INTRO TO INTEL®FPGAS AND INTEL®QUARTUS®PRIME SOFTWARE USING REMOTE HANDS-FREE CONSOLE

1. 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 the Knight Rider LEDR[0] is the only LED that turns on, you have not assigned the CLOCK_50 pin properly in your clock pin assignment (non-issue for remote console lab)

- Sometimes copying and pasting from text files into Quartus's TCL 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.

2. OBTAINING THE QUARTUS PRIME LITE DESIGN TOOLS

2.1 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, however this lab will run a remote development kit. The development kit will located in a server and you will use a GUI that looks like a video game of the actual development kit. You will be able to see LEDs and change switches to run your experiments.

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.

2.2 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 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

To get the free Intel Quartus Prime Lite tools, follow these steps:

☐ Visit this site: http://fpgasoftware.intel.com/?edition=lite.

☐ Select version 20.1 (or higher) and your PC's operating system (this lab manual is written for Windows)

☐ 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.

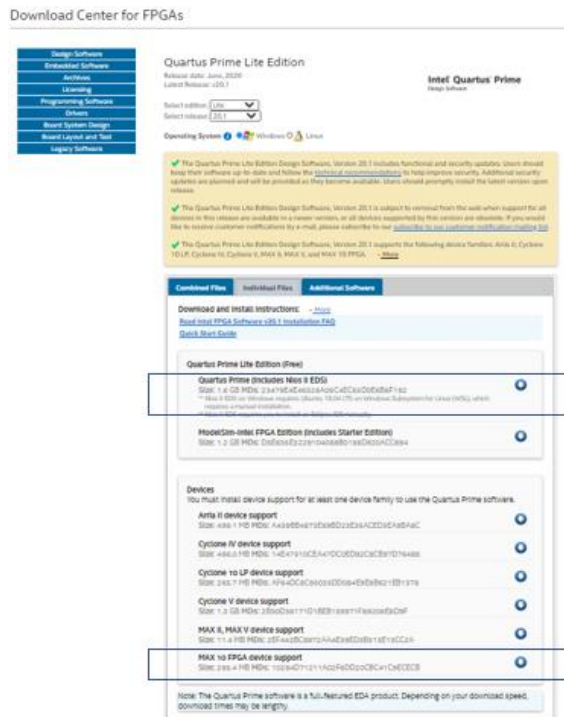*Figure 1: Quartus Prime Lite Minimum Required Files to Download*

3. UNARCHIVING A BLANK PROJECT

3.1 BACKGROUND

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.

3.2 BLANK PROJECT GITHUB DOWNLOAD

☐ To begin your Remote Hands-free project, click this link at the HandsFree Labs github site: https://github.com/intel/FPGA-Devcloud/blob/master/main/HandsFree/Devkits/DE10-Lite/Example_Projects/Intro_to_Quartus/BlankProject.qar

3.3 BLANK PROJECT ENVIRONMENT SETUP

☐ Download the file and store in a folder on your PC. We suggest saving in a directory that you will remember the location.

☐ Unarchive the BlankProject.qar file by double clicking on it. All the files you need to start your training are in this file.

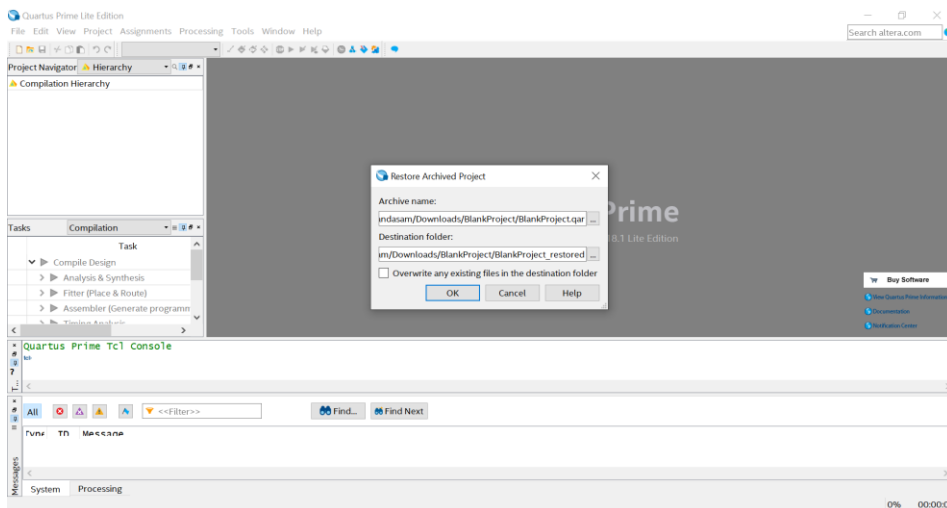*Figure 2: Restore Archived Project for BlankProject*

