Cyclone V device support
Size: 1.0 GB MD5: 446D7EE5999226CD3294F890A12C53CC

MAX II, MAX V device support
Size: 11.3 MB MD5: C3EDC556AC9770DB2DD63706EECA2654

MAX 10 FPGA device support
Size: 289.0 MB MD5: 75F2D4AF1E847FC53AC6B619A35BD2CF

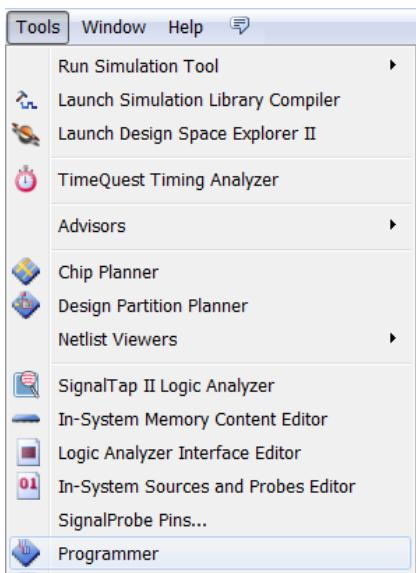
Download Selected Files

Note: The Quartus II software is a full-featured EDA product. Depending on your download speed, download times may be lengthy.

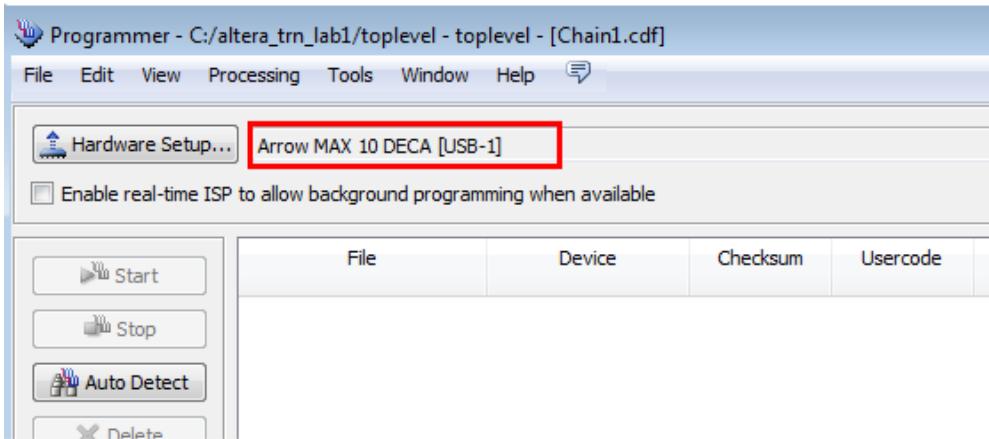
- 9.2.1.3 Click "Download Selected Files". The Akamai Download manger will open and report the progress of your download.
- 9.2.1.4 After the download is complete, run the installation executable. All of the defaults should be used for the installation.
- 9.2.1.5 The USB Driver installation should open automatically. Use the defaults to complete the installation.

9.3 Configure the Quartus the programmer

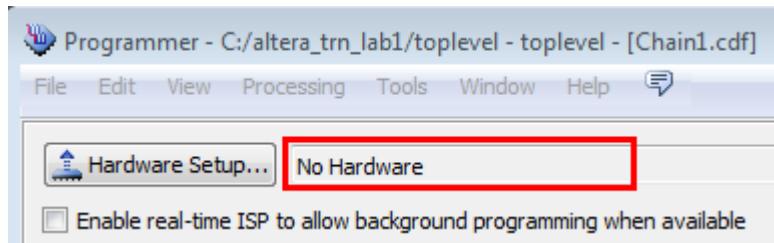
9.3.1.1 Open the Quartus programmer: Tools →Programmer



