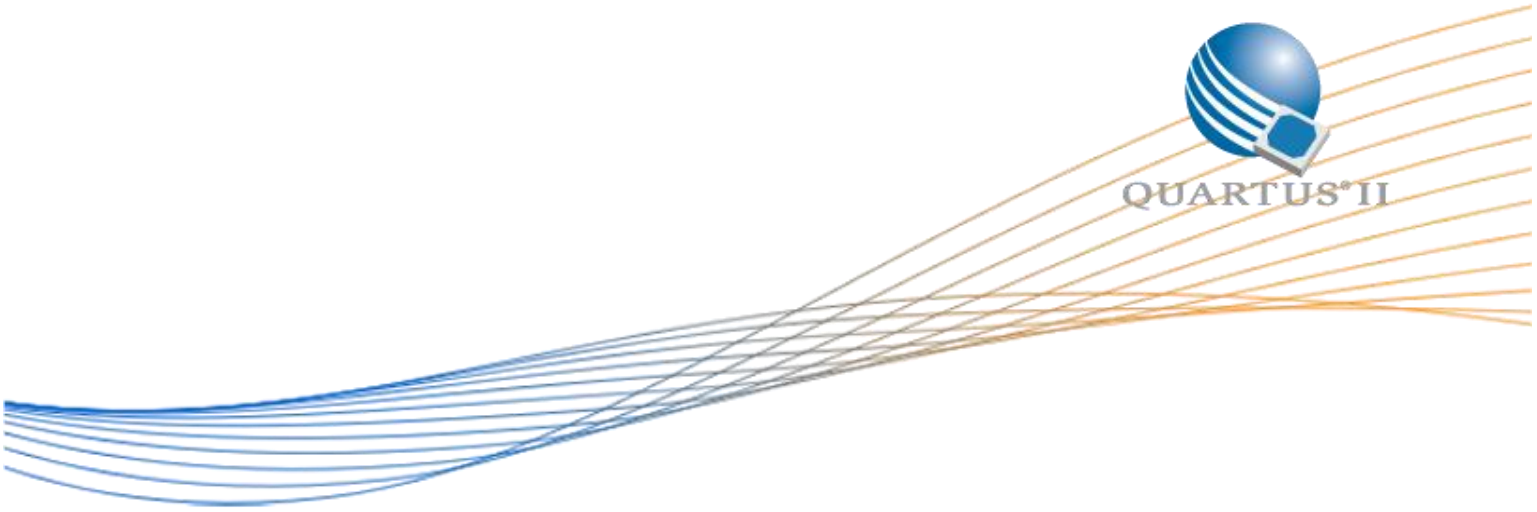


The Nios II Embedded “Hello World” Lab:

For the DE10-Lite Development Kit

April 2017

Version 2.0

Revision History

1.0	5/1/2015	L. Landis	Initial Release
1.1	6/2/2015	L. Landis	Added BeMicro
1.2	11/30/2015	I. Rush	Added CVE DevKit
1.3	12/2/2015	S Meer	Consolidated Sections
1.4	12/4/2015	I.Rush	Updated Pinout Table
1.5	3/18/2016	K. Kita	Separated Lab by Board
1.6	5/10/2016	J. Xia	Revised for university workshops
1.7	6/6/2016	P. Mayer	Added scrolling text
1.8	3/23/17	A. Weinstein	USB blaster installation
1.9	4/3/17	A. Weinstein	Added CVGX DevKit
2.0	4/18/17	A. Weinstein	Updated .qar files
3.0	10/23/17	D. Henderson	Port to DE10-Lite

Contents

Lab 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 such as the Nios II, available in Altera's FPGA families, have built in programmable logic fabric and can be easily modified to suit an applications' requirements. Altera's "SoC FPGA" families are hard processor built from "hard" standard cells that cannot be changed without redesigning the chip. 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.

The lab is organized to run on a number of Altera development kits. The links to the other kits can be found in the Design Store as a Design Example and type in "hello" in the search bar. This lab will show you how to install the Development kit pin settings, design the processor-based hardware system, download it to the Development Kit, and run a simple "Hello World" software program which displays text on your terminal. The initial section of the lab is split into a Hardware Section and Software Section.

Lab Notes

IMPORTANT! PLEASE READ AND FOLLOW THESE GUIDELINES THROUGHOUT THE LAB OR THE LAB WILL NOT WORK.

- **The lab will require you to choose files, components, and other objects; they must be spelled exactly as directed.**
- **DO NOT USE SPACES IN FILE NAMES OR DIRECTORIES.**

Quartus is Altera's design tool suite and it serves several functions. 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 if you use them consistently in your project. Design creation through the use of HDL languages or schematics

1. System creation through the Qsys graphical interface
2. Generation and editing of constraints: timing, pin locations, physical location on die, IO voltage levels
3. Synthesis of high level language into an FPGA netlist ("mapping" in FPGA terminology)
4. FPGA place and route ("fitting" in FPGA terminology)
    a. Generation of design image (used to program FPGA, "assembly" in FPGA terminology)
5. Timing Analysis
6. Programming/download of design image into FPGA hardware
7. Debugging by insertion of debug logic (in-chip logic analyzer)
8. Interfaces to 3rd party tools such as simulators
9. 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 version 17.0 of Quartus II.
Select version 17.0 and your PC's operating system.

For the smallest installation, and quickest download time, only select the technology family you are using based on your development board.

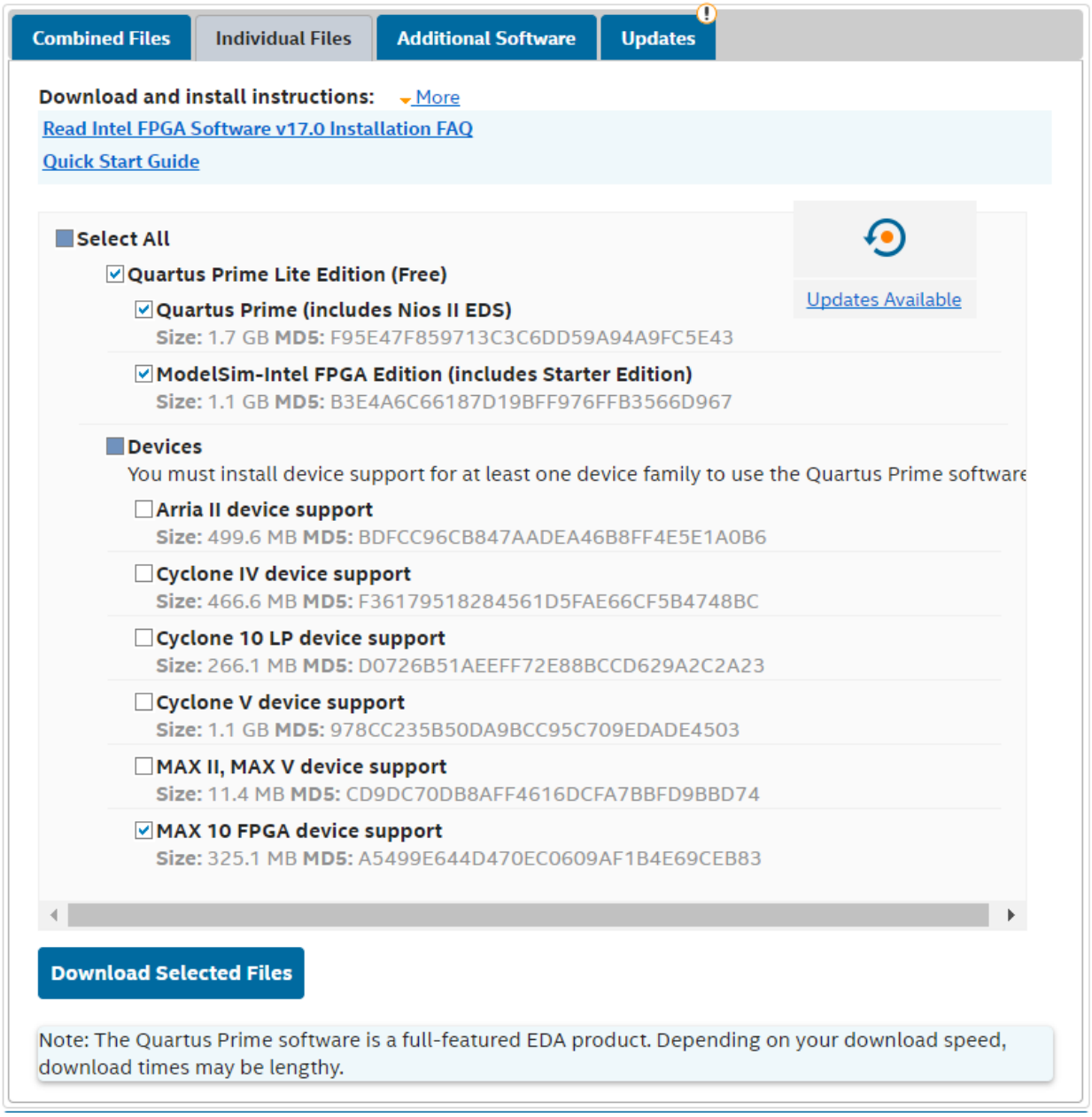


Figure 1: Quartus Download Page

Follow the download instructions provided from the web page. No license is required to run the Quartus Lite software.

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 a relatively easy thing to do as we will demonstrate in this lab.

The Qsys design flow diagram below 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.

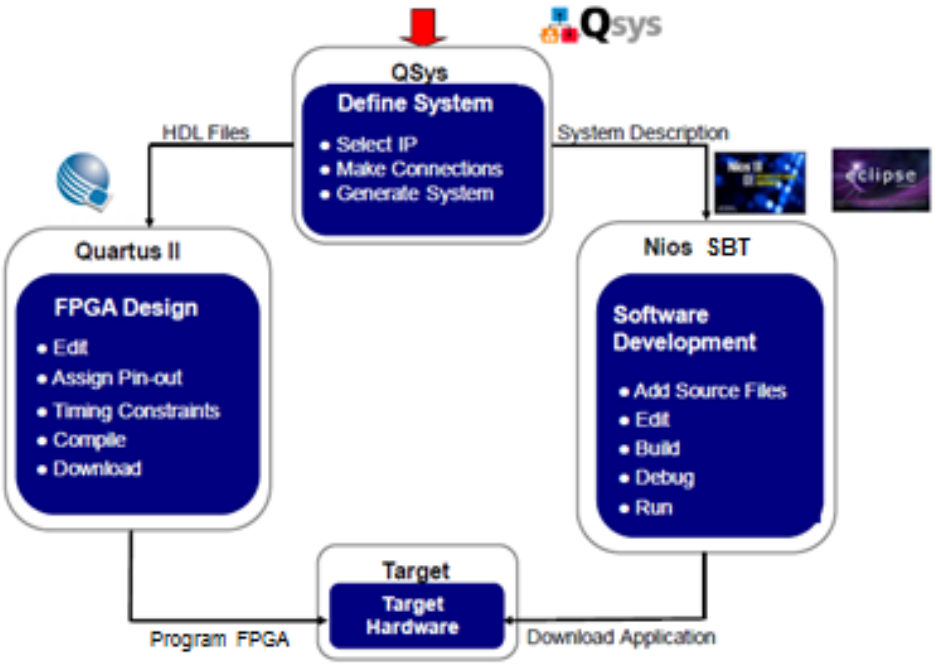


Figure 2: Qsys Development Flow

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.

Objective of the "Hello World" lab

This lab demonstrates how to use Qsys to design the hardware and software to print "Hello World" to your 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 button switches and LEDs to

allow interaction with the development kit. We will use connections to memory that the processor can access to map the various switches and buttons on the device to the LEDs and seven-segment display.

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.

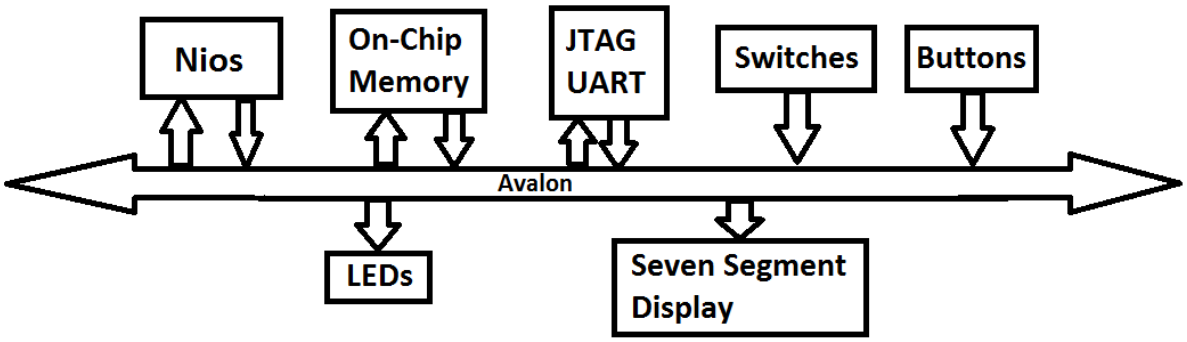


Figure 3: Nios II Based System Used in This Lab

PART 1: HARDWARE DESIGN

1.0: Get started with Quartus

Now you are ready to get started designing hardware!

1. Go to the following link: http://www.alterawiki.com/wiki/File:DE10_Lite_qsys_workshop.zip

   And download the folder named "**DE10_Lite_qsys_workshop**" to your computer.
2. Unzip the .zip file (Right click on the .zip folder and select ***Extract All***... Browse to the directory you want your unzipped files to go and press enter.
   a. Within the file there are three items
      i. DE10_C_CODE
      ii. DE10_Lite.qar
      iii. Hello_World_Lab_Manual_DE_10_Lite_UPDATED.pdf
3. **The lab will not work if you do not unzip the files!**
4. Double click on the (**.qar**) file.

If you have Quartus completely installed, the Quartus software should open the (**.qar**) file. (**.qar**) stands for "**Quartus Archive File**" and allows a user to store a project and its related files in a single, (**.qar**) and then restore the project later. This is done for your convenience and to make the overall lab time shorter.

5. Select "OK" for the first screen that appears when the (**.qar**) file opens.

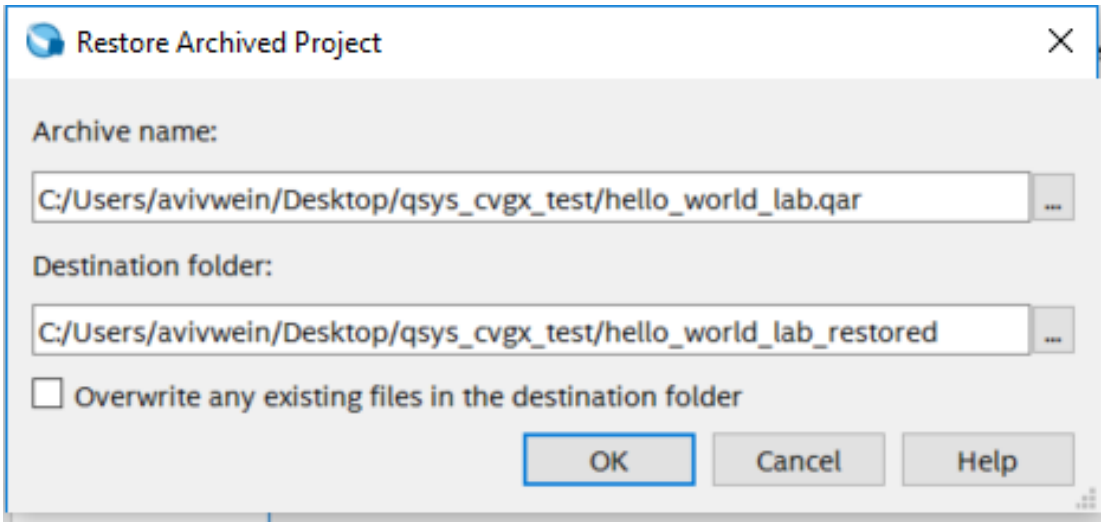


Figure 4: Selecting Archive Name and Destination Folder for the .qar file.

Once the (**.qar**) is done unpacking all its files, you will be able to navigate around the main Quartus window. We will start building our system by using Qsys.

2.0: Building your Qsys Based Processor System

The Qsys system panel diagram illustrates what you are designing in the Qsys environment. The system we are building will have a clock, a single master (the Nios II processor), and 11 slave devices.

Building the Qsys system is a highly efficient way of designing systems with or without a processor.

1. Launch Qsys from Quartus: *Tools* → *Qsys*. The initial screen you should see looks like this:

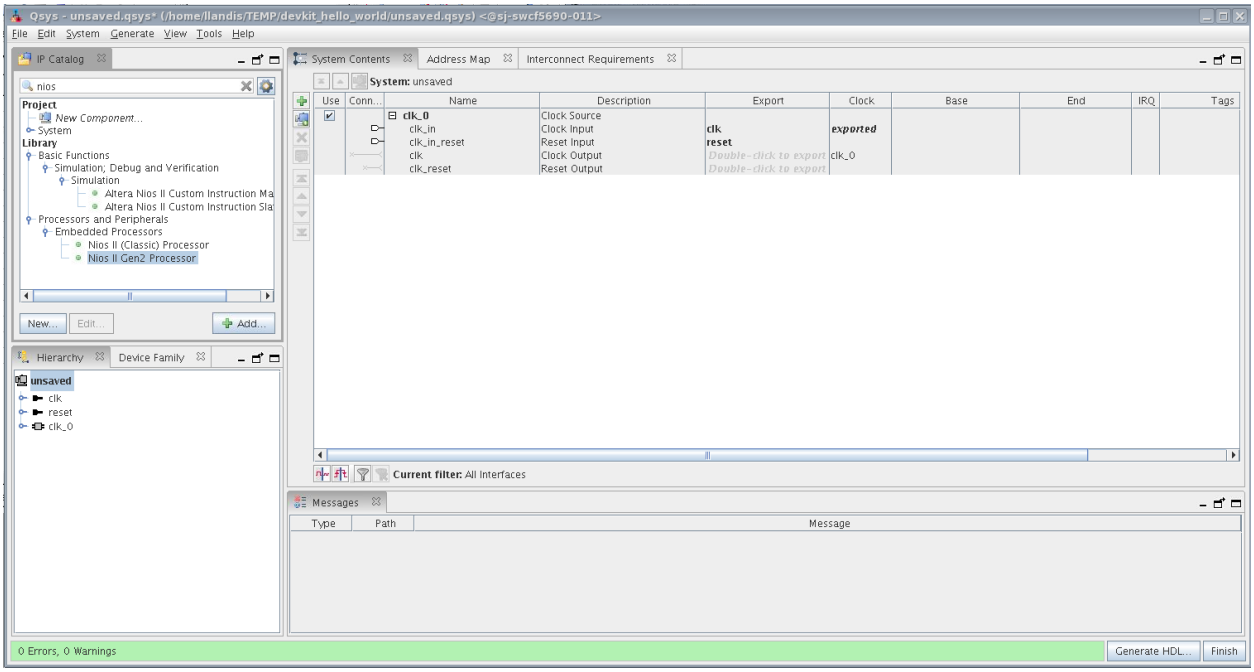


Figure 5: Qsys Main Panel

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

2.1: Adding in the Nios II Processor

Look for the IP catalog tab in the top left of the Qsys window. Below the IP catalog tab, you can search for the various components you want to add to your Qsys based system.

