

Lab 19: Hardware Debugging

In this lab we'll get familiar with the Integrated Logic Analyzer (ILA) and Virtual Input/Output (VIO) IPs from Xilinx.

We will use the stopwatch design, we introduced in Lab 15. As a base.

Go to `~/labs/lab19/` and open the project `lab19.xpr` with Vivado.

```
vivado lab19.xpr &
```

Exercise 0. Implementing the design

Before starting the actual exercise, run the implementation, and take note of the resource usage.

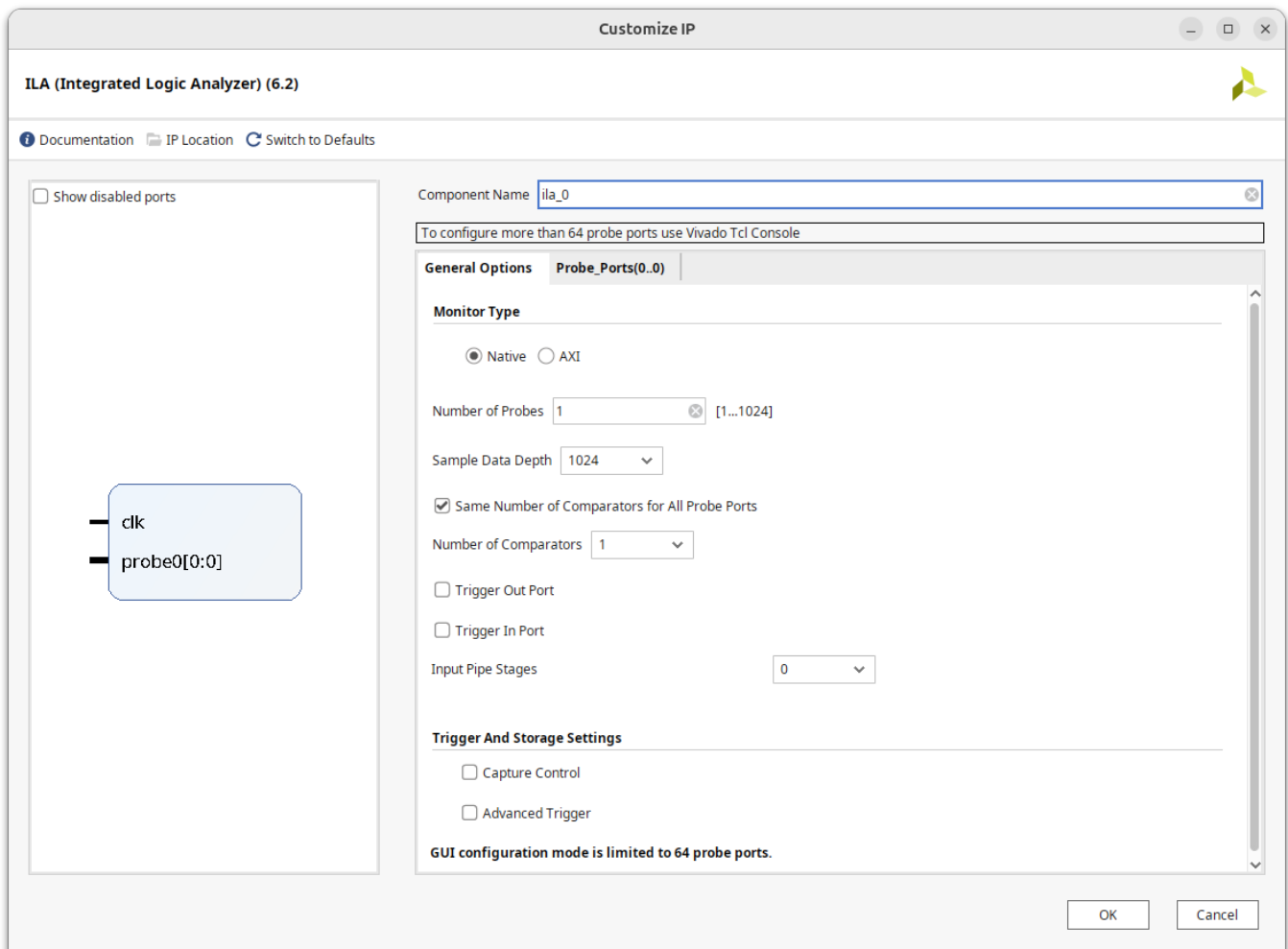
Reminder: To get the resource utilization, open the implemented design and click on *Report Utilization* in the sidebar.

