INTRO TO INTEL® FPGAS AND INTEL® QUARTUS® PRIME SOFTWARE

©2018 Intel Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, INTEL, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Intel Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Intel warrants performance of its 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 service.

CONTENTS

Hints and Tricks 3

Lab 1: Obtaining the Quartus Prime Lite Design Tools 4
- Background . . . 4
- Installation . . . 4

Lab 2: New Project Wizard 6
- Summary . . . 6
- Lab Instruction . . . 6
  - 2.0: Navigation of Quartus Prime . . . 6
  - 2.1: New Project Wizard . . . 6

Lab 3: Making Assignments 12
- Summary . . . 12
- Lab Instruction . . . 12
  - 3.0: Creating a New File . . . 12
  - 3.1: Adding Verilog Code . . . 13
  - 3.2: Assigning Pins . . . 14
  - 3.3: Compiling Your Code . . . 17
  - 3.4: Installing and Using the USB Blaster to Download Your Design to the FPGA 18
  - 3.5: Programming Your Design into the FPGA . . . 19
  - 3.6 Testing Your Design . . . 21

Lab 4: 2 to 1 Multiplexer 22
- Summary . . . 22
- Lab Instruction . . . 22
  - 4.0: Creating a Revision . . . 22
  - 4.1: 2-1 Mux Verilog Code . . . 22
  - 4.2: Revision Control . . . 25
  - 4.3: Checking Pin Locations and Editing (If Necessary) . . . 25
  - 4.4: Downloading Your Design to Your Device . . . 26
  - 4.5: Viewing Design Schematic . . . 26
  - 4.6: Working 2 to 1 MUX Verilog Code . . . 27

Lab 5: Knight Rider 28
- Summary . . . 28
- Lab Instruction . . . 28
  - 5.0: Knight Rider Verilog Code . . . 28
  - 5.1: Creating “knight_rider.v” . . . 30
  - 5.2: Debugging Code . . . 31
  - 5.3: Assigning Pins with the Pin Planner Tool . . . 31

5.4: Downloading Your Code to Your Device 31
5.5: More Debugging 32
5.6: Even More Debugging! 33

Lab 6: Test Your Knowledge **34**

Appendix **36**
List of Figures 36
List of Tables 36
Revision History 37

HINTS AND TRICKS

Some helpful things to keep in mind. Refer to these if you have problems!

- You should use the Lite version of the Intel Quartus Prime software. This version requires no license. The standard version is fine as well, but that version needs a license that is not provided through this lab work. The Intel® Quartus® Prime Pro Edition software will not work as it does not support the target hardware.
- When Intel Quartus Prime software runs for the very first time, it might ask you about purchasing a license. Select **Run Quartus**. All licenses are free for this lab.
- If something fails to compile, check **Top Level Entity Setting** → **Setting** → **Top Level Entity** and make sure that the module **<design>** matches your top level entity. This includes Verilog file names that don't match module names with case-sensitivity.
- If Programmer fails to program on the first try, hit **Start** twice.
- The errors in the Knight Rider Lab code are intentionally designed to give you an opportunity to practice debugging. Study the code carefully to fix errors.
- If the Knight Rider LEDR[0] is the only LED that turns on, you have not assigned the CLOCK_50 pin properly in your assignments.
- Check the LEDR[0] and LEDR[9] pins carefully in the Knight Rider Lab and see if they sequence properly. If not, study the code carefully!
- Sometimes copying and pasting from text files into Quartus's TLC console can have carriage return formatting errors. Links are provided with the code to help you solve this problem. If you see run-on lines with no carriage returns, you need to either copy code over line by line or add the appropriate file to your project. This is not specifically documented in the lab flow.

LAB 1: OBTAINING THE QUARTUS PRIME LITE DESIGN TOOLS

Background

A field-programmable gate array, or FPGA, is a digital semiconductor that can be used to build a wide variety of electronic functions. These data center accelerators, wireless base stations and industrial motor controllers to name but a few common applications. This is because FPGAs can be infinitely reconfigured to perform different digital hardware functions, which also makes for an excellent learning platform.

To configure an FPGA, first you describe your digital electronics with either a Hardware Description Language (HDL), such as Verilog or VHDL, or a schematic. Then you assign the "pins" of your FPGA based on how the Printed Circuit Board (PCB) connects the FPGA to various peripheral components on your board. Some examples of peripherals are switches, LEDS, memory devices and various connectors. Finally, you "compile" your design and program the FPGA to perform the function you have specified in the HDL or schematic.

The first step in this lab is to download the Intel FPGA design tools: the Quartus Prime Lite Edition. For lower complexity Intel FPGA devices, the Quartus Prime Lite design tools are entirely free. This FPGA development kit costs $55/$85 for the academic/public price. There are numerous labs and online trainings you can take based on this full-featured kit.

This training class assumes you have some prerequisite knowledge of how computers and digital electronics work, but by no means do you need an electrical engineering degree to follow along this introductory course.

Installation

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 (formerly Qsys) 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: http://fpgasoftware.intel.com/?edition=lite.

☐ Select version 17.1 (or higher) and your PC's operating system.

☐ For the smallest installation and quickest download time, select only the fields shown below in Figure 1.

☐ Follow the instructions to activate the Quartus tools on your PC.

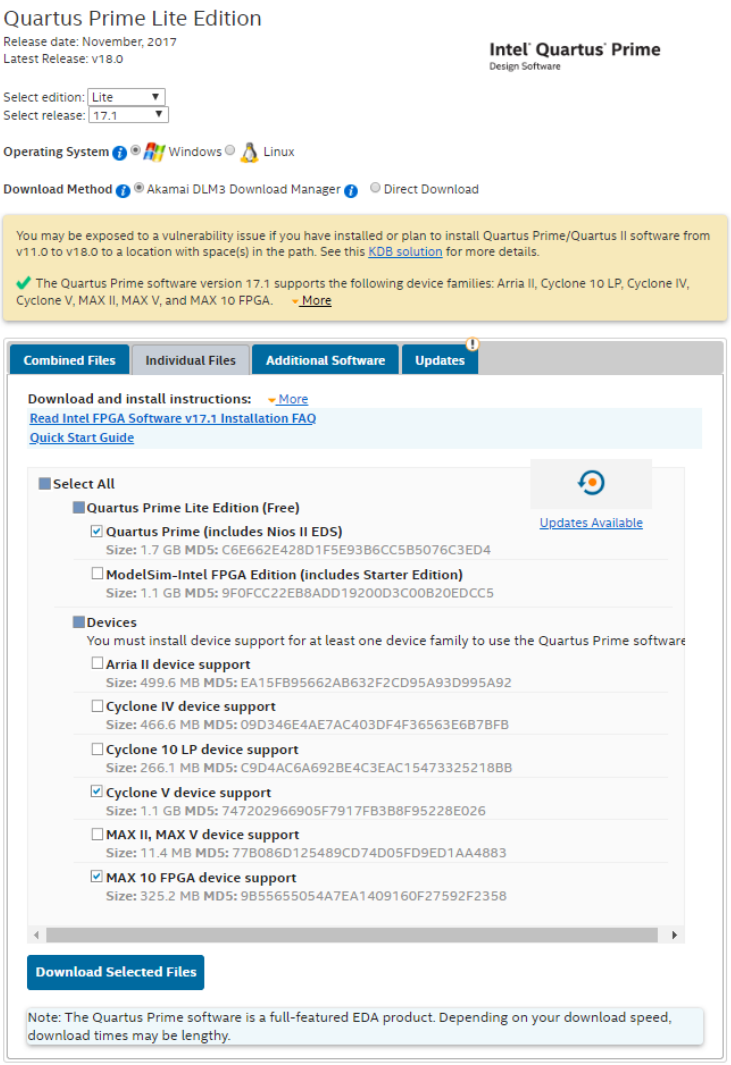

Figure 1: Quartus Prime Lite Minimum Required Files to Download

LAB 2: NEW PROJECT WIZARD

Summary

This is a short lab that completes the basic project setup. At the end of this lab, you will be able to start a new project using New Project Wizard in Quartus Prime Software. There are other related tutorial links provided for you to learn more about the software.

Lab Instruction

2.0: Navigation of Quartus Prime

☐ Open the tools by double clicking the Quartus Prime icon: . You should now see something similar to Figure 2.

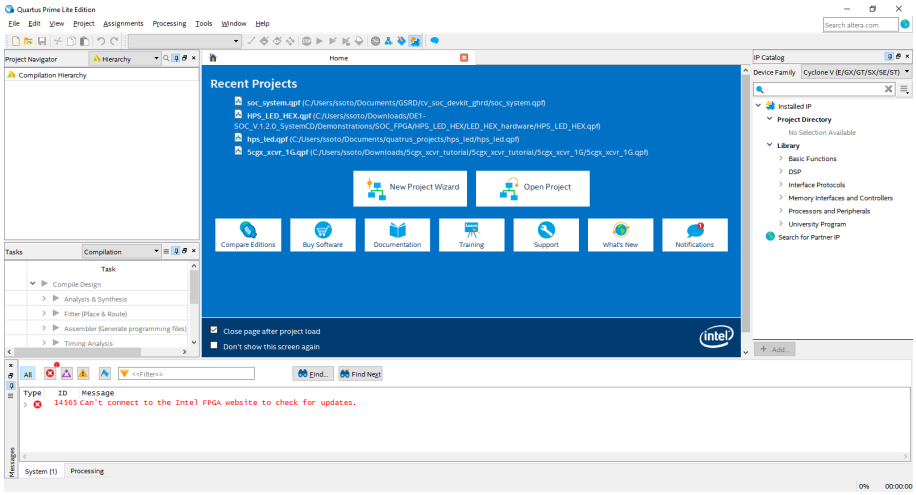

Figure 2: Quartus Prime main window

2.1: New Project Wizard

☐ In the main toolbar of Quartus, navigate to the **File** drop down menu and **New Project Wizard**.

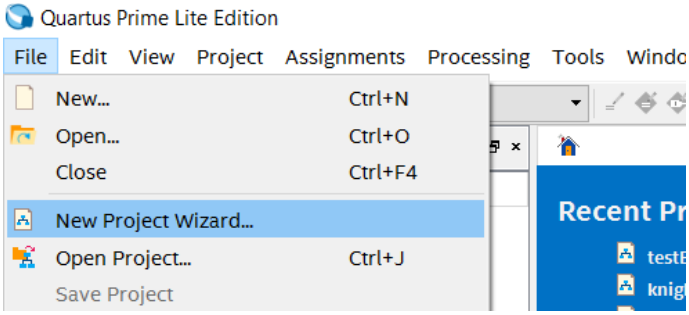

Figure 3: Quartus Prime file menu

☐ **Pane 1: Introduction**. There is nothing to configure on this pane. Click **Next** to continue.

☐ **Pane 2: Directory, Name, Top-Level Entity**.