2. Enter "Nios" in the search tab and select the **Nios II Processor (not the Classic Nios II)** from the library be double clicking. See Figure 6 on the following page for example.

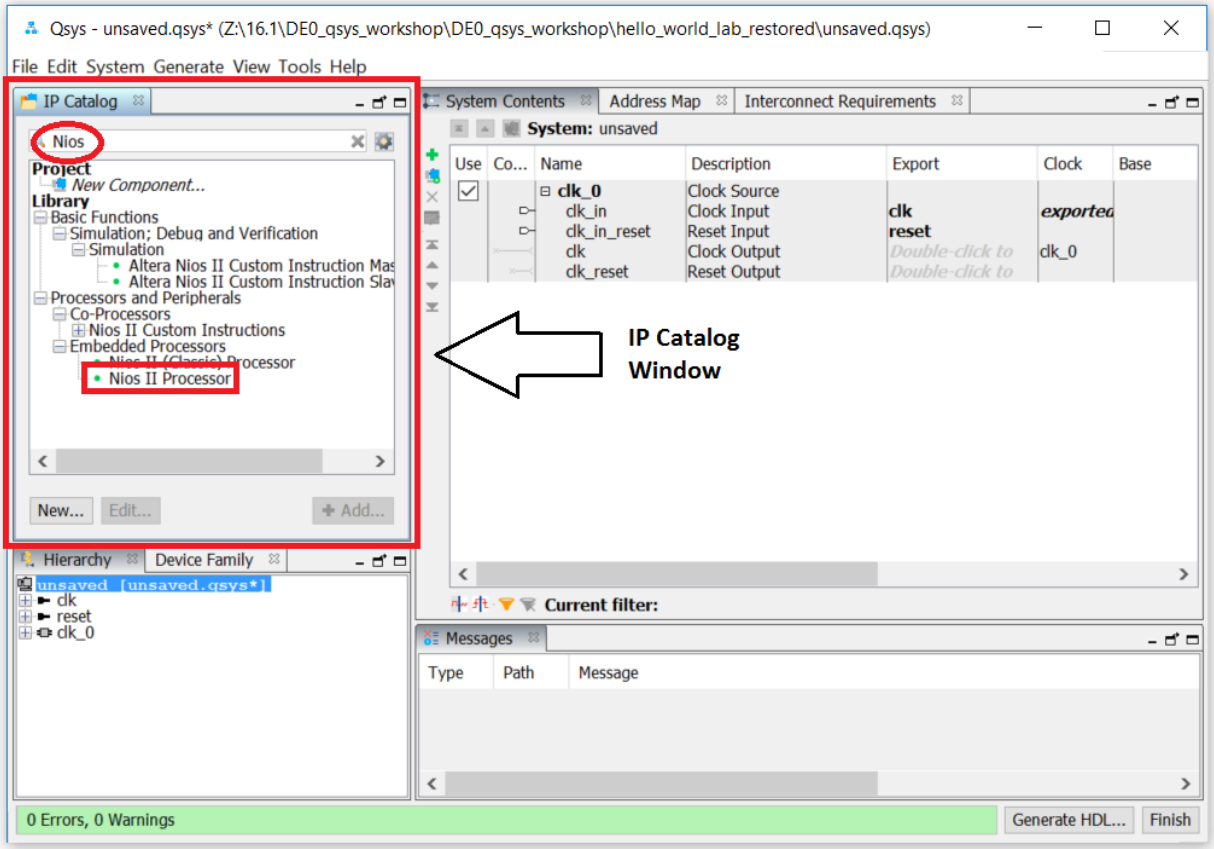


Figure 6: IP Catalog Tab

A configuration window will appear, in this select the **Nios II/e processor**. The '**e**' stands for economy and the '**f**' stands for fast. We will use the economy version in this lab.

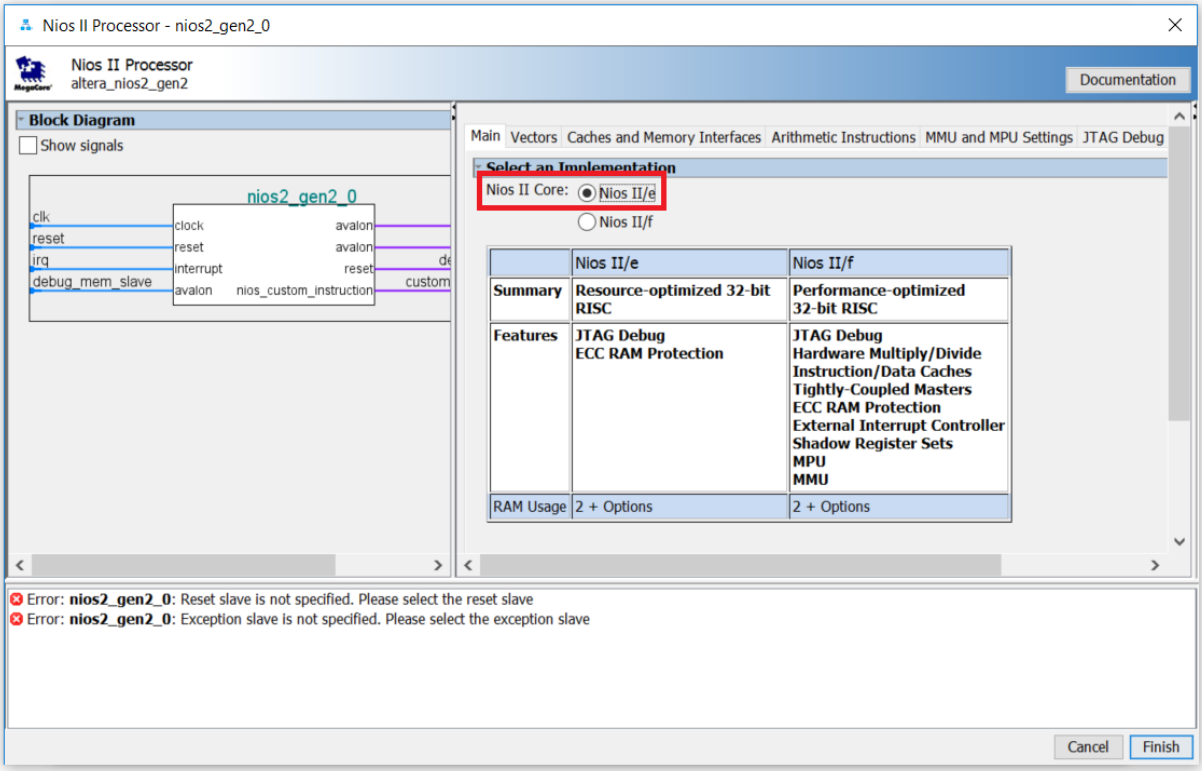


Figure 7: Nios II Gen2 Configuration Panel

3. Aside from choosing '**e**', keep the default settings and click **Finish** and you will see the **nios2_gen2_0 processor** in your connection diagram.

*****For now don't worry about the system errors reported, we will address them soon.*****

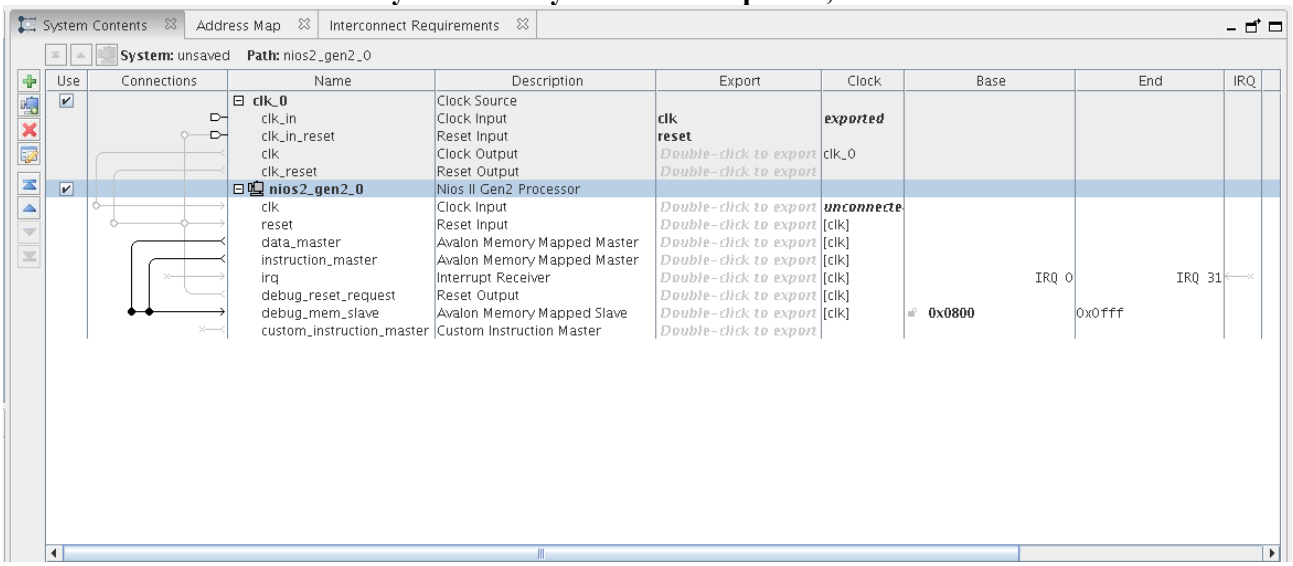


Figure 8: Qsys System Contents Panel

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.

2.2: Adding in On Chip Memory

With the Nios II processor added, you still need to add: **On Chip Memory**, **JTAG UART**, **pushbutton inputs**, **switch inputs**, **led outputs**, and the **7-segment display output** to your system.

4. Search for "*memory*" in the IP catalog. You will see many options for memory, select "*On Chip Memory→On-Chip Memory (RAM or ROM)*", as shown here:

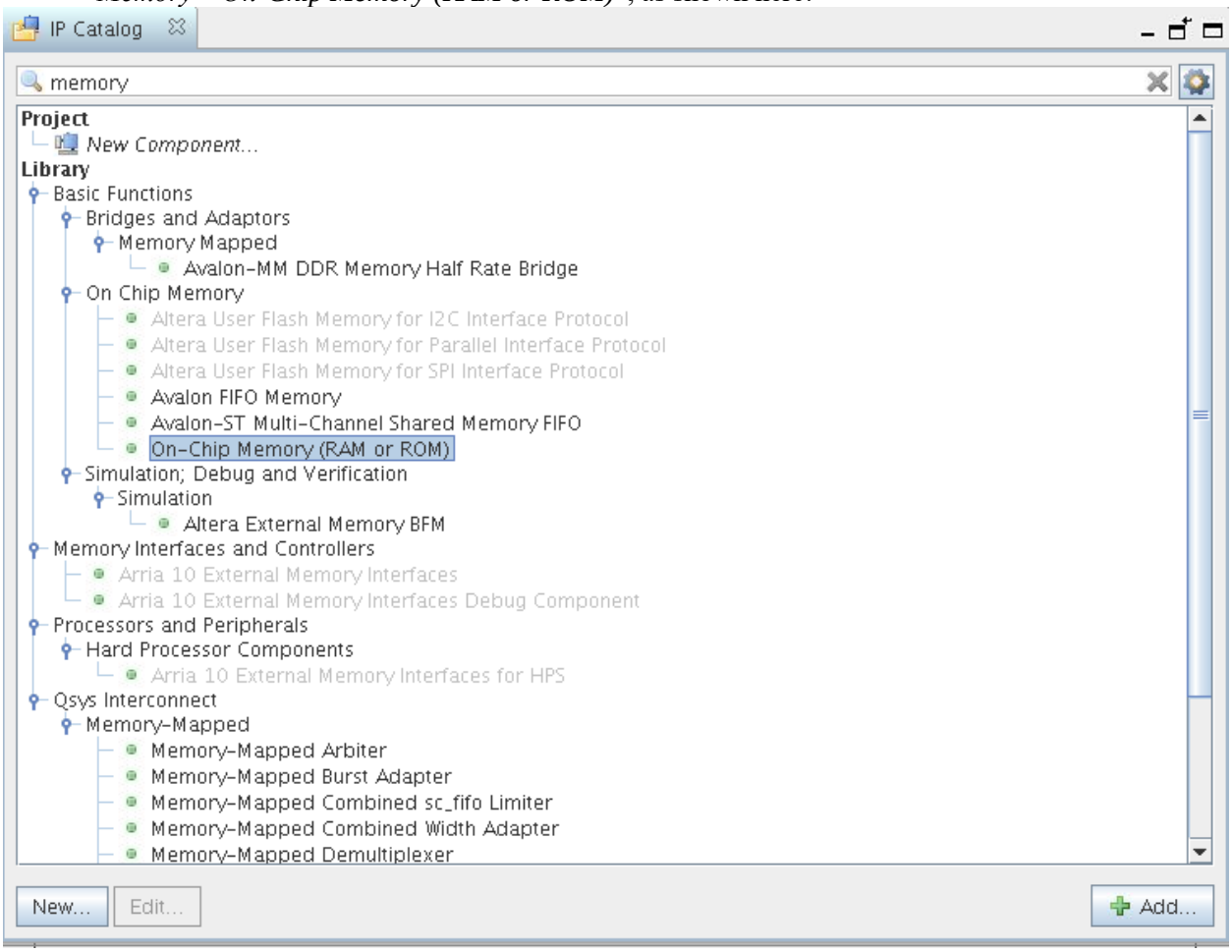


Figure 9: IP Catalog Search for On-Chip Memory

5. Locate the **On-Chip Memory (RAM or ROM)** component and double click on the component or click "*Add"*.
6. In the component settings memory panel that pops up, you need to change the memory size from **4096** to **65, 536**. This will ensure that you have a plenty of space for your software program.
7. 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.

See Figure 10 on the following page for the areas you need to edit.

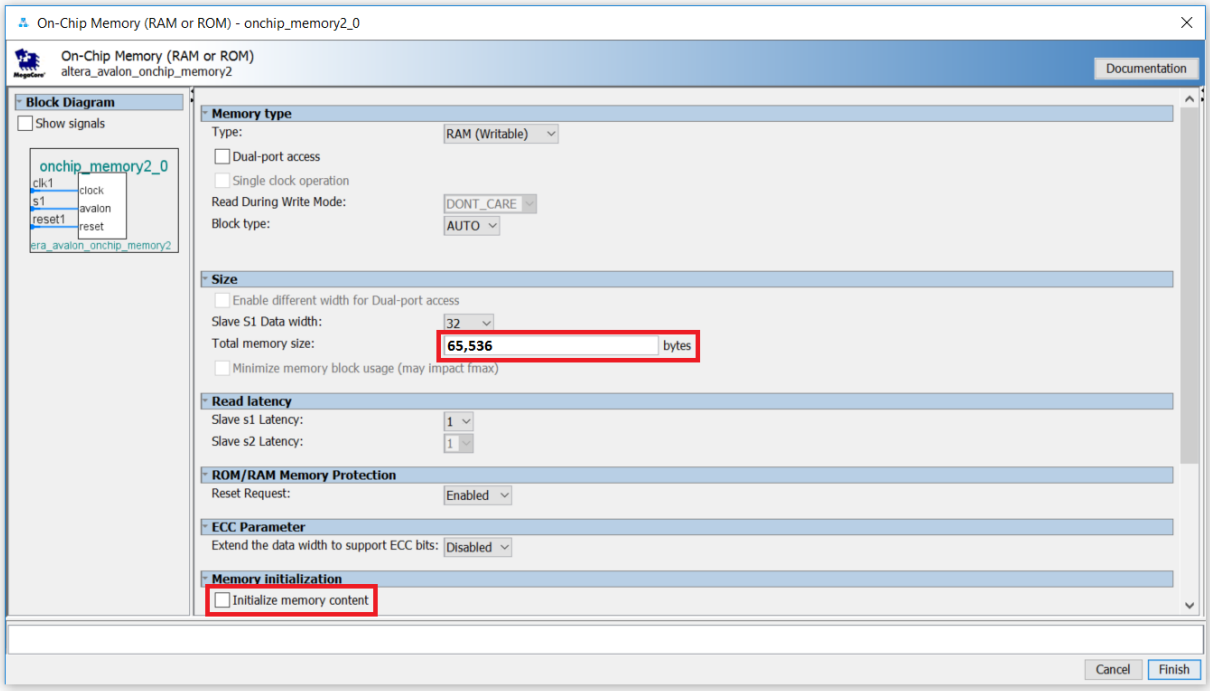


Figure 10: On-Chip Memory Configuration Panel

8. Click **Finish** and you will now see a total of three components in your Qsys system**:**
    a. **clk_0**
    b. **nios2_gen2_0**
    c. **onchip_memory2_0**.

See Figure 11 on the following page for what your Qsys window/Qsys system should look like at this point in the lab.

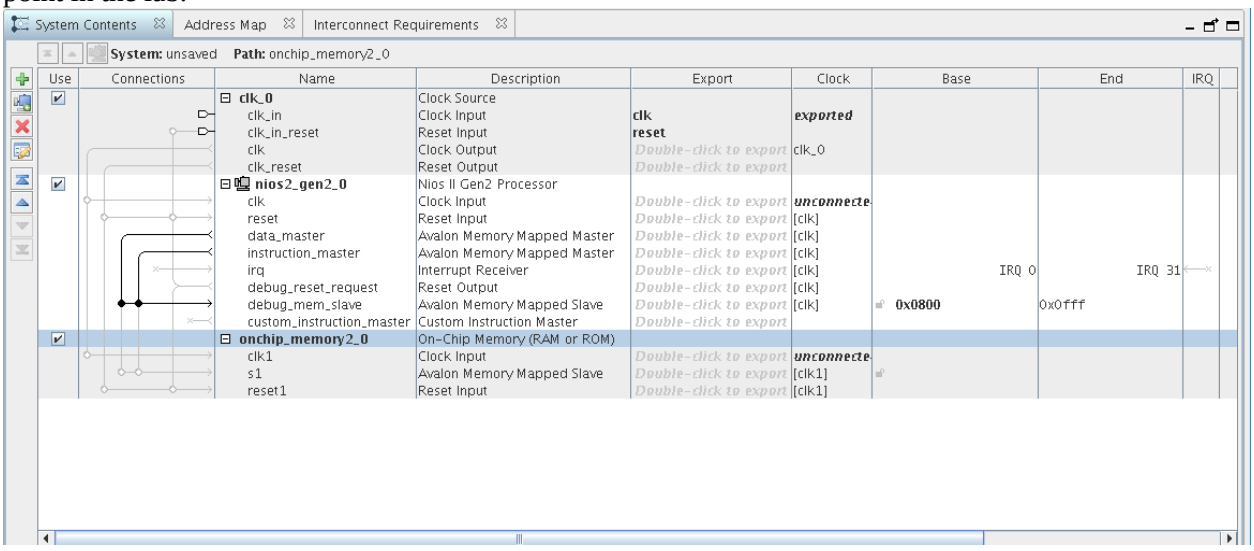


Figure 11: System Contents with Nios II and On-Chip Memory

2.3: Adding in JTAG UART Component

The next component you will add is the **JTAG UART**.

