LAB 3: *Sequential Logic Implementation on Basys3*

OBJECTIVES:

- Implement a BCD counter on the Basys3 board

- Implement a simple traffic light system on the Basys3 board

EQUIPMENTS:

Basys3 Artix-7 FPGA Trainer Board

Vivado software – Current Version 2019.2

INTRODUCTION:

In this experiment, we will first design a single-digit BCD counter. The counter will be displayed on the right most seven segment display at a frequency of 1 Hz. Since the Basys3's master clock is running at 100 MHz, the BCD counter's clock needs to be divided by 100 million. The BCD counter can be reset by pressing the centre push button on the Basys3 as shown in **Figure 1**. When the BCD counter counts to '9', LED0 will flash once. The asynchronous reset is active high, and the BCD counter's clock is triggered on the rising edge or positive going transition (PGT).

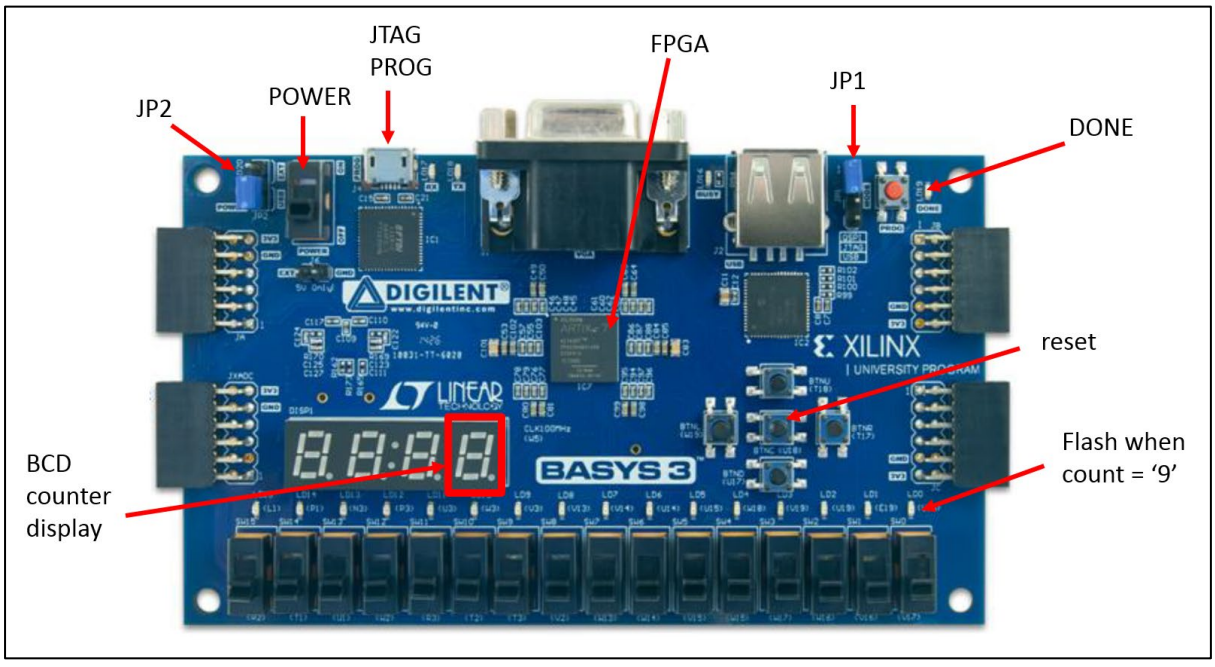

Figure 1: Basys3 Artix-7 FPGA Trainer Board Layout

In the next exercise, a traffic light system at a road junction will be designed and implemented, by using synchronous sequential logic finite state machine (FSM). LEDs are used to indicate green lights for traffic directions (amber and red lights are not implemented in this exercise for simplicity reason). More details will be provided in the procedure section.

