



# Introduction to Intel®FPGAS and the Quartus® Prime Software Using Remote Hands-Free Console and the SJ Campus 3101 Lab

---

© 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

HINTS AND TRICKS.....	4
Background .....	4
LAB 1: ACCESSING YOUR LAB PC .....	7
LAB 2: UNARCHIVING A Blank project.....	9
Summary .....	9
Lab Instruction.....	9
2.0: Blank Project GitHub Download .....	9
2.1    Blank Project Environment Setup.....	10
2.2    Understanding your Project Setup .....	11
LAB 3: DESCRIBING YOUR FPGA FUNCTION USING VERILOG .....	15
Summary .....	15
Lab Instruction.....	15
3.1    Creating a New File.....	15
3.2    Adding Verilog Code .....	16
3.3    Understanding the design hierarchy.....	17
3.4    Instantiating User Design in Top .....	19
3.5    Setting up your pin assignments .....	20
3.6    Compiling Your Code .....	21
Setting up the remote board programming environment .....	22
3.7.....	22
3.8    Programming Remote Board .....	25
3.9    Testing Your Design.....	26
LAB 4: 2 to 1 multiplexer .....	27
Summary .....	27
Lab Instruction.....	27
4.1    2-1 Mux Verilog Code.....	27
4.2    Working 2 to 1 MUX Verilog Code .....	29

4.3	Instantiating mux_2_to_1 into top module .....	30
4.4	Viewing the Design Schematic.....	31
LAB 5: Knight Rider .....		33
	Lab Instruction.....	34
5.1	Knight Rider Verilog Code .....	34
5.2	Creating “knight_rider.v” .....	36
5.3	Debugging Code .....	36
5.4	Running knight_rider on your remote board.....	36
5.5	More Debugging.....	37
5.5:	Even More Debugging! .....	38
6	Document Revision History .....	39

# HINTS AND TRICKS

---

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

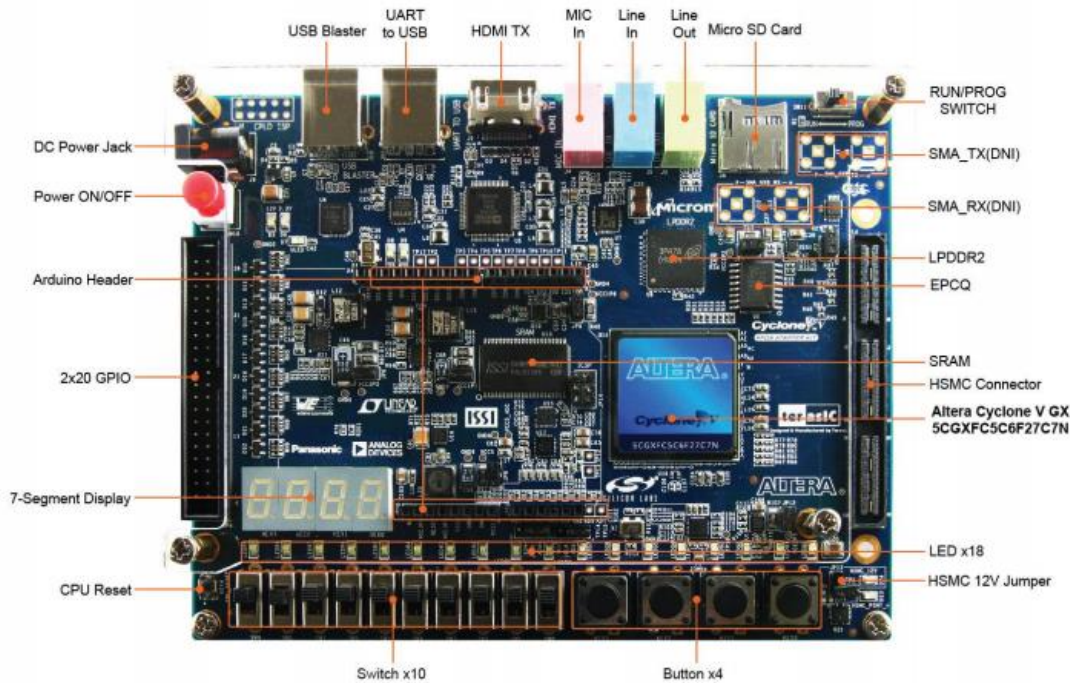
- You should use the 18.0 Standard version of the Intel Quartus Prime software. This version requires no license and is supported in the computer lab you are accessing. The Intel® Quartus® Prime Pro Edition software will not work as it does not support the target hardware.
- If something fails to compile, 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.
- 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!

## 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.

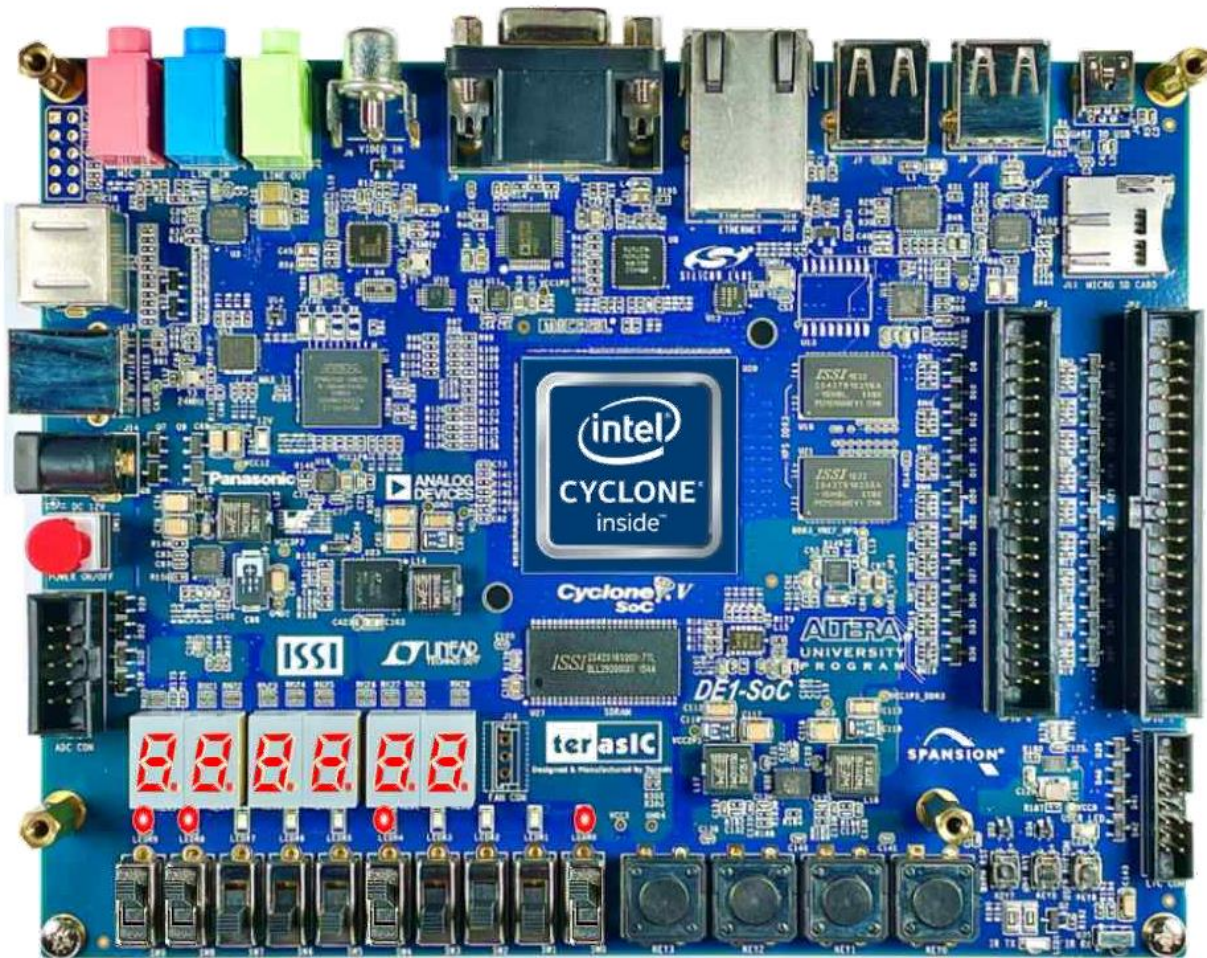
To configure an FPGA, first you describe your digital electronics with either a Hardware Description Language (HDL), such as Verilog or VHDL, or a schematic. Then you assign the “pins” of your FPGA based on how the Printed Circuit Board (PCB) connects the FPGA to various peripheral components on your 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 FPGA development kit you will use is either a [Cyclone V GX Starter Kit](#), or a development kit called the [DE1-SoC](#). Some of you in the class will end up connecting to a PC that hosts a Cyclone V GX Starter Kit and others will be connected to a PC hosting the DE1-SoC kit. The development kits are attached to laptop Windows PCs in a lab at Intel's San Jose, CA campus. In either case of board use, you will view your operational lab on a DE1-SOC board image. Operating the board switches and viewing LEDs/7-segments looks like a video game version of the actual development kit (see Figure 2). You will be able to see LEDs change state and move switches to run your experiments.



