Introduction

Traditionally, the debugging of digital circuits was accomplished using benchtop logic analyzers that were connected to digital circuits through the physical I/O pins of an FPGA or an ASIC. If internal signals needed to be monitored, these signals had to be assigned to I/O pins, which could increase the cost and complexity of the design. However, modern FPGAs and ASICs provide an alternative way of debugging which excludes the use of benchtop logic analyzers. By incorporating specific circuits into the design which can monitor the rest of the system and by sending information from these circuits to a computer, we can debug complex designs with few additional resources. In Xilinx, ChipScope Pro coupled with the Integrated Controller (ICON) core and one or more Integrated Logic Analyzers (ILAs) cores can be used to sample and forward any signals in a design to a computer for further analysis.
In this lab, your goals are to:

1. Learn how to use ChipScope to debug general digital designs
2. Use ChipScope to debug digital designs that incorporate state machines
3. Use ChipScope to debug digital designs that incorporate Opal Kelly functionality

Prelab Questions (10 points):

- What is the purpose of the ICON module? How many usable ICON modules can be instantiated in a design?
- What is the purpose of the ILA module? How many usable ILA modules can be instantiated in a design? What is the maximum number of signals that an ILA module can monitor in a design?
- What does “Sample Data Depth” parameter specify in settings for the ILA module? (See Slide 15 in the online slides to find this parameter.)
- What are the four signals in the interface to the ILA module that is instantiated in the top module used for? (See Slide 19 in the online slides to find these signals.)

ChipScope

To aid developers using FPGAs, Xilinx offers a pair of tools that provide full hardware debugging capabilities from a convenient software interface. By inserting an “Integrated CONtroller” (ICON) core along with an “Integrated Logic Analyzer” (ILA) core into a design and connecting a handful of control signals, a developer can create circuits that capture signals in real-time, and by using ChipScope Pro which is distributed with Xilinx Vivado, the same developer can configure triggers for the controlled capture of signals and can view those signals using a flexible waveform viewer.

To see how to use both of these tools, follow all of the steps provided in the ChipScope Pro tutorial that has been posted to the course webpage. At the end of the tutorial, you should be able to demonstrate a working counter example using ChipScope Pro.

Checkpoint 1

Go through the ChipScope Pro tutorial that is posted on the course website. At the end of the tutorial, you will be simulating the behavior of the counter using Vivado. Demonstrate a working counter example using ChipScope Pro to the TA.
Debugging State Machines with ChipScope Pro

To develop debugging skills that will be essential when writing control modules for sensors seen later in this course, you will consider a design example that will require careful execution and testing of a state machine. To this end, you will design a state machine for the traffic lights at a simple four-way intersection with pedestrian crosswalks. Here is the problem description:

- The intersection that you are controlling is composed of two sets of lights for cars and one set of lights for pedestrians. The car lights control vehicle movements in either the east-west direction or the north-south, while the pedestrian lights control movement for pedestrians moving in all directions.

- There are eight lights to operate. The red, amber, and green lights in the north-south direction will be designated as R1, A1, G1. Similarly, the lights in the east-west direction will be called R2, A2, and G2. There are red and green lights for the pedestrian, and you can name them R3 and G3.

- The car lights should cycle through their state such that the green light is on for 3 seconds, the amber light is on for 1 second, and the red light is on for at least 5 seconds at a time.

- If a pedestrian pushes a button, then the pedestrian green light will be enabled only after the car light that is currently in either the green state or the amber state finishes its state sequence and becomes red. In other words, the push button will not immediately turn the car lights to red but will rather wait until the car lights eventually return to red.

- The pedestrian green light will be on for 3 seconds. After this sequence, the car lights have to continue from where they left off before the button was pressed. If the north-south signal was green before the pedestrian light was on, then after the pedestrian light has gone through its green state to its red state, the east-west signal should turn to green.
Ultimately, you will want to be able to debug and demonstrate this design with Chipscope Pro, so some thought has to be put into how the ICON core and the ILA core will be integrated with your design (to deal with the limitations of the modules that you saw in Checkpoint 1). Early on, you should make a conscious decision about how to approach this by either (1) designing your circuit so that it can be easily reconfigured for debugging and demonstration purposes or (2) implementing your debugging functionality so that it is compatible with your demonstration functionality.

**Checkpoint 2**

You need to design and implement the state machine for the traffic light controller that is described above. You should use the LEDs to indicate the car and pedestrian light signals, and you should use ChipScope Pro to demonstrate that the state machine is transitioning properly. Show all necessary waveforms in ChipScope Pro to indicate correct transitions through all states.

Demonstrate the correct behavior of the state machine to the TA via both the LEDs and ChipScope Pro. In your lab report, you should include a state diagram for the finite state machine that you developed, and you should explain how the finite state machine cooperates with the components that are synthesized by your Verilog to provide the correct behavior.