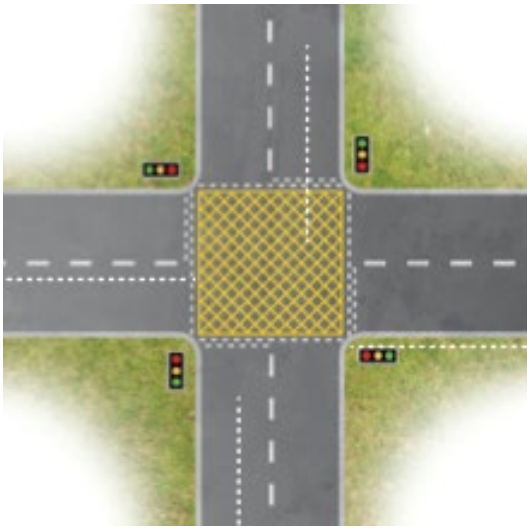

Figure 2: A simple road junction traffic light system

PROCEDURE:

1 Copy relevent files from Blackboard

1.1 Create a new folder **D:/ET0901>Lab>Lab3** in your computer. (If your computer have only one drive, then use that drive instead of D Drive)

1.2 Download the following three files from **Blackboard➔Learning Resources➔Lab** and copy into the new folder you have just created.

- bcdcounter.v
- display.v
- bcd_counter.xdc
- traffic_lights.v
- traffic_lights.xdc

2 Launch Vivado and create bcd_counter project targeting XC7A35TCPG236-1 and using the Verilog HDL

2.1 Open Vivado 2019.2 on your desktop.

2.2 Create a new project named "**lab3_bcd_counter**" under folder **D:/ET0901>Lab>Lab3**. While creating the project, add 2 sources files ***bcdcounter.v*** and ***display.v***. In addition, add constraint file ***bcd_counter.xdc*** into the project. By now you should be comfortable of creating new projects in Vivado, but do refer back to Lab1 lab sheet if in doubt.

2.3 In the *Sources* pane, double-click the **bcdcounter.v** and **display.v** entries to open the file in text mode. Analyze the Verilog source code and understand it. Answer the following questions:

2.3.1 In the *display* module instantiation, the signal _____ in **bcdcounter** module is connected to the *bcd* signal in **display** module.

2.3.2 The signal *bcdcounter_en* in **bcdcounter** module will be logic 1 when _____, otherwise logic 0.

2.3.3 The signal *counter_100M* is defined as a ____-bit register. The reason to have this number of bits is _____.

2.3.4 In the always blocks in **bcdcounter** module, the *reset* signal is Synchronous/Asynchronous to the clock. (Circle the correct choice)

2.3.5 In **display** module, what will happen if the *default* statement is taken out?

_____

_____

2.4 In the *Sources* pane, expand the *Constraints* folder and double-click the **bcd_counter.xdc** entry to open the file in text mode. Analyze the constraint file and understand it.

2.5 Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**. Click OK in the *Elaborated Design* dialog box. The design will be elaborated, and a block diagram of the design is displayed.

3 Simulate bcd_counter Design using the Vivado Simulator

3.1 In this experiment, we may skip the simulation as it is difficult to see the divide by 100M clock. However, simulation is still possible if "*max_count*" parameter value is changed to 99 with a simulation runtime of 15,000 ns. By applying the above value change, the effective frequency of the counter is 1 MHz, instead of 1 Hz. Nonetheless, the main purpose of the simulation is to validate the design logic.

3.2 Add simulation testbench file *bcd_counter_tb.v*, run the simulation and observe the signal waveforms in the simulator result. The desired waveform window is shown below in **Figure 3**. Observe that seg7_cat[6:0] signal has a total of 10 different pattern values, representing '0' to '9' in decimal.

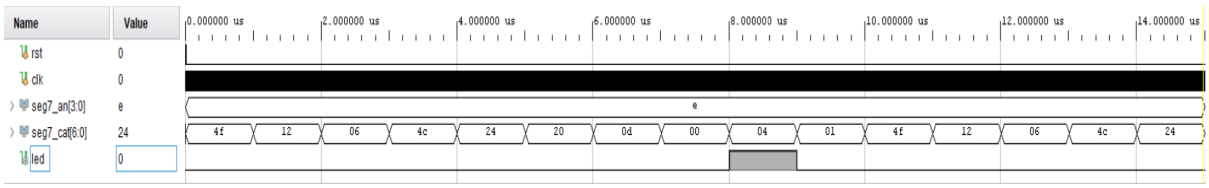

Figure 3: BCD counter simulator result

4 Synthesize, Implement, and Generate Bitstream for the bcd_counter Design

4.1 Change back the "*max_count*" parameter value to 99_999_999.

4.2 Follow the FPGA design flow process to complete Synthesis, Implementation, and Bitstream generation. By now you should be comfortable of the design flow in Vivado, but do refer back to Lab1 lab sheet if in doubt.