9. Search for "*JTAG"* in the IP catalog, locate the **JTAG UART** and double click or click add.
10. Keep the default settings and click **Finish**

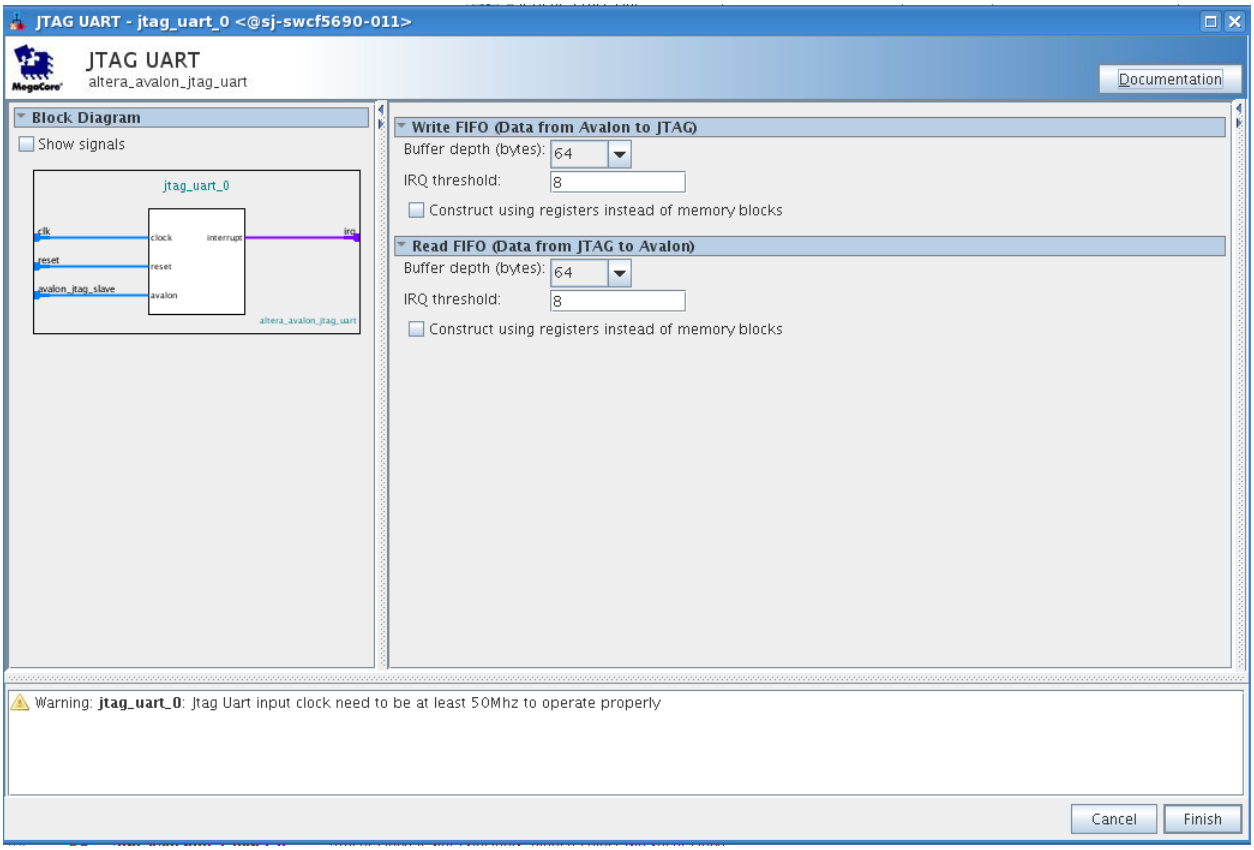


Figure 12: JTAG UART Configuration Panel

2.4: Adding Parallel IO (PIO) to your Qsys Design for Modeling Switches, Pushbuttons, LEDs, and the Seven Segment Display

The next five components, which handle the interfacing of the *switches*, *pushbuttons*, and *LEDs*, are configured instances of general purpose parallel IO components in the IP catalog. By using the PIO block for the switches, buttons, and LEDs, you will be able to map these values to address space and your C code will read and write these components.

11. Search for **parallel IO (PIO)** and select this block.
12. For the **pushbutton block**, we will set this up as a **4-bit, input interface** using the settings shown below
    a. There are two pushbuttons we would like to read from and two internal signals (a modification to HDL to support the DE10 board).
13. When you have setup your button input component interface as in Figure 13, click **Finish.**

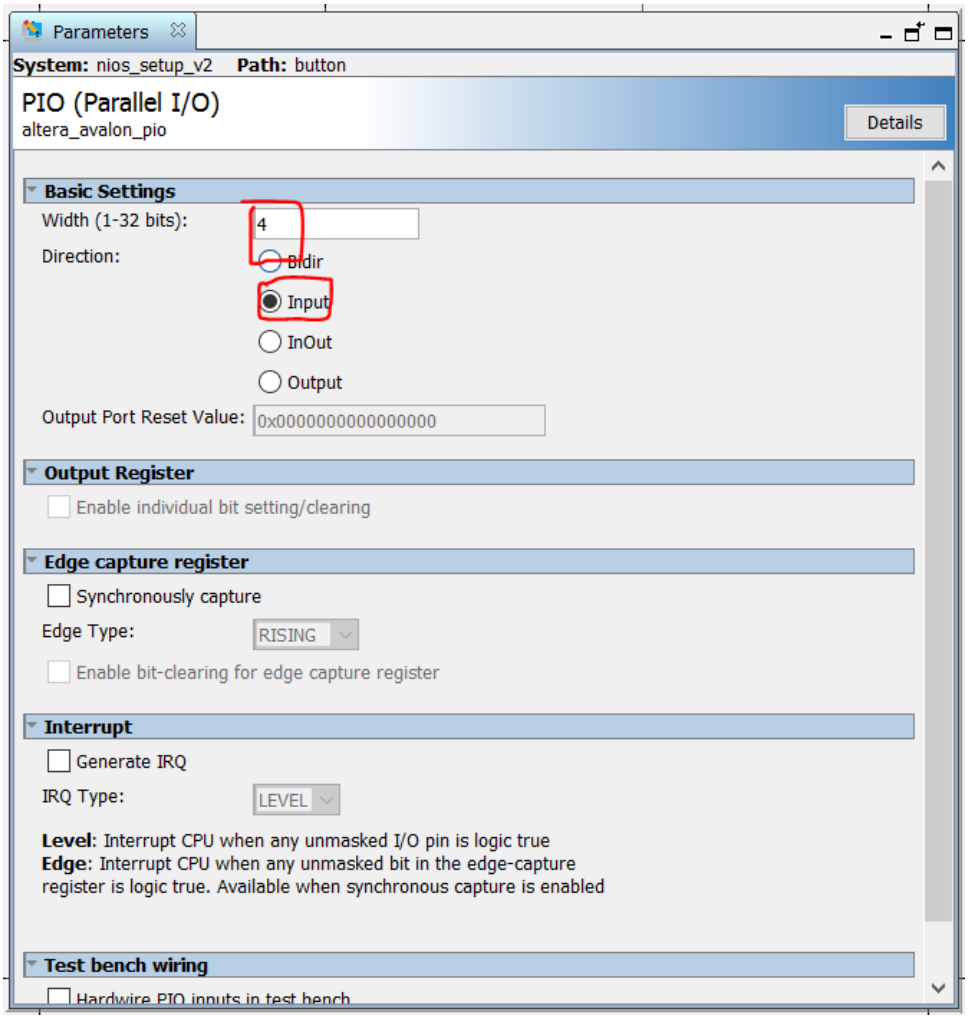


Figure 13: Parallel IO Configuration Panel for Pushbutton

Next, you will add a second PIO block. This one will be for the switches that the user can flip.

14. For the switch block, you will set this up as a **10 bit, input interface**
    a. Since there are 10 switches on the board, using the settings shown below.
15. When you have setup your switch input component interface as in Figure 14, click **Finish.**

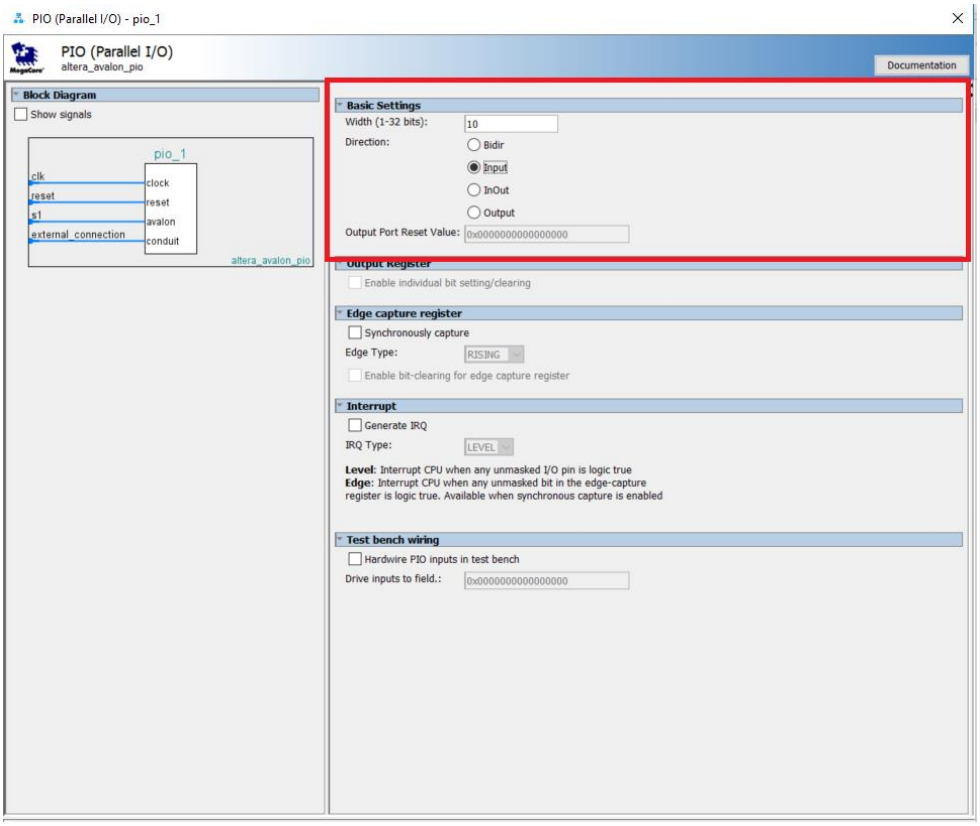


Figure 14: Parallel IO Configuration Panel for Switches

16. Double click on the PIO component as you did for the SWITCH and BUTTON. This time you will configure this component as the **LEDs** which is a **10 bit, output interface.**
    a. Since there are 10 Red LEDs on the board.
17. When you have setup your led output component interface as in Figure 15, click **Finish.**

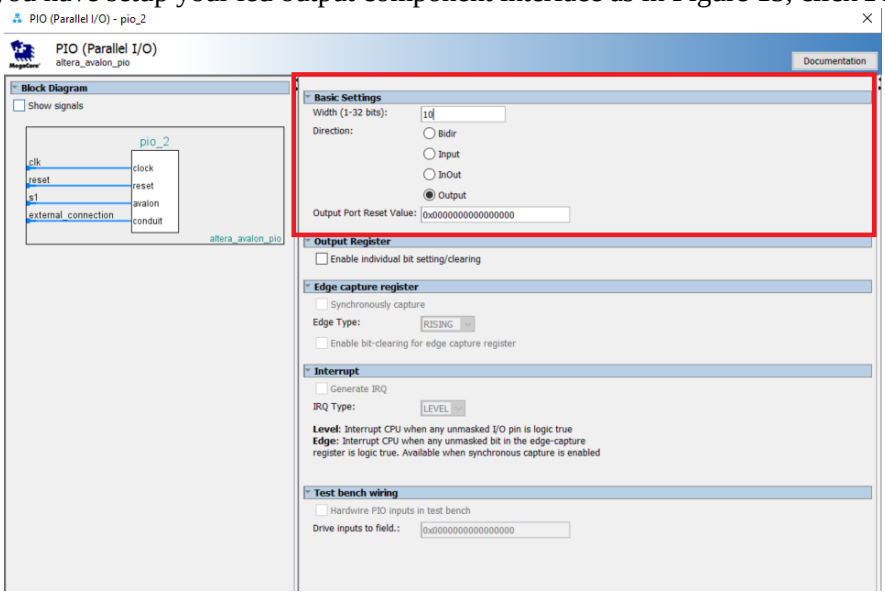


Figure 15: Parallel IO Configuration Panel for LED Outputs

Finally, we will add the six 7-segment displays that will allow us to display text on the board.

18. Create another PIO component, and configure it as a **7-bit output**
    a. Since we need one bit for each light.
19. When you have setup your seven segment display output component interface as in Figure 16, click **Finish.**

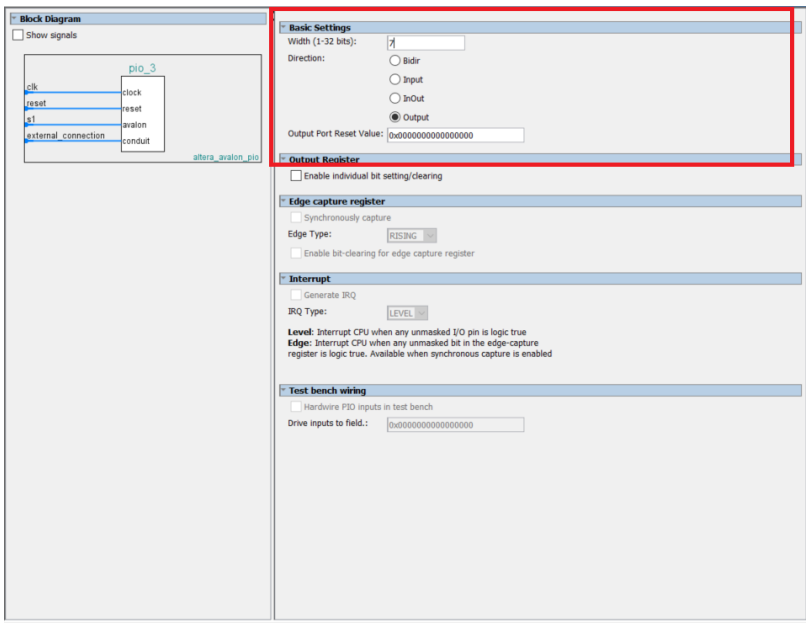


Figure 16: Parallel IO Configuration Panel for Seven-Segment Display Outputs

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

2.1: Connecting your Qsys System Components Together

1. In the system contents tab, right click on *clk*_0, select rename, and type in **clk**.
2. Select the *nios2_gen_2_0* component, select rename and type in **cpu**,
3. Similarly, rename the rest of the components as follows
    a. **onchip_memory**
    b. **jtag_uart**
    c. **button**
    d. **switch**
    e. **led**
    f. **hex0**

This will make these components' names easy to remember and reference in future steps. When you finish, your system contents panel should look like Figure 17: System Content Connections Starting Panel.

It is important that your names match these *exactly*, or your code may not compile!

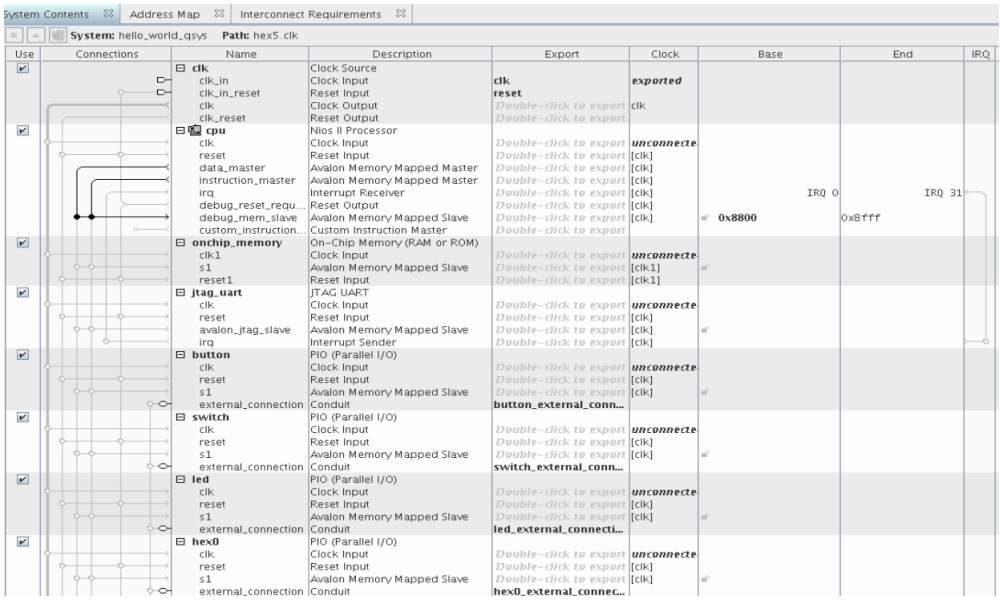


Figure 17: System Content Connections Starting Panel

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

4. Highlight the clock output coming out of the **clk** pin by clicking on the text that says "**clk**" above the "**clk_reset**" description. When first selected, it will be a gray color.
5. Make connections between the **clk** component and the *clk* inputs of each of the other components by clicking on the small open circles on the lines that intersecting with the other components.

You should see something like Figure 18 on the following page.