- ☐ Fill in with a directory of your choice. It is recommended to be a personal directory, and not a directory under Quartus installation which is the default.
- ☐ Call the project **Lab** and the top level entity **switch_to_led**.
  *The name of the top-level design entity is **case-sensitive** so ensure that you type the name in all lower-case.* See Figure 4 below for a completed Pane 2.

Note: the screen shots will have a different directory than what you will use for your project. This is fine!

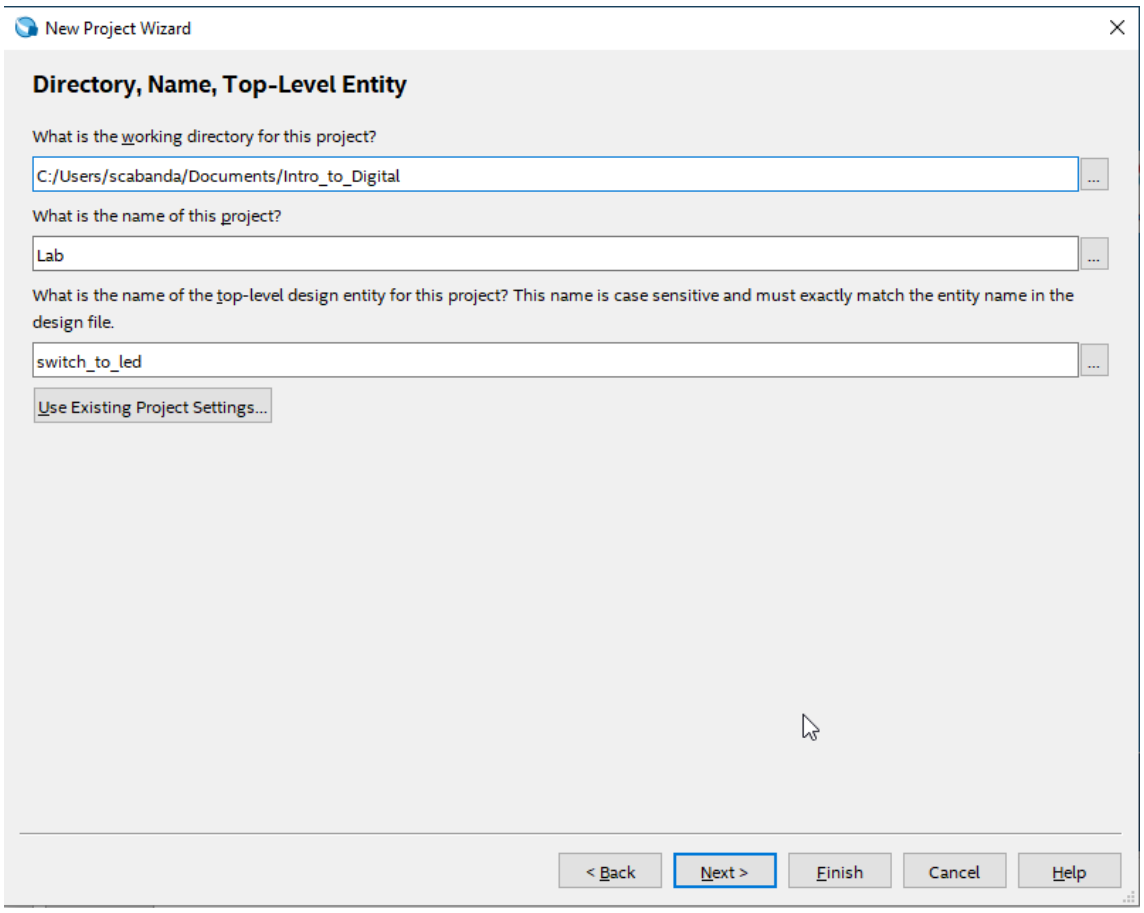

Figure 4: Pane 2 of the New Project Wizard

☐ **Pane 3: Project Type**. Select **Empty project**. Click **Next**.

☐ **Pane 4: Add Files**. Click **Next**. We will add project source files later.

☐ **Pane 5: Family, Device, and Board Settings**.
First take a look at your development board. You should notice a part number on the FPGA chip. This number is *very* difficult to see. If you are having trouble reading the part number on your FPGA chip, try using your phone flashlight to illuminate the FPGA and

better see the part number, or read the part number from the development kit sticker on the side of the box.

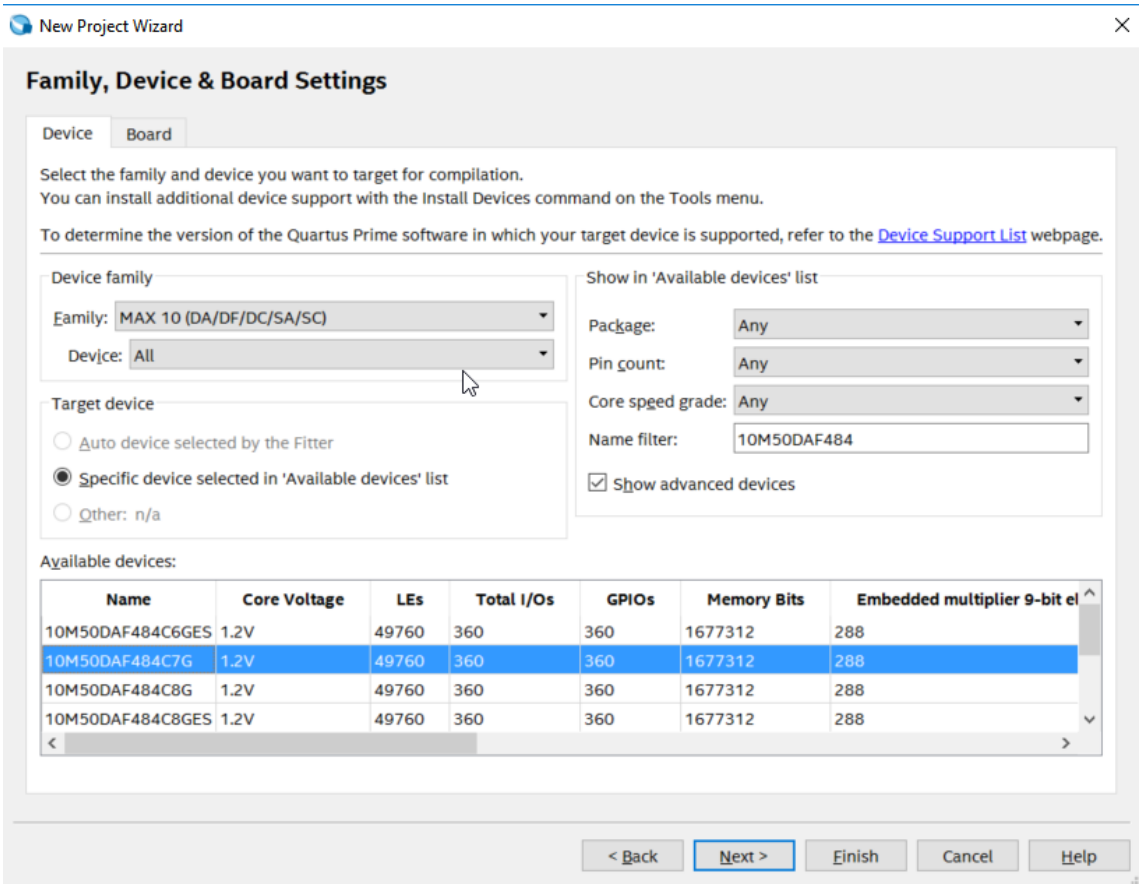

Figure 5: Correct settings for the DE10-Lite Development Board

- ☐ Make sure the tab is set to **Device**. Type this part number in the **Name filter** and choose the device in the **Available devices** panel.
- ☐ Refer to Figure 5 above for filling out the device number (depending on the board you are using).

☐ **Pane 6: EDA Tool Settings**.Skip this section and click **Next**. This section is only needed if you are using other development software besides Quartus Prime.

☐ **Pane 7: Summary**. Pane 7 should look similar to the image seen in Figure 6 on the following page.

*Note: you have a project called **Lab**, and top-level entity called **switch_to_led**, no files selected (yet) and are using a Max10 FPGA device. If you are using the DE0-CV board your Family Name and Device Settings will differ. With these settings, you are now ready to start designing your FPGA project.*

☐ Click **Finish**.

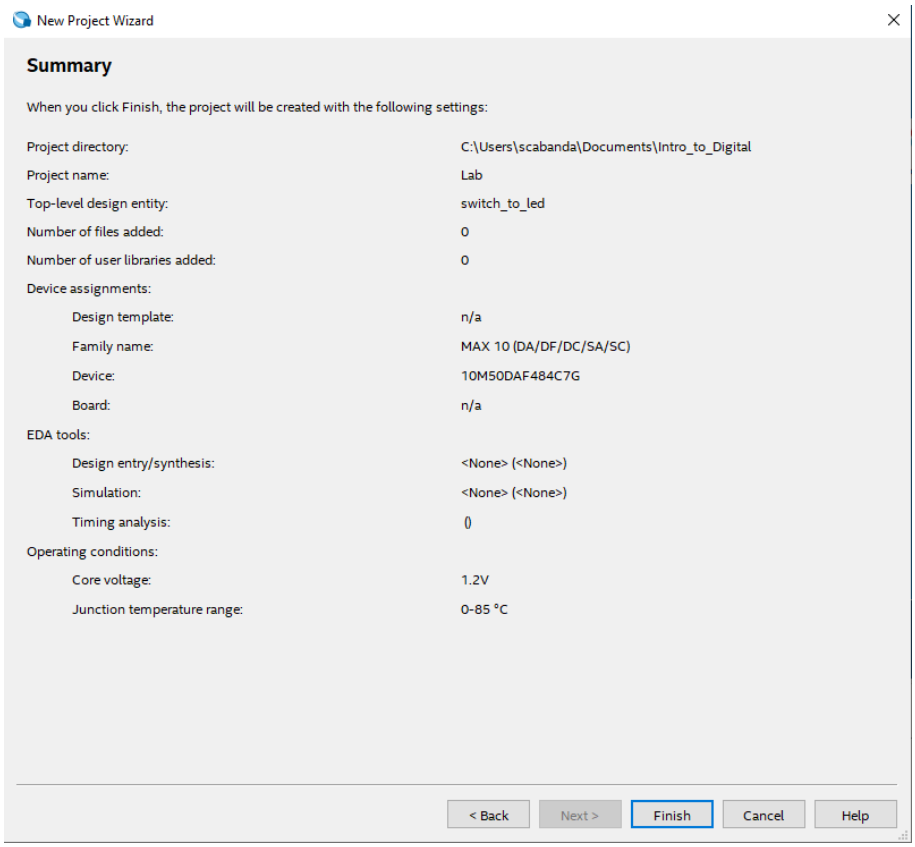

Figure 6: Summary page generated by the New Project Wizard

You should now see something similar to Figure 7. (The **Tool View Window** may just have a gray Quartus Prime screen. This is fine.)