4.3 Open Hardware Manager and program your FPGA with the generated bitstream.

4.4 Verify the functionality by checking that the BCD counter displays the correct count number on the right most seven segment display, the LED0 flash once when the display is '9' and the BCD counter is reset to '0' when the center push button is pressed.

4.5 When satisfied, close the hardware session by selecting **File > Close Hardware Manager.** Click **OK** to close the session.

4.6 Close the BCD counter project by selecting **File > Close Project**. Click **OK** to close the project and get ready for the next project creation.

5 Design and Implement a traffic light system for a road junction

5.1 In this section, we will design a simple traffic light system for a road junction, using synchronous sequential logic with finite state machine. Assuming a cross road junction as shown in **Figure 4**, there are four directions namely North, South, West, and East. Each direction allows 3 lanes of vehicles to pass through. At the junction, vehicles in the left, middle, and right lanes must turn left, go straight, and turn right respectively, according to the traffic light system signals. In this simplified system, only green lights are used to indicate whether a particular turning is permitted, whereas amber or red lights are not utilized.

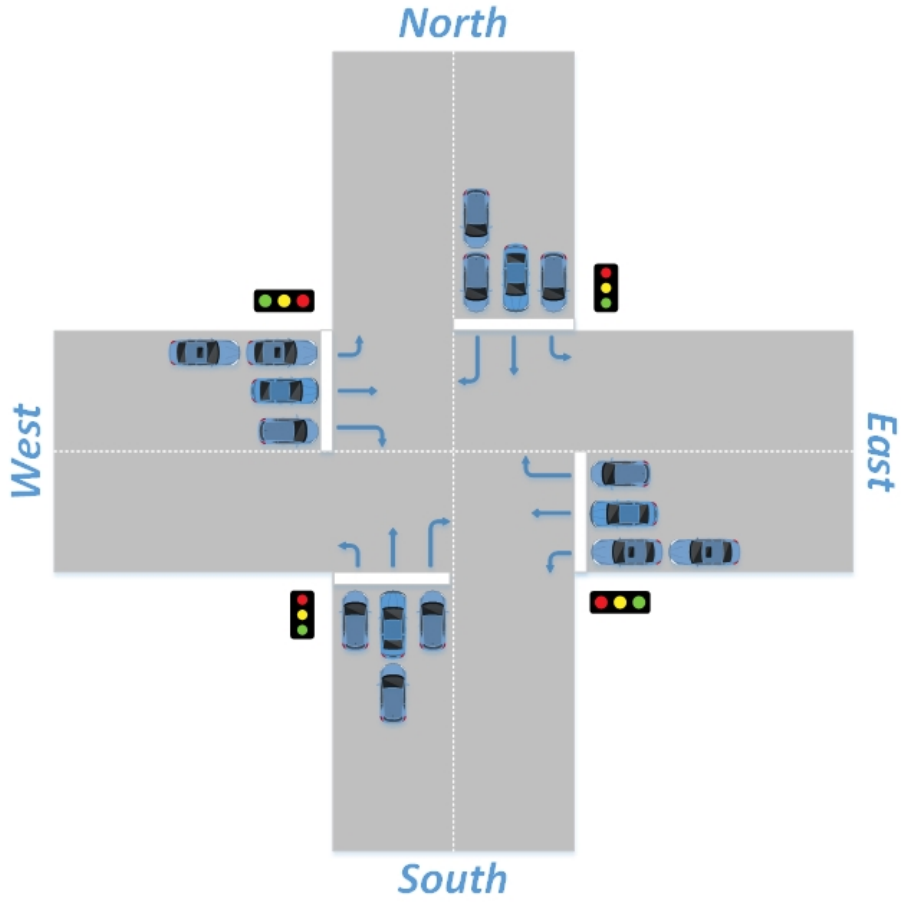

Figure 4: A Simple Cross Road Junction

In this digital system, 4 states shall be designed such that each state will turn ON a set of LEDs to allow vehicles travelling towards certain directions to pass through. **Figure 5** shows the detailed state conditions and state transition sequence, together with the summary below.

