Embedded Design with the Nios II Processor using HOL 3101 Lab

Version: [1.0]

Last updated: [02 Aug 2021]

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.

Copyright © 2021, Intel Corporation. All rights reserved.

Contents

[1.0 Introduction 4](#_Toc78895395)

[2.0 Contents 5](#_Toc78895396)

[3.0 Lab Notes 6](#_Toc78895397)

[4.0 Accessing your lab PC 8](#_Toc78895398)

[5.0 Design Flow 9](#_Toc78895399)

[6.0 Objective of “Hello World” Lab 10](#_Toc78895400)

[7.0 Get started with Quartus 11](#_Toc78895401)

[8.0 Part 1: Hardware Design 13](#_Toc78895402)

[8.1 Lab 1: Building Your Platform Designer Based Processor System 13](#_Toc78895403)

[8.1.1 Adding the Nios II Processor 13](#_Toc78895404)

[8.1.2 Adding On Chip Memory 15](#_Toc78895405)

[8.1.3 Adding the JTAG UART Component 18](#_Toc78895406)

[8.1.4 Adding Parallel IO (PIO) 19](#_Toc78895407)

[8.1.5 Connecting the System Components Together 23](#_Toc78895408)

[8.2 Building the Top Level Design 34](#_Toc78895409)

[9.0 Part 2: Software Design 41](#_Toc78895410)

[9.1 Lab 1: Creating the Software for the “Hello World” design 41](#_Toc78895411)

[9.2 Lab 2: Downloading the Hardware Image to the Development Kit 47](#_Toc78895412)

[9.3 Lab 3: Using the Seven Segment Display 52](#_Toc78895413)

[10.0 Lab Summary 54](#_Toc78895414)

[10.1 Revision History 55](#_Toc78895415)

# Introduction

This lab teaches you how to create an embedded system implemented in programmable logic using the Intel® Nios II processor, sometimes referred to as a ”soft” processor. The Nios II can be synthesized on any Intel® FPGA device, and has a built in programmable logic fabric that can be easily modified to suit an applications’ requirements. Intel® SoC FPGA devices contain a processor built from standard cells that cannot be changed without redesigning the chip, and are therefore called a hard processor system. The Nios II processor is supported by a rich set of peripherals and intellectual property (IP) blocks that can be configured and connected to the processor using the Platform Designer tool within the Intel® Quartus Prime software suite. Intel also distributes the Nios II Software Build Tools (SBT) within the Quartus download to use with Eclipse\* during software development.

This lab is organized to run on a number of Intel® FPGA development kits. The links to the other kits’ Design Examples can be found in the Design Store by typing “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 the text on your terminal. The initial section of the lab is split into a hardware section and a software section.

# Contents

Contents

[1.0 Introduction 4](#_Toc78889377)

[2.0 CONTENTS 5](#_Toc78889378)

[3.0 LAB NOTES 6](#_Toc78889379)

[4.0 Accessing your lab PC 8](#_Toc78889380)

[5.0 DESIGN FLOW 9](#_Toc78889381)

[6.0 OBJECTIVE OF THE ”HELLO WORLD” LAB 10](#_Toc78889382)

[7.0 GET STARTED WITH QUARTUS 11](#_Toc78889383)

[8.0 PART 1: HARDWARE DESIGN 13](#_Toc78889384)

[8.1 Lab 1: Building Your Platform Designer Based Processor System 13](#_Toc78889385)

[8.1.1 Adding the Nios II Processor 13](#_Toc78889386)

[8.1.2 Adding On Chip Memory 15](#_Toc78889387)

[8.1.3 Adding the JTAG UART Component 18](#_Toc78889388)

[8.1.4 Adding Parallel IO (PIO) 19](#_Toc78889389)

[8.1.5 Connecting the System Components Together 23](#_Toc78889390)

[8.2 Building the Top Level Design 34](#_Toc78889391)

[9.0 PART 2: SOFTWARE DESIGN 41](#_Toc78889392)

[9.1 Lab 1: Creating the Software for the “Hello World” design 41](#_Toc78889393)

[9.2 Lab 2: Downloading the Hardware Image to the Development Kit 47](#_Toc78889394)

[9.3 Lab 3: Using the Seven Segment Display 51](#_Toc78889395)

[10.0 LAB SUMMARY 53](#_Toc78889396)

[10.1 Revision History 53](#_Toc78889397)

# 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 THE FILE NAMES OR DIRECTORIES.***
* *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.*

Quartus Prime is Intel FPGA’s design tool suite. It serves a number of functions:

* Design creation through the use of HDL or schematics
* System creation through the Platform Designer graphical interface
* Generation and editing of constraints (timing, pin locations, physical location on die, I/O voltage levels)
* Synthesis of high level language into an FPGA netlist, formally known as mapping
* FPGA place and route, formally known as fitting
* Generation of design image used to program an FPGA, formally known as assembly
* Timing Analysis
* Download of design image into FPGA hardware, formally known as programming
* Debugging by insertion of debug logic (in-chip logic analyzer)
* Interfacing to third party tools such as simulators

Webex Best Practices:

* When using Webex training center, select a PC (see lower right corner). Your name is placed below the PC name. Select your own PC, not one that is already occupied.
* If you select a PC that has a name below it, you can observe what others are doing, ok to do if agreed upon.
* If Quartus is already open when you access your machine, close it out and follow instructions step by step to launch the correct version of Quartus
* Audio connectivity has a mode where if you select “YES” when you connect to a machine, you have your own audio breakout room when you are working on your PC. Others can privately talk within your room but you lose audio connectivity to the main session. “NO” keeps your audio feed in the main session and you will hear background chatter from others in the training.
* When you leave your PC session, you can return to your PC in it’s same state. Leave your PC session to view the chat window within the main session or listen to the main audio feed.
* Send a chat message in the main session if you are having problems and someone will join your session within the breakout room audio feed. When you rejoin your PC session select YES so you have the audio feed in your breakout room.
* Open up chat within your breakout room, that way the host/TA can communicate that way with you. Look for broadcast messages to all students. Periodically check chat the main session chat and your breakout session for announcements.
* If you are TA’ing, remind the host to make you a panelist so you can see the full chat feed.

Lab Best Practices:

* You must use the 18.0 Standard version of the Intel Quartus Prime software. This version requires no license and is supported in the computer lab you are accessing. The Intel® Quartus® Prime Pro Edition software will not work as it does not support the target hardware.

# Accessing your lab PC

Launch your Webex session. You will select your own PC through Webex Training, the Webex package that enables connection to remote PCs to use throughout the duration of this training. Your instructor will guide you how to connect to a lab PC. In the lower right corner of Webex, you should see lab machines available, or alternatively select the labs pull down from the top screen of Webex. Note that an occupied machine has a person’s name *below* the machine name. Select an unoccupied PC to connect to. You will be given a choice to hear audio from the group session or in your room with your local PC you selected. Since you will be all by yourself in your PC room, you might want to continue listening to the group feed until necessary to have a debug conversation with the leader or panelist.

Once you see a Windows login prompt, your login is student and the password is QPrime.1.

Once connected, you *might* see an existing Quartus session open from a previous student. Close that session, we will start with a fresh version of **Quartus Prime** **Standard 18.0**. Using the wrong version of Quartus Prime could lead to later problems in the lab. Quartus Prime Standard supports the lower complexity FPGA devices called MAX and Cyclone. The higher complexity devices offered by Intel called Arria, Stratix and Agilex use a version of tools called Quartus Prime Pro. There is a third version called Quartus Prime Lite which is entirely free and supports MAX and Cyclone class FPGA devices. This version has less optimization and IP options than the Quartus Prime Standard but is ideally suited for university level coursework and capstone projects.

* Open the desktop folder called Quartus Shortcuts. Beneath that folder open the folder called Intel FPGA 18.0.0.614 Standard Edition.
* Launch the Quartus executable under this folder: Quartus Prime Standard Edition 18.0.0.614. The Quartus GUI will launch and occupy the entirety of your screen.
* Determine which board you are connected to. This can be achieved by launching this tool: Tools 🡪 Programmer. Next to the Hardware Setup you will see either USB-Blaster [USB-0] or DE-SoC [USB-1]. Please take note of which type of development kit your remote machine is directly connected to:

|  |  |
| --- | --- |
| Development Kit | Hardware Setup |
| Cyclone V GX Starter | USB-Blaster [USB-0] |
| DE1-SoC | DE-SoC [USB-1] |

Make note of whether you are using a Cyclone V GX Starter or DE1-SoC. If that field comes up ***blank***, immediately logout from your connected machine and try another computer. If you fail to follow the unique instructions per board you will be unable to complete the lab successfully. Let the instructor know via chat if the programmer fails to find a board.

# 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 Platform Designer graphical system design tool this is a relatively easy thing to do as we will demonstrate in this lab.

The design flow diagram below illustrates how an overall system is integrated using the combination of the Platform Designer system integration tool, Quartus for mapping (aka synthesis), fitting (aka place and route), and the NIOS II Software Build Tool (SBT) for software development.

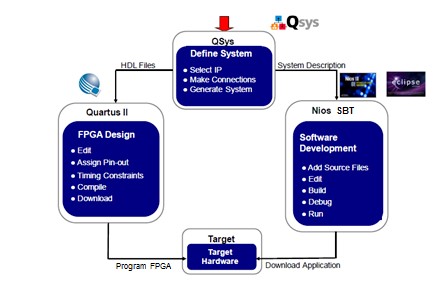


Figure 1: Platform Designer Development Flow

The above diagram depicts the typical flow for Nios II system design. Hardware System definition is performed using Platform Designer tool; the resultant HDL (.Qsys) files from the Platform Designer system are used by the Quartus design software to map, fit and download the hardware image into the FPGA device. Quartus also generates information that describes the configuration of the system designed in Platform Designer 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 “Hello World” Lab

This lab demonstrates how to use Platform Designer tool 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 hardwarecentric, 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. Intel utilizes the Platform Designer 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 Platform Designer and the Quartus Design Software, you will build the Nios II system entirely from scratch.

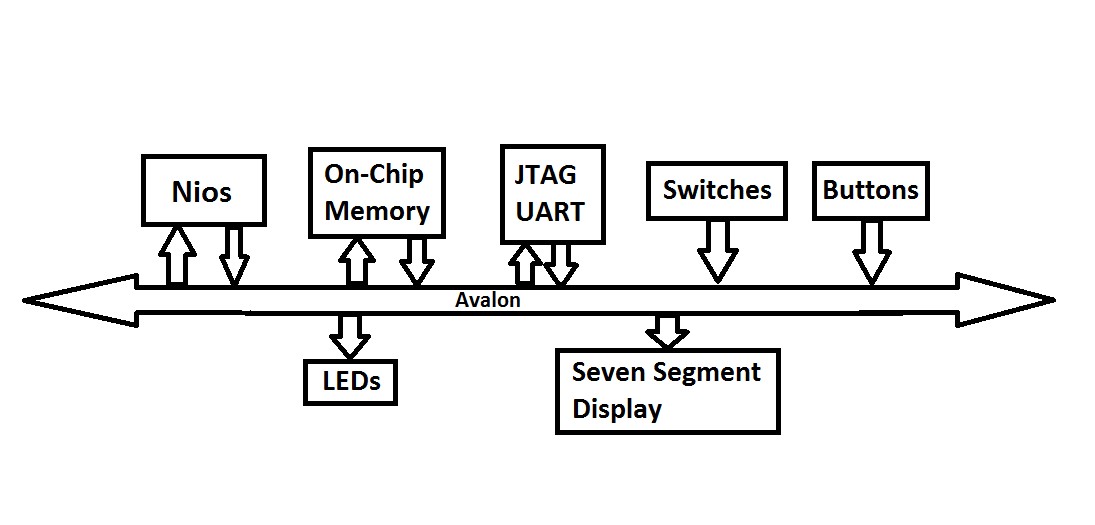


Figure 2: Nios II Based System Used In This Lab

# Get started with Quartus

This lab demonstrates how to use Platform Designer tool to design the hardware and software to print “Hello World” to your screen. This requires a working processor to execute the code.

Follow the instructions below depending on what board you have for performing the lab. Depending on the board you are using, get the associated .zip folder from the design files you downloaded for this lab. **There are two options for each board. In the short manual, the hardware design is already done for you and you need to start at Part 1 Section 2.0 Step 7 and compile your design with the “play” button and then proceed to the Part 2: Software design. This particular system will save roughly an hour of work in completing your lab, but is less impactful in understanding the entire flow.**

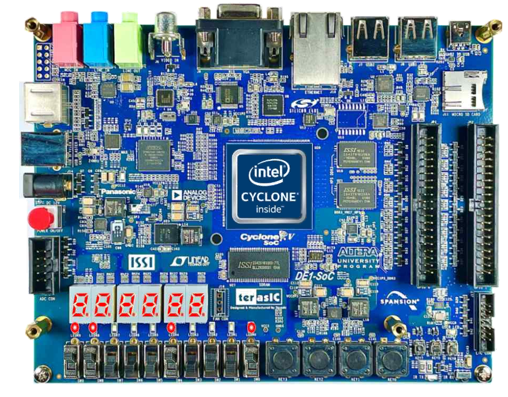


Figure 3: DE1-SoC development kit

Unzip the EmbeddedNiosHOL.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.

The zip file has two items: C\_CODE and a .qar file for your respective board (depending on your board) **The lab will not work if you do not unzip the files!**

Quartus 18.0 standard should still be open. You need to open up the archive file from Quartus. Do not double click the .qar file to open the file as Quartus Prime Pro will launch which does not support Cyclone V libraries needed for this course.

Select a destination folder under the downloads directory (suggest initials\_Nios eg LL\_Nios) where you want your project to be restored, by clicking on **…** near the destination folder. *Make sure your destination folder is a C:// drive folder where you installed Quartus or your documents location where you want your project to be.*

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

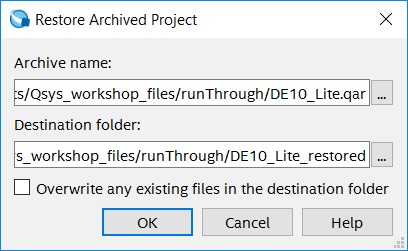


Figure 4: Selecting Archive Name and Destination Folder for the .qar file. The name might differ based on the development board you are using.

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 Platform Designer.

# Part 1: Hardware Design

## Lab 1: Building Your Platform Designer Based Processor System

The Platform Designer system panel diagram illustrates what you are designing in the master (the Nios II processor), and 11 slave devices.

