

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

---

© 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

<b>HINTS AND TRICKS .....</b>	<b>4</b>
<b>LAB 1: OBTAINING THE QUARTUS PRIME LITE DESIGN TOOLS .....</b>	<b>5</b>
Background.....	5
Installation.....	5
<b>LAB 2: UNARCHIVING A BLANK PROJECT .....</b>	<b>8</b>
Summary .....	8
Lab Instruction .....	8
2.0: Blank Project GitHub Download .....	8
2.1: Blank Project Environment Setup.....	9
<b>LAB 3: DESCRIBING YOUR FPGA FUNCTION USING VERILOG .....</b>	<b>11</b>
Summary .....	11
Lab Instruction .....	11
3.0: Creating a New File.....	11
3.1: Adding Verilog Code.....	12
3.2: Remote Board Pin Assignment .....	13
3.3: Instantiating User Design in Top .....	15
3.4: Compiling Your Code .....	16
3.5: Programming Remote Board .....	17
3.6: Testing Your Design.....	18
<b>LAB 4: 2 TO 1 MULTIPLEXER.....</b>	<b>19</b>
Summary .....	19
Lab Instruction .....	19
4.0: 2-1 Mux Verilog Code.....	19
4.1: Revision and Instantiation Control .....	22
4.2: Viewing Design Schematic .....	23
4.3: Working 2 to 1 MUX Verilog Code.....	23
<b>LAB 5: KNIGHT RIDER .....</b>	<b>25</b>

Lab Instruction .....	26
5.0: Knight Rider Verilog Code .....	26
5.1: Creating “knight_rider.v” .....	28
5.2: Debugging Code .....	28
5.3: Running Your Code to Your Board .....	28
5.4: More Debugging .....	29
5.5: Even More Debugging!.....	30
1 Document Revision History .....	32

## 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.
- The board GUI runs for 10 minutes before timing out to allow others access. If you have successfully compiled your code and just needed extra time on the board, open the TCL Console and run the following command:

```
set a [open "|quartus_sh --script ../tcl/main.tcl" r+]
```

- 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, however this lab will run a remote development kit. The development kit will be 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.

## 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

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 20.1 (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.

Design Software  
Embedded Software  
APIs/Tools  
Learning  
Programming Software  
Others  
Board System Design  
Board Layout and Tool  
Legacy Software

### Quartus Prime Lite Edition

Released date: June, 2020  
Latest Release: v20.1

Select edition: Lite  
Select release: 20.1

Operating System: Windows Linux

The Quartus Prime Lite Edition Design Software, Version 20.1 includes functional and security updates. Users should keep their software up-to-date and follow the [Quartus Prime Security Advisory](#) to help improve security. Additional security updates are planned and will be provided as they become available. Users should promptly install the latest version upon release.

The Quartus Prime Lite Edition Design Software, Version 20.1 is subject to removal from the web when support for all devices in this release are available in a newer version, or all devices supported by this version are obsolete. If you would like to receive customer notifications by e-mail, please subscribe to our [Quartus Prime Software Notification Mailing List](#).

The Quartus Prime Lite Edition Design Software, Version 20.1 supports the following devices: Cyclone IV, Cyclone IV LP, Cyclone IV V, MAX 10, MAX 10 FPGA, and MAX 10 FPGA.

Combined Files  
Individual Files  
Additional Software

#### Download and Install Instructions

[Read Intel FPGA Software v20.1 Installation FAQ](#)  
[Quick Start Guide](#)

#### Quartus Prime Lite Edition (Free)

**Quartus Prime (includes Nios II EDS)**  
 Size: 1.4 GB MD5: 23479E4E68328A9C4C8C8C0B8F182  
\*\* Max 3.020 on Windows requires Ubuntu 18.04 LTS or Windows Subsystem for Linux (WSL) which requires a virtual installation. [Release Notes](#) [Release Notes](#)