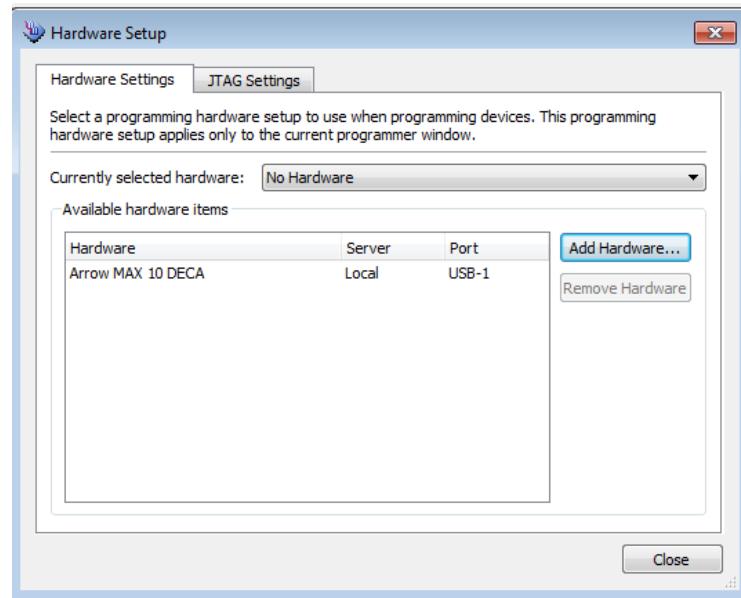
9.3.1.2 The Programmer window will appear. Verify the Hardware setup looks like this:



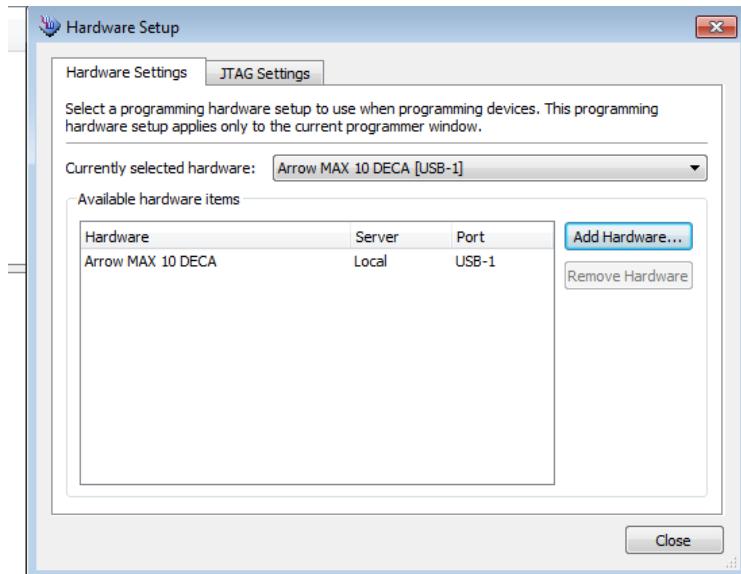
9.3.1.3 If it states No Hardware (as seen below), it is necessary to run the hardware setup.



9.3.1.4 Run the Hardware setup (if necessary) by clicking on the Hardware Setup button.



9.3.1.5 Use the pull-down menu to select Arrow Max 10 DECA board, and select Arrow Max 10 DECA and then select Close.



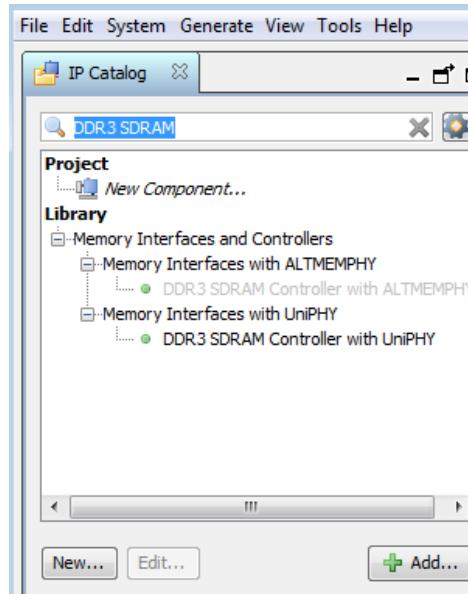
9.4 Adding DDR3 DRAM to your design

Here is a reference guide on adding DDR3 DRAM to a DECA design

9.4.1 DDR3 SDRAM Controller

This DDR3 SDRAM Controller IP, which will be inside the FPGA, connects the FPGA to the DDR3 SDRAM.

