



EMBEDDED FPGA DESIGN WITH THE NIOS II PROCESSOR SHORTENED EDITION

To learn more, visit <http://fpgauniversity.intel.com>.

© Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. 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.

CONTENTS

Lab Overview	2
Lab Notes	3
Design Flow	5
Objective of the "Hello World" Lab	6
Get Started with Quartus	7
Part 1: Hardware Design	9
Lab 1: Building Your Platform Designer Based Processor System	9
1.1: Adding the JTAG UART Component	9
1.2: Connecting the System Components Together	10
Lab 2: Building the Top Level Design	13
Part 2: Software Design	18
Lab 1: Creating the Software for the "Hello World" design	18
Lab 2: Downloading the Hardware Image to the Development Kit	22
Lab 3: Using the Seven Segment Display	26
Lab Summary	28
Appendix	29
List of Figures	29
List of Tables	29
Revision History	30

LAB OVERVIEW

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 textithard processor system. The Nios II processor is supported by a rich set of peripherals and intellectual property (IP) blocks built 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 for 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 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 it, 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 a 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 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 mappin
- 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
- Launching of Software Build Tools (Eclipse) for Nios II

To download Quartus Prime Lite, follow these instructions:

- ☐ Visit this site: <https://fpgasoftware.intel.com/?edition=lite>.
- ☐ Select version 18.1 and your PC's operating system.
- ☐ For the smallest installation and quickest download time, select only the fields shown on the following page in Figure 1. Under Quartus Prime Lite Edition, select only the Quartus Prime, uncheck ModelSim. Under devices select **MAX 10 FPGA device support**. Uncheck the rest.

Select All



Quartus Prime Lite Edition (Free)

☒ **Quartus Prime (includes Nios II EDS)**
 Size: 1.7 GB MD5: F0D752D67B18C89FBC0043CEE676896D

☐ **ModelSim-Intel FPGA Edition (includes Starter Edition)**
 Size: 1.1 GB MD5: 7FDBE5899A9929AEDD517F410079AA35

Devices

You must install device support for at least one device family to use the Quartus Prime software

☐ **Arria II device support**
 Size: 499.6 MB MD5: D87CA20C91596BC8C7BCE84253D956B7

☐ **Cyclone IV device support**
 Size: 466.6 MB MD5: 9E32B85F83A440604154BD729B143D5C

☐ **Cyclone 10 LP device support**
 Size: 266.1 MB MD5: 72AAE619D358FF6B8E42849B3BFCFADD

☐ **Cyclone V device support**
 Size: 1.1 GB MD5: 75F5029A9058F64F969496B016EE19D4

☐ **MAX II, MAX V device support**
 Size: 11.4 MB MD5: ED990775F76C35D308877F27A30B7555

☒ **MAX 10 FPGA device support**
 Size: 330.9 MB MD5: E87E56DAB144529EFC515C2452F1B1FE

Download Selected Files

[Troubleshoot download problems](#)

Note: The Quartus Prime software is a full-featured EDA product. Depending on your download speed, download times may be lengthy.

Figure 1: Quartus Download Page

- ☐ Follow the instructions to activate the Quartus Prime Lite version 18.1 tools on your PC. 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 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.

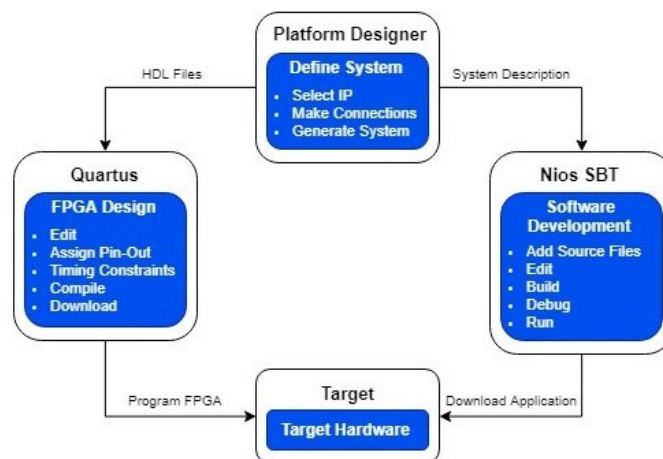


Figure 2: Platform Designer Development Flow

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 THE "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 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. Intel utilizes the Platform Designer network-on-chip interconnect to connect the master and slave devices together. In the interest of time, you are given a nearly completed NIOS II based system to start. The lab starting file will omit the JTAG UART which provides connectivity to the keyboard and display. You will be given detailed instructions on how to connect the JTAG UART to the Avalon bus.

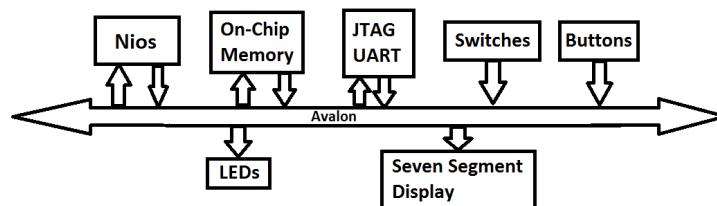


Figure 3: Nios II Based System Used In This Lab

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 for performing the lab.

- Get the associated .zip folder from the design files you downloaded for this lab. The file, **DE10_Lite_qsys_workshop_semicomplete.zip** will contain the semi-completed project.

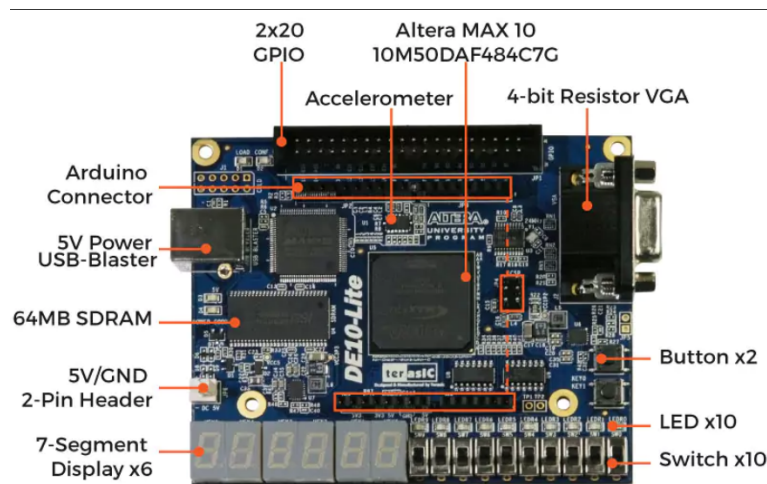


Figure 4: DE-10 Lite

Board	Files
DE-10 Lite	DE10_Lite_qsys_workshop_semicomplete.zip

Table 1: Resource File

- 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. Within the file there are two items: C_CODE and DE10_Lite_semicomplete.qar. **The lab will not work if you do not unzip the files!**
- Double click on the **DE10_Lite_qsys_semicompleted.qar** file. [DE10_Lite_qsys_semicompleted.qar](#) 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.
- Select a destination folder where you want your project to be restored, by clicking on ... near the destination folder. *Make sure your destination folder is a C:// drive where your installed Quartus or your documents location where you want your project to be and contains no spaces.* Select **OK** for the first screen that appears when the (.qar) file opens.