*Figure 1: Cyclone V GX Starter Kit physically connected to the hands-on Lab PC*





32-bit Parameter 0

32-bit Parameter 1

```
Board launched!
Connected to USB-Blaster [USB-0]
Time remaining: 11:17
Ping: 27 ms
```

*Figure 2: DE1-SoC remote console used to change switch values and see LEDs change state*

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.

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 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

## LAB 1: 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 will be provided by your instructor.

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 you fail to follow unique instructions per board you will be unable to complete the lab successfully.



# LAB 2: Unarchiving a blank project

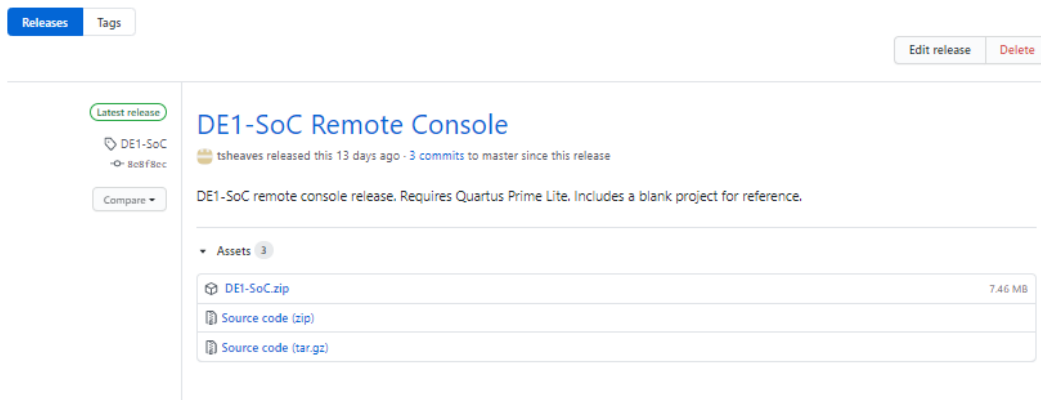
## Summary

This short lab that completes the basic project setup. You will be given a blank project that has the starting point files to make the remote console connections shown in . We will explain more about how this works in the coming sections.

## Lab Instruction

### 2.0: Blank Project GitHub Download

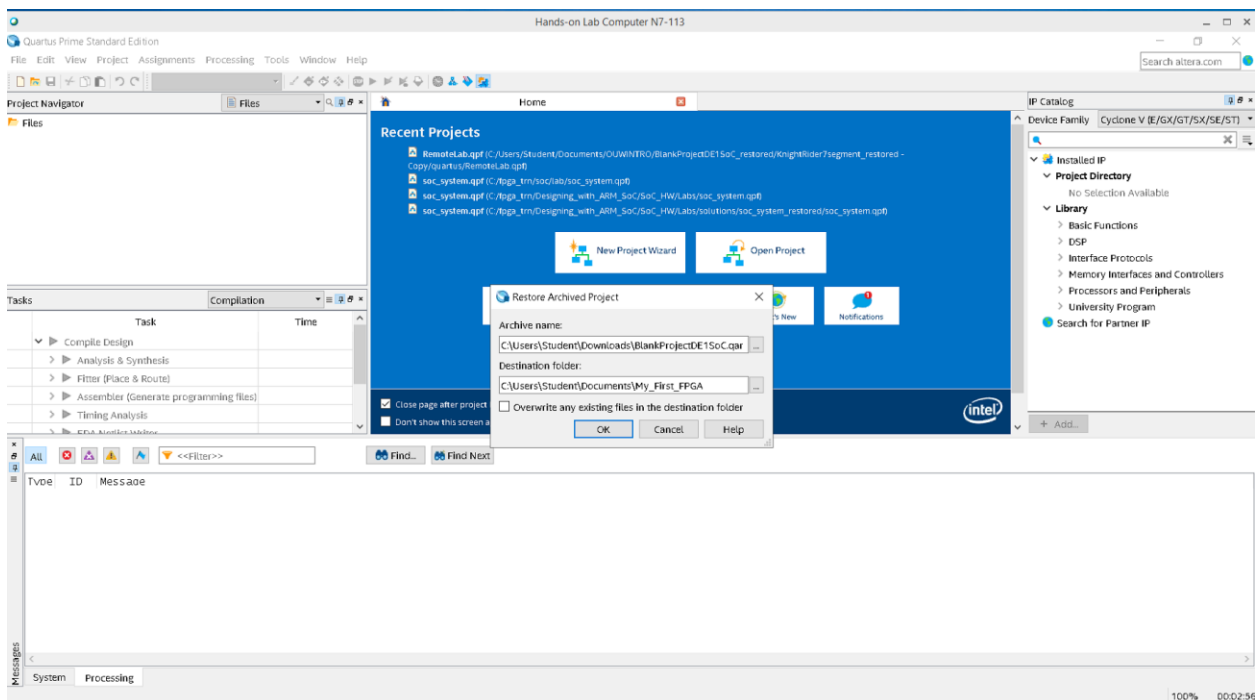
- ☐ To begin your Remote Hands-free project, go to the [GitHub site](#). Open the link in the chrome browser that is part of your webex training session, not your local PC.
- ☐ You should see a page that looks like below. Click on the zip file link, followed by download and save it. Save the DE1-SoC.zip file to a directory that does not have spaces in the path name! IMPORTANT: No spaces such as C:\Users\Student\Documents\My First FPGA . You need to make the directory C:\Users\Student\Documents\My\_First\_FPGA .
- ☐ Right click the zip file and extract the contents.
- ☐ The blankProj.qar file is called a Quartus archive file that contains project information compressed into a single file.



*Figure 3: Blank Project GitHub download*

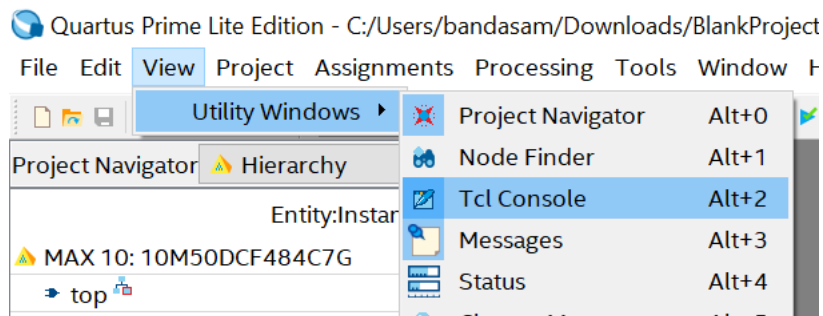
## 2.1 Blank Project Environment Setup

- ☐ Bring your Quartus window to the front of your screen.
- ☐ Unarchive the BlankProjectDE1SoC.qar file by clicking File → Open, and change the lower right filter to show Quartus Prime Archive file (\*.qar). Navigate to the .qar file you downloaded and click Open.



*Figure 4: Restore Archived Project for Blank Project*

- ☐ When you open the .qar file you will see the message from Figure 4 above. Click **OK** to restore the project.
- ☐ Some windows may not be shown by default. To customize what windows are shown, click on the **View** tab and look under the **Utility Windows** drop down as seen in Figure 5. The setup in Figure 4 is good for this exercise.