**ModelSim-Intel FPGA Edition (includes Starter Edition)**  
 Size: 1.2 GB MD5: D8F634F3229F04B880198D80A4C384

#### Devices

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

**Arria II device support**  
 Size: 488.1 MB MD5: A439D54E75E95D22E36ACED3EAB8A8C

**Cyclone IV device support**  
 Size: 494.8 MB MD5: 14E47F19CEA4070C8D86C8C8B7D76488

**Cyclone IV LP device support**  
 Size: 245.7 MB MD5: AF44C8C96038E0584E8E8B21EB1378

**Cyclone V device support**  
 Size: 1.3 GB MD5: 28F6D38171D18E81E881F8A008F9D8F

**MAX 10, MAX 10 V device support**  
 Size: 11.4 MB MD5: 25F442BC8879A4A8E8E8B18E19CC2A

**MAX 10 FPGA device support**  
 Size: 288.4 MB MD5: 10294D71211A02F6D0D0C8C8C8C8C8C8

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

Figure 1: Quartus Prime Lite Minimum Required Files to Download

# LAB 2: UNARCHIVING A BLANK PROJECT

## Summary

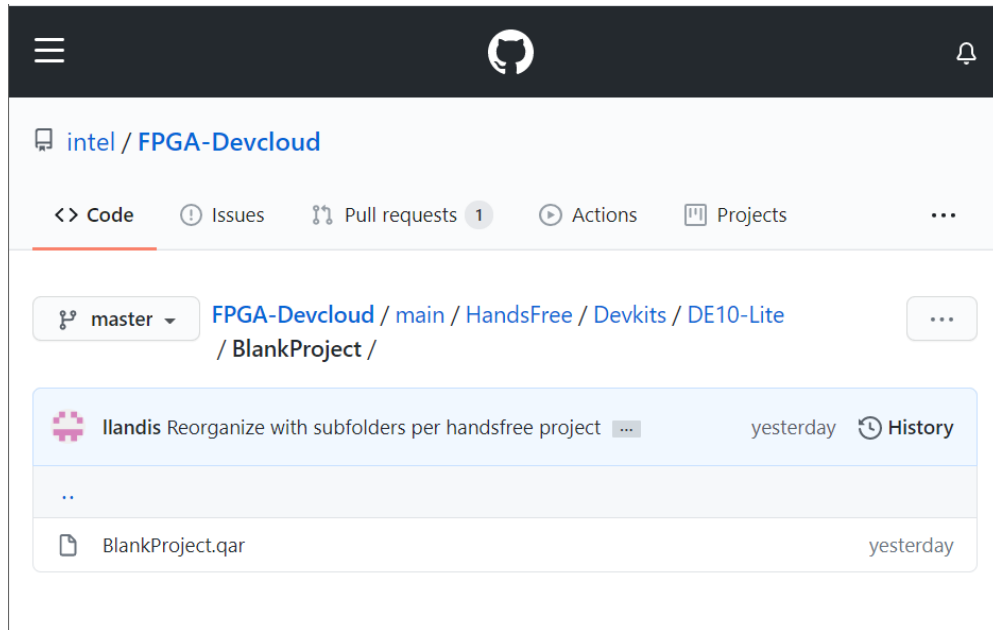
This is a short lab that completes the basic project setup. At the end of this lab, you will be able to start a new project using New Project Wizard in Quartus Prime Software. There are other related tutorial links provided for you to learn more about the software.

## Lab Instruction

### 2.0: Blank Project GitHub Download

- ☐ To begin your Remote Hands-free project, go to the GitHub site:  
<https://github.com/intel/FPGA-Devcloud/tree/master/main/HandsFree/Devkits/DE10-Lite/BlankProject>
- ☐ Download the .qar file BlanketProject.qar and save it to a directory that does not have spaces in the path name! IMPORTANT: No spaces such as C:\Users\llandis\Documents\My First FPGA . You need to make the directory C:\Users\llandis\Documents\My\_First\_FPGA
- ☐ You should see a page that looks like Figure 2 below





