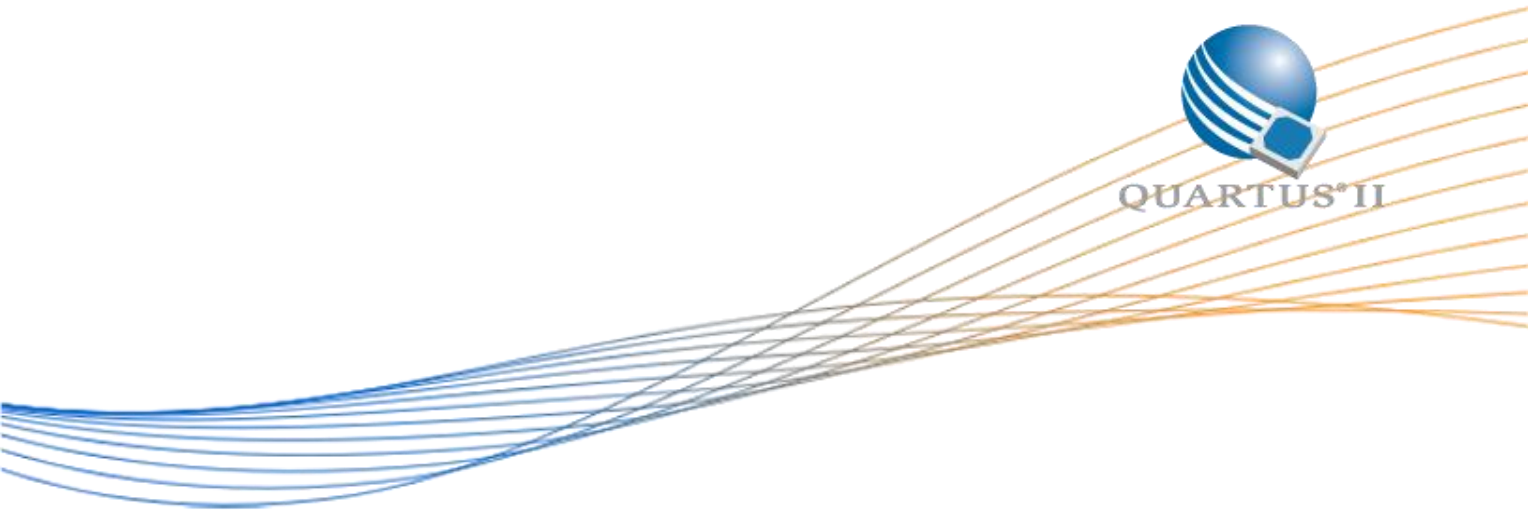

Cyclone V Nios II Embedded “Hello World” Lab:

Cyclone V E FPGA Development Kit

September 2015

Version 1.05

Contents

Overview

This lab teaches you how to create an embedded system implemented in programmable logic using Altera's "soft" Nios II processor. A soft processor is built from the programmable logic fabric and hence can be easily modified to suit applications requirements whereas a hard processor is built from "hard" standard cells that cannot be changed without redesigning the chip. Soft processors can be built in any of Altera's FPGA families, while hard processors can be found in Altera's "SoC FPGA" device families such as Cyclone V SoC or Arria V SoC.. The Nios II processor is supported by a rich set of peripherals and "IP" blocks built that can be configured and connected to the processor using Altera's Qsys tool within the Quartus II design tool set. Altera also distributes the Nios II Software Build Tools (SBT) for Eclipse (for software development) within the Quartus development suite.

This lab will show you how to install the Cyclone V E FPGA Development kit pin settings, design the processor-based hardware system, download it to the dev Kit, and run a simple "Hello World" software program which displays text on your terminal and uses buttons to control LEDs. The lab is split into a Hardware Section and Software Section. You can skip the hardware section and move directly to the software section should you choose.

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 is necessary for consistency and to ensure that each step works properly in the lab, when creating your own systems you can choose your own names as long as you use them consistently in your project. The directory paths shown in the figures are for Linux (forward slash directory delimiter). If you are using Windows, the paths will be shown with backslash directory delimiters.

Quartus Installation

Quartus is Altera's design tool suite. It serves a number of functions:

1. Design creation through the use of HDL languages or schematics
2. System creation through the Qsys graphical interface
3. Generation and editing of constraints: timing, pin locations, physical location on die, IO voltage levels
4. Synthesis of high level language into an FPGA netlist ("mapping" in FPGA terminology)
5. FPGA place and route ("fitting" in FPGA terminology)
6. Generation of design image (used to program FPGA, "assembly" in FPGA terminology)
7. Timing Analysis
8. Programming/download of design image into FPGA hardware
9. Debugging by insertion of debug logic (in-chip logic analyzer)
10. Interfaces to 3rd party tools such as simulators
11. Launching of Software Build Tools (Eclipse) for Nios II

To download Quartus, follow these instructions:
Visit this site: http://dl.altera.com/?edition=web to download the latest version of Quartus II.
Select version 15.0 and your PC's operating system.
For the smallest installation, and quickest download time, enter only the entries shown below.

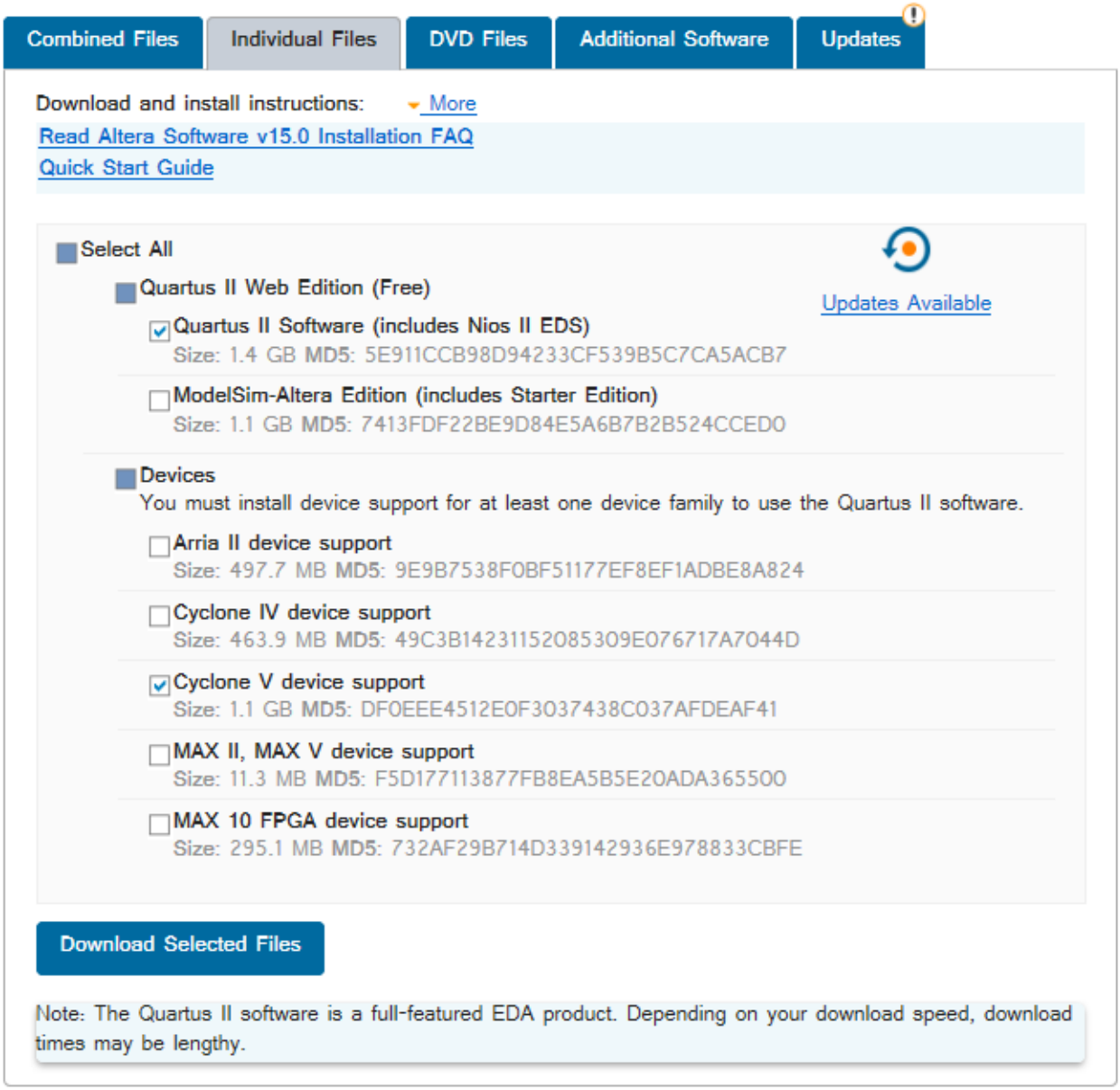

Figure 1: Quartus Download Page

Follow the download instructions provided from the web page. No license is required to run Cyclone V FPGAs on the Quartus software.

You can also download Update for your Quartus release. Navigate to the Updates Tab and download Update 2. Follow the instruction to install the update.

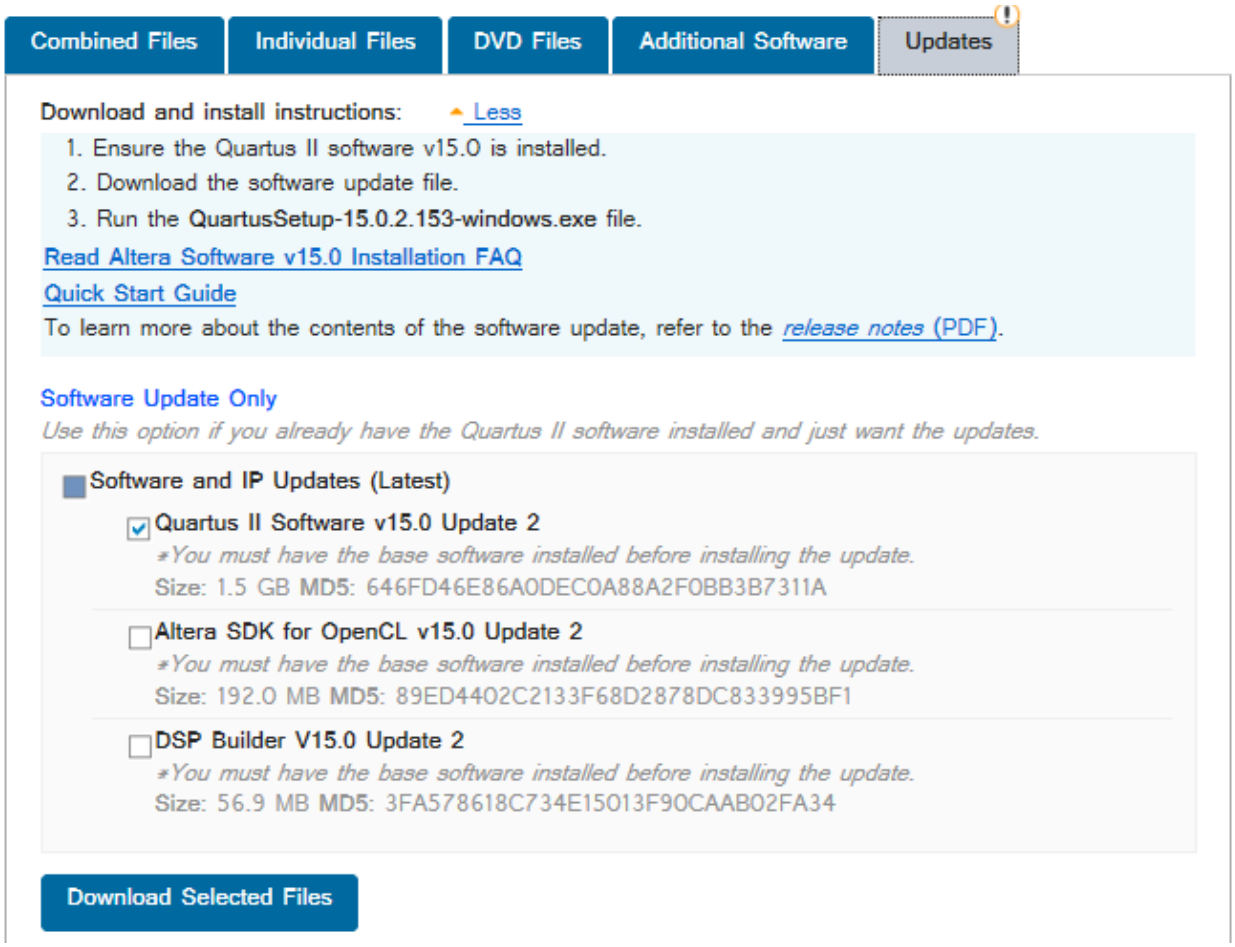

Figure 2: Quartus Update Tab

Design Flow

Unlike system development with hard processors, development with soft processors enables you to optimize the processor system to your application requirements and use the FPGA to add the performance and interfaces required by your system. This means that you need to know how to modify the processor system hardware; this may sound challenging but thanks to the Qsys graphical system design tool this is actually a relatively easy thing to do as we will demonstrate in this lab.

The Figure 1Figure 3 illustrates how an overall system is integrated using the combination of the Qsys system integration tool, Quartus for mapping (FPGA terminology for synthesis), fitting (FPGA terminology for place and route), and the NIOS Software Build Tool (SBT) for software development.

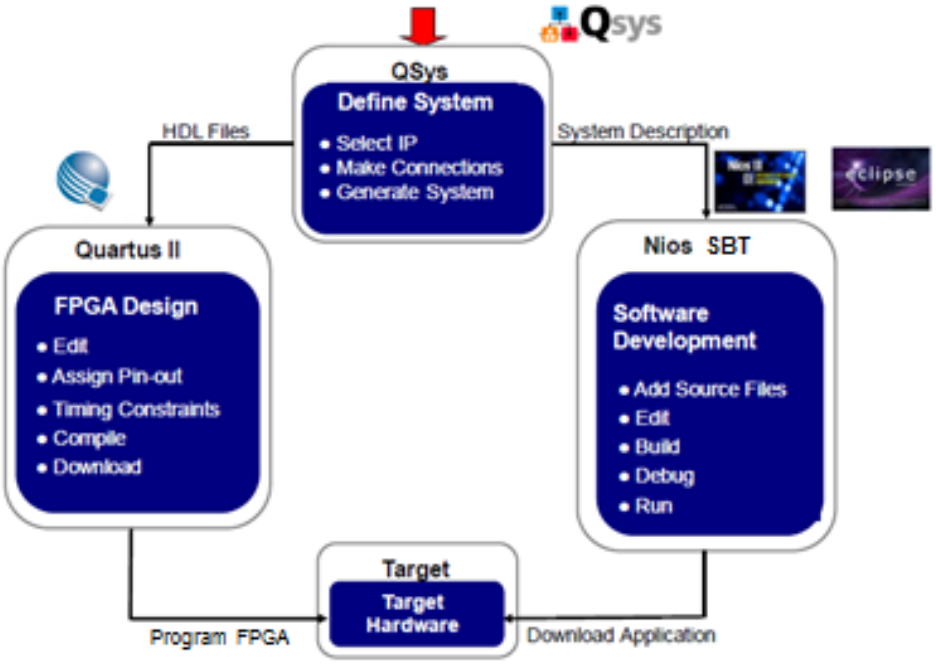

Figure 3: Development Flow

The above diagram depicts the typical flow for Nios II system design. Hardware System definition is performed using Qsys; the resultant HDL files from the Qsys system are used by the Quartus II FPGA design software to map, fit and download the hardware image into the FPGA device. Quartus II also generates information that describes the configuration of the system designed in Qsys so that the Nios II SBT can be configured to create a software library that matches the hardware system and contains all the correct peripheral drivers.

Introduction to the Cyclone V E FPGA Development Kit

Altera and its design partners create a number of development kits to allow users a quick and convenient starting point for designing with Cyclone devices. These kits include schematics and bill of materials should you want to use the Development Kit for production or make a derivative product based on the bill of materials. The Cyclone V E FPGA Development Kit offers a comprehensive general purpose development platform for many markets and applications, including Industrial Networking, Military, and Medical applications. The kit features an Altera Cyclone V device and a multitude of on-board resources including multiple banks of DDR3 and LPDDR2 memory, LCD character display, LEDs, user push buttons, USB, and RJ-45 connectors. In this lab, we will not use many of the interfaces, but there is a wealth of design examples and reference material demonstrating how to use these interfaces. You can find more information of this kit here.

Figure 4: Cyclone V E FPGA Development Kit Features

Objective of Hardware Design for the "Hello World" Lab

For the simplest example, a "hello world" lab, the processor will load a program that prints "Hello World" to the screen. This requires a working processor to execute the code, on-chip memory to store the software executable, and a JTAG UART peripheral to send the "Hello World" text to a terminal. To make the lab a little bit more interesting and hardware-centric, we will utilize the push buttons and LEDs to allow interaction with the development kit.

The lab hardware is constructed with the components shown below. Altera utilizes the Qsys network on chip interconnect to connect the master and slave devices together. To get a clear understanding of how quickly one can build an Embedded System using Qsys and the Quartus design software you will build the Nios II system entirely from scratch.

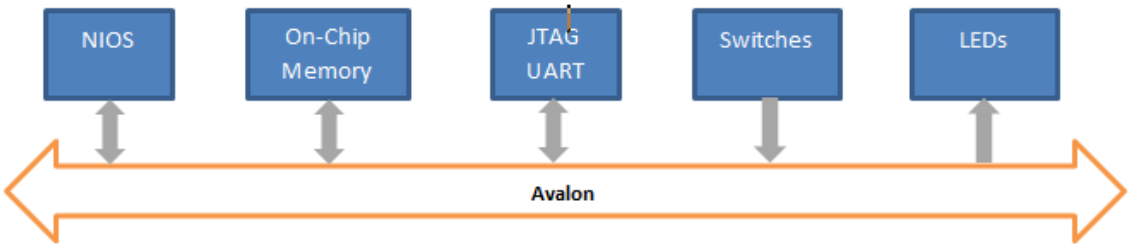

Figure 5: Nios II Based System Used in This Lab

HARDWARE DESIGN

Initial Setup

Should you want to skip the hardware design section, continue in the section called SOFTWARE DESIGN.

Altera provides a starting point design to get the FPGA device pinouts associated with the development kits layout and your design via what is called the Baseline design. Navigate to Altera's design store: https://cloud.altera.com/devstore

Click on Design Examples.

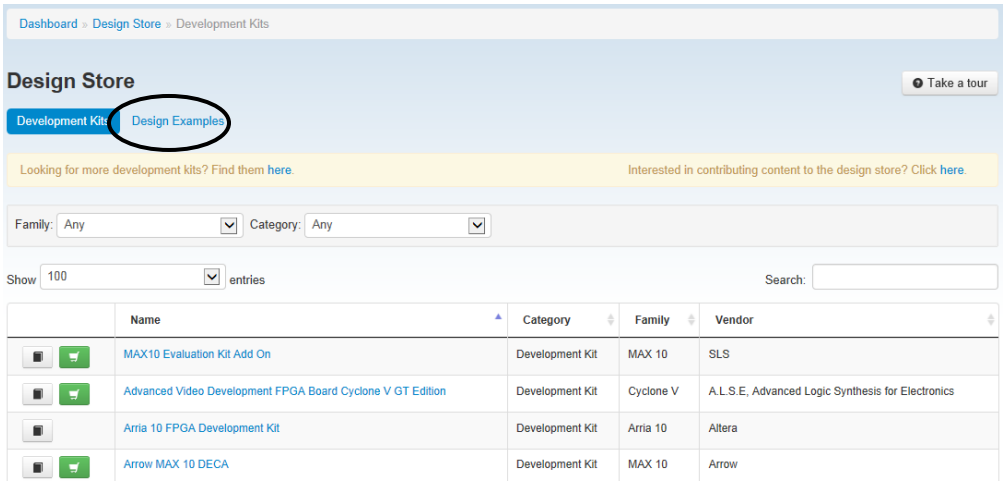

Figure 6: Design Store

Once in Design Examples, filter by the Cyclone V E FPGA Development Kit.

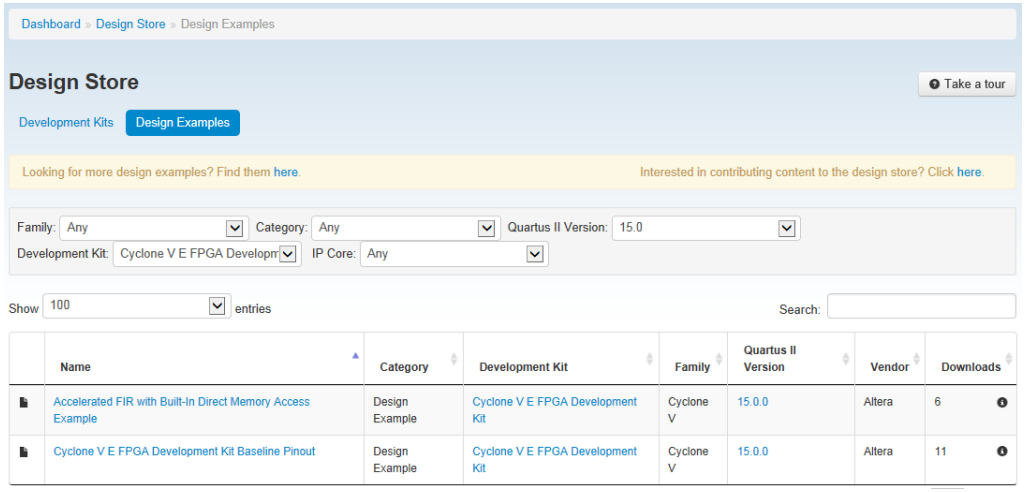

Figure 7: Design Example under Design Store (Note this list changes over time)

Select the Cyclone V E FPGA Development Kit Baseline Pinout.

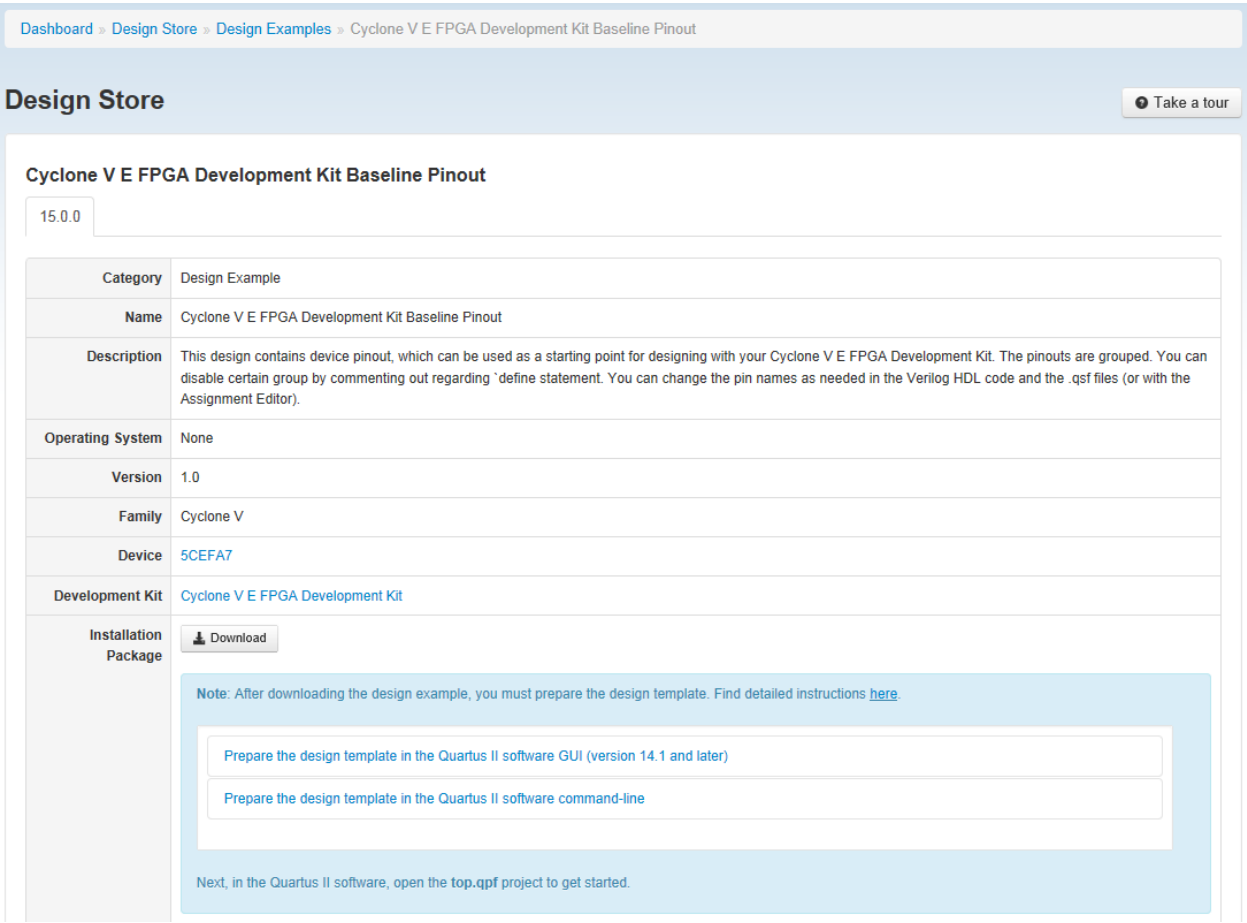

Figure 8: Cyclone V E FPGA Development Baseline Design Example

Select the Download button and save the baseline_c5e.par design locally to your lab working directory (call the directory devkit_hello_world).

Get Started with Quartus

Now you are ready to get started designing hardware! Launch Quartus by double clicking the Quartus icon. Click on Help → About Quartus and check if you are running Quartus v15.0 (Build 145) Some older versions of Quartus don't support Cyclone V devices.

Next you will launch the New Project Wizard from Quartus from the main panel or alternatively File → New Project Wizard.

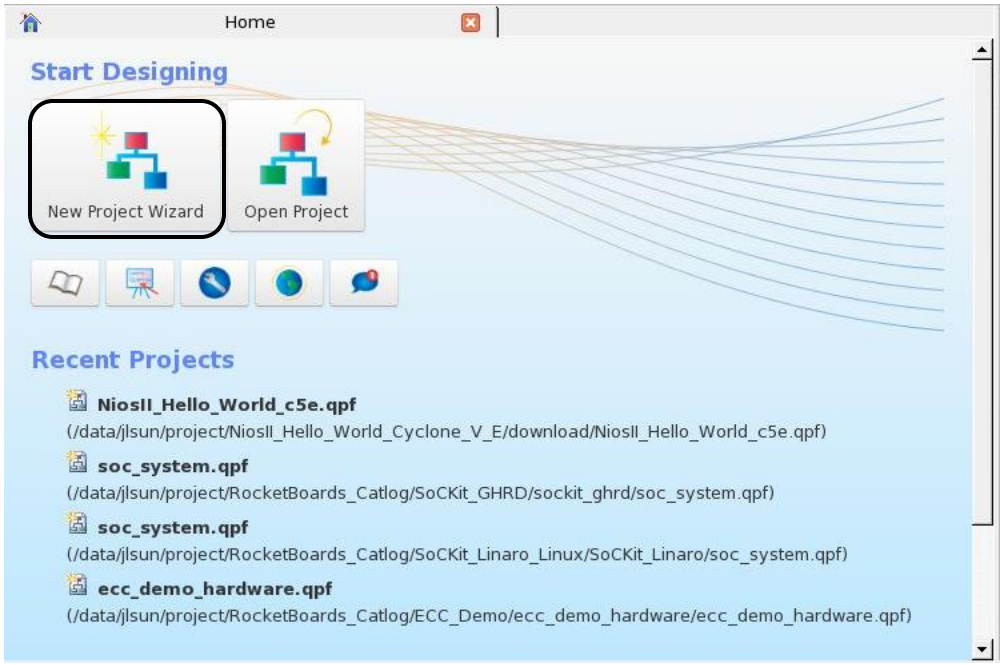

Figure 9: Quartus Main Panel

The Introduction panel will pop up the first time you use Quartus. You can check “do not show me this introduction again.” Then click Next.