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

Launch Platform Designer tool from Quartus: **Tools** → **Platform Designer** (or ”Qsys” prior to version 17.1). The initial screen you should see looks like this:



Figure 5: Platform Designer Main Panel

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

### Adding the Nios II Processor

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

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

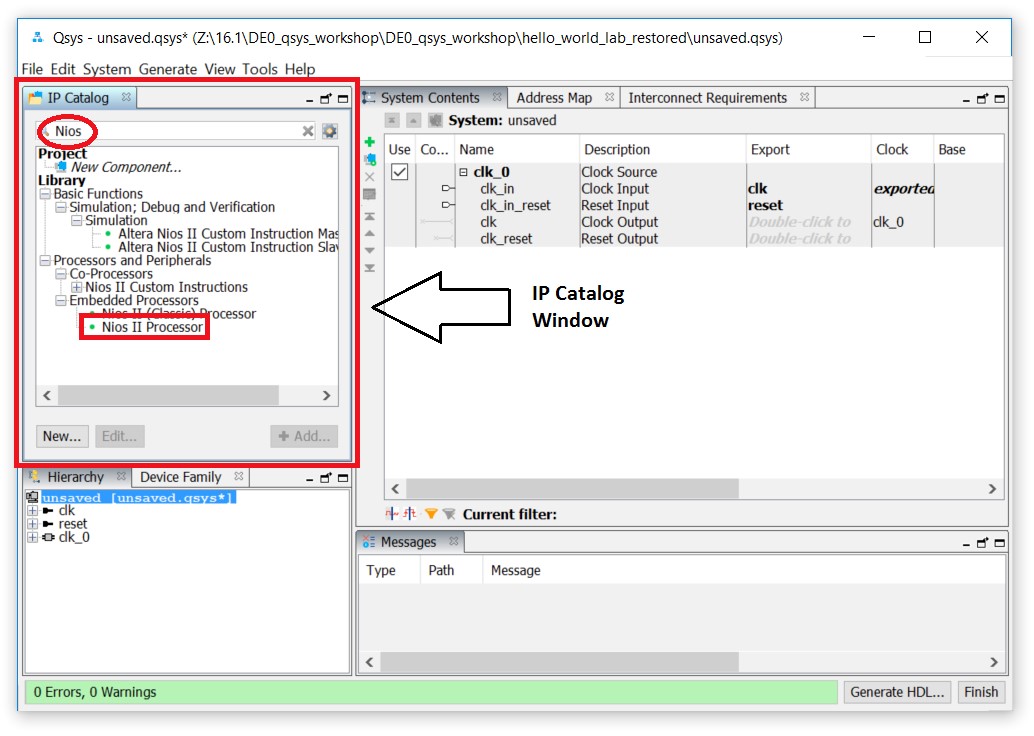


Figure 6: IP Catalog Tab

A configuration window will appear. 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.

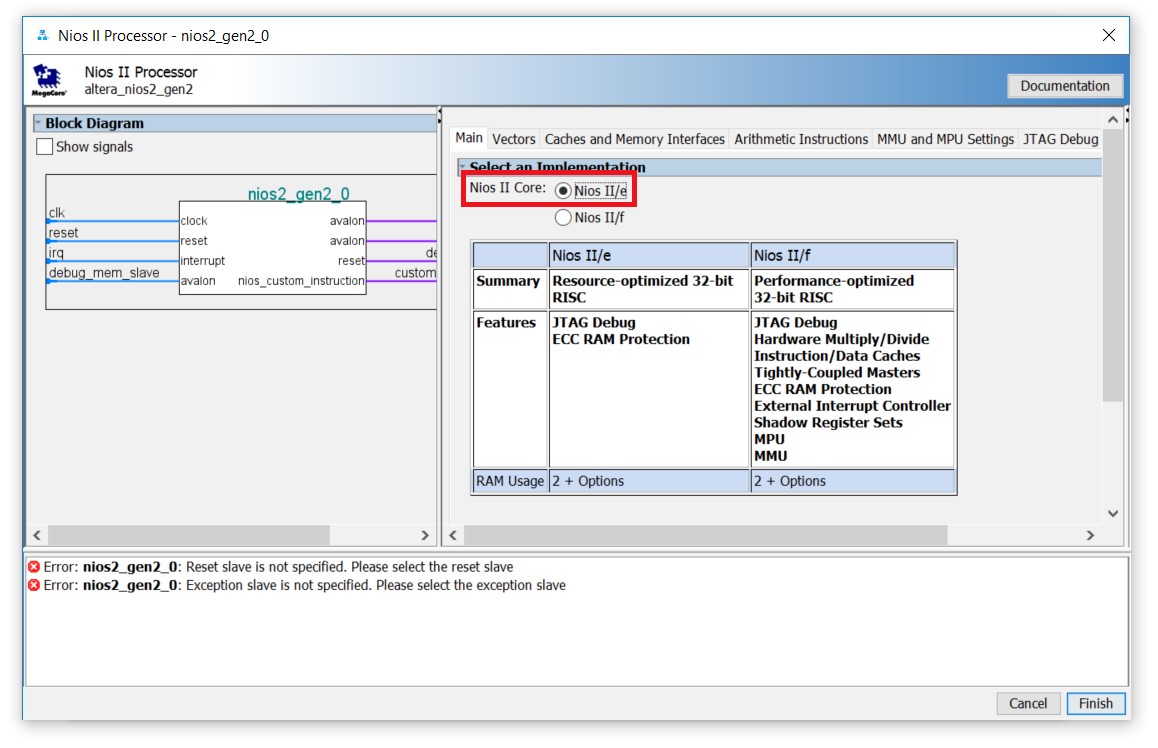


Figure 7: Nios II Gen2 Configuration Panel

Aside from choosing ‘e’, keep the default settings and click **Finish** and you will see the **nios2\_gen2\_0** processor in your connection diagram.

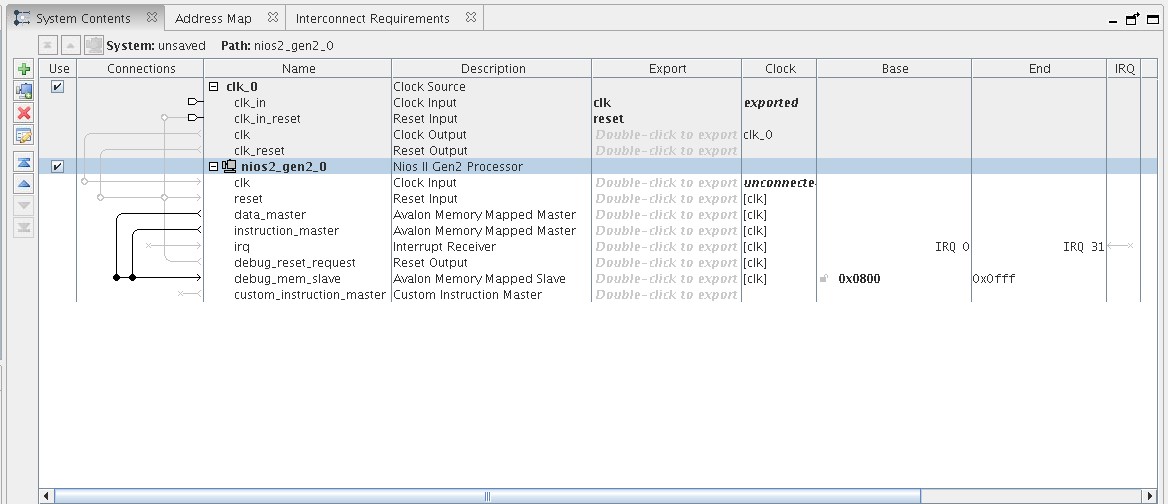


Figure 8: Platform Designer System Contents Panel

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

Note Platform Designer has a very elegant and efficient way of making connections by clicking on the nodes or ‘wires’ in the connections panel on the second 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.

### Adding On Chip Memory

With the Nios II processor added, you still need to add: **On Chip Memory**, **JTAG UART**, **pushbuttoninputs**, **switchinputs**, **LEDoutputs**, and the **7-segmentdisplayoutput** to your system.

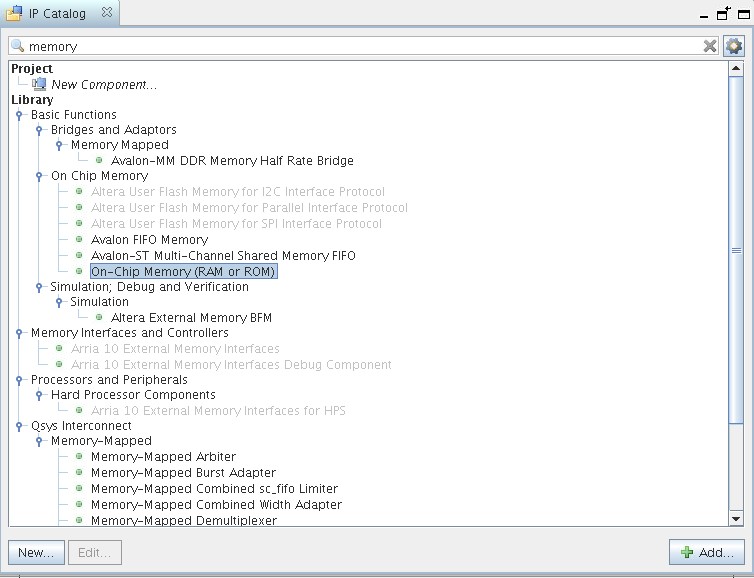


Figure 9: IP Catalog Search for On-Chip Memory

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 above.

Double click on the component or click **Add**.

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.

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.

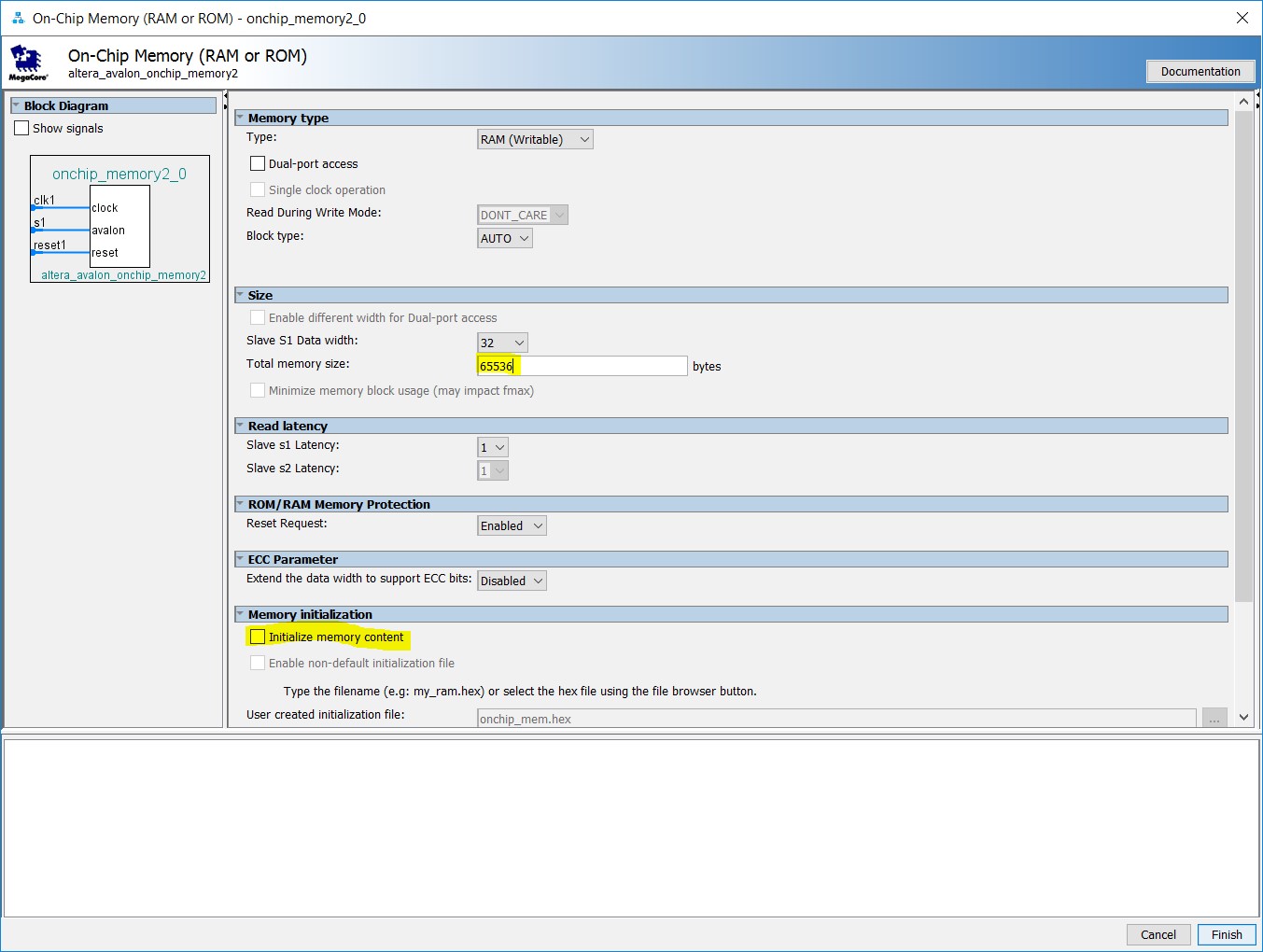


Figure 10: On-Chip Memory Configuration Panel

Click **Finish** and you will now see a total of three components in your Platform Designer system:

* clk\_0
* nios2\_gen2\_0
* onchip\_memory2\_0

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

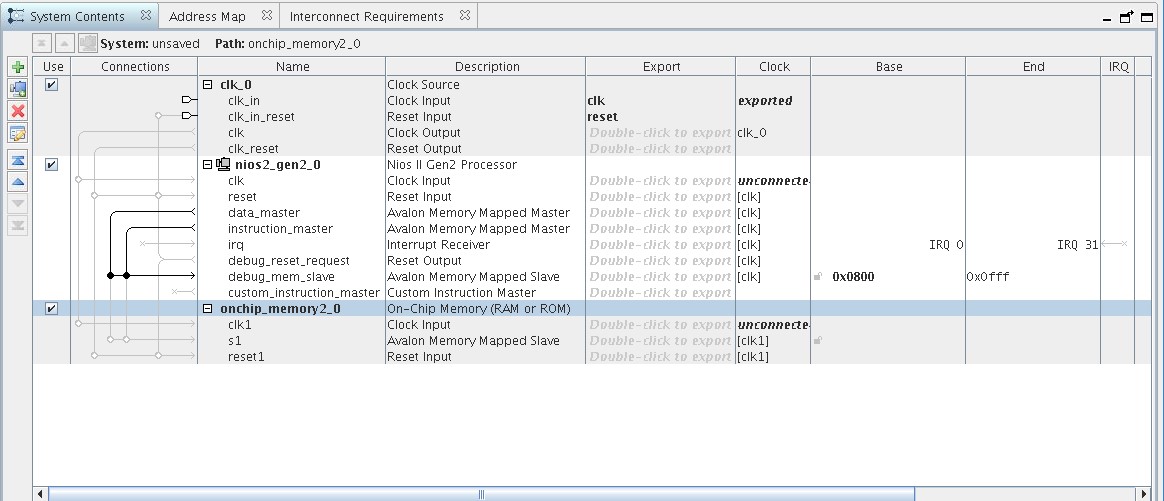


Figure 11: System Contents with Nios II and On-Chip Memory

### Adding the JTAG UART Component

The next component you will add is the JTAG UART.