*Figure 5: Utility Window dropdown*

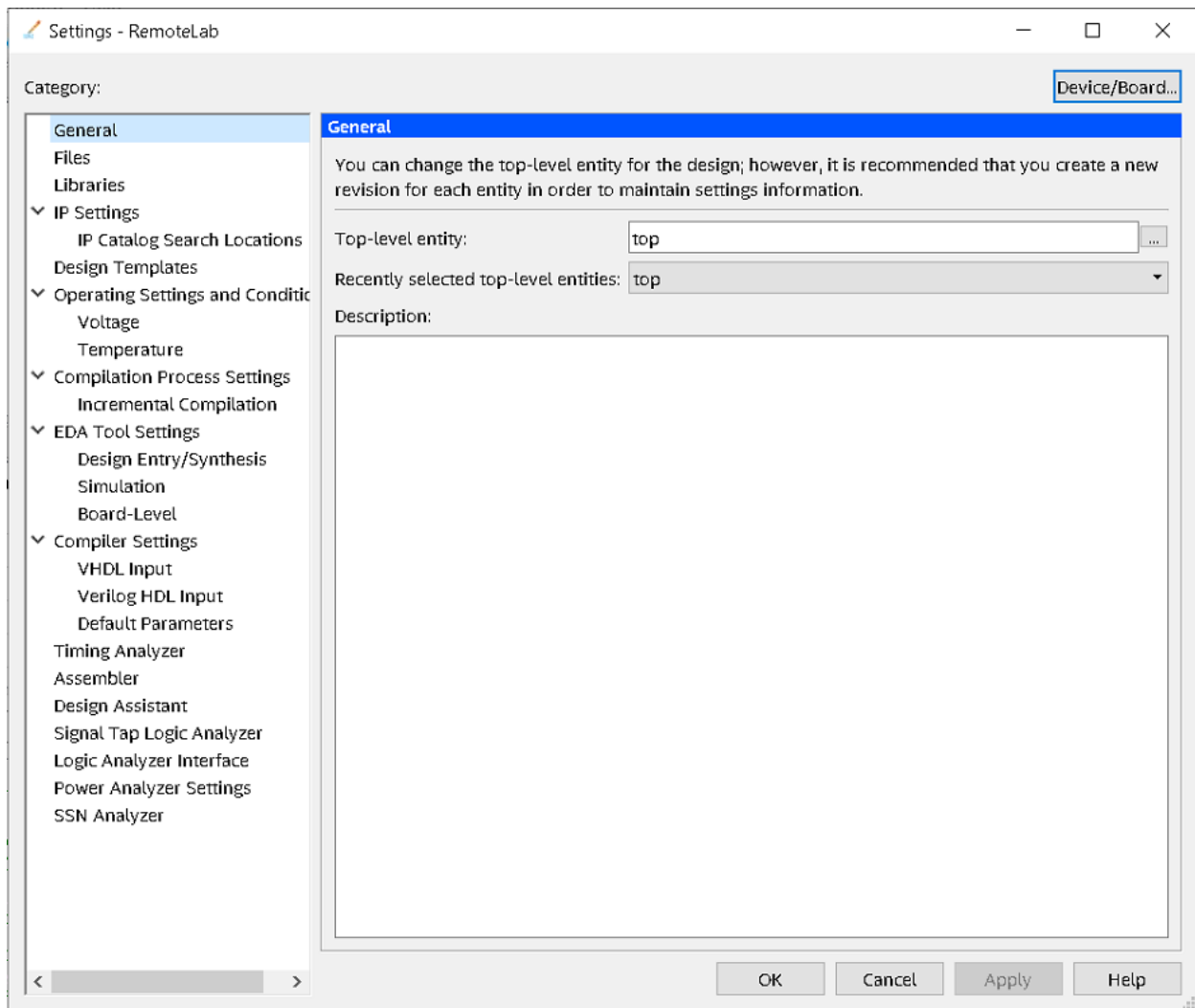
## 2.2 Understanding your Project Setup

Understanding the organization of compiler tool projects, whether it be software or hardware projects is important to know at the outset of your design work.

The remote console setup we are using in this lab unarchived a project for the DE1-SoC development board. However, some of you are likely connected to the Cyclone V GX Starter Kit board that is physically connected to the lab PC.

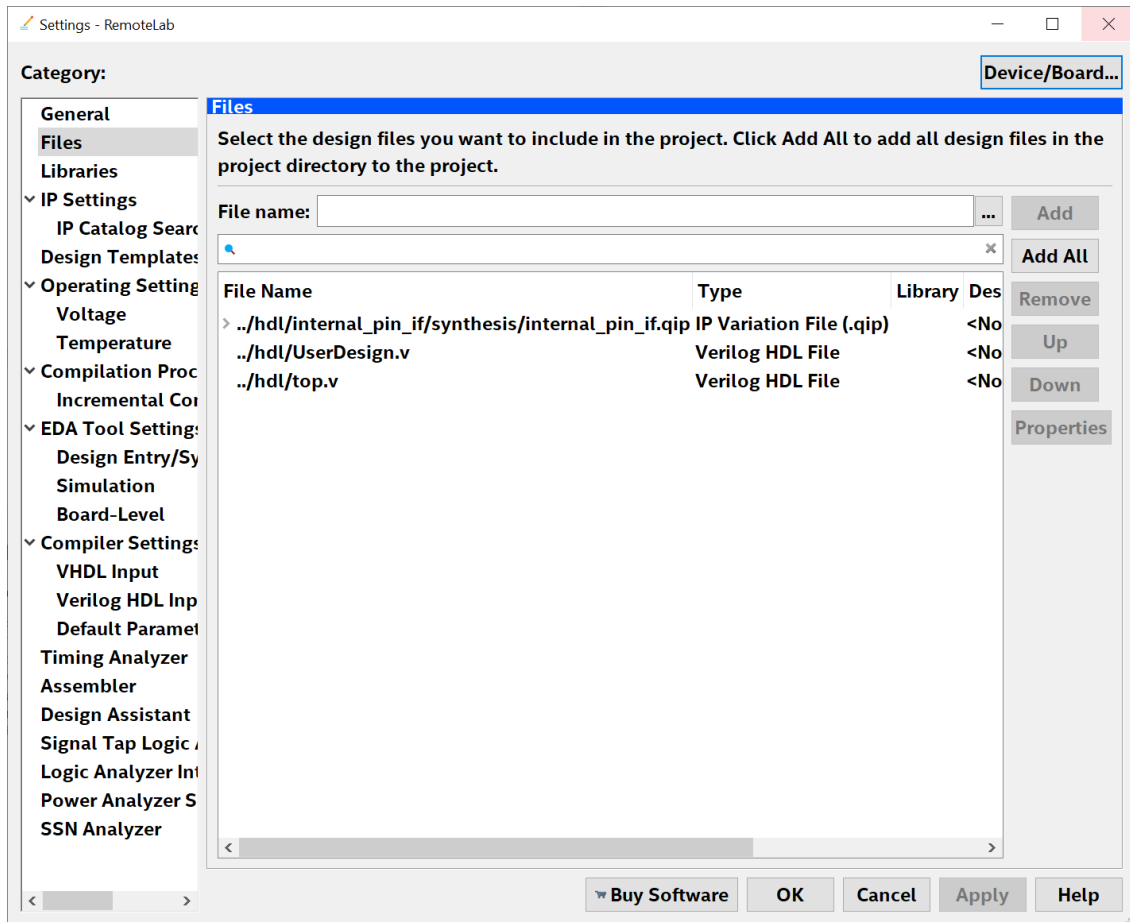
When you use the Cyclone V GX Starter kit, the DE1-SoC blank project you unarchived uses a different FPGA device part number than the Cyclone V GX starter kit physically connected to your PC. Only if you are using the Cyclone V GX Starter kit, should you make the changes below. Refer back to lab 1 to identify which development kit is connected to your PC. Start by changing to the proper part number.

- ☐ Open Assignments → Device
- ☐ In the name filter, type 5CGXFC5C6F27C7. This will give you a single match, select that one. When it asks if you want to remove current IO settings, select Yes. Click OK.
- ☐ Next, let's explore other settings. In Quartus: Assignments → Settings. In category, select General. Note that the top level entity is set to top. This is the top level of your design hierarchy and will remain the same for the different experiments we will run in this lab.



*Figure 6: Settings tab*

- ❑ Next click the Category: Files. Here you see the various design source files and other relevant project files needed to compile your design. You can add and remove files from this panel as needed.



*Figure 7: Files included in your project*

- ☐ Highlight ../hdl/UserDesign.v and click remove. We will insert a different design file into your design in subsequent steps.
- ☐ Observe that your project is called RemoteLab which is listed on the top bar of the Quartus window. Note that this name is different than your top-level entity although it could be named the same. It is possible to create multiple revision projects that will copy over assignments to create new versions of your project. This can be helpful for experimentation with settings.

# LAB 3: Describing your FPGA function in Verilog

## Summary

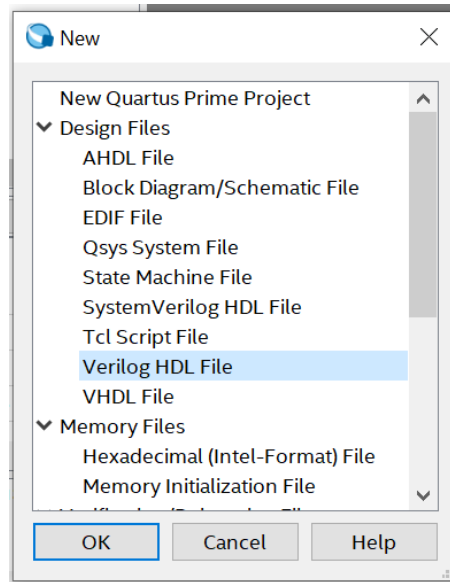
This lab will step you through the process of a simple design from generating your first Verilog file to synthesize and compile. Synthesis converts your Verilog language file to an FPGA specific “netlist” that programs the programmable FPGA lookup tables into your desired function. Compilation figures 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

### 3.1 Creating a New File

- ☐ Create a Verilog HDL file. Go the **File** dropdown menu and select **New**.
- ☐ A window, shown in Figure 7, should pop up. Click on **Verilog HDL File** and then **OK**.





*Figure 8: New File Window*

## 3.2 Adding Verilog Code

- ☐ Create a simple module in your Verilog HDL file by typing the code shown in Code Snippet 1. You can also copy/paste this code from the file [switch\\_to\\_led.v](#) found in the design files you downloaded for this workshop.