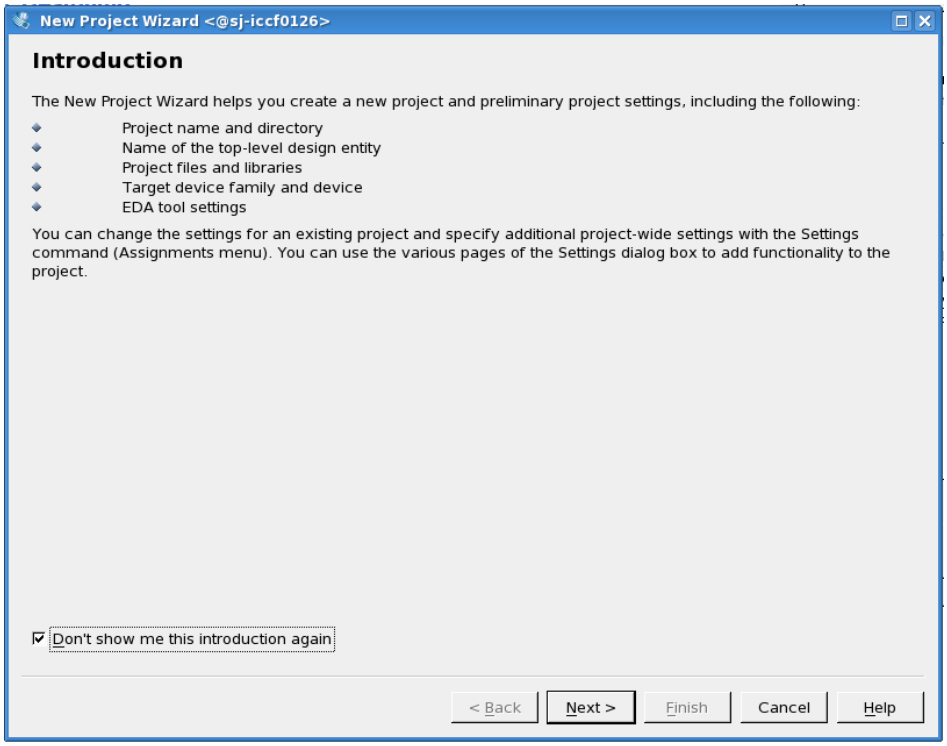

Figure 10: New Project Wizard Introduction

Fill in the next panel with your devkit_hello_world directory and project which we call hello_world_lab. Then click “Next”.

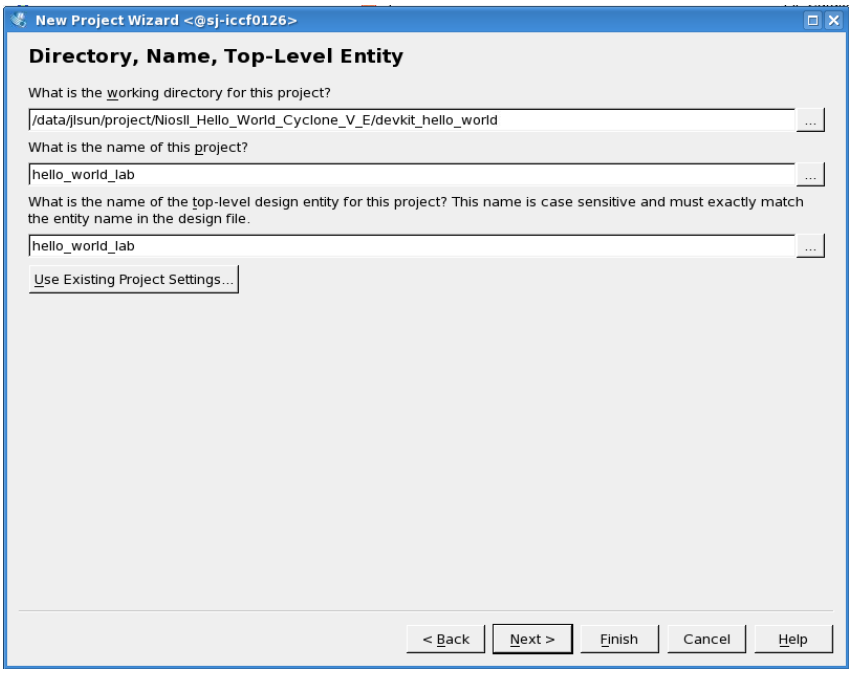

Figure 11: New Project Wizard Panel 1

Select project template and click “Next”.

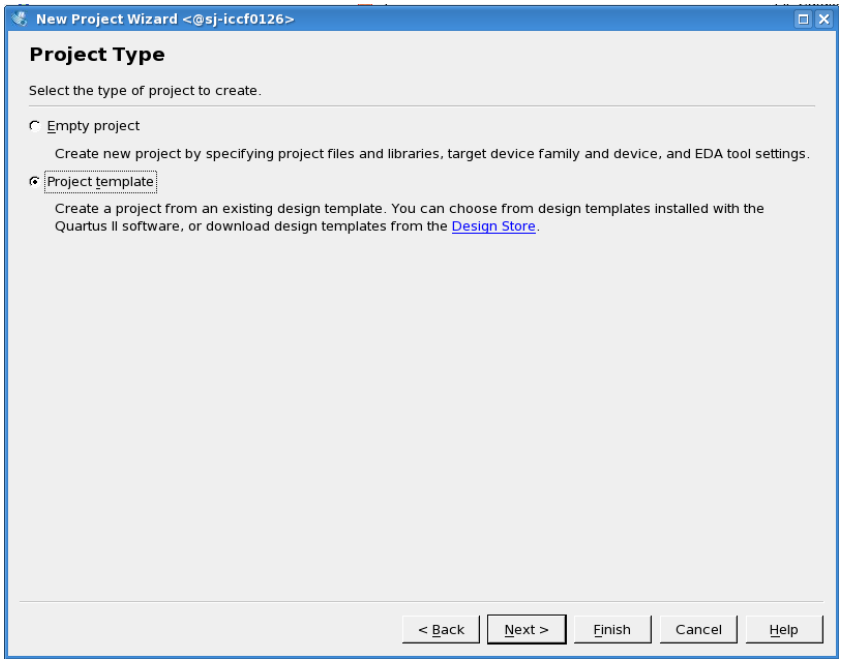

Figure 12: New Project Wizard Panel 2

Click “Install the design templates” then choose the basedline_c5e.par we just downloaded.

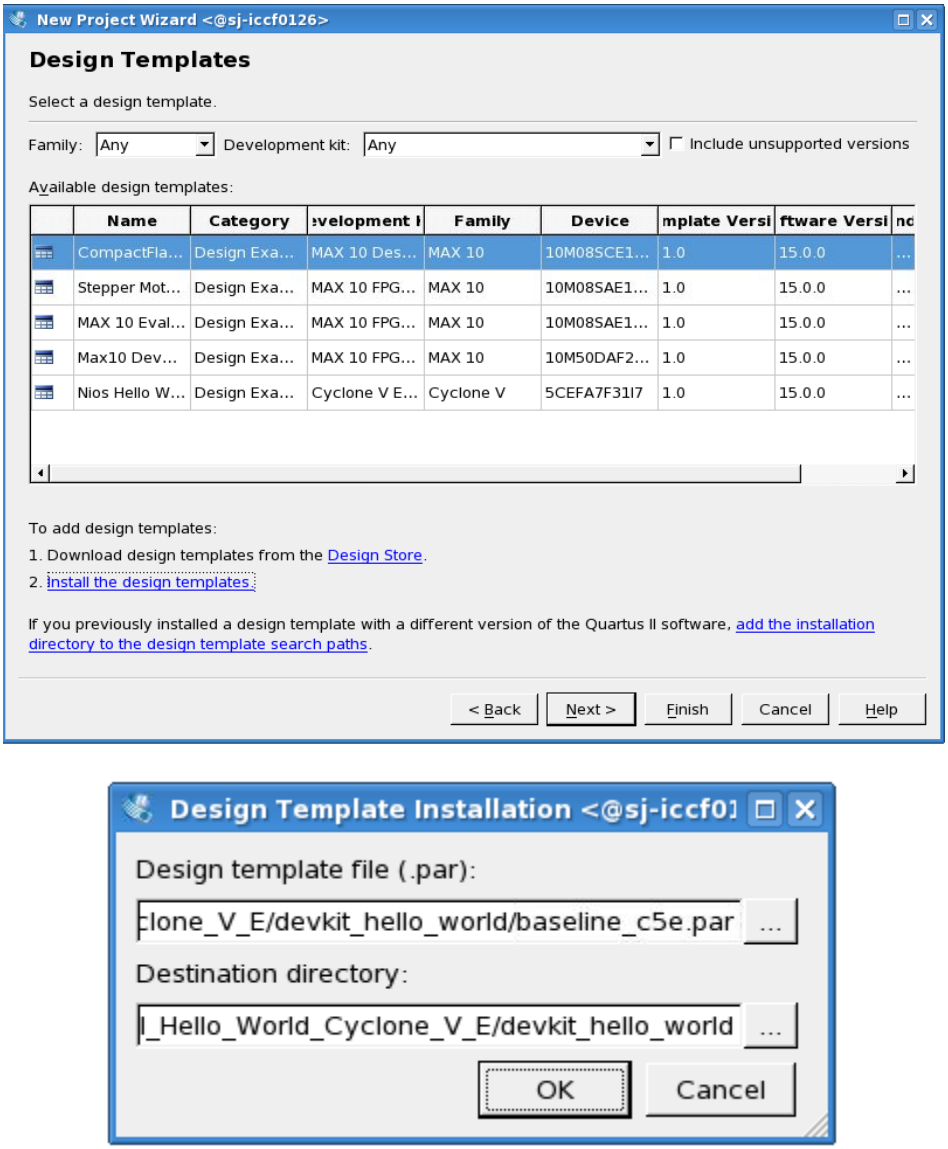

Figure 13: Install Desgin Template

Once you hit ok, Quartus will start installing this design template. This design will then appear in the list of design templates. Click the templates we just installed then hit “Next”. You will see a summary of this design. After going through the summary, click “Finish”. Now you can see that this starting point design that contains the pinout for the Cyclone V E device is successfully loaded in Quartus. Note that only a handful of pins are needed for the lab, but you can rely on the settings utilized in the Baseline project to make sure the right pin locations and voltage settings are correct for your project.

Building Your Qsys Based Processor System

Now we can move on to build our Qsys system.

Launch Qsys from Quartus: Tools → Qsys. The initial screen looks something like this:

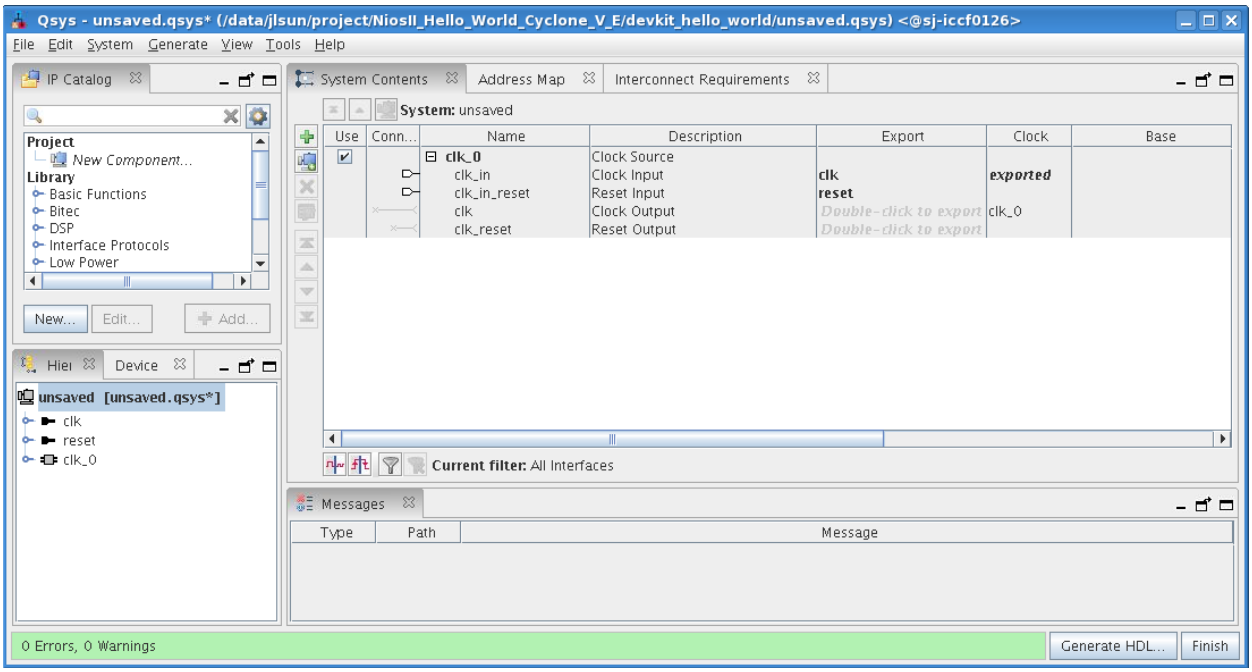

Figure 14: Qsys Main Panel

Next, we will add various components of the system and make the connections between them. By default Qsys inserts a clock module. We will keep this and connect this later on in the lab.

Below the IP catalog tab on the left, you can search for the various components you want to add to your Qsys based system. Enter Nios in the search tab and select the “Nios II Processor” from the library. Double Click on the name or click “Add”.

A configuration window will appear, in this window select the “Nios II/e”. This version of the Nios II processor is resource optimized and will work well for this lab implementation.

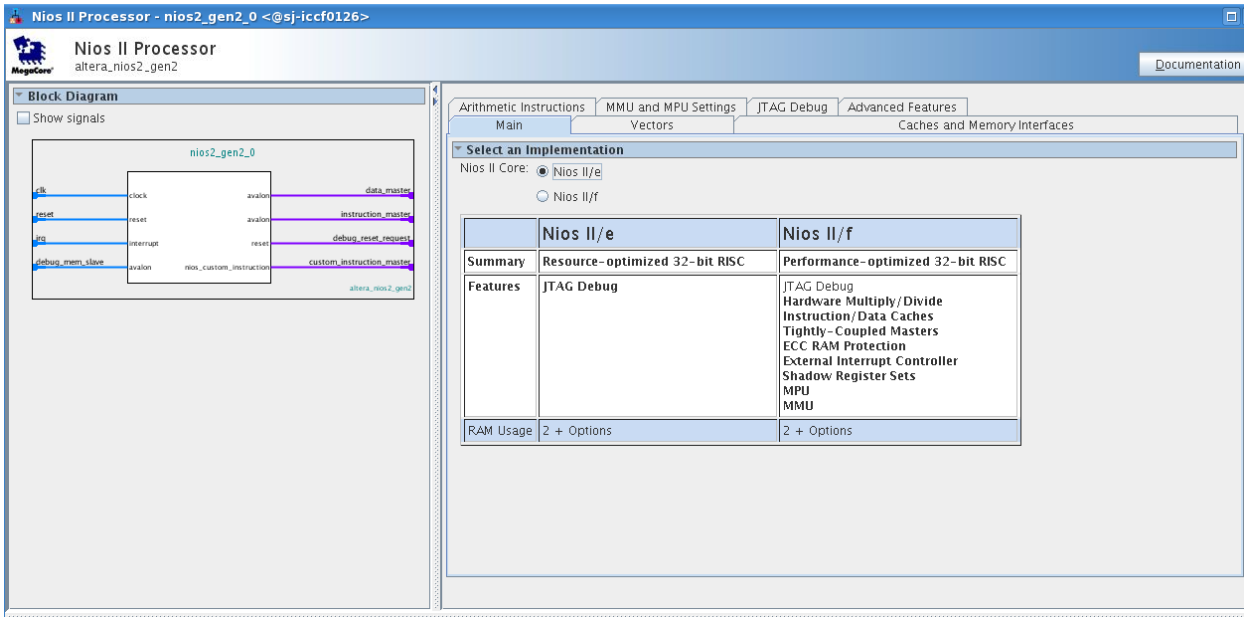

Figure 15: Nios II Processor Configuration Panel

Click finish and you will see the Nios II/e processor in your connection diagram. For now don't worry about the system errors reported, we will address them soon.

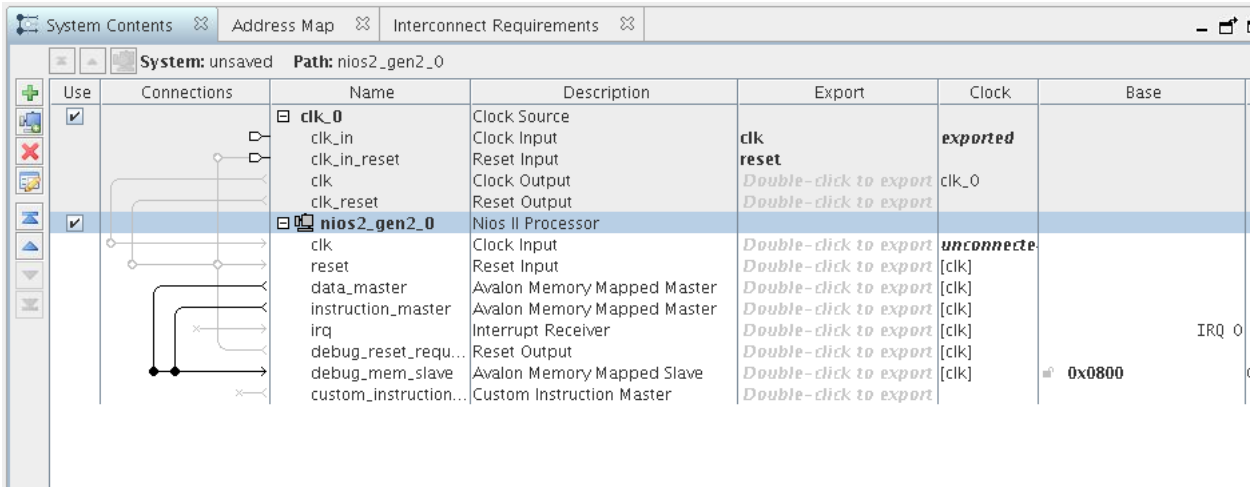

Figure 16: Qsys System Contents Panel

Qsys has a very elegant and efficient way of making connections by clicking on the nodes on 'wires' in the connections panel on the 2nd column from the left. You can add the connections as you add components, but it's often easier to make all the connections once you have finished adding the various blocks.

With the Nios II processor added, you still need to add the On Chip Memory, JTAG UART, PBs and LED to your system.

Search for memory in the IP catalog. You will see many options for memory.

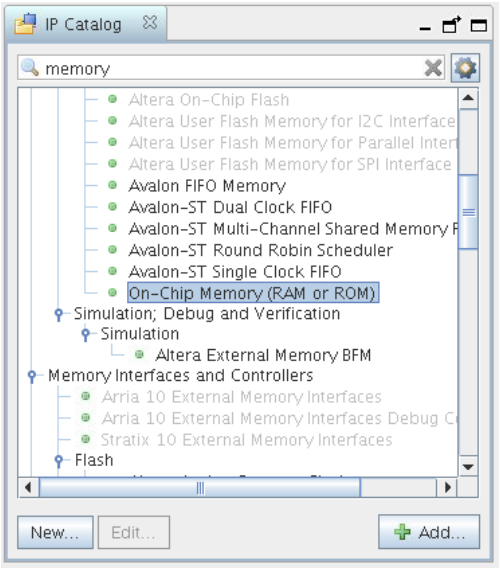

Figure 17: IP Catalog Search for On-Chip Memory

Locate the On-Chip Memory (RAM or ROM) component and click “Add”. You can use all of the default settings except that you need to change the memory size from 4096 to 16384. This will ensure that you have a plenty of space for your software program. Uncheck initialize memory content. This feature includes the software executable in the hardware image. For this lab, you will initialize the software executable from Eclipse.

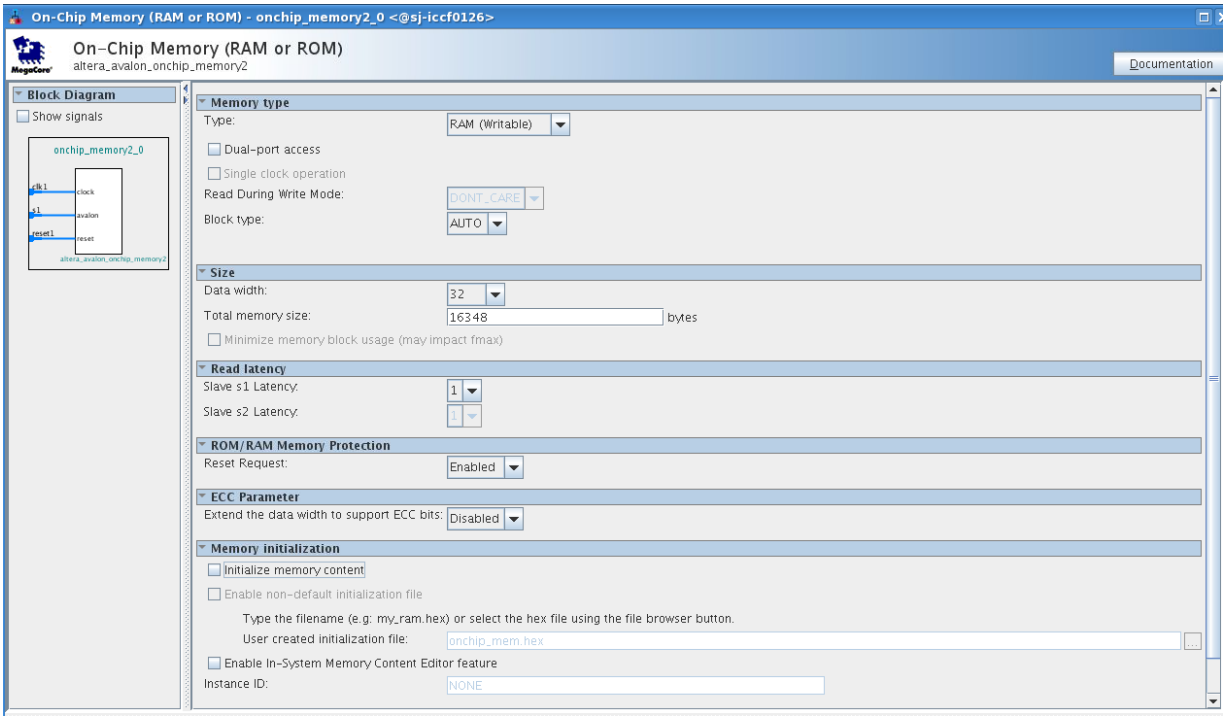

Figure 18: On-Chip Memory Configuration Panel

Click “Finish” and you will now see a total 3 components in your Qsys system: clock, Nios II processor and on-chip memory.

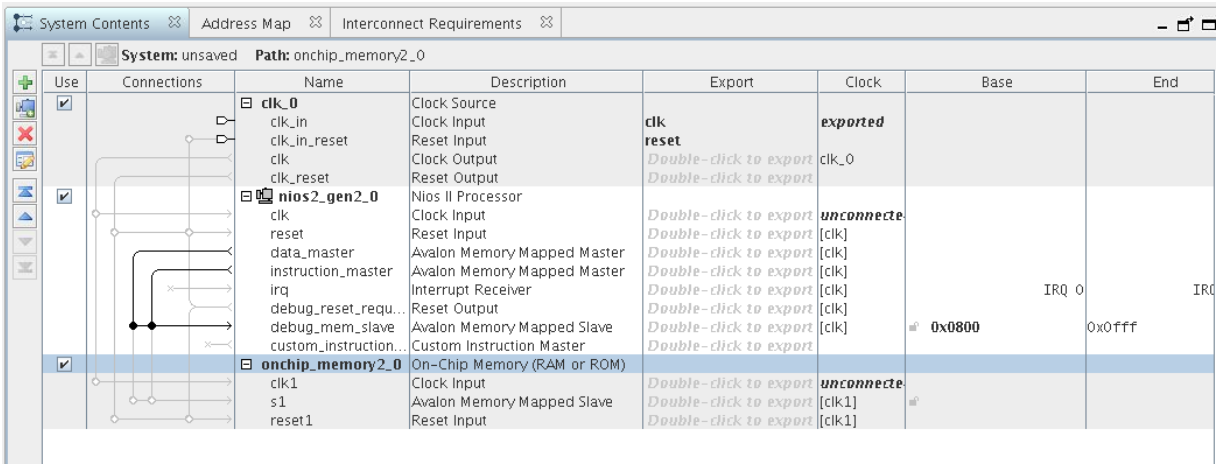

Figure 19: System Contents with Nios II and On-Chip Memory

The next component you will add is the JTAG UART. Search for JTAG in the IP catalog, locate the JTAG UART, double click or click “Add” to add that component. Keep the default settings and click “Finish”.

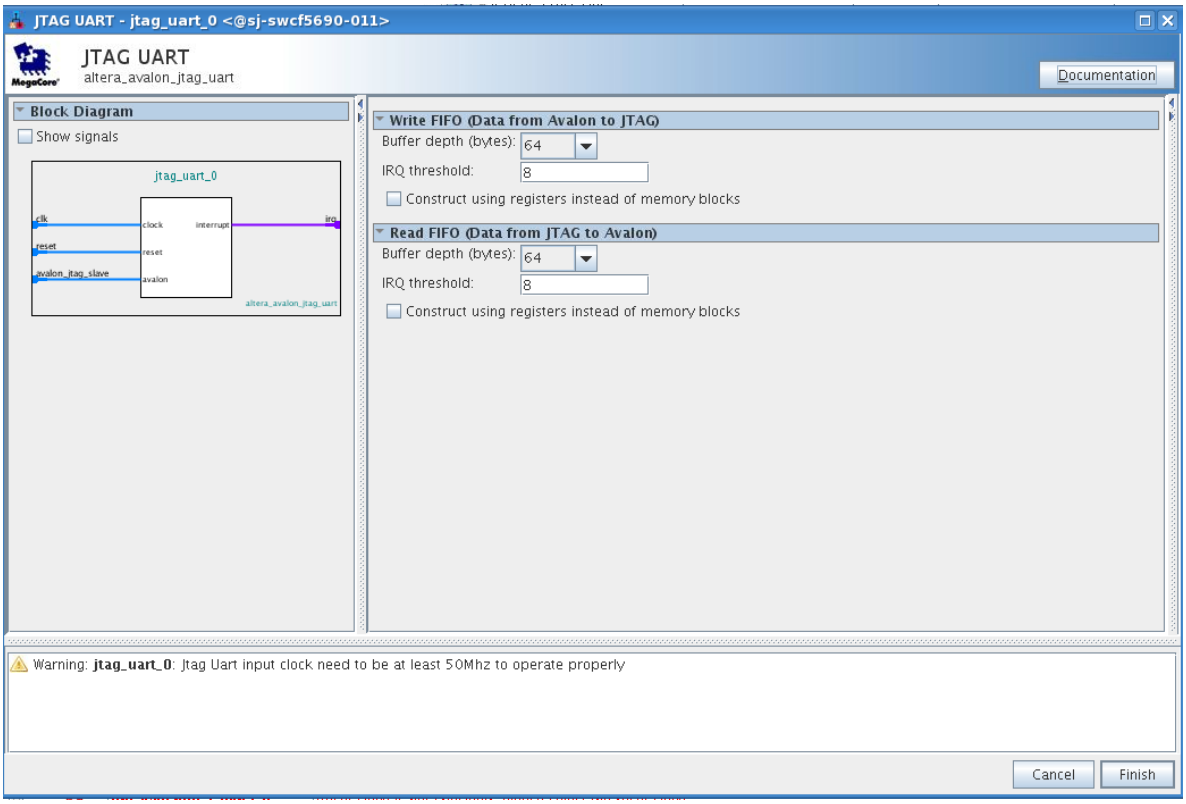

Figure 20: JTAG UART Configuration Panel

The next two components PB and LED are actually configured instances of general purpose parallel IO components in the IP catalog.

Search for parallel IO (PIO) and select this block. For the PB block, you will set this up as a 2 bit input interface using the settings shown below. Click “Finish”.