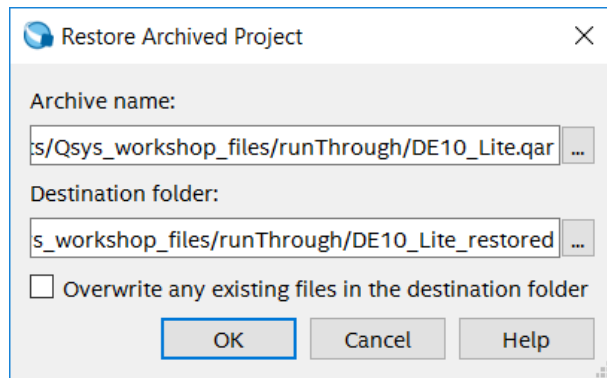


Figure 5: Selecting Archive Name and Destination Folder for the .qar file.

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

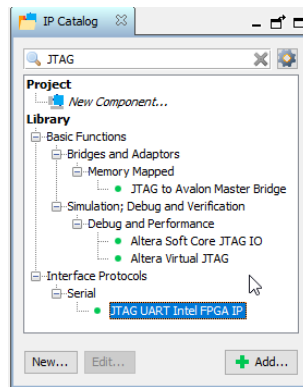


Figure 7: JTAG UART IP

- ☐ Keep the default settings and click **Finish**:

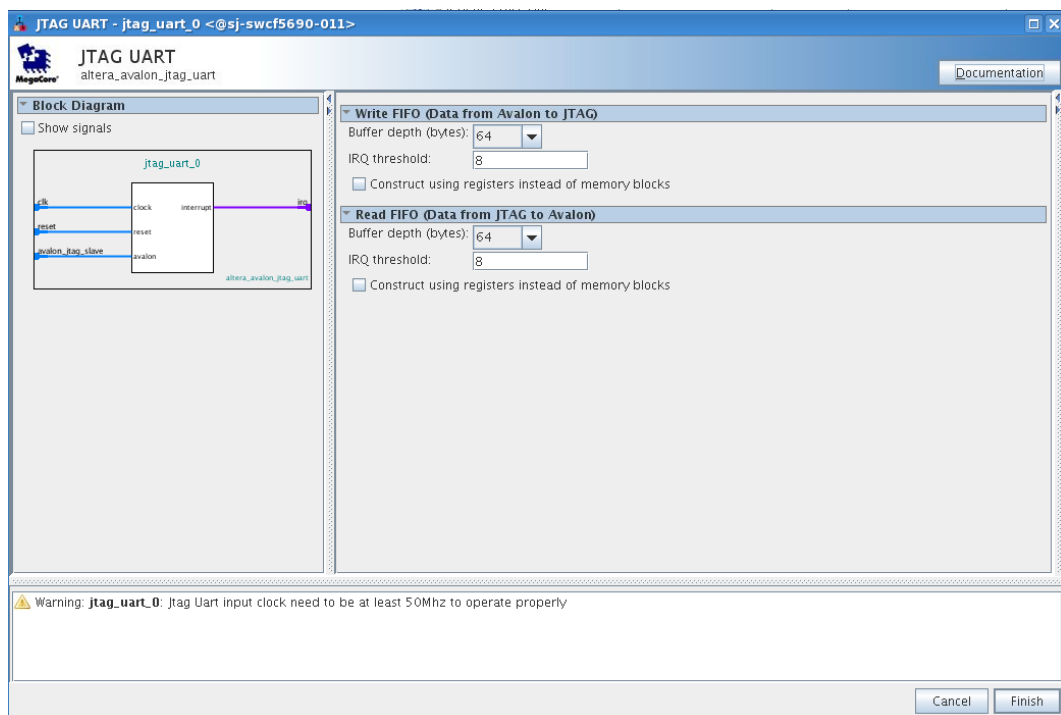


Figure 8: JTAG UART Configuration Panel

- ☐ Rename the JTAG UART by right-clicking, pressing rename, and renaming to **jtag_uart**.

1.2: Connecting the System Components Together

The next step consists of making the appropriate connections between the components and the JTAG UART component within Platform Designer.

- ☐ Underneath **jtag_uart** select the **clock input**.

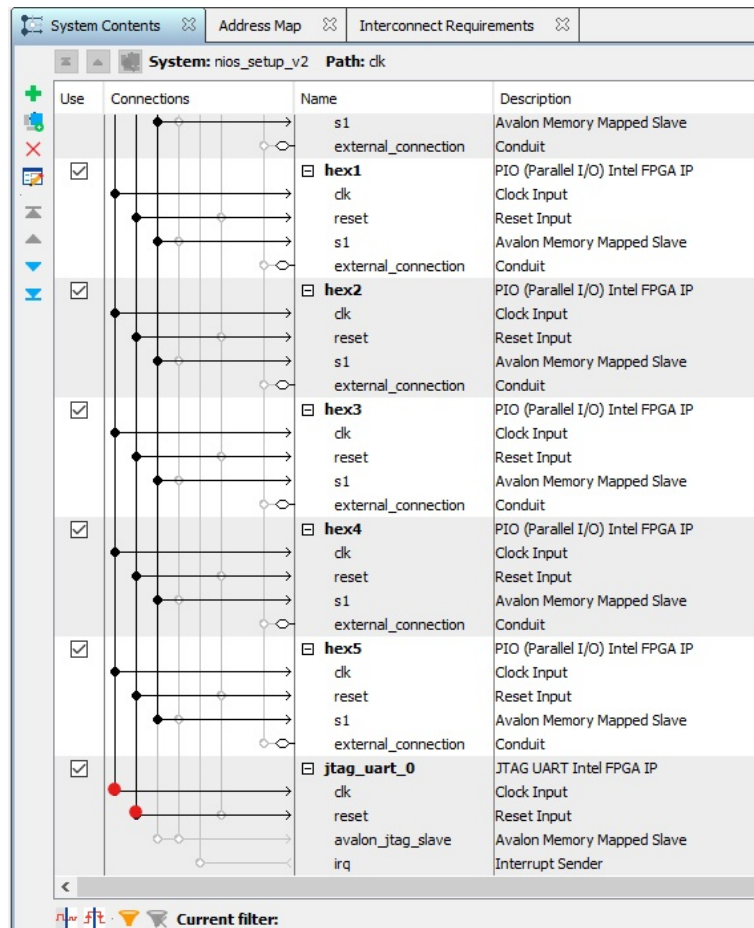


Figure 9: JTAG Clock and Reset Connection

- Make the connection between the clk component's output and the JTAG UART's clock input by clicking on the small circle on the line that intersects with the component. Do the same for the reset signal of the JTAG UART component and the reset signal from the clk.