☐ When you open the .qar file you will see the message from Figure 2 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 3.
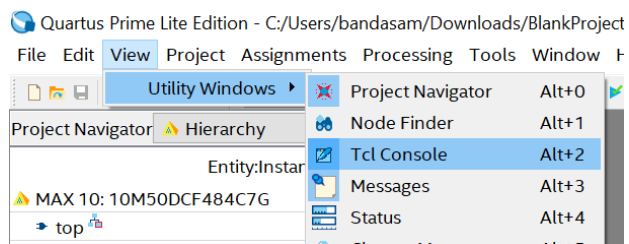
Figure 3: Utility Window dropdown

4. DESCRIBING YOUR FPGA FUNCTION USING VERILOG OR VHDL

4.1 BACKGROUND

This lab will step you through the process of a simple design from generating your first Verilog file to synthesize and compile. This lab manual is primarily written in Verilog however, we also include the code for the first lab in VHDL to understand how to mix and match both languages in the same design. 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.

4.2 CREATING A NEW FILE

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

☐ A window, shown in Figure 4, should pop up. Click on **Verilog HDL File** and then **OK**. For **VHDL**, skip this step.
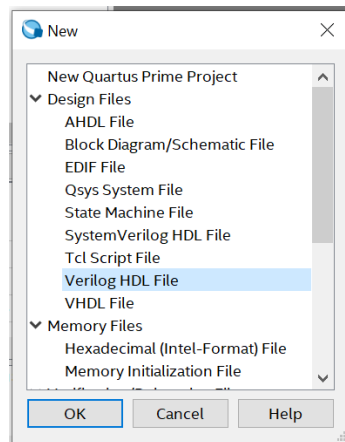
Figure 4: New File Window

4.3 ADDING VERILOG/VHDL 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. You can also download the **Verilog** file from this github link . See below for VHDL and download here.

```verilog
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
```

```
-- VHDL
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity switch_to_led is
        port
        (
                SW      : in signed (9 downto 0);
                LEDR  : out signed (9 downto 0)
        );

end entity;

architecture rtl of switch_to_led is
begin

        LEDR <= SW;

end rtl;
```

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.

Verilog is case sensitive, while VHDL is not. Quartus can compile projects that combine both languages. Although the syntax of Verilog and VHDL is considerably different, the functionality is similar. Note you need to include either switch_to_led.v or switch_to_led.vhd in your project, not both. If you have both modules, right click and remove one from the project navigator panel.

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 (.v)/VHDL (.vhd) file. Make sure you save the file in the hdl directory.

11

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 5 below.

Figure 5: Analysis and Elaboration check mark
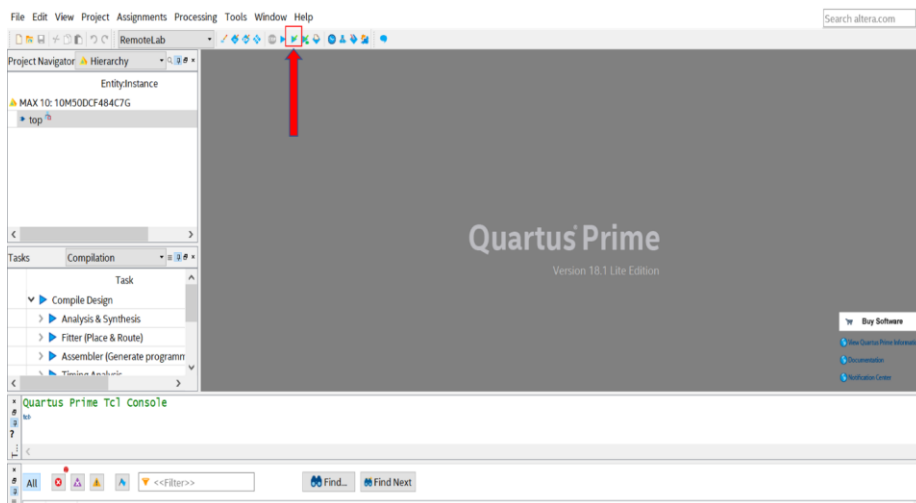
4.4 REMOTE BOARD PIN ASSIGNMENT