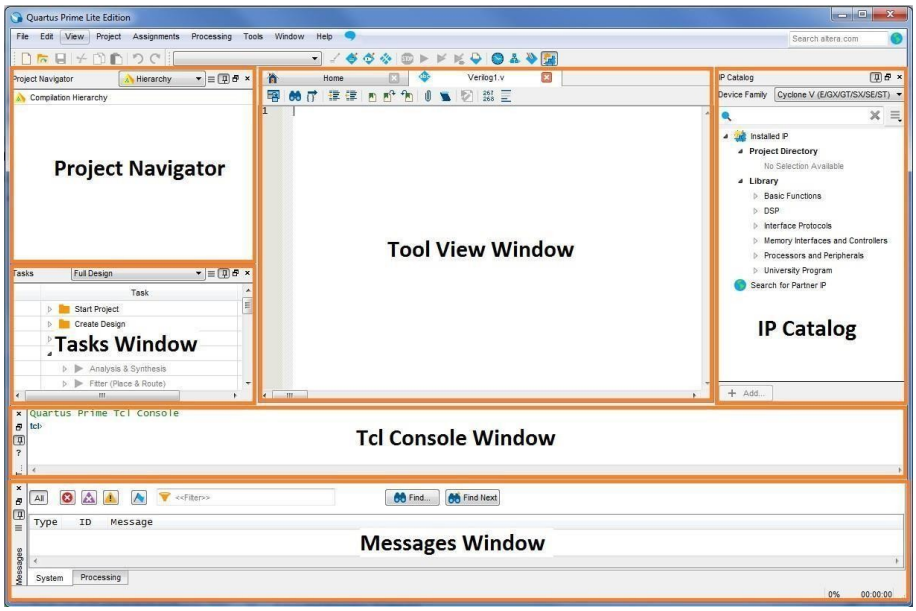

Figure 7: Quartus Prime project window

☐ Some windows may not be shown by default. To customize what windows are shown, click on the **View** tab and look under the **Utlity Windows** drop down as seen in Figure 8.

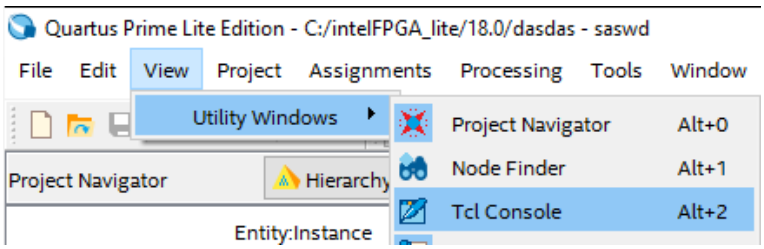

Figure 8: Utility Windows dropdown

If you navigate in Windows Explorer to your project directory, you will see some files and directories created by the New Project Wizard as part of the setup process.

LAB 3: MAKING ASSIGNMENTS

Summary

This lab will step you through the process of a simple design from generating your first Verilog file to synthesize and compile. Synthesis converts your Verilog language file to an FPGA specific "netlist" that programs the programmable FPGA lookup tables into your desired function. Compilation figures out the location of the lookup table cells used in the FPGA and generates a programming image that is downloaded to your Intel FPGA Development kit. At the end of this lab, you will be able to test the functionality of the example digital electronic circuits by toggling the switches and observing the LEDs for proper circuit operation.

Lab Instruction

3.0: Creating a New File

☐ Create a Verilog HDL file. Go the **File** dropdown menu and select **New**.

☐ A window, shown in Figure 9, should pop up. Click on **Verilog HDL File** and then **Ok**.

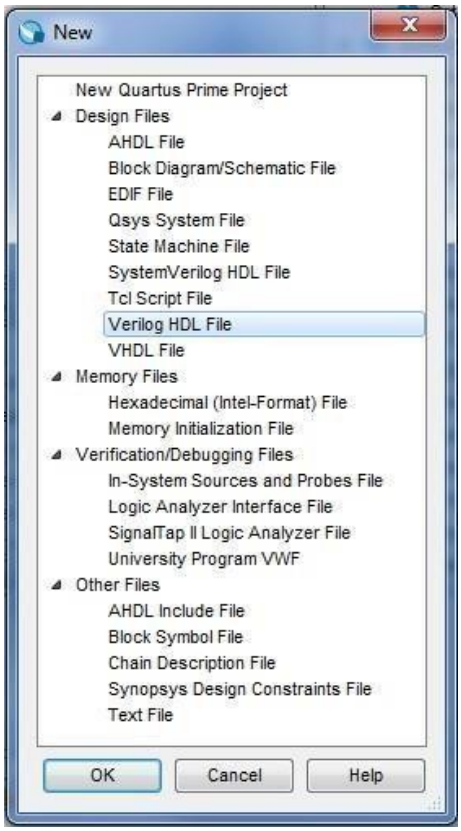

Figure 9: New File Window

3.1: Adding Verilog Code

☐ Create a simple module in your Verilog HDL file by typing in the following code. You can also copy/paste this code from the file switch_to_led.v found in the design files you downloaded for this workshop.

```
module switch_to_led(SW, LEDR); //create module switch_to_led

input    [9:0] SW;  // input declarations: 10 switches
output [9:0] LEDR;  // output declarations: 10 red LEDs
assign  LEDR = SW;  // connect switches to LEDs

endmodule
```

☐ Make sure carriage returns and new lines are in the right location or your code will not compile properly! Verilog treats all blank space (spaces or tabs) the same.
BRAIN EXERCISE : Check your syntax carefully! Can you explain what this circuit does?

☐ Click on **File**, name the file as **switch_to_led** (ensuring case-sensitivity), and click **Save As...** to save your Verilog file.

Next you will run Analysis and Elaboration. Analysis and Elaboration checks the syntax of your Verilog code, resolves references to other modules and maps to FPGA logic. If you see any errors during the Analysis and Elaboration step, carefully review your Verilog code for syntax errors and re-run this step.

☐ To run Analysis and Elaboration, click the **Play** button with a green check mark, shown in Figure 10 below.

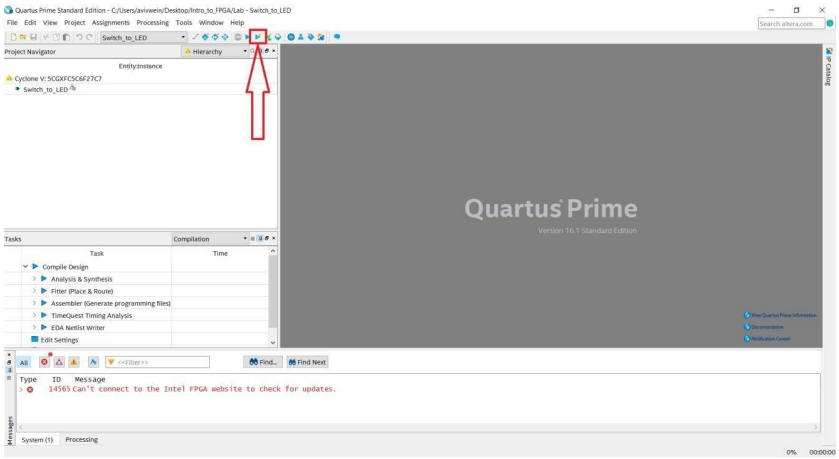

Figure 10: Analysis and Elaboration check mark

3.2: Assigning Pins

By default, Quartus Prime does not know how the FPGA pins on the DE10-Lite or DE0-CV development board are connected to the switches and LEDs used in this circuit. Because our FPGA is already on a PCB, we need to tell Quartus what pins to use. Although Quartus allows you to select a development board with a predefined pinout, this lab shows you how to define your own pinout as an exercise.

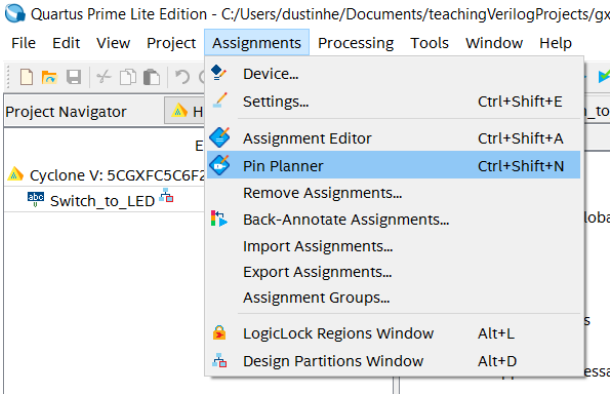

Figure 11: Quartus Assignments menu

The next steps will assign the switches and LED signals in your code to the appropriate pins.

- ☐ Using the main toolbar at the top of the Quartus window, navigate to the **Assignments** dropdown menu as shown above in Figure 11. Click on **Pin Planner** and a window similar to the image in Figure 12 should open.

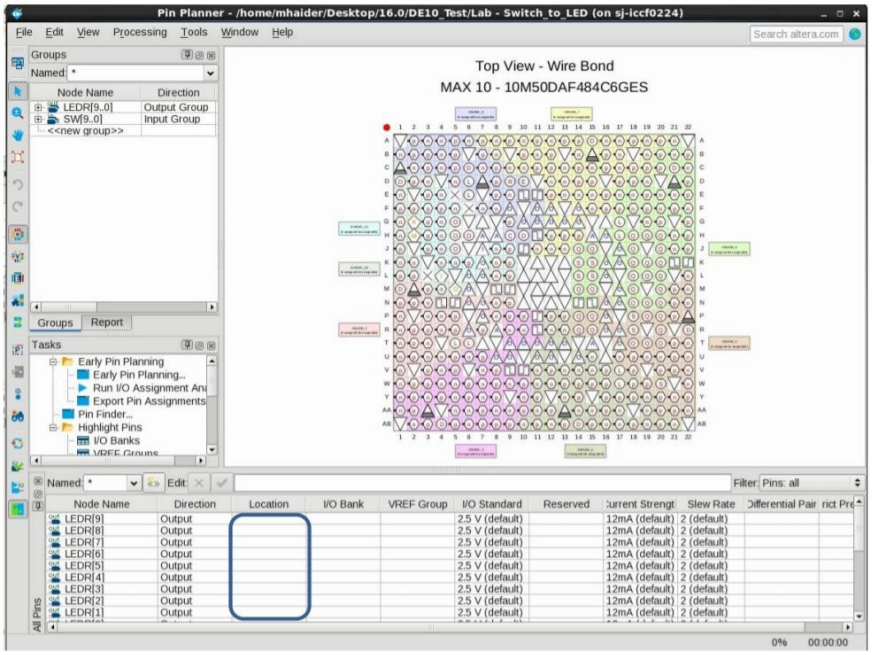

Figure 12: Pin Planner Assignment window

We can see the I/O pins have not been assigned to any locations yet. To make the right pin assignments for this project, we have to look it up in development board vendor's manual.