*Figure 2: Inside BlankProject folder of FPGA-Devcloud GitHub*

## 2.1: 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.

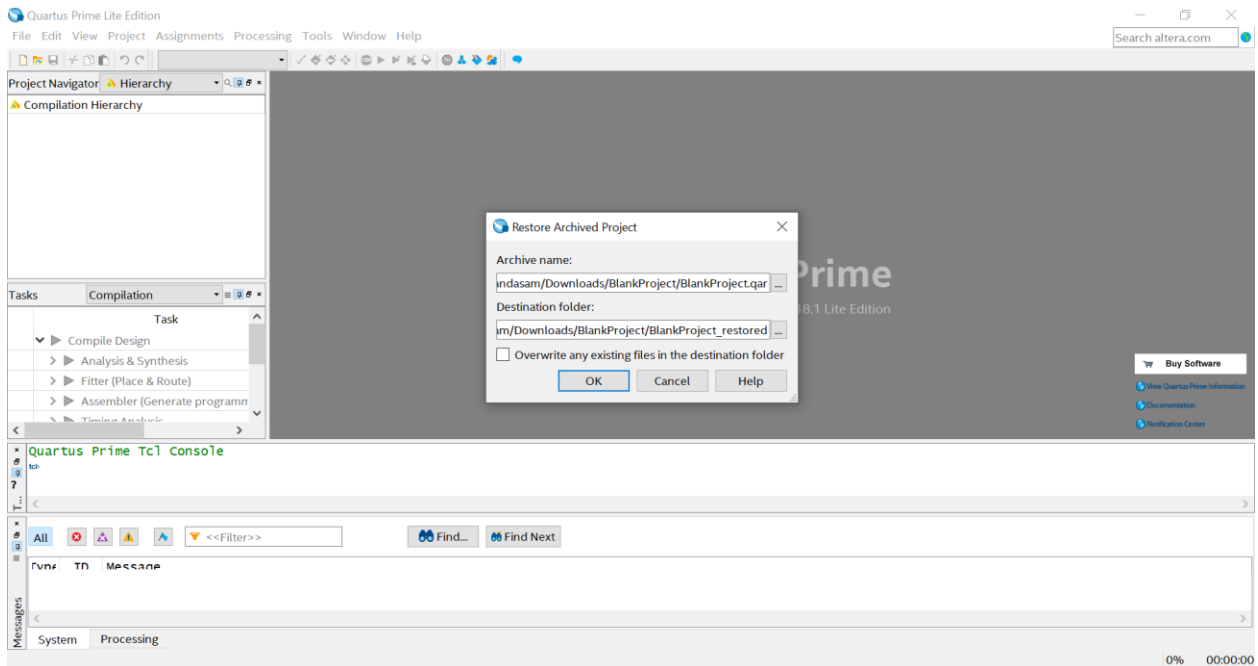


Figure 3: Restore Archived Project for BlankProject

- ☐ When you open the .qar file you will see the message from Figure 3 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 4.

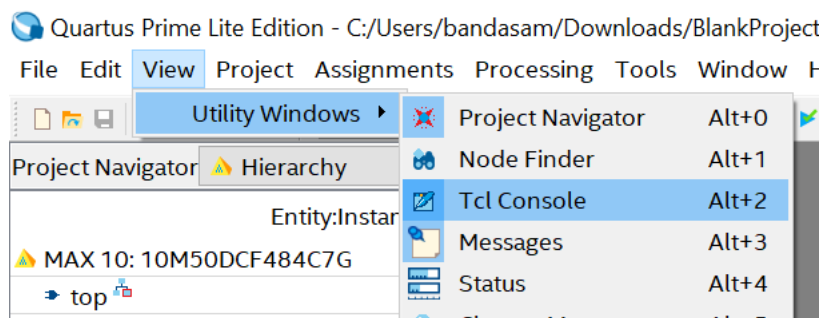


Figure 4: Utility Window dropdown

# LAB 3: DESCRIBING YOUR FPGA FUNCTION USING 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

### 3.0: Creating a New File

- ☐ Create a Verilog HDL file. Go the **File** dropdown menu and select **New**.
- ☐ A window, shown in Figure 5, should pop up. Click on **Verilog HDL File** and then **OK**.

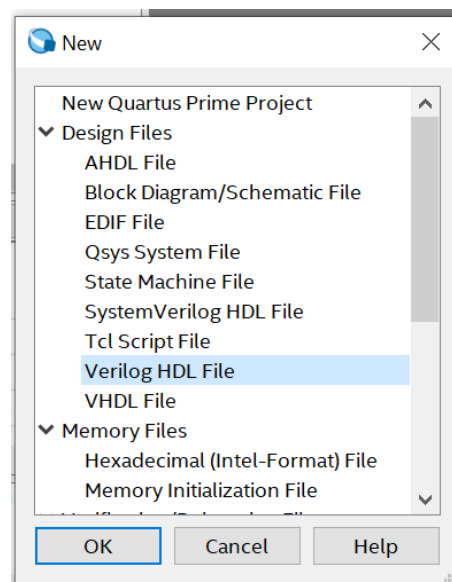


Figure 5: New File Window

### 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.v** (ensuring case-sensitivity), and click **Save As...** to save your Verilog file.
- ☐ Make sure Project Navigator is showing Files. If not use the pull down to select Files as shown in Figure 6.