If you had a board in your hands, you would need to use the **Pin Planner** to tell Quartus what FPGA pins on the DE10-Lite development board are connected to the switches and LEDs used in the circuit. Because our board is working remote, the pins have already been assigned for you. To understand the process, refer to the diagram on Figure 6 below.
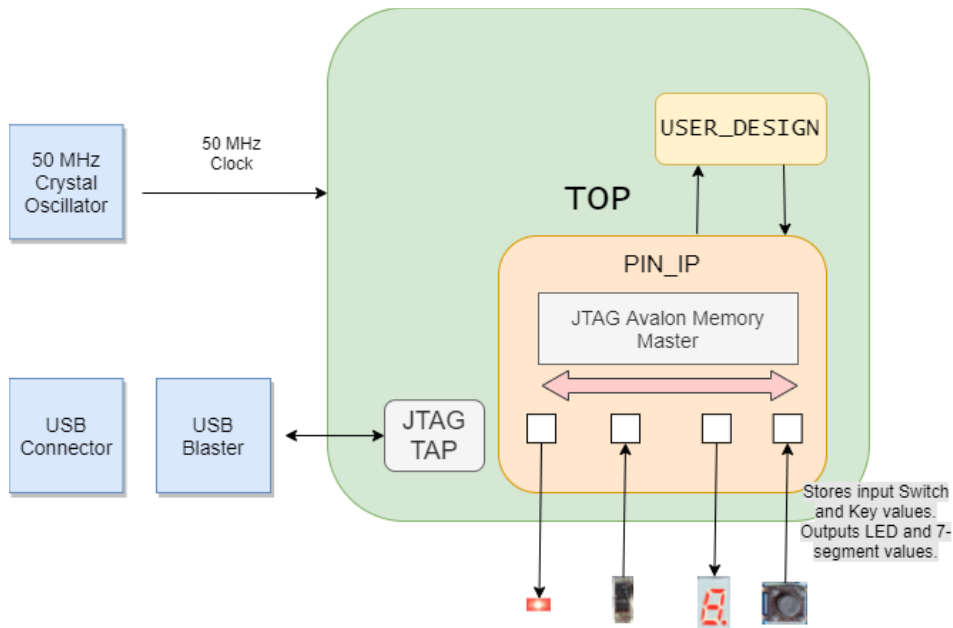
Figure 6: Remote Board Functionality Diagram

- There are three main levels to the Hands-free remote board: **Top**, **Pin_IP**, and the **User_Design**.

- **Top** is the main level of the design and can fit all kinds of design files. At this level of the design hierarchy, it is necessary for the user to instantiate their design (**User_Design**) in order to have the system run their program. Your user design for this lab is switch_to_led. Should you have a development kit in hand, this top level would not be needed, as you would need to use the pin assignment tool for your connections.

- **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 the LED and seven segment values to run your User Design on the remote console.

13

- As seen in Figure 6, at the **Top** level the only pin that goes to the FPGA is the clock. This 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.

4.5 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.

- ☐ Go to your Project Navigator and double click on Top to open your top.v file.
- ☐ You will see a comment that will direct you to type your user design below. Below that comment you will initiate your design by typing the following code. This will work for either a Verilog or VHDL description of switch_to_led.

```
switch_to_led i_switch_to_led (.SW(SW), .LEDR(LEDR)); //module_name instance_name
```

- ☐ Your `top.v` file should look like Figure 7 below.