If you have a DE10-Lite Development Kit, you can go to Terasic's website (https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=218&No=1021&PartNo=1) and download the DE10-Lite User Manual under **Resources** to locate the correct pin assignments. Pin assignments can also be found in **Table 1**.

If you have a DE0-CV Development Kit, you can go to the Terasic's website (https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=921) and download the DE0-CV User Manual under **Resources** to locate the correct pin assignments. Pin assignments can also be found in **Table 2**.

> *Note: The switch_to_led lab does not require the CLOCK signals so you can ignore these for the time being.*

SIGNAL NAME	FPGA PIN #	DESCRIPTION
LEDR[9]	PIN_B11	LED [9]
LEDR[8]	PIN_A11	LED [8]
LEDR[7]	PIN_D14	LED [7]
LEDR[6]	PIN_E14	LED [6]
LEDR[5]	PIN_C13	LED [5]
LEDR[4]	PIN_D13	LED [4]
LEDR[3]	PIN_B10	LED [3]
LEDR[2]	PIN_A10	LED [2]
LEDR[1]	PIN_A9	LED [1]
LEDR[0]	PIN_A8	LED [0]
SW[9]	PIN_F15	Slide Switch[9]
SW[8]	PIN_B14	Slide Switch[8]
SW[7]	PIN_A14	Slide Switch[7]
SW[6]	PIN_A13	Slide Switch[6]
SW[5]	PIN_B12	Slide Switch[5]
SW[4]	PIN_A12	Slide Switch[4]
SW[3]	PIN_C12	Slide Switch[3]
SW[2]	PIN_D12	Slide Switch[2]
SW[1]	PIN_C11	Slide Switch[1]
SW[0]	PIN_C10	Slide Switch[0]
ADC_CLK_10	PIN_N5	10 MHz clock input for ADC (Bank 3B)
MAX10_CLK1_50	PIN_P11	50 MHz clock input (Bank 3B)
MAX10_CLK	PIN_N14	50 MHz clock input (Bank 3B)

Table 1: DE10-Lite Development Board

SIGNAL NAME	FPGA PIN #	DESCRIPTION
LEDR[9]	PIN_L1	LED [9]
LEDR[8]	PIN_L2	LED [8]
LEDR[7]	PIN_U1	LED [7]
LEDR[6]	PIN_U2	LED [6]
LEDR[5]	PIN_N1	LED [5]
LEDR[4]	PIN_N2	LED [4]
LEDR[3]	PIN_Y3	LED [3]
LEDR[2]	PIN_W2	LED [2]
LEDR[1]	PIN_AA1	LED [1]
LEDR[0]	PIN_AA2	LED [0]
SW[9]	PIN_AB12	Slide Switch[9]
SW[8]	PIN_AB13	Slide Switch[8]
SW[7]	PIN_AA13	Slide Switch[7]
SW[6]	PIN_AA14	Slide Switch[6]
SW[5]	PIN_AB15	Slide Switch[5]
SW[4]	PIN_AA15	Slide Switch[4]
SW[3]	PIN_T12	Slide Switch[3]
SW[2]	PIN_T13	Slide Switch[2]
SW[1]	PIN_V13	Slide Switch[1]
SW[0]	PIN_U13	Slide Switch[0]
CLOCK_50	PIN_M9	50 MHz clock input (Bank 3B)
CLOCK2_50	PIN_H13	50 MHz clock input (Bank 7A)
CLOCK3_50	PIN_E10	50 MHz clock input (Bank 8A)
CLOCK4_50	PIN_V15	50MHz clock input (Bank 4A)

Table 2: DE0-CV Development Board

TO ASSIGN PINS

☐ Match the **Signal Name** (1st column) with the **FPGA Pin #** (2nd column) in table above or manual copy. Assign **LEDR[9]** to **PIN_L1** by typing PIN_L1 in the location column in the Pin Planner.

Note: the signal names in your code and names in the manual don't have to match. As long as you connect the names in your design to the proper pin location, your design will be connected properly. In this lab, we have named the signals as a vector so pin names are of the form LEDR[0], LEDR[1] and so on.

☐ An alternate method is to left click on the **Node Name** in the Pin Planner and drag the pin on top of the ball grid map location assigned in the table. Release the pin on the proper location. Hit the escape key and move to the next pin. Assign **LEDR[8]** to **PIN_L2** using this method.

☐ When you finish, you can just close the window – the Planner does not have a Save button, but it will save anyways. *This switch_to_led lab does not require the CLOCK signals so you can ignore these for the time being.* The clock signal is required for the Knight Rider lab, so you will need to assign it prior to compiling that lab.

☐ Last, we will assign the remaining pins using a TCL commands. In Quartus, proceed to **View** → **Utility Windows** → **TCL Console**. You should see a window across the bottom of your Quartus window as shown in Figure 13. If it doesn't show up, try the command again as this toggles the window on and off.

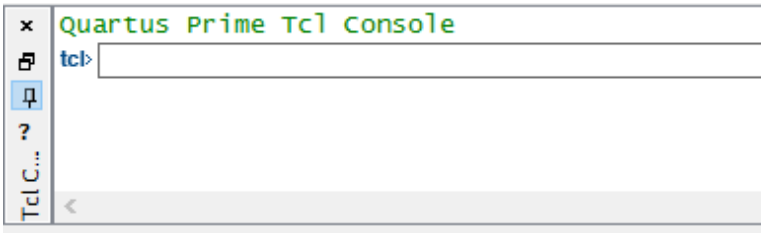

Figure 13: TLC Console window

☐ From the design files you downloaded, open the .tcl file that corresponds to your development kit; `DE10_Lite_pins.tcl` or `DE0_CV_pins.tcl`. If you are using a different board, you can write your own .tcl file by examining the template and using the pin guides from the user manual.

☐ Copy the commands from the file and enter them in the TCL Console window. Note that if you reassign pins that have already been assigned in the pin planner, this will not cause a problem, as long as the pin locations are correct. Make sure that carriage returns are properly copied over from these TCL commands into the TCL utility window. If all 20 TCL commands show up as a single continuous line, you will need to copy and paste each command one at a time. The beginning of each line starts with `set_location_assignment` or `set_instance_assignment`.

☐ Now the remaining pins have been assigned for you by the script. To check what has been assigned you can return to the Pin Planner application or alternatively open the Assignment Editor Window (**Assignments** → **Assignment Editor**) to check that the TLC commands have properly set the pinout for your Switch to LED design. Try both methods to familiarize yourself with different techniques to manage and observe pin constraints.

3.3: Compiling Your Code

☐ Click , located at the top of the main Quartus window, to start the full compilation of your code. You can also go to: **Processing** → **Start Compilation**.

☐ After roughly 1 to 2 minutes depending on your machine type and amount of RAM, the compilation should complete and there should be 0 errors. (You can ignore warnings.)

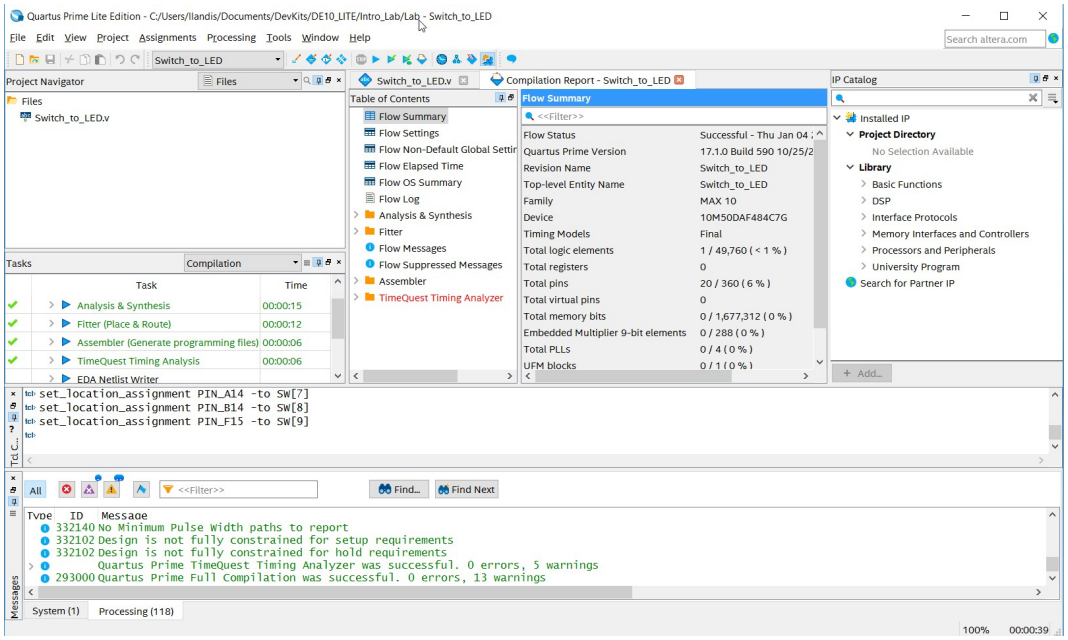

Figure 14: Quartus window showing successful compilation

3.4: Installing and Using the USB Blaster to Download Your Design to the FPGA

☐ To download your completed FPGA design into the device, connect the USB Blaster cable between your PC USB port and the USB Blaster port on your development kit. If you are not using the DE10-Lite, you may have to plug the kit into power using a wall adapter. Upon plugging in your device, you should see flashing LEDs and 7-segment displays counting in hexadecimal, or other lit up LEDs and 7-segments depending on previous projects that have been downloaded to the development kit.

☐ To use the USB Blaster to program your device, you need to install the USB Blaster driver. To begin, open your Device Manager by hitting the Windows Key and typing **Device Manager**. Click the appropriate tile labeled Device Manager that appears.

☐ Navigate to the **Other Devices** section of the Device Manager and expand the section below.