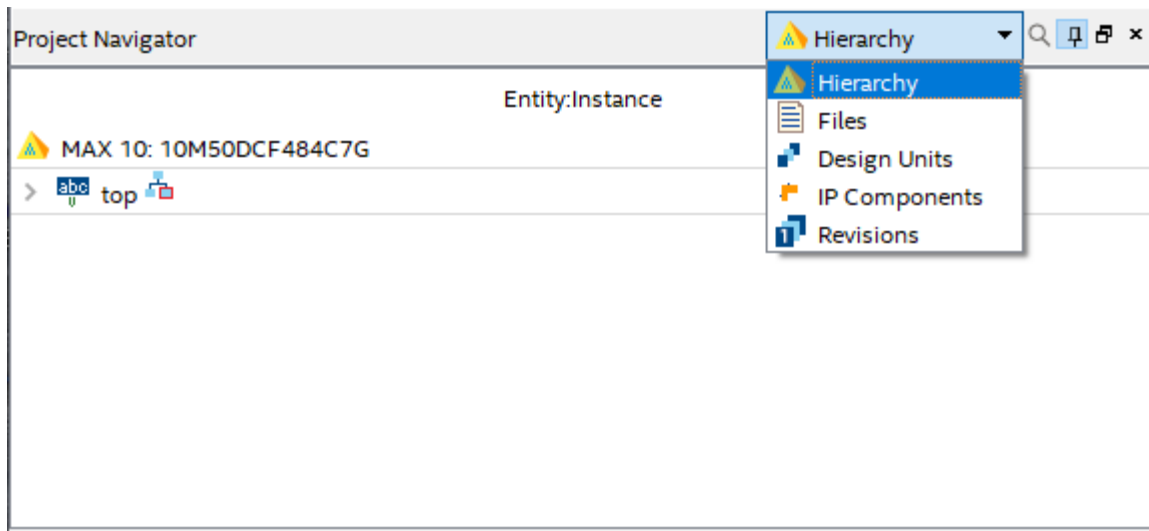
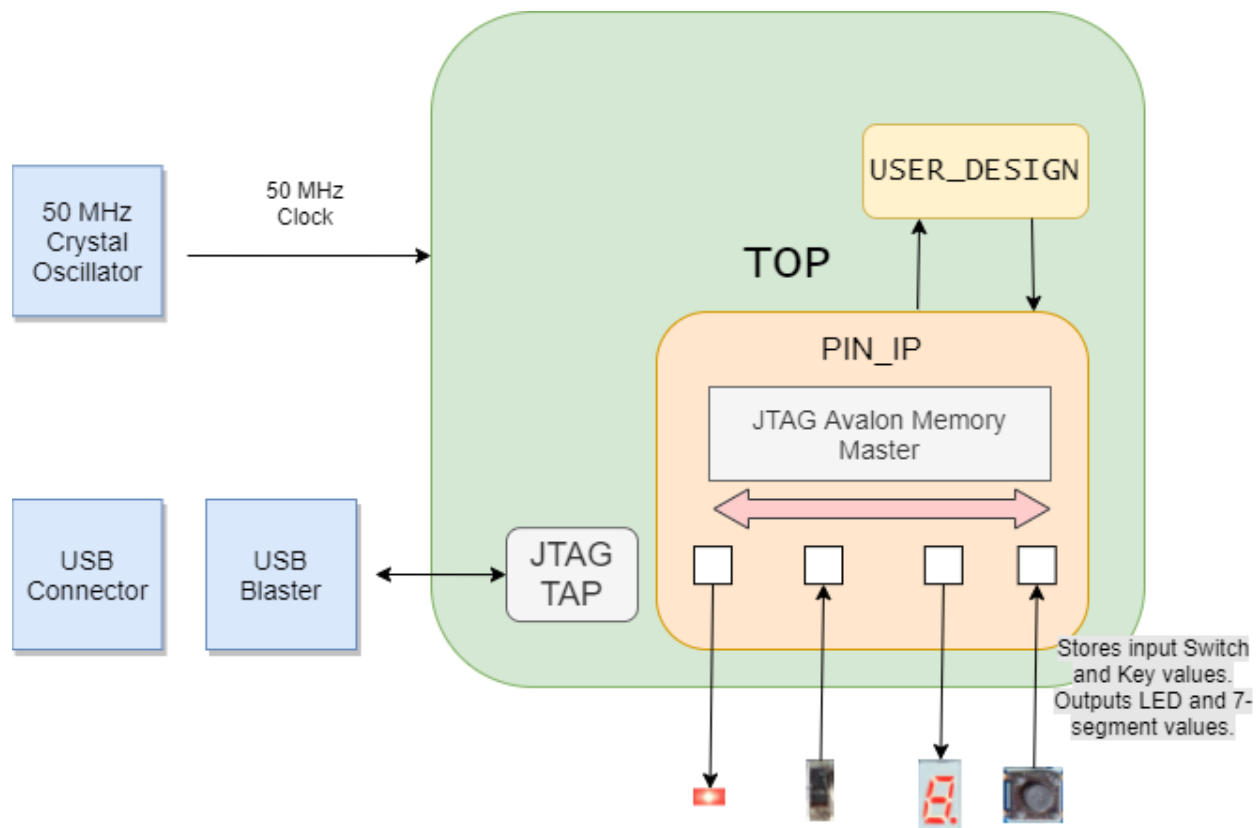