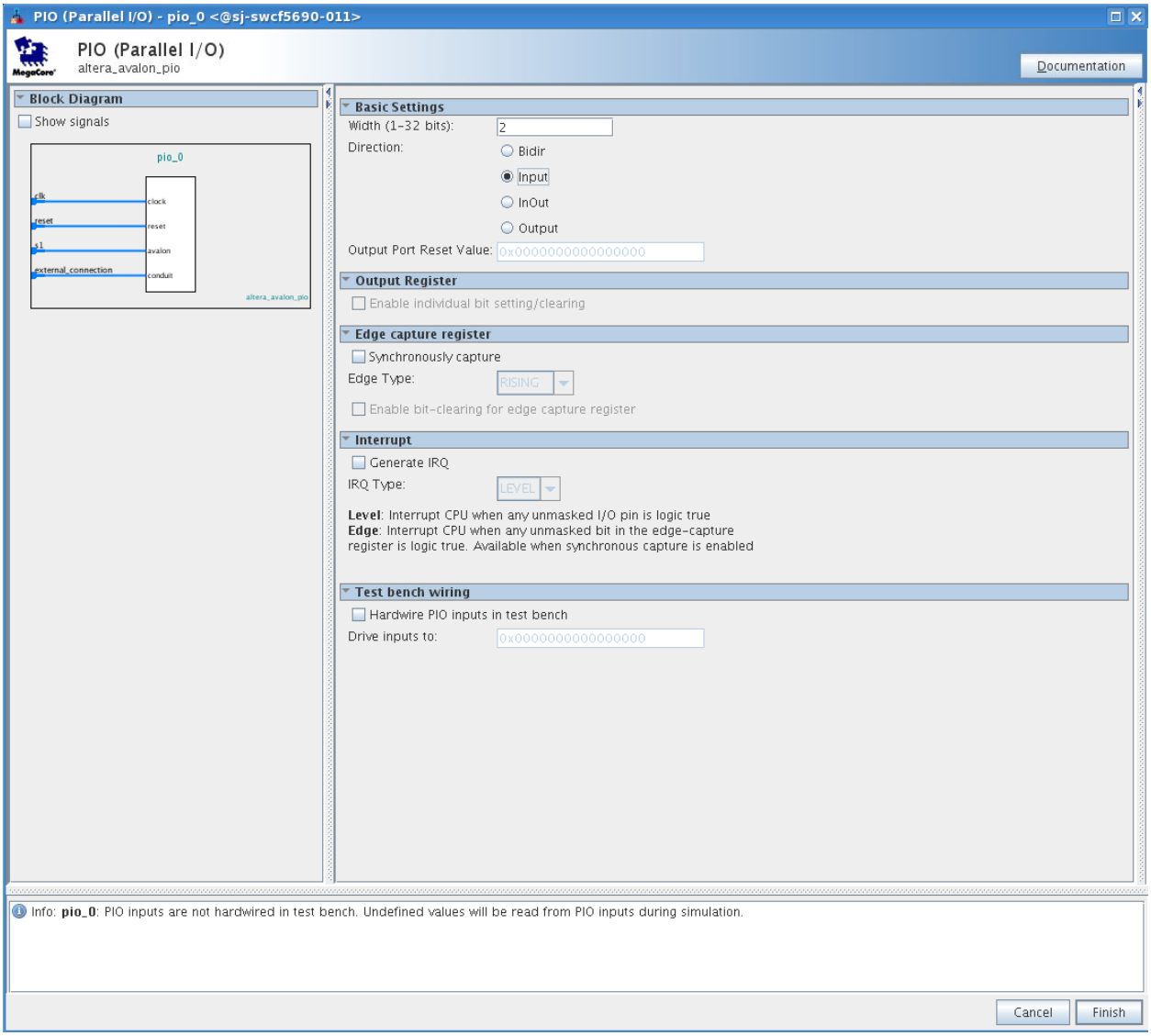

Figure 21: Parallel IO Configuration Panel for the Push Button Inputs

Next, you will add a second PIO block. Double click on the PIO component as you did for the PB. This time you will configure this component as the LED which is a 2 bit output. Click “Finish”.

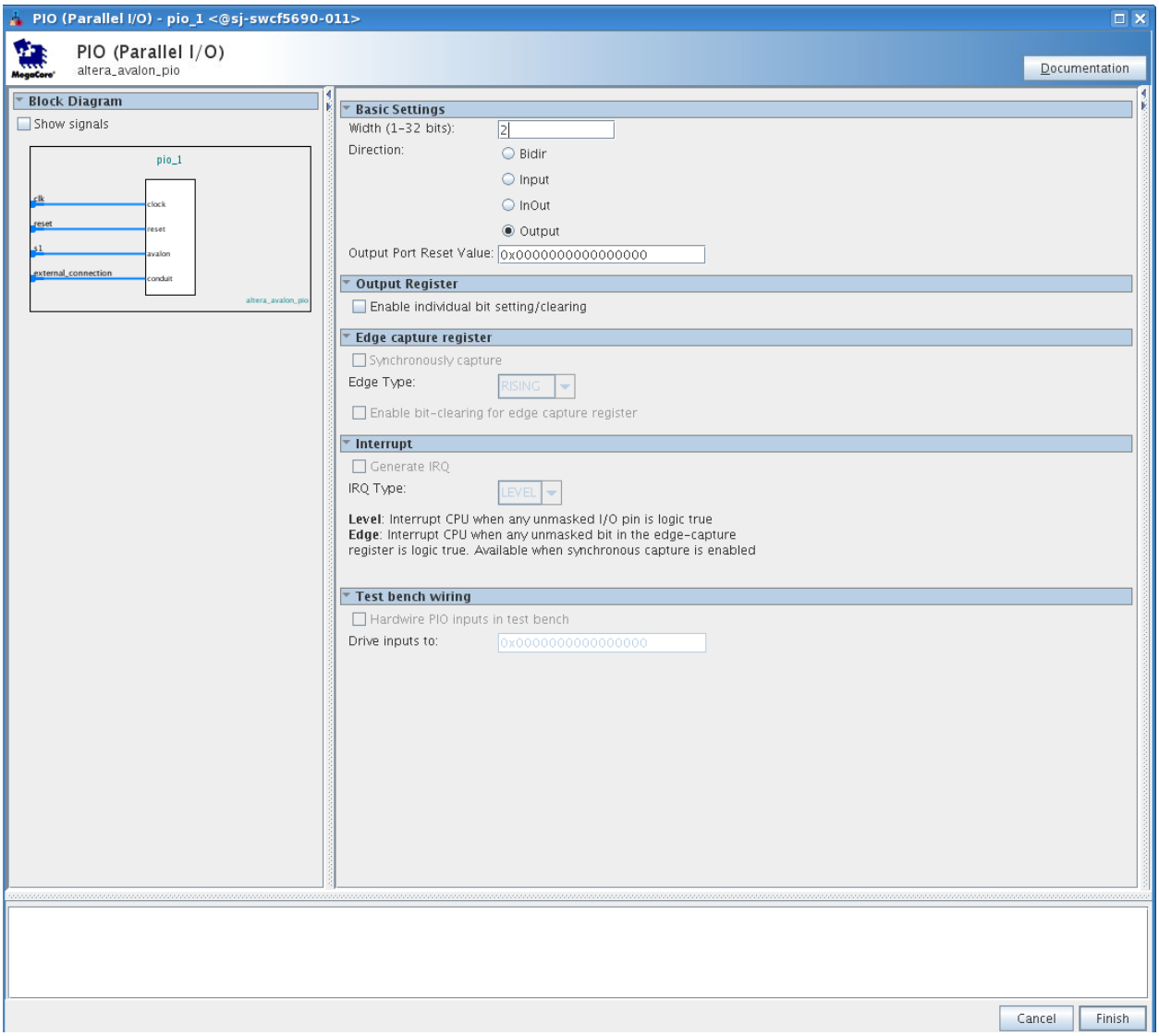

Figure 22: Parallel IO Configuration Panel for LED Outputs

Now you have completed adding the 6 components that make up your Qsys system. Next you will rename the components in the design with names that are easy to remember.

In the system contents tab, right click on the nios2_gen_2_0 component, select rename and type in nios2e. Similarly rename the rest of the components to onchip_memory, uart, pb and led. This will make these components names easy to remember and refer in future steps. Figure 23 shows all the components in our Qsys system after renaming.

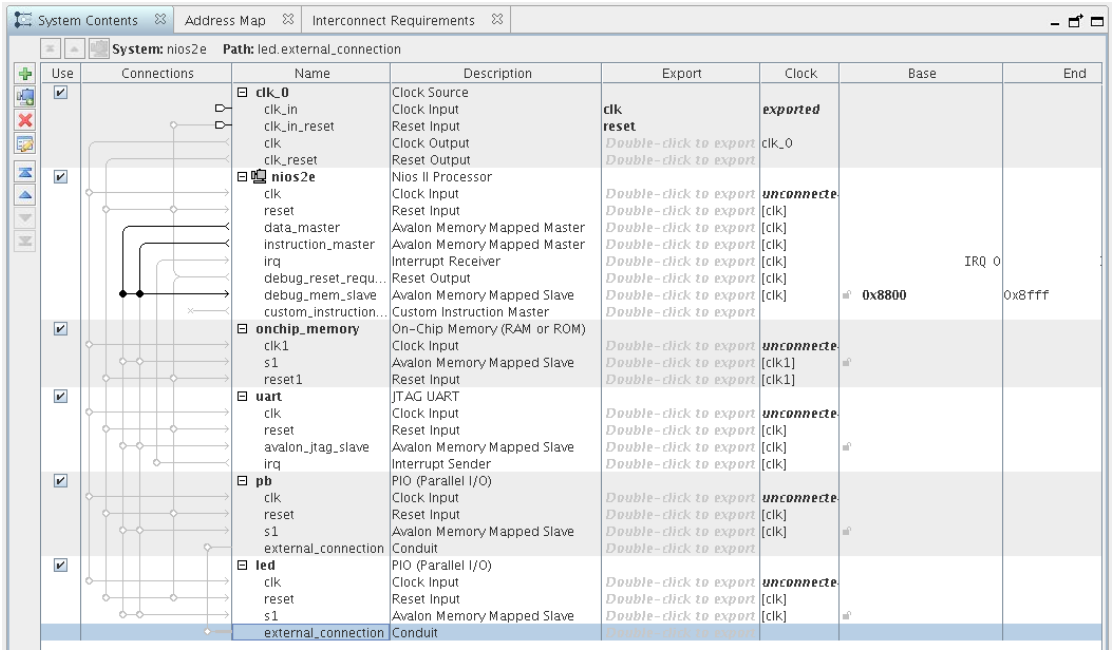

Figure 23: Qsys System Contents after Renaming

The next step is making the appropriate connections between the components within Qsys.

Click on the “clk” net coming out of “clk_0”. When first selected, it will be gray color. Make connections by clicking on the small open circles on the lines that intersecting with the 5 other components.

You should see something similar to Figure 24.

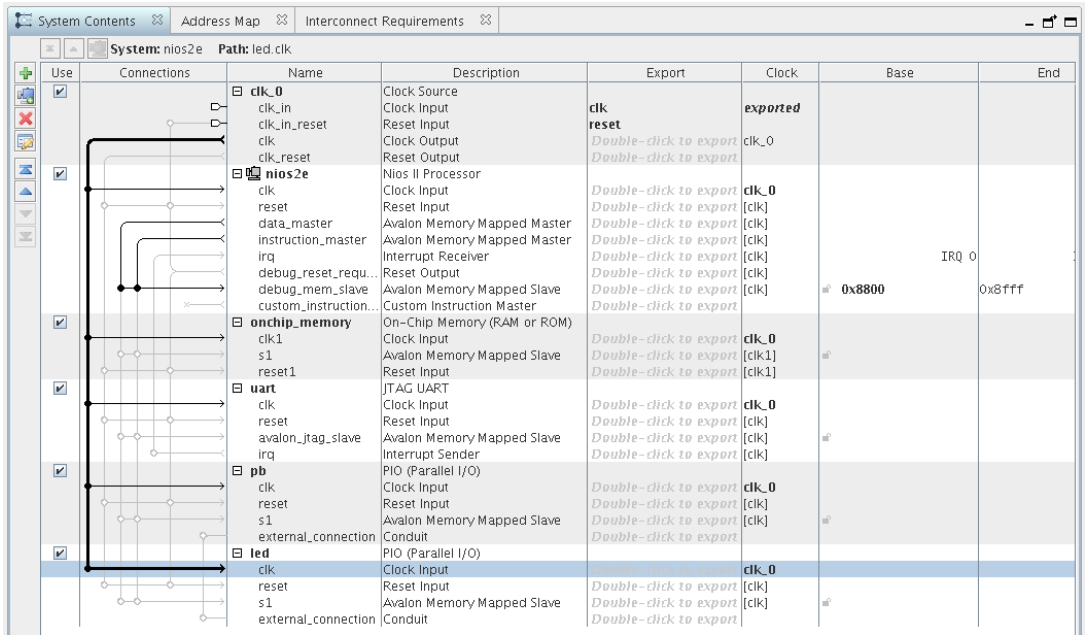

Figure 24: System Contents after Clock Connection

Perform the same operation to connect the “clk_reset” coming out of “clk_0” to the reset nets on the other components.

Next, connect the “data_master” of “nios2e” to all the slaves.

Now the system should looks like Figure 25.

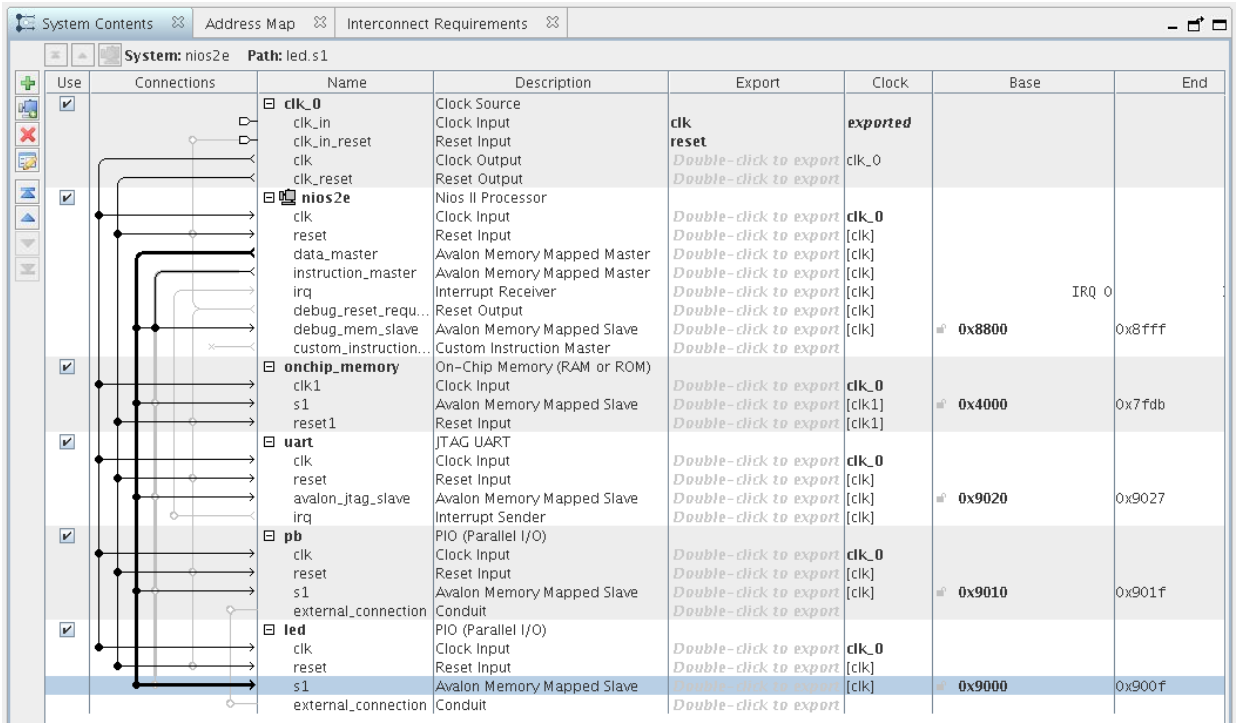

Figure 25: System Contents after Data Master/Slave Connection

The instruction master signal from the nios2e component does not need to be connected to each slave component as it only needs access to memory that contains the software executable. Make the connection between the “instruction_master” of “nios2e” and the “s1” of “onchip_memory”. As shown in Figure 26.

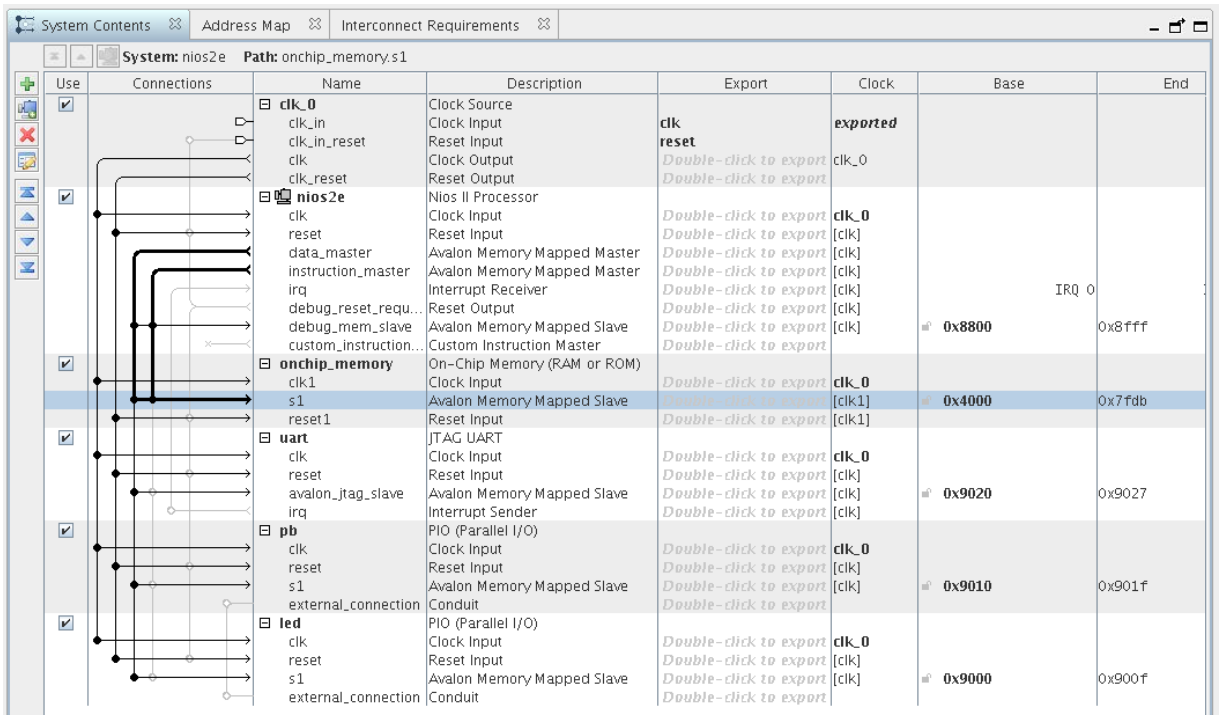

Figure 26: System Contents after Instruction Master/Slave Connection

The next connections to make are the processor interrupt request (IRQ) signals. The UART can drive interrupts and hence needs to be wired to the nios2e processor interrupt lines. Make this connection as shown in Figure 27. We will use the default setting for the IRQ number.

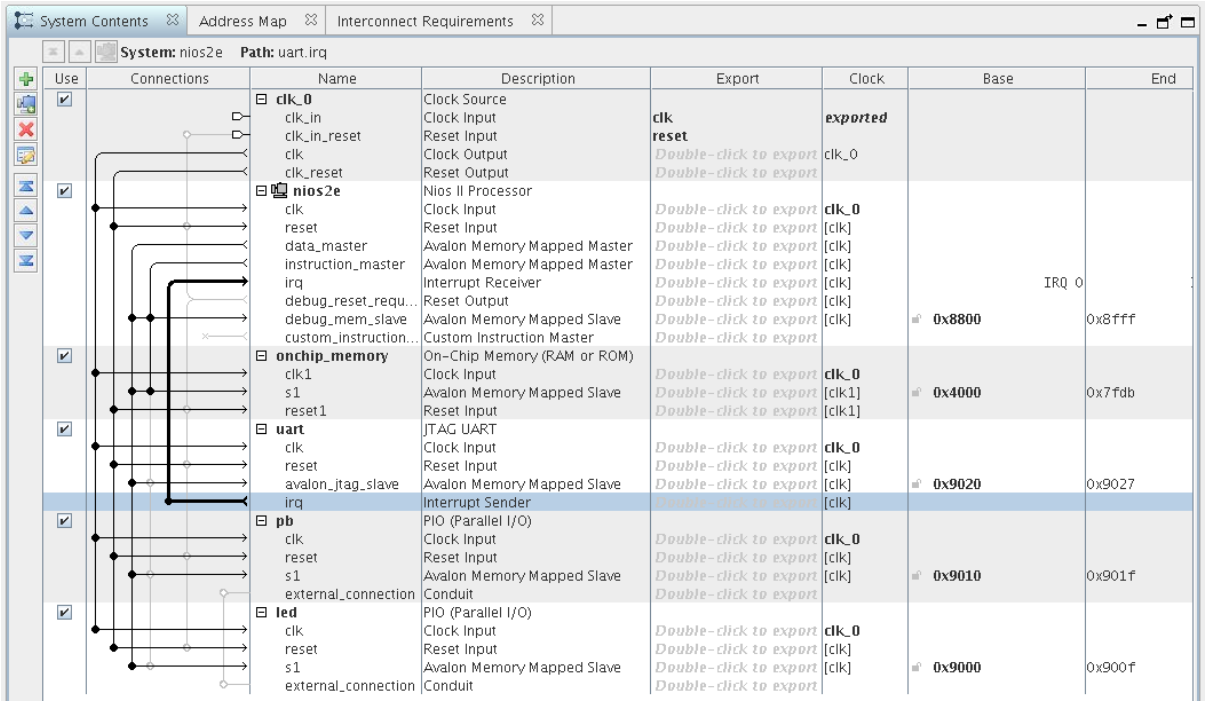

Figure 27: System Contents after Interrupt Connections

You have now completed the internal connections for this Nios II processor based system. The next step is to make the external connections that connect the Qsys based system to the next higher level in the hierarchy of your FPGA design, or to FPGA device pins that connect to the PCB. Double click on the pb and led conduit items under the export column circled in Figure 28. This will bring these ports out of the Qsys component to connect to the top level design.

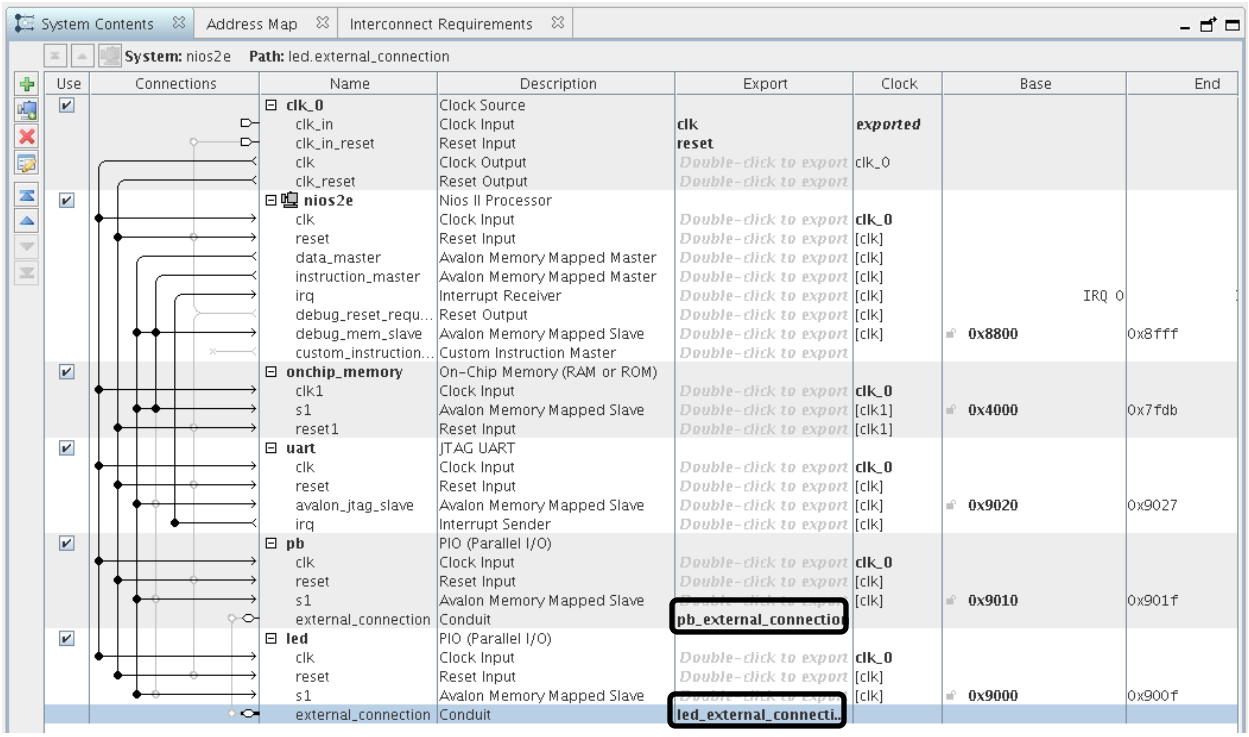

Figure 28: System Contents after Exporting PIO Push Button and LED

Next you will need to generate the base Addresses for your Qsys system. This is achieved by using the command System → Assign Base Addresses.

Save your Qsys system by using File → Save As and pick a name for the Qsys system that you will remember. Note that in the lab we call it “nios2e”. The information is saved in what is called a .qsys file. Although you are not entirely finished, it’s always a good practice to save edits along the way.

You will see 2 error messages in the Message Console of Qsys, as shown in Figure 29.

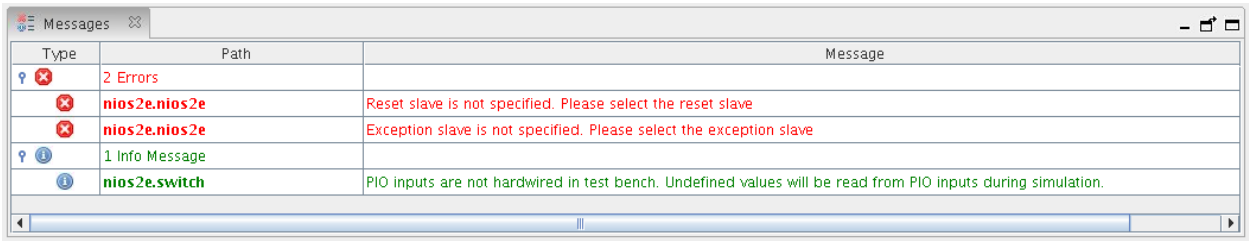

Figure 29: Error Message Prior to Reset and Exception Specification

These error messages have something to do with the fact that nios2e processor doesn't know where the software code that handles resets and exceptions is located. This is fairly straightforward to fix.

Double click on the nios2e component, Click tab "Vectors". Set the "Reset Vector Memory" and "Exception Vector Memory" both to "onchip_memory.s1". This will set the system to execute from the on-chip memory at these respective locations upon reset or interrupt. The 2 errors that were shown in Figure 29 should now be resolved.