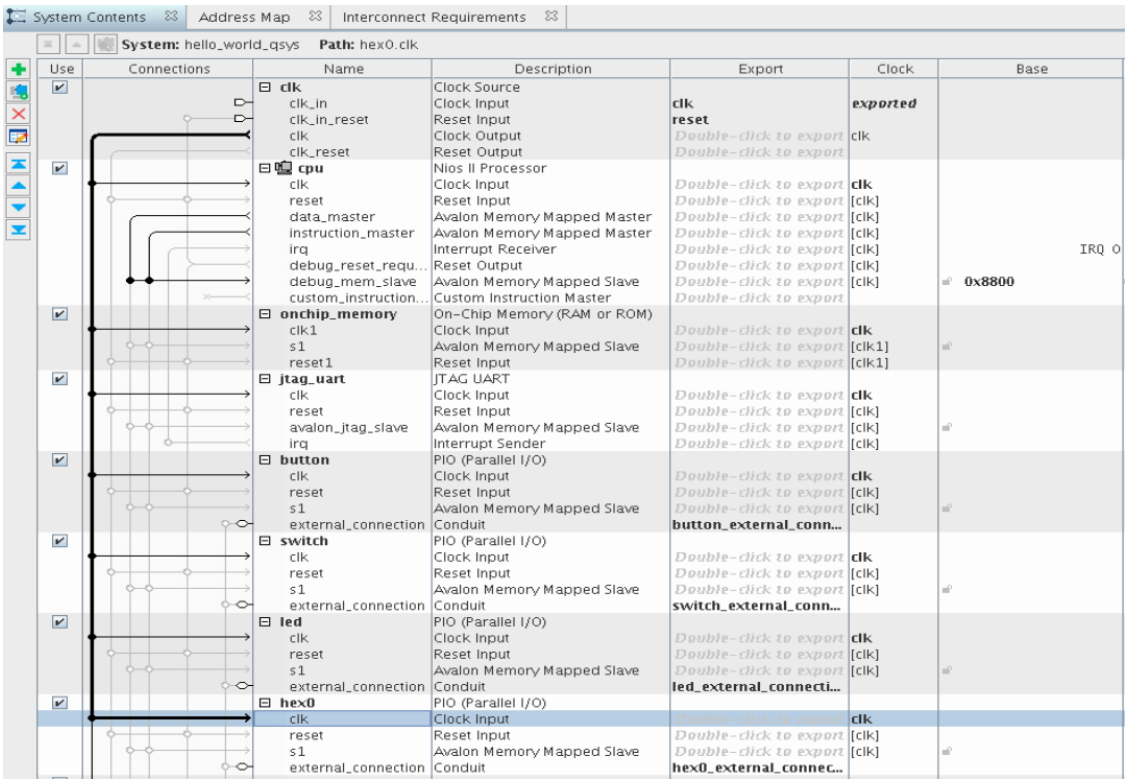


Figure 18: System Contents after Connecting the Clock

6. Perform the same operation to connect the "**clk_reset"** from the clock component to the "**reset**" signals on the other components.
    a. At this stage, your Qsys design should look like Figure 19 below.

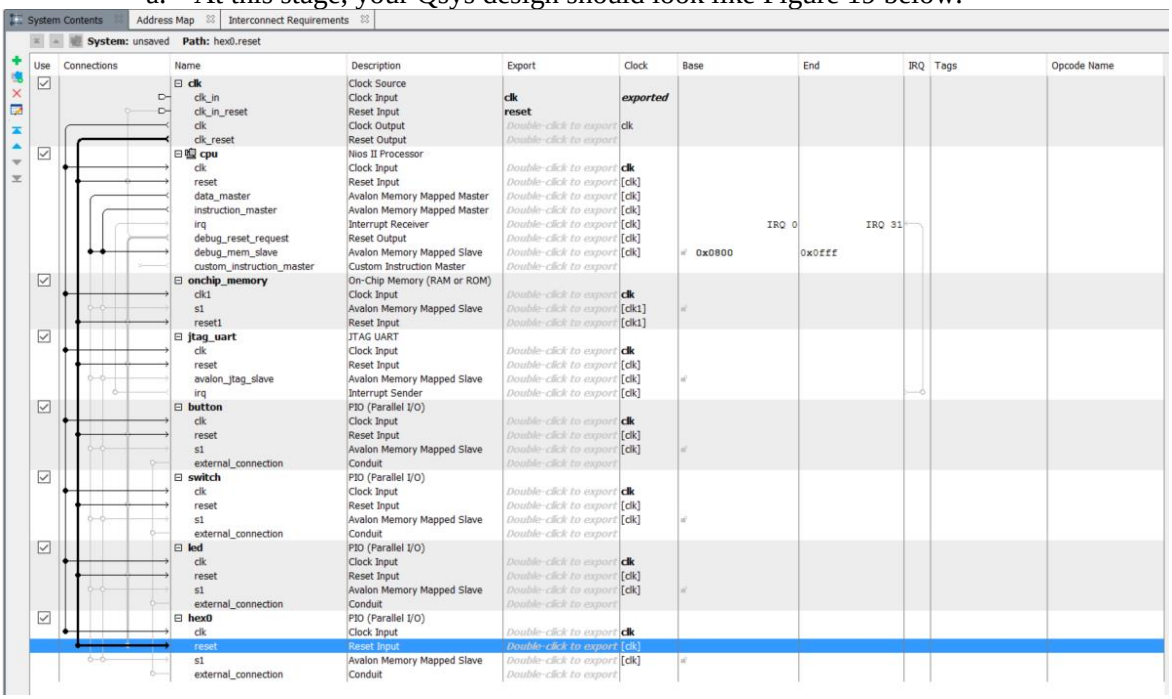


Figure 19: 'clk' and 'clk_reset' Connected in Qsys

7. Connect the **cpu.data_master** to the slaves. Make the connections between the:
    a. **cpu.data_master** and the **s1** connection of the *onchip memory*
    b. **cpu.data_master** and the **avalon_jtag_slave** on the *uart* component,
    c. **cpu.data_master** and the **s1** port on the *button* component,
    d. **cpu.data_master** and the **s1** port on the *switch* component
    e. **cpu.data_master** and the **s1** port of the *led* component,
    f. **cpu.data_master** and the **s1** port on the *hex0* component
    g. **cpu.instruction_master** and the **s1** port of the *onchip_memory*
8. **Instruction master** is by default connected to **debug_mem_slave**.

Figure 20 below shows the Qsys system with the **cpu.data_master** signal and **cpu.instruction_master** signal connected to the other components in the proper locations.

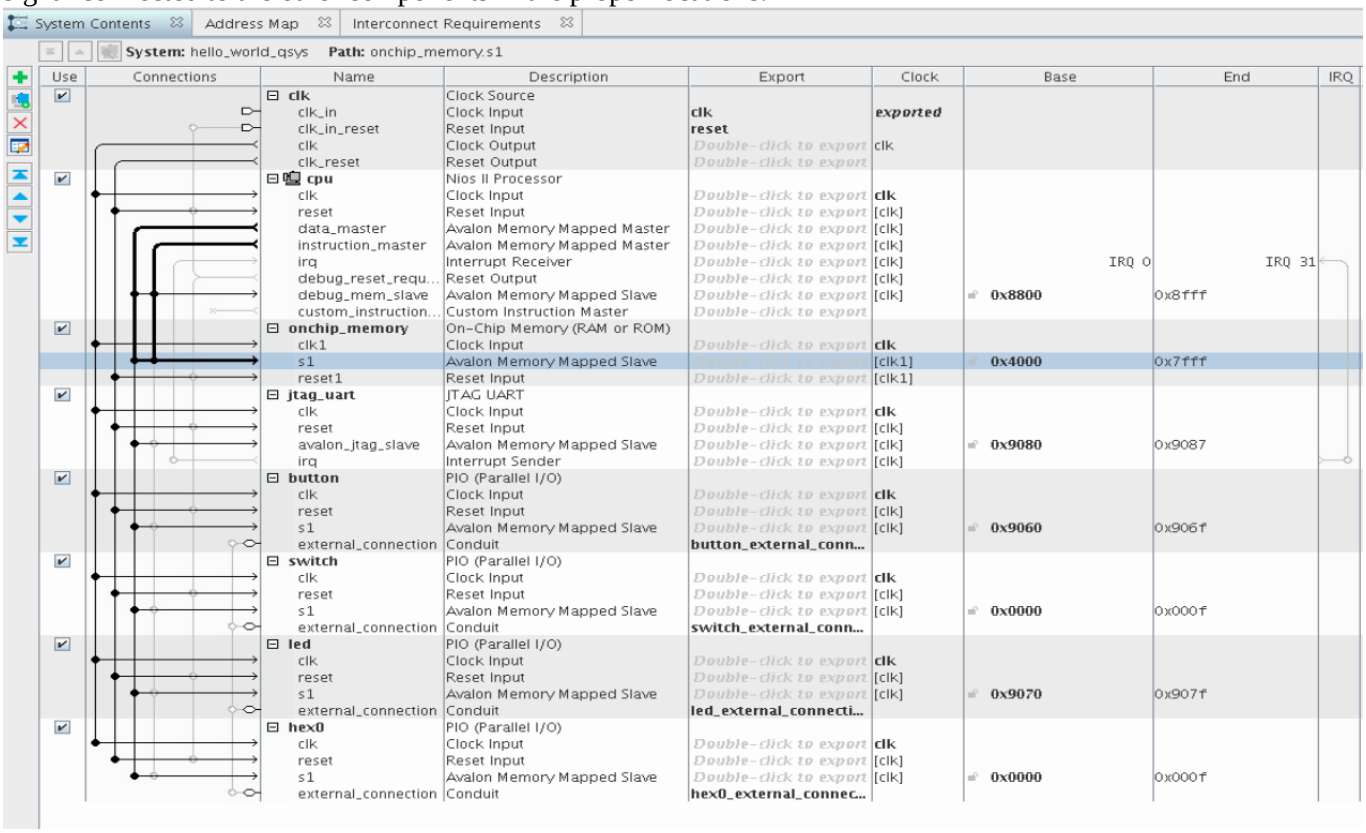


Figure 20: System Contents after Data Master/Slave Connection

The ***instruction_master*** signal from the **cpu** component does not need to be connected to each slave component as it only needs access to memory that contains the software executable.

9. The next connections to make are the processor interrupt request (**IRQ**) signals. Make this connection as shown in Figure 21 by clicking on the empty bubble.
    a. We will use the default setting for the **IRQ** number.

The **UART** can drive interrupts, and hence needs to be wired to the cpu processor interrupt lines.

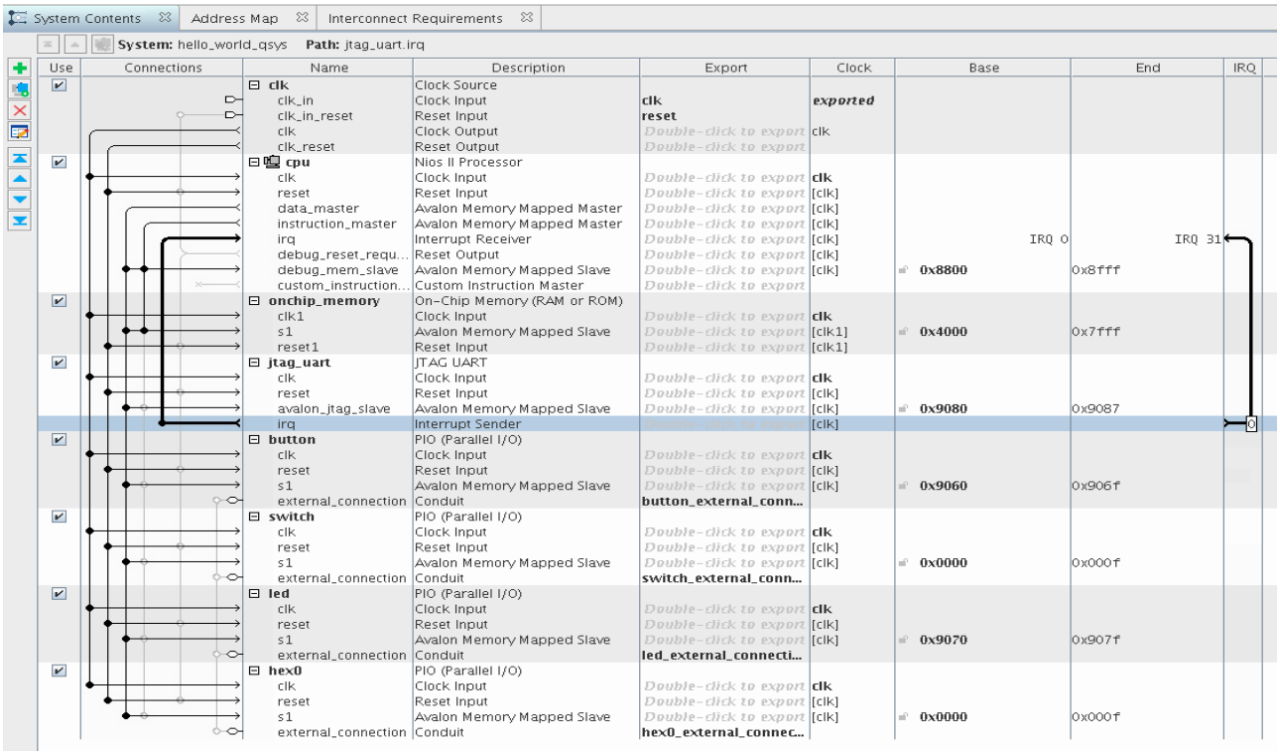
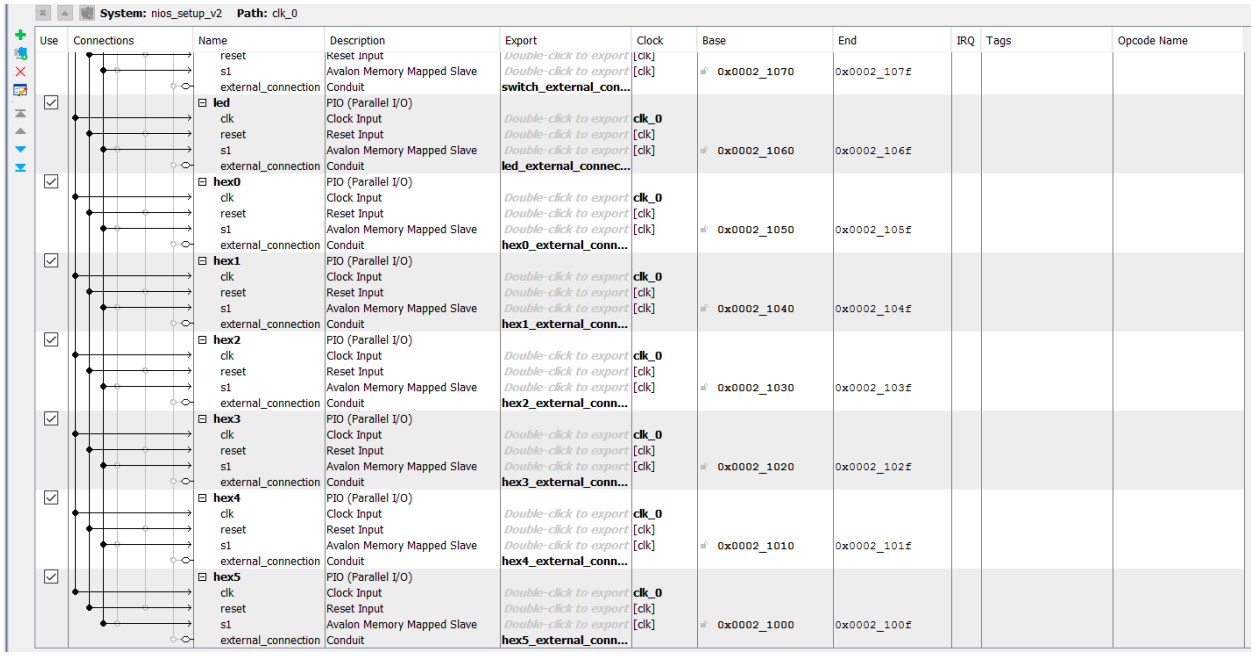


Figure 21: System Contents after Interrupt Connections

Now that the connections are made, we need to add the other three seven-segment displays (as there are six of them in total on the board).

10. Select **hex0**, right-clicking, and select "*duplicate*."
    a. Alternatively, you can click on hex0 and press "*ctrl-D*" to duplicate the module.

11. Once you have six of them (0-5), rename the new ones so they form the following list (pictured in

12. 

13. Figure 22): **hex0, hex1, hex2, hex3, hex4, hex5.**

14. In case the connections were not kept when duplicating the new PIOs, be sure to connect:
    a. "**clk**" from the clock component to the clock signal of each hex component
    b. "**clk_reset**" from the clock component to "**reset**" of the hex component
    c. "**cpu.data_master**" from the **cpu** component to "**s1**" of each hex component
15. Do this for all six of the hex PIOs.

When all is said and done, your system contents panel should look like

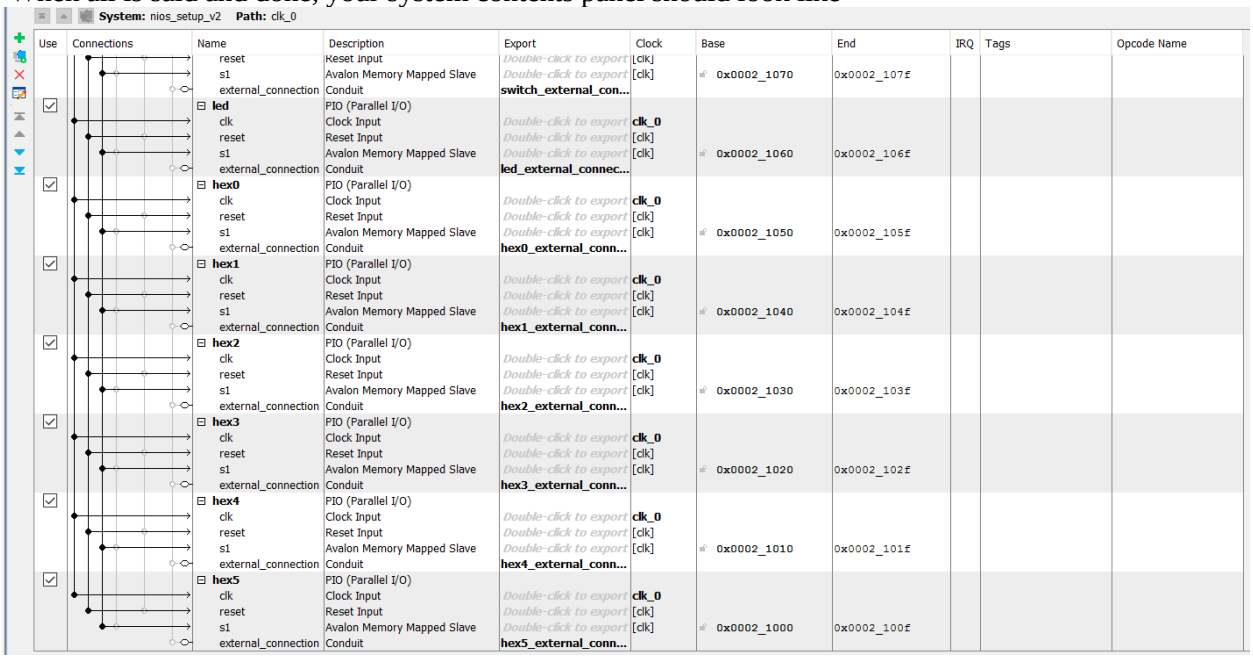


Figure 22 on the following page.