Figure 6: Project Navigator

- ☐ In your project navigator view (upper left): Right click on UserDesign.v and remove file from project. The switch\_to\_led design is what will now become your UserDesign.
- ☐ 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.

### 3.2: 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. 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. Because your DE10-Lite development kit is remotely connected to a server, the pins have already been assigned for you. To understand the process, refer to the diagram on Figure 7 below.



*Figure 7: 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`.
- **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 the LED and seven segment values to run your User Design.

- As seen in Figure 7, 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.

### 3.3: 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 that will direct you to type your user design below. Below that comment you will initiate your design by typing the following code. This is around line 31 in the top.v file that is open in a tab.

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

- ☐ Your top.v file should look like Figure 8 below.

```


15     wire [9:0] SW;
16     // Use these wires as active low, push-button inputs
17     wire [1:0] KEY;
18     // These function as programmable register inputs to a
19     // design, useful for adjusting inputs using a keyboard
20     wire [31:0] param1, param2, param3;
21
22     // User instantiates design below
23
24     /*
25     **
26     ** Instantiated design connects to pin_ip connected wires
27     ** these wires function as memory-mapped i/o which is
28     ** translated to widgets on the GUI
29     **
30     */
31
32     switch_to_led    i_switch_to_led (.SW(SW),.LEDR(LED0));
33
34     // IP to allow simple user design interfacing with development kit
35
36
37
38     pin_ip            platform_designer_pin_ip( .MAX10_CLK1_50(MAX10_CLK1_50
39                                                    .LEDR(LED0),
40                                                    HEX0(HEX0))

```

*Figure 8: Instantiating switch\_to\_led to top*

- ☐ Save your design.