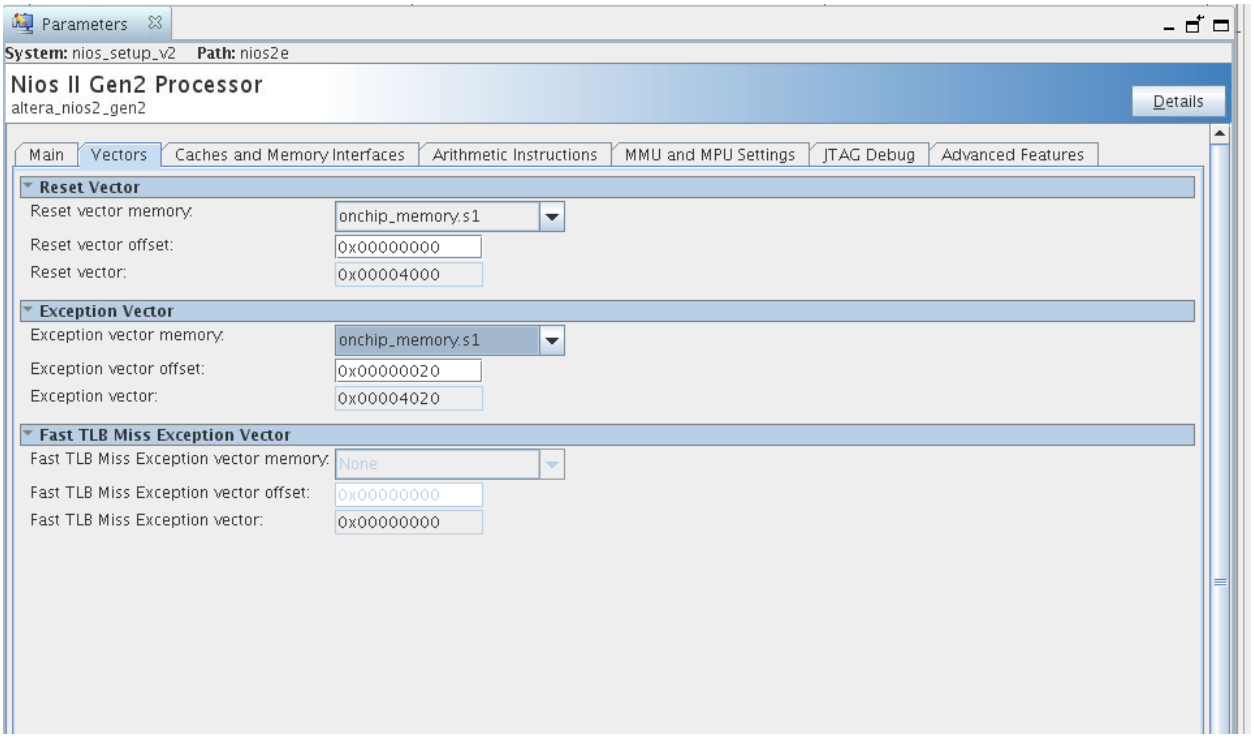

Figure 30: Assign Reset and Exception Vectors in the nios2e Panel

Save your design by clicking File → Save.

Note that by saving, you still have not generated the files that you need for the Quartus II compilation and the Eclipse SBT.

Click on the button "Generate HDL" on the lower right of Qsys.

Use the default settings and click "Generate".

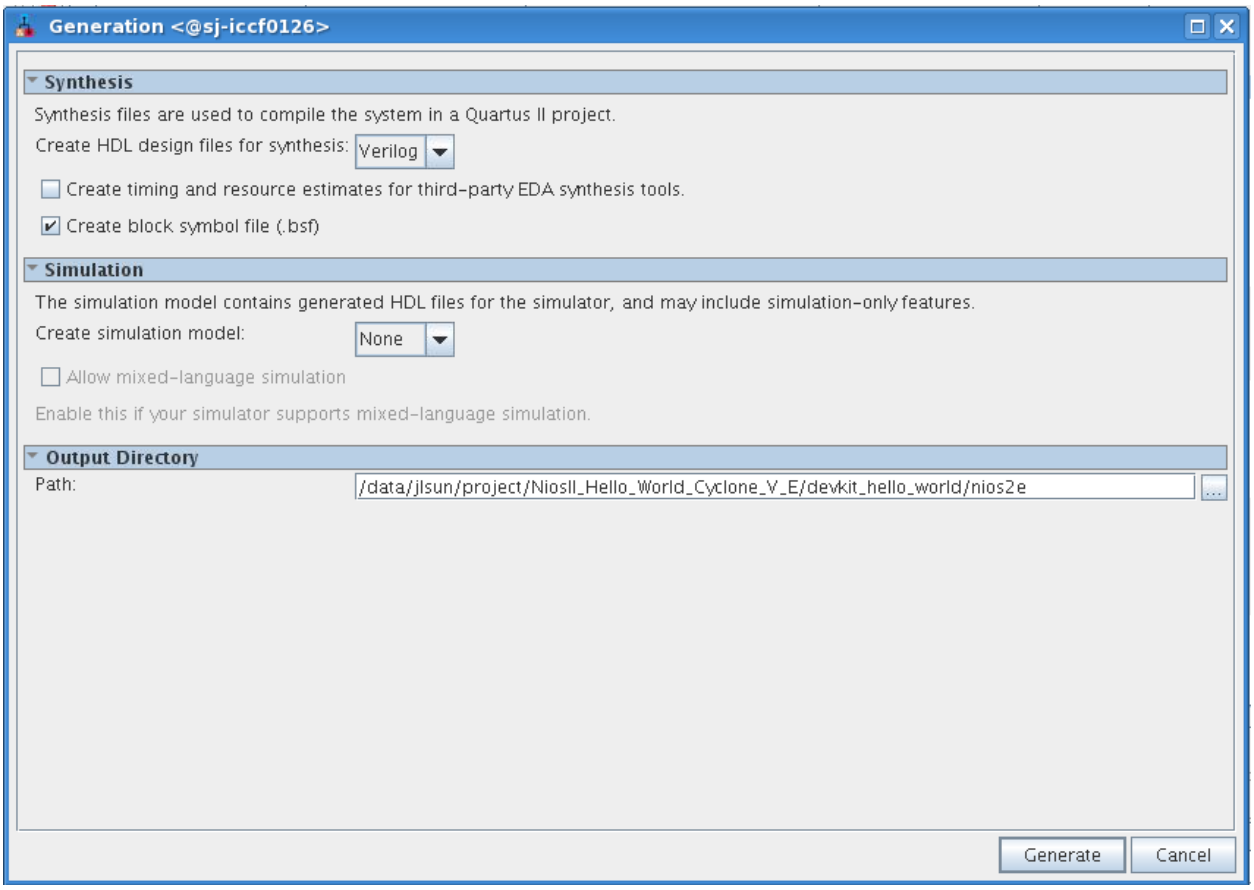

Figure 31: Generating HDL File for the Qsys System

This may take some time. After generating HDL successfully, close the window. Now you can see a folder named “nios2e” along with 2 files (“nios2e.qsys” and “nios2e.sopcinfo”) in your “devkit_hello_world” project folder.

After closing Qsys, you will see this information pop up in Quartus II, as shown in Figure 32. This asks you to include the “.qip” into the Quartus project. We will do this in the later lab.

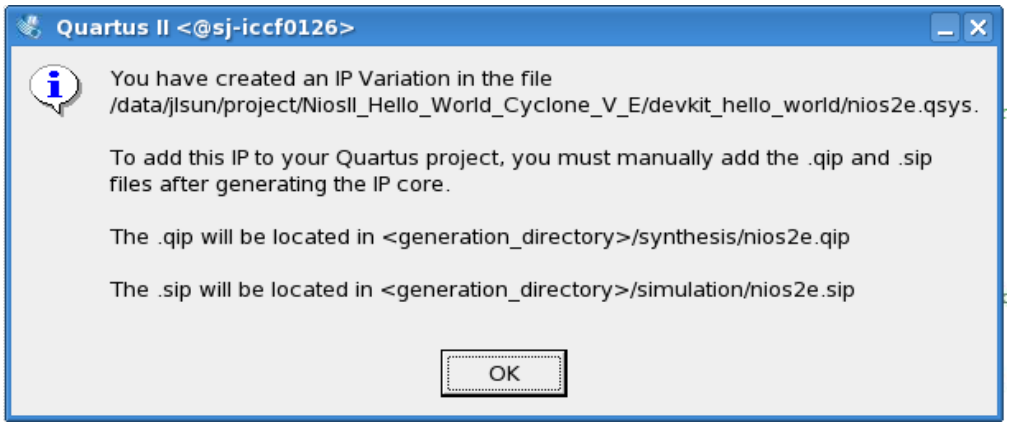

Figure 32: Information after Generating HDL from Qsys

Congratulations, you completed the Qsys section of this lab.

Building the Top Level Design

The next step will take a little bit of knowledge in Verilog. Should you want to use a schematic capture graphical editor, jump to the Appendix A. If you are familiar with VHDL, you can make the same connections in VHDL, but you will have to change the design to VHDL on your own. For ease of following along the lab document, we recommend continuing the lab in Verilog. During the early steps using the project wizard, you loaded the baseline design, and have a baseline_c5e.v preloaded in the Quartus project. We will take a look at this starting point baseline_c5e.v file and strip out the unnecessary signals, while only leaving the signals that are needed to run the Hello World design.

Quartus should be open, bring that to the front of your screen. Make sure the hierarchy tab is highlighted and double click the baseline design. Note that for this design, there is a clock, reset, push button inputs, LED outputs, and a JTAG UART. The JTAG UART pins are hard wired into the FPGA so you don't need to add them in your Verilog source file. The 4 pins: TCLK, TDI, TMS and TDO that constitute a 4 wire JTAG interface are at a fixed location in your FPGA and they don't need to be added to your Verilog source file. Only pins that are synthesized from your RTL source code need to be specified. The baseline design includes all non-hard-wired device pins and you will need to delete extra pins and include the following pins in the port list: clkin_50_fpga_top, cpu_resetn, user_pb, user_led. Delete all other pins from the port list.

Examine basline_c5e.v. Modify the code as shown in Figure 33. The module name is changed to "hello_world", this is also the name of our top-level entity in Quartus II. Note that the name of the .v file must match the name of the top-level entity in order for Quartus to compile. Click File → Save as. Save the file as "hello_world.v". The file should be added to the project directly.

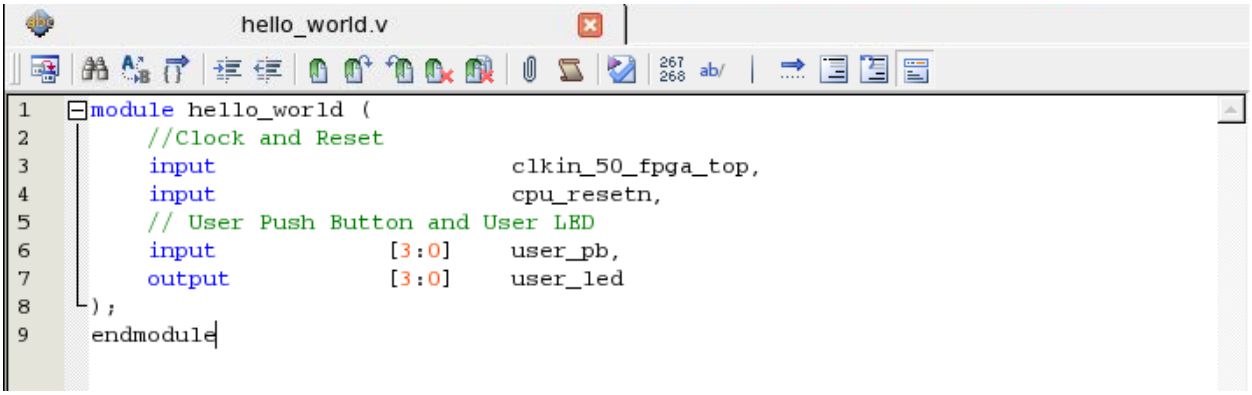

Figure 33: Edited .v File (Saved as hello_world.v)

We can check whether the file is included in the project by clicking the "Files" tab in the "Project Navigator" on the upper left of Quartus II, as shown in

Figure 34. We need to remove the original baseline design file from the project. We can do this by right click on "baseline_c5e.v" and select "Remove File from Project". Now the "hello_world.v" should be the only file in the project. You can also add or remove file by clicking Project → Add/Remove Files in Project, as shown in Figure 35.

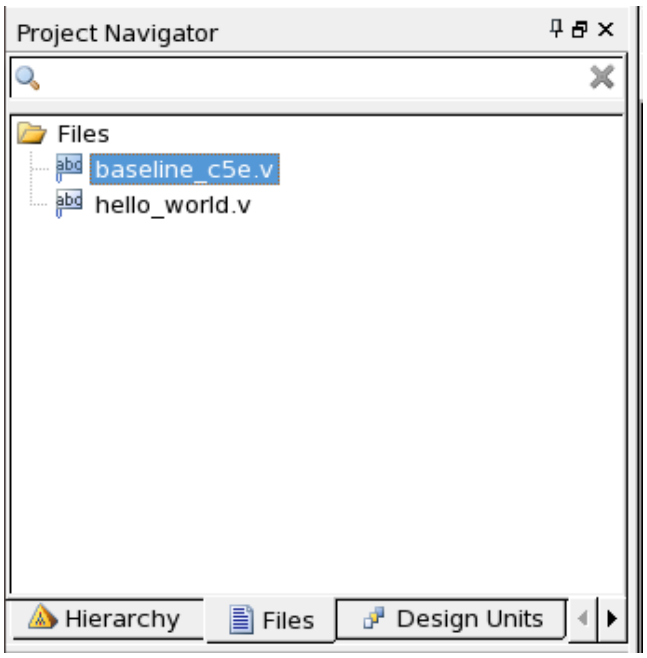

Figure 34: Checking Files Included in the Project

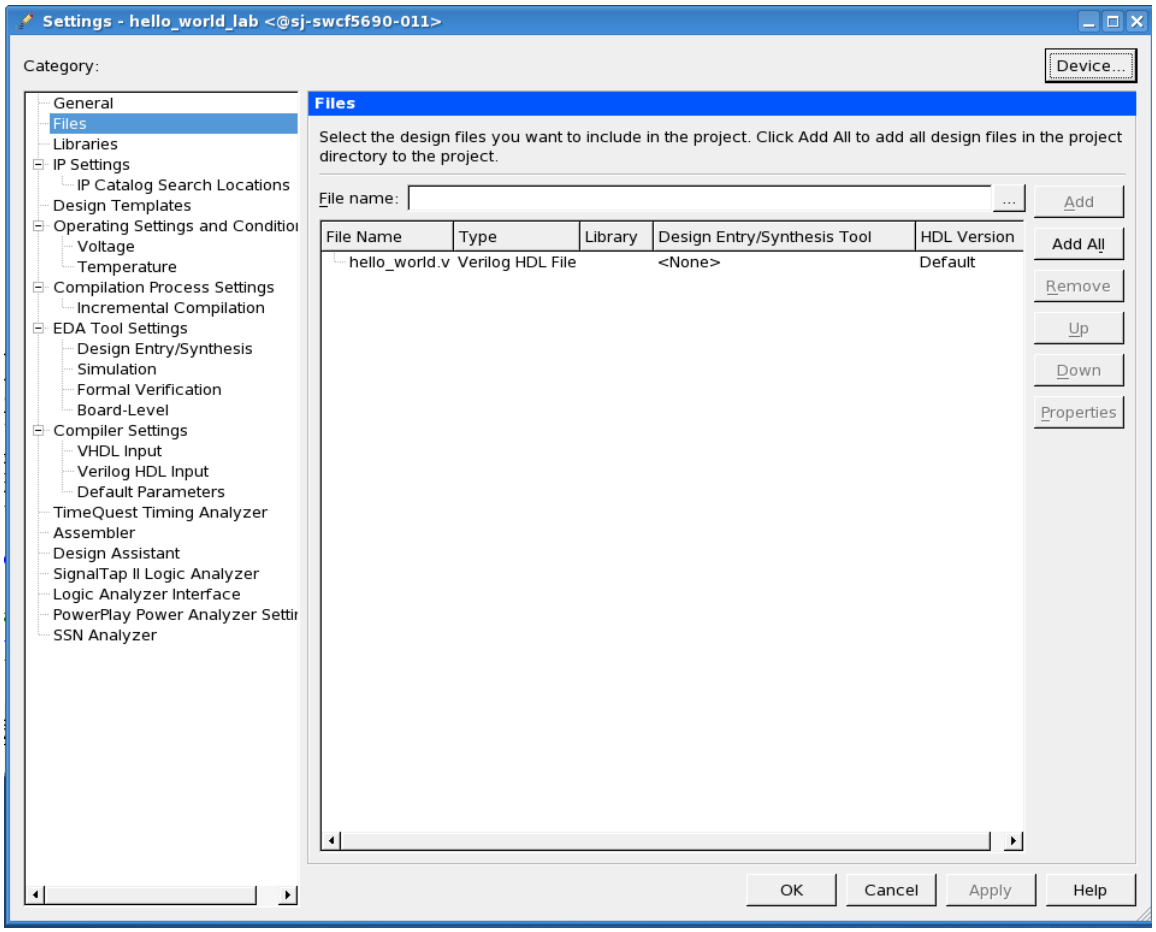

Figure 35: Add/Remove Files Panel

Next you need to make the top level entity “hello_world” since its currently set as “baseline_c5e”. In the same window, click on “General”. Change “Recently selected top-level entities” from “baseline_c5e” to “hello_world”. Click “OK” when complete. You can also change by right clicking on the hello_world.v in Project Navigator and choosing “set as top-level entity”. Now if you go back to “Heirarchy” tab in Project Navigator, you should see the top-level entity is set to “hello_world”.

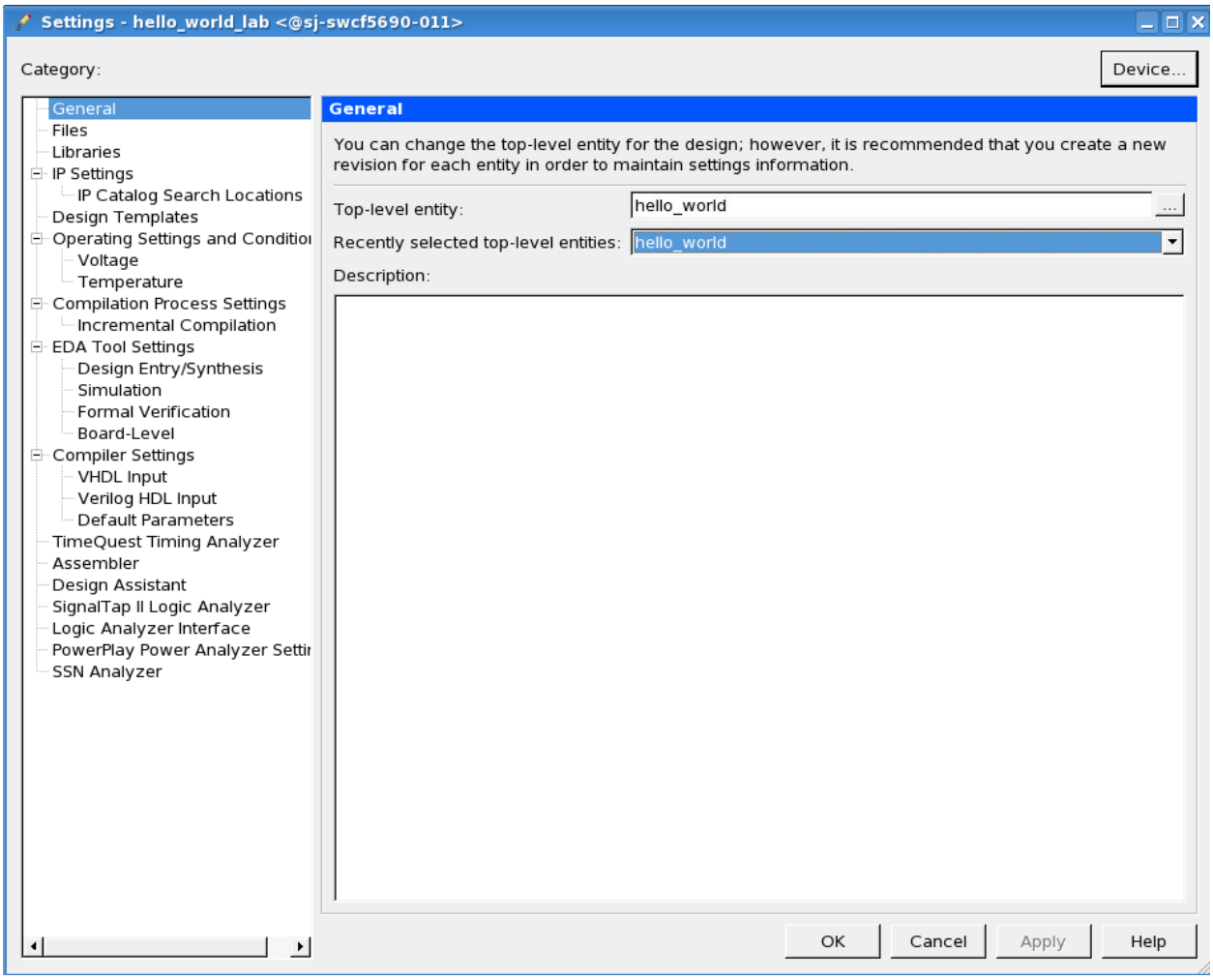

Figure 36: Settings Panel

Now it is a good idea to make sure your Verilog is syntax correct. Return to the main Quartus window to do analysis and synthesis. Double-Click on “Analysis/Synthesis” in the “Task” Panel on the left or click the icon  in the tool bars. This may take some time.

You will get warnings but you should get no errors. If you do get an error, it’s likely to be the syntax error (e.g. missing semicolon). Check your code, make changes, save, and continue to run “Analysis/Synthesis” until the Verilog runs error free (ignore dangling pin warning for now).

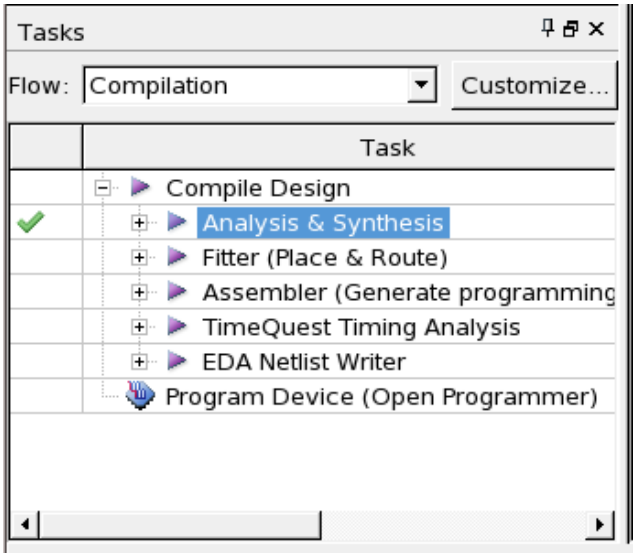

Figure 37: Task Panel

The baseline design that you loaded in the prior steps contain all of the pin settings for your convenience. You can also do that by yourself by checking the schematic and enter the pin locations manually in Pin Planner. Now let's inspect the pin assignments to understand where they come from. Launch Assignments → Assignment Editor. You will see a list of pins in spreadsheet form that contain pin locations, IO standard and current strength settings. Note that you are not using all the pins in the design, but this is ok – Quartus will ignore pin assignments that are not referenced in your design.

hello_world.v | Compilation Report - hello_world_lab | Assignment Editor

<<new>> | Filter on node names: * | Category: All

	ati	From	To	Assignment Name	Value	Enabled	Entity
1	✓		ddr3_a[0]	Location	PIN_A16	Yes	
2	✓		ddr3_a[1]	Location	PIN_G23	Yes	
3	✓		ddr3_a[2]	Location	PIN_E21	Yes	
4	✓		ddr3_a[3]	Location	PIN_E22	Yes	
5	✓		ddr3_a[4]	Location	PIN_A20	Yes	
6	✓		ddr3_a[5]	Location	PIN_A26	Yes	
7	✓		ddr3_a[6]	Location	PIN_A15	Yes	
8	✓		ddr3_a[7]	Location	PIN_B26	Yes	
9	✓		ddr3_a[8]	Location	PIN_H17	Yes	
10	✓		ddr3_a[9]	Location	PIN_D14	Yes	
11	✓		ddr3_a[10]	Location	PIN_E23	Yes	
12	✓		ddr3_a[11]	Location	PIN_E20	Yes	
13	✓		ddr3_a[12]	Location	PIN_C25	Yes	
14	✓		ddr3_ba[0]	Location	PIN_J18	Yes	
15	✓		ddr3_ba[1]	Location	PIN_F20	Yes	
16	✓		ddr3_ba[2]	Location	PIN_D19	Yes	
17	✓		ddr3_casn	Location	PIN_L20	Yes	

Figure 38: Assignment Editor

You can also check the pin locations in Pin Planner, which is more intuitive. Launch Assignments → Pin Planner. It looks like Figure 39.

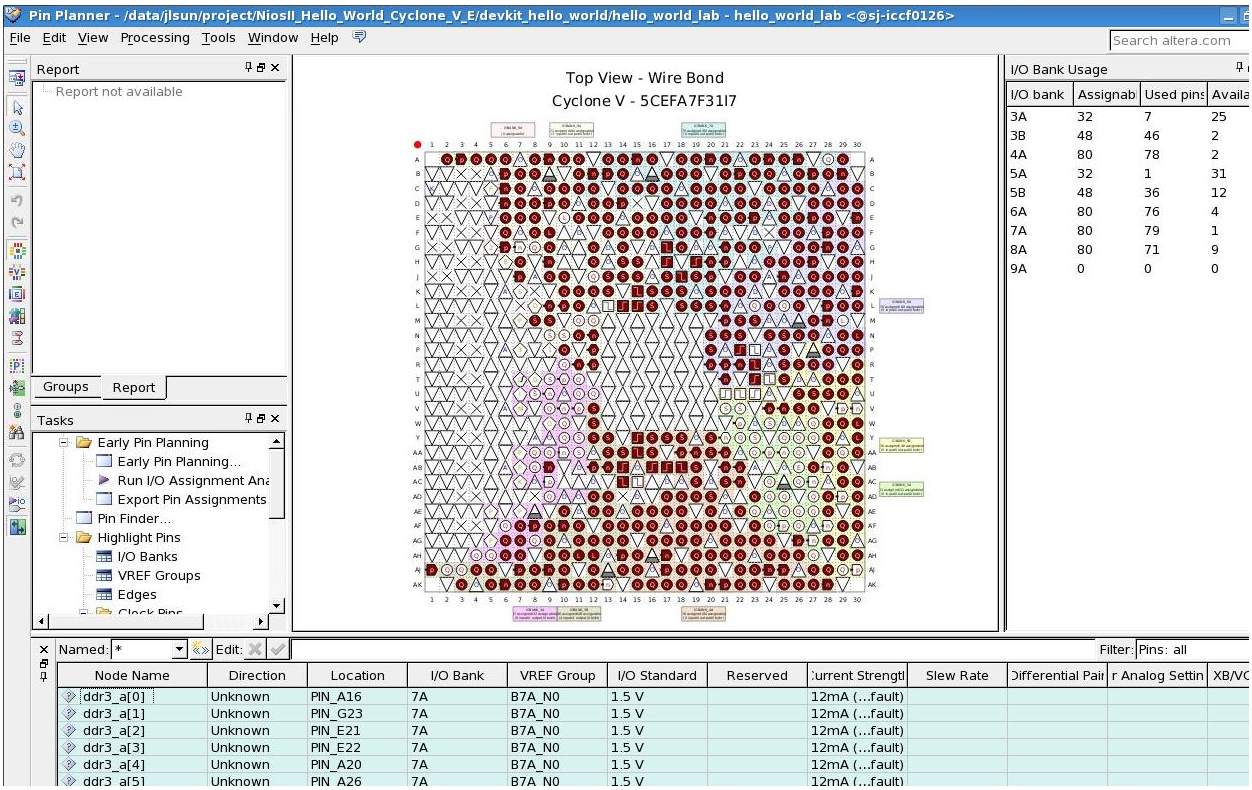

Figure 39: Pin Planner

The baseline design we used doesn't assign the location for the "cpu_resetn" port. We need to assign that in Pin Planner. Filter Ports by typing "cpu_resetn" in the "Named" entry then press Enter. Double click the location entry and type "AA26" as shown in Figure 40.

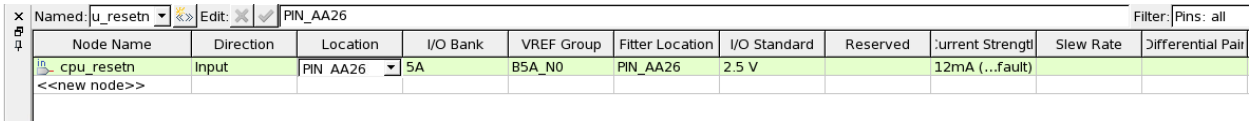

Figure 40: Assign cpu_resetn Port Location in Pin Planner

Adding the Nios II System into Your Design

Now that you have the hello_world entity completed and syntactically correct, you will need to add the Nios II Qsys system into your design. Qsys makes this task quite convenient. Go to File → Open and navigate to the name of the Qsys project you created (the one shown in this lab is called nios2e). You should see a file called "nios2e_inst.v". Open this file and you can see how to instantiate the Qsys system. The content of this file is shown in Figure 41.

Figure 41: nios2e_inst.v

Copy this entire code to “hello_world.v”. Connect the IO ports of Quartus to the Qsys system by replacing the fancy words in the parenthesis to the names of ports we defined, as shown in Figure 42.