- **State_1**
    - From North : Left + Straight
    - From South : Left + Straight

- **State_2**
    - From North : Right
    - From South : Right
- **State_3**
    - From West : Left + Straight + Right
- **State_4**
    - From East : Left + Straight + Right

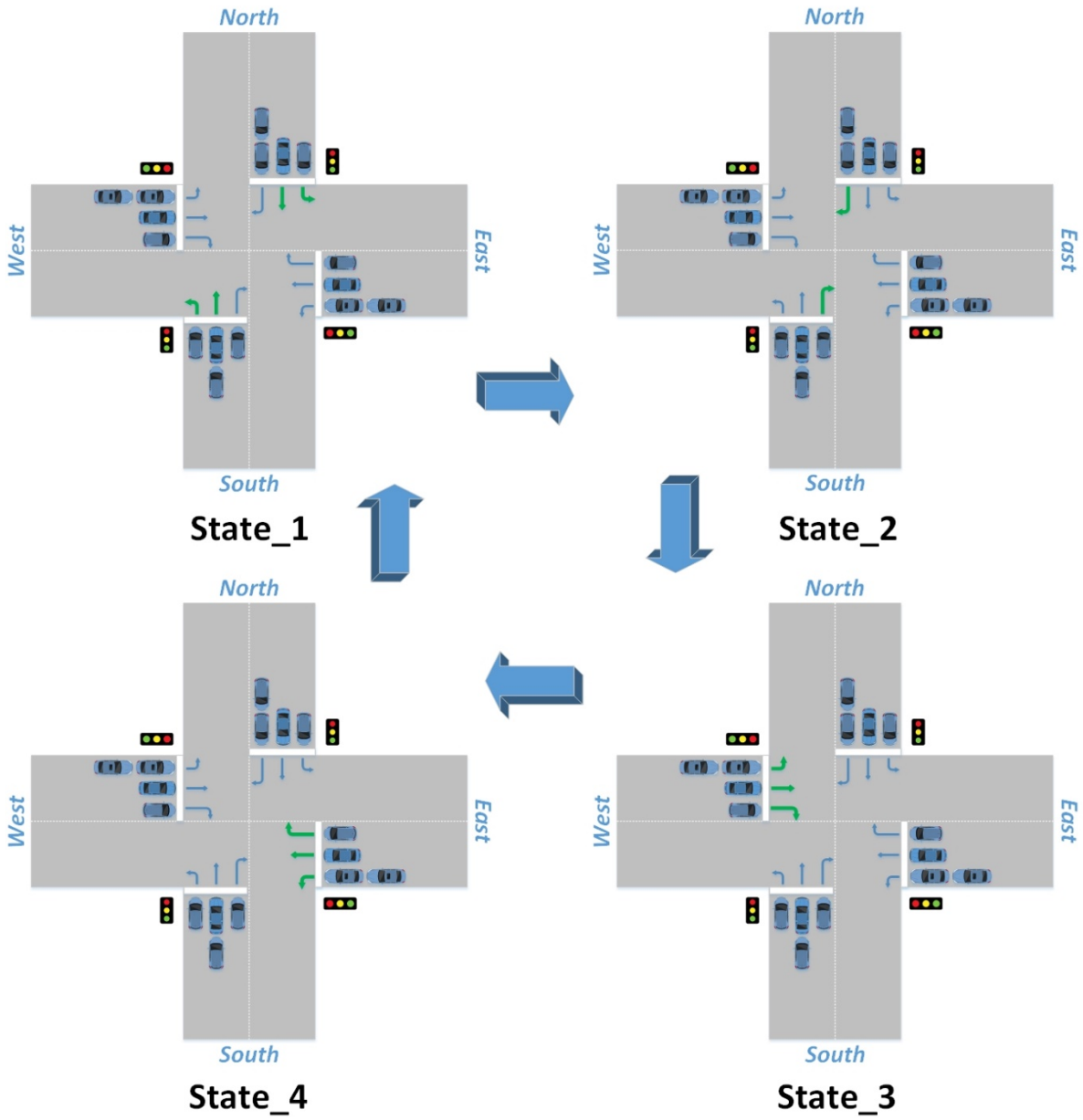

Figure 5: Block Diagram for state conditions and state transition sequence

The triggering event for the state machine to transit from one state to the next state is raised by a 1-bit signal named "*trig*". This "*trig*" signal is a 1-clock cycle pulse generated **every 5 seconds**. Whenever the state machines detects a rising edge (or positive going transition) of Signal "*trig*", the current state will transit to the next state at the next clock edge, e.g. from State_2 to State_3. The Moore Model state diagram is shown in **Figure 6**. The LED outputs that indicate all the traffic lights are summarized in **Table 1** below.