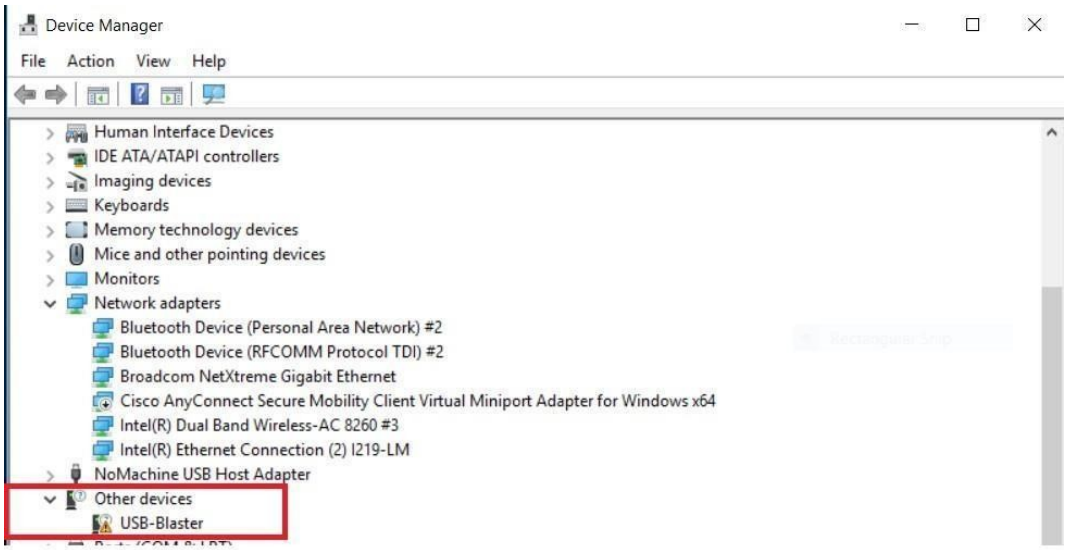

Figure 15: Device Manager with uninstalled USB Blaster driver

☐ Right click the USB Blaster device and select **Update Driver Software**.

☐ Choose to browse your computer for driver software and navigate to the path shown below in Figure 16.

Figure 16: 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.

3.5: Programming Your Design into the FPGA

The next step will take the FPGA "image" and download it to your DE10-LITE or DE0-CV development kit.

☐ Return to Quartus Prime and click on to open the Programmer. This button is located at the top of the main Quartus window. A screen similar to the one seen in Figure 17 below should appear.

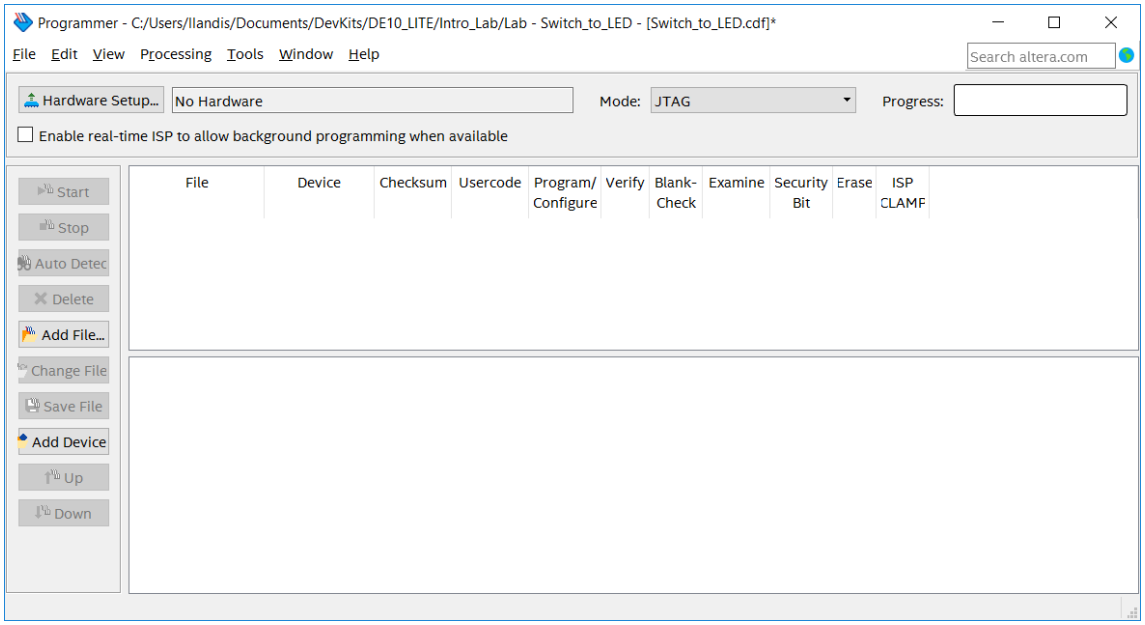

Figure 17: The Programmer window

☐ Click **Hardware Setup** in the upper left corner and choose USB Blaster (if the driver is properly installed) and close.

☐ Next, click on **Add File** on the left panel.

☐ Navigate to the **Output Files** folder, choose **switch_to_led.sof** and then **Open**. This file is called a SRAM Object File and is the compiled image that programs your FPGA. Check the small **Program Configure** box in the middle of your screen.

☐ Click **Start**. Observe the progress meter complete to "100% (Successful)." Figure 18 below shows the programmer window after a successful .sof programming file has been uploaded to the FPGA. On occasion, you need to click the Start button twice to get this step to work properly.

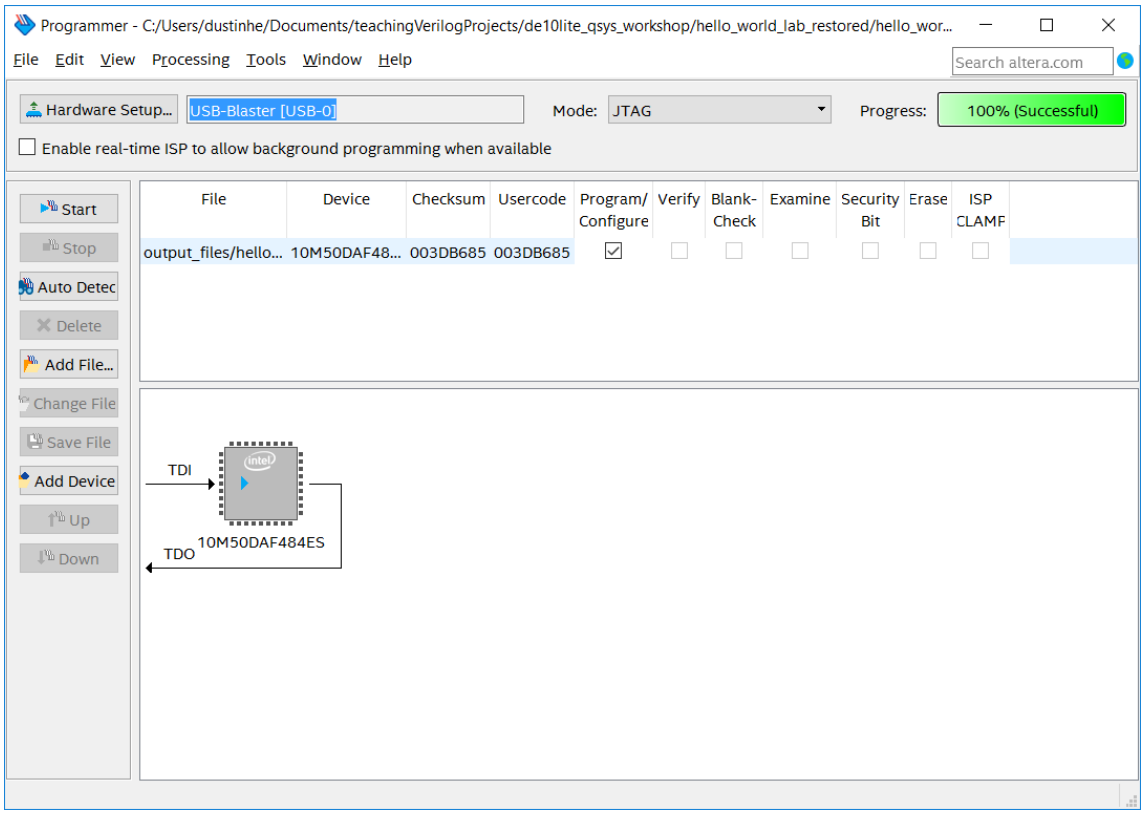

Figure 18: Successful programming of the .sof file to the FPGA

3.6 Testing Your Design

☐ Check the functionality of the circuit by toggling the sliding switches (not the push buttons) and see the LEDs turn on and off.

Congratulations!

You have just completed the switch_to_led lab using the DE10-Lite or DE0-CV Development Board.

☐ Close the programmer. If Quartus displays a dialogue box that asks you to save changes to your chain file, press the **No** selection.

LAB 4: 2 TO 1 MULTIPLEXER

Summary

Follow the steps from last lab and implement a 3 bit 2-to-1 multiplexer. A 2-1 multiplexer selects one of 2 data inputs. If the "S" pin is logic 0, M gets the value X, else (if S is logic 1) M gets the value Y. Note this lab uses arrays. To define an array, refer back to the switch_to_led.v code in Section 3.1 where we used syntax such as input [9:0] SW to define the input signal.

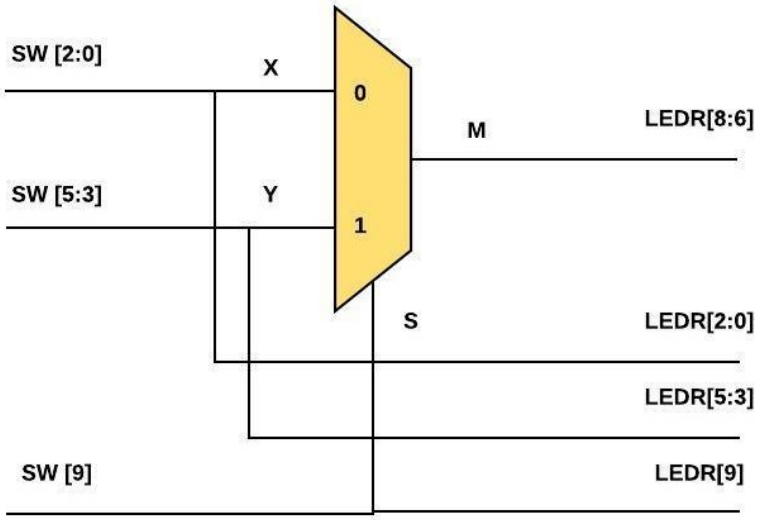

Figure 19: Logic Flow for a 2 to 1 Mux

Lab Instruction

4.0: Creating a Revision

In this lab, we are going to use a handy feature in Quartus called revisions. Using revisions will save you time since you can reuse the pin settings you made in the Pin Planner tool and carry them over to other projects.

☐ Launch the Revision tool by navigating to **Project** → **Revisions**.

☐ Add a new revision by doubling clicking on the **New Revision** selection and make the revision name **mux_2_to_1**. *Ensure that the revision name is entered with case-sensitivity to avoid compilation errors.* On the following page, continue by clicking **OK**.