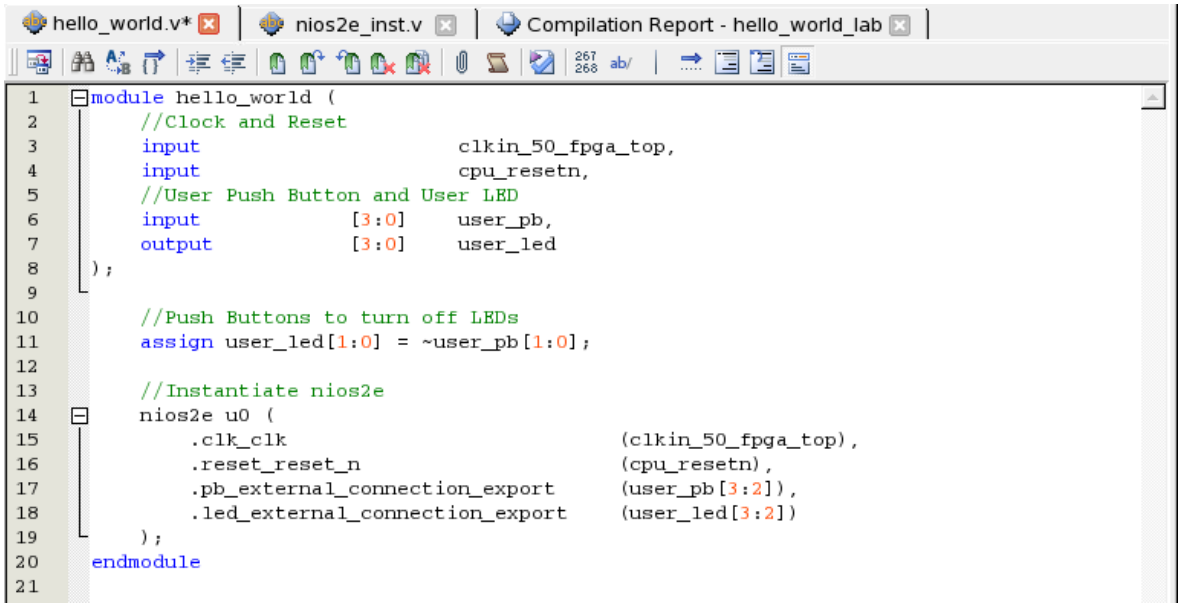

Figure 42: hello_world.v with Harware Assignment and nios2e Instantiation

In this code we also connect Push Button [1:0] to LED [1:0] in an inverted way. We will also connect Push Button [3:2] to LED [1:0] in software later in this lab. In this way, pushing buttons 1 and 0 will turn *off* LED1 and 0 respectively through hardware connection, while pushing buttons 3 and 2 will turn *on* the LED 3 and 2 respectively in software applications.

Click the save icon or File → Save.

The last step before compilation is adding “nios.qip” file into the Quartus project. The qip file contains pointers to the location of all the generated source files generated from Qsys and necessary settings required to compile.

Click Project → Add/Remove Files, navigate using the 

button and select the “nios.qip” file located in “nios2e/synthesis” folder. Hit Add followed by OK.

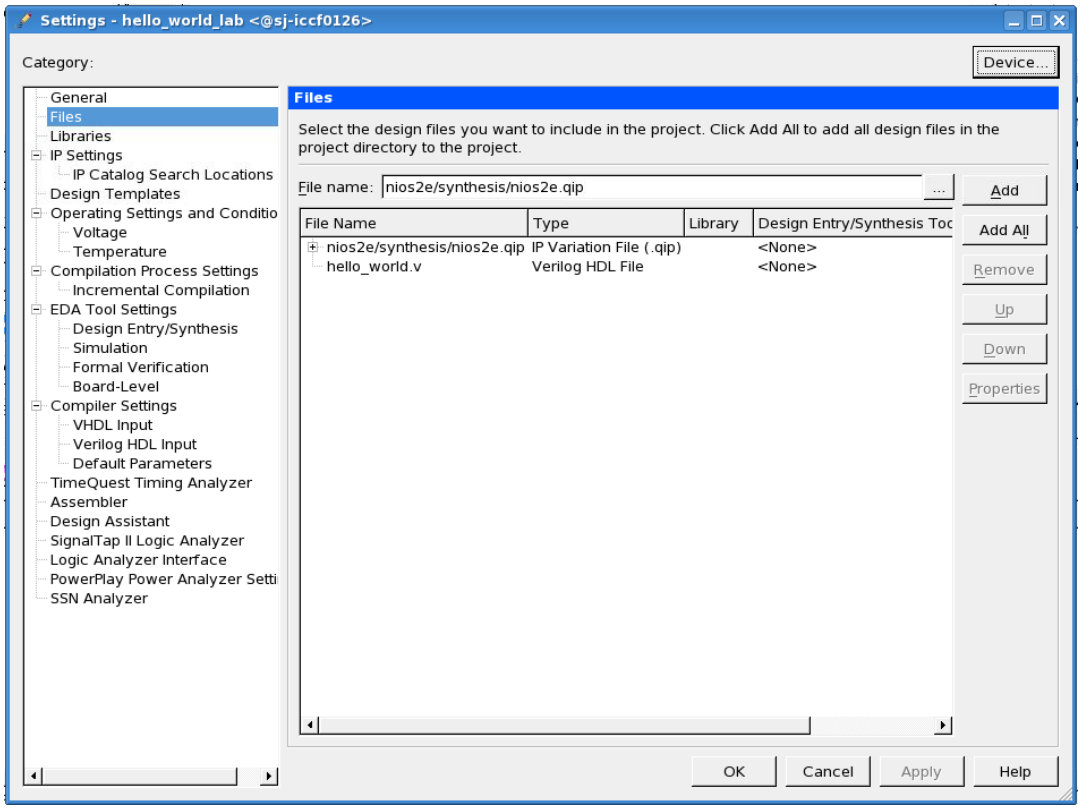

Figure 43: Add .qip File to Quartus Project

Now you can compile your design which will run Analysis & Synthesis, Fitter (place and route in FPGA terminology), Assembler (generate programming image) and TimeQuest (the static timing analyzer). This can be achieved by clicking on the play button  or double clicking the "Compile Design" in the Task Panel. Note that if you are using ES dev kit, change device to the one ended with ES to make it match.

Note that some warnings and information messages come up in the bottom window. You can filter them by message level. Errors are filtered with the  button. Critical warnings are filtered with the  button. Warnings are filtered with the  button. Informational messages are filtered with the  button. You cannot proceed if you have errors.

In this case there are only standard warnings. This is primarily because we did not add timing constraints to this project. Due to the simplicity of this design and low frequency, it's okay to start without timing constraints. Consult other Altera online training courses for instructions on how to add timing constraints to your design.

Congratulations, your FPGA hardware design is now complete.

SOFTWARE DESIGN

Creating the Software for the "Hello World" Design

Should you choose to start directly in the Software Design section and skip the Hardware Design section, consult with your lab facilitator to get these two files: nios2e.sopcinfo and hello_world.sof, which are supposed to be generated from the Hardware Design lab. You will be able to complete all subsequent steps with these two files.

The Nios II Software Build Tools (SBT) for Eclipse are included as part of Quartus. These tools will help manage creation of the application software and Board Support Package (BSP). Board Support Package contains libraries, linker file, drivers for IP blocks, along with Hardware Abstraction Layer (HAL). Launch it from Tools → Nios II Software Build Tools for Eclipse. You can use the default location that Eclipse picks for you. It's usually better to create a folder called "software" in your Quartus project folder "devkit_hello_world" and choose it as the workspace. Click "OK" to continue.

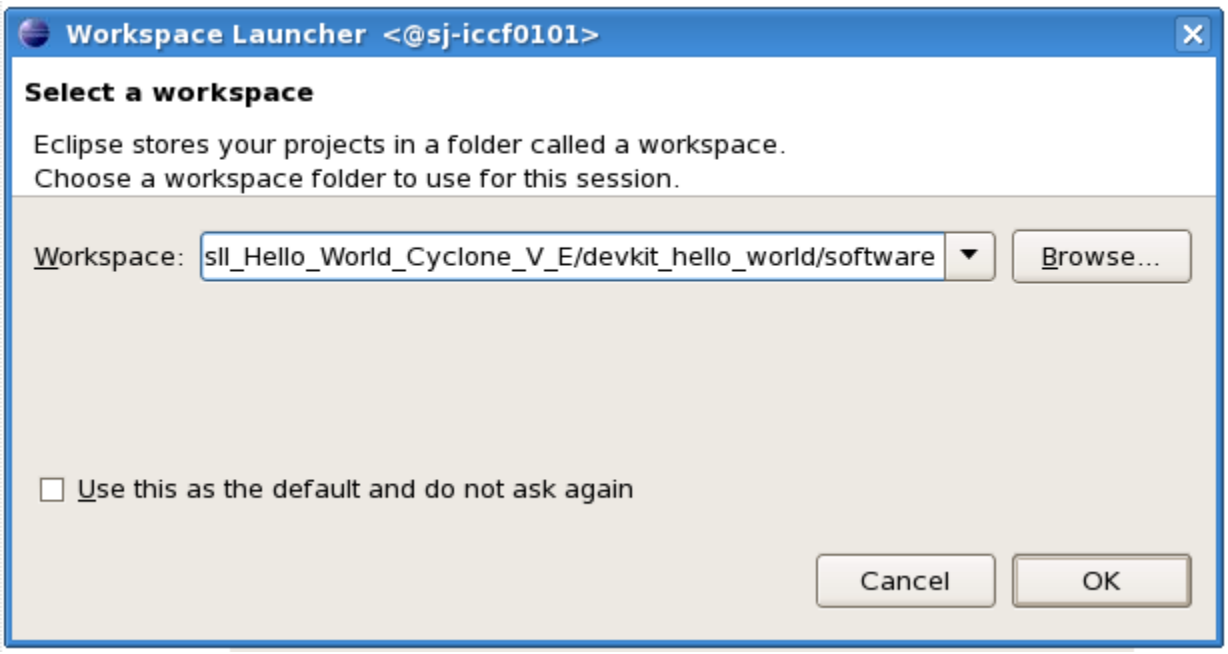

Figure 44: Eclipse Workspace Setup

After Eclipse SBT finishes the launch process, right click in the area called "Project Explorer" and select New→Nios II Application and BSP from Template.

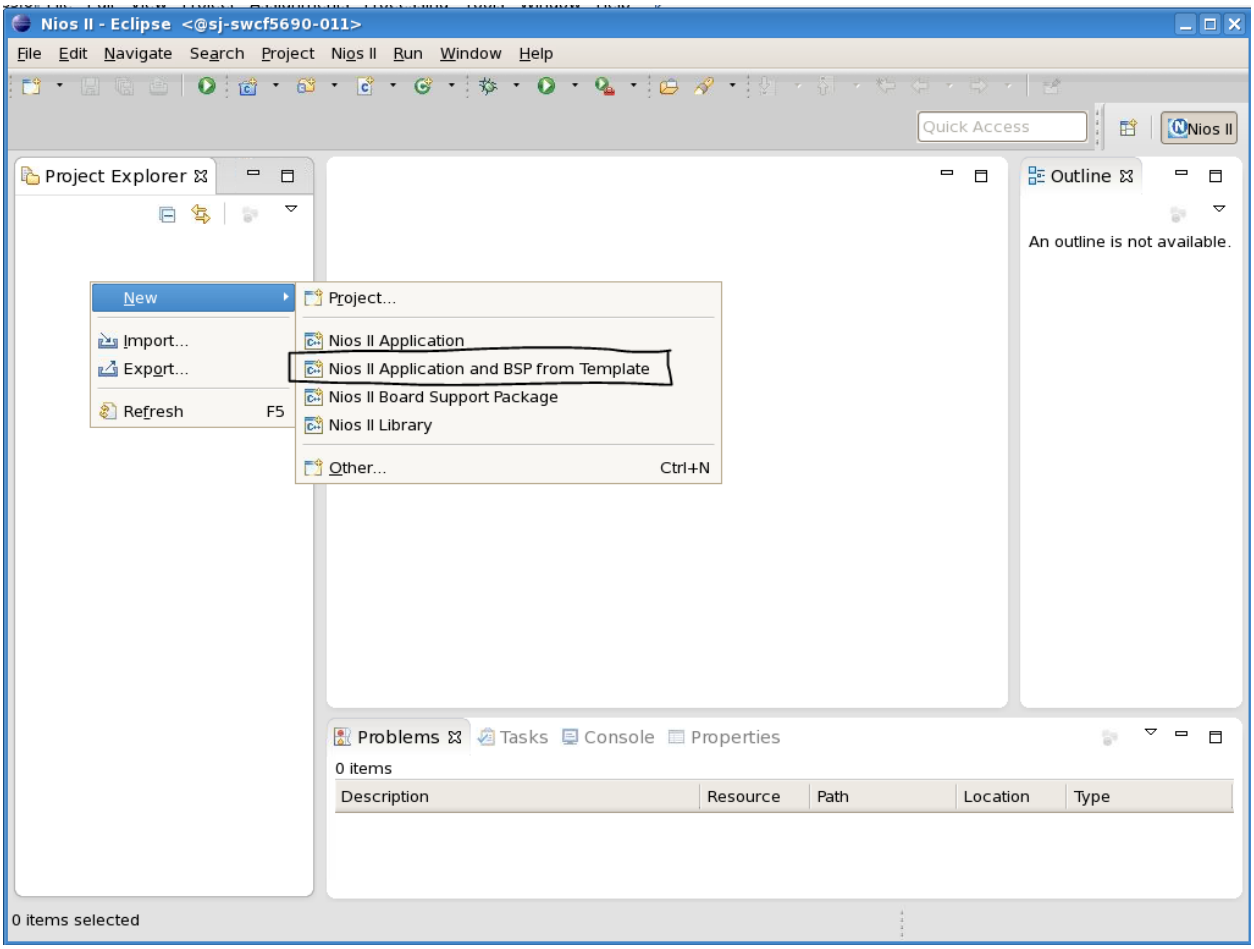

Figure 45: Creating Project from Template in Eclipse SBT

Next you will see a panel that requests information to setup your design. First you need to setup the target hardware information. Click  to navigate to your Quartus project directory and click on the “nios2e.sopcinfo” file. The .sopcinfo file tells Eclipse what your Qsys system contains. Click OK.

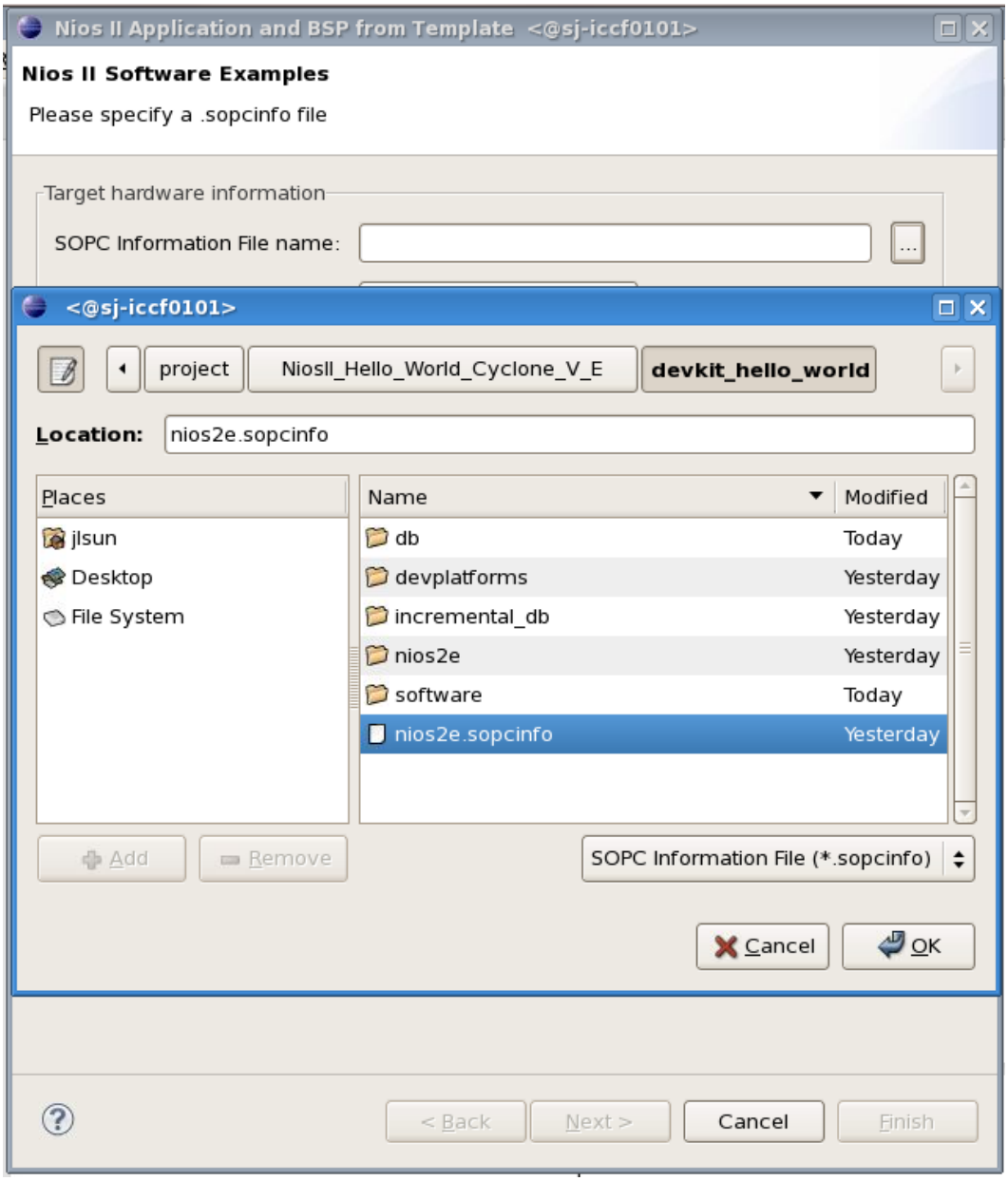

Figure 46: Navigating to the Correct .sopcinfo File

Fill in the Project name. Call it hello_world_sw. Next you will be asked to pick a template design. The Hello World Small is a software application to write “Hello from Nios II” to the screen. Click Finish. Note: make sure to pick Hello World Small and not Hello World or you will not have enough memory in your FPGA design to store the program executable. Click Finish.

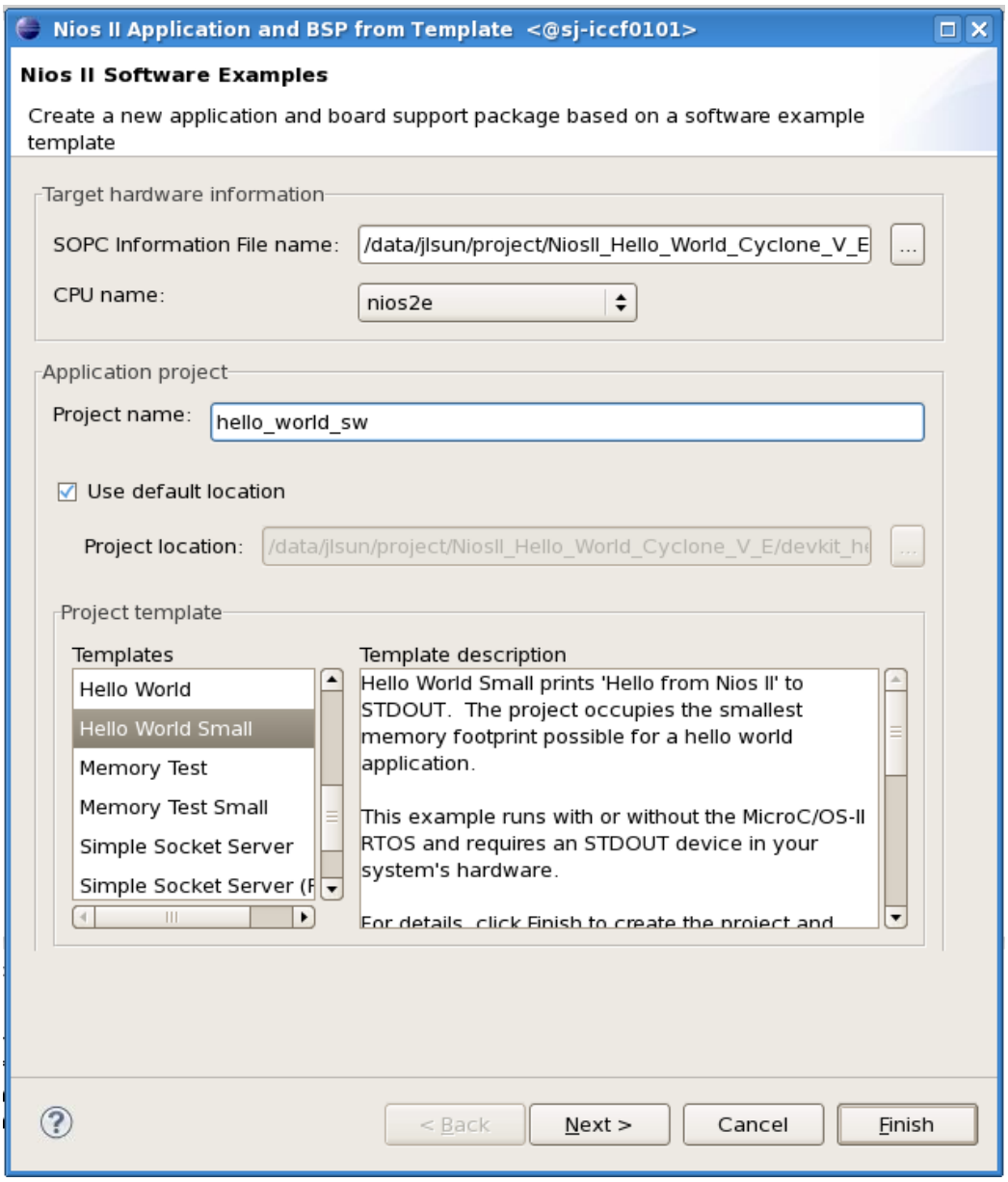

Figure 47: Completing Setup with Project Name and Template

We will now make some modifications to the code to connect the LEDs to the Push Buttons through software. Click the right arrow next to hello_world_sw. It will show the contents of your project. Double-click hello_world_small.c to inspect the code. The command alt_putstr is used to write text to the terminal. This is a function of the Altera HAL. A complete list of these functions can be found in the Nios II Software Developer's Handbook https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2.pdf .

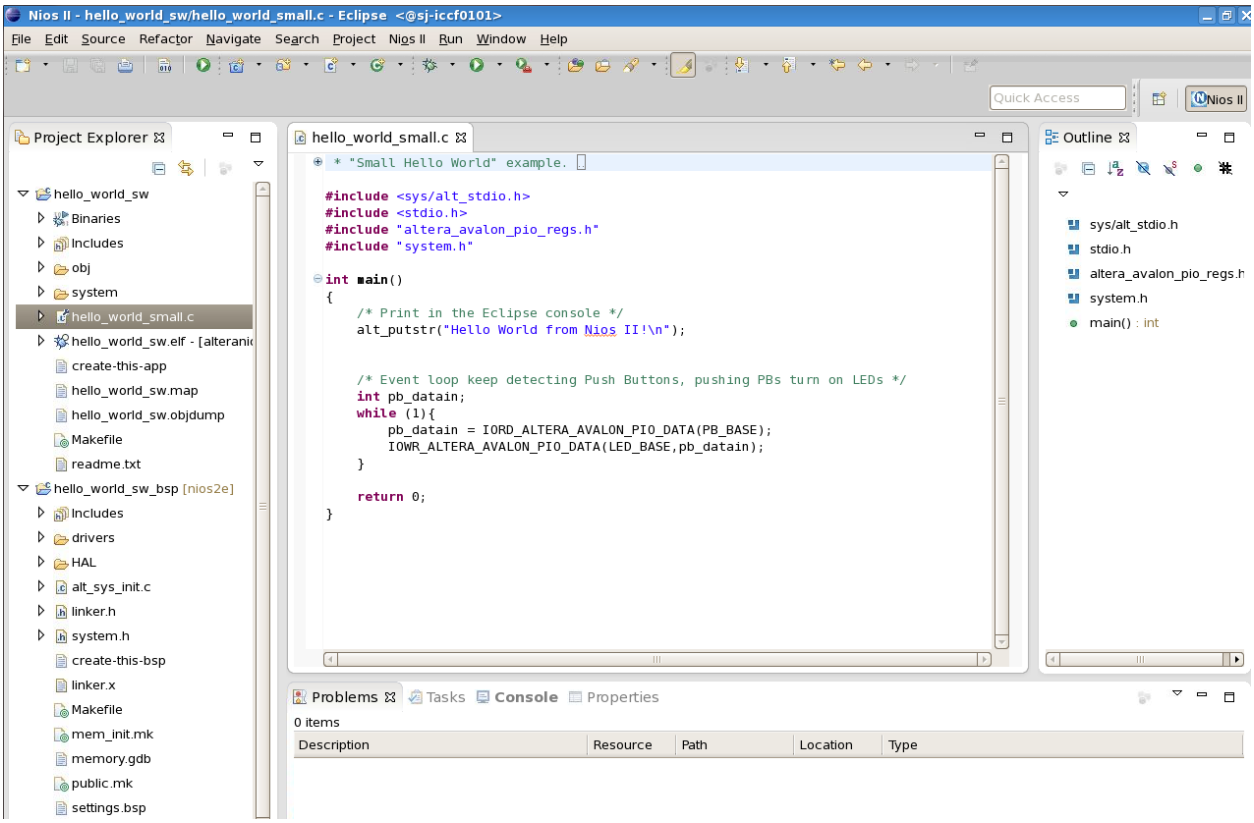

Figure 48: Code Change in Eclipse SBT

Next you need to add a library declaration, define integer pb_datain, and a few HAL functions to connect the LEDs to the Push Buttons. Edit the code as shown in Figure 48. You can copy the code from here:

```
#include <sys/alt_stdio.h>
#include <stdio.h>
#include "altera_avalon_pio_regs.h"
#include "system.h"

int main()
{
  /* Print in the Eclipse console */
  alt_putstr("Hello World from Nios II!\n");


  /* Event loop keep detecting Push Buttons, pushing PBs turn on LEDs */
  int pb_datain;
  while (1){
    pb_datain = IORD_ALTERA_AVALON_PIO_DATA(PB_BASE);
    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, pb_datain);
  }

  return 0;
}
```

Note the use of the variables PB_BASE and LED_BASE. These variables are created by importing the information from the .sopcinfo file. You can find defined variables in the system.h file under the hello_world_sw_bsp project. Double click on system.h file and inspect the defined variable names for PB_BASE and LED_BASE. These must match your hello_world_small.c code.

Save the file by clicking the  icon. Right click on the hello_world_sw project then select Build. This compiles the software application and the BSP (drivers).