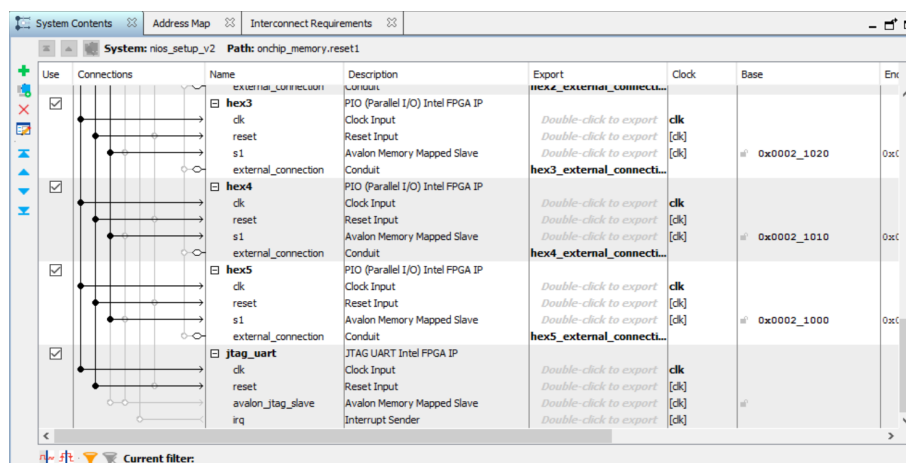


Figure 10: JTAG UART Connections

- The next connections to make are the processor interrupt request (IRQ) signals. Make this connection by selecting the empty bubble for the JTAG UART IRQ signal. 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.
- ☐ Last you need to connect the **avalon_jtag_slave** with the **data master** connection coming out of the cpu.
- ☐ Your final connections should look like the following figure.

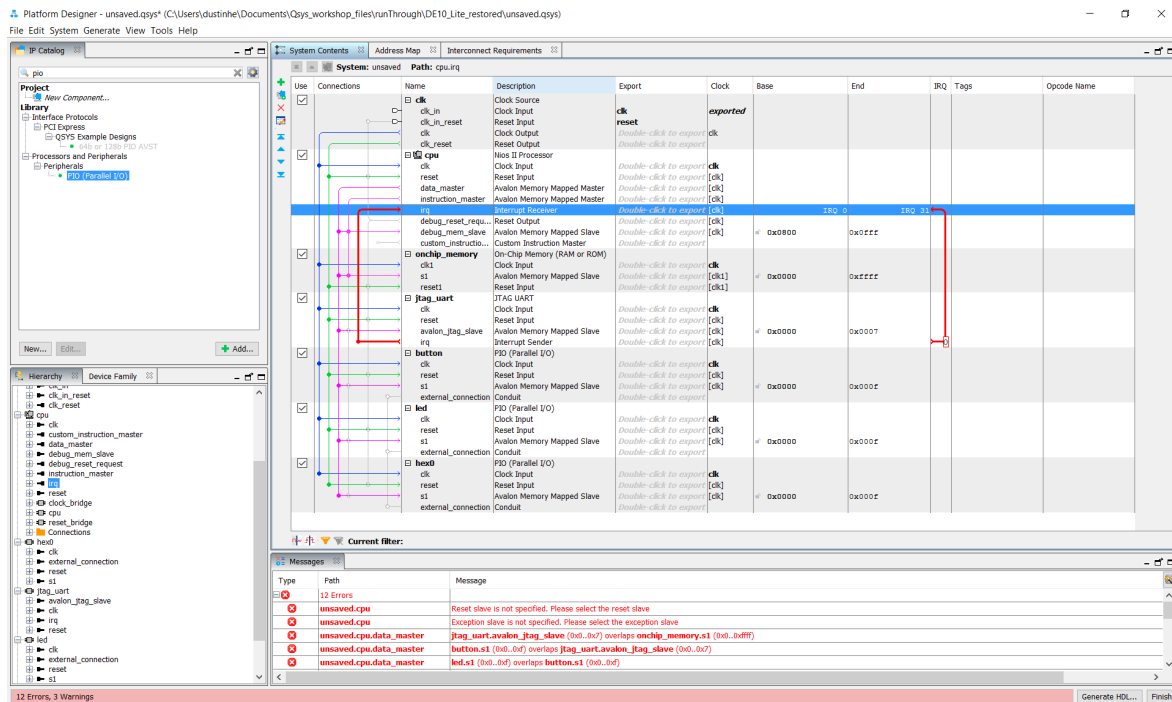


Figure 11: System Contents After Interrupt Connections

- ☐ 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** The information is saved in a .qsys file.
- ☐ 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 10 should appear.
- ☐ Click **Generate** on the panel that appears.
- ☐ When the file generation is complete, click **Finish** to exit the Platform Designer window.