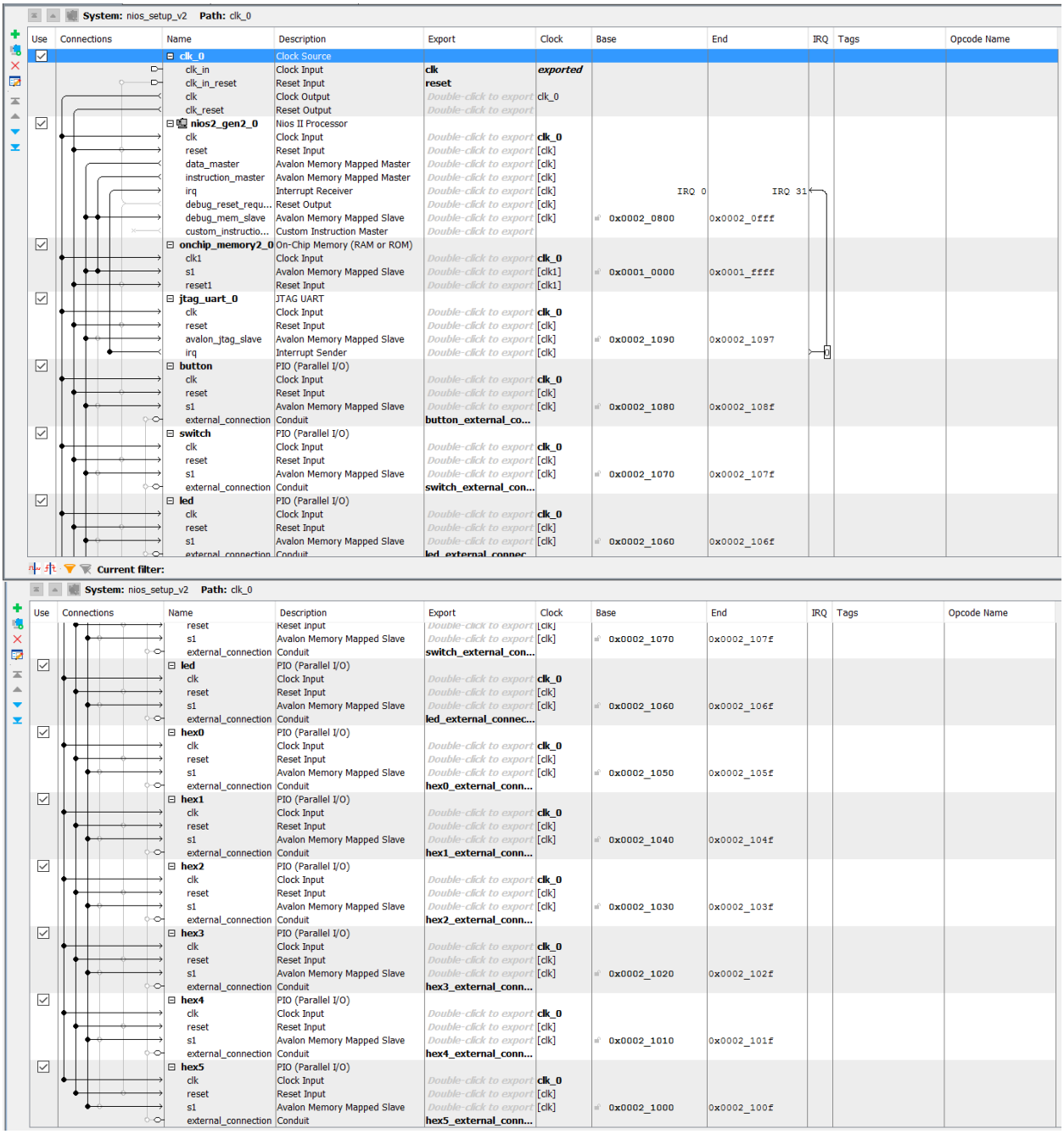


Figure 22: System Contents after Adding All 4 Seven Segment Displays

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.

16. Double click on the *button*, *switch*, *led*, and *hex0-hex5* conduit items under the export column circled in Figure 23 on the following page. This will bring these ports out of the Qsys component to connect to the top-level design.

*****Be sure the names of the components and exports match what is in Figure 27 *EXACTLY*, or the design may not compile at runtime*****

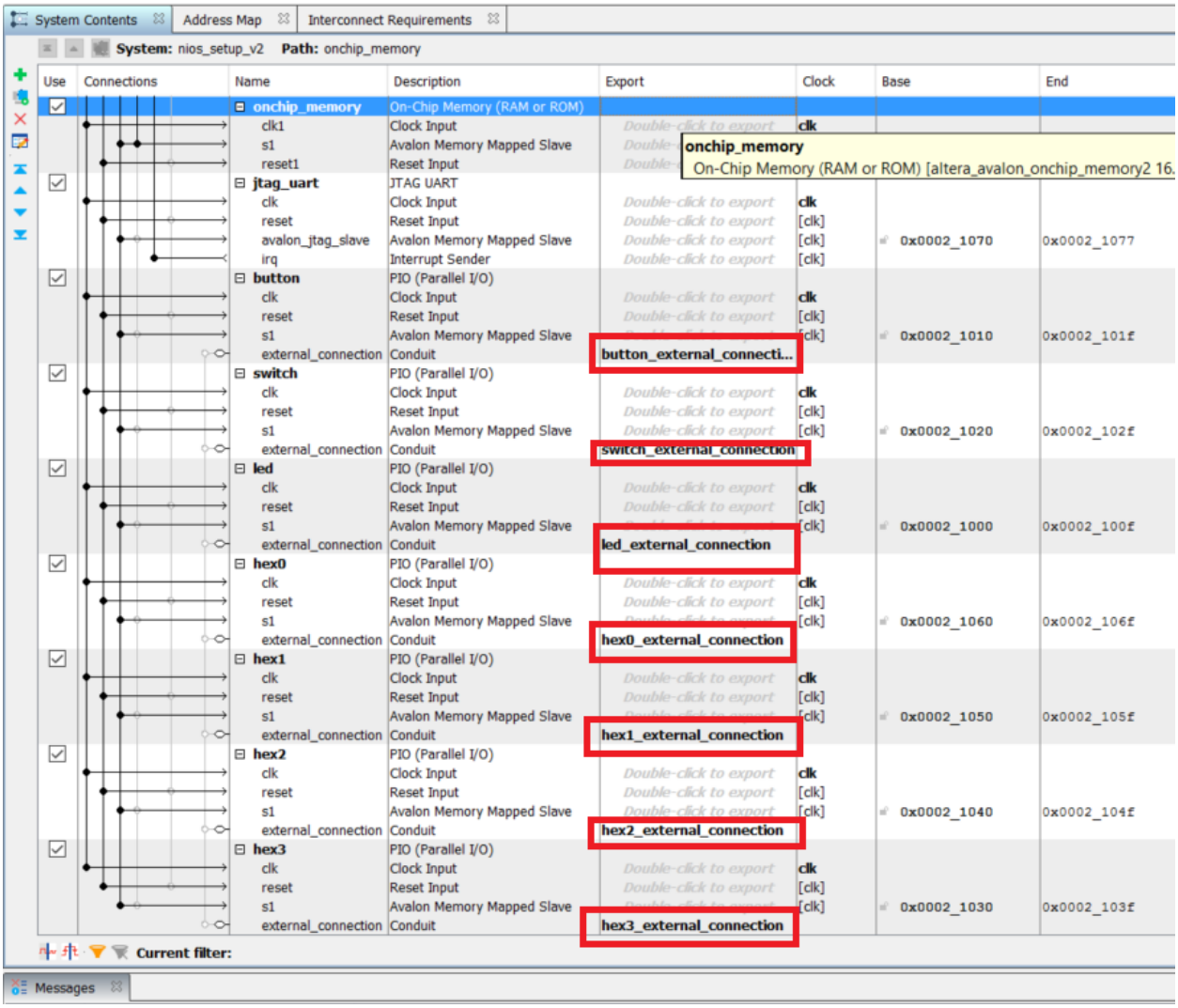


Figure 23: System Contents after Exporting PIO Switch and LED

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

18. Save your Qsys system by using ***File → Save As*** and pick a name for the Qsys system that you will remember. The information is saved in a (**.qsys**) file.
    a. Note that the lab figures call it **nios_setup_v2** so to avoid confusion you should name your (**.qsys**) file the same.

Although you are not finished, it is good practice to save edits along the way.

You should see two error messages in the Message Console of Qsys.

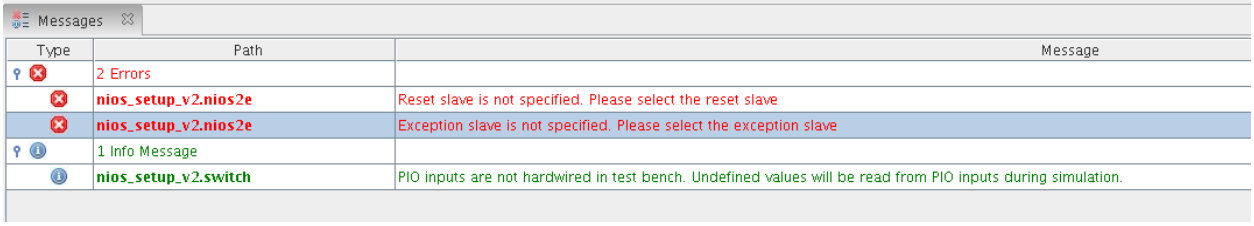


Figure 24: Error Message Prior to Assigning the CPU Memory Location to Execute From

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

19. Double click on the **cpu** component and select the **Vectors** tab.
20. Set the **reset vector memory** and **exception vector memory** both to **onchip_memory.s1**.
    a. Both the data master and the instruction master form the cpu need to be connected to the S1 port of the onchip memory for this to work.
    b. See Figure 29 below for example.

This will set the system to execute from **onchip memory** at these respective locations upon reset or interrupt. The two errors that were shown in Figure 24: Error Message Prior to Assigning the CPU Memory Location to Execute From should now be resolved. (If you don't have the option to select onchip_memory.s1 double check your Qsys connections to the on chip memory S1 port)

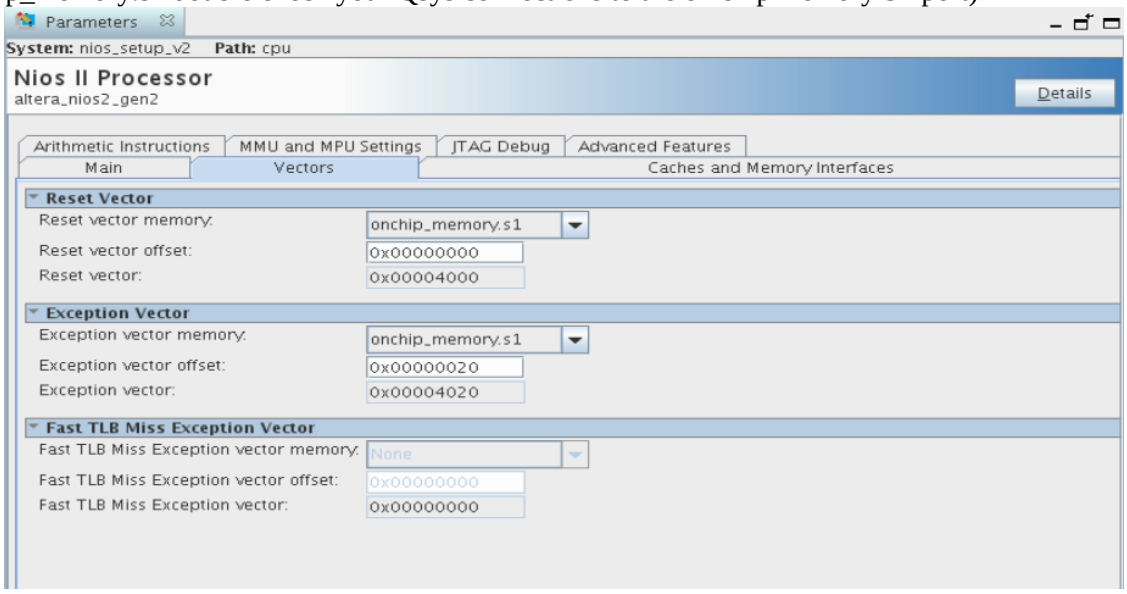


Figure 25: Assign Vectors in the Nios II Parameters Panel

21. Save your design once again.
    a. Note that by saving, you still have not generated the files that you need for Quartus II compilation or with the Eclipse SBT.

22. Click on the button '**Generate HDL**'. A screen like Figure 26 should appear.
23. Click **Generate** on the panel that appears.
24. When the file generation is complete, click **Finish** to exit the Qsys window.

Congratulations! This completes the Qsys section of the lab.

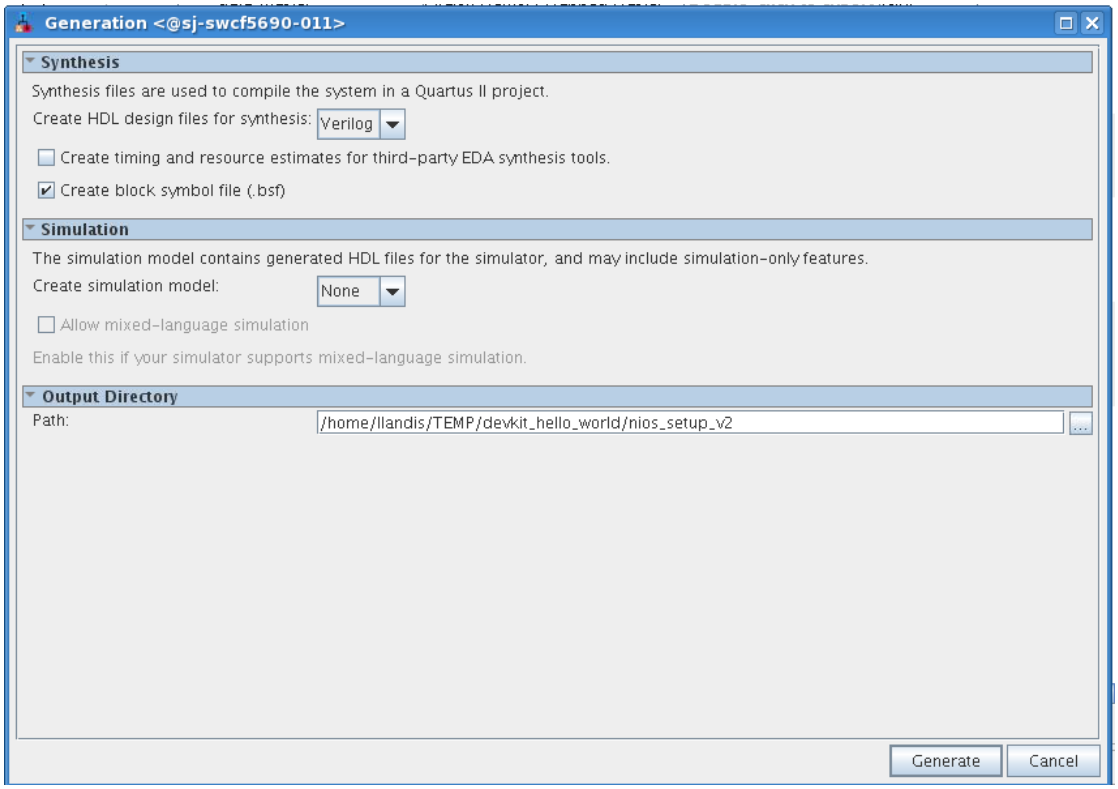


Figure 26: Screen that Appears after "Generate HDL"

3.0: Building the Top Level Design

The next step is binding together your Qsys system with Verilog code.

Quartus should be open, bring that to the front of your screen. Note that for this design there is a clock, reset, push button inputs, switch inputs, LED outputs, six HEX outputs (the seven-segment displays), 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.

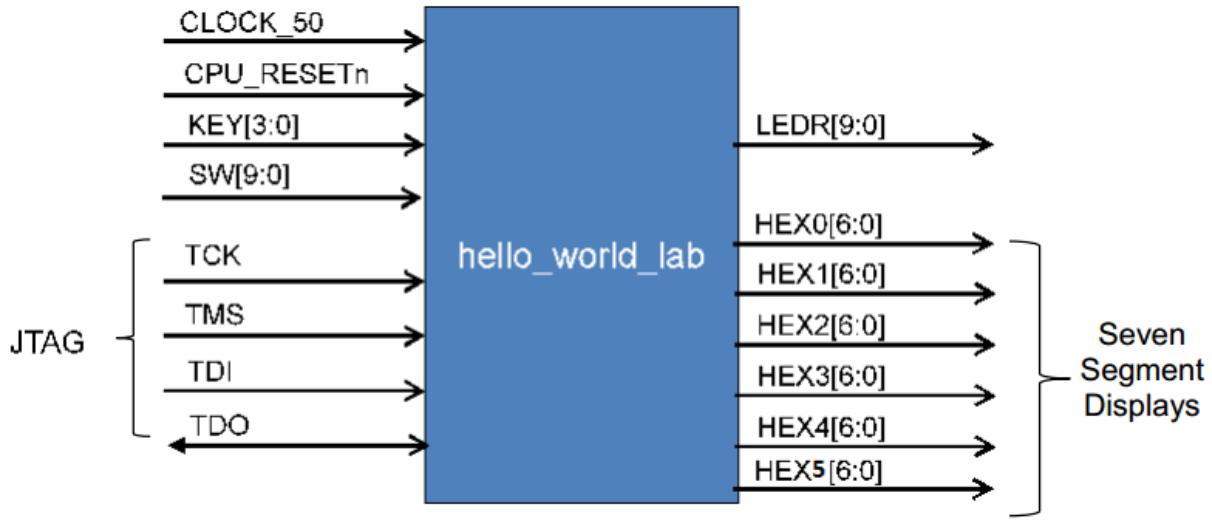
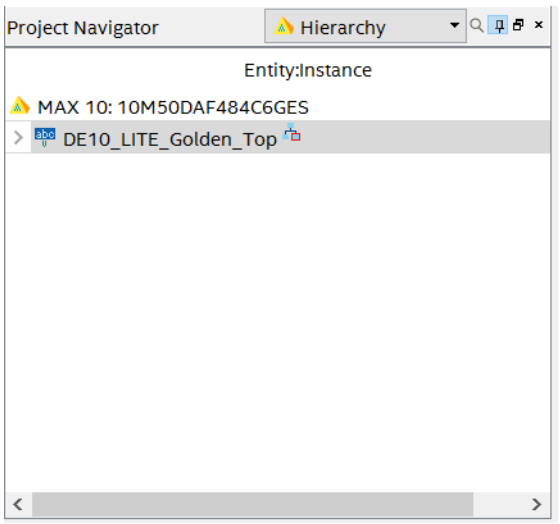


Figure 27: Block Diagram of hello_world_lab Design for the CV_GX Development Kit

1. The top-level entity is called **DE10_LITE_Golden_Top.v**. You can see it by double clicking on "DE10_LITE_Golden_Top.v" under the Project Navigator section.