Exercise 1. Use the ILA

In the first exercise, we want to monitor the signals coming in and out of the `display_controller` module. These are:

Signal	Type	Width
digits_vec(0)	std_logic_vector	4
digits_vec(1)	std_logic_vector	4
digits_vec(2)	std_logic_vector	4
digits_vec(3)	std_logic_vector	4
RST	std_logic	1
START	std_logic	1
STOP	std_logic	1
seg_anodes	std_logic_vector	4
seg_cathodes	std_logic_vector	8

Click on IP Catalog, on the left side bar, search for ILA and create a new ILA IP.



In the **General Options** tab, write the correct number of probes in the corresponding box. Keep the default for the other settings and go to the *Probe Ports* tab(s).

Modify the Probes' widths, to accommodate for the width of the signals that we want to monitor. Keep the default for the other properties and click OK, once you are done.

Generate the IP, and wait until the output products are produced.

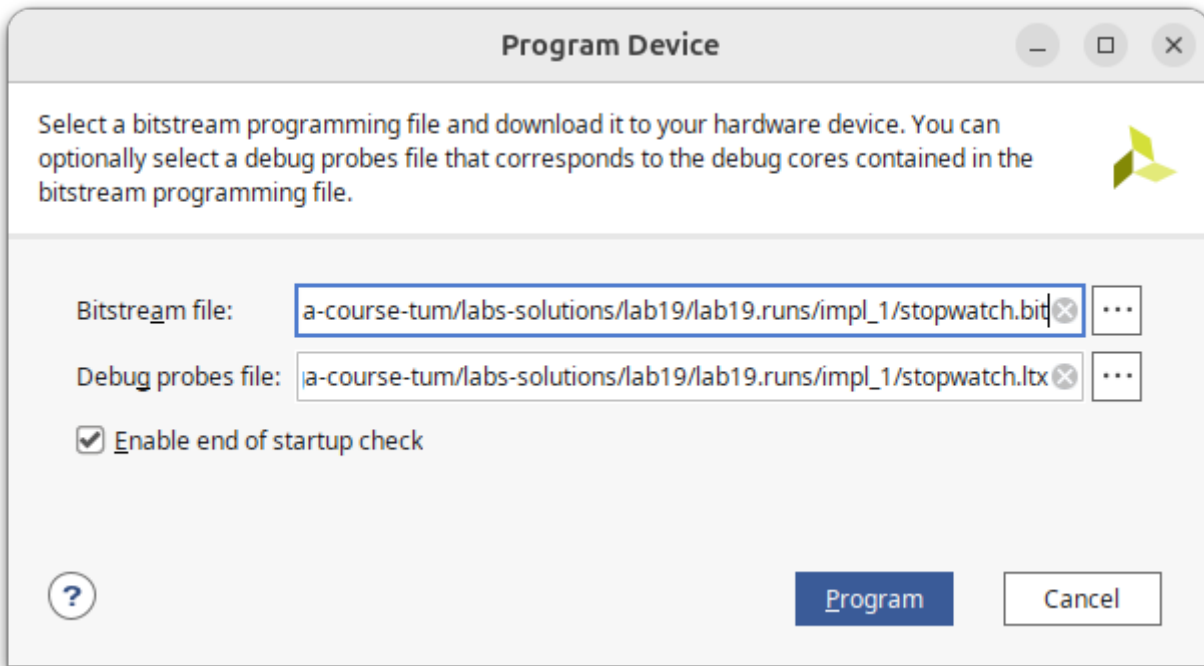
Now, instantiate the ILA IPs in your code inside the `stopwatch.vhd` file, as usual, and map its port to the correct signals to monitor.

Tip: You can use the `ToSLV` function, declared in `display_pkg.vhd` to convert a `std_logic` into a `std_logic_vector`.

Once you are done, run the implementation, and check again the resource utilization. What do you notice?

Generate now the bitstream, and load it to the board.

In the Program Device window, you should now see also the *Debug Probe file* box filled.



Once you program it you will be prompted with the following screen.



Here, we have a waveform viewer, similar to the one we used in simulation, and on the bottom pad, the controller for our trigger signals.

The ILA, will capture the data, once you click on the play button in the bottom left pad (*Status*). By default, this will sample the signals at the moment the ILA receives the input you just gave it.

Alternatively, we can make the ILA capture the data, using a trigger signal.

On the right bottom pad *Trigger Setup*, click on the plus button. Here you can select one of the signal in the ILA as a trigger. Let's select the START signal for instance.