Within Qsys, add this component, type **DDR3 SDRAM** in the IP Catalog as seen below:

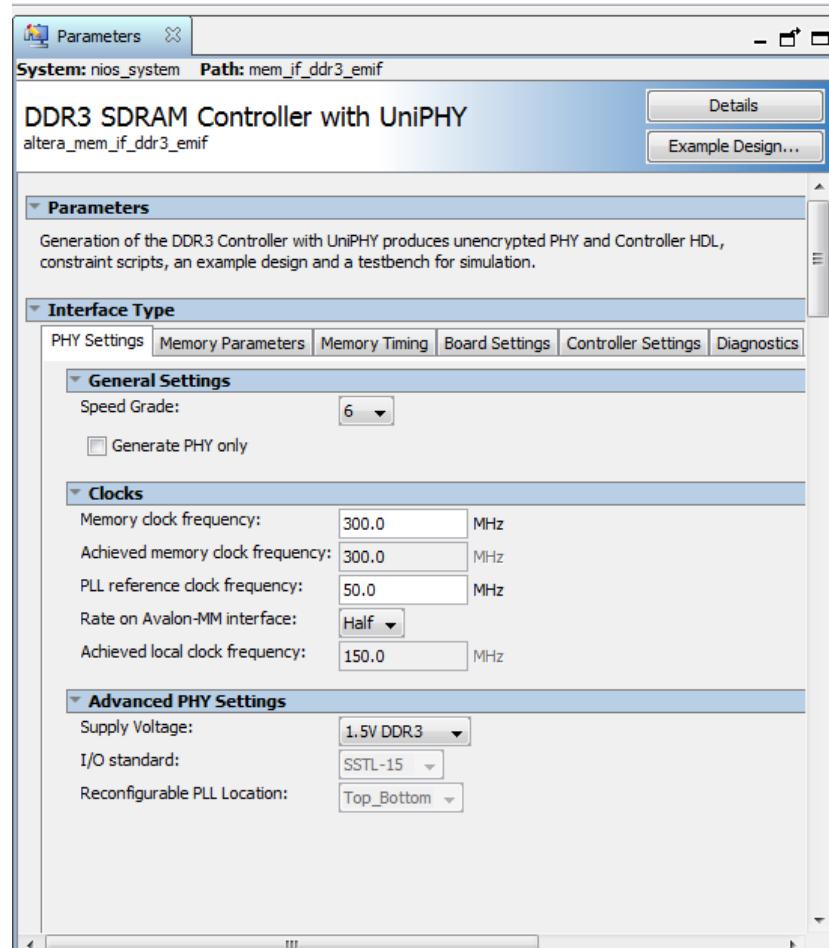


and Select Add.

There are various tabs for the DDR3 SDRAM Controller IP.

Note: Ensure the settings for the DDR3 are set as the following so that it can correspond to the DDR3 that is on the board.

9.4.1.1 Under the PHY Setting tab, the following settings should be set as:



The DDR3 settings correspond to the MAX 10 and DDR3 component on the PCB.

- The speed grade (6) corresponds to the speed grade of the FPGA device (MAX 10) that is on the DECA design.
- The DDR3 has a clock frequency of 300 MHz, while the PLL reference clock is set to 50 MHz.
- The supply voltage for the DDR3 (eg on the PCB) is 1.5 V DDR3.

9.4.1.2 Under the Memory Parameters tab, the following settings should be set as:

Interface Type

- PHY Settings
- Memory Parameters**
- Memory Timing
- Board Settings
- Controller Settings
- Diagnostics