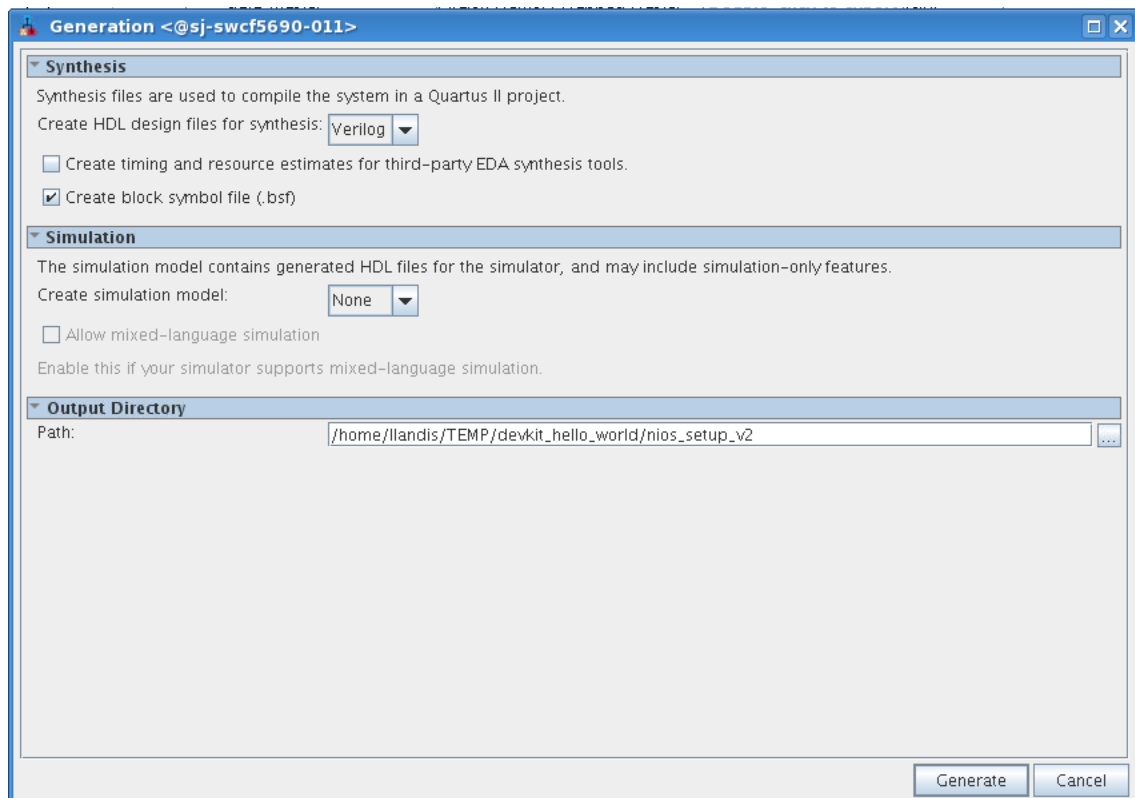


Figure 12: HDL Generation Panel

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

Lab 2: Building the Top Level Design

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

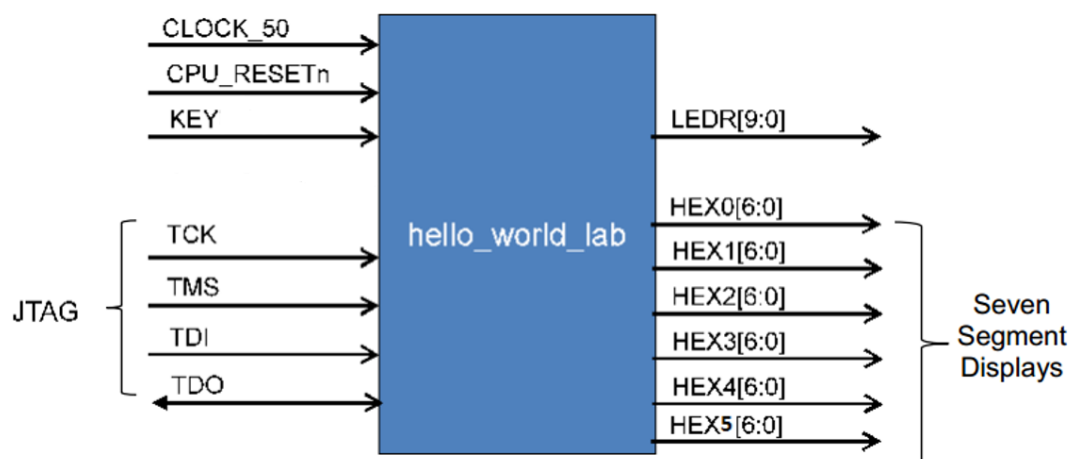


Figure 13: Block Diagram of hello_world_lab Design

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 **DE10_LITE_Golden_Top** if you are using a DE-10 Lite development kit.

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.

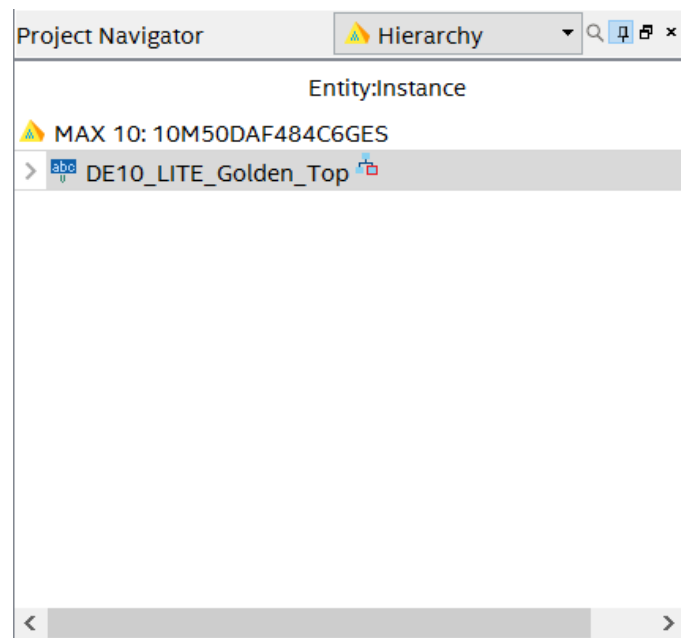


Figure 14: Project Navigator View of Golden Top File

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 31.

```

nios_setup_v2 u0 (
    .clk_clk (MAX10_CLK1_50), //clk.clk
    .led_external_connection_export (ledFromNios[9:0]), //led_external_connection.export //CHANGED TO LEDR
    .reset_reset_n (1'b1), //reset.reset_n
    .button_external_connection_export (KEY[1:0]), //button_external_connection.export
    .switch_external_connection_export (SW[9:0]), //switch_external_connection.export
    .hex0_external_connection_export (HEX0), //hex0_external_connection.export
    .hex1_external_connection_export (HEX1), //hex1_external_connection.export
    .hex2_external_connection_export (HEX2), //hex2_external_connection.export
    .hex3_external_connection_export (HEX3), //hex3_external_connection.export
    .hex4_external_connection_export (HEX4), //hex4_external_connection.export
    .hex5_external_connection_export (HEX5) //hex5_external_connection.export
);

//Uncomment the line below by deleting the "/"
assign LEDR[2] = SW[2];

//ignore what is below here

wire [9:0] ledFromNios;
assign LEDR[1:0] = ledFromNios[1:0];
assign LEDR[9:3] = ledFromNios[9:3];
assign HEX0[7] = 1'b1;
assign HEX1[7] = 1'b1;
assign HEX2[7] = 1'b1;
assign HEX3[7] = 1'b1;
assign HEX4[7] = 1'b1;
assign HEX5[7] = 1'b1;
endmodule

```

Figure 15: Contents of nios_setup_v2_inst.v

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 (DE10_LITE_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.) Press the “...” button to open the file dialog box.
 - 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 32 for what your Add/Remove Files window should look like. (There may be an extra .sdc file in the list. This is fine.)