The "**DE10_LITE_Golden_Top.v**" code connects the pushbutton inputs to the LED outputs in software. Keep in mind that the clock, reset, push button, and LED pin names need to reflect the names for the DE10-Lite Development Kit.

If you were wondering how to hook up the nios_setup_v2 module yourself, you can check **nios_setup_v2_inst.v**, which was auto-generated from **nios_setup_v2.qsys** inside the *nios_setup_v2* directory of your project. Open this file and you see how to instantiate the Qsys system. The contents of this file are shown in Figure 28.

```
72  nios_setup_v2 u0 (
73     .clk_clk (MAX10_CLK1_50), //clk.clk
74     .led_external_connection_export (ledFromNios[9:0]), //led_external_connection.export
75     .reset_reset_n (1'b1), //reset.reset_n
76     .button_external_connection_export (keyMod[3:0]), //button_external_connection.export
77     .switch_external_connection_export (SW[9:0]), //switch_external_connection.export
78     .hex0_external_connection_export (HEX0), //hex0_external_connection.export
79     .hex1_external_connection_export (HEX1), //hex1_external_connection.export
80     .hex2_external_connection_export (HEX2), //hex2_external_connection.export
81     .hex3_external_connection_export (HEX3), //hex3_external_connection.export
82     .hex4_external_connection_export (HEX4), //hex4_external_connection.export
83     .hex5_external_connection_export (HEX5) //hex5_external_connection.export
84  );
85
86
87  //Uncomment the line below by deleting the "//"
88  //assign LEDR[0] = SW[0];
```

Figure 28: Contents of nios_setup_v2_inst.v

We need to specify the top-level entity of our project and the add the Verilog code generated by the Qsys system we just created to the project.

2. In the top file (DE10_LITE_Golden_Top) **uncomment line 88** by deleting the "//" at the beginning of the line.
    a. By uncommenting this line, we directly drive led 0 on the board with switch 0 through the FPGA hardware. No software is required for this led to operate.
3. In the Quartus main window, go to *Project → Add/Remove Files in Project*.
4. Add the **nios_setup_v2.qip** files. (You can also just add the nios_setup_v2.qsys file)
    a. The ***nios_setup_v2*.qip** file should be found under *nios_setup_v2 → synthesis* directory in your project
    b. You will need to change the filter to display "all files" if you cannot see it.

The (**.qip**) file contains the information for the Nios II Qsys system that we created in the last step. The (**.v**) file connects the Qsys system we made to the inputs and outputs of our board.

5. Click *Apply* once you have added the file.

See Figure 29 for what your Add/Remove Files window should look like. (There may be an extra .sdc file in the list. If this shows up it is ok.)

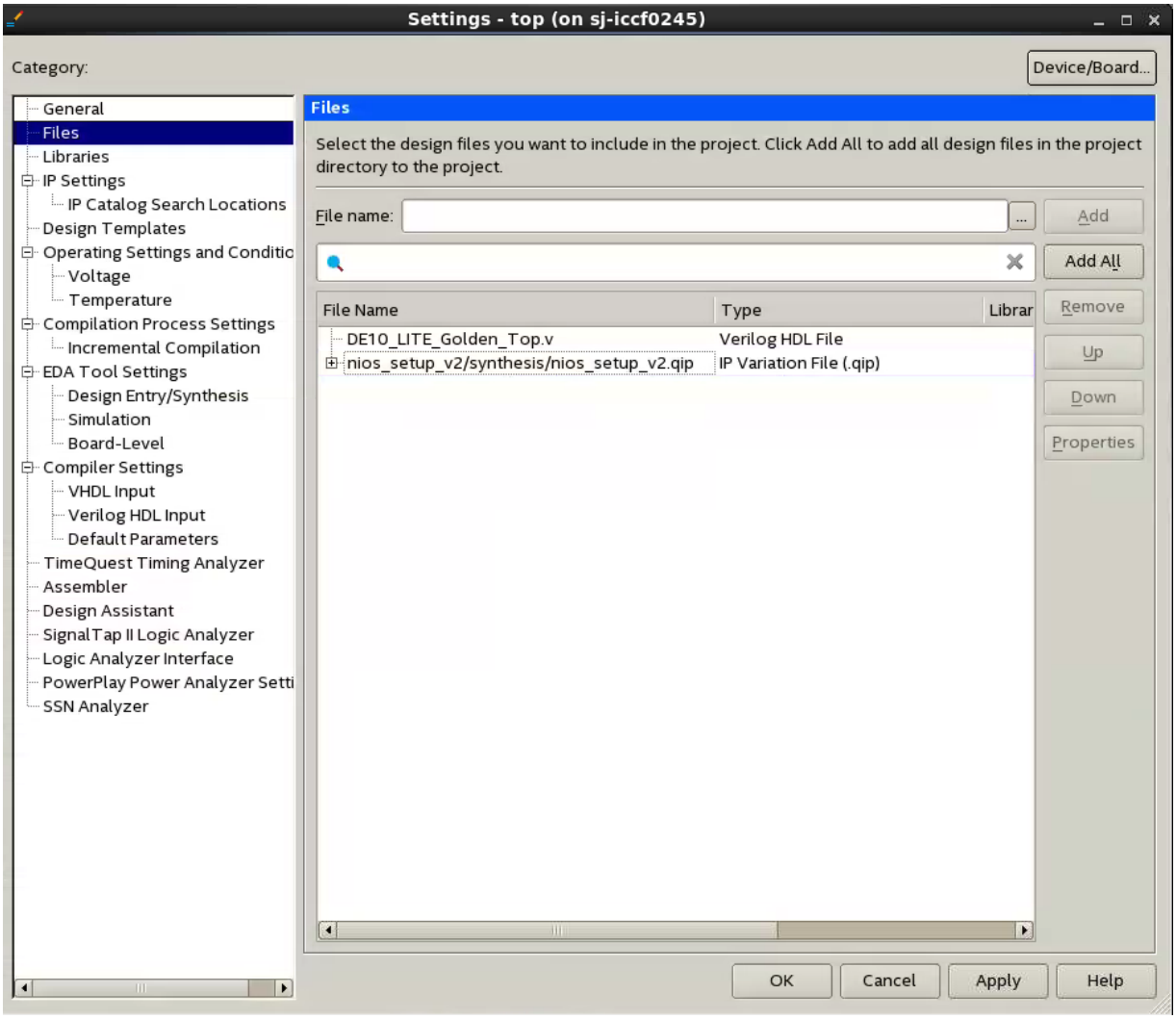


Figure 29: Quartus Add/Remove Files Pane

Almost there! We have pre-included and setup the pin assignments for the DE10-Lite development kit for you so you do not have to manually set dozens of pins using the pin planner. These commands handle routing the pins and voltage levels so they can be easily transferred between projects that use the same board.

6. To view the pin assignments, go to *Assignments* → *Assignment Editor*.

Figure 30 below is what the **Assignment Editor** window should look like. After compiling your design, the blue diamonds with question marks inside should change to show whether those pins are inputs or outputs.

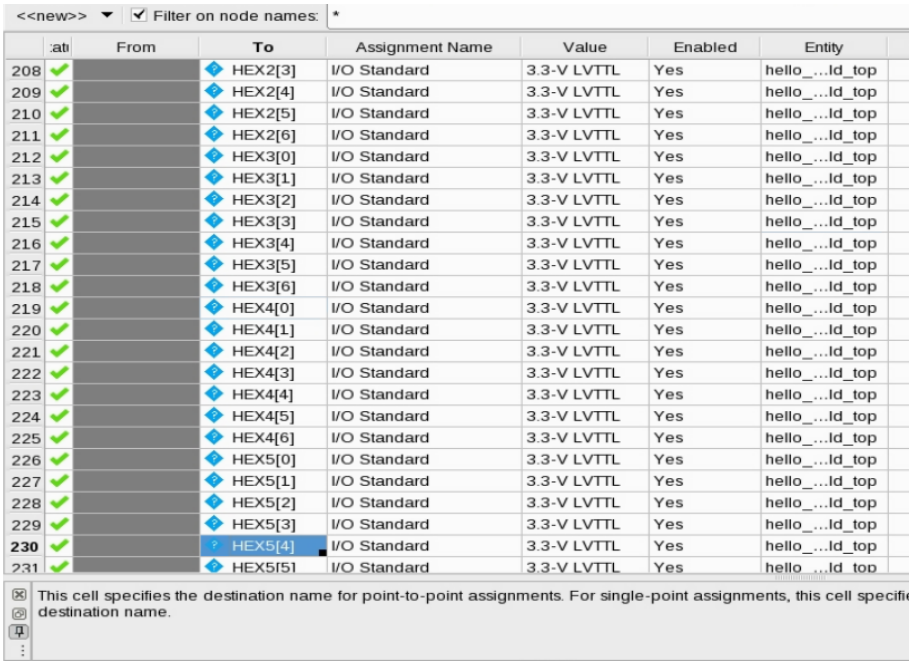


Figure 30: Quartus Assignment Editor Window

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).

7. Click on the play button as shown in Figure 31

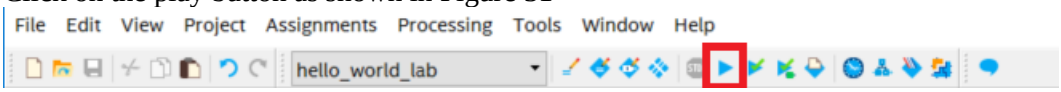


Figure 31: Compilation Button on Quartus Toolbar

Note that some warnings and information messages come up in the bottom window. You can filter by message level. The errors are filtered with the  button, critical warnings with the  button, warnings with the  button, and informational messages with the  button. You cannot proceed if you have errors. In this case, there are only critical and standard warnings, 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. Now we will create software that will run on the board and take advantage of the Nios2 processor that we just configured.

PART 2: SOFTWARE DESIGN

1.0: 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: **nios_setup_v2.sopcinfo** and **top.sof** as if you generated them from the Hardware Design lab. You will be able to complete all subsequent steps with these two files.

The NIOS Software Build Tools for Eclipse are included as part of Quartus. These tools will help manage creation of the application software and Board Support Package (BSP).

1. Launch the *SBT Tools → NIOS II Software Build Tools for Eclipse* (*Tools -> Nios II Software build tools for Eclipse*). You can use the default location that Eclipse picks for you.

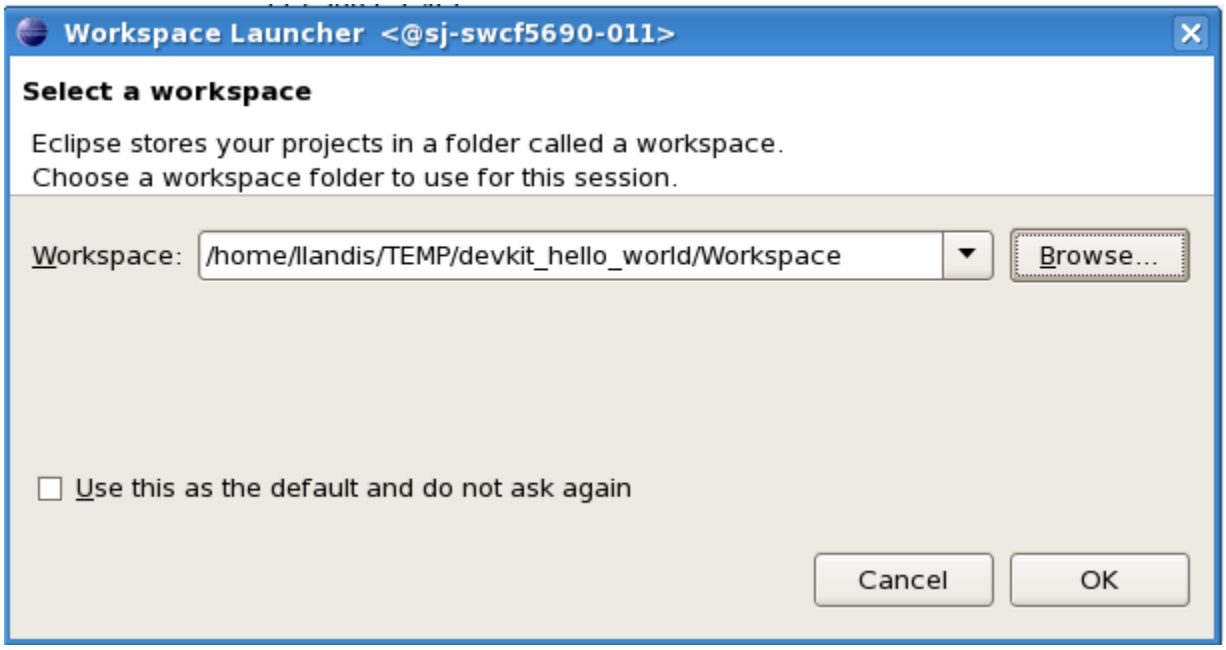


Figure 32: Initial Workspace Setup

2. Click **OK** in the Workspace launcher.

Next, the Eclipse SBT will launch.

3. Right click in the area called Project Explorer and select *New→Nios II Application and BSP from Template*.

   The BSP is the "Board Support Package" that contains the drivers for things like translating "printf" C commands to the appropriate instructions to write to the terminal.

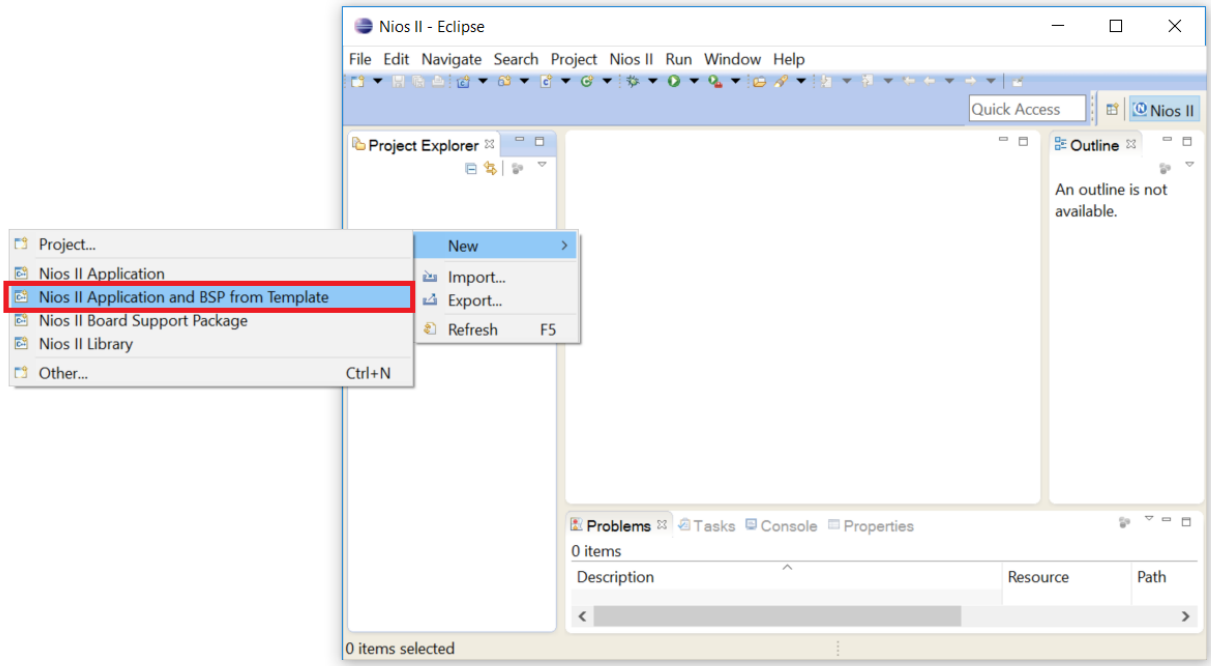


Figure 33: Creating the Initial Project in the Eclipse SBT

Next you will see a panel that requests information to setup your design.

4. Navigate to your working directory and click on the (**.sopcinfo**) file. The (**.sopcinfo**) file informs Eclipse on what your Qsys system contains.
5. Click OK.