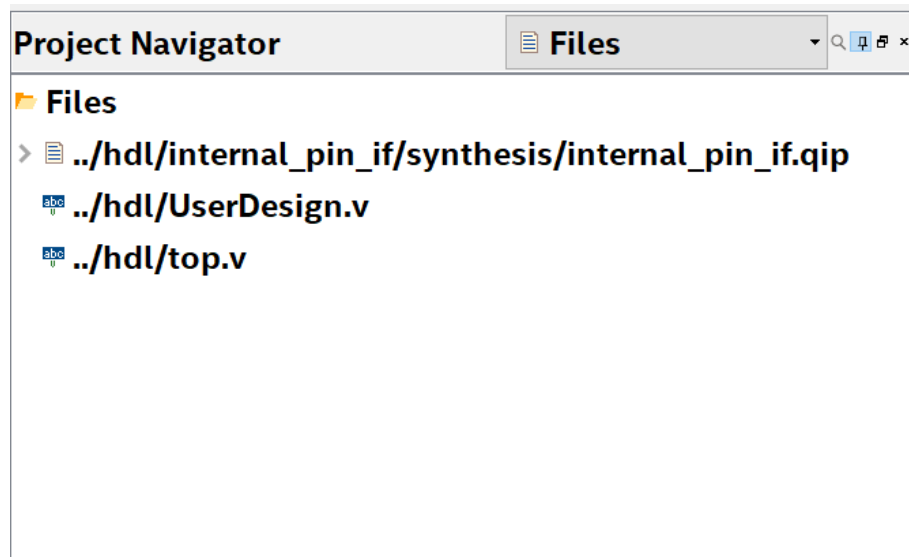
```
module switch_to_led(SW, LEDR); //create module Switch_to_LED
    input    [9:0] SW;           // input declarations: 10 switches
    output   [9:0] LEDR;         // output declarations: 10 red LEDs
    assign   LEDR = SW;          // connect switches to LEDs
endmodule
```

*Code Snippet 1: Switch-to-LED Verilog Code*

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.v** (ensuring case-sensitivity), and click **Save As...** to save your Verilog file as ../hdl/switch\_to\_led.v .
- ☐ Make sure Project Navigator is showing Files. If not use the pull down to select Files as shown in Figure 9.



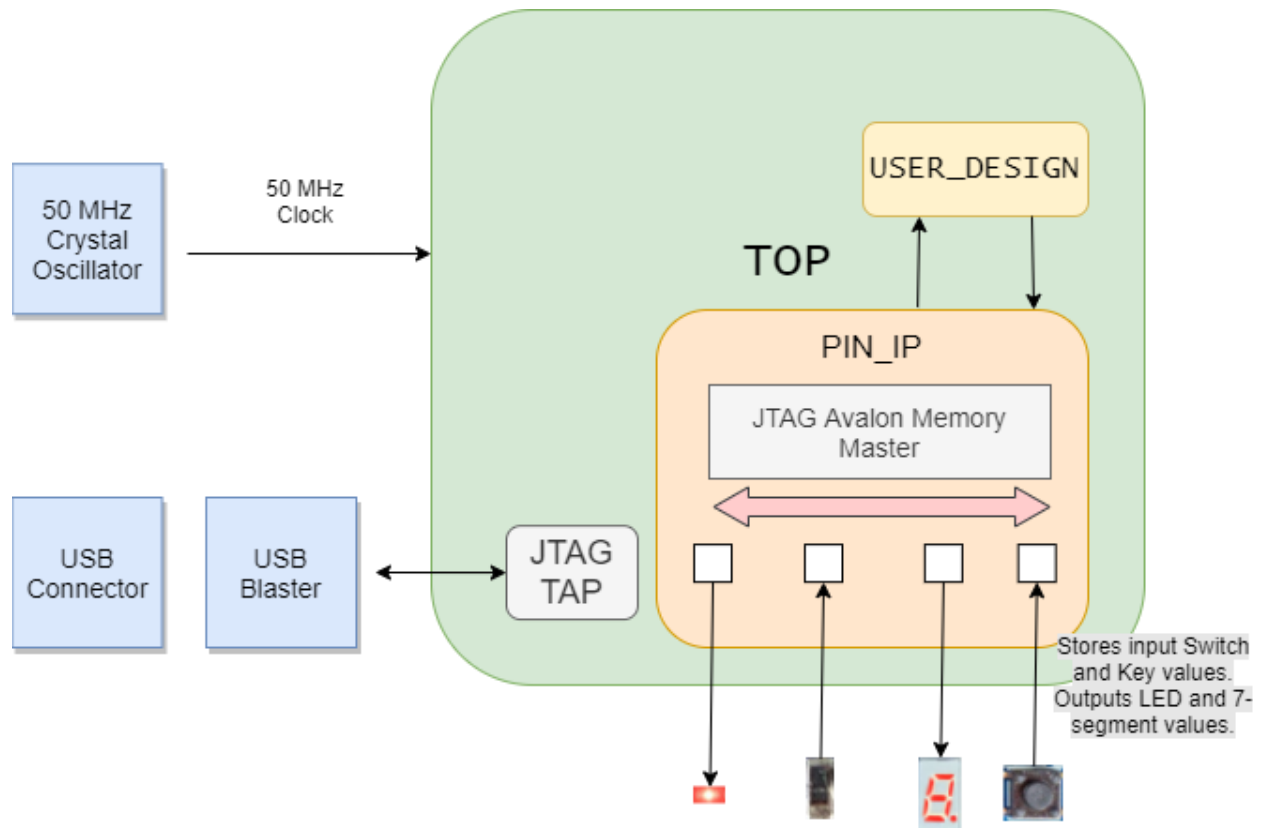
*Figure 9: Project Navigator*

- ☐ Right click on ../hdl/top.v and select Set as Top Level Entity.
- ☐ Double click on ../hdl/top.v and the file will open in a tab.

Note line 2: ``include "../hdl/pin_ip.v"` shows a fourth design file in your project not listed in Project navigator → Files. This is part of the design hierarchy as well. An alternate way of including this would have been to add this file through the settings → files menu, but that is not necessary in this case.

### 3.3 Understanding the design hierarchy

If you had a physical board in your hands, you would use the **Pin Planner** to tell Quartus what FPGA pins on the Cyclone V GX Starter Kit development board are connected to the switches and LEDs used in the circuit. Remember that the FPGA is very flexible, including the solder balls on the bottom of the integrated circuit that connect the FPGA to LEDs, switches and other components routed in the printed circuit board. To understand the process, refer to the diagram on below.



*Figure 10: Remote Board Functionality Diagram. There are three main modules to the Hands-free remote board: Top, Pin\_IP, and the User\_Design. Note that your user design in this case is called `switch_to_led.v` and will be compiled as part of your overall project. Should you have had a physical board in your hands, you would have only needed `switch_to_led.v` as a single design file in your project, made the top level entity `sw_to_led` and assigned the SW and LEDR ports to pins on the FPGA. In this case, there is a single port called `CLOCK_50` that is listed in your `top.v` design file that needs assignment. You might ask yourself why don't the SW and LEDR pins need to be assigned? That's because they are virtualized in the remote console we have designed, and you can't change the live SW pins or view LEDs from the board.*

- **Top** is the top level of the design hierarchy and connects the `pin_ip` to the `User_Design` module. In this level, it is necessary for the user to *instantiate* their design (**User\_Design**) in order to have the system run their hardware design. Without the remote board, this top level would not be needed, as you would need to use the pin assignment tool to directly connect your user design to the pins of the FPGA.
- **Pin\_IP** is a “module” in your design that exercises the inputs and monitors the outputs of your `User_Design`. This gives the remote board the same functionality as FPGA pins. Inputs to **Pin\_IP** directly corresponds to the GUI interface of switches, push buttons (KEYS), LEDs and what are called seven segment displays useful for displaying letters and numbers.

- Although it may look like we are physically trying to connect a SW, KEY, etc. to the **Top** level, we connect your User Design (in this case Switch\_to\_LED) to the **Pin\_IP block**, that collects the input Switch and Key Values and sends out to the LED and seven segment values to run your switch to LED design.
- As seen in Figure 10, the **Top** level has a single signal that leaves the FPGA in the Verilog code and that is the clock. The clock signal is synchronization signal for the overall design. This input at the **Top** level is necessary because the complex module that handles communication wirelessly requires a clock. You can also observe some signals in the diagram from the JTAG TAP. These are not programmable by the User and are part of every FPGA design. You cannot access these signals from Verilog code.

### 3.4 Instantiating User Design in Top

Instantiating is a word often used in hardware design meaning to infer your design into a higher-level part of the design hierarchy. Now that we understand the levels of our remote environment, we can begin to instantiate our design to get our online board up and running.