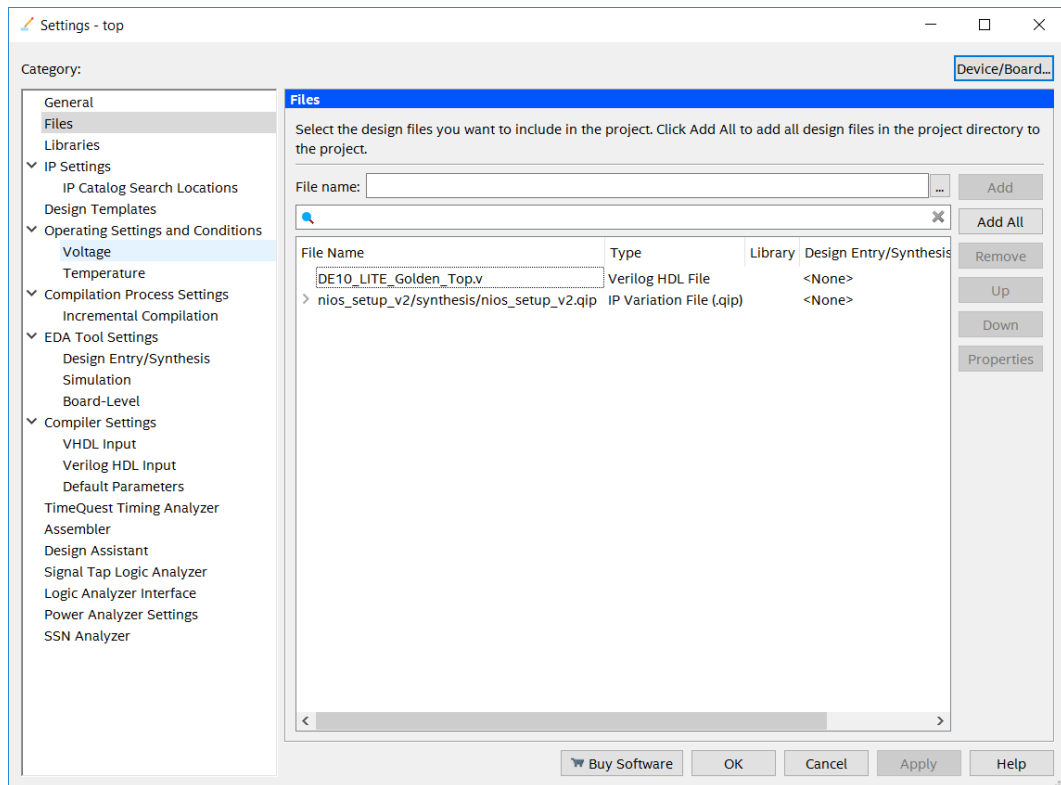


Figure 16: Quartus Add/Remove Files Pane

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**.

Assignment Editor									
Filter on node names: *									
	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
85	✓		HEX0[0]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
86	✓		HEX0[1]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
87	✓		HEX0[2]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
88	✓		HEX0[3]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
89	✓		HEX0[4]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
90	✓		HEX0[5]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
91	✓		HEX0[6]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
92	✓		HEX0[7]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
93	✓		HEX1[0]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
94	✓		HEX1[1]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
95	✓		HEX1[2]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
96	✓		HEX1[3]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
97	✓		HEX1[4]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
98	✓		HEX1[5]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
99	✓		HEX1[6]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
100	✓		HEX1[7]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
101	✓		HEX2[0]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
102	✓		HEX2[1]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
103	✓		HEX2[2]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
104	✓		HEX2[3]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
105	✓		HEX2[4]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
106	✓		HEX2[5]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
107	✓		HEX2[6]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		
108	✓		HEX2[7]	I/O Standard	3.3-V LVTTTL	Yes	DE10..._Top		

Figure 17: Quartus Assignment Editor Window

Figure 33 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 34.

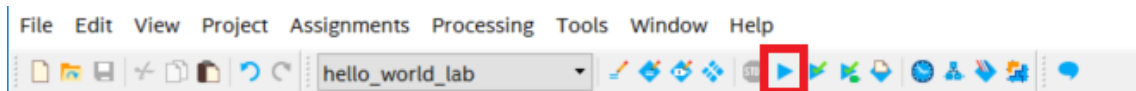






Figure 18: Compilation Button on Quartus Toolbar

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.

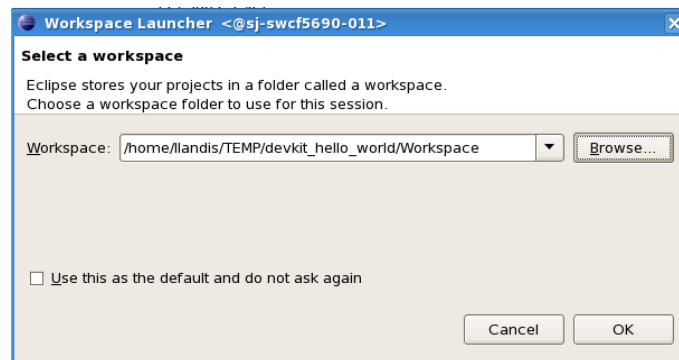


Figure 19: Initial Workspace Setup

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

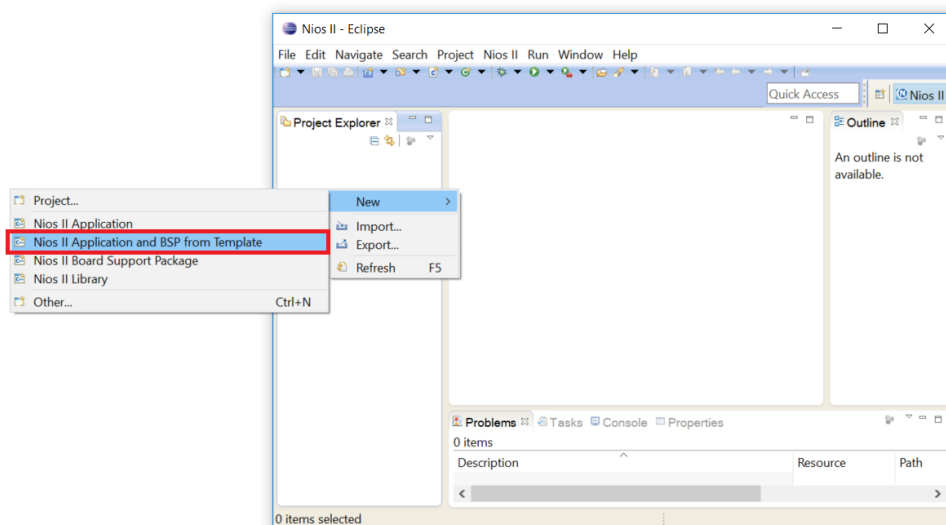


Figure 20: Creating the Initial Project in the Eclipse SBT

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

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**.