### 3.4: 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)



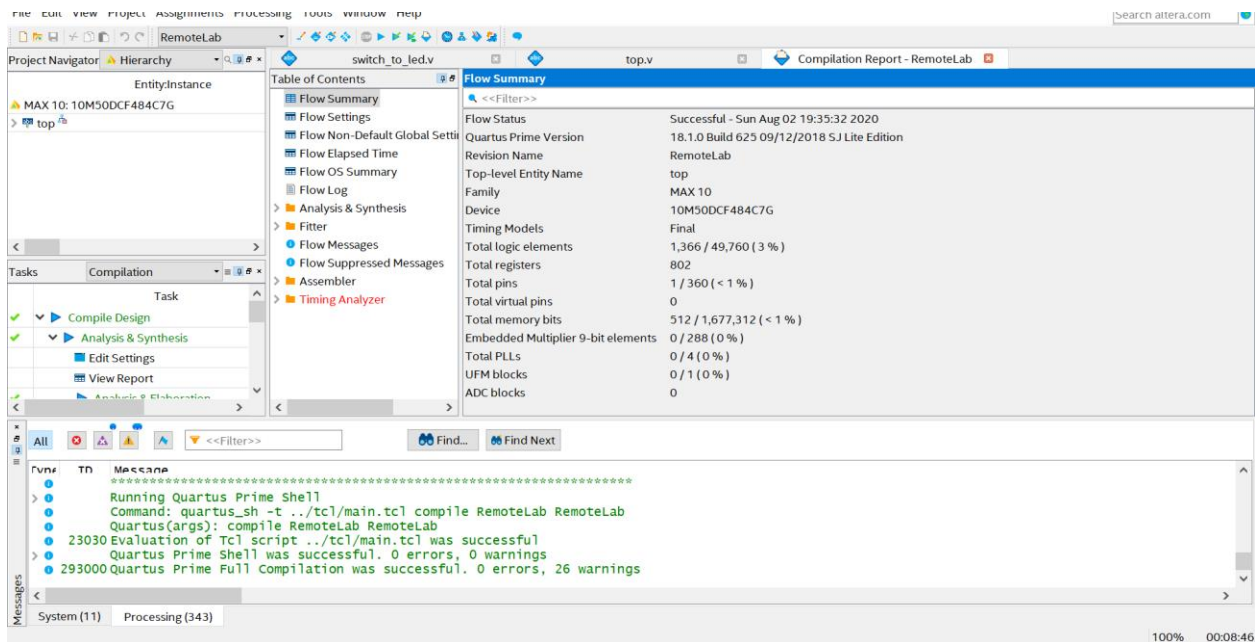
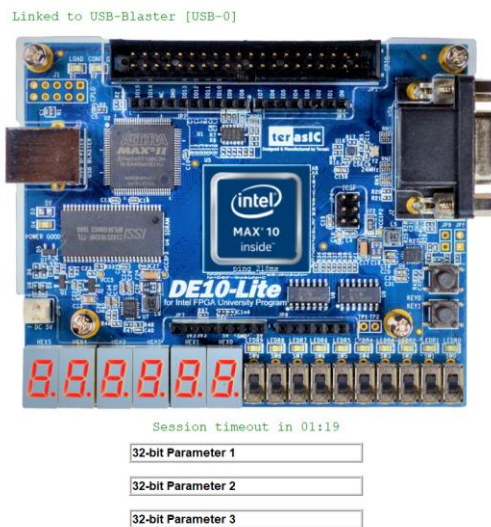


Figure 9: Quartus window showing successful compilation

### 3.5: Programming Remote Board

- ☐ When you have compiled successfully, the remote board should pop up automatically.
- ☐ To see your program run, you will first need to click on Board Select on the top left corner, show in Figure 10. Click on an available board, or you could also select Auto Connect for the system to look for an available board for you.
- ☐ It should take 1-2 minutes for the board to program. Note that your board will automatically exit after two minutes.



*Figure 10: Development Kit Viewer Window*

### 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 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 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.

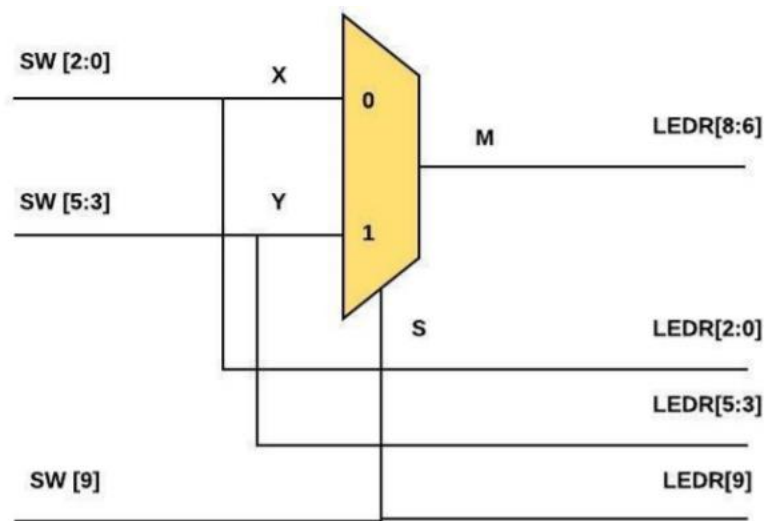


Figure 11: Logic Flow for a 2 to 1 Mux

### Lab Instruction

#### 4.0: 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 3.1 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 in section 4.3 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 4.3