14

```
11        // Peripheral interconnect wires
12        wire [9:0]  LEDR;
13        wire [7:0]  HEX0, HEX1, HEX2,
14                    HEX3, HEX4, HEX5;
15        wire [9:0]  SW;
16        wire [1:0]  KEY;
17
18        // Parameter interconnect wires
19        wire [31:0] param1, param2, param3;
20
21        // User instantiates design below
22
23          /* In this design successive LEDs illuminate back
24          ** and forth acrross the GUI's LED array.
25          **
26          ** The speed of the LED scan can be adjusted by changing
27          ** parameter 1 on the GUI. 0x19 is approximately 1 second
28          ** per LED.
29          **
30          ** 7-segment displays show the BCD value of the LEDs.
31          **
32          ** Resets to the design can be made by pressing either
33          ** pushbutton, or toggling SW0.
34          */
35
36          switch_to_led i_switch_to_led (.SW(SW), .LEDR(LEDR)); //module_name instance_name
37
38        // IP to allow simple user design interfacing with develoment kit
39
40        pin_ip          platform_designer_pin_ip(  .MAX10_CLK1_50(MAX10_CLK1_50),
41                                                    .LEDR_top(LEDR_top),
42                                                    .HEX0_top(HEX0_top),
43                                                    .HEX1_top(HEX1_top),
```

Figure 7: Instantiating switch_to_led in to the top module

☐ Save your design. In the upper left Project Navigator Section with the tab set to Files (default is Hierarchy), you should see the following files in your project.
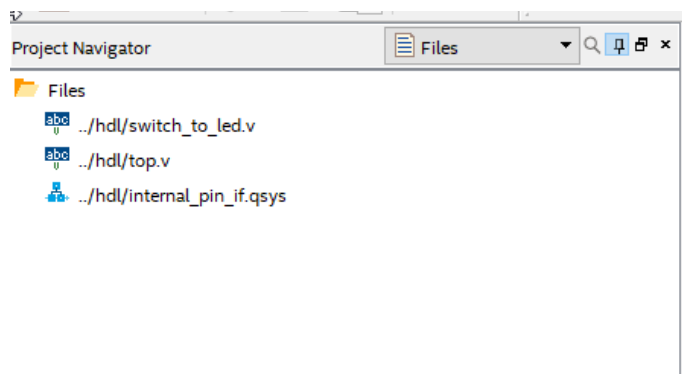

Figure 8: Project Navigator Files tab

4.6 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.

☐ After roughly 2 to 7 minutes depending on your CPU type and amount of memory, the compilation should complete and there should be 0 errors. (You can ignore warnings).
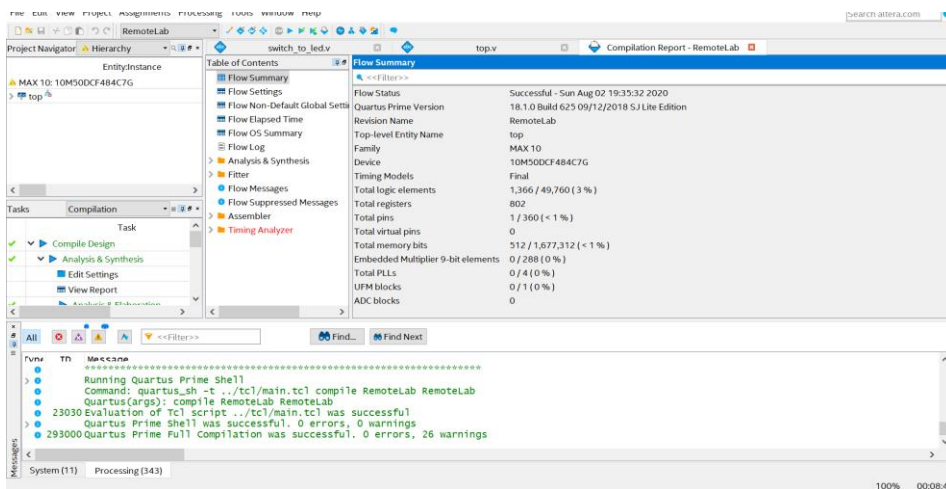

Figure 9: Quartus window showing successful compilation

5. PROGRAMMING THE REMOTE BOARD

☐ If you are on the same network as the development board, follow the subsequent checkbox items below. If not, navigate to the folder where you are running the project and look for the output_files/RemoteLab.sof . Send the .sof file to the instructor so they can program the DE10-Lite development kit and demonstrate proper operation. Once verified, move to the next lab.

16

☐ The tasks menu drop down that is currently in Compilation should be changed to GUILaunchDE10Lite.

☐ Right click on the task DE10LiteTask and left click Start.