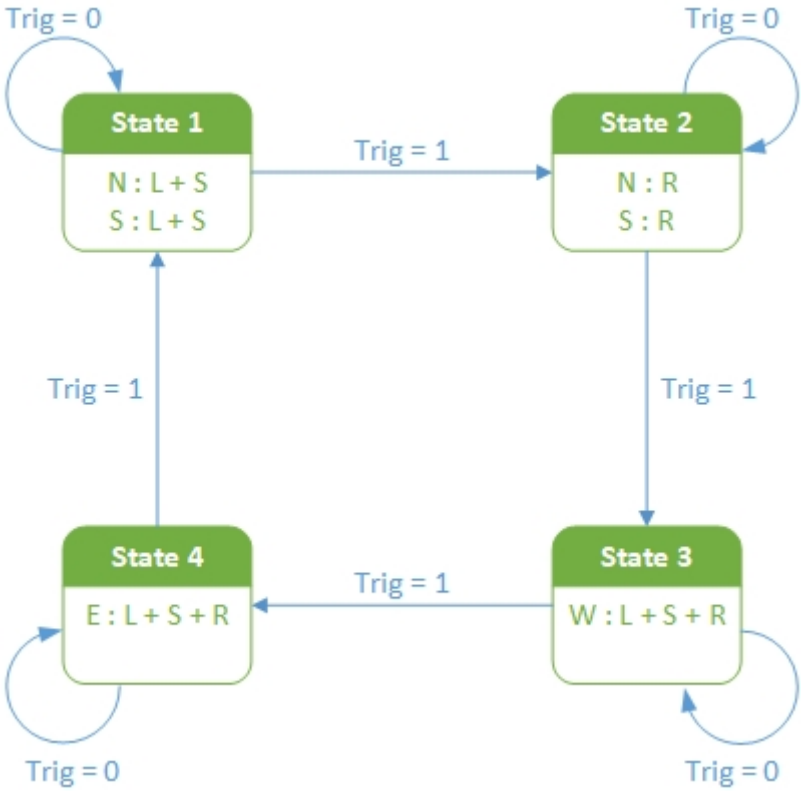

Figure 6: Traffic Light System State Diagram

Table 1: Traffic Lights LED Assignment

	North	**West**	**South**	**East**
Left Turn	led[15]	led[11]	led[7]	led[3]
Straight	led[14]	led[10]	led[6]	led[2]
Right Turn	led[13]	led[9]	led[5]	led[1]

5.2 Create a new project named "traffic_lights" in Vivado. Add Verilog source code file "*traffic_lights.v*" and constraint file "*traffic_lights.xdc*" into the project.

5.3 Complete synthesis, implementation, bitstream generation, and eventually program the FPGA on the Basys3 board.

5.4 Verify the functionality by checking that all 4 states exist and state transition happens every 5 seconds. Verify that all LEDs are are lighting up in correct sequence.

5.5 **Extra Practice 1:** *Traffic Centre Override Control*

Stop the 5 seconds auto state transition sequence. Instead, Use the **centre push button btnC** to control traffic lights changing sequence. That is, State machine proceeds to the next state once button is pressed. Please note that one press should lead to one state change only.

Modify the source code and constraint file to achieve this function. Verify the function on the Basys3 board.

Hint: Generate a 1-clock pulse for "trig" for one press of btnC button. Un-comment btnC pin assignment in the XDC constraint file.

5.6 **Extra Practice 2**: *Add **Pedestrian Crossing signals** in the system*

Use **4 up-down switches** to indicate pedestrian-crossing intention. The 4 switches and green-man LEDs for pedestrians are summarized in **Table 2**.

Table 2: Pedestrian Crossing Switches and LEDs Assignment

	North	**West**	**South**	**East**
Pedestrian Switch	sw[12]	sw[8]	sw[4]	sw[0]
Green-man LED	led[12]	led[8]	led[4]	led[0]

Modify the source code and constraint file to achieve this function. Verify the function on the Basys3 board.

Hint: Add "if/else" statements in the output control "always" block in the source code. Un-comment relevant switches and LEDs in the XDC constraint file.

5.7 When satisfied, power **OFF** the board. Close the hardware session by selecting **File > Close Hardware Manager.** Click **OK** to close the session.

5.8 Close the **Vivado** program by selecting **File > Exit** and click **OK**.

---------- ***END OF THIS LAB*** ----------