- ☐ 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
```

### 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.

## 4.1: 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.

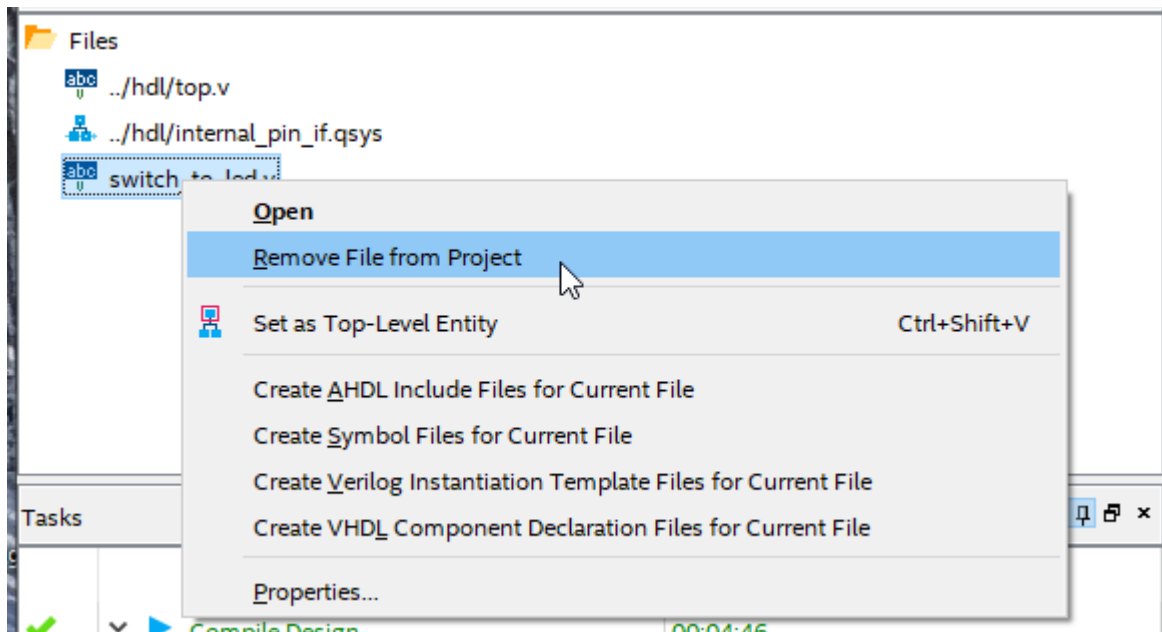



Figure 12: Changing Top Level Entity

- ☐ Right click **switch\_to\_led.v** and remove this source file from your project.
- ☐ Refer back to Section 3.3 and 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));
```

- ☐ 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.

## 4.2: 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 can navigate the design hierarchy and observe your mux\_2\_to\_1 design in a schematic form.