Search for **JTAG** in the IP catalog and locate the **JTAG UART**. Double click or click **Add**.

Keep the default settings and click **Finish**.

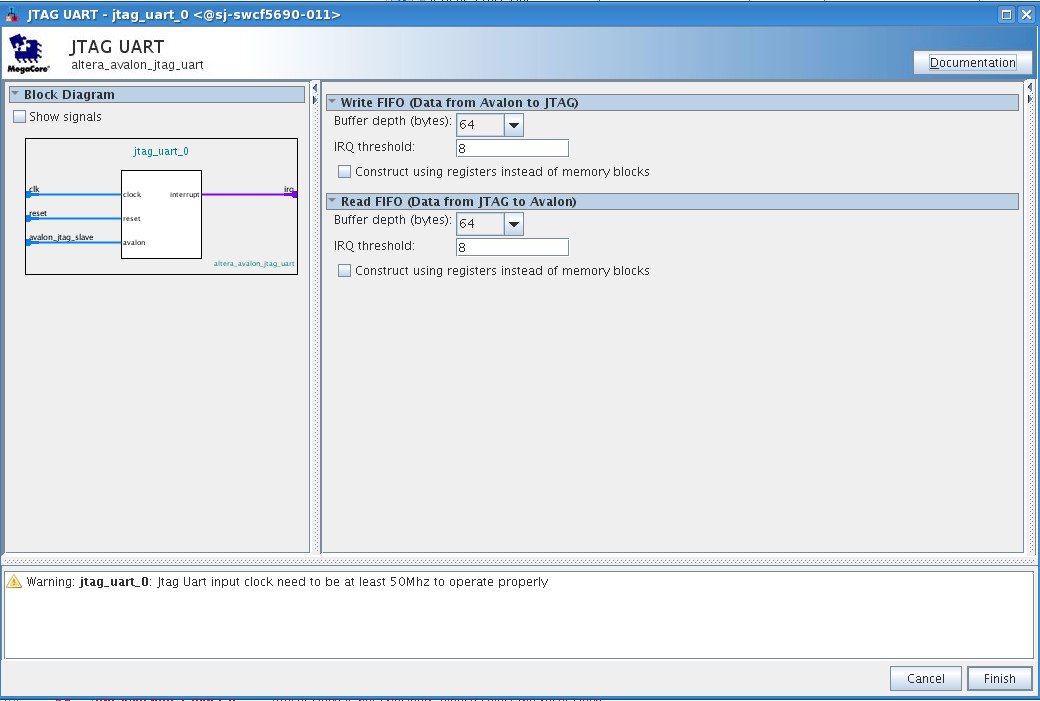


Figure 12: JTAG UART Configuration Panel

### Adding Parallel IO (PIO)

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 an address space, and your C code will read and write these components.

Search for **Parallel IO (PIO)** and select the correct block.

For the **pushbutton** block, we will set this up as a **2-bit input interface** using the settings shown below. There are two pushbuttons we would like to read from and two internal signals (a modification to HDL to support the DE10 board).

When you have setup your button input component interface as in Figure 13, click **Finish**.

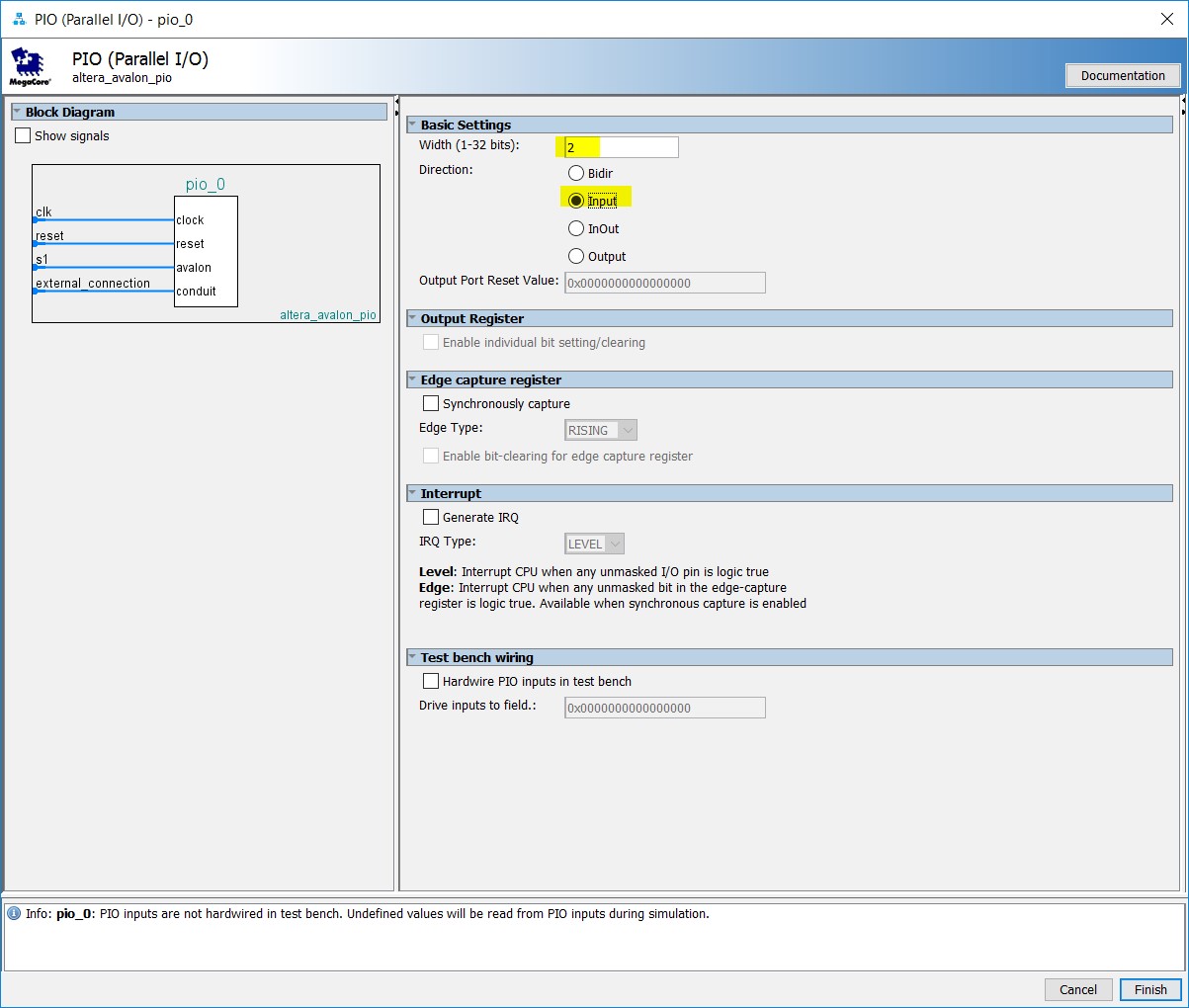


Figure 13: Parallel IO Configuration Panel for Pushbuttons

Double click on the PIO component as you did for the pushbuttons. This time you will configure this component as the LEDs: a **9 bit, output interface**. There are 10 LEDs.

However, only nine of them will be controlled by the NIOS.

When you have setup your LED output component interface as in Figure 14, click **Finish**.

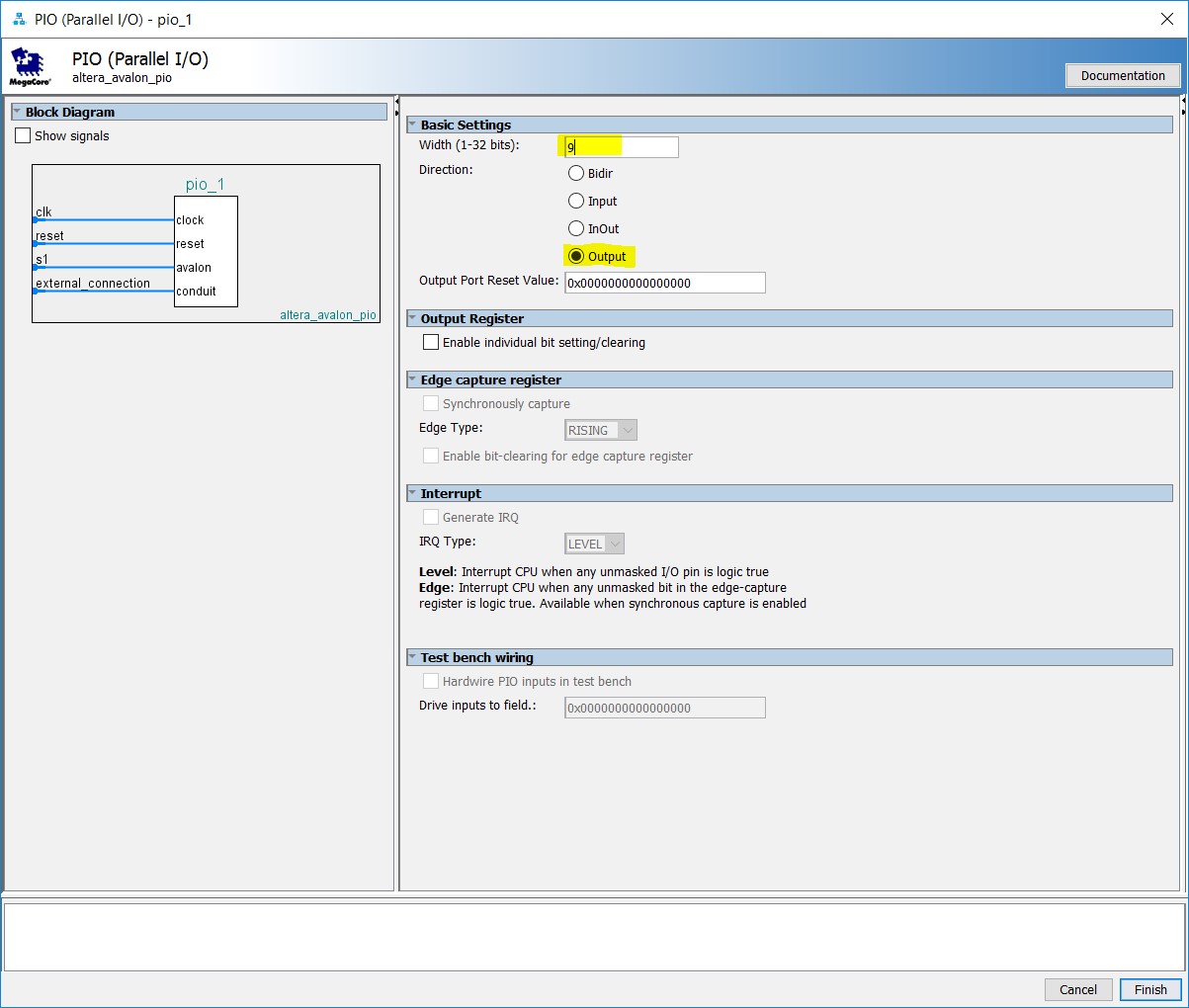
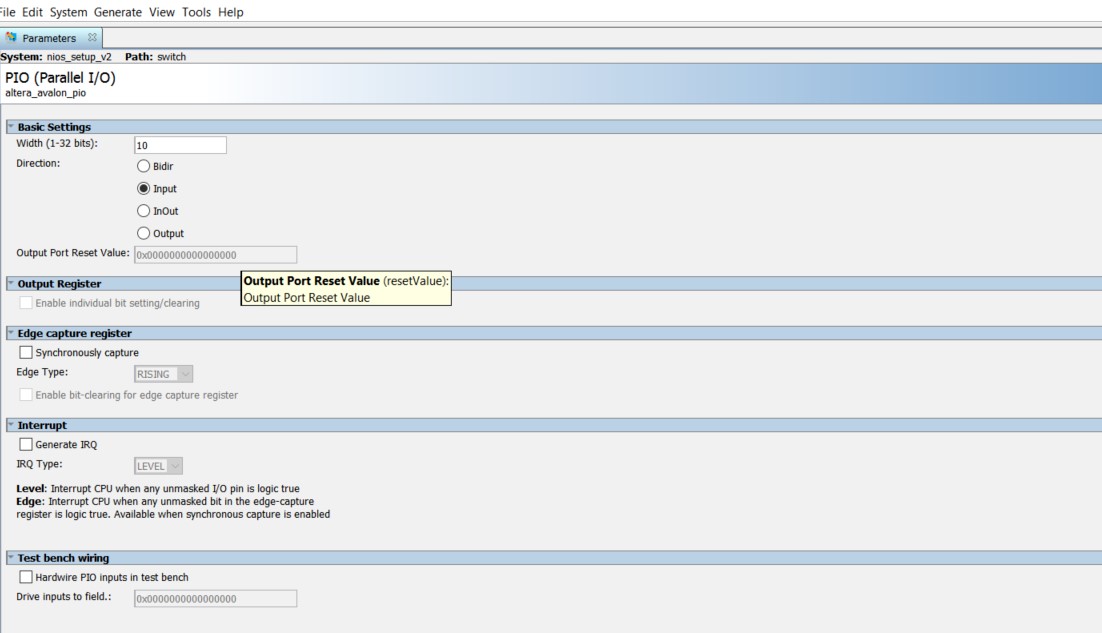


Figure 14: Parallel IO Configuration Panel for LED Outputs

Again, double click on the PIO component. Configure this component as the switches: a **10 bit input interface**.