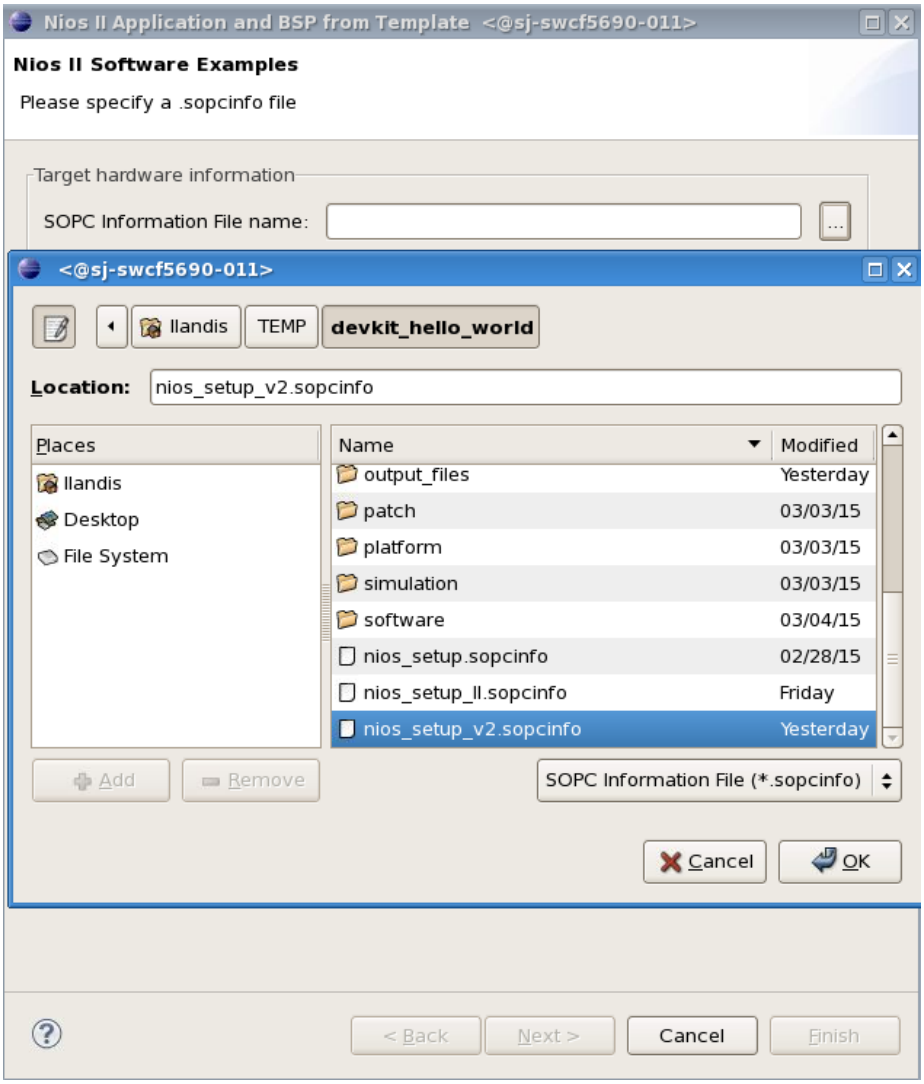


Figure 34: Navigating to the '.sopcinfo' File

6. Fill in the Project name, call it hello_world_sw.
7. Next you will be asked to pick a template design. Select the The *Hello World* application template. This template writes "*Hello from Nios II*" to the screen.
    a. 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.
8. Click **Finish**.

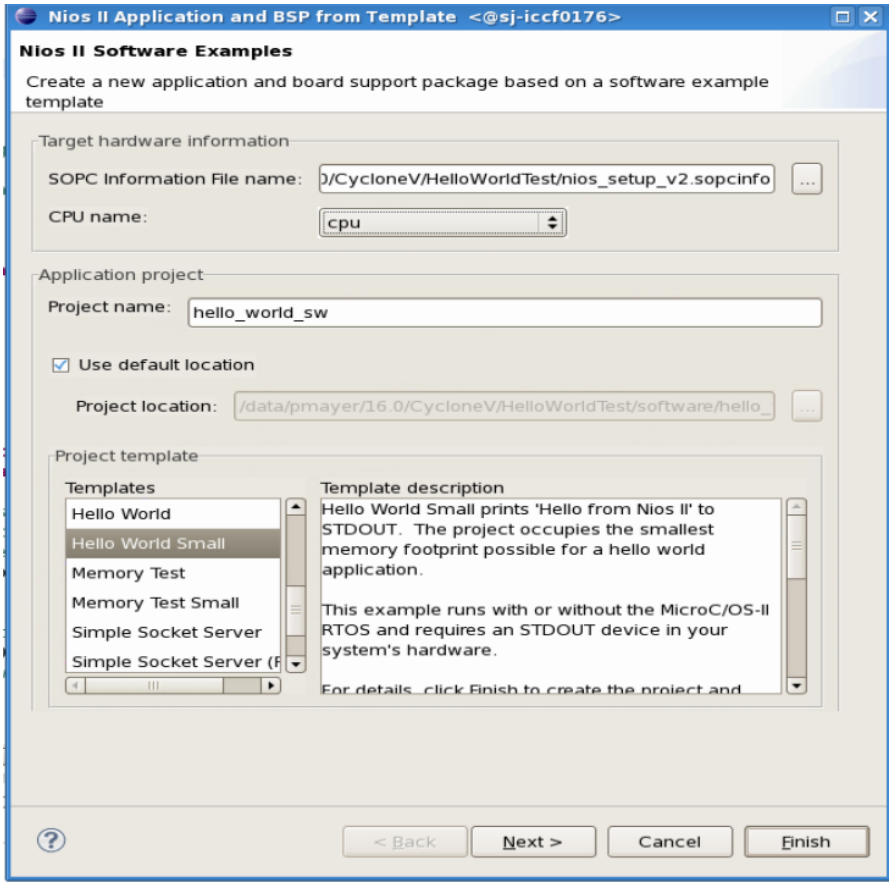


Figure 35: Completing the Nios II Software Examples Setup Screen with Project Name and Project Template

We will now make some modifications to the code to display the results of the pushbuttons (KEY1-0) on LEDs 3-2.

9. Click the right arrow next to *hello_world_sw*. It will show the contents of your project. Double-click **hello_world_small.c**.

Note the command *alt_putstr* to write text to the terminal. This is part of the Altera HAL (Hardware Abstraction Layer) set of software functions. Note that the *alt_putstr* command is used versus a standard C printf function because the code space is more compact using the HAL commands. Code using HAL functions without an operating system is referred to as "bare metal" programming. 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 .

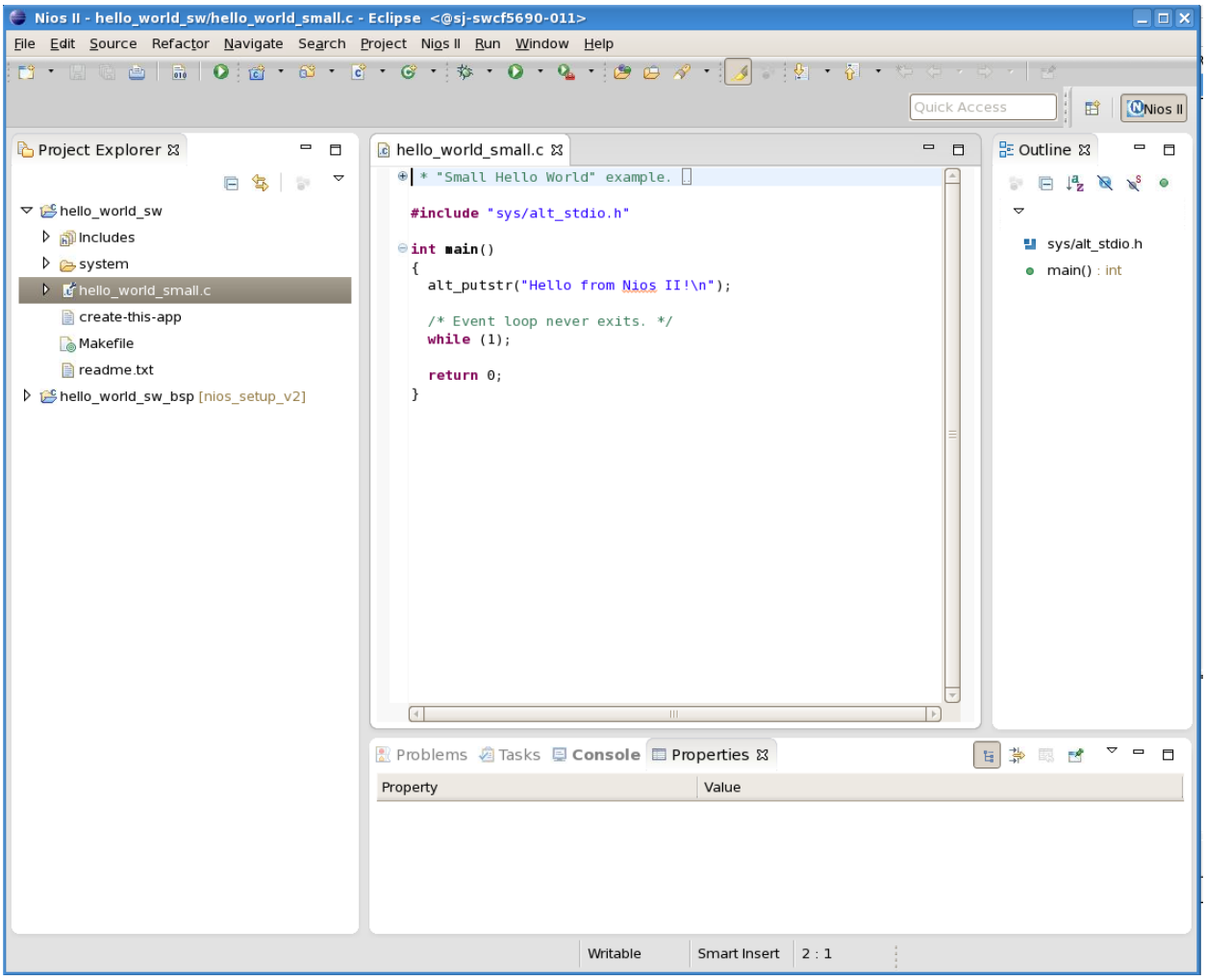


Figure 36: Eclipse Window of "hello_world_small.c"

Next you need to add a library declaration, define integer switch_datain, and a few HAL functions to connect the LEDs to the Push Buttons.

10. Drag and drop the file "**DE10_hello_world_code.c**" found in the subfolder *DE10_C_CODE* in the folder *DE10_qsys_workshop*, into **hello_world_small.c** located in Eclipse.
11. Delete the pre-made **hello_world_small.c** file in your *hello_world_sw* folder in the **Eclipse Project Exploerer**. This can be done by right clicking on **hello_world_small.c** and selecting **Delete** from drop down menu that appears.

The code may appear somewhat cryptic, so we will now take the time to explain what the various lines do. IOWR_ALTERA_AVALON_PIO_DATA(Location)gets the data from the specified Location (Given in the system.h file under the hello_world_sw_bsp folder) and reads it into a variable. Calling the function with two parameters, as in: IOWR_ALTERA_AVALON_PIO_DATA(Location, Value) writes the numeric Value to the given Location. Notice how we are using this function to read the data from the switches and then write this value to LEDs.

Note the use of the variables BUTTON _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 BUTTON_BASE and LED_BASE. These must match your hello_world_small.c code.

12. Click the save icon.

13. Now that we have written our code, right click on the *hello_world_sw* in the Project Explorer and click on **Build Project**. This compiles the software application and the BSP (drivers).

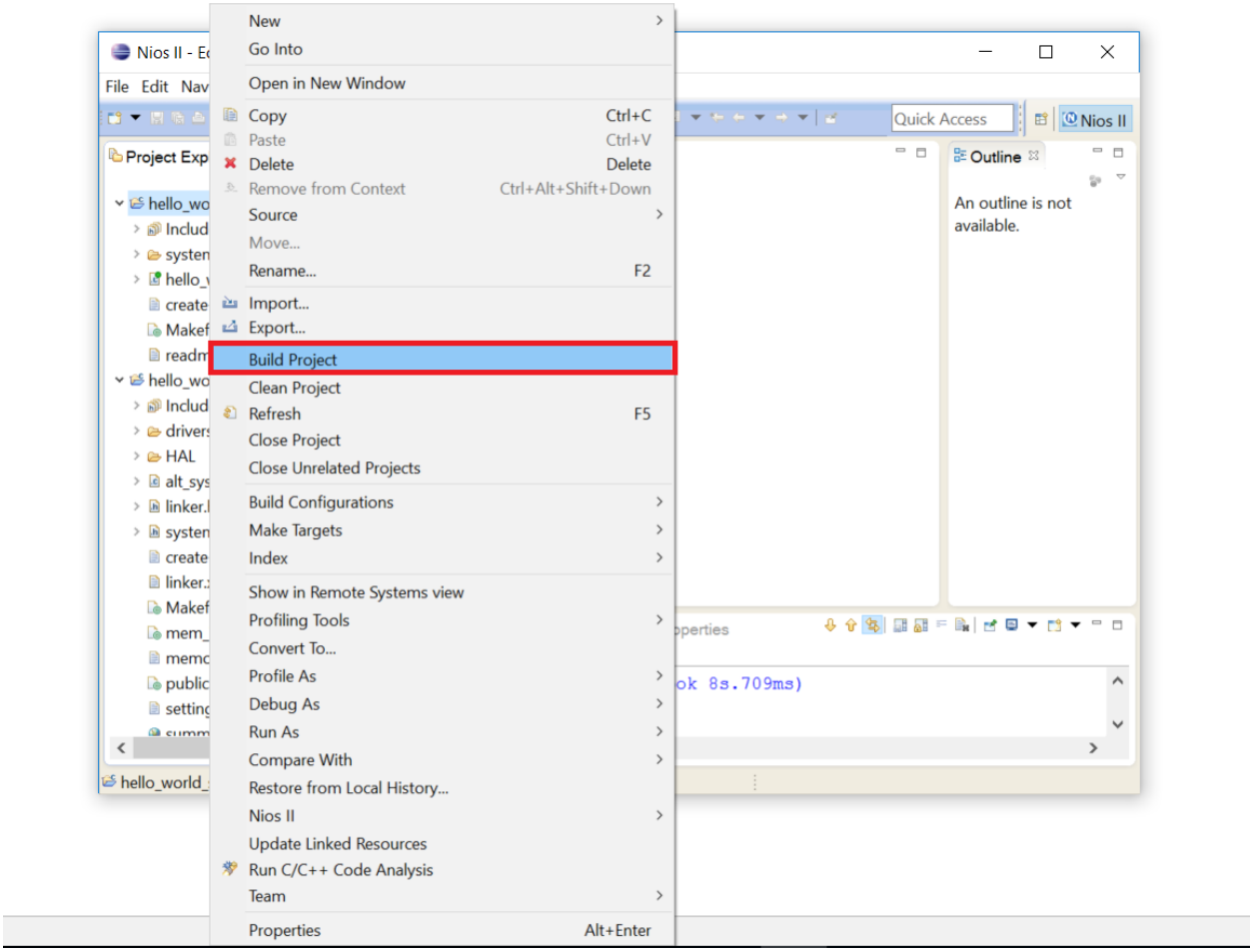


Figure 37: Building the "hello_world_sw" Project

14. Once the build completes, you should observe an **.elf** file (executable load file) under the **hello_world_sw** project. 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, correct, and rerun **Build Project**. Typical problems can be missing semicolons, mismatched brackets and such.

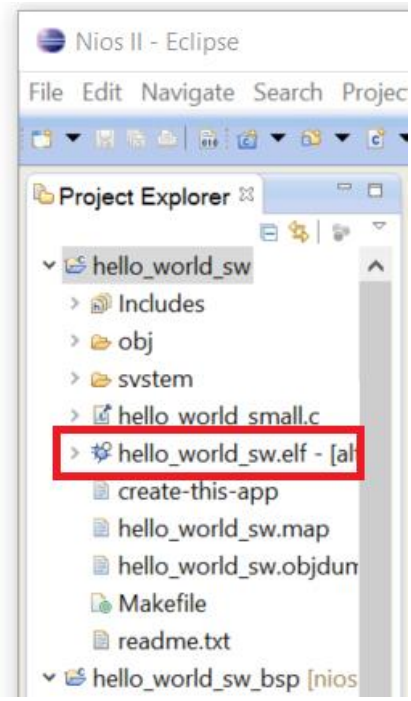


Figure 38: The Presence of "hello_world_sw.elf" Indicates if the Software Build Ran Successfully

2.0: Downloading the Hardware image to the MAX10

If you have never used the USB blaster before, you will need to follow these steps to update your USB blaster's driver software. If you have used the USB blaster before on your computer, you may skip this portion of the manual. To work with the Max10 in the context of this lab, you will need to connect a USB cable connecting the kit to a host PC. The USB blaster utilizes circuitry that formats the image into a data stream that downloads from the PC to FPGA.

To install the USB Blaster, follow these steps

1. To begin, make sure you connect your board to your computer via a USB cable. (The board also needs to be powered on)
    a. Be sure to power your board as well. See Figure 46 for where to connect a USB cable to your Max10 kit.
2. Hit the windows key  and type "**device manager**"
3. Click on the device manager tile that appears.
4. Navigate to the "other devices" section of the device manager and expand the section.
    a. Figure 39 depicts what the Device Manager will look like when the USB Blaster Drivers are not installed.

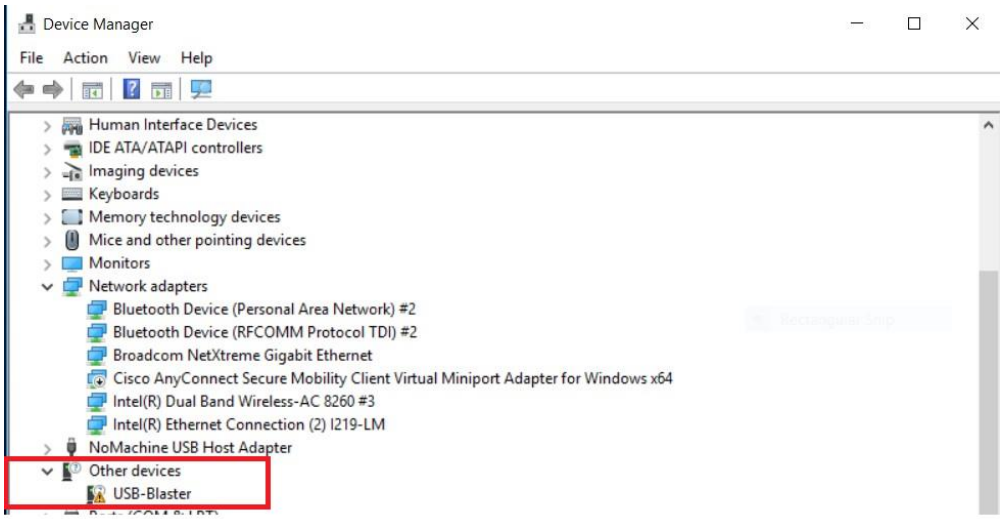


Figure 39: Device Manager Showing USB Blaster Drivers not Installed

5. Right click the USB-Blaster device and select "**Update Driver Software**".
6. Choose to browse your computer for driver software. See Figure 40

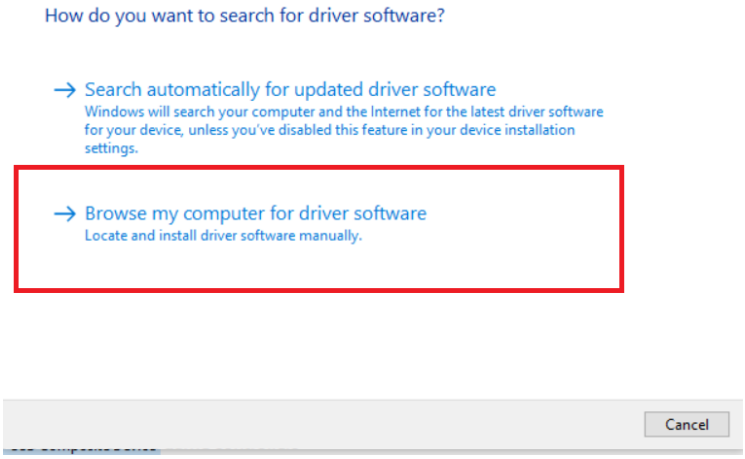


Figure 40: Selecting to Browse for Driver Software Directory

7. Navigate to the path shown in Figure 41

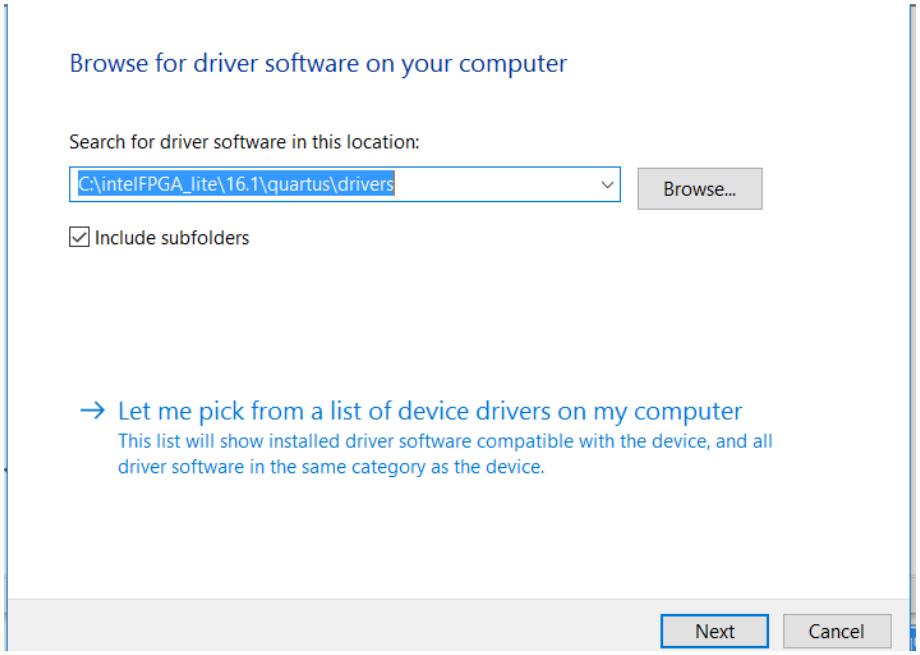


Figure 41: Directory Containing USB Blaster Drivers

8. Once you have the proper file path selected, click on "**Next**" and the driver for the USB Blaster should be installed.