☐ After about 20 seconds, you should see the pop-up menu. If you don't see it, there is a chance it might be hidden behind the main Quartus window. If it's iconized, the icon has a picture of a feather on it. Click on the icon.

☐ If you have a locally connected DE10-Lite development kit, IPv4 should state "local" and the password field can remain blank. If you are connecting to a remote board, you will be given an IPv4 address and a password. Enter those fields.
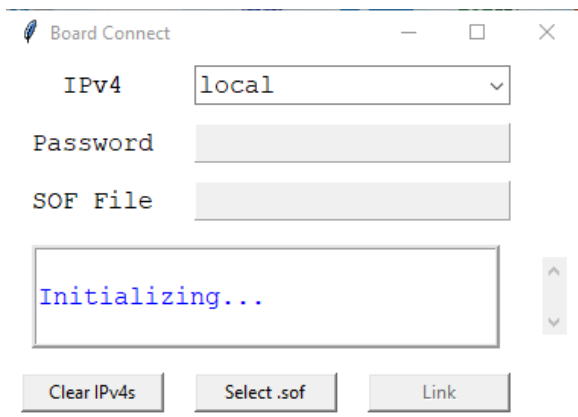

Figure 10: Board Connection Launch GUI

☐ Click Select .sof file. Navigate to your project directory → output_files. Select RemoteConsole.sof. The .sof file is the FPGA programming image. Click link. After roughly one minute, the remote console GUI will launch.

☐ This project connects the 10 slide slider switches to LEDs on the remote console development kit. You can change the values of the slider switches by left clicking on them and you will see the LEDs turn on and off. If you have a

local board you can change the control from the physical hardware switches or from the remote console. Try out both modes.

6. 2 TO 1 MULTIPLEXER (MUX)

6.1 BACKGROUND

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 select "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, we simply define a 10 bit wire SW of the form wire [9:0] SW.
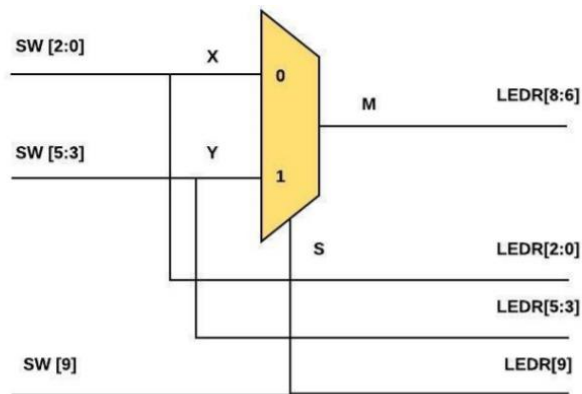

Figure 11: Logic Flow for a 2 to 1 Mux

6.2 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 section 4.3 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 coding, the other option is to review, copy, and paste the code snippet below and paste it a new Verilog file. You can also download the entire file from this link. The mux_2_to_1.v file should be stored in the hdl folder.

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.

CONTINOUS ASSIGNMENT:

```
assign    M = (S==1) ? Y:X;        //If S then M = Y else M = X

                       //All signals are of type wire
```

PROCEDURAL ASSIGMENT "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 ASSIGMENT "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.

6.3 REVISION AND INSTANTIATION 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 and other files to run the remote console, so you will need to remove switch_to_led.v from your project.
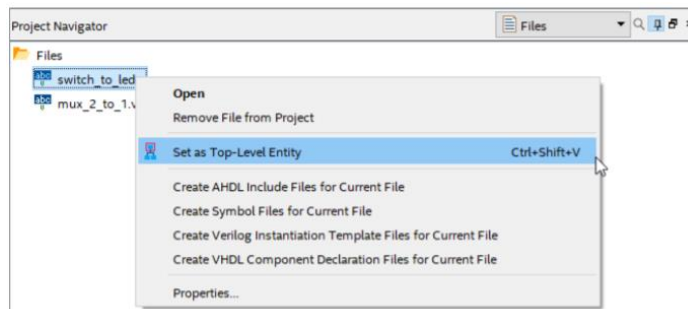

Figure 12: Changing Top Level Entity

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

☐ Navigate to Project → Add/Remove Files from project. To the right of the File Name: you will see an icon that looks like: ☐ . Click on the icon and navigate to the file ../hdl/mux_2_to_1.v . Double click on the file and it will show up in the list of the project files.