When you have setup your switch input component interface as in **Figure 15**, click **Finish**.



**Figure 15: Parallel IO Configuration for Switch Input Panel**

Finally, we will add the six seven segment displays that will allow us to display text on the board. Create another PIO component, and configure it as a **7-bit output**, one for each light.

When you have setup your seven segment display output component interface as in Figure 16, click **Finish**.

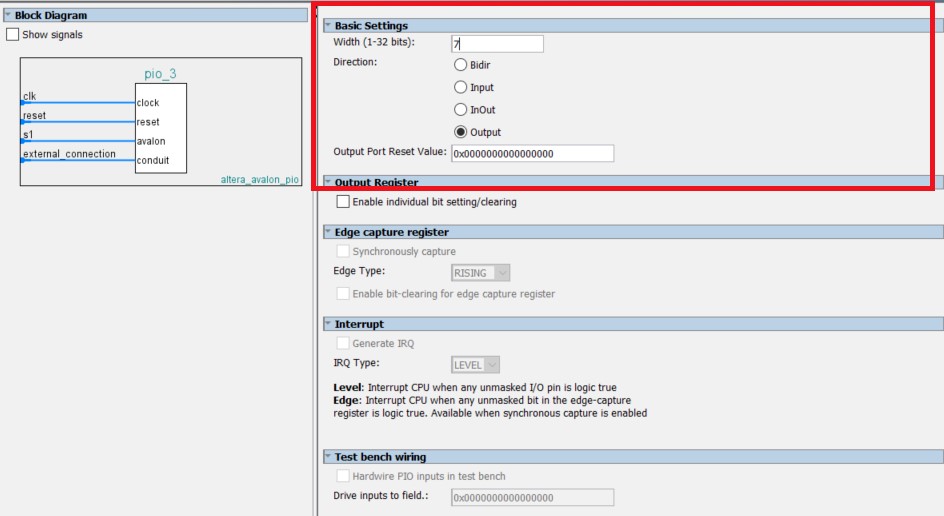


Figure 16: Parallel IO Configuration Panel for Seven-Segment Display Outputs

You have completed adding the components that make up your Platform Designer system.

Next you will rename the components in the design with names that are easy to remember.

### Connecting the System Components Together

In the system contents tab, right click on **clk\_0**, select rename, and type in **clk**.

Select the **nios2\_gen\_2\_0** component, select rename and type in **cpu**.

Similarly, rename the rest of the components as follows:

* **onchip\_memory**
* **jtag\_uart**
* **button**
* **led**
* **switch**
* **hex0**

Double check to make sure you’ve selected the correct PIO before renaming. For example, the button and switch components are both inputs but with different widths, and the LED is an output. Incorrectly naming a component could lead to errors when compiling!

Renaming the components 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.

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

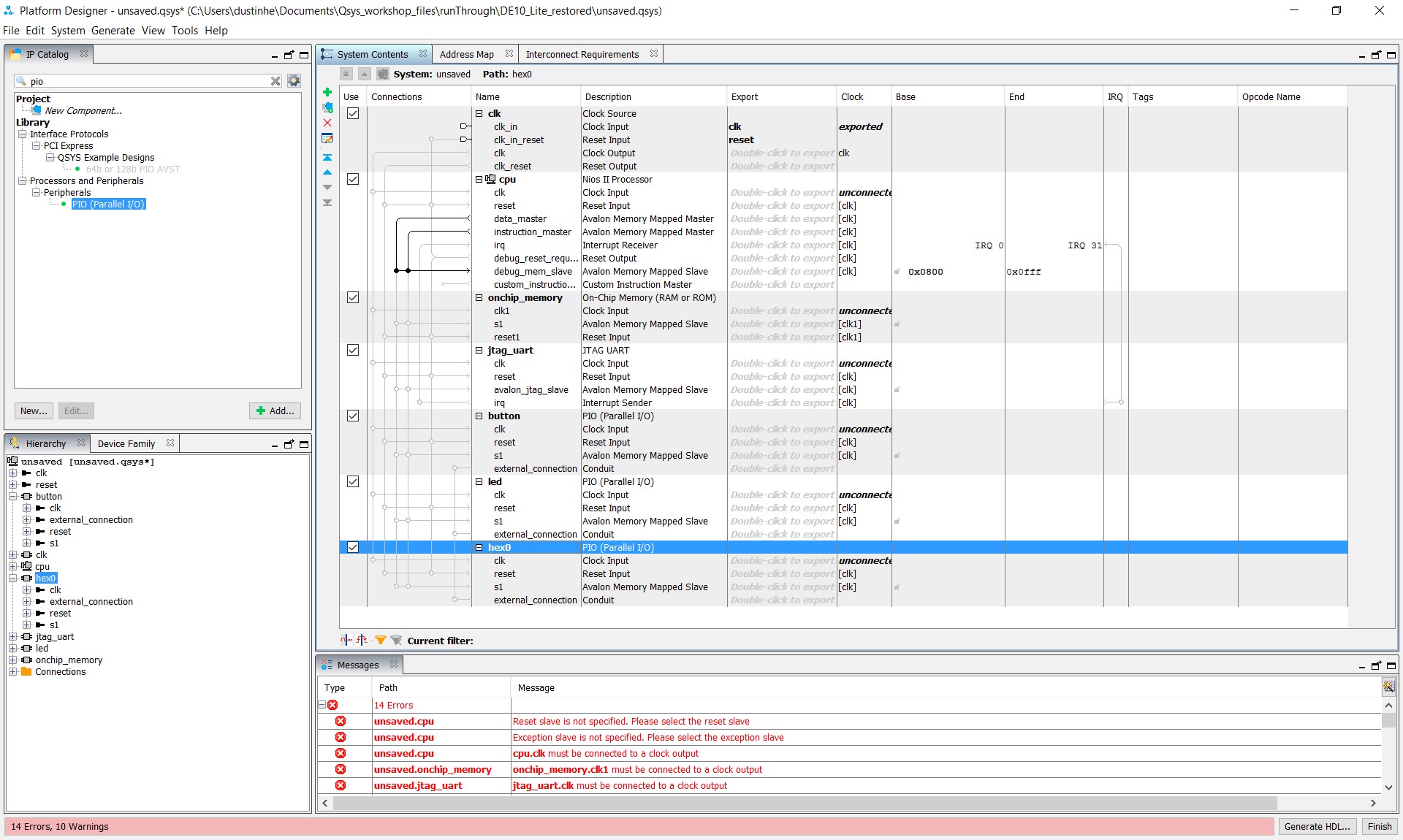


Figure 17: System Content Connections Starting Panel

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

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.

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.

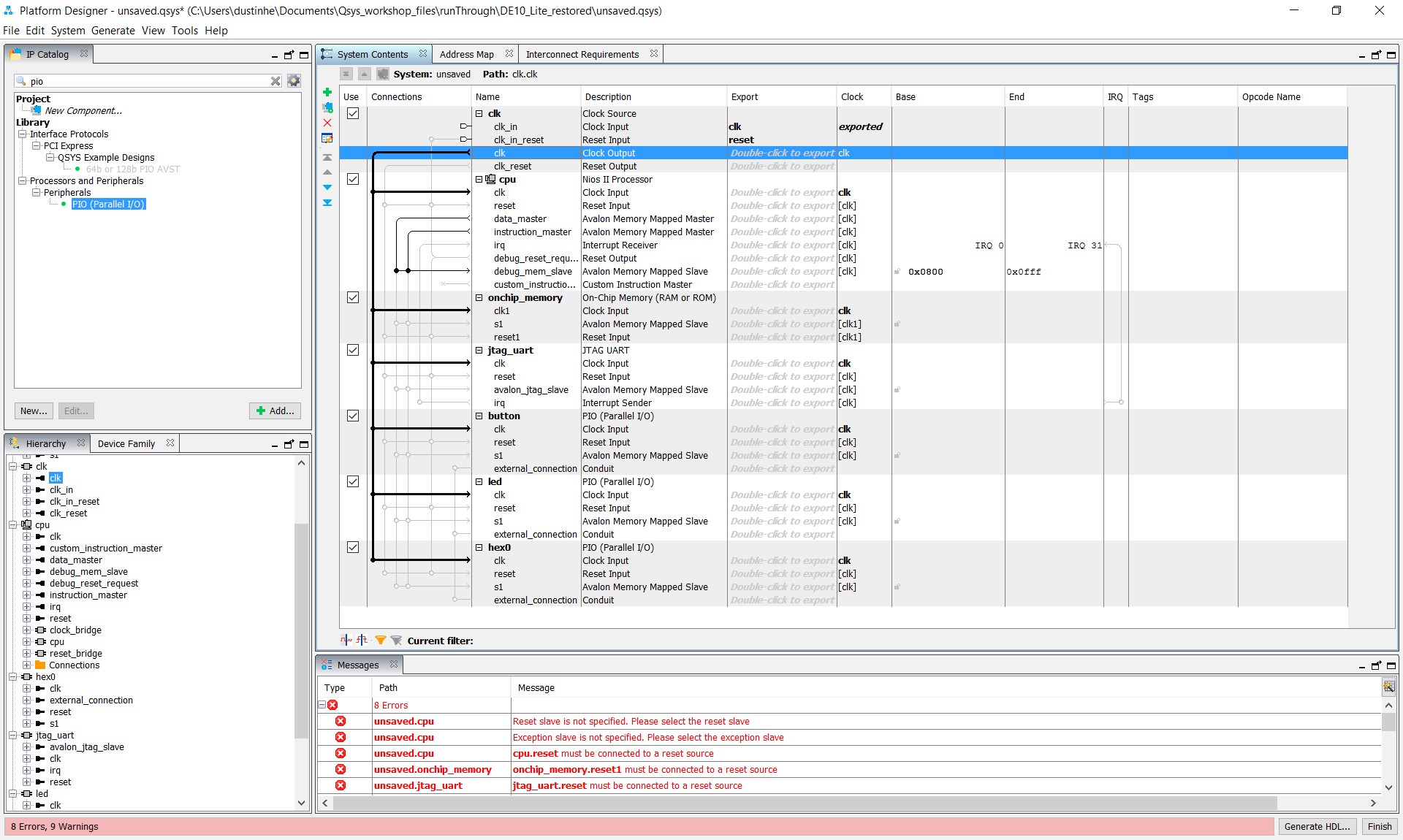


Figure 18: System Contents after Connecting the Clock

Perform the same operation to connect the **clk\_reset** from the clock component to the **reset** signals on the other components. At this stage, your design should look like Figure 19 below. (Color-coded for clarity.)

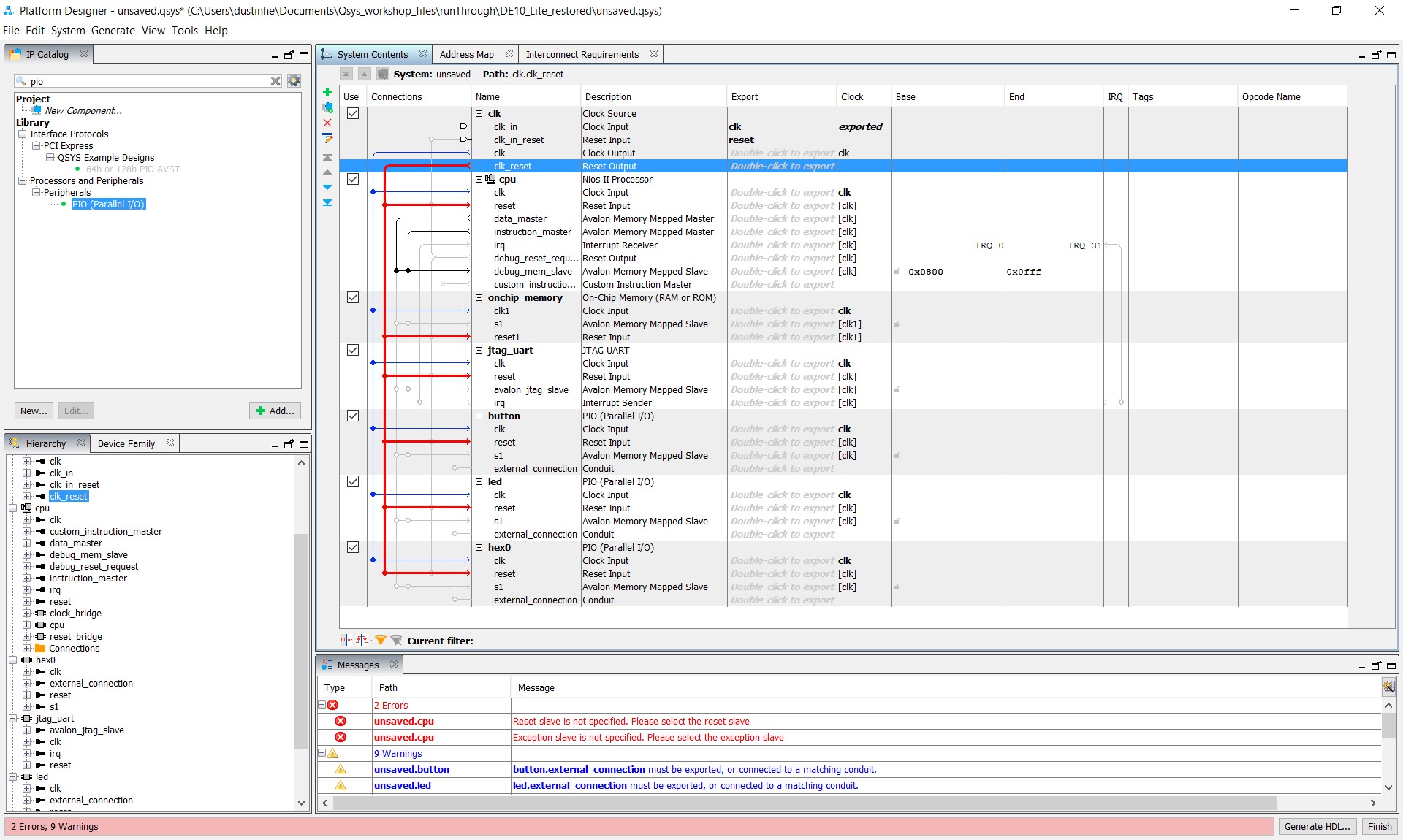


Figure 19: clk and clk\_reset Connected in Platform Designer

Connect the **cpu.data\_master** to the slaves. Make the connections between the:

* **cpu.data\_master** and the **s1** connection of the onchip\_memory
* **cpu.data\_master** and the **avalon\_jtag\_slave** on the UART component,
* **cpu.data\_master** and the **s1** port on the button component,
* **cpu.data\_master** and the **s1** port on the switch component
* **cpu.data\_master** and the **s1** port of the led component,
* **cpu.data\_master** and the **s1** port on the hex0 component
* **cpu.instruction\_master** and the **s1** port of the onchip\_memory

**instruction\_master** is by default connected to **debug\_mem\_slave**. Also, the **instruction\_master** needs to be connected to the **onchip\_memory**’s s1 port.