- You will see a comment on line 16 that will direct you to type your user design below. Below that comment you will instantiate your design by typing the code in Code Snippet 2 into line 21.

```
switch_to_led    i_switch_to_led (.SW(SW), .LEDR(LEDR));
```

*Code Snippet 2: switch\_to\_led Instance*

- Your `top.v` file should look like Figure 11 below.

```



13 wire [6:0] HEX0, HEX1, HEX2,
14     HEX3, HEX4, HEX5;
15 wire [9:0] SW;
16 wire [3:0] KEY;
17
18 // Parameter interconnect wires
19 wire [31:0] param1, param2, param3;
20
21 // User instantiates design below
22 switch_to_led i_switch_to_led (.SW(SW), .LEDR(LED));
23
24 // IP to allow simple user design interfacing with development kit
25 pin_ip platform_designer_pin_ip( .CLK_50(CLOCK_50),
26                                  .LEDR_top(LED_top),
27                                  .HEX0_top(HEX_top_0),
28                                  .HEX1_top(HEX_top_1),
29                                  .HEX2_top(HEX_top_2),
30                                  .HEX3_top(HEX_top_3),
31                                  .HEX4_top(HEX_top_4),
32                                  .HEX5_top(HEX_top_5),
33                                  .HEX0(HEX0),
34                                  .HEX1(HEX1),
35                                  .HEX2(HEX2),
36                                  .HEX3(HEX3),
37                                  .HEX4(HEX4) );

```

*Figure 11: Instantiating switch\_to\_led into the top module*

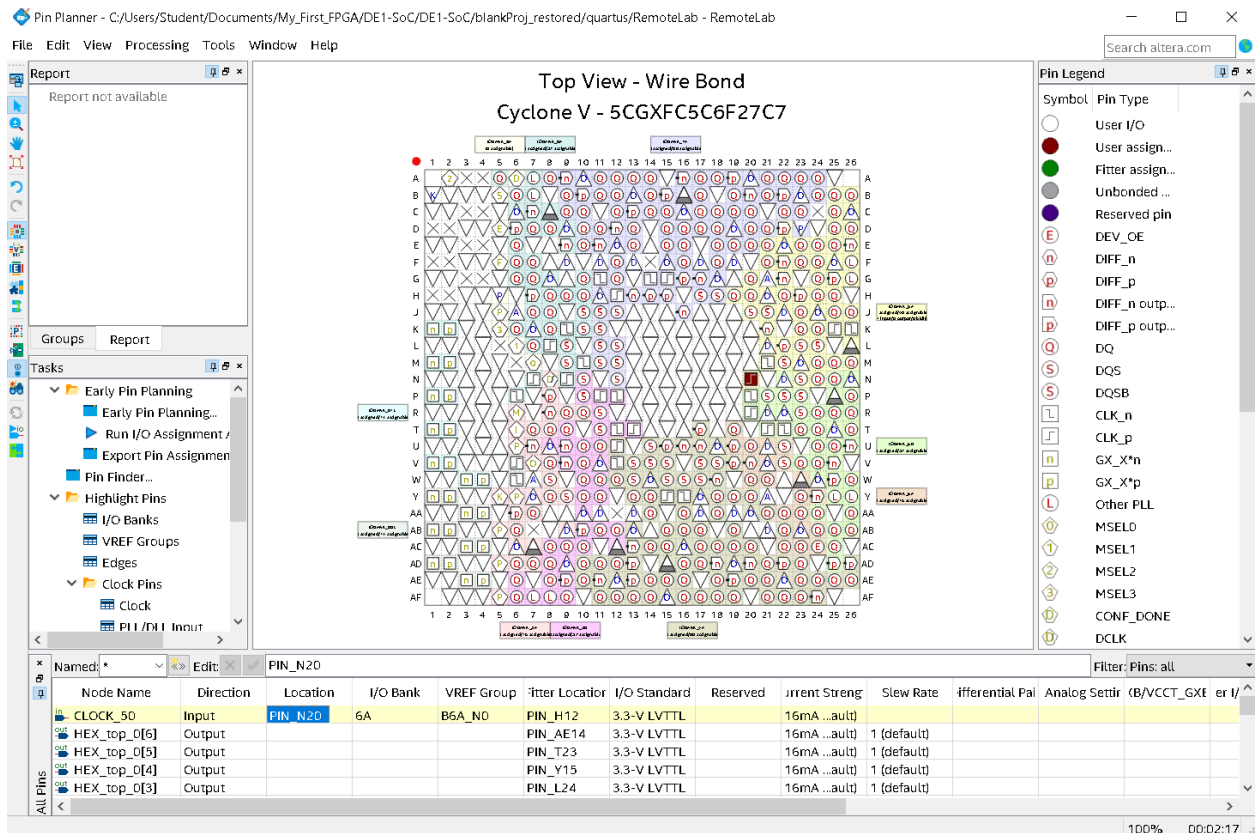
### 3.5 Setting up your pin assignments

In this step you will bind the CLOCK\_50 pin to a physical pin location on the FPGA. You have changed the device type and added new code, and by running the first step in the design flow called synthesis, you can run the subsequent pin planning step.

- ☐ Towards the top of the Quartus GUI, click on this icon: . This is called Analysis and Synthesis. This will bind the project with the new device and allow Quartus to understand the ports in your design. It will also show syntax errors should you have any problems in your Verilog code. This step takes roughly 1:30 minutes and seconds.
- ☐ There are several ways to assign pins. We will use the Pin Planner since it is visual and will give you a view into how the FPGA solder balls on the bottom are connected to the PCB.
- ☐ Start the pin planning by clicking on this icon: .
- ☐ A GUI will show the bottom of the FPGA with a legend of various power and ground pins and special signals.
- ☐ We are going to add the proper pin location for the CLOCK\_50 pin according to the table below:


Development Kit	CLOCK_50 pin location
Cyclone V GX Starter	PIN_N20
DE1-SoC	PIN_AF14

- . Other pins will show up as unassigned. This is ok with the remote console; it only needs a clock. Close the pin planner.



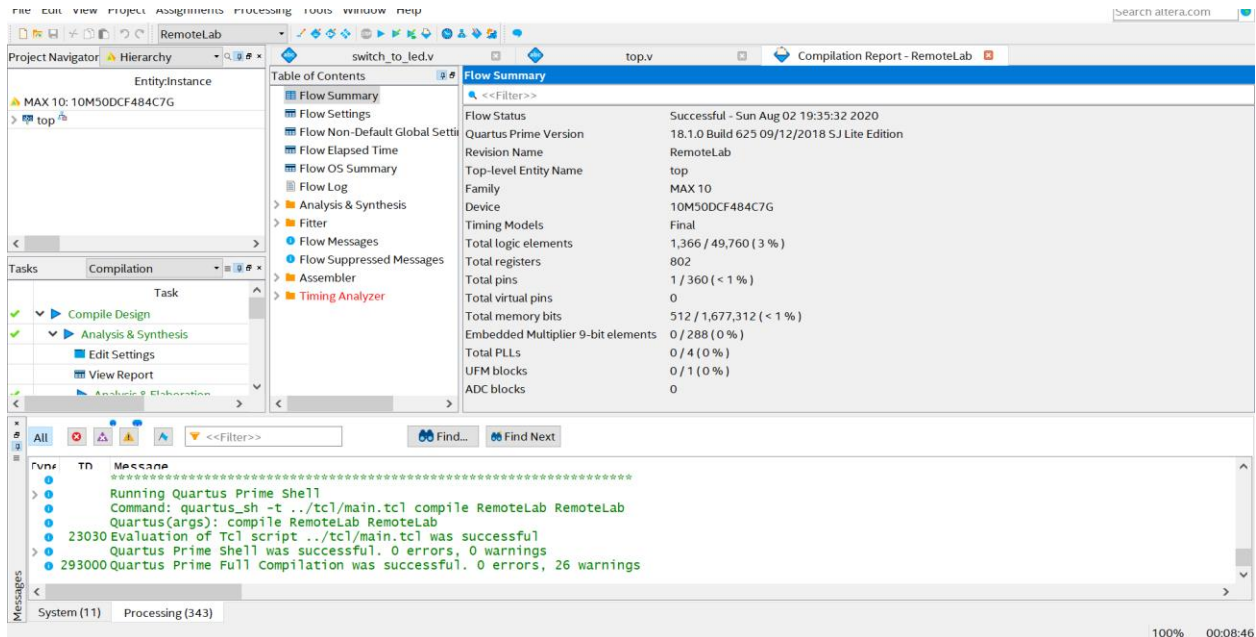
*Figure 12: Pin Planner view for Cyclone V GX Starter Kit (note DE1-SoC will have a slightly different image and device number)*

### 3.6 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. This will synthesize your Verilog code into lookup tables, run place and route (fitting in

FPGA speak), assembly which produces a programming image. The resulting file is called output\_files/RemoteLab.sof. This step takes roughly 2:45.