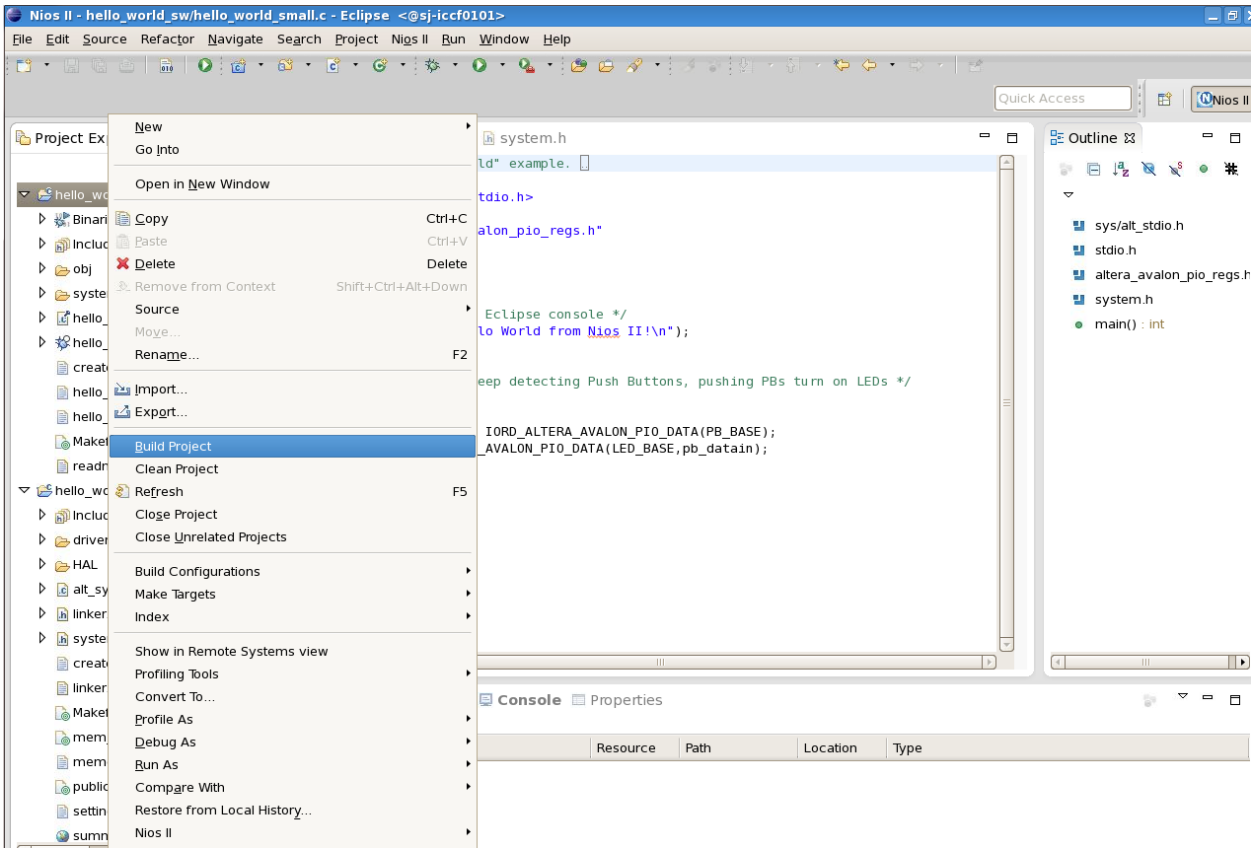

Figure 49: Launching the Build

Once the build completes, you should observe a ".elf" file (executable load file) under the hello_world_sw project, as shown in Figure 50. If the .elf file does not exist, the project did not build properly. Inspect the problems tab on the bottom of the Eclipse SBT and determine if there are syntax problems. If any, correct them and rerun Build Project. Typical problems can be missing semicolons, mismatched brackets and such.

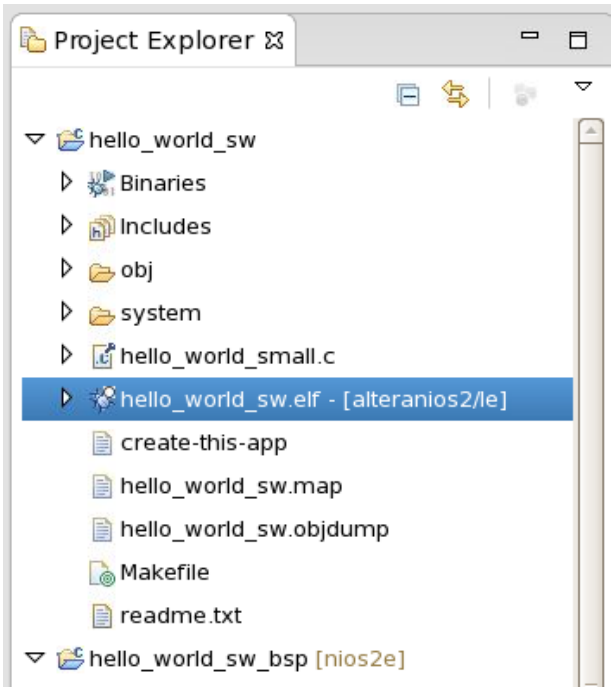

Figure 50: hello_world_sw.elf Generated by Eclipse after Build Success

Downloading the Hardware Image

To work with the Cyclone V E FPGA Development Kit in the context of this lab, you will need to connect the power supply to the DC Input and a USB cable connecting the kit to a host PC.

Make sure your development kit is powered up (there is a switch near the Power Jack) and LEDs are on.

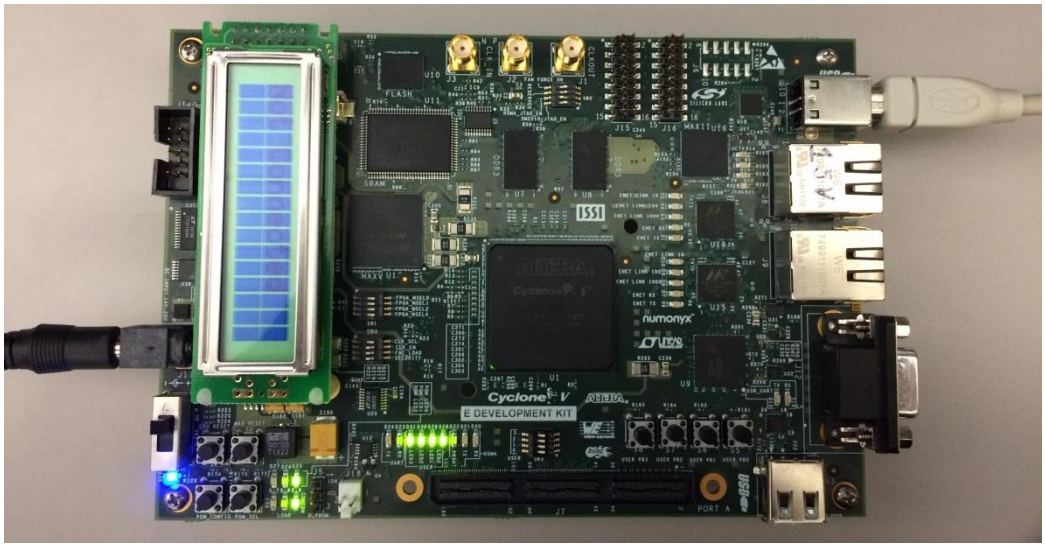

Figure 51: Cyclone V E FPGA Dev Kit Setup

If you are only performing the Software design lab, you must first launch Quartus. If you have performed the Hardware Design lab, then Quartus should already be open.

In Quartus II, launch the Programmer: Tools → Programmer.

Click Auto Detect and you should see something similar to Figure 52.

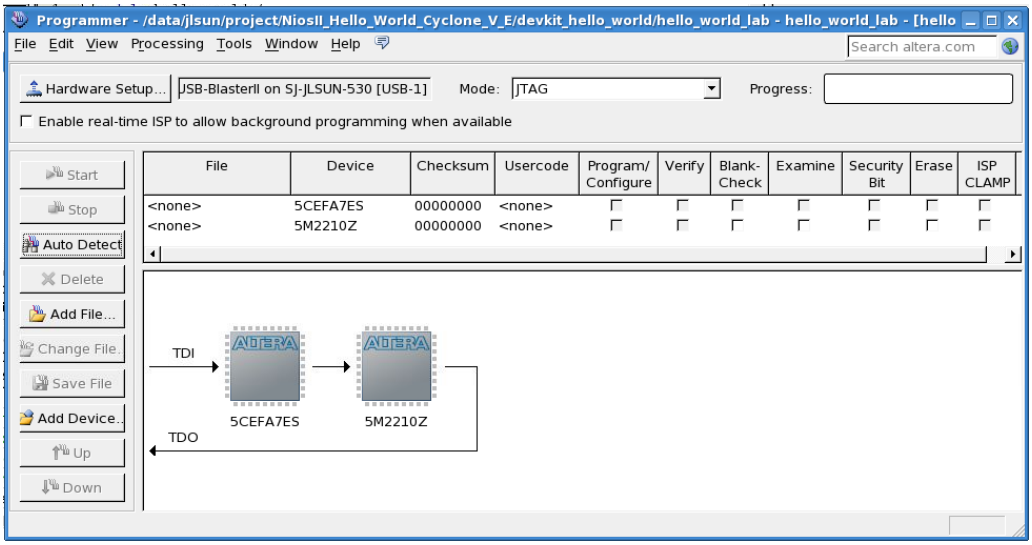

Figure 52: Programmer after Auto Detect

Next, you need to download the ".sof" (SRAM Object File). This is the programming image file that gets downloaded in the FPGA. The default location is <working_directory>/output_files. Right click on the first row <none> under File and click on Change File. Navigate to the output_files directory and select "hello_world.sof". Click Open. In the first row under Program/Configure click in the check box as shown in Figure 53.

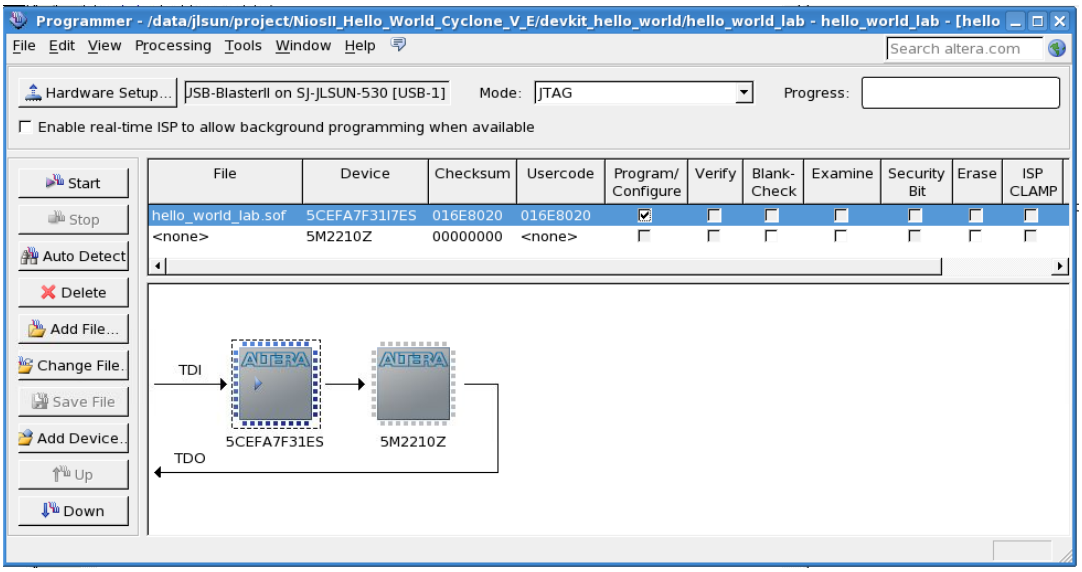

Figure 53: Programmer after Adding hello_world.sof File

Click Start. When programming is complete, the Progress meter should read 100% (Successful).

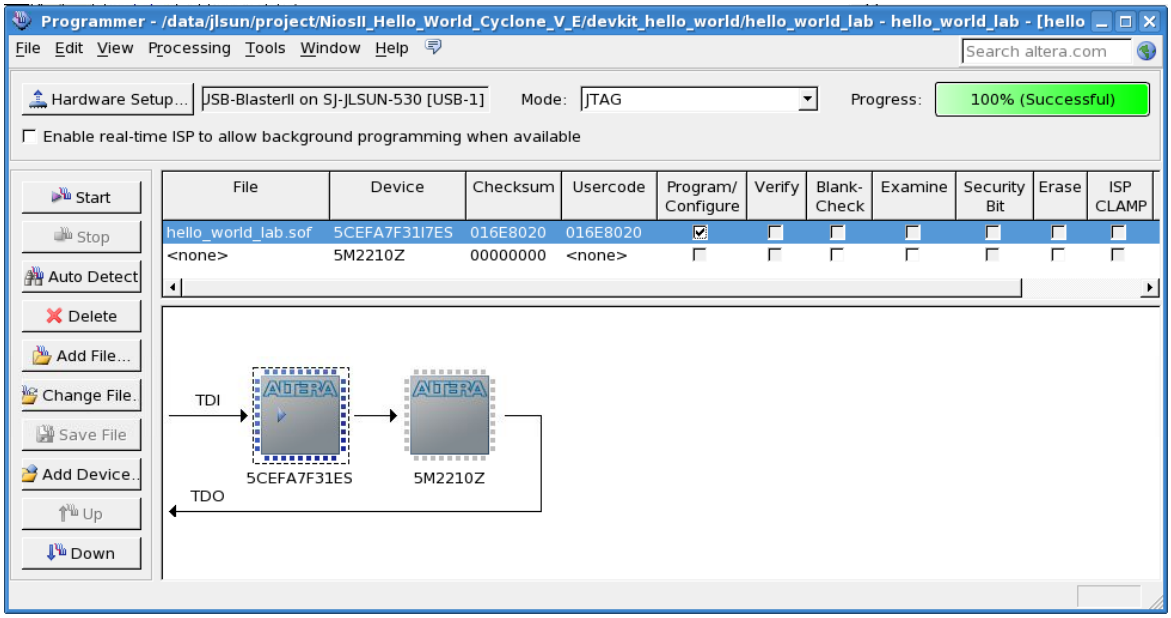

Figure 54: Programmer after Completing .sof Download

Now you can try to push PB1 and PB0 to see if they can turn off LED1 and LED0 respectively. If so, you downloaded your hardware image successfully.

Running Software

Now it is time to run the .elf (software executable) in the Nios II e processor. Return to the Eclipse SBT. Right click on "hello_world_sw.elf" and select Run as → Run Nios II Hardware.

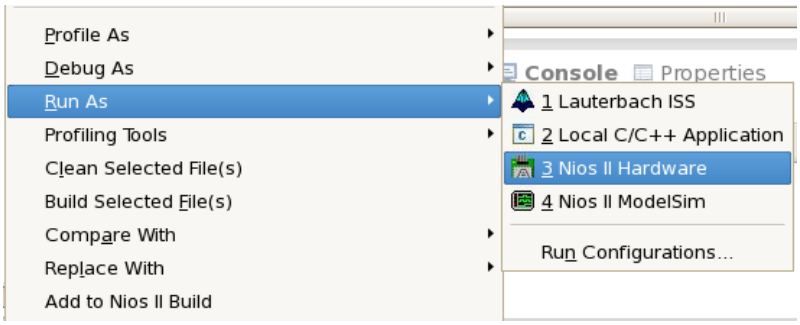

Figure 55: Run .elf File as Nios II Hardware

Click the "Target Connection" tab. The connection should indicate that Eclipse has connected to USB-blaster. If the connection is not identified, you can Click Refresh Connections. Note that you might need to stretch the window wider to see the Refresh Connections button. Also remember to check the two options in System ID checks (only for this lab). You should observe something similar to **Error! Reference source not found.**. Click Run.

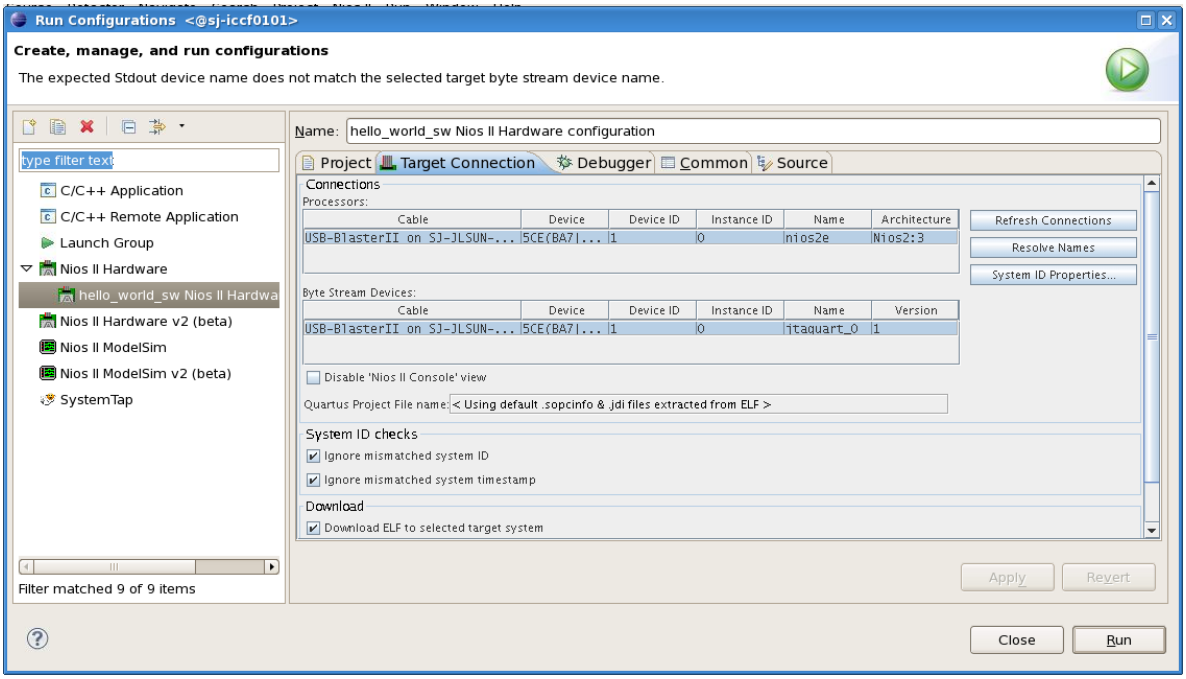

Figure 56: Run Configurations Window in Eclipse.

Now you have the software running in the Nios II processor you downloaded to the hardware. You should observe “Hello World from Nios II” in the Nios II Console tab.

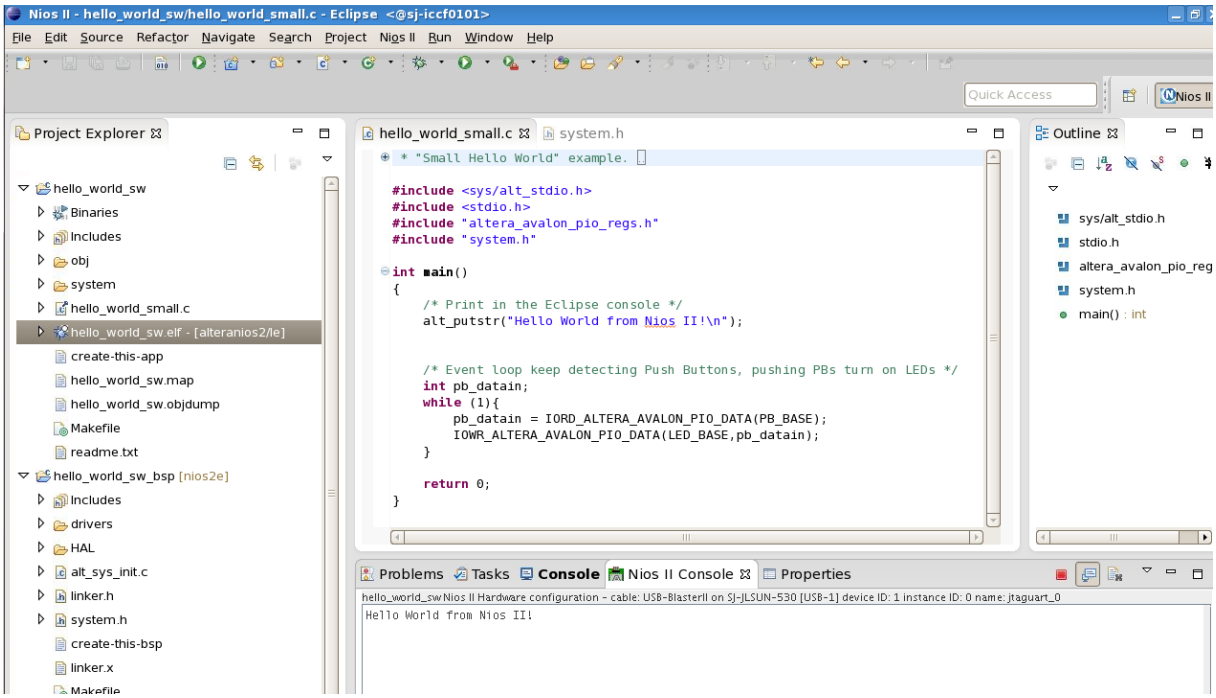

Figure 57: “Hello World from Nios II!” Displayed on the Nios II Console

Also you will notice that LED3 and LED2 are now off. You can push PB3 and PB2 to turn them on respectively. This is achieved in the C code we built in Eclipse SBT. Recall that LED [1:0] and PB [1:0] are connected in inverted manner in hardware so by default LED1 and LED0 are on. This is shown in Figure 58.

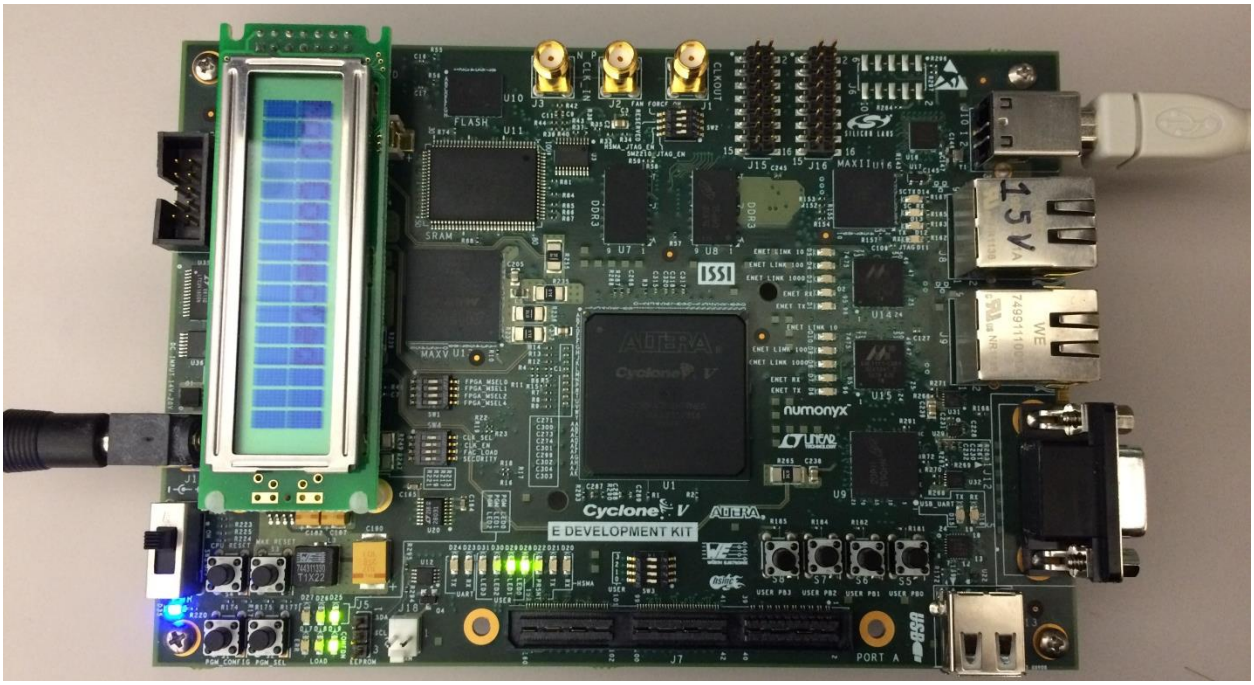

Figure 58: Final Hardware Display of the lab

Lab Summary

You now have completed the hardware and software sections of this lab. This includes:

1. Loading the Device Kit pin settings into Quartus
2. Using Qsys to build a Nios II based system
3. Instantiating the Qsys component into your top level design
4. Add some connections between push buttons and LEDs
5. Compiling your hardware
6. Importing the Nios II based system into the Eclipse Software Build Tools
7. Building a software project
8. Modifying a software template to perform some simple IO functions
9. Compiling your software
10. Downloading the hardware image into the Cyclone V E Dev Kit
11. Running the software executable in the Cyclone V E Dev Kit
12. Testing the hardware

There is a wealth of resources from Altera and partners to take classes on Embedded Hardware, Embedded Software and reference design starting points to advance your skills using Altera's powerful Nios II based hardware and software tools.

Appendix A: Using Schematic Capture in Place of Writing Verilog for the top Level Module

In the section Building the Top Level Design, we showed you how to create the top level design that instantiates the Nios II Qsys system by typing in the Verilog description of the connections. An alternative means to make these connections is by using the schematic capture tool within Quartus. The two means to create the top design: Verilog edits and schematic capture are generally a matter of preference and familiarity with the Verilog language. The industry trends toward editing Verilog or VHDL hardware description languages, but either means will generate a working FPGA design.

In this section, we will show you how to build the top-level design using schematics. This is just as an alternative way of finishing the section Building the Top Level Design. You can follow all steps in other sections in this user guide to complete this lab.

Launch the schematic capture tool by invoking File → New → Block Diagram/Schematic File You will see a blank schematic as shown in Figure 59.

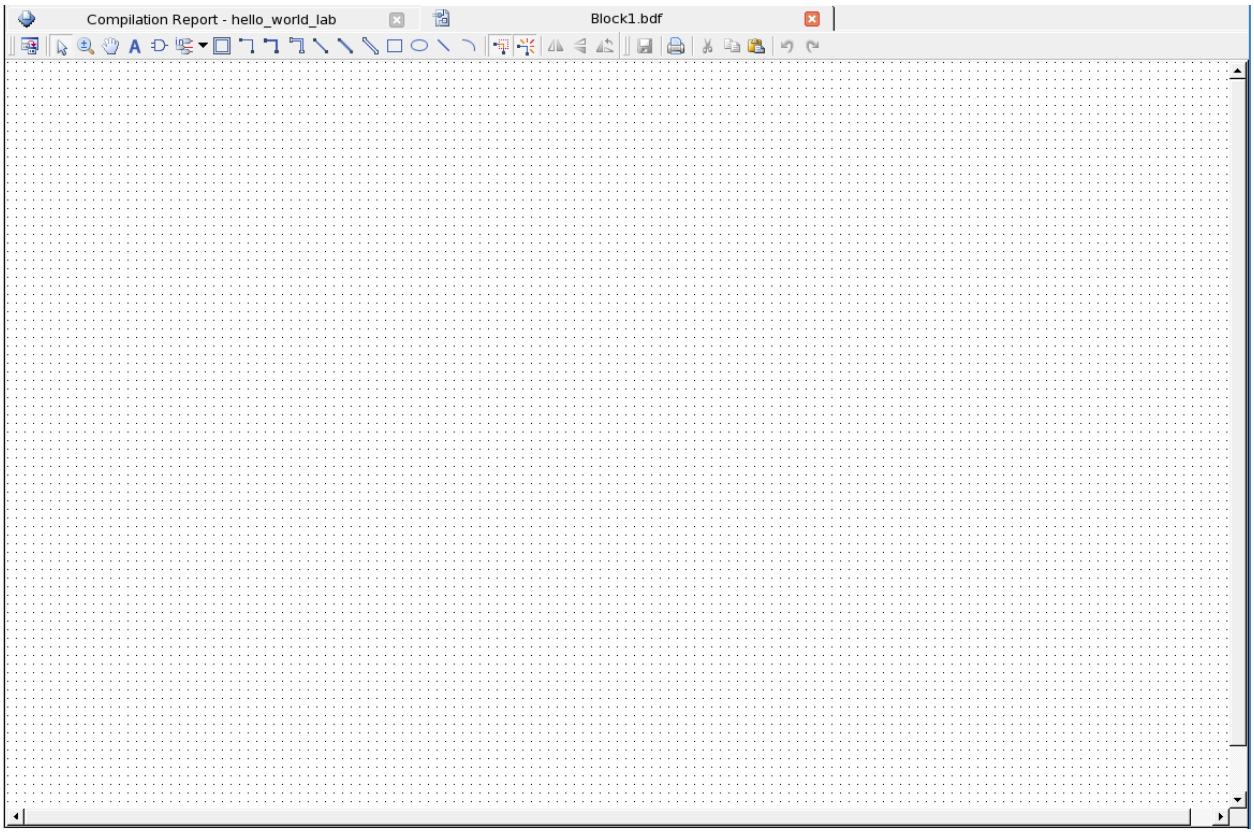

Figure 59: Blank Schematic Page

Next you will need to add the Qsys system block. Right click and invoke insert → symbol. Under name, navigate to working directory/nios2e and select nios2e.bsf. Click Open. Then click OK.