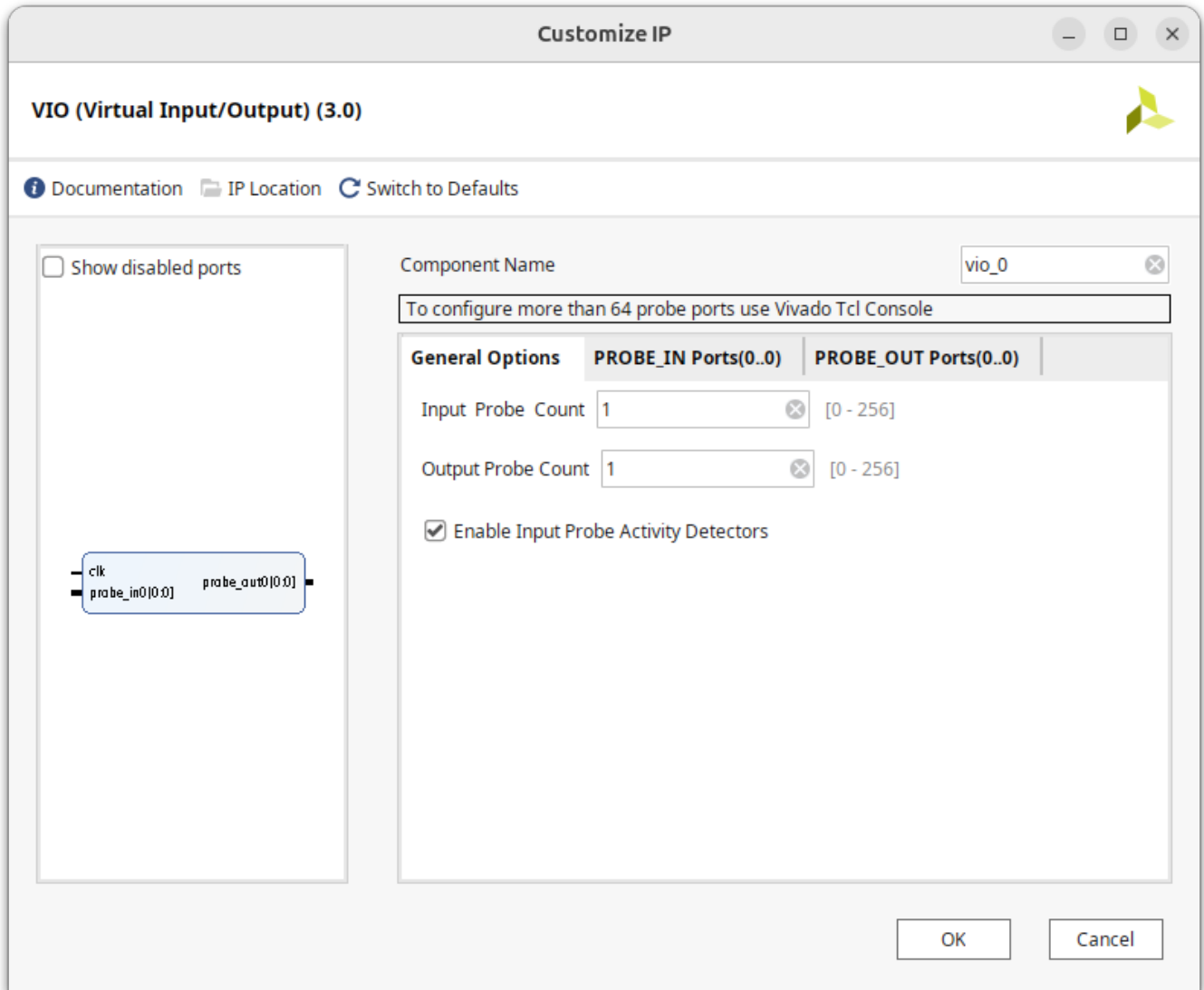
It should then appear in the box. Click in the *Value Cell*, and change its value to *B (Both Transitions)*. This means that our trigger will run every time there is any transition on the START signal.

In the Status box on the left you can activate the trigger by clicking on the Play button . You can also keep the trigger active indefinitely clicking on the re-trigger button .

Click on Play. You should see that the status changes now to *Waiting for Trigger*. Push the *START* button on the board, so the trigger will be enabled, and the waveform captured.

Exercise 2. Use the VIO

Click again in *IP Catalog*, and search VIO, and double click on the IP. You will be prompted with the configuration screen.



The VIO will allow us to drive signals directly from the hardware manager.

With this VIO, we want to manage the START, STOP and RESET signals directly, and monitor the four digits.

Therefore set the number of input probes to 4 and output probes to 3. In the *PROBE IN* tab, set the width of each probes to 4. In the *PROBE OUT*, check that the widths of all probes is 1.

