JTAG DEBUGGING TUTORIAL (DRAFT VERSION)

ECE 437 - Sensors and Instrumentation
University of Illinois at Urbana-Champaign
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 Vivado, this process is simplified.
In lab, we will be using a Xilinx Platform Cable USB II programmer/debugger along with the Opal Kelly XEM7310 integration module, but these steps should be similar for other programmers or integration modules. Before you begin, make sure that you hook up the programmer/debugger to both the JTAG port on the integration module and a USB port on a computer.
Why do we want to use this?

- Simulation results can fool you
  Something works in simulation may not be synthesizable. This debugger help you prevent this kind of errors.

- Painful experiences when you took ECE 385
  Sometimes you really don’t know why your code doesn’t work after you run your code. You really want something like GDB in C++ to tell you what is the value of a certain variable. This debugger is a hardware version of GDB.

- It can really save your lab
In this tutorial, we will consider a simple example of an 8-bit counter with reset functionality and an overflow indicator. The tutorial will specifically reference the code provided above that was initially written without any intention of debugging. We plan to use button as triggers for sampling real-time data and the count as the data we want to monitor. Hint: you can copy these codes from the Notes below.
The first step is to add the ILA (Integrated Logic Analyzer) core to your project. Start by clicking the IP Catalog under the PROJECT MANAGER.
Under the IP Catalog tab, click Debug & Verification and then Debug. Double-click the ILA (Integrated Logic Analyzer).
A window pops up which allows you to configure certain parameters associated with this IP. In this example, the name for the component is ila. Under the **General Options Panel**, make sure you choose **Native** as the monitor type.
You can choose any number from 1 to 256 as the **Number of Probes**, but remember that the total number of bits (sum of all probe ports) cannot exceed 65,536 bits. We choose 2 as the number of probes in this example, because we want one probe to be a data probe and another to be a trigger probe.

We set the **sample data length** equal to 1024, which means that the debugger will give us 1024 samples whenever we want real-time waveform. You can increase the length if you want to.

The **Number of Comparators** should have a value at least 2 when you choose **Capture Control**. If you do not want to use capture control, set the **Number of Comparators** to be 1.

**Trigger In Port** is optional. It will generate a 1-bit-input port called `trig_in` and an output port called `trig_in_ack` for the module.

**Capture Control** is optional. It will make your process of debugging easier.
Now go to **Probe Port Panels**. We can set the width of each probe and the type of probe here. We set the width of **PROBE0** to be 32, which means that we can monitor 32-bit data at the same time. You can increase this number if you need to monitor more data. We also have a 4-bit **PROBE1** as triggers. Set **PROBE2** and **PROBE3** as shown. We will compare the setting here later. Although you can choose the type of a probe to be **DATA AND TRIGGER**, we recommend that you separate data and trigger into two probes.
Status messages will be printed to the console to let you know if the task is completed. Your source panel will look like the screenshot on the left. Double click ila ila.v, scroll down, and you will get something like the screenshot on the right. It tells you how to instantiate the ila module.
Instantiate the ila module in your `top` module. Your top module will now look like this screenshot.
Now, connect the debugger to the board and to PCs as well. The status light on the debugger will become green. Click **Open Hardware Manager**. Hardware panel will appear. Click auto connect, which is encompassed by a red rectangle.
The Hardware panel will look like the screenshot on the left after you press **auto connect**. The next step is to generate the bitstream file and to use **Program Device** to load the bitstream file to the board. A **hw ila_1** should appear below **XADC** under the **Hardware panel**. If it does not appear, try **Refresh Device**. If it still does not appear, load the bitstream file again. From now on, you don’t need to load your bitstream with FrontPanel. Please load the bitstream file with **Program Device**.
A Waveform panel and other setup will appear.
Click **Window**, then **Debug Probes**. You can add any signals which you want to see but do not appear in the **Waveform panel** there. Right click the signal, **overflow** in this example, and choose **Add Probes to Waveform**. This signal will now appear in the **Waveform panel**.
Now let’s compare the setting we did before. *Overflow* is a **DATA** probe, so you can only choose **Add Probes to Waveform** button[3:2] is for **PROBE1** which is a **TRIGGER** probe. As you can expected, you cannot add this probe to waveform. **PROBE2**, however, has two options for you
In **Setting**, choose the **Trigger mode** to be **BASIC_ONLY** and the **Capture mode** to be **ALWAYS**. In **Trigger Setup**, press the + button to add rules for triggering.
Let’s choose the 4-bit button for now and make the setup like the screenshot in the middle. This setting means that the debugger will begin sampling the real-time waveform when the button value equal to 1101 as binary, that is when you press the button. You can also choose other operators.
Now, press the **Play** button in the **Waveform** panel. You will see the ila is waiting for trigger. The trigger is armed. If you want to stop the trigger, you can just press the red solid square.
Press button 1, which activates the trigger. You will get the waveform shown above. This waveform is the real-time data receiving from FPGA. You can now show the results to TA and start doing the rest of the lab. You can also keep reading if you want to know a bit more.
Now, let’s change the **Trigger mode** to be **TRIG_IN_ONLY**. Recall that we use button[0] as our trig_in value. The debugger will start sampling when this value changes from **low to high**. So, let’s hold the button0 first and then press the **play** button. When you release the button0, you will see a similar waveform you got before.
Recall that we enable the **Capture Control** in ILA. In **Setting**, choose the **Trigger mode** to be **BASIC_ONLY** and the **Capture mode** to be **Basic**. Set the **Capture Setup** like the screenshot shown above. Now, arm the trigger and press button1.
In **Status**, you can see total samples increase as you press button3 and button1 back and forth.
After the debugger captured 1024 samples, a waveform different from what we previously got appear. It actually divides the total 1024 samples available into four parts. Every two parts are separated with a red line. The left most window has 256 samples after you first pressed button1. The next 256 samples were captured after you pressed button3...
Remember to show your waveform in slide 22 to your TA