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.

Figure 20 below shows the Platform Designer system with the cpu.data\_master signal and cpu.instruction\_master signal connected to the other components in the proper locations.

(Color-coded for clarity.)

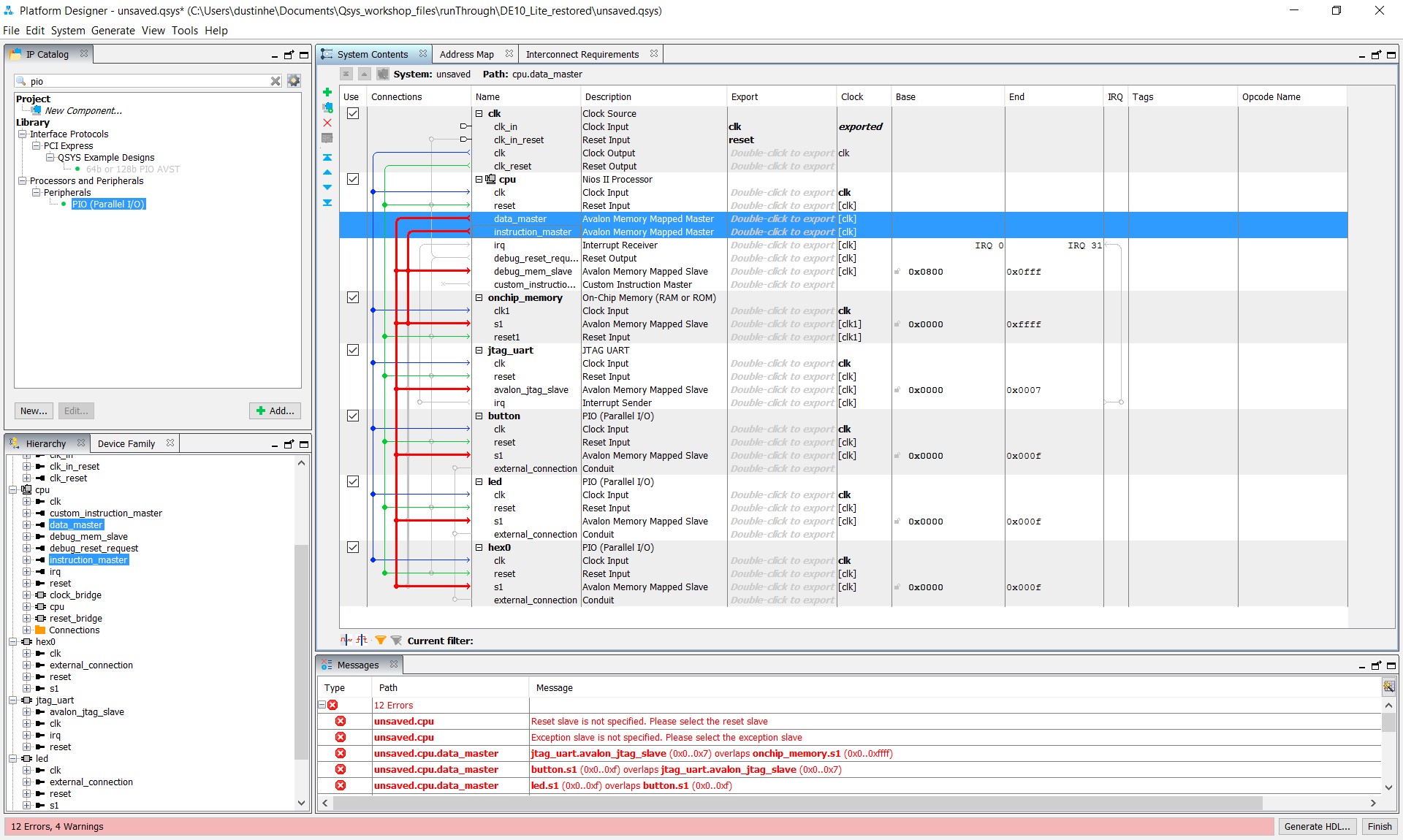


Figure 20: clk and clk\_reset Connected in Platform Designer

*Make sure that the instruction\_master signal from the cpu component is connected to the s1 slave of the onchip\_memory.*

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. 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.

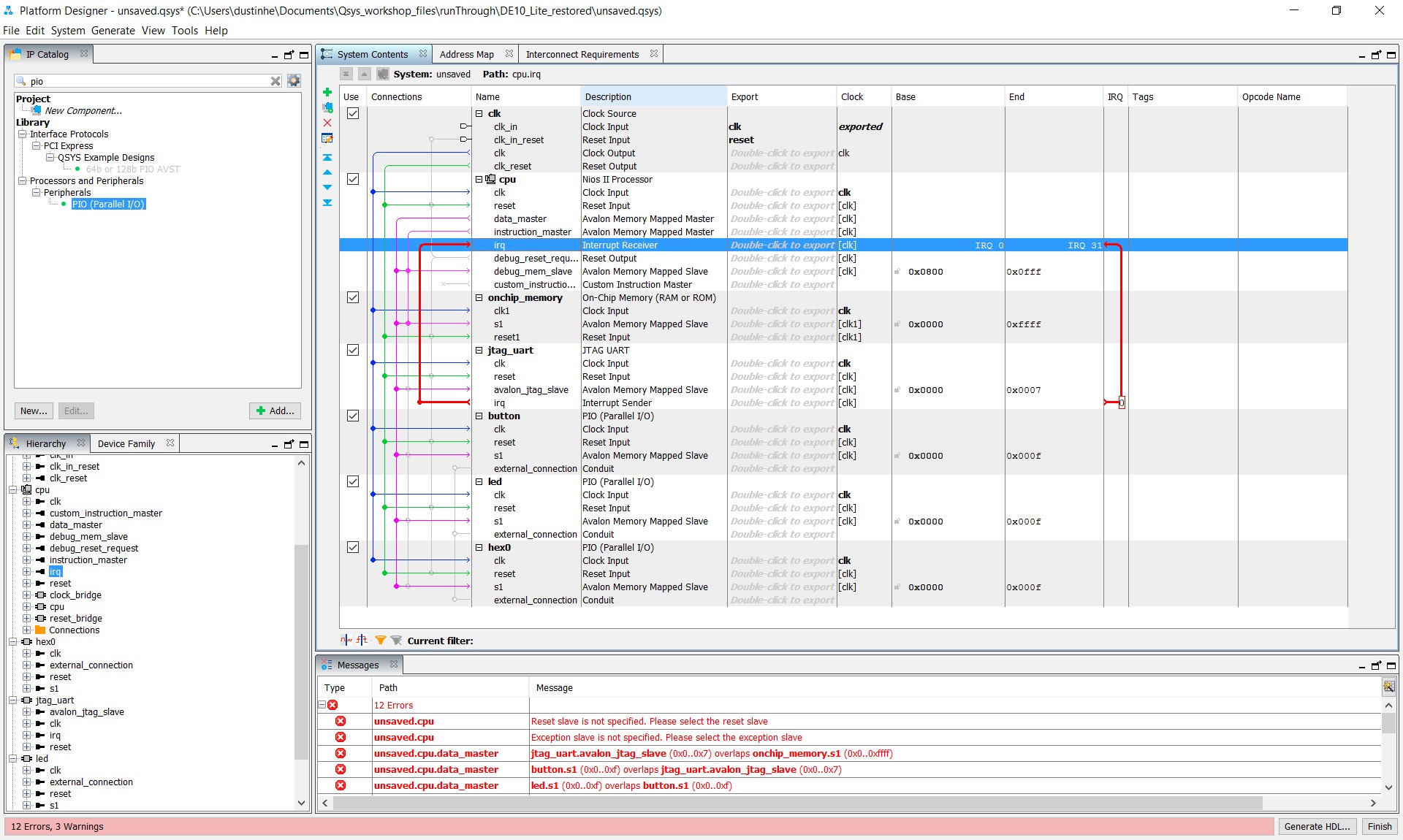


Figure 21: System Contents After Interrupt Connections

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

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

Once you have six of them (0-5), rename the new ones so they form the following list (pictured in Figure 24): **hex0**, **hex1**, **hex2**, **hex3**, **hex4**, **hex5**.

In case the connections were not kept when duplicating the new PIOs, be sure to connect:

* **clk** from the clock component to the clock signal of each hex component
* **clk\_reset** from the clock component to **reset** of the hex component
* **cpu.data\_master** from the cpu component to **s1** of each hex component

Once you have done this for **all of the six hex PIOs**, your systems contents panel should mirror Figure 22 on the following page.

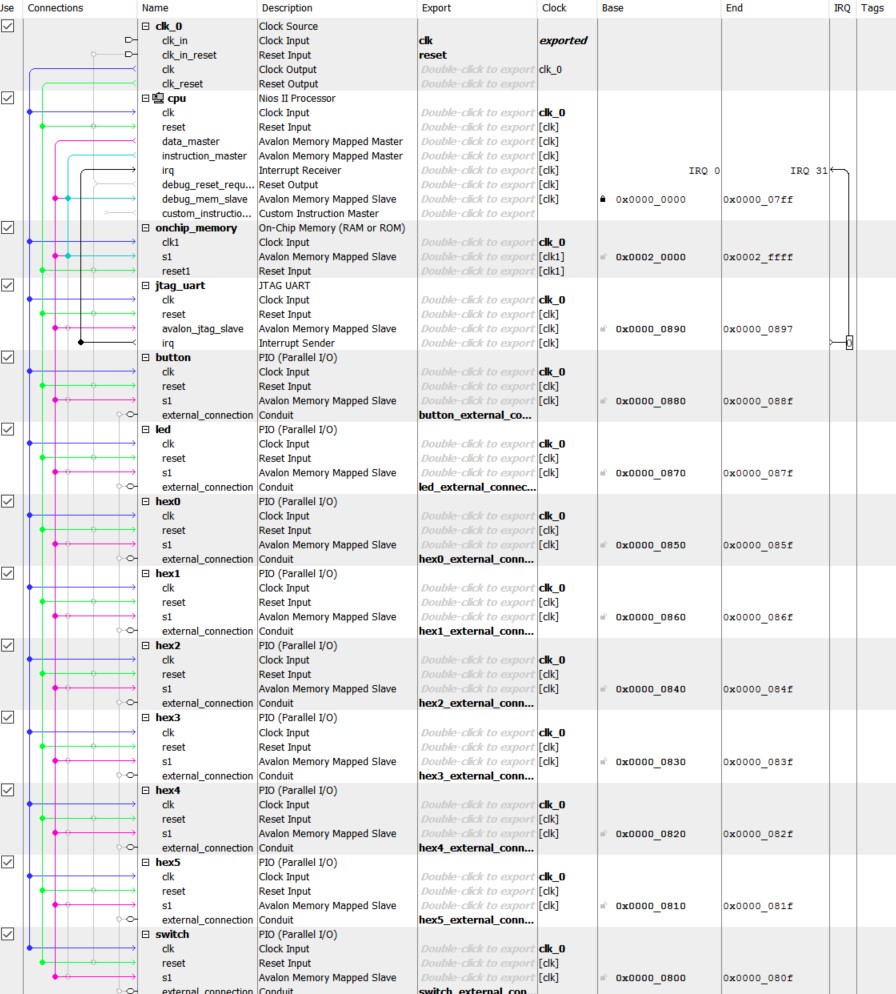


Figure 22: System Contents after Adding All 4 Seven Segment Displays

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 Platform Designer 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 button, led, switch 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 Platform Designer component to connect to the top-level design.

*Be sure the names of the components and exports match what is in Figure 25 EXACTLY, or the design may not compile at runtime.*

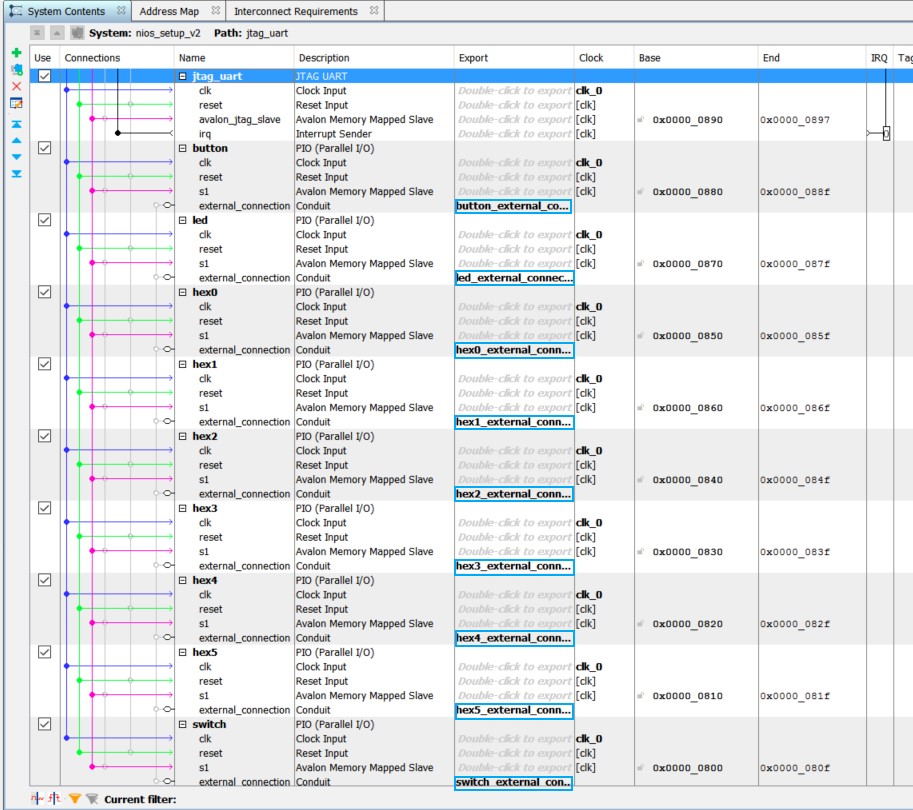


Figure 23: System Contents after Exporting PIO Switch and LED

Next you will need to generate the base addresses for your Platform Designer system.