☐ 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 top) 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-sensitive.

☐ Refer back to Section 4.5 and instantiate your new mux_2_to_1 module and remove switch_to_led in module top in the file top.v .

☐ Note that you will need to remove or comment out any previous files (switch_to_led.v)

☐ Compile your design by pressing ▶ along the top of the Quartus window. Test you design on hardware using the same means you used for switch_to_led

6.4 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 13. Once opened, you should see a 2-to-1 mux similar to what is seen in Figure 14.

	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 13: RTL Viewer selection

22

Figure 14: RTL Viewer of a 2 to 1 Mux

7. KNIGHT RIDER

7.1 BACKGROUND

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 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.
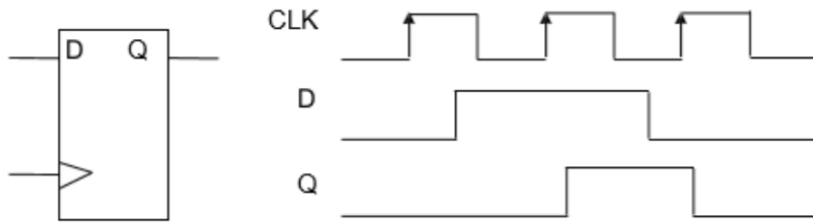
23

Figure 15: 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.

5.0: KNIGHT RIDER VERILOG CODE

☐ The Verilog code is the starting point for your Knight Rider design, but we will use a different version of the code that also utilizes the seven segment displays. Study this simplified code and see if you can figure out what the code does.
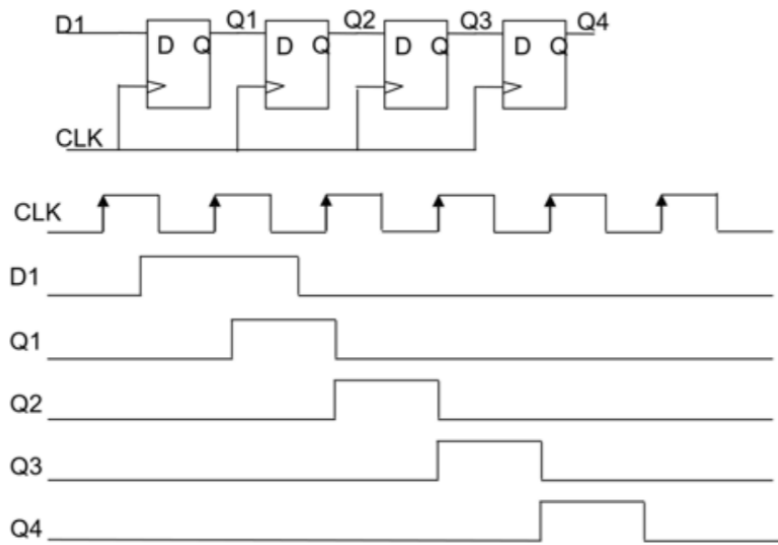
24

Figure 16: Shift Register diagram

```verilog
module knight_rider(
    input wire MAX10_CLK1_50,
    output wire [9:0] LEDR);
    wire slow_clock;
    reg [3:0] count;
    reg count_up;
    clock_divider u0(.fast_clock(MAX10_CLK1_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 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,
```

```verilog
    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"

☐ Download the knight_rider.v file.
☐ Store the file in the hdl folder.
☐ Instantiate your knight_rider.v file into Top and comment or remove out any previous modules you instantiated (eg mux_2_to_1). You can cut and paste the cut below into top.v and fill in the appropriate connections in between the parenthesis for the unconnected signals. Note that there are ports that connect to the FPGA IO pins called HEX0_top and wires between knight_rider and the pin_ip module called HEX0. It is your job to determine which is the right signal to connect between the parenthesis so that things work properly.