Apply memory parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

Memory vendor: **Micron**

Memory format: **Discrete Device**

Memory device speed grade: **800.0 MHz**

Total interface width: **16**

DQ/DQS group size: **8**

Number of DQS groups: **2**

Number of chip selects: **1**

Number of clocks: **1**

Row address width: **15**

Column address width: **10**

Bank-address width: **3**

Enable DM pins

Memory Initialization Options

Mirror Addressing: 1 per chip select: **0**

Address and command parity

Mode Register 0

Read Burst Type: **Sequential**

DLL precharge power down: **DLL off**

Memory CAS latency setting: **7**

Mode Register 1

Output drive strength setting: **RZQ/7**

Memory additive CAS latency setting: **Disabled**

ODT Rtt nominal value: **RZQ/6**

Mode Register 2

Auto selfrefresh method: **Manual**

Selfrefresh temperature: **Normal**

Memory write CAS latency setting: **6**

Dynamic ODT (Rtt_WR) value: **RZQ/4**

The reason for these settings are to ensure that the settings correspond to the Micron, DDR3, that is on the PCB.

- 9.4.1.3 Under the Memory Timing settings, these are settings from the manufacturer (Micron in this case) data sheet.
- 9.4.1.4 Make changes to the timing parameters to correspond to the following timing parameters.

Interface Type	
	PHY Settings
	Memory Parameters
	Memory Timing
	Board Settings
Apply timing parameters from the manufacturer data sheet	
Apply device presets from the preset list on the right.	
tIS (base):	185 ps
tIH (base):	130 ps
tDS (base):	55 ps
tDH (base):	55 ps
tDQSQ:	125 ps
tQH:	0.38 cycles
tDQSCK:	225 ps
tDQSS:	0.25 cycles
tQSH:	0.4 cycles
tDSH:	0.2 cycles
tDSS:	0.2 cycles
tINIT:	500 us
tMRD:	4 cycles
tRAS:	35.0 ns
tRCD:	13.75 ns
tRP:	13.75 ns
tREFI:	7.8 us
tRFC:	260.0 ns
tWR:	15.0 ns
tWTR:	6 cycles
tFAW:	45.0 ns
tRRD:	7.5 ns
tRTP:	7.5 ns

- 9.4.1.5 Under the Board Settings tab, there are settings that used to model the board-level effects in timing analysis.

Make the following changes so that the parameters are correct. In particular, note the changes for the board skew that will need to be changed.

Interface Type

PHY Settings | Memory Parameters | Memory Timing | **Board Settings** | Controller Settings | Diagnostics

Use the Board Settings to model the board-level effects in the timing analysis.

The wizard supports single- and multi-rank configurations. Altera has determined the effects on the output signaling of these configurations and has stored the effects on the output slew rate and the channel uncertainty within the UniPHY MegaWizard.

These values are representative of specific Altera boards. You must change the values to account for the board level effects for your board. You can use HyperLynx or similar simulators to obtain values that are representative of your board.

Setup and Hold Derating

Channel Signal Integrity

Board Skews

PCB traces can have skews between them that can cause timing margins to be reduced. Furthermore skews between different ranks can further reduce the timing margin in multi-rank topologies.