This is achieved by using clicking on **System** → **Assign Base Addresses**.

Save your Platform Designer system by using **File** → **Save As** and pick a name for the

Platform Designer system that you will remember. Note that the lab figures call it nios\_setup\_v2, so to avoid confusion you may want to name your file the same. The information is saved in a .qsys file.

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

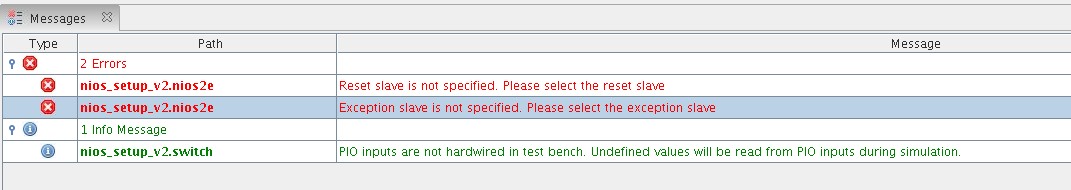


Figure 24: Error Message Prior to Assigning the CPU Memory Location

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 a straightforward fix. Double click on the **cpu** component and select the **Vectors** tab.

Set the **reset vector memory** and **exception vector memory** both to **onchip\_memory.s1**.

* 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.
* See Figure 25 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 should now be resolved. If you don’t have the option to select onchip\_memory.s1, double check your Platform Designer connections to the on chip memory S1 port.

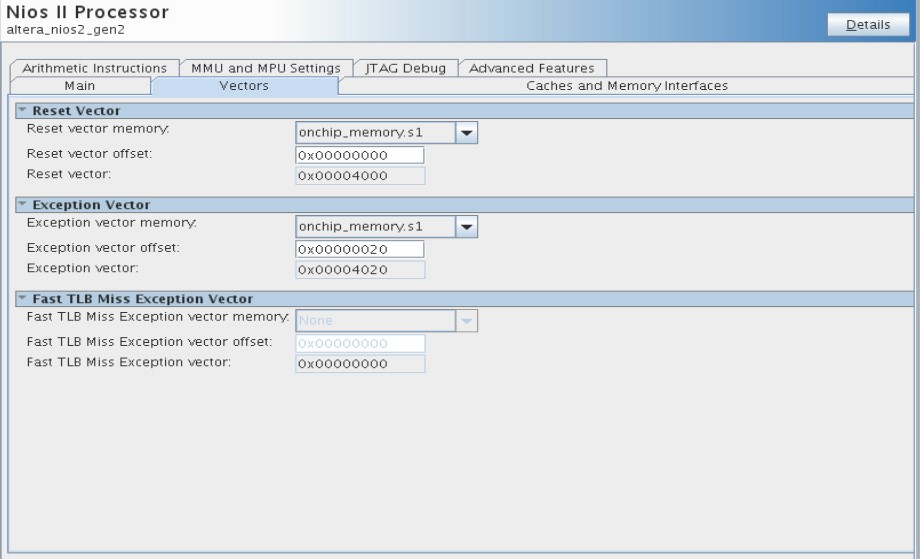


Figure 25: Assign Vectors in the Nios II Parameters Panel

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

Click on the button **Generate HDL**. A screen like Figure 26 should appear.

Click **Generate** on the panel that appears.

When the file generation is complete, click **Finish** to exit the Platform Designer window.

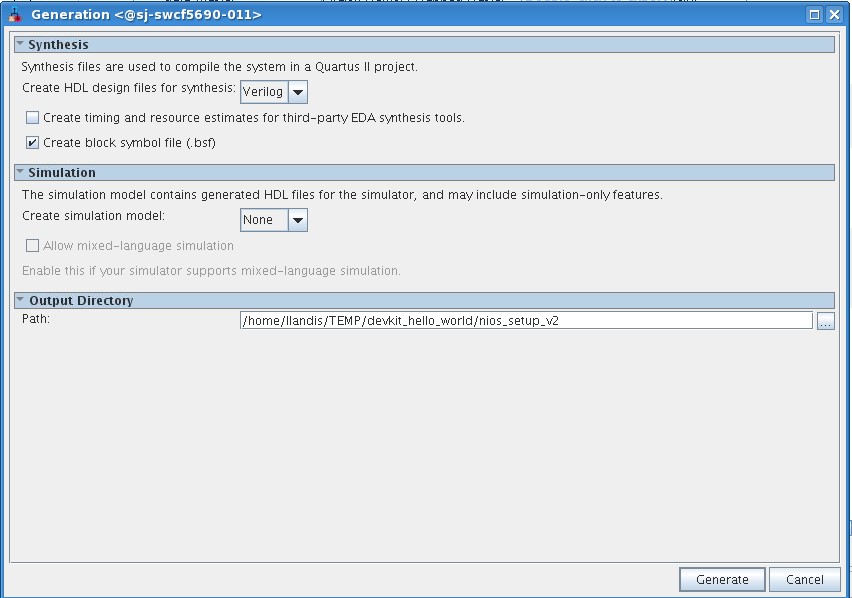


Figure 26: HDL Generation Panel

Congratulations! This completes the Platform Designer section of the lab.

## Building the Top Level Design

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

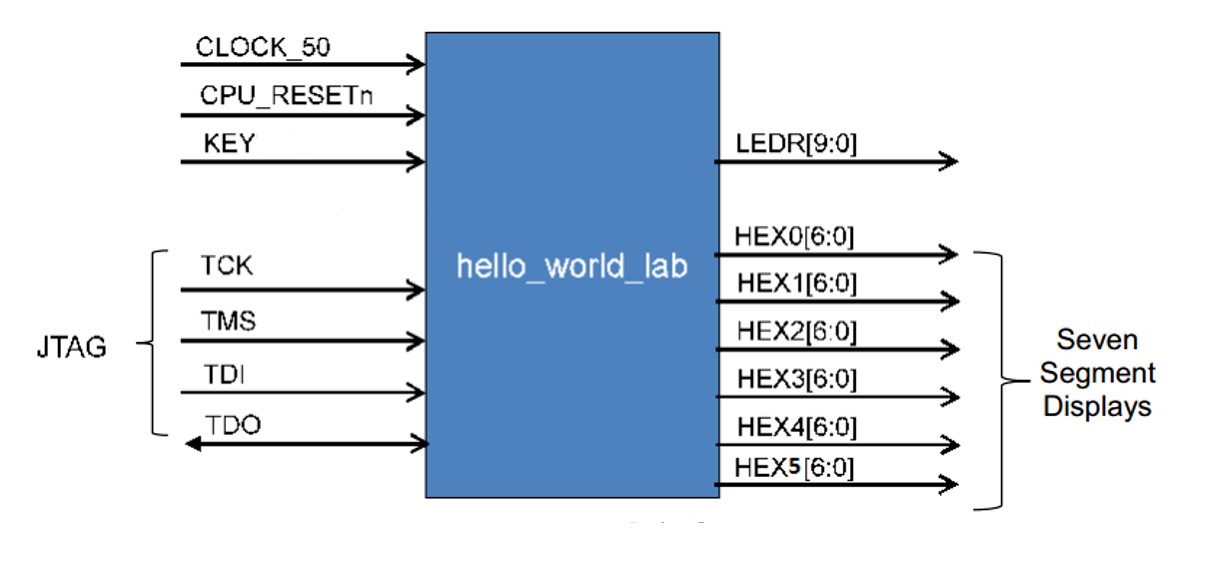


Figure 27: Block Diagram of hello\_world\_lab Design

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.

The top-level entity is in a file called **DE1\_SOC\_Golden\_Top** if you are using a DE-10 Lite development kit. If using CV GX Starter Kit, it is called **CV\_GX\_Starter\_Golden\_Top**.

Golden top is a naming convention that Intel FPGA often uses to designate the connections between the FPGA and all of the external components on the development board. This file is generally provided by the manufacturer of the development board, but we provide this code as part of the Quartus Archive (.qar) file for this course. You can see it by double clicking on file under the Project Navigator section.

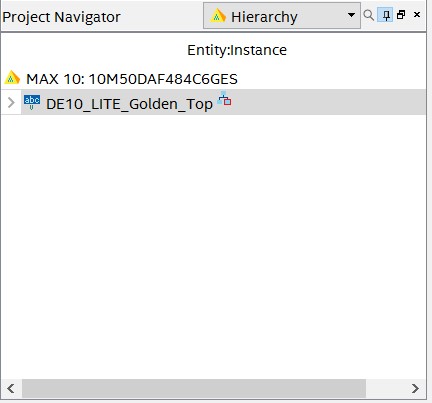


Figure 28: Project Navigator View of Golden Top File (note your board may differ)

The 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 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 Platform Designer system. The contents of this file are shown in Figure 29.

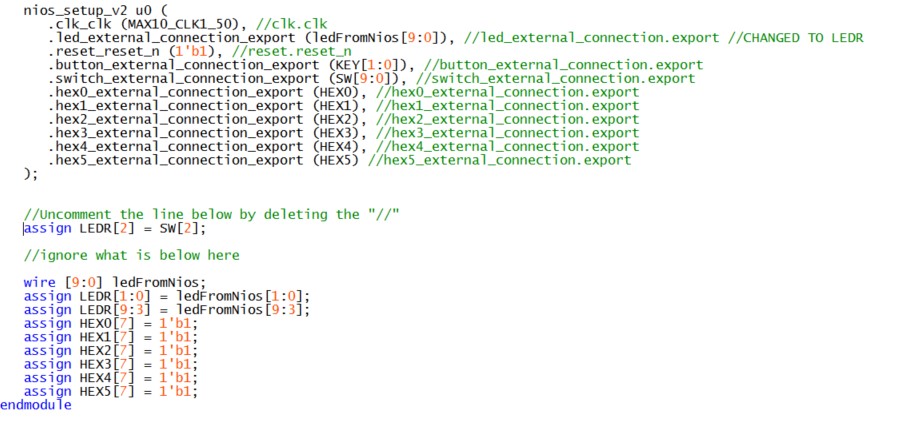


Figure 29: Contents of nios\_setup\_v2\_inst.v (Note clock name may differ)

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

In the top file (DE1\_SOC\_Golden\_Top.v or CV\_GX\_Starter\_Golden\_Top.v) uncomment line 88 by deleting the “//” at the beginning of the line.

* By uncommenting this line, we directly drive LED 2 on the board with switch 2 through the FPGA hardware. No software is required for this LED to operate.

In the Quartus main window, go to **Project** → **Add/Remove Files**.

Add the **nios\_setup\_v2.qip** file. (You can also just add the nios\_setup\_v2.qsys file.)

* The nios\_setup\_v2.qip file should be found under **nios\_setup\_v2** → **Synthesis** directory in your project.
* You will need to change the filter to display **All files** if you cannot see it.

The **.qip** file contains the information for the processor system that we created in the last step. The **.v** file connects the Platform Designer system we made to the inputs and outputs of our board.

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

See Figure 30 for what your Add/Remove Files window should look like. (There may be an extra .sdc file in the list. This is fine.)

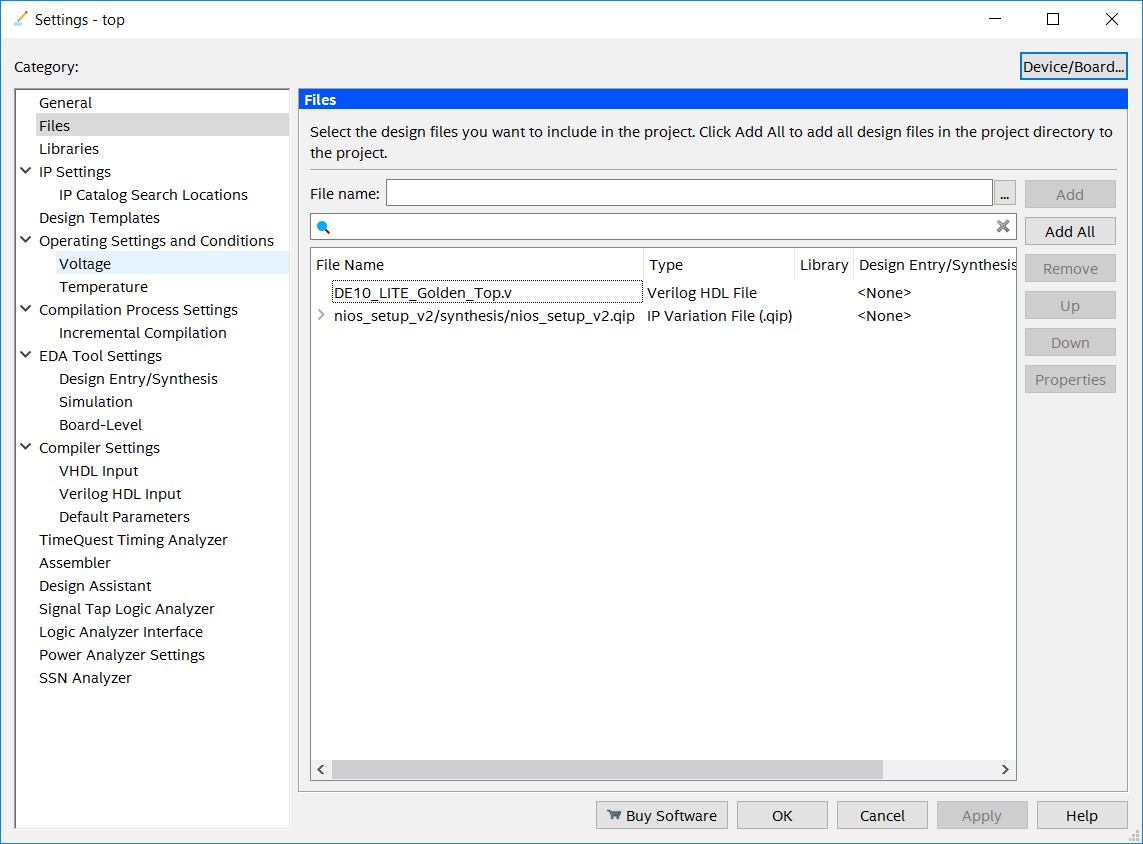


Figure 30: Quartus Add/Remove Files Pane

Almost there! We have pre-included and set up the pin assignments for the 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.