- ☐ The compilation should complete and there should be 0 errors. (You can ignore warnings)



*Figure 13: Quartus window showing successful compilation*

### 3.7 Setting up the remote board programming environment

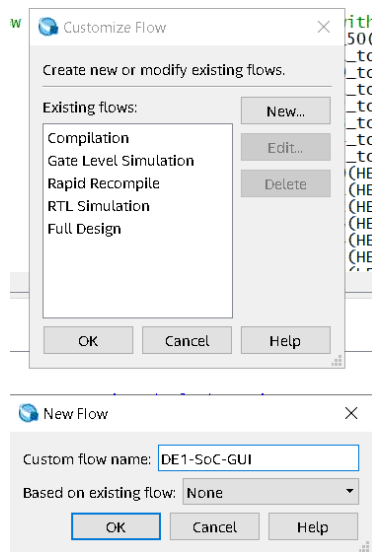
In order to get your remote environment up and running, you will create a new task in Quartus to launch the remote GUI.

- ☐ In the Tasks window, there is an icon with 3 horizontal bars. Click on that and customize.



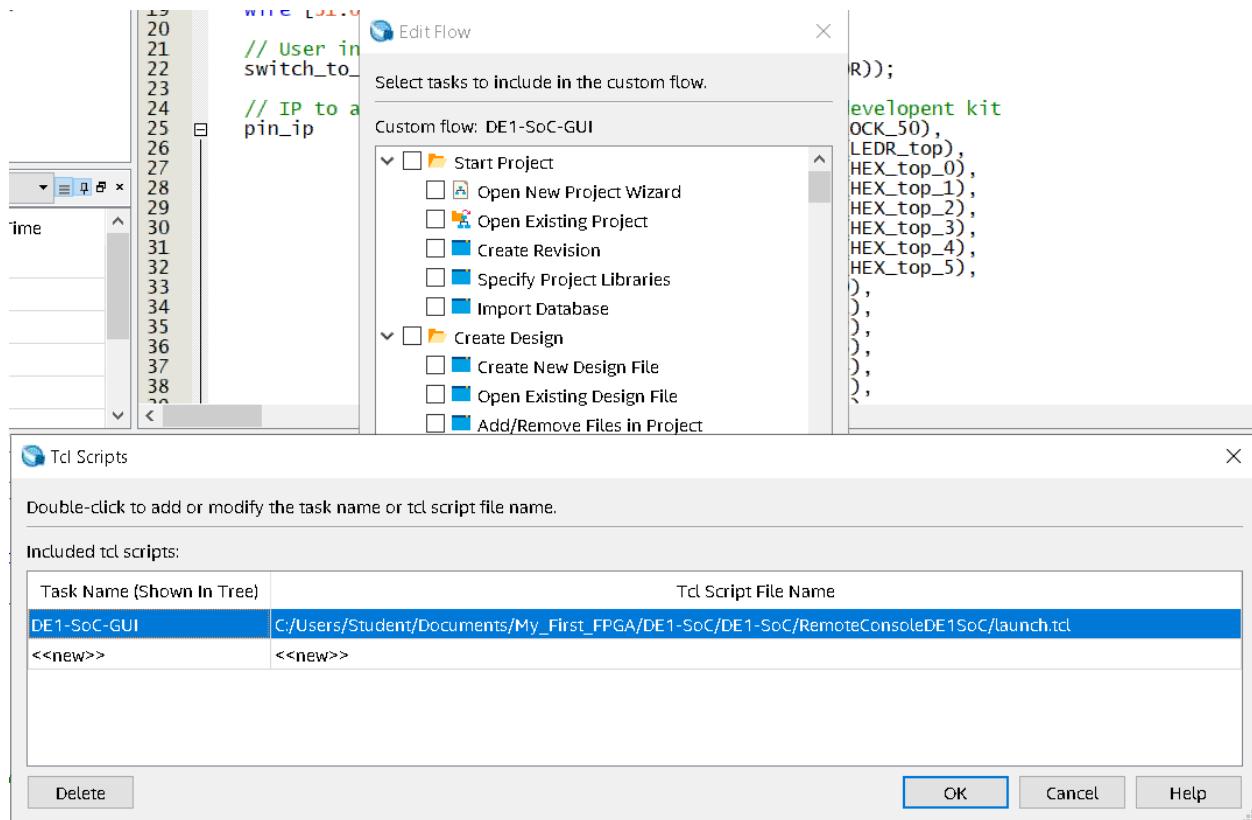
- ☐ Click on New.





*Figure 14: New Task*

- ☐ Name the Custom flow DE1-SoC-GUI. Keep existing flow as None.
- ☐ Select “TCL Scripts...”.
- ☐ Enter DE1-SoC-GUI as the Task Name and the path to the launch.tcl file in the downloaded package. Click OK through windows and your Task should show up in the Task Window.



*Figure 15: Settings for a new task*

- ☐ This next step is a bit of a hack allow us to use the DE1-SoC GUI for the Cyclone V GX Starter kit. Once we have a native Cyclone V GX Starter Kit, this step will go away. Navigate to this folder: C:\Users\Student\Documents\My\_First\_FPGA\DE1-SoC\RemoteConsoleDE1SoC using File explorer.

**The rest of these steps in Section 3.7 only apply if you are using the Cyclone V GX Starter Kit:**

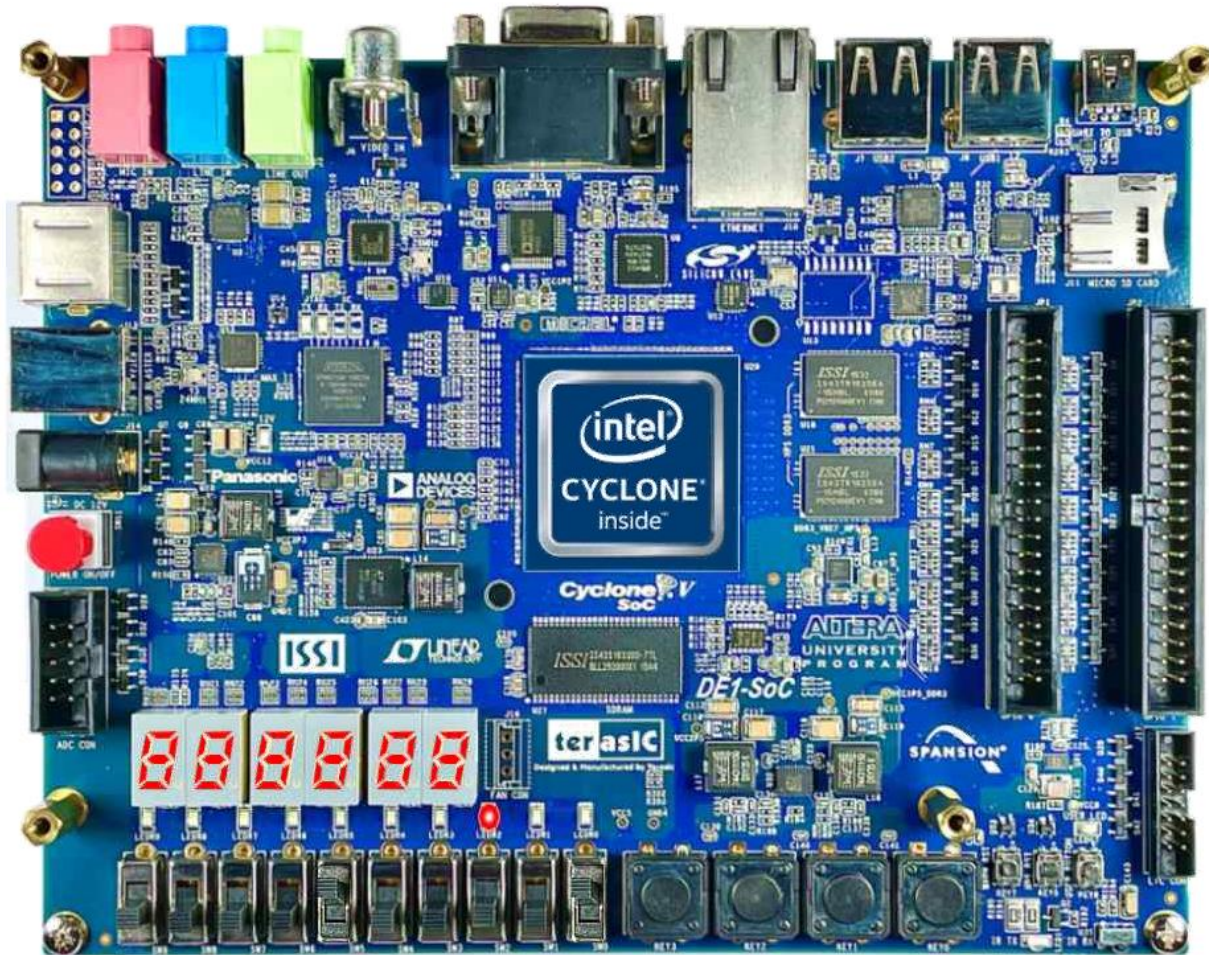
- ☐ Right click on setup.tcl and select your favorite editor listed.
- ☐ Replace lines 6 through 8 with the lines shown in Code Snippet 3.
- ☐ Save the file.

```
set cfg_part "5CGXFC5C6F27C7"
set ign_part ""
set prog_regex "*Blaster*"
```

*Code Snippet 3: setup.tcl line replacement*

### 3.8 Programming Remote Board

- ☐ Right click on DE1-SoC-GUI and hit Start
- ☐ Click on “select .sof file” and navigate to output\_files/RemoteLab.sof, select, and click link. Wait about 20 seconds for the GUI to pop up. If its hidden from view, highlight the feather icon at the bottom of your screen.