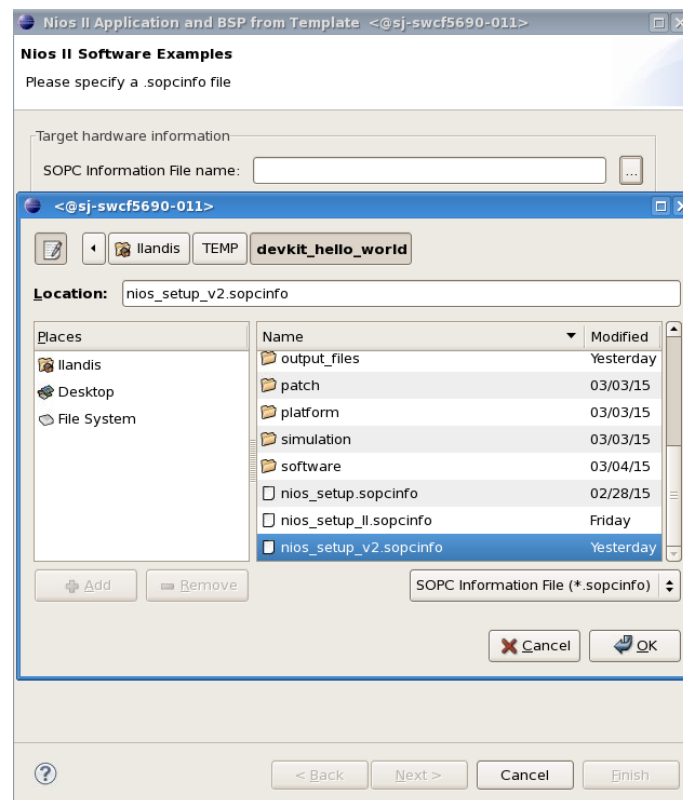


Figure 21: Navigating to the .sopcinfo File

- ☐ 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.

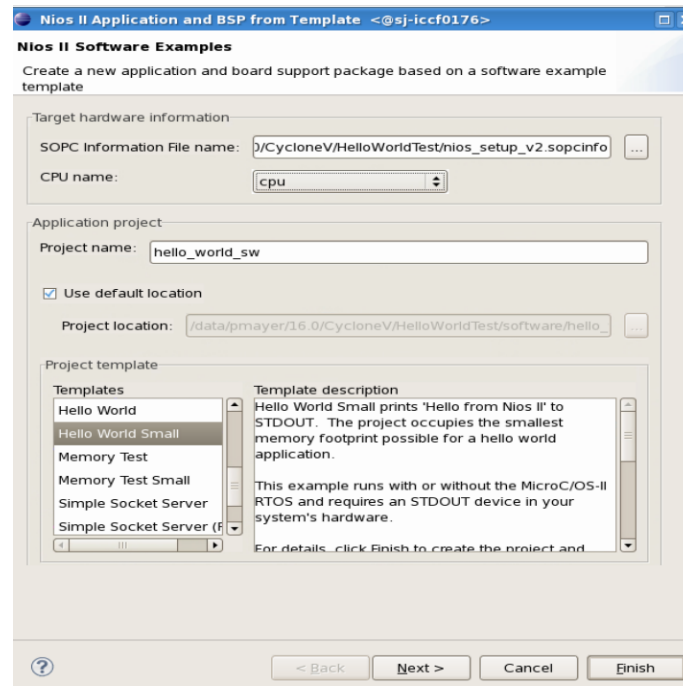


Figure 22: Completing the Nios II Software Examples Setup Screen

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

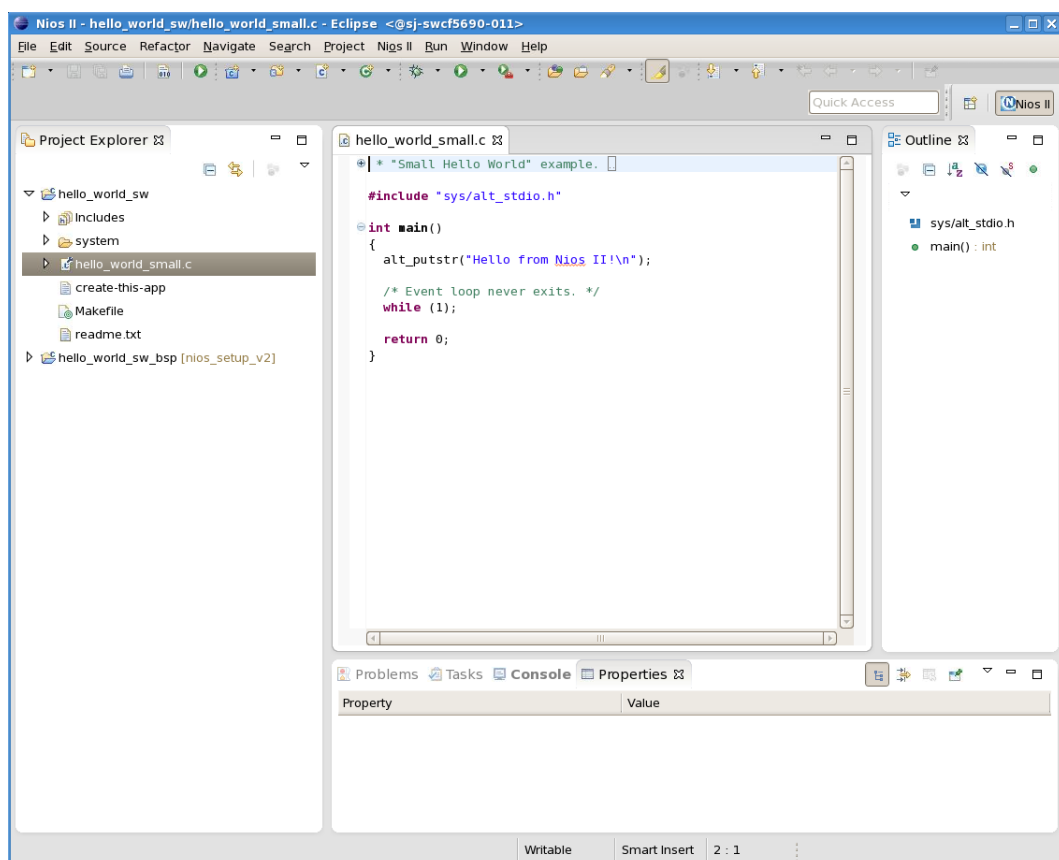


Figure 23: Eclipse Window of “hello_world_small.c”

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>.

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 step 11.
- ☐ 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_INTELPSG_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_INTELPSG_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.

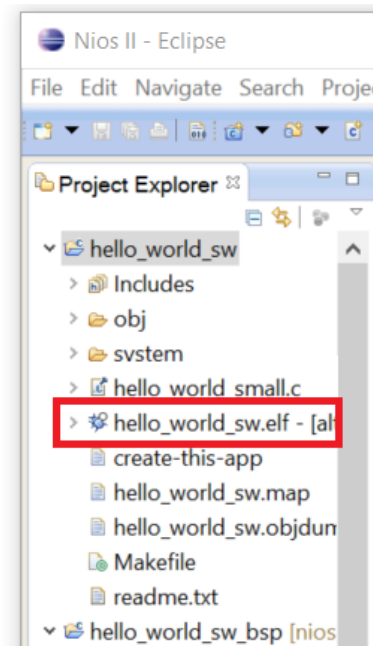



Figure 24: Window View of “hello_world_sw.elf”

Lab 2: Downloading the Hardware Image to the Development Kit

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:

- ☐ To begin, make sure you connect your board to your computer via a USB cable. Depending on your board model, you may need to plug your board into power.
- ☐ Hit the windows key  and type **Device Manager**.
- ☐ Click on the Device Manager tile that appears.
- ☐ Navigate to the **Other Devices** section of the device manager and expand the section.