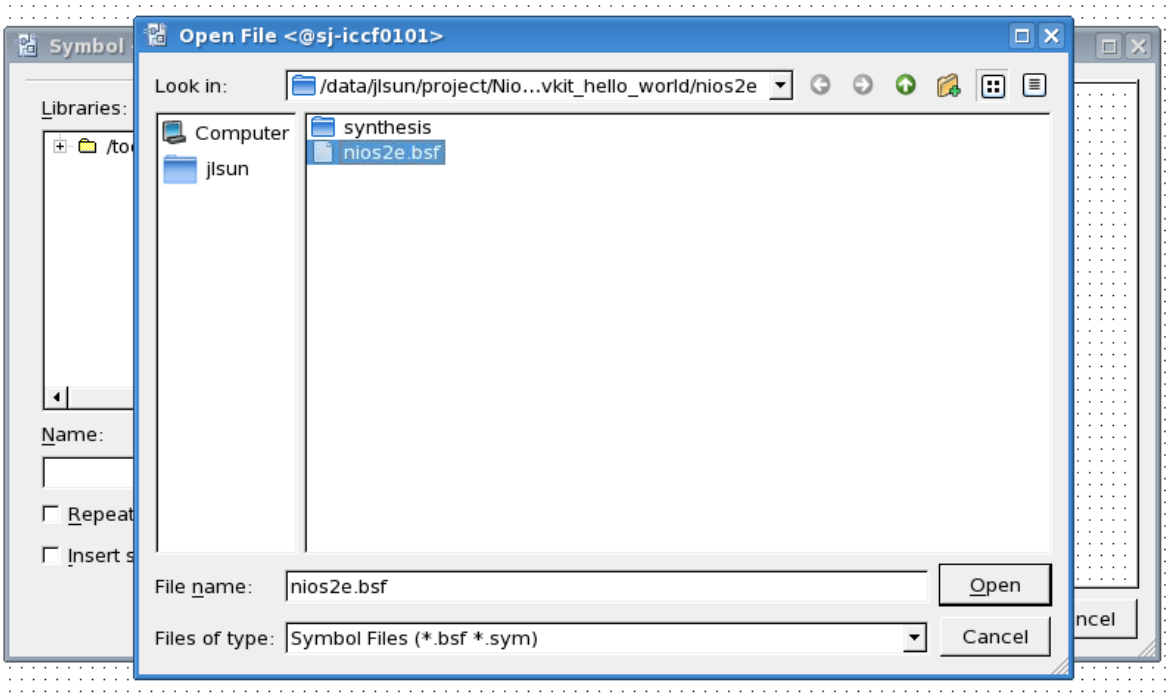

Figure 60: Adding the nios2e Symbol into Schematic

Click in the middle of the screen to drop the symbol in the schematic. This time you will be adding the inversion between the USER_PB [1:0] and USER_LED [1:0]. Right click and invoke insert → symbol. In libraries, expand and navigate to primitives → logic → not and highlight. Click OK. Drop the NOT symbol below the nios2e symbol.

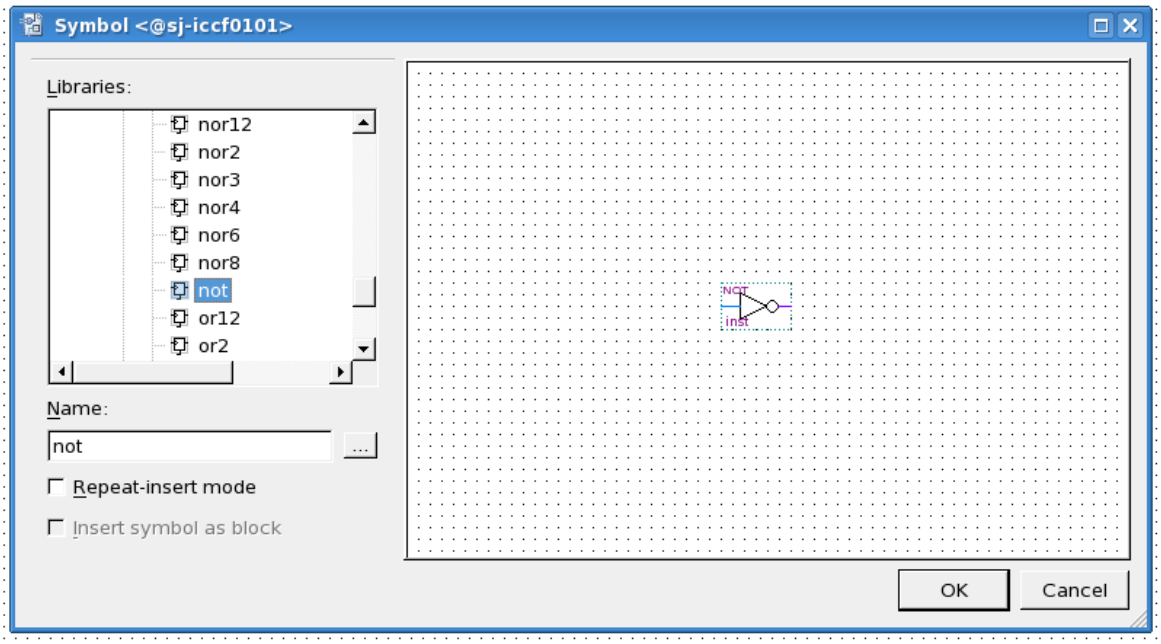

Figure 61: Adding the "not" logic Symbol

Now you need to make the connections to the schematic. Locate the pin icon  and select input. Snap the input to clock, reset and switch on the nios2e and to the input of the not. Hit escape. Select the output pin and snap to the output of not and drop another one to the left of the nios2e symbol without connecting it since it's pointing in the wrong direction. Hit escape. Right click the output pin near the nios2e and Flip Horizontal. Left click the output pin and snap it to the led_external_connection_export port on nios2e.

Now you need to determine the appropriate names for the pin connections. Open Assignment Editor, locate the names of the signals for reference: "clkin_50_fpga_top", "cpu_resetn", "user_pb", and "user_led". Right click on the pins in your schematic and invoke properties and change the pin names to the appropriate top-level-port names. You can also do this by double-clicking on the port. Make sure the names are identical to the ones in Assignment Editor. Note that bus notation is of the form "name[3..0]". Once you have assigned the port names, you have completed the schematic.

Click save as and enter "hello_world_schematics.bdf". Make sure the name of this file is not the same as the .v file if you have created one before. This is because that Quartus can't compile duplicate declarations of entity. The completed schematic is shown in Figure 62.

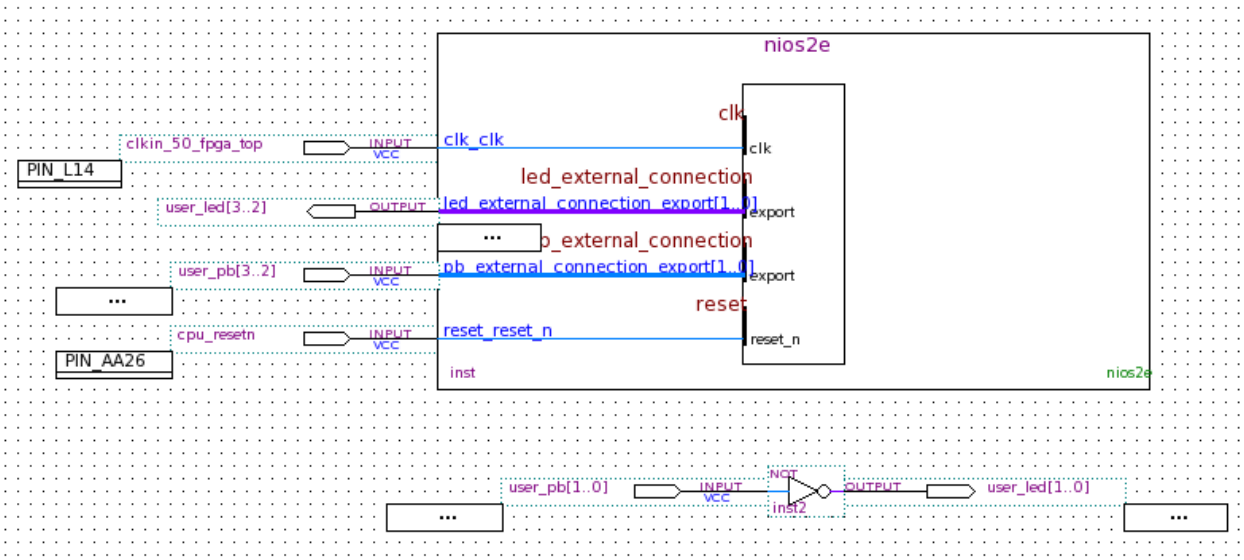

Figure 62: Completed Schematic Design of This Lab

Now you need to add the hello_world_schematics.bdf to your project by clicking Project → Add/Remove Files in Project, if it's not already there. Set it as the top-level entity. Now you are ready to compile your design as we discussed before.

Appendix B: Merging the Nios Executable into the FPGA Configuration File (Hardware Image)

In prior steps, you downloaded the hardware image first and then you ran the nios executable from Eclipse SBT. In this section, we will guide you with how to merge the software executable file (.elf file) into hardware image (.sof file). This gives you the .sof with the nios executable incorporated in it so you only need to download it to the FPGA to get the complete function.

Once your software is stable in Eclipse, right click on the project folder or the .elf file and select Make Target → Build. A window pops up. Select mem_init_generate followed by Build. This generates a .hex file that is in the location where your software build is located (e.g.):

./Hello_World_Dev_kit/software/hello_world_sw/mem_init/nios2e_onchip_memory.hex

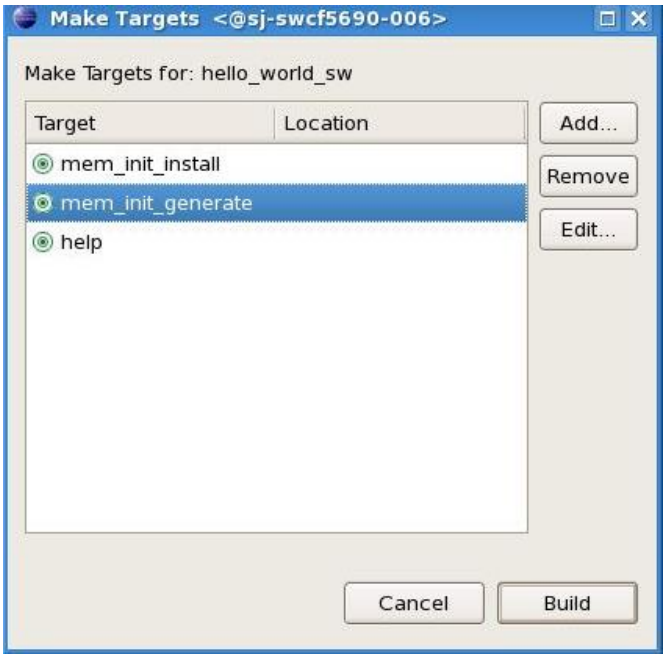

Figure 63: Make Targets for nios Executable

Next, you need to return to Qsys and change the onchip_memory component by double clicking it. Check all the boxes in “Memory Initialization” and specify the “User created initialization file”. to the “nios2e_onchip_memory.hex” file we just generated from Eclipse.

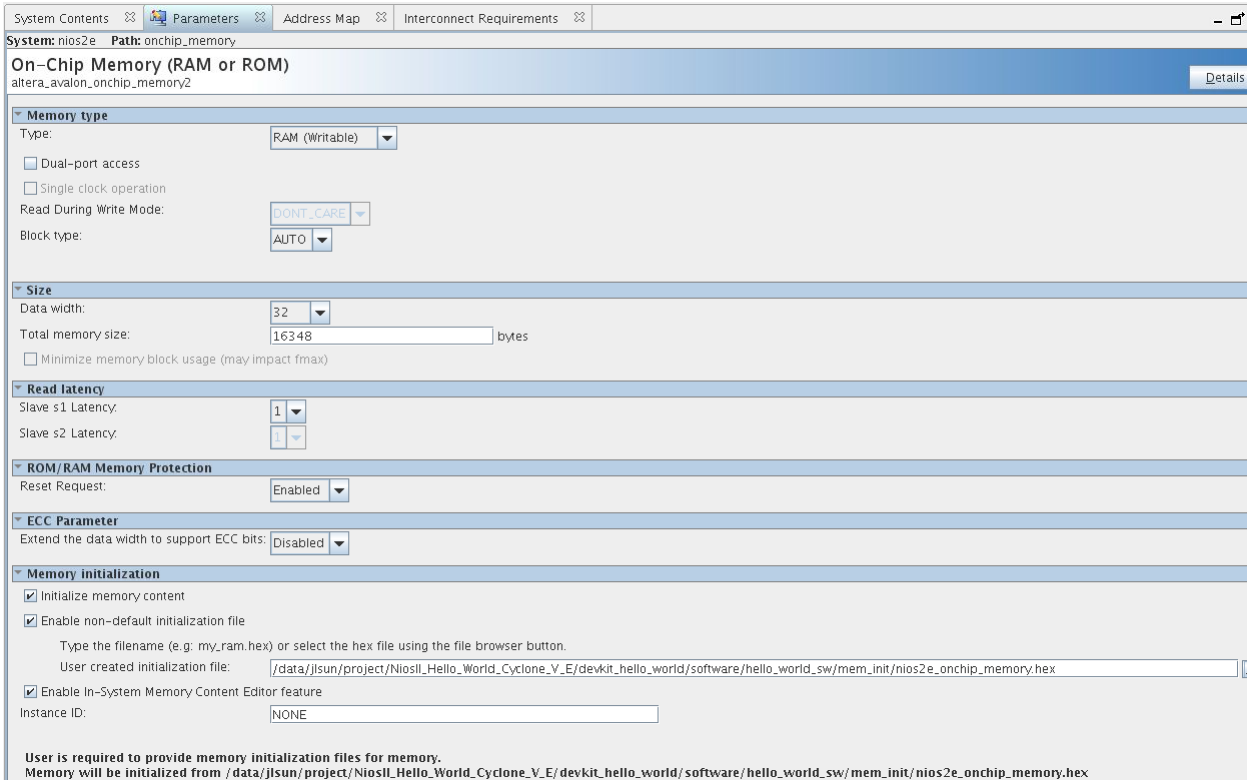

Figure 64: On-chip Memory Initialization

Save Qsys, click Generate HDL →Generate. Return to Quartus and compile the design. After compilation, a new .sof file with the nios executable included will be genrated. Run the programmer and download the new .sof file.

If the .sof file with software executables is downloaded successfully, you can see the same behavior as before. You can push PB [1:0] to turn off LED [1:0] and push PB [3:2] to turn on LED [3:2].

Appendix C: Using Interrupt Service Routines (ISR) in a Nios Based System

Theory of Operation

The hello_world_small.c we built previously uses a polling technique to monitor the values of the push buttons. An infinite while(1) loop is used and each time through the loop the value of the push buttons are monitored. Another way to monitor events is through the use of interrupts.

The theory of operation of how hardware interrupts work is shown in Figure 65. Interrupts gain priority over any code such as while loops, for loops etc. which are executing. This appendix explains how to implement hardware interrupts using Nios II processor and hardware abstraction layer (HAL) functions.

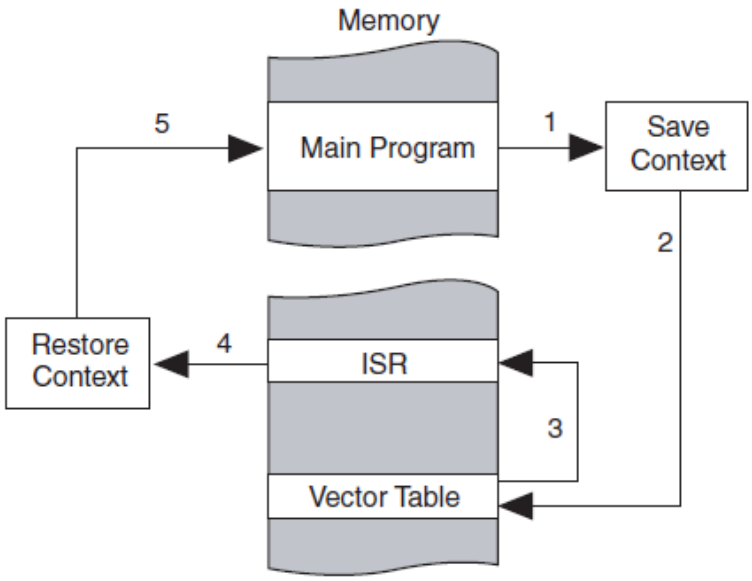

Figure 65: Interrupt Architecture

Software registers an Interrupt Service Routine (ISR) using the alt_ic_isr_register() HAL function. The HAL registers the ISR by configuring the vector table. The return code from alt_ic_isr_register() is zero if the function succeeds, and nonzero if it fails. If the HAL registers an ISR successfully, the associated Nios II hardware interrupt (as defined by irq) is enabled on return from alt_ic_isr_register().The function and its main components are as follows:

```
alt_ic_isr_register(SWITCH_IRQ_INTERRUPT_CONTROLLER_ID, SWITCH_IRQ,
  handle_button_interrupts, edge_capture_ptr, 0x0);
```

1. SWITCH_IRQ_INTERRUPT_CONTROLLER_ID – Interrupt controller ID (refer to system.h)
2. SWITCH_IRQ – IRQ number (refer to system.h)
3. handle_button_interrupts(ISR) – Pointer to ISR function
4. edge_capture_ptr – Argument passed to the ISR
5. 0x0 – Flag (reserved)

The ISR reads the value of the push buttons and updates the pointer to the variable edge_capture. The pointer edge_capture_ptr reads the new value on the push buttons. For more details about alt_ic_isr_register() and its parameters, please refer Exception Handling, Nios II Software Developer's Handbook - Altera.

Compile Hardware and Software to Run the Interrupt Driven “Hello World” Design

- The qsys system needs a few changes to enable interrupts.
- Increase the memory size by double clicking on onchip_memory to update the total memory size to 131136.
- Double click on pb, enable “Edge capture register” and “Interrupt” and have the type set to RISING and EDGE respectively for them.

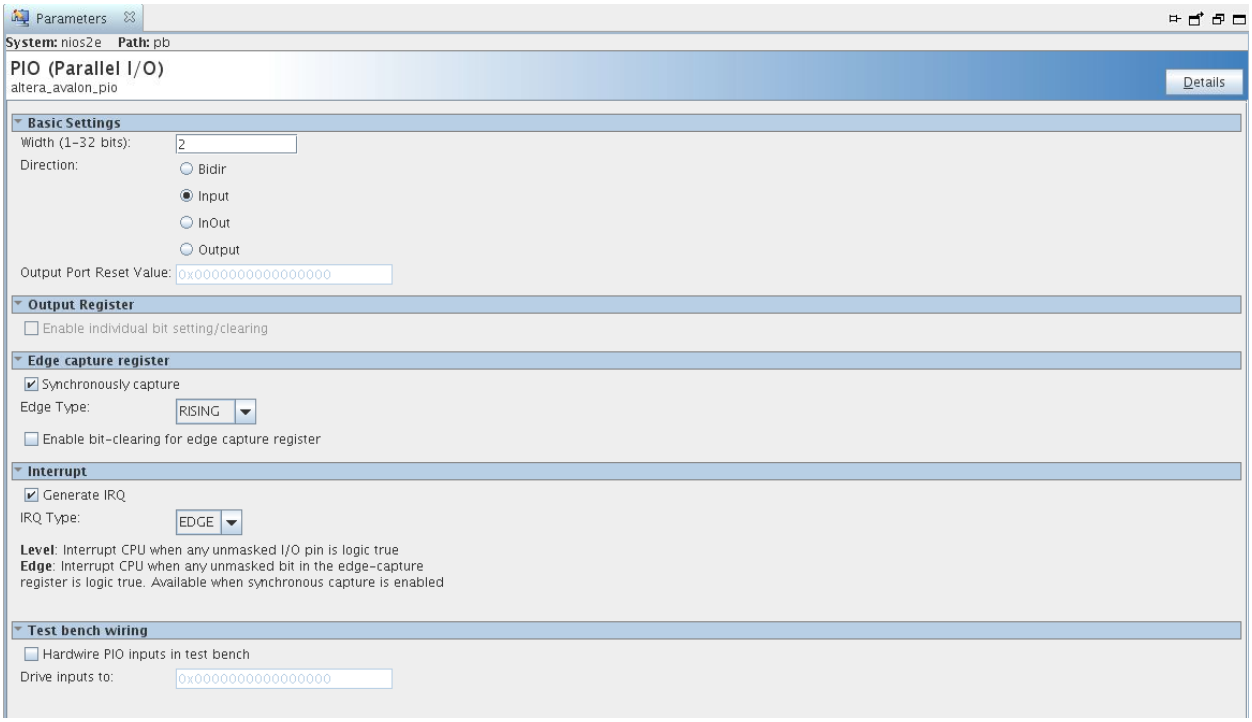

Figure 66: Changing Parameter Setting for pb Component

- To get rid of the errors, go to System → Assign Base Addresses.
- To get rid of the warnings, you need to connect the irq pin under the pb component to the irq from nios2e component. To do this, click on the dot next to irq.

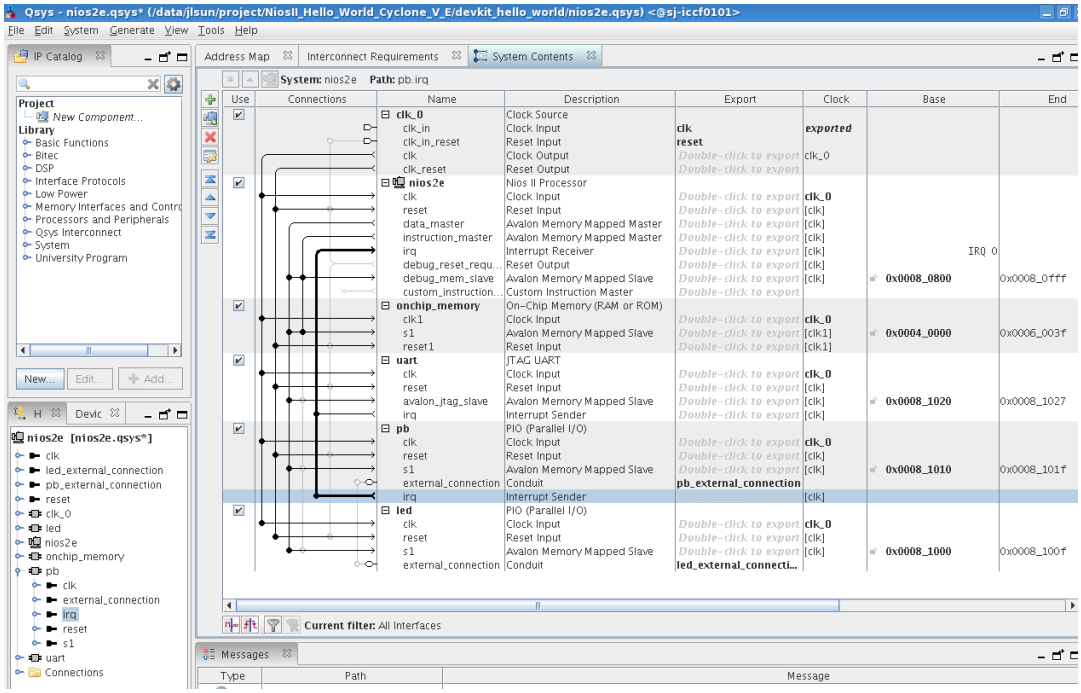

Figure 67: IRQ Connection of pb Component

- Scroll to the right in System Contents and change the IRQ number for uart and pb respectively in the IRQ Column. The values can be updated by double clicking on them. The final setting should be as follows:

Name	Description	Export	Clock	Base	End	IRQ
clk_0	Clock Source					
clk_in	Clock Input	**clk**	***exported***			
clk_in_reset	Reset Input	**reset**				
clk	Clock Output	*Double-click to export*	clk_0			
clk_reset	Reset Output	*Double-click to export*				
nios2e	Nios II Processor					
clk	Clock Input	*Double-click to export*	**clk_0**			
reset	Reset Input	*Double-click to export*	[clk]			
data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31	
debug_reset_requ...	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...	Custom Instruction Master	*Double-click to export*				
onchip_memory	On-Chip Memory (RAM or ROM)					
clk1	Clock Input	*Double-click to export*	**clk_0**			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	**0x0004_0000**	0x0006_003f	
reset1	Reset Input	*Double-click to export*	[clk1]			
uart	JTAG UART					
clk	Clock Input	*Double-click to export*	**clk_0**			
reset	Reset Input	*Double-click to export*	[clk]			
avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0008_1020**	0x0008_1027	
irq	Interrupt Sender	*Double-click to export*	[clk]			1
pb	PIO (Parallel I/O)					
clk	Clock Input	*Double-click to export*	**clk_0**			
reset	Reset Input	*Double-click to export*	[clk]			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0008_1010**	0x0008_101f	
external_connection	Conduit	**pb_external_connection**				
irq	Interrupt Sender	*Double-click to export*	[clk]			0
led	PIO (Parallel I/O)					
clk	Clock Input	*Double-click to export*	**clk_0**			
reset	Reset Input	*Double-click to export*	[clk]			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0008_1000**	0x0008_100f	
external_connection	Conduit	**led_external_connecti...**				

Figure 68: IRQ Number Setting

- Create a new folder and name it as Interrupt, go to File → Save as and save Qsys in the Interrupt folder.
- Generate the HDL files again, which would update all the qsys files with the interrupt settings.
- The new .qip file needs to be included in the project and the older .qip file has to be deleted.
- Compile the design to generate the new .sof and .sopcinfo files.
- Repeat the steps mentioned in software section to create a new project from “hello world small template” using the new .sopcinfo file, call it hello_world_interrupt.
- Delete the hello_world_small.c file by right clicking on it from the project explorer and selecting Delete.
- Import the .c file with the hardware interrupt logic by right clicking on the folder in the project explorer and select import. The following window comes up:

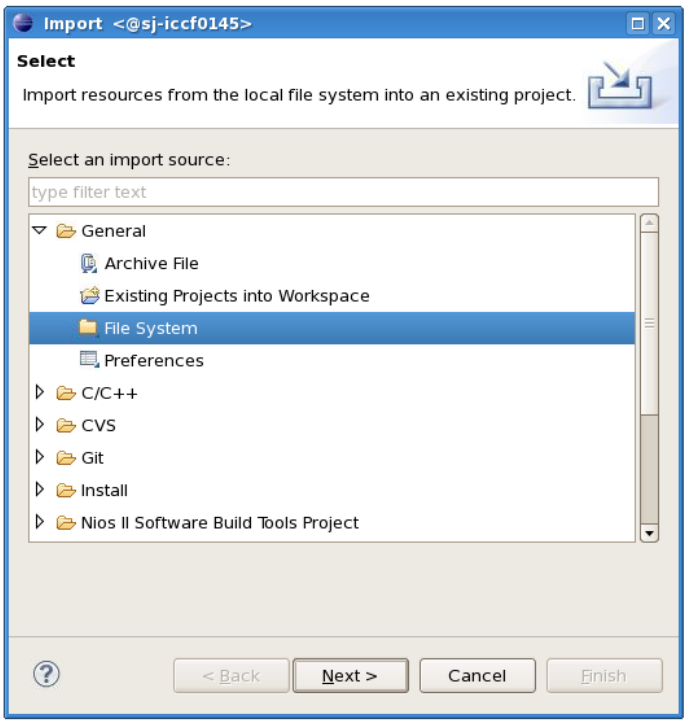

Figure 69: Eclipse Imports New Files for Interrupt Driven Software

- Hit next and browse to the “software” folder in the lab resource you downloaded from Design Store, select the “hello_world_hardware_interrupt.c” and “board_diag.h” files, followed by Finish. This imports both the files.
- Right click on the hello_world_interrupt_bsp folder and go to Nios II → BSP Editor and uncheck the enable_small_c_library. Hit generate to save the changes, as shown in Figure 70.
- Build the design by right clicking on the project folder (hello_world_interrupt) and clicking on Build Project. Make sure there is no error and the .elf file is generated.