	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

## 4.3: 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 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

```

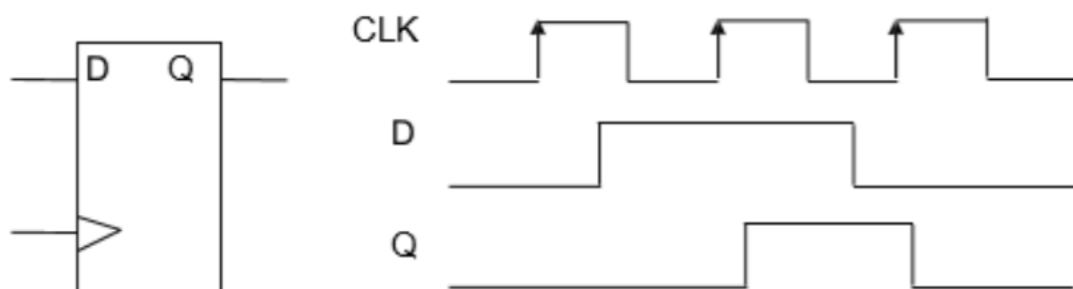


## LAB 5: KNIGHT RIDER

### 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 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.

## 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.

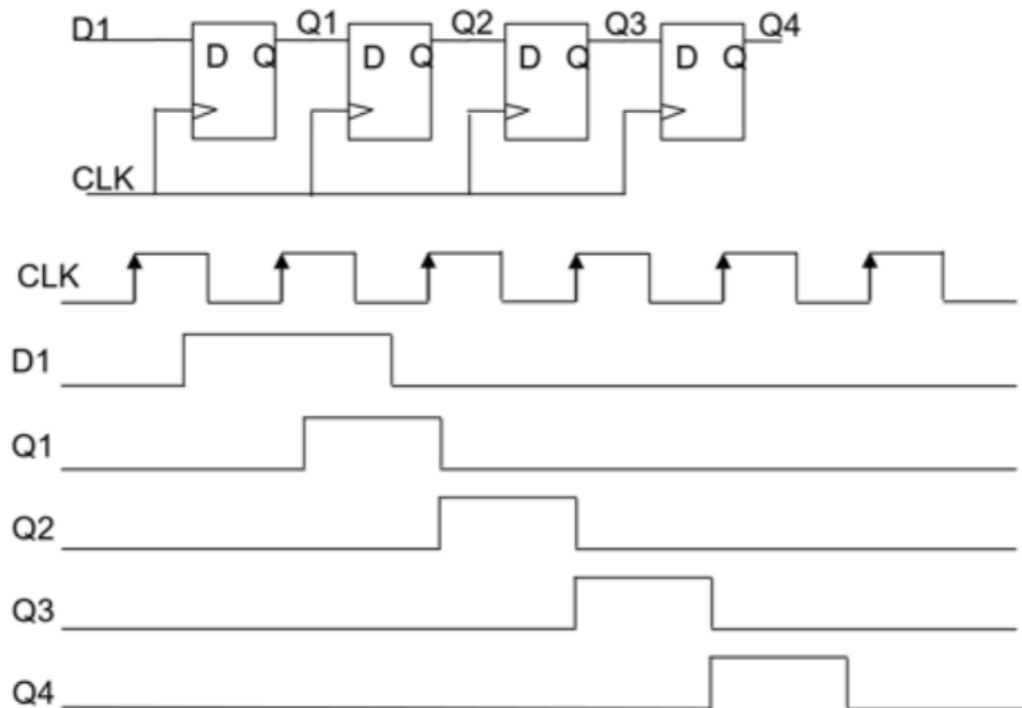


Figure 16: Shift Register diagram

- The code given **intentionally** has errors. See if you can find them all.

```

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,
    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”

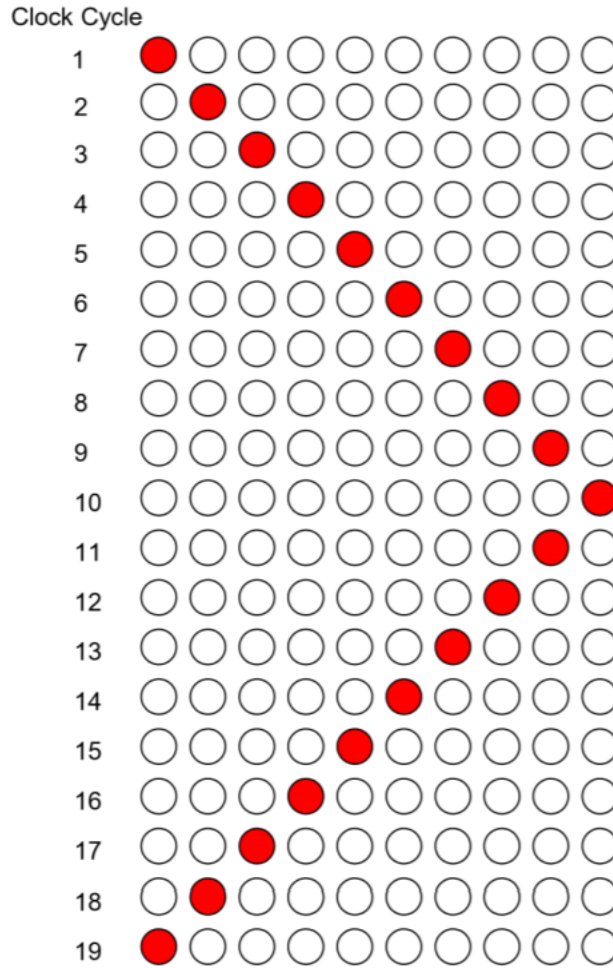
- ☐ Open a new Verilog file and save it as **knight\_rider.v**. Copy and paste the code above into knight\_rider.v.
- ☐ Make sure to instantiate your knight\_rider.v file into Top and comment out any previous files you instantiated.

## 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: Running Your Code to 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. 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 DE10-LITE 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.

- ☐ 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 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.



## 1 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