4.1: 2-1 Mux Verilog Code

There are several approaches to this lab. If you are brand new to coding in Verilog you may copy and paste the code from Section 4.5 (not the code snippets below). Should you choose this option, once you copy the code and save the Verilog file to the name **mux_2_to_1.v**, you

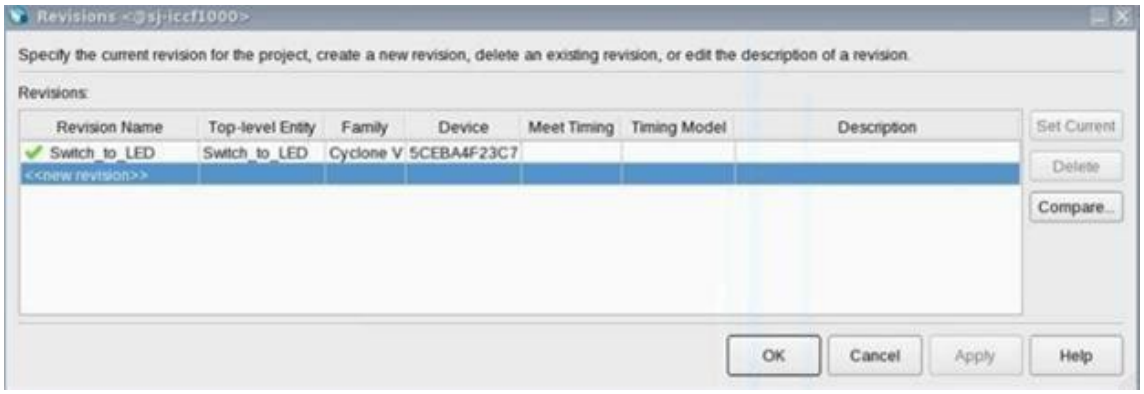

Figure 20: Quartus revisions window

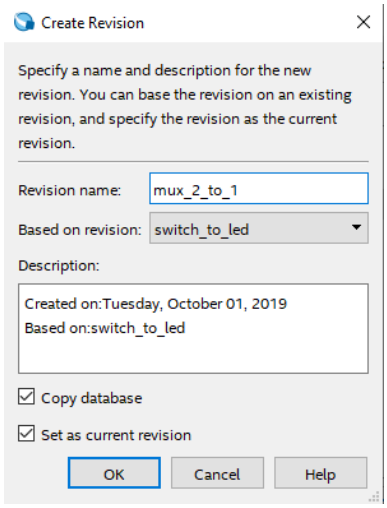

Figure 21: Final Revision Creation window

may skip to the next section of this lab manual.

The other option is to create a Verilog file from scratch for the 3-bit wide 2-to-1 multiplexer in your project. Take a look at the code from Section 3.1 on how to declare the ports on your module. This means to include the module statement and inputs/output definitions.

CHECKLIST:

- ☐ Use switch **SW[9]** as the **S** input (the selection bit of the multiplexer), switches **SW[2:0]** as the **X** input and switches **SW[5:3]** as the **Y** input.
- ☐ With assign statements, display the value of the input **S** on **LEDR[9]**, input **X** on **LEDR[2:0]**, input **Y** on **LEDR[5:3]**.
- ☐ Assign **M** to **LEDR[8:6]**.

There are several ways to define a multiplexer in Verilog. Pick one of the three styles shown below. If you have time, try a couple of different coding styles for practice. Place these lines after the module definition and before the end `module` statement. You can also get

the completed code from the design files mux_snippet_continuous.v, mux_snippet_if.v, and mux_snippet_case.v.

CONTINUOUS ASSIGNMENT:

```
assign M = (S==1) ? Y : X; // If S then M = Y else M = X
         // All signals are of type wire.
```

PROCEDURAL ASSIGNMENT "IF" STATEMENT:

```
always @ (S or X or Y) begin  // If any of the signals S, X or Y
          // change state, execute this code.
          // Note that signals to the left of an
          // equal sign in an always block
          // need to be declared of type reg so
          // declare M as:
          // reg [2:0] M;
if(S == 1)
  M <= Y;     // Note the non-blocking operator ‘’<=

else
  M <= X;
end
```

PROCEDURAL ASSIGNMENT "CASE" STATEMENT:

```
always @ (S or X or Y) begin case (S)
’1b0: M <= X;
’1b1: M <= Y;
endcase end
```

Also note that variables that are assigned to the left of an equal sign (= or <=) in an always block must be defined as `reg`. Other variables are defined as `wire`. If undeclared, variables default to a 1 bit wire.

With the above port and signal assignments, we will see the output X when the select input S is low and we will see Y when S is high.

4.2: Revision Control

Now you need to make sure you have the proper files included in your project.

☐ To the right of the Project Navigator Window, change **Hierarchy** to **Files**. You will only be operating on the mux_2_to_1.v so you will need to remove switch_to_led.v from your project.

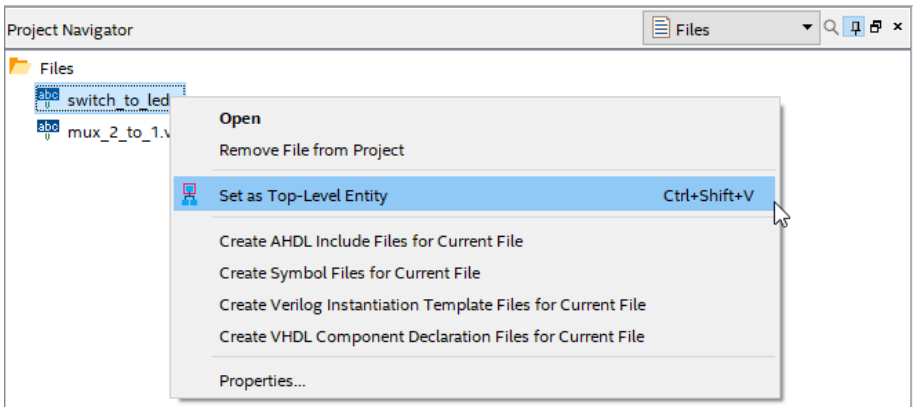

Figure 22: Changing Top Level Entity

☐ Right click **switch_to_led.v** and remove this source file from your project.

☐ Next you need to change your top-level entity from switch_to_led to mux_2_to_1. Right click mux_2_to_1.v and **Set as Top-Level Entity**. Now your revision and your top level entity is mux_2_to_1.

It is important to note the difference between a project, revision, top level entity, and top level Verilog file of your overall project. A single project can have multiple revisions. The project name does not have to match the name of the top level entity of your design. Similarly, the Verilog file name might not have the same name as the top level entity and can indeed contain many modules (entities).

☐ A common compilation error is a mismatch between what the top level module in your code is versus the one assigned in the Quartus settings. If you have a compilation error of this nature, check: **Assignments** → **Setting** → **General** and make sure the top level entity (in this case mux_2_to_1) is indeed set to the one you think you are compiling your Verilog source code. Ensure that top level entity names match with their corresponding Verilog file names as they are case-senstive.

☐ Compile your design by pressing along the top of the Quartus window.

4.3: Checking Pin Locations and Editing (If Necessary)

In your project, the required pin assignments for your DE10-LITE or DE0-CV Development board should have carried over from the previous lab since the pin names are the same.

☐ Open up the **Assignment Editor** to make sure the pin names are indeed assigned to the appropriate pin locations. **Assignments** → **Pin Planner**. (Note: pin assignments can also be seen through **Assignments** → **Assignment Editor**.)

4.4: Downloading Your Design to Your Device

☐ Once you have successfully compiled the project, download the resulting .sof file onto the FPGA chip as you did in section 3.5.

☐ Test the functionality of the 3-bit wide 2-to-1 multiplexer by toggling the switches and observing the LEDs. Remember that we want the lights LEDR[2:0] to display input X, LEDR[5:3] to display Y, LEDR [8:6] to display the multiplexed result, LEDR[9] to display the switch SW[9] result (selection bit).

4.5: Viewing Design Schematic

When you compile Verilog, or any HDL (Hardware Description Language), Quartus synthesizes your code into hardware. This hardware can be viewed through the RTL Viewer (Register Transfer Level Viewer).

☐ The RTL Viewer can be found under task on the left-hand side by looking into the **Analysis & Synthesis** dropdown and then the **Netlist Viewers** dropdown, as shown in Figure 22. Once opened, you should see a 2-to-1 mux similar to what is seen in Figure 23.

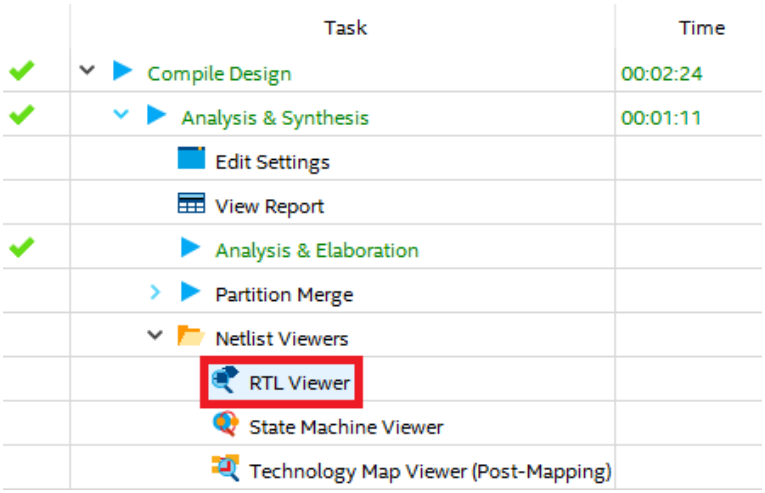

Figure 23: RTL Viewer selection

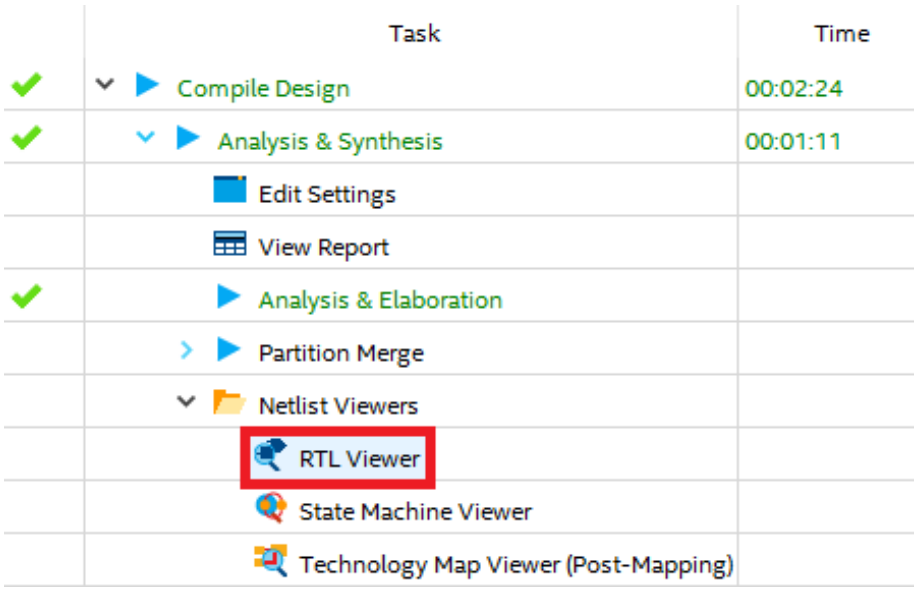

Figure 24: RTL Viewer of a 2 to 1 Mux

4.6: Working 2 to 1 MUX Verilog Code

In case you had problems getting things working, there is one complete implementation of the mux in the design file mux_2_to_1.v

```
module mux_2_to_1 (SW, LEDR);    //Create module mux_2_to_1

input [9:0]SW;  //Input Declarations: 10 slide switches
output[9:0]LEDR;  //Output Declarations: 10 red LED lights

wire S; //Declare the Select signal
wire [2:0] X, Y, M; //Declare the inputs and outputs to the MUX

assign S = SW[9]; //Assigning input switches to internal signals
assign X = SW[2:0];
assign Y = SW[5:3];

assign LEDR[8:6] = M; //Assigning internal signals to output LEDs
assign LEDR[9] = SW[9];
assign LEDR[2:0] = SW[2:0];
assign LEDR[5:3] = SW[5:3];

assign M = (S == 0) ? X : Y;  //MuxSelect Function
endmodule
```