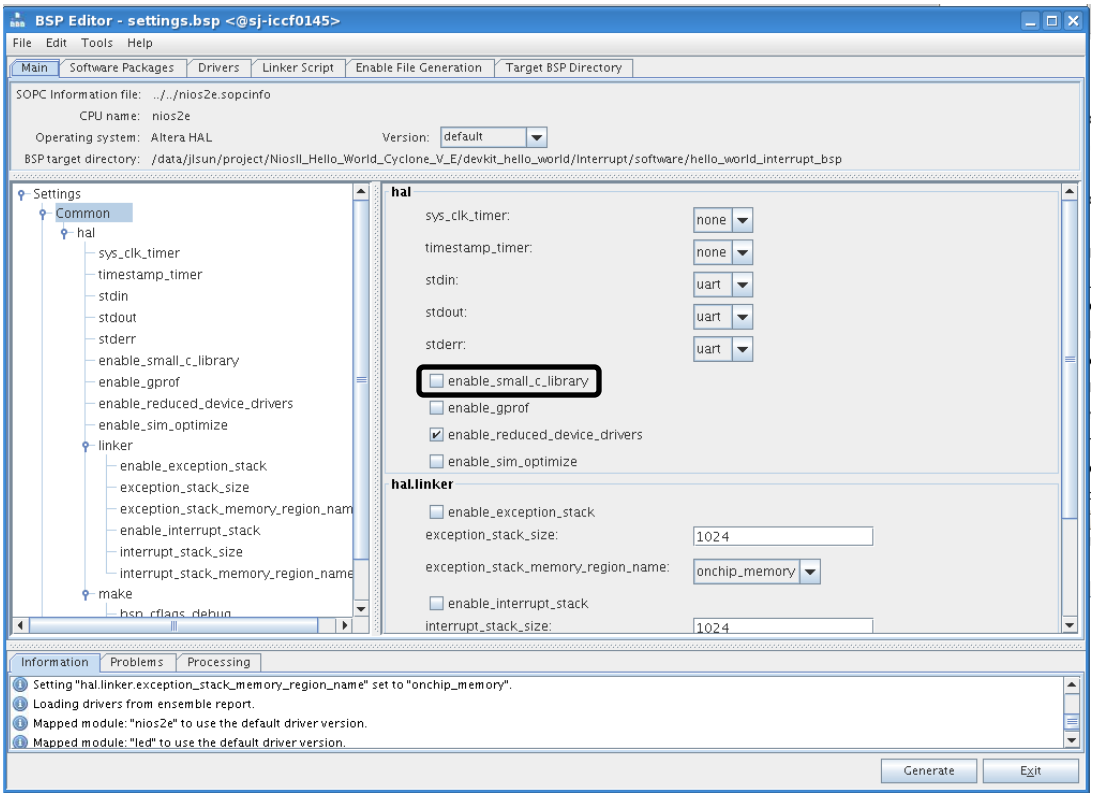

Figure 70: Uncheck enable_small_c_Library

- In Quartus, launch Programmer and program the board with the new .sof file with the changes in Qsys for interrupts.
- In Eclipse, run the .elf file as introduced before.

Design Demonstration

After running the executable successfully, you will see a welcome message in the Nios II Console along with the main menu. You can follow the instruction to explore this example. The menu is displayed by calling the functions TopMenu, MenuBegin and MenuEnd. It gives the user an option to select between the modes of operation. The input from the user is read using the function GetInputString. The console displays it as:

```
Hello from Nios II!


---------------------------------
 Select mode of operation
---------------------------------
 Main Menu
     a:  LED 2 Blinks continuously waiting for hardware interrupt on PB3
     b:  LEDS 2 and 3 blink number of times entered by user
     q:  Exit
---------------------------------

Select Choice (a or b): [Followed by <enter>]
```

a: Blink continuously waiting for hardware interrupt

If you choose mode a, you will get the following message:

```
*****************************************************

 To start blinking of the LED.
 Press push button 2

*****************************************************
```

Push PB2 and you will see LED2 start blinking continuously. To generate an interrupt, press PB3, then the following message will be displayed. Now the LED2 will stop blinking.

```
**************HARDWARE INTERRUPT WAS OBSERVED ****************

Button 3 (PB3) Pressed.

 To resume blinking press push button 2 again
            OR
 To exit press push button 3 again
```

If you wish to continue blinking you can push PB2 or you can exit by press PB3.

b: Blink number of times entered by user

After exiting from the previous mode, you will see the main menu again. Now you can try mode b by type “b” followed by enter. In this mode of operation the GetInputString() function reads a number in the range 0 to 9 entered by the user. The LED2 and LED3 will blink for the number of times specified by the user and then exit the loop. The image below shows the messages you will observe in this mode:

```
Select Choice (a or b): [Followed by <enter>]
b

Enter a number between 1 to 9 (followed by <enter>)
5
The leds will blink 5 times and then exit out of the loop


*****************************************************

 Exiting out of the loop.
 Done blinking 5 times.

*****************************************************
```

Appendix D: NiosII Hello World Lab with LCD Character Display

In this section, we will show you how to add a LCD module to the system based on your previous Nios II Hello World Design. This demo enables display on a 2 × 16 LCD using parallel data interface to system board. You can download the completed design here from Altera Cloud Design Store and launch the project in Quartus II, as described in the "Initial Setup" section. Name the project and top-level entity as "NiosII_Hello_World_C5e".

Cyclone V E FPGA Development Kit includes a single 14-pin 0.1" pitch dual-row header that interfaces to a 2 × 16 Lumex character LCD. We will use a technique called "Bit Banging", which is a way of achieving serial communication in software. We'll delve into more details later. Set up this kit as shown in Figure 51.

After launching the Quartus II project, double click on the name of the top-level entity to open the "NiosII_Hello_World_C5e.v" file.

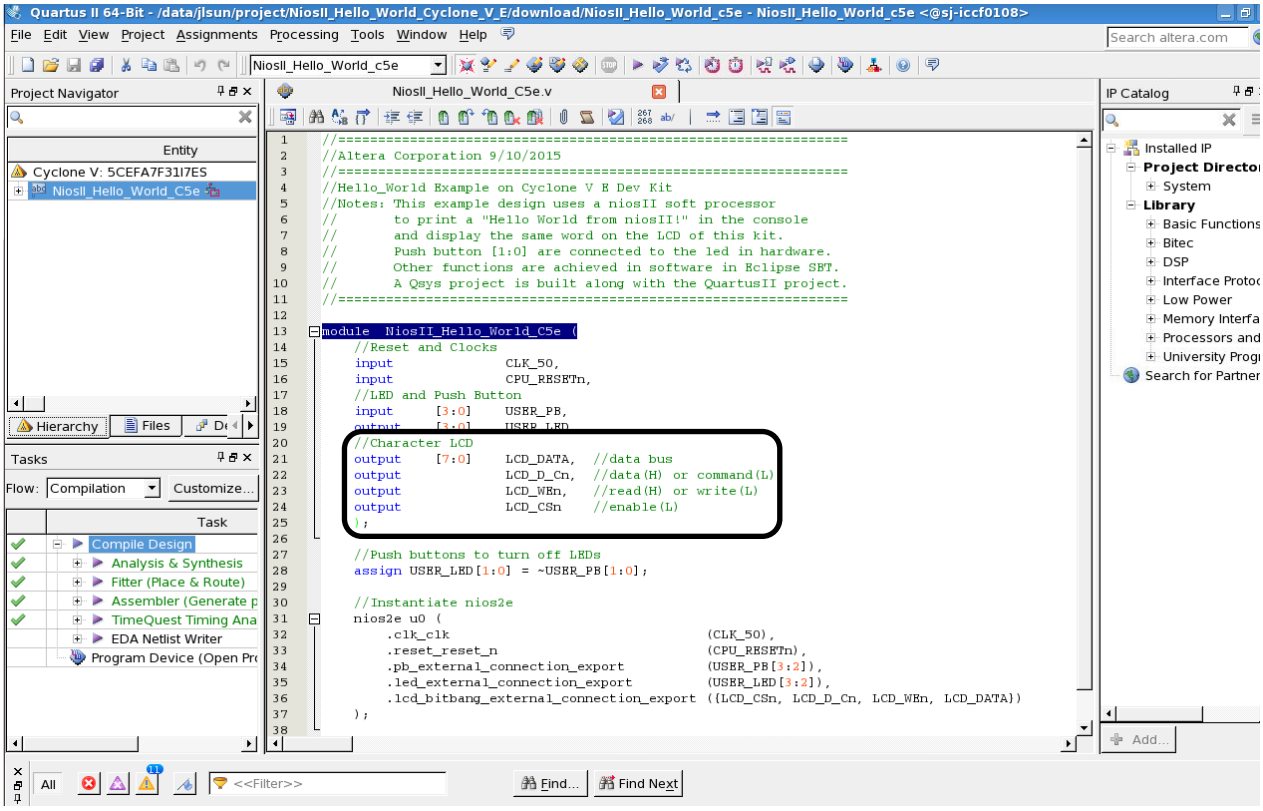

Figure 71: Quartus II Project

Compare with the previous top-level design, here we added more output ports for LCD, as highlighted in Figure 71. The following table lists the LCD pin definitions, and is an excerpt from

the Cyclone V E Dev Kit reference manual. The three power supply pins are wired to the power supply on the board. From the perspective of FPGA, the “LCD_DATA” (DB0-DB7) is the 8-bit bus used to output data to the LCD. “LCD_D_Cn” (RS) is used to select whether we are going to output data or command. “LCD_WEn” (R/W) is used to select whether we want to read data from LCD or write data to LCD. Lastly, “LCD_CSn” (E) is used to enable LCD.

Pin Number	Symbol	Level	Function	
1	V_{DD}	—	Power supply	5 V
2	V_{SS}	—		GND (0 V)
3	V_0	—		For LCD drive
4	RS	H/L	Register select signal H: Data input L: Instruction input	
5	R/W	H/L	H: Data read (module to MPU) L: Data write (MPU to module)	
6	E	H, H to L	Enable	
7–14	DB0–DB7	H/L	Data bus—software selectable 4-bit or 8-bit mode	

Now we can look at the changes in the Qsys system. Launch Qsys from QuartusII: Tools → Qsys. Open the project “nios2e.qsys”, as shown in Figure 72.

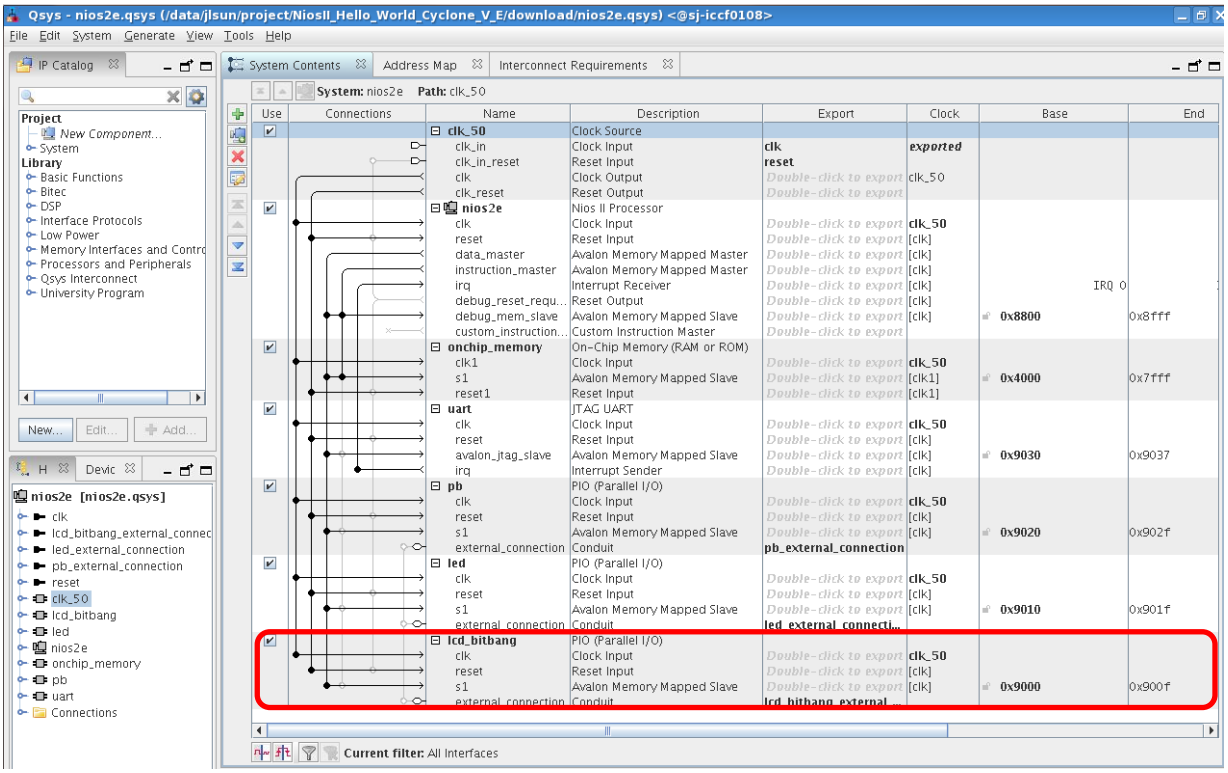

Figure 72: Qsys Project

Compare with the previous Qsys design, the only change we made here is that we added another 11-bit output PIO component (name it lcd_bitbang). The reason it is 11 bits is because we need to export both the data ports (8 bits) and the control ports (3 bit) so we can implement the Bit Banging in software. The following Verilog code (NiosII_Hello_World_C5e.v) shows the instantiation of the Qsys system. The highest 3 bits are connected to LCD_CSn, LCD_D_Cn and LCD_Wen. The lowest 8 bits are connected to LCD_DATA.

```
//Reset and Clocks
input              CLK_50,
input              CPU_RESETn,
//LED and Push Button
input      [3:0]   USER_PB,
output     [3:0]   USER_LED,
//Character LCD
output     [7:0]   LCD_DATA,  //data bus
output             LCD_D_Cn,  //data(H) or command(L)
output             LCD_WEn,   //read(H) or write(L)
output             LCD_CSn    //enable(L)
);

//Push buttons to turn off LEDs
assign USER_LED[1:0] = ~USER_PB[1:0];

//Instantiate nios2e
nios2e u0 (
    .clk_clk                                (CLK_50),
    .reset_reset_n                          (CPU_RESETn),
    .pb_external_connection_export          (USER_PB[3:2]),
    .led_external_connection_export         (USER_LED[3:2]),
    .lcd_bitbang_external_connection_export ({LCD_CSn, LCD_D_Cn, LCD_WEn, LCD_DATA})
);
```

You can compile the design by clicking the play button  on the top or double-clicking the "Compile Design" in the Task panel. After compilation, an sof file will be generated in the "output_files" folder. There is a pre-generated sof file in the "master_image" folder, which can be used to program the FPGA directly.

To program the FPGA, follow the steps of downloading the hardware image introduced before. When programming is complete, you can push Button [1:0] to turn off LED [1:0]. This is the same as we did in the previous lab. The LEDs are wired to Push Buttons in hardware. You can also use this as a way to test whether the hardware image has been downloaded.

Now we can move on to the software design. Launch Nios II SBT for Eclipse from QuartusII: Tools → Nios II Software Build Tools for Eclipse. You can choose any folder to be the workspace but the "software" folder in you Quartus II project folder is a good choice. After the workspace is set up, right click in the blank space of the "Project Explorer" panel on the left. Choose "Import", expand "General", choose "Existing Projects into Workspace" and then click "Next >". Select "Select archive file" then click "Browse". Select the "NiosII_Hello_Wolrd.zip" file in the "software" folder then click "OK", followed by "Finish". This is shown in Figure 73.

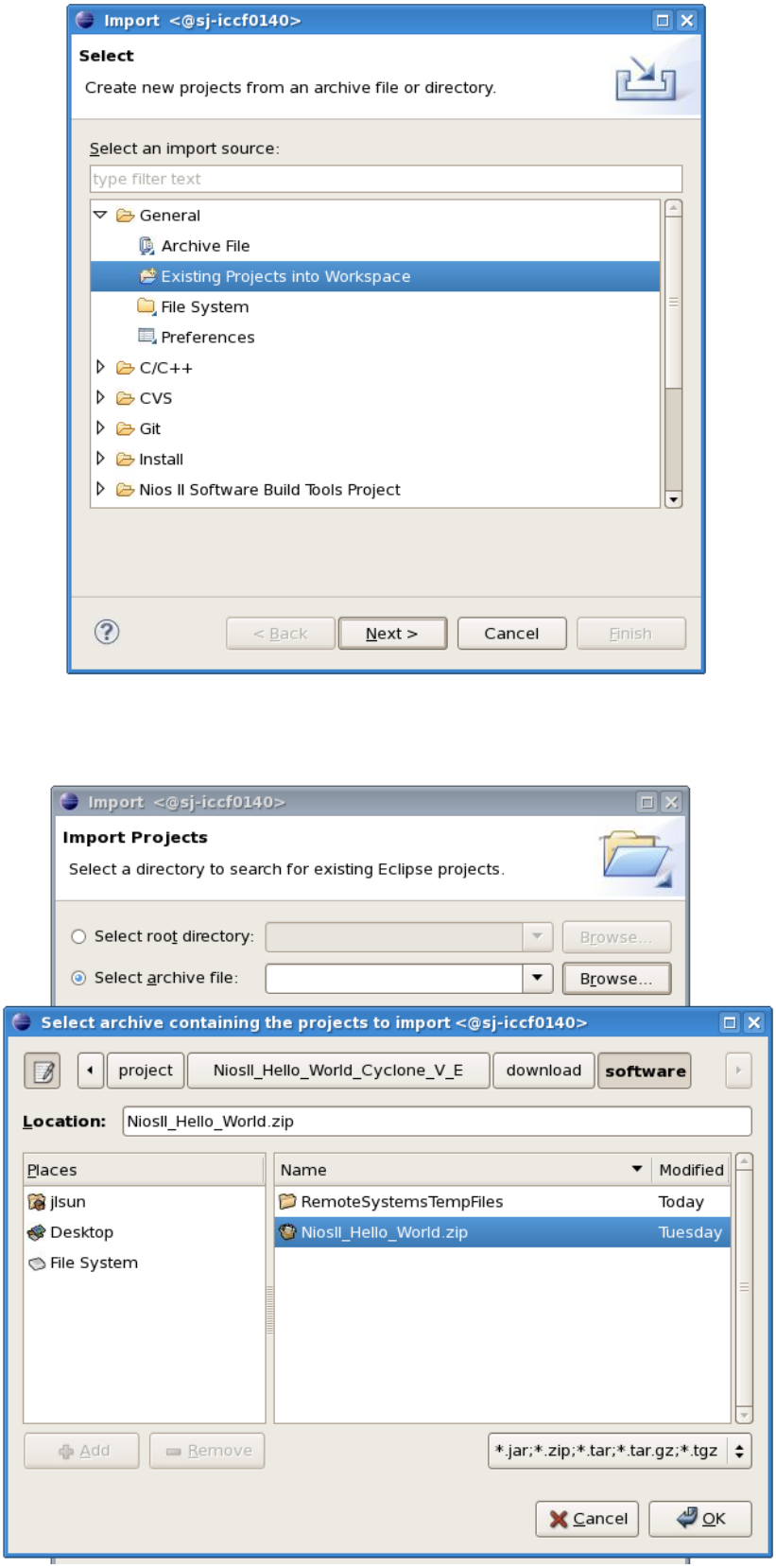

Figure 73: Import Existing Project to Eclipse Workspace

After the project is imported successfully, you can find there are two folders in the “Project Explorer” panel. The first one contains the project files and the second one is the bsp files (Board Support Package), which has all the header files and hardware abstraction. Expand the first folder and double click on “hello_world_small.c” which contains the main function.

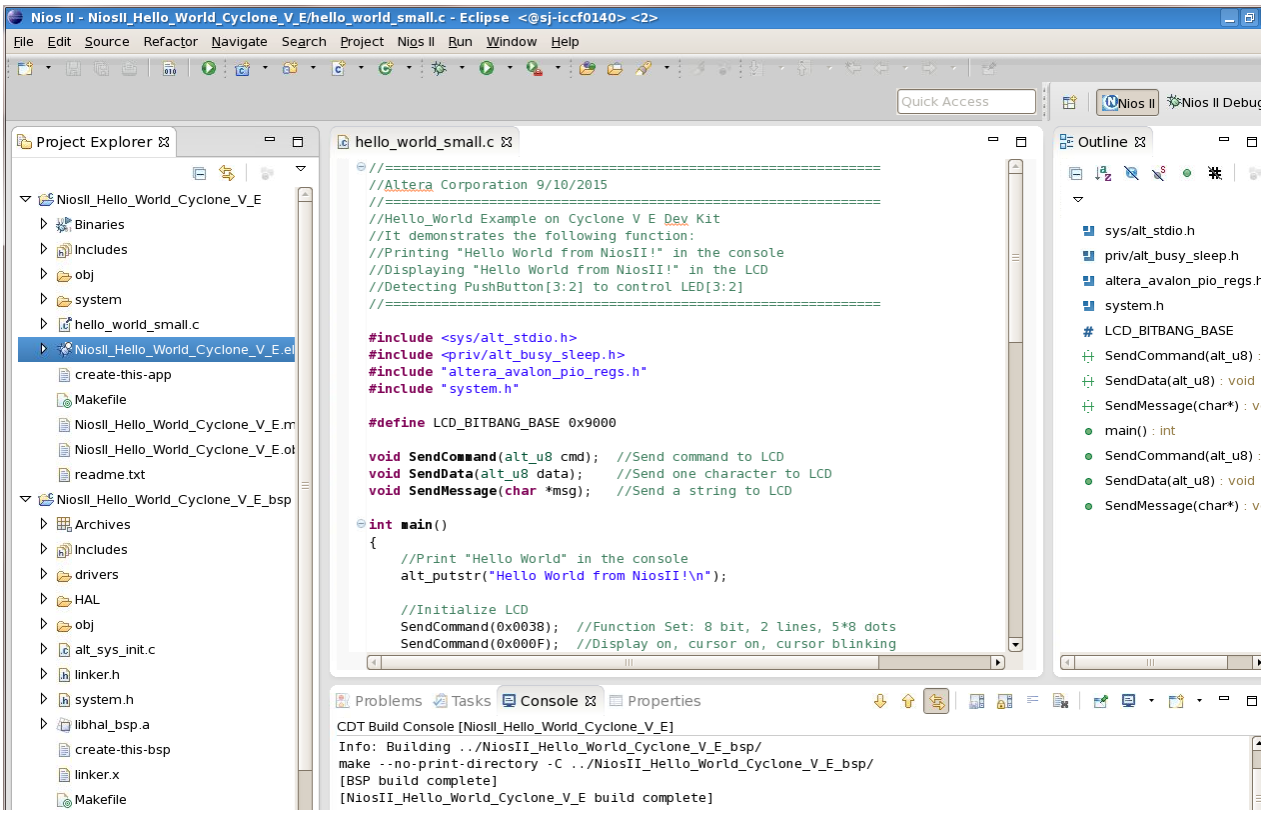

Figure 74: Project in Eclipse Workspace

Now we can go through the code and discuss about how it works. It contains the same code we built in previous lab, which will print a string in the console and scan the Push Button [3:2] to control the LED [3:2]. For now, we will focus on the rest part of this code, which is used to control the character LCD display.

First of all, let’s talk about how Bit Banging works. This LCD uses HD44780 LCD Controller protocol and interface, which support ASCII character set. The table below shows the definition of each bit of the LCD_BITBANG, which is built in Qsys. Data and commands are sent to LCD using the 8 data bus. R/W decides whether we want to read from LCD (H) or write to LCD (L). In this case, we only need to write to LCD so we will keep this bit LOW. RS tells LCD whether the 8 data bits are data (H) or a command (L). To achieve LCD display, we not only need to send data but also need to send command to clear display, set the position of the cursor, set entry mode and so on. Thus this bit needs to be specified. In general, data or command is read on the falling edge of the Enable bit thus we can control LCD follow these 3 steps:

1) Set Enable to HIGH
2) Set RS and D0-D7 desired values
3) Set Enable to LOW

The table lists some example of how to send command and data. Also, there are some timing constraints for this. You can find more specifications in the LCD datasheet and reference manual here. Even though this is not the same LCD on the Cyclone V E Dev Kit, most features of them are the same so it's a good reference.

LCD_BITBANG											Description
Enable	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	0	0	0	0	1	Clear Display
0	0	0	0	0	0	0	0	1	1	0	Right Moving Cursor
0	1	0	0	0	1	1	0	0	0	0	Send Data "0"
0	1	0	0	1	0	0	0	0	0	1	Send Data "A"

Now let's discuss about how this is implemented in software. As shown below, SendCommand() and SendData() functions implements Bit Banging, as we discussed before. The data is sent in hex format to make it succinct. Since this demo only mean to demonstrate the LCD, we don't have strict timing requirements. We choose to make the data hold for 1ms, which will definitely meet the timing constraints. Send Message() sends each character of string so we can use it in the main function by just passing a string as the argument, which makes it more user-friendly.

```
void SendCommand(alt_u8 cmd)  //bitbang
{
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0400 | cmd);
    alt_busy_sleep(1000);
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0000 | cmd);  //Enable
    alt_busy_sleep(1000);
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0400 | cmd);
    alt_busy_sleep(1000);
}

void SendData(alt_u8 data)  //bitbang
{
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0600 | data);
    alt_busy_sleep(1000);
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0200 | data);  //Enable
    alt_busy_sleep(1000);
    IOWR_ALTERA_AVALON_PIO_DATA(LCD_BITBANG_BASE, 0x0600 | data);
    alt_busy_sleep(1000);
}

void SendMessage(char *msg)
{
    for(;*msg!= 0;msg++)
    {
        SendData(*msg);
    }
}
```

The following code is cut out from the main function. It initializes the LCD first. Then it displays “Hello World” on the first line. Next, we change the DDRAM location inside the LCD module to 40H which is mapped to the second line of the LCD. After this, we can display “from NiosII!” on the second line.

```
//Initialize LCD
SendCommand(0x0038);  //Function Set: 8 bit, 2 lines, 5*8 dots
SendCommand(0x000F);  //Display on, cursor on, cursor blinking
SendCommand(0x0001);  //Display Clear
SendCommand(0x0006);  //Entry Mode: right-moving cursor (address increment), no display shift

//Write first line message to LCD
SendMessage("Hello World");

//Change DDRAM locations to 40H to map to the second line
SendCommand(0x00C0);  //Set DDRAM address to 40H

//Write second line message to LCD
SendMessage("from NiosII!");
```

Now we are ready to build the software project and run it on the hardware image that we just downloaded to the Cyclone V E Dev Kit. Right Click on the project folder “NiosII_Hello_World_Cyclone_V_E” and click “Build Project”. An .elf file which is an executable will be generated in this folder. Right Click on this .elf file and choose Run as → Nios II Hardware. Click the “Target Connection” tab. If the connection is not identified, you can click “Refresh Connections”. Once the USB Blaster connection is established, click “Run”.
If the program runs successfully, you can push button [3:2] to turn on LED [3:2]. Also, you can see “Hello World from NiosII!” printed in the Nios II console. The LCD will display the same string, as shown in Figure 75.

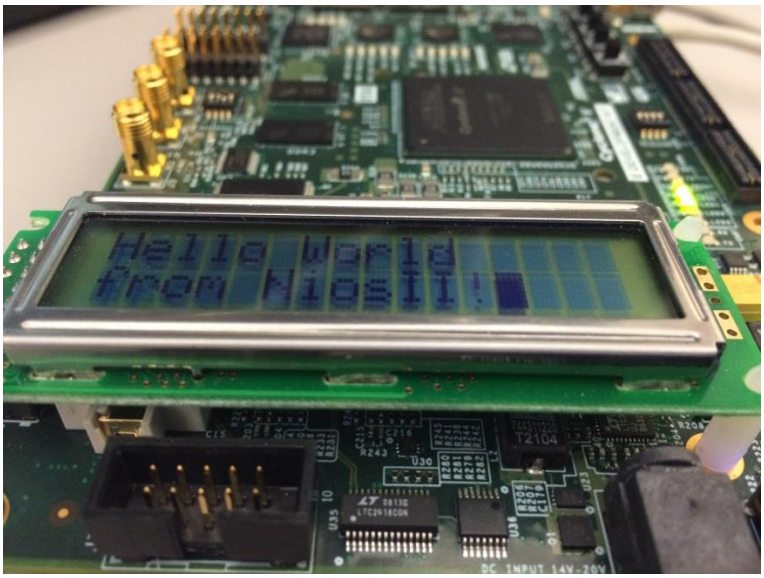

Figure 75: Character LCD Display

You can simply change the characters you want to display by changing the arguments in the SendMessage() function. Remember that this LCD can display at most 16 characters per line. After make changes to the code, save it and build the project again to update the .elf file.

References:

1. Cyclone V E FPGA Development Board Schematics
2. Cyclone V E FPGA Development Board Reference Manual
3. PmodCLP Parallllell LCD Diispllay Module Reference Manual
4. How to drive a character LCD displays using DIP switches