32-bit Parameter 0

32-bit Parameter 1

```
Board launched!  
Connected to USB-Blaster [USB-0]  
Time remaining: 17:45  
Ping: 17 ms
```

*Figure 16: Development Kit Viewer Window*

### 3.9 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.

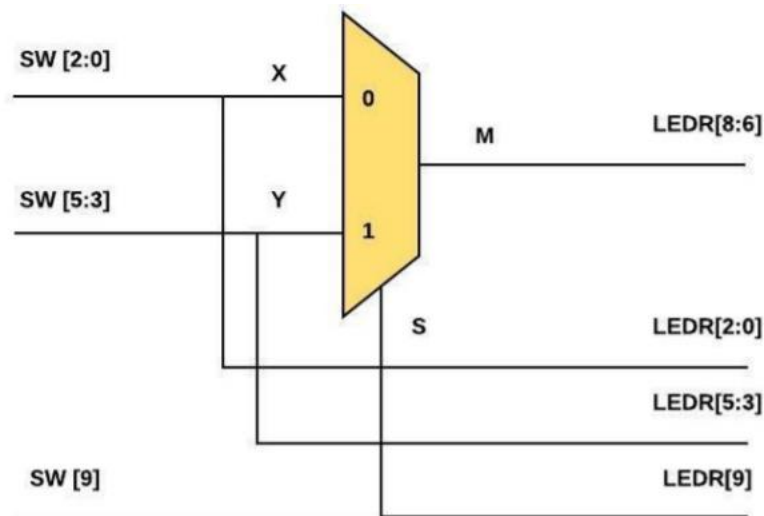
Congratulations!

You have just completed the switch\_to\_led lab using the DE1-SoC Remote Board. Make sure to close the remote GUI upon completion.

## LAB 4: 2 to 1 multiplexer

### Summary

Follow the steps from last lab and implement a 3 bit wide 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 17: Logic Flow for a 2 to 1 Mux*

### Lab Instruction

#### 4.1 2-1 Mux Verilog Code

There are several approaches to this lab. One 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 switch\_to\_led.v on how to declare the ports on your module. This means to include the module statement and inputs/output definitions.

If you are brand new to Verilog coding, the other option is to review, copy, and paste the code from in section 4.2 and then return back to this section of the lab manual. Once you copy the code, File → New, paste (control-v) and save the Verilog file as **mux\_2\_to\_1.v**.

#### CHECKLIST IF YOU DON'T COPY CODE FROM SECTION 4.2

- ☐ 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.

#### CONTINUOUS ASSIGNMENT:

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

*Code Snippet 4 Continuous-style MUX*

#### 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

```

*Code Snippet 5: If-style MUX*

## PROCEDURAL ASSIGNMENT “CASE” STATEMENT

```

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

```

*Code Snippet 6: Case-style MUX*

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 Working 2 to 1 MUX Verilog Code

Code Snippet 7 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 slides switches
    output [9:0] LEDR;         //Output Declarations:10 red LED Lights

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

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

    assign LEDR[8:6]=M;         //Assign internal signal 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

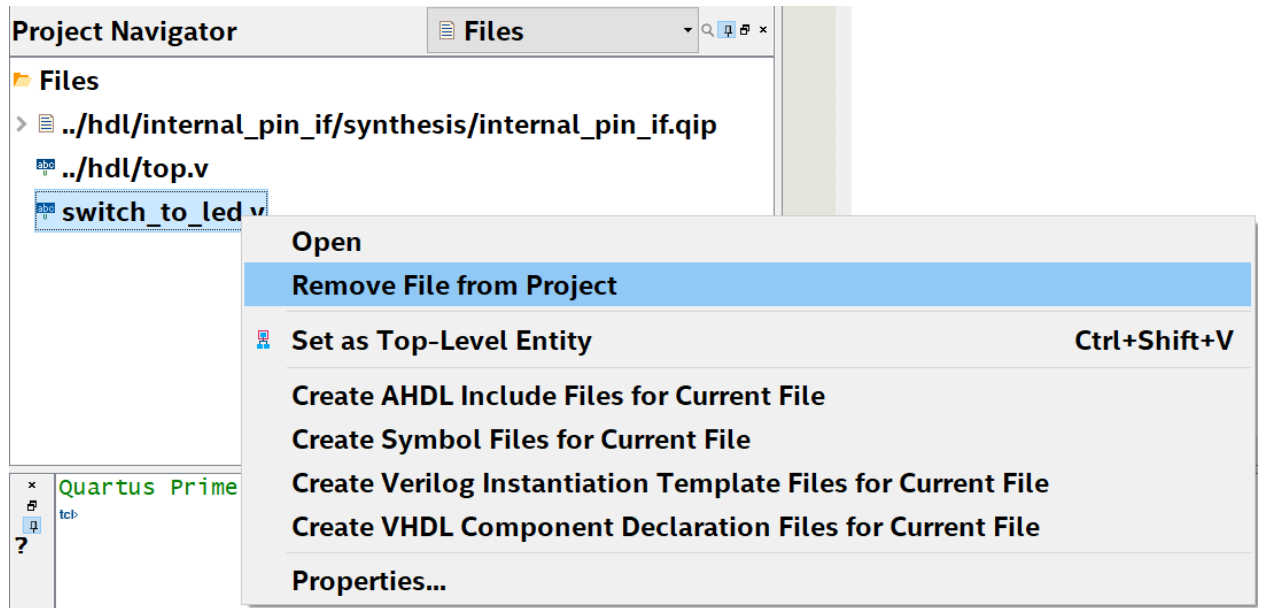
```

#### *Code Snippet 7: Complete MUX implementation*

### 4.3 Instantiating mux\_2\_to\_1 into top module

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 18: Changing Top Level Entity*

- ☐ Right click **switch\_to\_led.v** and remove this source file from your project.
- ☐ Instantiate your new mux\_2\_to\_1 module into module top. Hint the code to copy looks like this:

```
mux_2_to_1    i_mux_2_to_1 (.SW(SW), .LEDR(LEDR));
```

*Code Snippet 8: MUX instantiation*

- ☐ Note that you will need to remove or comment out any previous files (switch\_to\_led).
- ☐ Compile your design by pressing  along the top of the Quartus window.
- ☐ The GUI will come up as it did for the switch\_to\_led lab. Test the switches and LEDs on the remote console. Note that switch 9 selects between the values on either switches 2-0 or 5-3 and puts their value on LEDs 8-6.

## 4.4 Viewing the 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 Tools → Netlist Viewers → RTL Viewer. Once opened, you can navigate the design hierarchy and observe your mux\_2\_to\_1 design in a schematic form. Use the left and right mouse buttons to navigate through the design hierarchy.

	Task	Time
✓	▼ ▶ Compile Design	00:02:24
✓	▼ ▶ Analysis & Synthesis	00:01:11
	■ Edit Settings	
	■ View Report	
✓	▶ Analysis & Elaboration	
	> ▶ Partition Merge	
	▼ 📁 Netlist Viewers	
	🔍 RTL Viewer	
	🔍 State Machine Viewer	
	🔍 Technology Map Viewer (Post-Mapping)	

*Figure 19: RTL Viewer selection*

## 4.5 Viewing the chip planner

Another topic that is interesting is looking at the chip level design and see the various physical objects in your design.

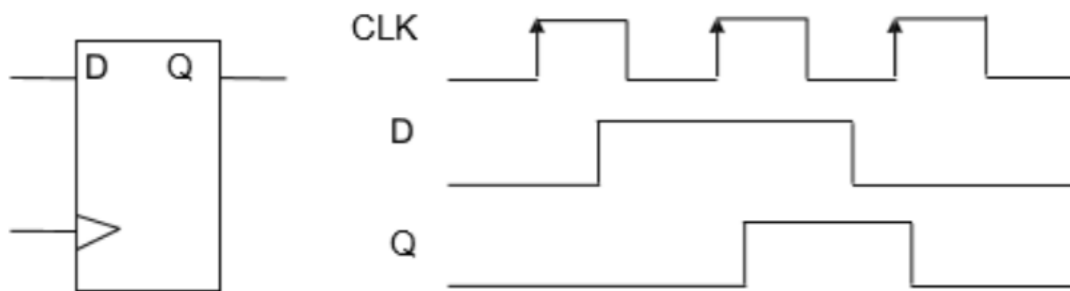
- Close the RTL Viewer
- Open the Chip Planner: Tools → Chip Planner
- Zoom in and out of the chip planner view by clicking on the magnifying glass and clicking right and left mouse to zoom in/out. The darker region rectangles contain the lookup tables that are utilized in this small design. Chip Planner has a wide range of features. Highlighting the arrow  and selecting an object will give its description in the right panel.