```
knight_rider knight_rider_instance(
 .MAX10_CLK1_50(MAX10_CLK1_50),
 .rst(~KEY[0]),
 .LEDR(LEDR),
 .HEX0(),
 .HEX1(),
 .HEX2(),
 .HEX3(),
 .HEX4(),
 .HEX5(),
 .counter_tap(param1));
```

5.3: RUNNING YOUR CODE ON YOUR BOARD

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:
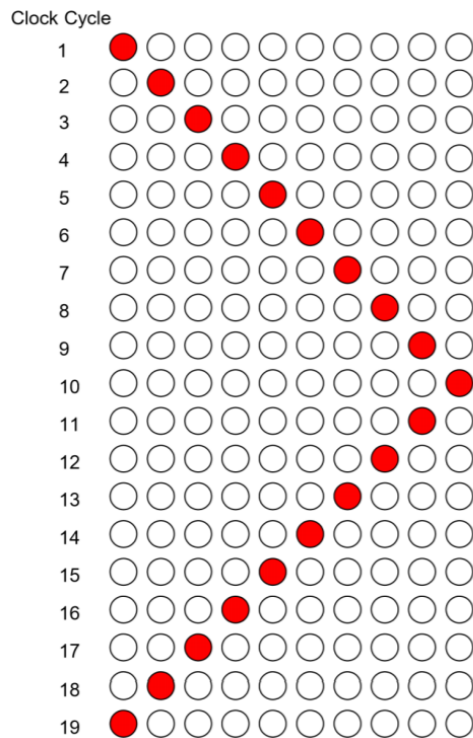
Figure 17: Example of Knight Rider sequence

What do you see? **(If it does not work, look at the next section.)**
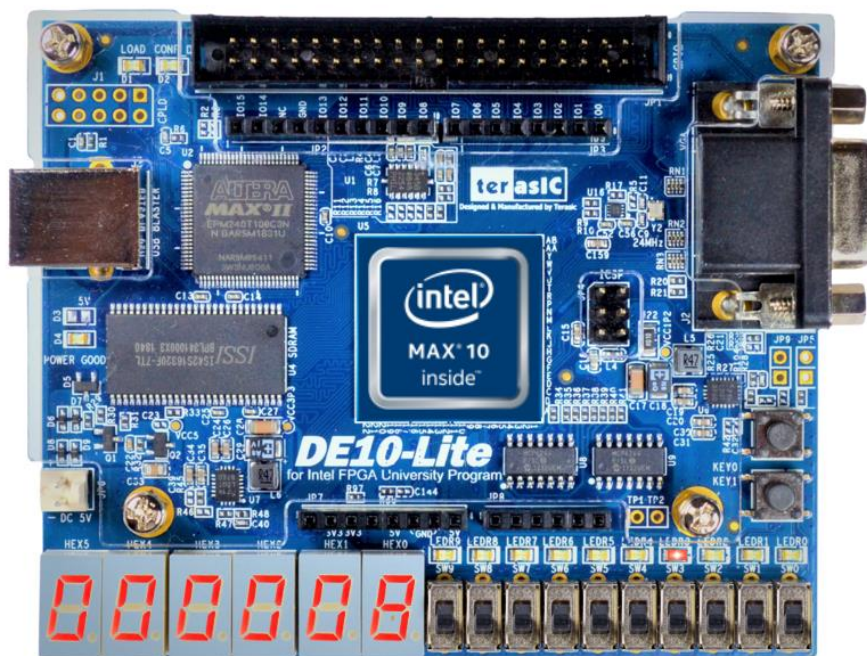
5.4: MORE DEBUGGING

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 which is too fast to 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.

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. Once you figure out the proper divide ratio, you will enter the value in the remote console Parameter 0 entry in hexadecimal.

- [ ] Basically, think about a divide ratio that is 2^N where N is the width of the counter. Calculate the counter tap 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 parameter setting. Hint you will need to use the log function.
- ☐ Adjust the Parameter 0 value on the remote console with the hexadecimal value you calculated.

Thanks for taking time learning how to develop Intel FPGA products. We hope you found this lab informative.

8 DOCUMENT REVISION HISTORY

List the revision history for the application note.

Name	Date	Changes
Shawnna Cabanday	11/26/2019	Initial Release of guide
Shawnna Cabanday	12/4/2019	Revision for NoMachine implementation
Shawnna Cabanday	1/19/2020	Revision for downloaded Quartus Lite implementation
Shawnna Cabanday	1/27/2020	Minor revisions, added new figures
Shawnna 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 profilier report.
Larry Landis	11/4/2020	Updates for remote console.
Larry Landis	11/16/2020	Added switch_to_led VHDL version