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

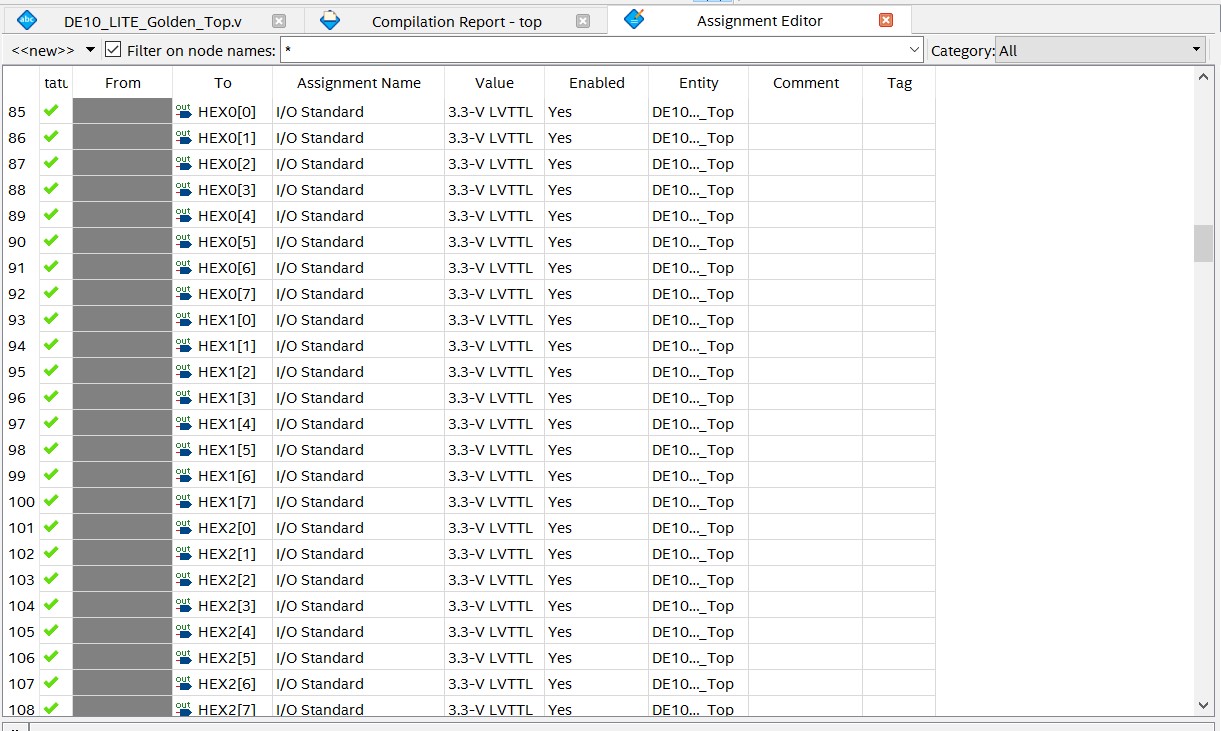


Figure 31: Quartus Assignment Editor Window

Figure 31 above 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.

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).

Click on the play button as shown in Figure 32: Compilation Button on Quartus Toolbar.



Figure 32: 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 Intel FPGA 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 Nios II processor that we just configured.

# Part 2: Software Design

## Lab 1: 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).

Launch **Tools** → **NIOS II Software Build Tools** for Eclipse. You can use the default location that Eclipse picks for you.

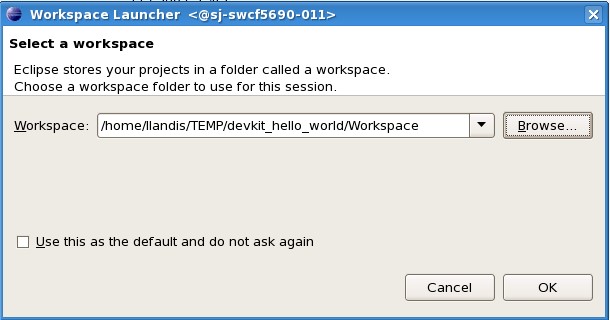


Figure 33: Initial Workspace Setup

Click **OK** in the Workspace launcher. Next, the Eclipse SBT will launch.

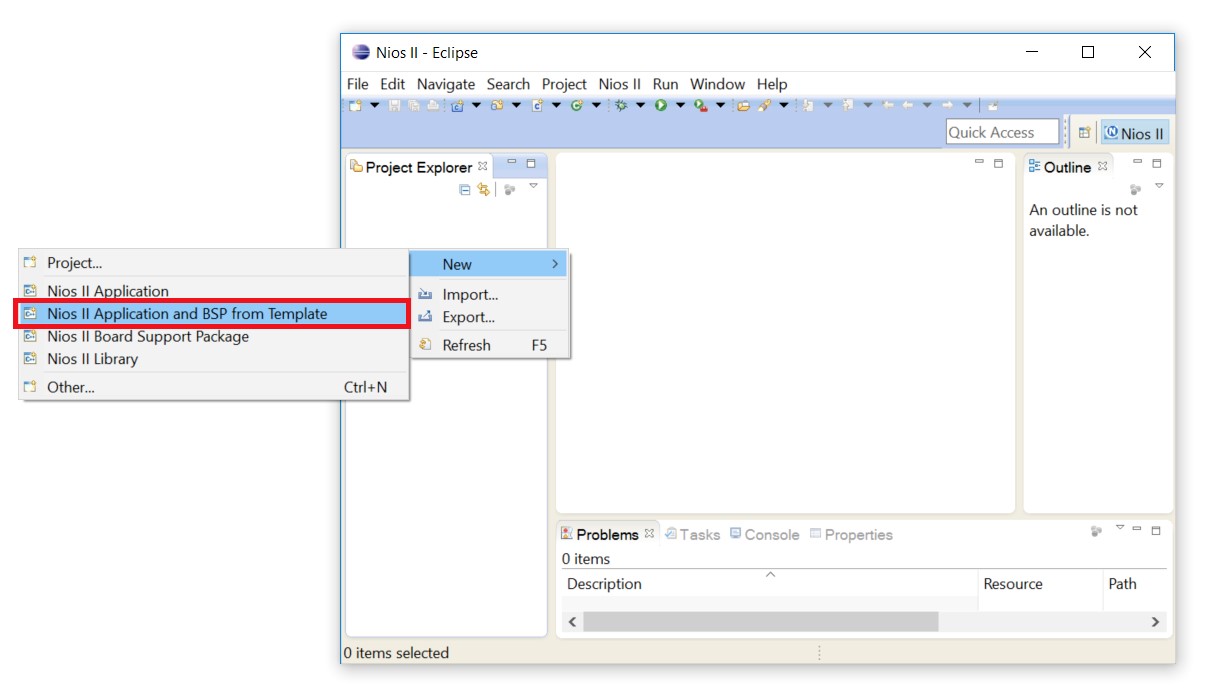


Figure 34: Creating the Initial Project in the Eclipse SBT

Right click in the area called Project Explorer and select **New** → **Nios II Application and BSP from Template** as shown in Figure 34: Creating the Initial Project in the Eclipse SBT above.

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.

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

Navigate to your working directory and click on the **.sopcinfo** file. The **.sopcinfo** file informs Eclipse on what your Platform Designer system contains.

Click **OK**.

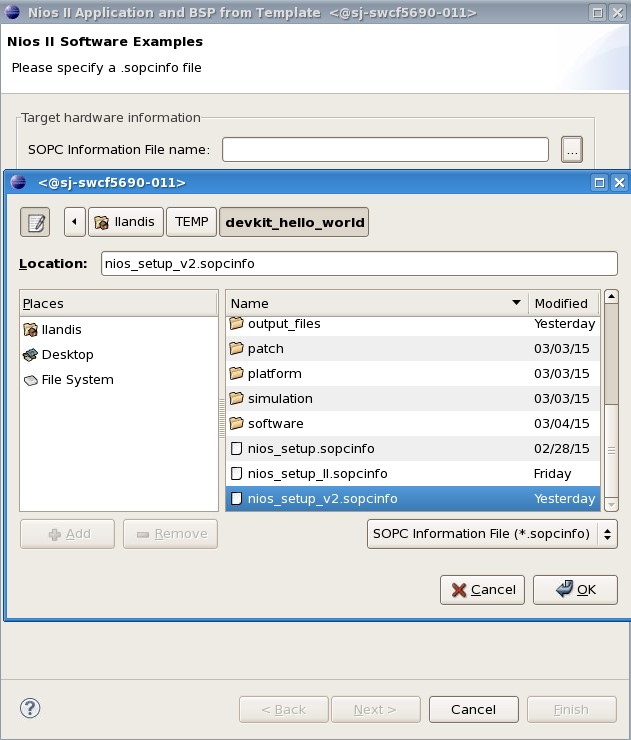


Figure 35: Navigating to the .sopcinfo File

Fill in the Project name, call it **hello\_world\_sw**.

Next you will be asked to pick a template design. Select the **Hello World Small**” application template. This template writes “Hello from Nios II” to the screen.

• 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**.

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

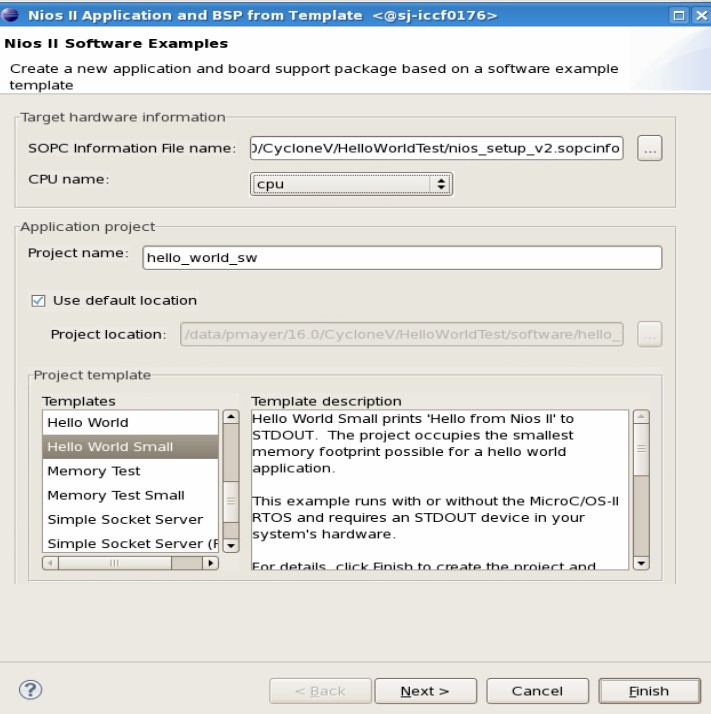


Figure 36: Completing the Nios II Software Examples Setup Screen Click the right arrow next to hello\_world\_sw. It will show the contents of your project.

Double-click **hello\_world\_small.c**.

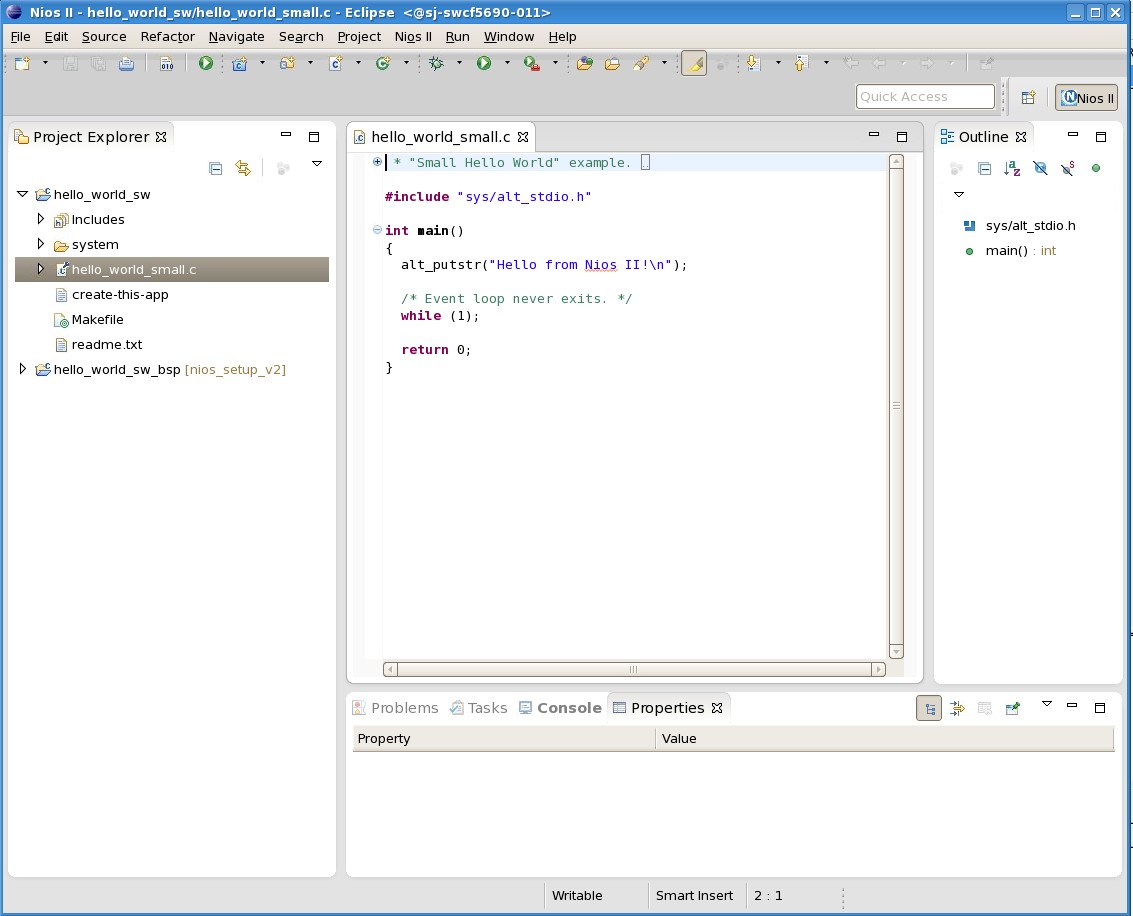


Figure 37: Eclipse Window of “hello\_world\_small.c”

Note the command alt\_putstr to write text to the terminal. This is part of the Hardware Abstraction Layer (HAL) 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.intel.com/content/www/us/en/programmable/products/ processors/support.html](https://www.intel.com/content/www/us/en/programmable/products/processors/support.html).