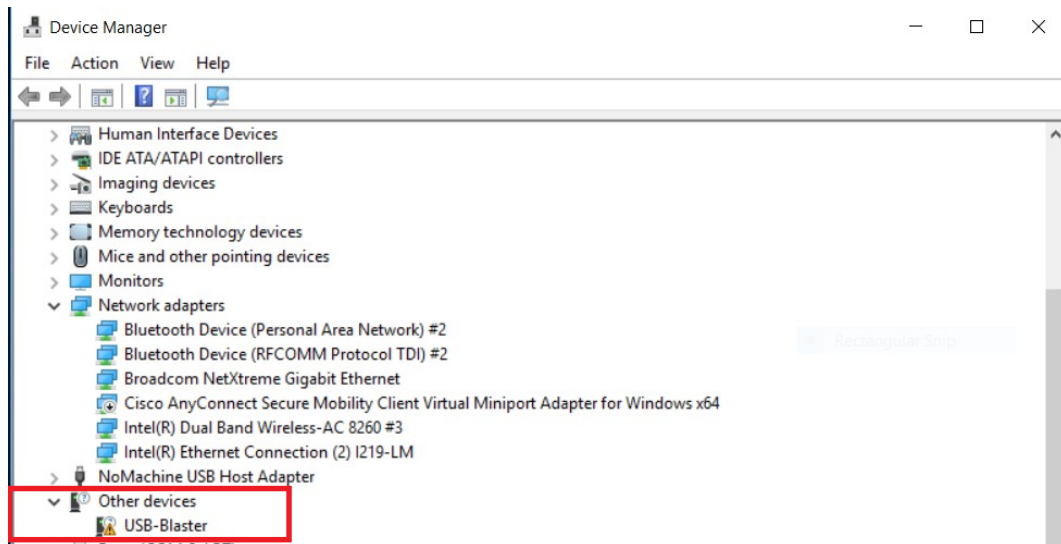


Figure 25: Device Manager Showing USB Blaster Drivers Not Installed

- ☐ Right click the USB-Blaster device and select **Update Driver Software**".
- ☐ Choose to browse your computer for driver software.

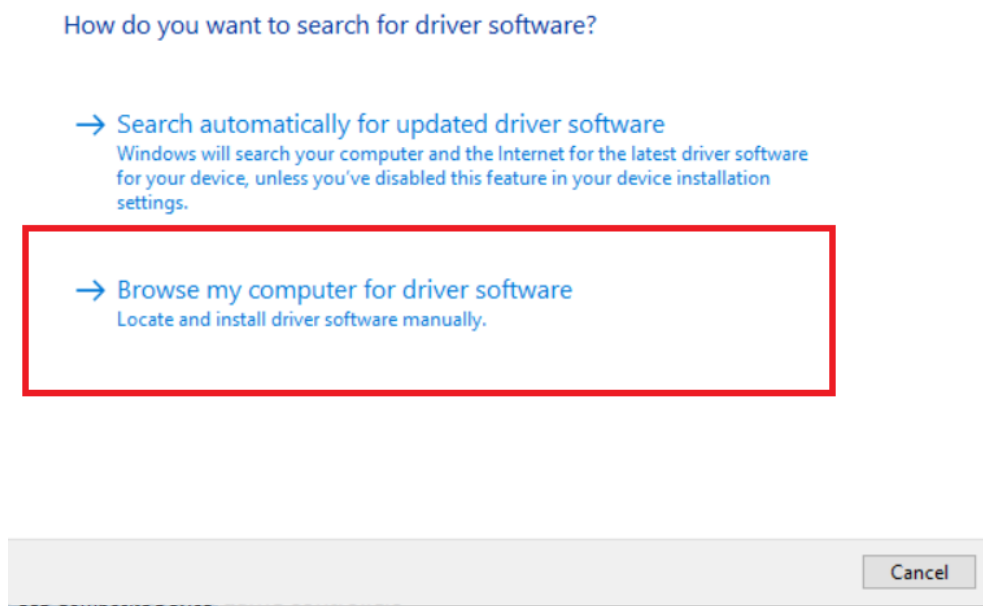


Figure 26: Selecting to Browse for Driver Software Directory

- ☐ Navigate to the path shown in Figure 42. This should be the path where you have installed Quartus on your computer.

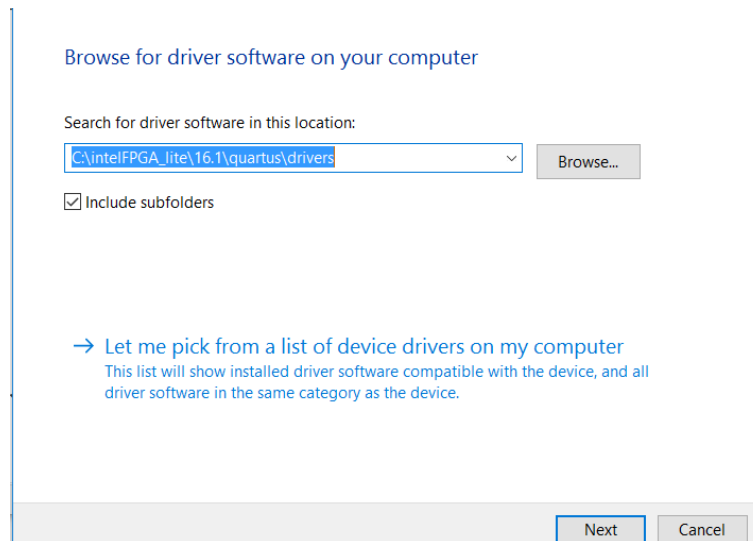


Figure 27: Directory Containing USB Blaster Drivers

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

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	Erase	ISP CLAMP
output_files/hello...	10M50DAF48...	003DB685	003DB685	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 28: Program/Configure Checkbox

- ☐ In the first row under **Program/Configure** click in the check box as shown in Figure 44 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**.