Restore default values

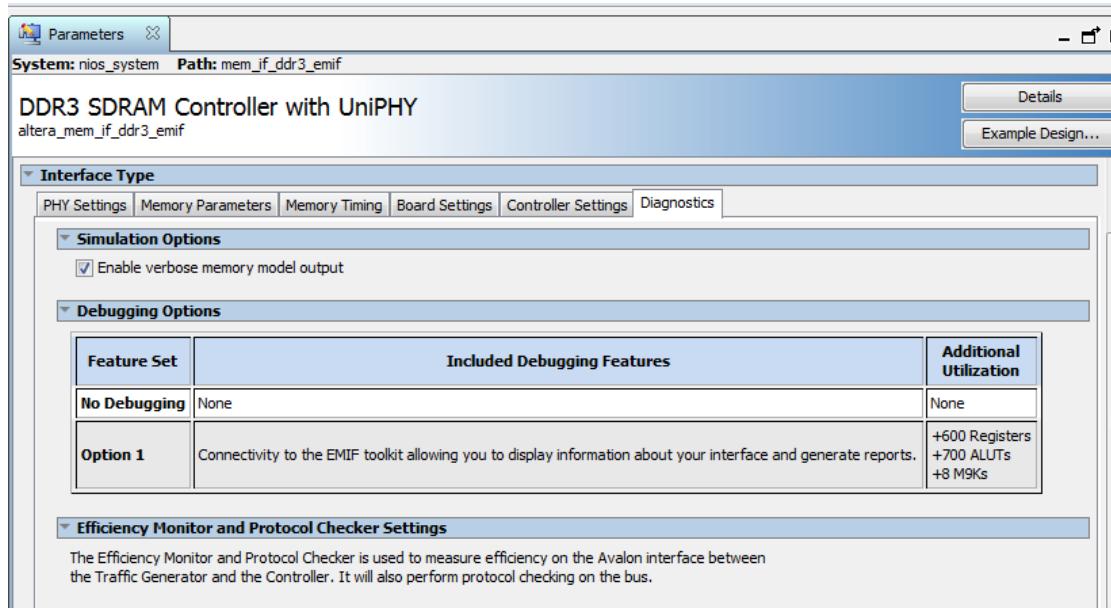
Maximum CK delay to DIMM/device:	0.183	ns
Maximum DQS delay to DIMM/device:	0.181	ns
Minimum delay difference between CK and DQS:	0.002	ns
Maximum delay difference between CK and DQS:	0.007	ns
Maximum skew within DQS group:	0.008	ns
Maximum skew between DQS groups:	0.005	ns
Average delay difference between DQ and DQS:	-6.0E-4	ns
Maximum skew within address and command bus:	0.017	ns
Average delay difference between address and command and CK:	-1.3E-4	ns

9.4.1.6 Under the Controller Setting , ensure the settings are set as follows:

Interface Type

- Avalon Interface**
 - Generate power-of-2 data bus widths for Qsys or SOPC Builder
 - Generate SOPC Builder compatible resets
 - Maximum Avalon-MM burst length:
 - Enable Avalon-MM byte-enable signal
 - Avalon interface address width: bits
 - Avalon interface data width: bits
- Low Power Mode**
 - Enable Self-Refresh Controls
 - Enable Auto Power-Down
 - Auto Power-Down Cycles: cycles
- Efficiency**
 - Enable User Auto-Refresh Controls
 - Enable Auto-Precharge Control
 - Local-to-Memory Address Mapping:
 - Command Queue Look-Ahead Depth:
 - Enable Reordering
 - Starvation limit for each command: commands
- Configuration, Status and Error Handling**
 - Enable Configuration and Status Register Interface
 - CSR port host interface:
 - Enable Error Detection and Correction Logic
 - Enable Auto Error Correction

9.4.1.7 Under Diagnostics, select the following:



Select Finish.

9.4.1.8 Rename the DDR3 II: Right click on the name, select Rename and name it as `mem_if_ddr3_emif`:

mem_if_ddr3_emif		DDR3 SDRAM Controller with UniPHY
pll_ref_clk		Clock Input
global_reset		Reset Input
soft_reset		Reset Input
afi_clk		Clock Output
afi_half_clk		Clock Output
afi_reset		Reset Output
afi_reset_export		Reset Output
memory		Conduit
avl		Avalon Memory Mapped Slave
status		Conduit
pll_sharing		Conduit

9.4.1.9 Export the signals

Since this is DDR3 IP, some of the pins are exported. These pins will be connected to the PCB. To do this export, select on the Export column and enter the following signal names.

- mem_if_ddr3_emif_pll_ref_clk
- mem_if_ddr3_emif_status
- mem_if_ddr3_emif_pll_sharing

<input type="checkbox"/>  mem_if_ddr3_emif	DDR3 SDRAM Controller with UniPHY	
pll_ref_clk	Clock Input	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
global_reset	Reset Input	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
soft_reset	Reset Input	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
afi_clk	Clock Output	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
afi_half_clk	Clock Output	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
afi_reset	Reset Output	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
afi_reset_export	Reset Output	mem_if_ddr3_emif_pll_ref_clk <i>Double-click to export</i>
memory	Conduit	memory <i>Double-click to export</i>
avl	Avalon Memory Mapped Slave	mem_if_ddr3_emif_status <i>Double-click to export</i>
status	Conduit	mem_if_ddr3_emif_sharing <i>Double-click to export</i>
pll_sharing	Conduit	