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.

Drag and drop the file **DE\_hello\_world.c** (found in the subfolder C\_CODE) into Eclipse Project Explorer tab under the **hello\_world\_small\_sw** project folder.

• If you cannot drag and drop, copy and replace the code from the DE\_hello\_world.c into the hello\_world\_small.c and skip the next step.

Delete the pre-made **hello\_world\_small.c** file in your **hello\_world\_sw** folder in the Eclipse Project Explorer. This can be done by right clicking on **hello\_world\_small.c** and selecting **Delete** from the 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. IORD\_INTEL PSG\_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\_INTEL PSG\_AVALON\_PIO\_DATA (Location, Value) writes the numeric Value to the given Location. We are using this function to read the data from the push buttons 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.

Click the save icon.

Now that we have written our code, click **Project** → **Build All**.

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 All**. Typical problems include missing semicolons and mismatched brackets.

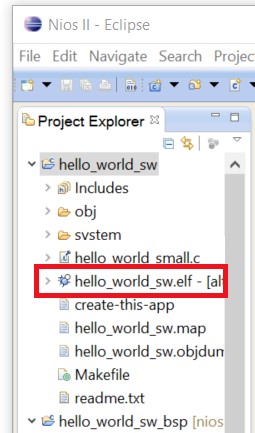


Figure 38: Window View of “hello\_world\_sw.elf”

## Lab 2: Downloading the Hardware Image to the Development Kit

To work with the Cyclone V board 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.

With the USB blaster drivers properly installed, launch the Programmer by clicking 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.

Right click on the first row <none> under File and click on Change File. Navigate to the output\_files directory and select top.sof.

Click Open.

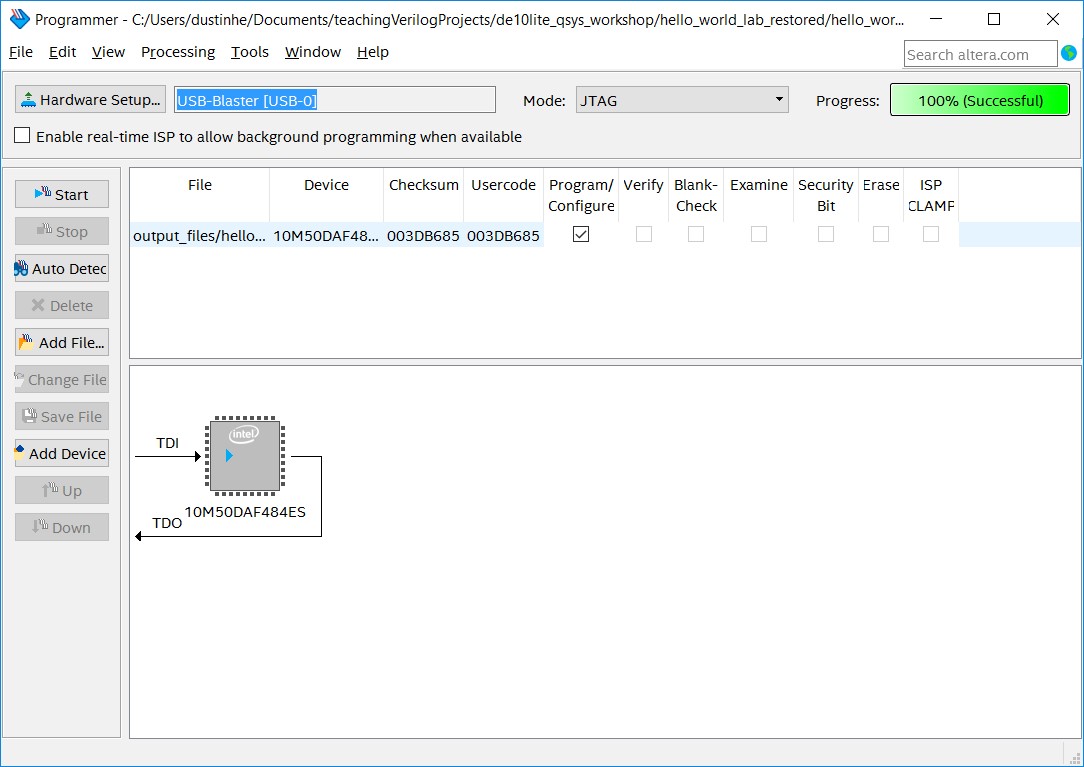


Figure 39: Program/Configure Checkbox

In the first row under **Program/Configure** click in the check box as shown in Figure 39: Program/Configure Checkbox above.

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**.

Note for the DE1-SoC board. If you use the DE1-SoC board, click auto-detect. You see two devices in the chain, SOCVHPS first followed by the 5CSEMASF31. If you don’t see this, click auto detect and you will. Right click on the file for the 5CSEMASF31, click change and navigate to your .sof file in the output\_files files directory and program your board.



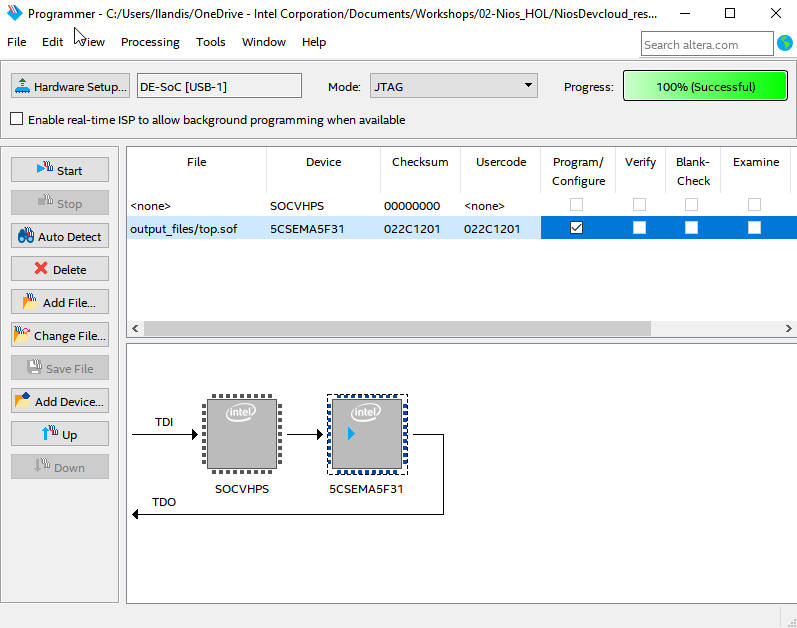


Figure 40: Programmer Progress Successful

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

Now that the FPGA is programmed the hardware is operating. However, we have not programmed the software for the NIOS CPU yet.

**Note with the HOL setup and no access to a physical board, you will not be able to complete this step**: *To demonstrate the hardware is functioning, even while the NIOS processor is not, press the switch SW2 to on (towards the LEDS). You should see only one LED light up. Follow the steps below then try pressing the keys again. Note how the hardware driven LED does not need the software executable file .elf to operate. Have your instructor demonstrate this step.*

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

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.

Click on the **Target Connection** tab.

* The connection should indicate that Eclipse has connected to the USB-blaster.
* If the connection is not identified, you can click **Refresh Connections**.
* You might need to stretch the window wider to see the Refresh Connections button.

Once the connection is made to the USB-Blaster, you should observe something like Figure 41.

Click **Run**. If the run button is grayed out but your device shows up under the connections window, you may need to select **Ignore mismatched system ID** and **Ignore mismatched system timestamp**.

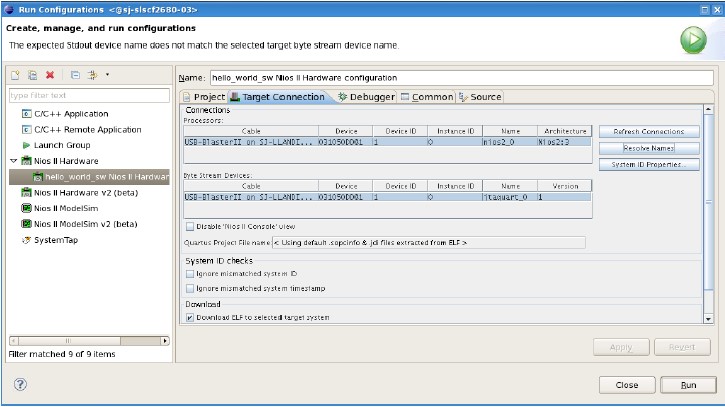


Figure 41: Eclipse SBT Tools after Connection is made to the USB-Blaster

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

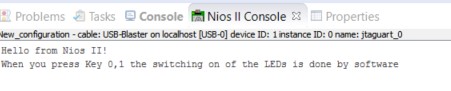


Figure 42: ”Hello from Nios II!” on Nios II Console Tab

**Note with the HOL setup and no access to a physical board, you will not be able to complete this step**: *You can also test the connections between push button and LEDs. Push buttons 0-1 should now turn LEDs 0-1 on when pressed. The pushbuttons and LEDs were connected through our Platform Designer system and the C code we have running on our development kit. Have your instructor demonstrate this step.*

**KEY CONCEPTS**:

* When you push buttons 0 and 1, LEDs 0 and 1 will light up. This is because of **software**.
* When you flick switch SW2, LED 2 will light up. This is because of **hardware**.

## Lab 3: Using the Seven Segment Display

Note: **This section requires a physical board in hand to view and will not work with the Hands-On lab. Have your instructor demonstrate this step.**

One of the nice things about the NiosII 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 NiosII processor to display text on the seven segment displays and make pushbuttons speed up and slow down the text.

Drag and drop the file named **DE\_seven\_segment\_display.c** into the hello\_world\_sw project folder in Eclipse. DE\_seven\_segment\_display.c can be found in the C\_CODE subfolder in the DE10\_qsys\_workshop folder. **If you cannot drag and drop the file, copy and replace the code from DE\_seven\_segment\_display.c into the .c file already present and skip the next step**.

Remove the file **DE\_hello\_world.c** by right clicking on the file and selecting **Delete**.

Right click on the hello\_world\_sw in the Project Explorer and click on **Clean Project**.

When the program is finished, right click on hello\_world\_sw again and select **BuildProject**.

Once the build completes, the **.elf** file under the hello\_world\_sw project should be updated. 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.

Right-click on the **hello\_world\_sw** folder in the Project Explorer on the right and select **Run as** → **Run Nios II Hardware**. This will run the new C program on the NiosII processor. 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.

You can control the text in the following manner using the two push buttons:

* Press KEY0 to perform multiple functions. The console outputs the current step.
  + Press to speed up (hold down to speed up more).
  + Press again to speed up more (hold down to speed up more).
  + Press again to go even faster (might go so fast all LEDs appear on).
  + Press again to slow down.
  + Press again to change scroll direction (to the right). - Press again to flip letters upside-down.
  + Press again to make the letters scroll up (or dance)
  + Press again to make the letters scroll down (or dance)
  + Press again, to clear screen
* Press KEY1 change text. Look at the console for further instructions.

If you are fluent in C, try modifying the program to add functions for some of the other switches. 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:

* Loading the Device Kit pin settings into Quartus.
* Using Platform Designer to build a Nios II based system.
* Instantiating the Platform Designer component into your top level design.
* Add some connections between push buttons, switches and LEDs.
* Compiling your hardware.
* Importing the Nios II based system into the Eclipse Software Build Tools.
* Building a software project.
* Modifying a software template to perform some simple IO functions.
* Compiling your software.
* Downloading the hardware image into the development kit.
* Downloading the software executable into the development kits.
* Testing the hardware.

Please visit [http://fpgauniversity.intel.com](http://fpgauniversity.intel.com/) to discover more embedded systems, NIOS, and software development trainings and reference designs from Intel and our technology partners.

## Revision History

|  |  |  |
| --- | --- | --- |
| DATE | NAME | DESCRIPTION |
| 05/01/2015 | L. Landis | Initial release |
|  |  |  |
| 06/02/2015 | L. Landis | Added BeMicro |
| 11/30/2015 | I. Rush | Added CVE DevKit |
| 12/02/2015 | S. Meer | Consolidated sections |
| 12/04/2015 | I. Rush | Updated pinout table |
| 03/18/2016 | K. Kita | Separated lab by board |
| 05/10/2016 | J. Xia | Revised for university workshops |
| 06/06/2016 | P. Mayer | Added scrolling text |
| 03/23/2017 | A. Weinstein | USB blaster installation |
| 04/03/2017 | A. Weinstein | Added CVGX DevKit |
| 04/18/2017 | A. Weinstein | Updated .qar files |
| 10/23/2017 | D. Henderson | Port to DE10-Lite |
| 02/15/2018 | A. Joshipura | Added location where to unzip files |
| 03/21/2018 | A. Joshipura | Added switch in the manual and changed figures for it; added SW2-LED2 connection |
| 04/02/2018 | A. Joshipura | Edited functionality of seven segment display to do all functions in button 0 |
| 04/08/2018 | R. Nevin | Fixed switch PIO direction, clarified guidance for “hello world small” template & instructions to import  DE10LITE\_hello\_world.c |
| 04/11/2018 | A. Joshipura | Added a single page for different workshop links; added images of both boards and changed Qsys to Platform Designer. |
| 04/25/2018 | A. Joshipura | Added Intel logo 7 explanation on the links |
| 07/06/2018 | S. Soto | Fixed System Done code by uncommenting line 88 (for DE10-Lite) and line 172 (for DE0-CV) in golden\_top.v; fixed the order of components listed in the beginning of Lab 1.5 and emphasized double checking components were named properly |
| 07/06/2018 | H. Martinez | Edited seven segment screen code; cleaned up syntax and added console text for clarity |
| 08/22/2018 | H. Martinez |  |
| Transferred from .docx to LATEX; updated figure numbers and enforced cross referencing; revised minor grammar issues; formatted according to Intel branding guidelines |
| 08/08/2019 | R. Nevin | Fixed incorrect URLs and product names |
| 08/03/2021 | L. Landis | Adjusted for Hands-On lab + DE1-SoC/CVGX Starter Kits |

Table 2: Revision Control History