# 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<sup>1</sup> 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.



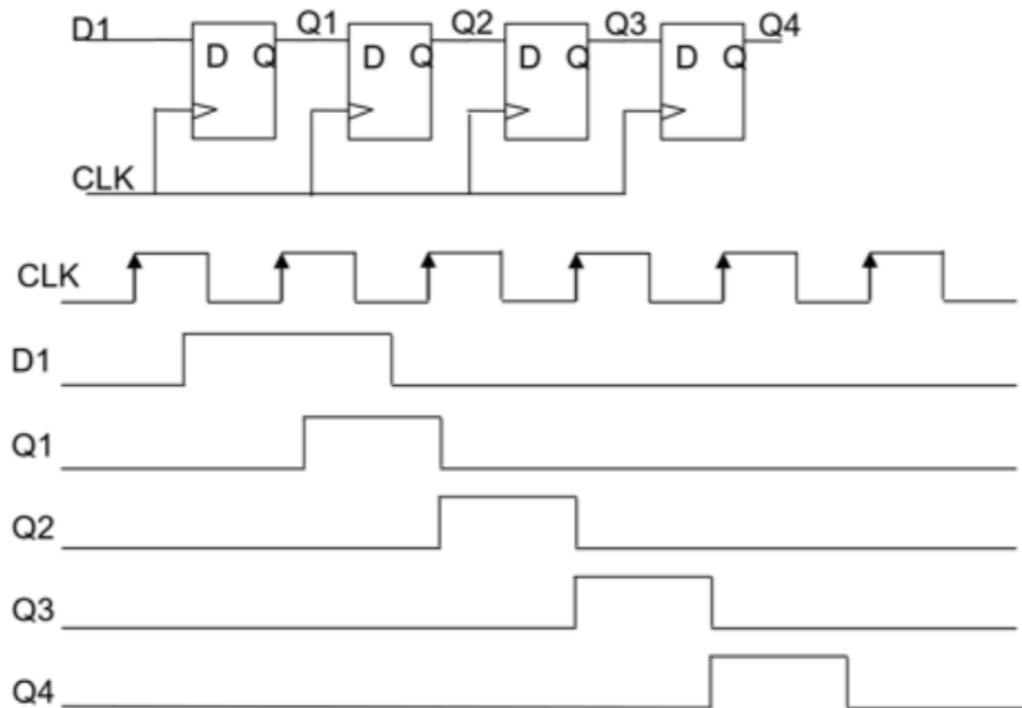
*Figure 20: Shift Register Diagram*

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

### 5.1 Knight Rider Verilog Code

- The following Verilog code is the starting point for your Knight Rider design, but there are some bugs. You will remove your mux\_2\_to\_1.v file and add the knight\_rider.v file to your project.



*Figure 21: Shift Register Diagram*

- The code given in Code Snippet 9 **intentionally** has an error. See if you can find it.

```

module knight_rider(    input wire CLOCK_50,
                        input resetn,
                        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 or negedge resetn)
    begin
        if (~resetn)
            count <= 0;
        else if (count_up)
            count <= count +1'b1;
        else
            count <= count - 1'b1;
    end
    always @ (posedge slow_clock)
    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);

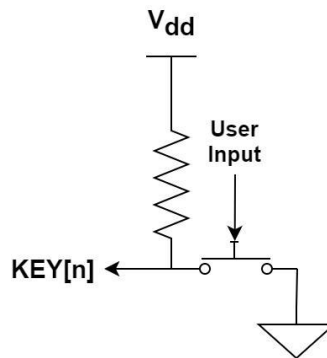
    parameter COUNTER_SIZE = 5;
    parameter COUNTER_MAX_COUNT = (2 ** COUNTER_SIZE) -1
    reg [COUNTER_SIZE-1:0] count;
    always @(posedge fast_clock)
    begin
        if(count==COUNTER_MAX_COUNT)
            count <= 0;
        else
            count<=count+1'b1;
    end
    assign slow_clock = count[COUNTER_SIZE-1];
endmodule

```

*Code Snippet 9: Knight Rider w/ bugs*



## 5.2 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 to instantiate your knight\_rider module into module top and comment out any previous modules you instantiated in module top (eg mux\_2\_to\_1). Note the CLOCK\_50 and LEDR connections are fairly obvious as the ports they connect to have the same name, however for resetn, requires an active low switch or key (KEY is Terasic terminology for push button). The keys are active low, meaning when not pressed produce a “1” and when pressed produce a “0” state. Directly connecting resetn to KEY[0] will produce the desired results.



*Figure 22: KEY function*

## 5.3 Debugging Code

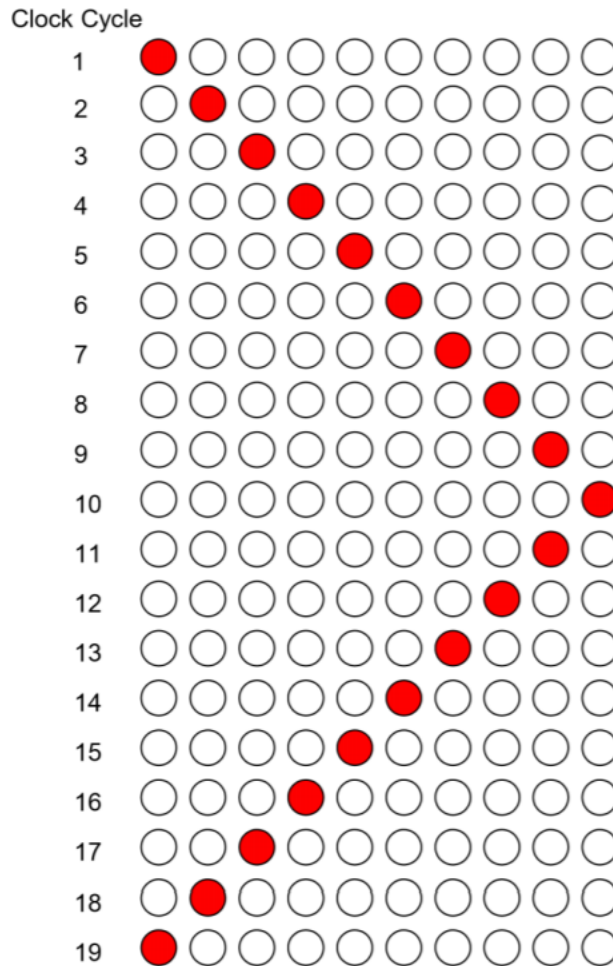
- ❑ Click on the **Play**  button and run **Analysis & Elaboration**. This source code has one syntax bugs. Look at the transcript window on the bottom and observe the errors that are flagged with  symbol. Note sometimes the flagged line number might be one line away from the offending code.
- ❑ Carefully look at the source code and fix the errors and continue to recompile until the compilation steps run to completion.

## 5.4 Running knight\_rider on your remote board

- ❑ By now you should have the hang of how to program the FPGA image into the DE1-SoC 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 with the LED taking about 1 second to blink all 10 LEDs in one direction.



*Figure 23: Example of Knight Rider sequence*

What do you see?

## 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 run the DE1-SoC remote board.

## 5.5: 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. The remote console isn't as smooth as the actual hardware, so it might be hard to catch. You can increase **COUNTER\_WIDTH** by one, recompile, and if slow enough you will observe the error.

- ☐ Dang! That lab instructor created another error in the design, LEDs 9 and 0 double blink! 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 DE1-SoC remote 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.

Note: The **COUNTER\_SIZE** looks good when set to 23 or 24.

## 6 Document Revision History

List the revision history for the application note.

Name	Date	Changes
Shawwna Cabanday	11/26/2019	Initial Release of guide
Shawwna Cabanday	12/4/2019	Revision for NoMachine implementation
Shawwna Cabanday	1/19/2020	Revision for downloaded Quartus Lite implementation
Shawwna Cabanday	1/27/2020	Minor revisions, added new figures
Shawwna Cabanday	1/28/2020	Added more information in ModelSim simulation section, minor grammar revisions and mistakes
Larry Landis	2/13/2020	Corrected some typos, added diagram for the Nios embedded system.
Damaris Renteria	2/18/2020	Added few Linux commands that differ from Window's, minor revisions.
Damaris Renteria	3/2/2020	Incorporated profiling on a target (DE10-Lite)
Damaris Renteria	3/13/2020	Corrected some typos, changed diagrams (without GPIO), added more information about profiler report.
Sam Banda/Larry Landis	8/4/2020	Adapt to the remote console
Larry Landis	2/15/2021	Adapt to the Webex Training Center and De1-SoC console with CV GX Starter Hardware
Larry Landis	3/1/2021	Add support for either CV GX Starter or DE1-SoC