9. With the USB blaster drivers properly installed, launch the Programmer: *Tools → Programmer*.

   Next, you need to download what is called a (**.sof**) file or SRAM object file. This is the programming image file that gets downloaded in the FPGA. The default location is *<working_directory>/output_files.*

10. Right click on the first row <none> under File and click on **Change File**. Navigate to the *output_files* directory and select **top.sof**.
11. Click **Open**.
12. In the first row under **Program/Configure** click in the check box as shown in Figure 42: Programmer Panel with Program/Configure Checkbox

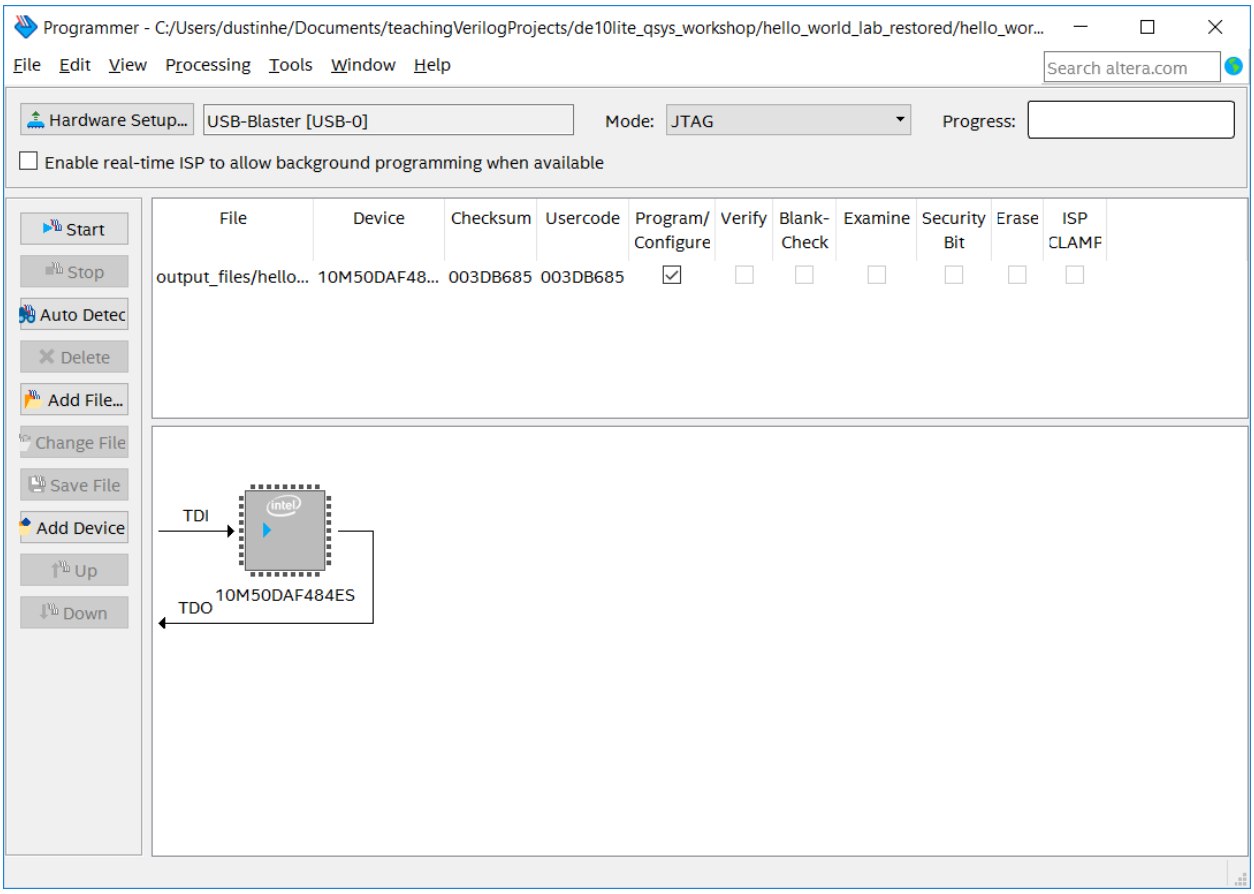


Figure 42: Programmer Panel with Program/Configure Checkbox

13. Click on Hardware Setup, located in the top left corner of the programmer window. In the Currently selected hardware section, click on the drop-down menu and select the *USB Blaster*.

14. Click **Start**, located on the left of the programmer window. When programming is complete, the Progress meter should read 100% (Successful).

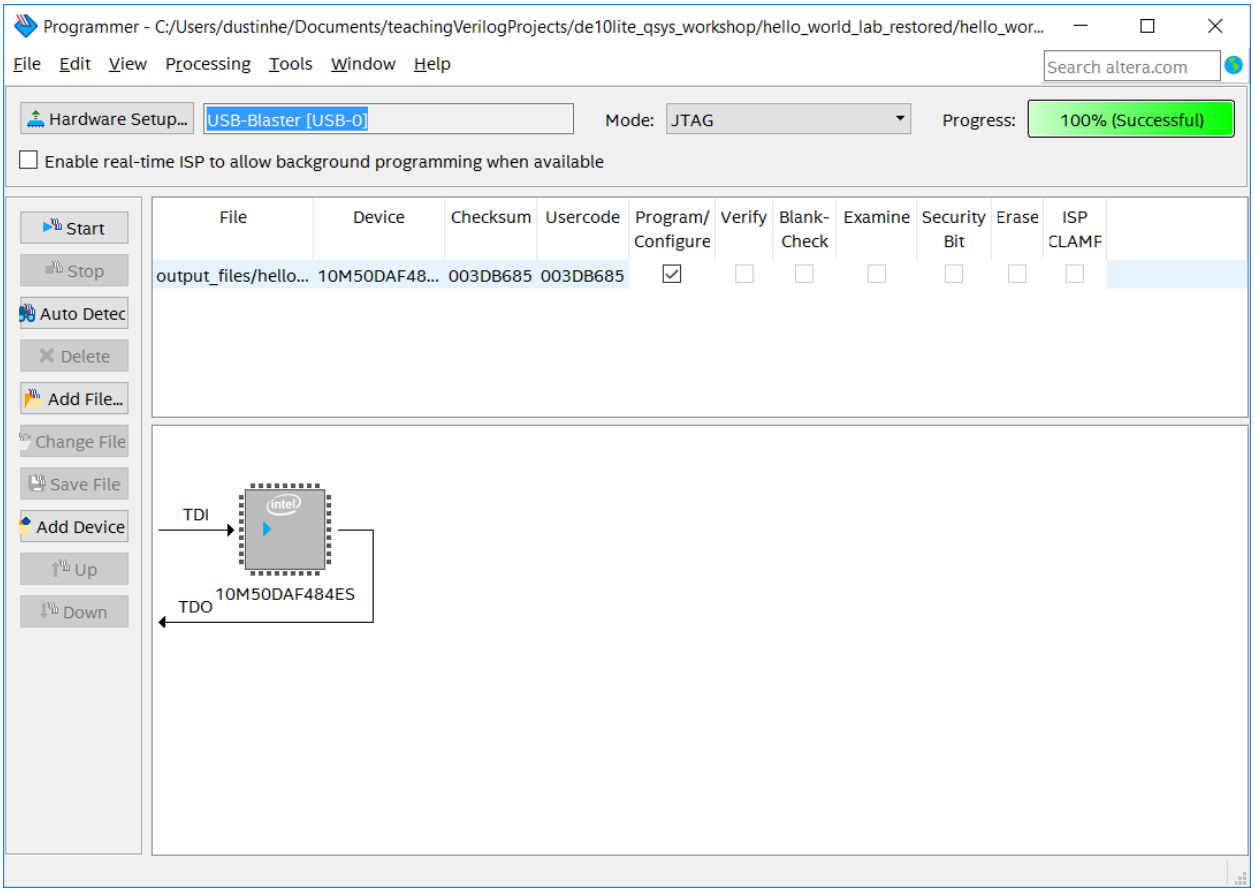


Figure 43: Programmer Progress Successful

Now that the FPGA is programed the hard ware is operating. However, we have not programed the software for the NIOS CPU yet. To demonstrate the hardware is functioning but the software is not press the keys and switch SW1 to on (towards the LEDS). You should see only one LED light up. Follow steps 15-18 below then try pressing the keys again. Note how the hardware driven LED does not need the software executable file (.elf) to operate.

Now it is time to download the (**.elf**) (software executable) into the Nios IIe processor.

15. Return to the Eclipse SBT tools. Right click on **hello_world_sw** and select *Run as → Run Nios II Hardware*. A window should appear as shown below.
16. Click on the Target Connection tab.
    a. The connection should indicate that Eclipse has connected to the USB-blaster.
    b. If the connection is not identified, you can click *Refresh Connections*.
    c. **Note that you might need to stretch the window wider to see the Refresh Connections button**.
17. Once the connection is made to the USB-Blaster, you should observe something like Figure 44.
18. Click **Run**.
    a. If the run button is greyed out but your device shows up under the connections window, you may need to select "**Ignore mismatched system ID**" and "**Ignore mismatched system timestamp.**"

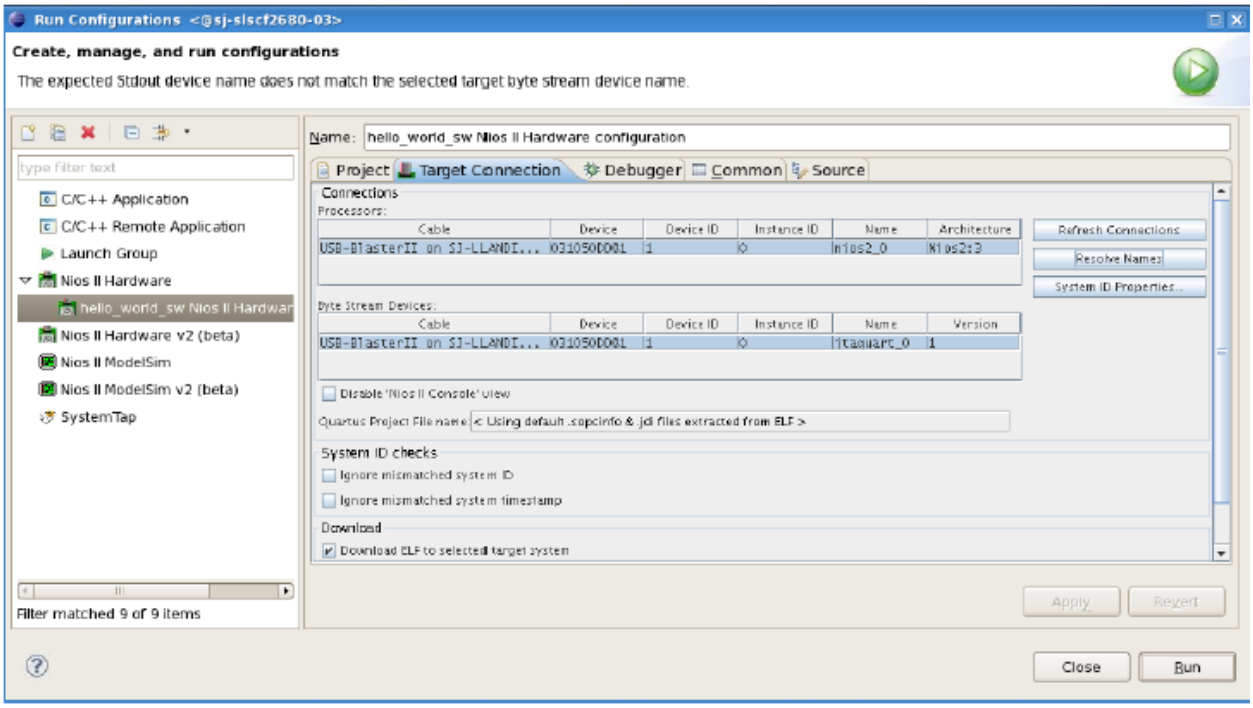


Figure 44: Eclipse SBT Tools after Connection is made to the USB-Blaster

19. Now you have hardware and software downloaded into your board. You should observe "Hello from Nios II!" printed on the Nios II Console tab.

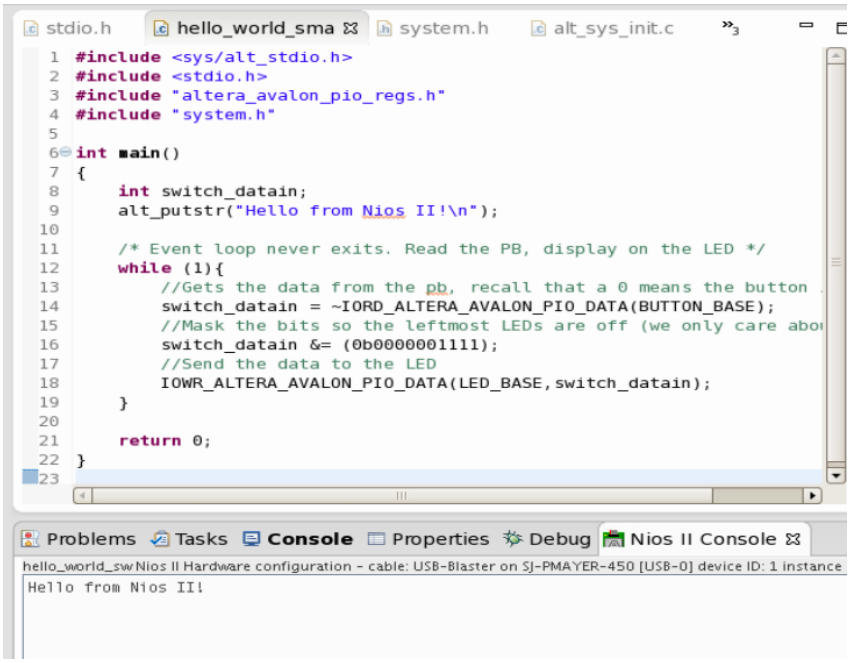


Figure 45: "Hello from Nios II!" on Nios II Console Tab

20. You can also test the connections between push button and LEDs. Push buttons 0-1 should now turn LEDs 2-3 on when pressed.
    a. The pushbuttons and LEDs were connected through our Qsys system and the C code we have running on our development kit.

3.0: Using the Seven-Segment Display:

One of the nice things about the Nios2 processor is that since we have already designed the hardware, we can now change the software without having to reprogram the FPGA. We will now program the Nios2 processor to display text on the seven-segment displays and make pushbuttons speed up and slow down the text.

1. Drag and drop the file named **seven_seg_dsplay_DE0.c** into the *hello_world_sw* project folder in Eclipse.
    a. **seven_seg_dsplay_DE0.c** can be found in the *DE10_C_CODE* subfolder in the *DE10_qsys_workshop* folder.
2. Remove the file **DE0_hello_world_code.c** by right clicking on **DE0_hello_world_code.c** and selecting **Delete.**
3. Right click on the *hello_world_sw* in the Project Explorer and click on **Clean Project.**
4. When the program is finished, right click on *hello_world_sw* again and select **Build Project**.
5. Once the build completes, the (**.elf**) file under the **hello_world_sw** project should be updated.
    a. To check, right click on the (**.elf**) file and go to *Properties*. The time under the "Last Modified" section should reflect the time the last build was completed.
6. Right-click on the **hello_world_sw** folder in the project explorer on the right and select *Run as* → *Run Nios II Hardware*.

    a. This will run the new C program on the Nios2 processor.
7. Now a prompt should appear in the console telling you to enter text. Type something like “Hello World” into the console and press ENTER. The text should appear on the seven-segment display.
8. You can control the text in the following manner using the following switches and buttons:
    a. Press KEY0 to reverse the direction the text scrolls
    b. Press KEY1 change text (look at the console for further instructions)
    c. SW9 pauses and un-pauses the display.
    d. SW8 Controls the direction the letters cascade.
    e. SW7 controls the direction the letters face.
        i. If SW7 is down, the letters will be right-side up. If SW7 is up, the letters will be upside-down.
    f. SW6 mutes the display.

If you are fluent in C, try modifying the program to add functions for some of the other switches (SW5-1). When modifying, or writing your own program, the variable switch_datain is assigned the value of the switches.

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 development kit.
11. Downloading the software executable into the development kits.
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.