LAB 5: KNIGHT RIDER

Summary

Perhaps some of you have heard of or watched a TV show called Knight Rider that aired from 1982 to 1986 and starred David Hasselhoff. The premise of the show was David Hasselhoff was a high-tech crime fighter (at least high technology for 1982) and drove around an intelligent car named "KITT". The KITT car was a 1982 Pontiac Trans-Am sports car with all sorts of cool gadgets. The interesting gadget of interest for this lab were the headlights of KITT which consisted of a horizontal bar of lights that sequenced one at a time from left to right and back again at the rate of about 1/10th of a second per light. Check out this short YouTube[1] video for crime fighting and automotive lighting technology's finest moment. This lab will teach you a thing or two about sequential logic and flip-flops. Let's quickly review how flip-flops work.

Flip-flops are basic storage elements in digital electronics. In their simplest form, they have 3 pins: D, Q, and Clock. The diagram of voltage versus time (often referred to as a waveform) for a flip-flop is shown below. Flip-flops capture the value of the "D" pin when the clock pin (the one with the triangle at its input transitions from low to high). This value of D then shows up at the Q output of the flip-flop a very short time later.

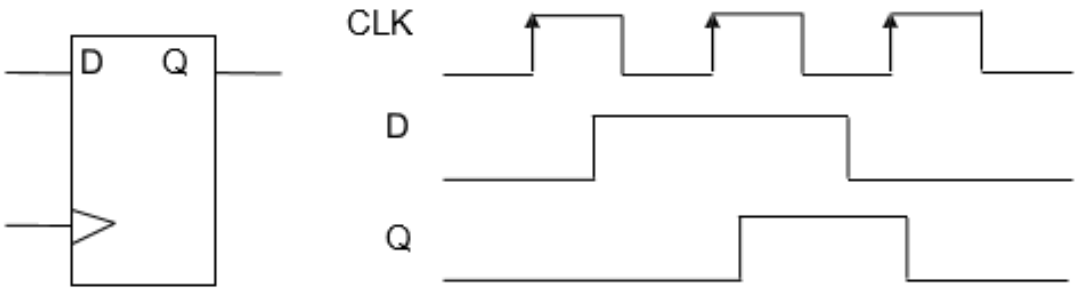

Figure 25: Flip-flop diagram

When you connect several flip-flops together serially you get what is known as a shift register. That circuit serves as the basis for the Knight Rider LED circuit that we will study in this lab. Note how we clock in a 1 for a single cycle and it "shifts" through the circuit. If that "1" is driving an LED each successive LED will light up for 1/10 of a second.

Lab Instruction

5.0: Knight Rider Verilog Code

- ☐ The following Verilog code is the starting point for your Knight Rider design, but there are some bugs. Start a new revision of the project "lab" as you did in section 4.0 and call it `knight_rider` with similar settings as the previous labs.

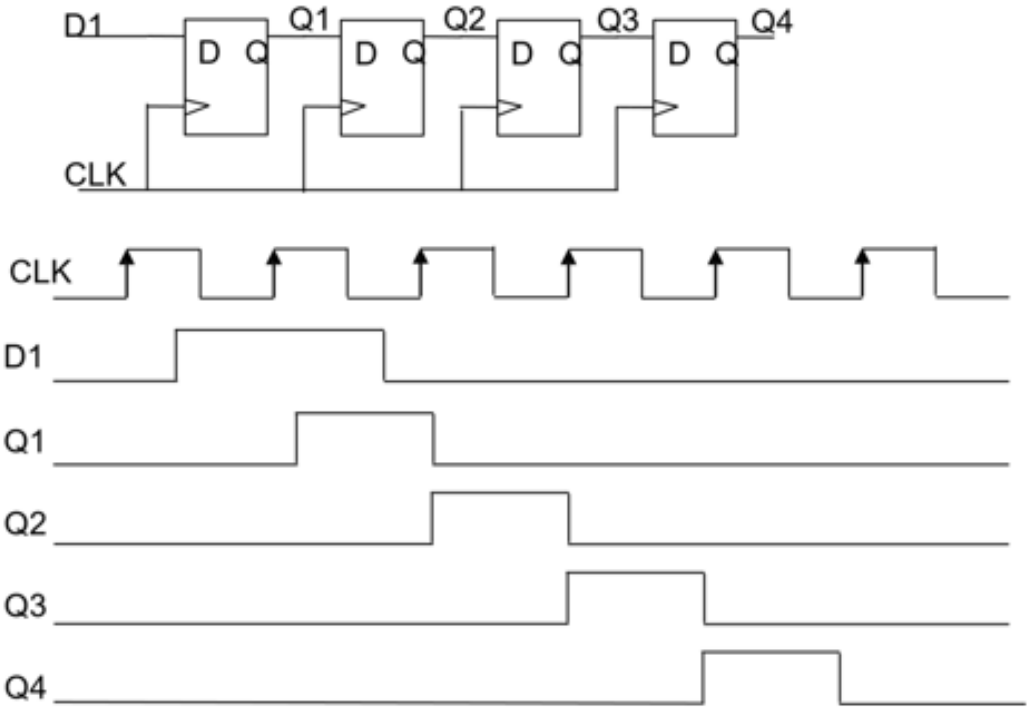

Figure 26: Shift Register diagram

☐ The code given **intentionally** has errors. See if you can find them all. You can also find this in design file knight_rider.v

```
module knight_rider(
  input wire CLOCK_50,
  output wire [9:0] LEDR
);
  wire slow_clock;

  reg [3:0] count;
  reg count_up;

  clock_divider u0 (.fast_clock(CLOCK_50),.slow_clock(slow_clock));

  always @ (posedge slow_clock)
  begin
    if (count_up)
      count <= count + 1'b1
    else
      count <= count - 1'b1;
  end

  always @ (posedge clk)
  begin
    if (count == 9)
```

```
        count_up <= 1'b0;
      else if (count == 0)
        count_up <= 1'b1;
      else
        count_up <= count_up;
    end

    assign LEDR[9:0] = (1'b1 << count);

  endmodule


  module clock_divider(
    input fast_clock,
    output slow_clock
  );

    parameter COUNTER_SIZE = 5;
    parameter COUNTER_MAX_COUNT = (2 ** COUNTER_SIZE) - 1;

    reg [COUNTER_SIZE-1:0] count;

    always @(posedge fast_clock)
    begin
      if(count==COUNTER_MAX_COUNT)
        count <= 0;
      else
        count<=count + 1'b1;
    end

    assign slow_clock = count[COUNTER_SIZE-1];

  endmodule
```

5.1: Creating “knight_rider.v”

- ☐ Open a new Verilog file and save it as **`knight_rider.v`**. Copy and paste the code above into `knight_rider.v`.
- ☐ Make sure your top level entity is called `knight_rider` and the source file is `knight_rider.v`. Delete any other Verilog files in the current project.

  In the upper left Project Navigator window, you should see something similar to this:

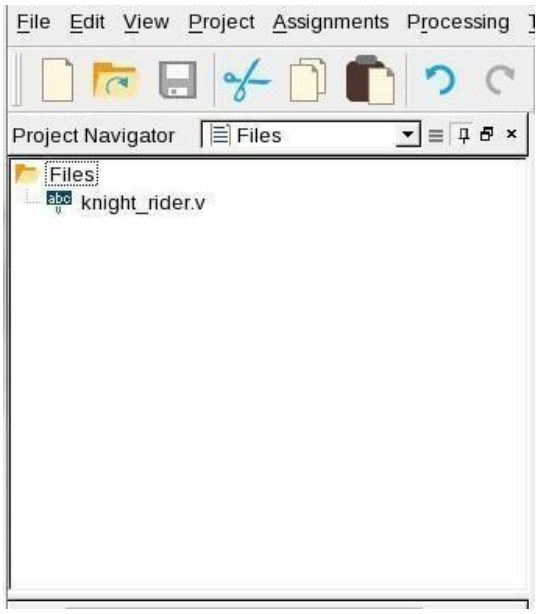

Figure 27: Example of `knight_rider.v` in Quartus Window

5.2: Debugging Code

☐ Click on the **Play** button and run **Analysis & Elaboration**. This source code has several syntax bugs. Look at the transcript window on the bottom and observe the errors that are flagged with symbol.

☐ Carefully look at the source code and fix the errors and continue to recompile until the compilation steps run to completion.

5.3: Assigning Pins with the Pin Planner Tool

☐ Next you need to make sure the pins are in the right place. Open up the **Pin Planner** as done in previous section. You can also click this icon along the top of the Quartus window.

☐ Check the pins and make sure the LEDs are assigned to the same locations as the first lab. Note that there is an additional pin name called `CLOCK_50` in this design that needs to be assigned. The clock should be connected to `PIN_P11` (50 MHz).

Note: the pin planner will use default locations that don't match your development kit unless specifically instructed.

☐ Hit compile after changing the pinout.