Click OK, and generate the output products.

Open now the `stopwatch.vhd` file, and transform the RST, START, STOP ports to internal signals in the architecture declaration part.

Instantiate the VIO, as usual, and map the ports. The input ports of the VIO should be connected to the `digits_vec` signals, while the output port to the `RST`, `STOP` and `START` control signals.

Since the VIO requires `std_logic_vector` as inputs, connect them to the `std_logic_vector` signals we created before.

This time you also need to assign the values of the `std_logic` signals `RST`, `STOP` and `START` to the first index of the `std_logic_vector` signals. E.g.

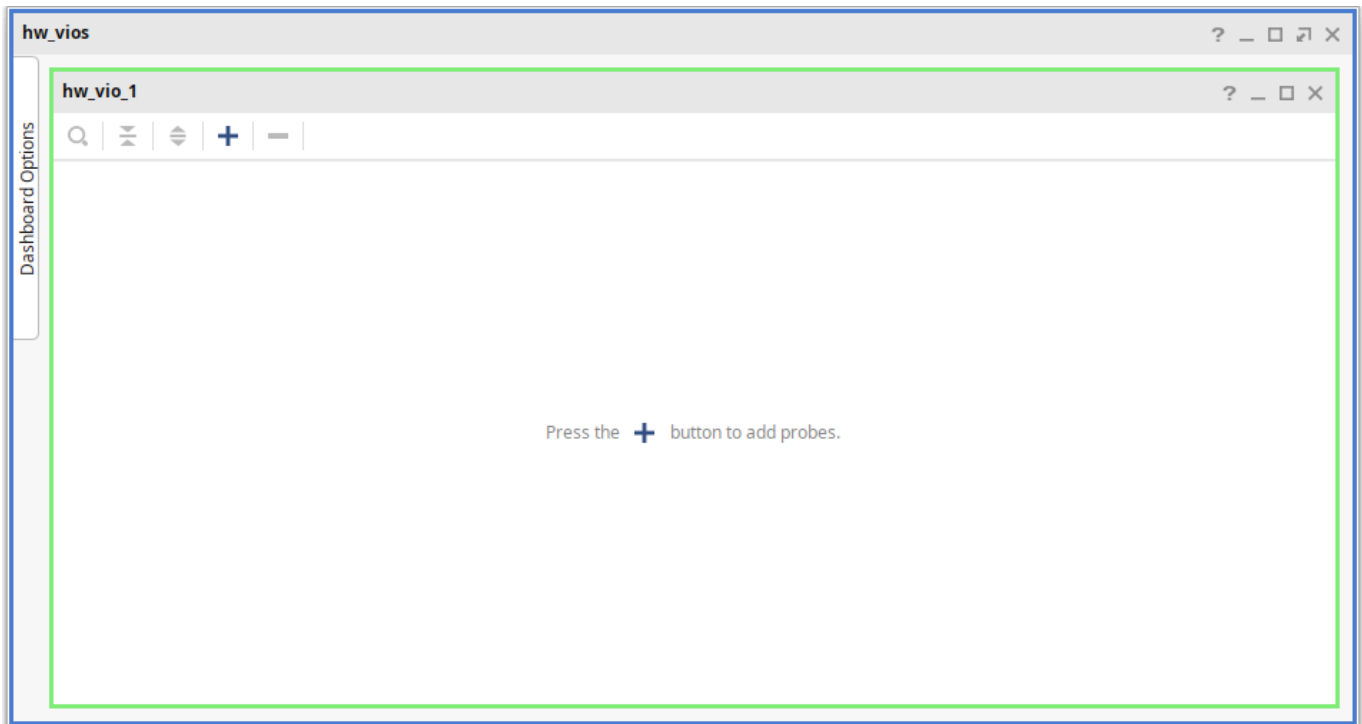
```
START <= START_slv(0);
```

Open now the `basys3.xdc` constraint file and comment out any reference to the RST, START and STOP ports.

Save everything, implement the design and regenerate the bitstream.

You can also have another look at the implemented design, to check for the difference in resource usage.

Open again the Hardware manager, load the new bitstream. This time you should also have tab for the VIO (`hw_vios`).



Click on the plus button and add all the signals in the list. Select all of them. Select all the control signals, click with the right button, and set them *Active-High Button*. Select now the digit signals, right click, and set the Radix as unsigned decimal.

Click now on the START button. You should see the stopwatch digits starting to increase both in the VIO and on the 7-segment display on the board. Continue to play with the other control signals, to verify the correct operation of the stopwatch.