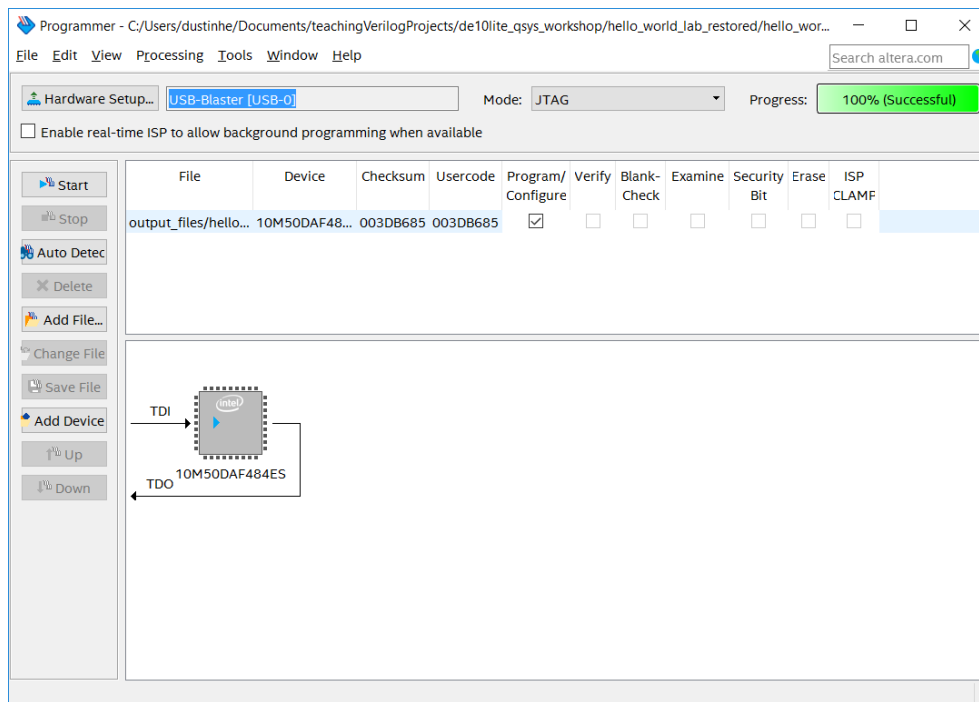


Figure 29: Programmer Progress Successful

- ☐ 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. 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 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.

- ☐ 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 44.
- ☐ 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**.

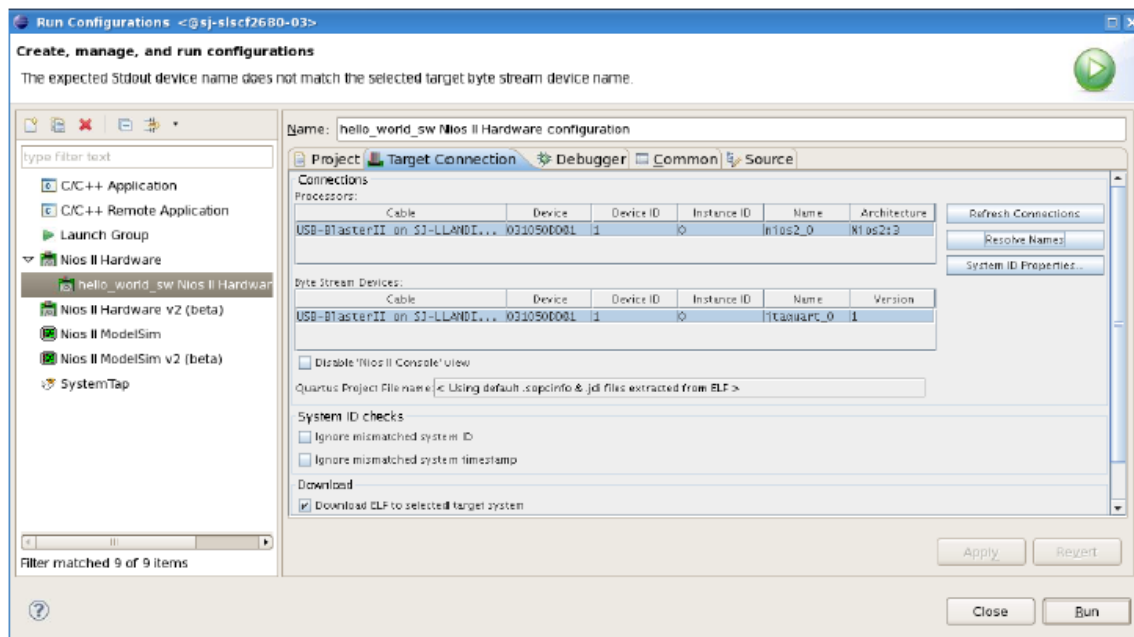


Figure 30: Eclipse SBT Tools after Connection is made to the USB-Blaster

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

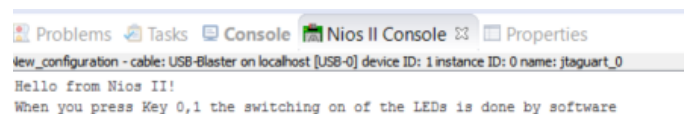


Figure 31: "Hello from Nios II!" on Nios II Console Tab

- 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.

KEY CONCEPTS:

- When you push buttons 0 and 1, LEDs 0 and 1 will light up. This connectivity is made through **software**.
- When you flick switch SW2, LED 2 will light up. This connectivity is made through **hardware**.

Lab 3: Using the Seven Segment Display

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 Nios2 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 **Build Project**.
- ☐ 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 Nios2 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 <https://fpgauniversity.intel.com> to discover more embedded systems, NIOS, and software development trainings and reference designs from Intel and our technology partners.

APPENDIX

List of Figures

1	Quartus Download Page	4
2	Platform Designer Development Flow	5
3	Nios II Based System Used In This Lab	6
4	DE-10 Lite	7
5	Selecting Archive Name and Destination Folder for the .qar file.	8
6	Platform Designer Main Panel	9
7	JTAG UART IP	10
8	JTAG UART Configuration Panel	10
9	JTAG Clock and Reset Connection	11
10	JTAG UART Connections	11
11	System Contents After Interrupt Connections	12
12	HDL Generation Panel	13
13	Block Diagram of hello_world_lab Design	13
14	Project Navigator View of Golden Top File	14
15	Contents of nios_setup_v2_inst.v	15
16	Quartus Add/Remove Files Pane	16
17	Quartus Assignment Editor Window	16
18	Compilation Button on Quartus Toolbar	17
19	Initial Workspace Setup	18
20	Creating the Initial Project in the Eclipse SBT	18
21	Navigating to the .sopcinfo File	19
22	Completing the Nios II Software Examples Setup Screen	20
23	Eclipse Window of "hello_world_small.c"	20
24	Window View of "hello_world_sw.elf"	22
25	Device Manager Showing USB Blaster Drivers Not Installed	23
26	Selecting to Browse for Driver Software Directory	23
27	Directory Containing USB Blaster Drivers	24
28	Program/Configure Checkbox	24
29	Programmer Progress Successful	25
30	Eclipse SBT Tools after Connection is made to the USB-Blaster	26
31	"Hello from Nios II!" on Nios II Console Tab	26

List of Tables

1	Resource File	7
2	Revision Control History	31

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
09/12/2019	R. Schutz	Edited to create short form lab. Made JTAG UART only component needing to be added

Table 2: Revision Control History