5.4: Downloading Your Code to Your Device

By now you should have the hang of how to program the FPGA image into the DE10-LITE Board. Go ahead and try it out. Do you see the infamous Knight Rider pattern? When working

properly, you should see something like the following:

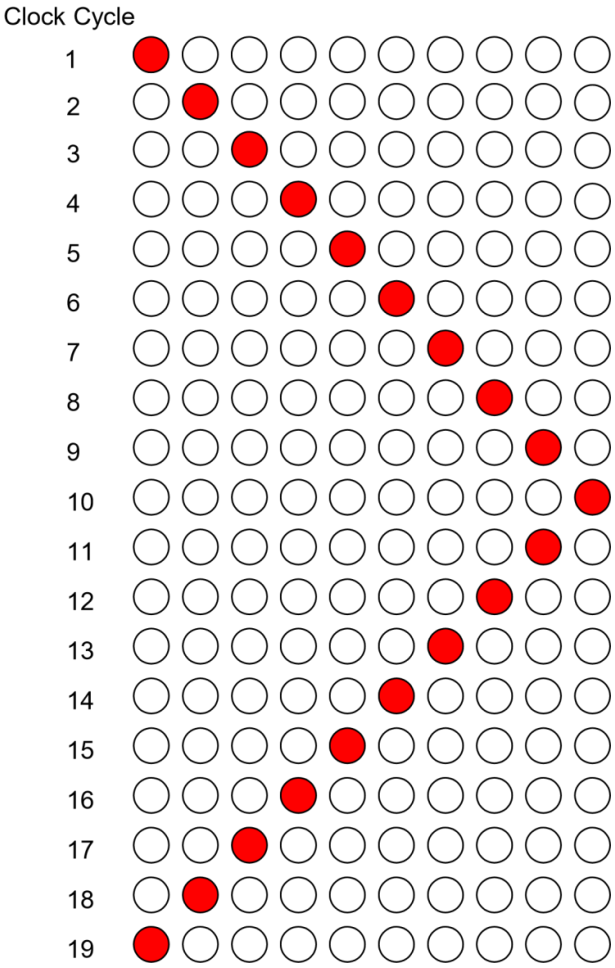

Figure 28: Example of `Knight Rider sequence`

What do you see? **(If it does not work look at the next section.)**

5.5: More Debugging

You knew we weren't going to make it that easy, did you? How come the lights don't sequence? Here is a portion of the explanation. The selected clock frequency of the DE10-LITE Board is 50 MHz. That means the clock changes 50 million times per second. If you change the LEDs at that rate, you cannot view them with the naked eye.

When you go through the code you will see a module in your code called **`clock_divider`**. You want the output clock to toggle at around 10 Hz (10x per second). This `clock_divider` module takes the 50 MHz clock and divides down the clock to a slower frequency. Your lab instructor goofed and did not calculate the right divide ratio to slow the 50 MHz clock down to 10 Hz.

You need to do a bit of math (including the log function!) to determine how to derive the proper size of the counter to divide 50 MHz to roughly 10 Hz.

- ☐ Basically, think about a divide ratio that is 2^N where N is the width of the counter. Adjust the parameter to **`COUNTER_SIZE`** to the appropriate ratio and recompile and reprogram the FPGA.
- ☐ Work out N based on the following equation: $10 = 50,000,000/2^N$. Round N up to the nearest integer to discover the proper `WIDTH` parameter setting.
- ☐ Recompile and program the DE10-LITE development board.

5.6: Even More Debugging!

Is the Knight Rider sequence working properly? Does each LED stay on for about 1/10 second? If not, redo your math to find the right `WIDTH` parameter. Look at the sequencing carefully. Does each LED illuminate once and proceed to its neighboring LED? As you will observe LED[0] and LED[9] will blink twice.

- ☐ Dang! That lab instructor created another error in the design! Look at the source code in the knight_rider.v code and see if you can find the error.
- ☐ Change the code, recompile and reprogram your DE10-LITE or DE0-CV development kit until your Knight Rider LEDs are sequencing properly.
- ☐ Try viewing your knight rider hardware through the RTL Viewer (Register Transfer Level Viewer) like you did for the mux 2 to 1 design!

Thanks for taking time learning how to develop Intel FPGA products. We hope you found this lab informative. Long live David Hasselhoff[2]!

LINKS:

1. https://youtu.be/oNyXYPhnUIs?t=8s
2. https://youtu.be/PJQVlVHsFF8

LAB 6: TEST YOUR KNOWLEDGE

Feel free to attempt this lab if you have extra time! Allot 1-2 hours to work through it.

This is a short lab to test your knowledge in assembling a small FPGA project on the DE10-Lite development kit. The circuit adds the values of two counters (0-98) that are enabled with push button switches. The two addends and the sum are displayed on the seven segment displays. Since the sum can overflow to a 3rd hundreds place digit and there are only 6 seven segment displays, the hundreds position should be represented by lighting up all 10 LEDs simultaneously.

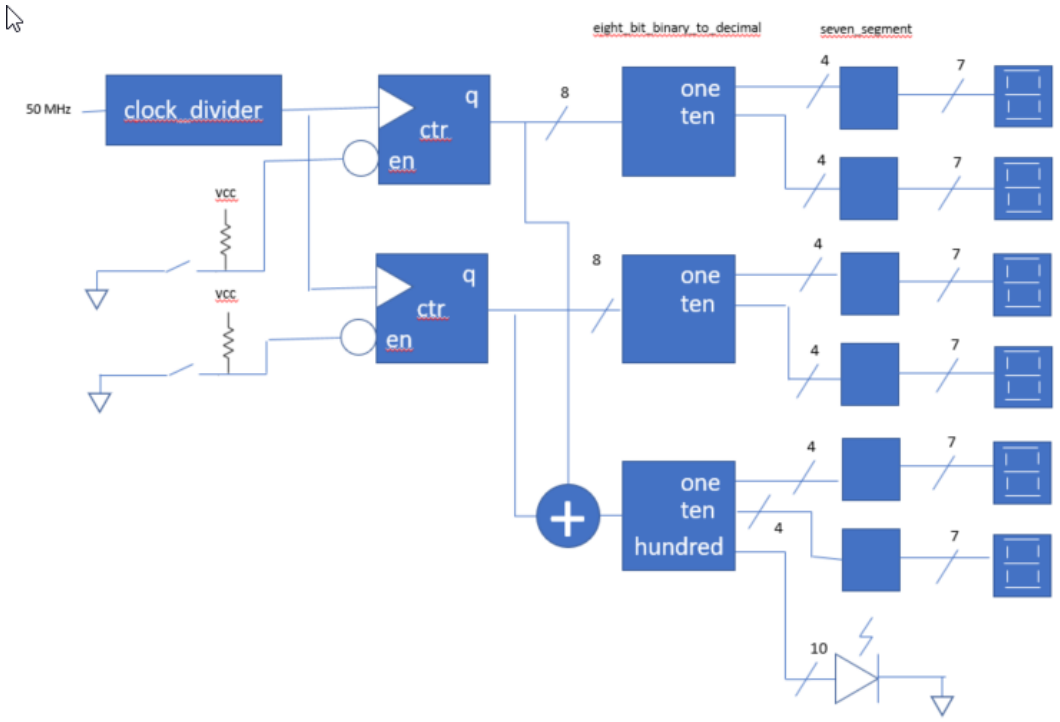

Figure 29: Example of `Block Diagram`

Modules Provided

- `ctr 0-98`: change to 0-99 (built from IP library)
- `clock_divider`: 50 MHz to 2 Hz (Verilog)
- `eight_bit_binary_to_decimal`: converts binary to decimal (Verilog)
- `seven_segment`: converts decimal to seven segment display (Verilog)

Starting point .qar file is in the design files titled Addit.qar

- ☐ Create top module based on provided wrapper. This requires editing the Verilog file and wiring up the blocks.

- ☐ Compile and program on board.
- ☐ Change `ctr` to count to 99.
- ☐ Now make the hundreds position light up 10 LED(s). Add LEDs to top level and check pin assignments.
- ☐ **Bonus Section (Go for it!)**: Make `ctr` count up or down, by controlling its direction with Switch 0. Note that the counter is a parameterized IP. To edit the counter: **Project Navigator** → **Change to IP Components**. Right click on LPM_COUNTER and change to include up-down input port. You will need to add the up-down port to the top level and connect to appropriate SW[0] signal which is preassigned in the assignment editor.

Solution is provided only upon request.

APPENDIX

List of Figures

1 Quartus Prime Lite Minimum Required Files to Download 5
2 Quartus Prime main window 6
3 Quartus Prime file menu 6
4 Pane 2 of the New Project Wizard 7
5 Correct settings for the DE10-Lite Development Board 8
6 Summary page generated by the New Project Wizard 9
7 Quartus Prime project window 10
8 Utility Windows dropdown 11
9 New File Window 12
10 Analysis and Elaboration check mark 13
11 Quartus Assignments menu 14
12 Pin Planner Assignment window 14
13 TLC Console window 17
14 Quartus window showing successful compilation 18
15 Device Manager with uninstalled USB Blaster driver 19
16 Directory containing USB Blaster drivers 19
17 The Programmer window 20
18 Successful programming of the .sof file to the FPGA 21
19 Logic Flow for a 2 to 1 Mux 22
20 Quartus revisions window 23
21 Final Revision Creation window 23
22 Changing Top Level Entity 25
23 RTL Viewer selection 26
24 RTL Viewer of a 2 to 1 Mux 27
25 Flip-flop diagram 28
26 Shift Register diagram 29
27 Example of `knight_rider.v` in Quartus Window 31
28 Example of `Knight Rider sequence` 32
29 Example of `Block Diagram` 34

List of Tables

1 DE10-Lite Development Board 15
2 DE0-CV Development Board 16
3 Revision Control History 37

Revision History

DATE	NAME	DESCRIPTION
03/21/2016	L. Landis	Initial Release
04/13/2016	L. Landis	Remove Modelsim from download package
05/24/2016	P. Mayer	Fixed broken links, updated for Quartus 16.0, added a few extra assignments
06/03/2016	L. Landis	Added solution to 2:1 Mux lab
06/07/2016	L. Landis	Added revision for copying pin assignments
07/20/2016	L. Landis	Clarify Mux_2_to_1 copy and paste code
10/03/2016	L. Landis	Clarify no driver image; typos
03/16/2017	A. Weinstein	Added USB Blaster driver installation instruction. Added table of figures and figure numbers. Made instructions clearer w.r.t. revision control and when writing Verilog code for labs. Added a solution for the 3-1 MUX lab.
10/10/2017	D. Henderson	General document formatting and clean up. Additionally, updated wiki links and some screen shots. Last, added TCL script instructions for assigning pins.
11/15/2017	D. Henderson	Cleaned up naming from previous port of the documentation
12/01/2017	S. Girisankar	Updated from 4-bit to 3-bit 2 to 1 Mux
01/04/2018	L. Landis	Changed TCL from file download to tcl console
04/10/2018	A. Joshipura	Changed from .pdf to .word format.
07/10/2018	S. Soto	General document formatting and clean up. Updated cross references. Created download links for all tcl scripts and Verilog code. Updated screenshots. Added Test Your Knowledge Lab. RTL Viewer of Mux 2-to-1 added.
07/24/2018	H. Martinez	Updated format to fit Intel Branding Guidelines. Changed formatting from Microsoft Word to LaTeX
08/08/2019	R. Nevin	Removed all references to Altera links and correct some trademark names
10/01/2019	S. Cabanday	Updated Verilog file names from upper-case to lower-case and screenshots to include new, all lower-cased file names

Table 3: Revision Control History