San Jose State University

Department of Electrical Engineering

EE178, Fall 2021

# Laboratory Assignment #5

## Objectives

This lab is an introduction to basic hardware debugging with Xilinx Vivado. Basic hardware debugging includes use of tools for the purpose of observation and control of the device under test. No new logic design concepts are presented in this lab. The goal of this lab is for you to become more familiar with the tools. Please read carefully, pay attention, and take your time. This lab is not a race to see who gets done first.

In order to receive credit for this lab, you must demonstrate to the instructor that your final design works correctly in hardware.

## Bibliography

This lab draws heavily from documents on the Xilinx website [https://www.xilinx.com](https://www.xilinx.com/). I would like to thank Xilinx for making this material available. This lab is effectively a customized introduction to Xilinx Vivado using Verilog-HDL and the Real Digital Blackboard.

## Design Entry

Referring to the previous assignments if necessary, open Vivado and create a new project. After you have created a new project, create a new design source named fsm.v and replace the automatically generated contents of the file with the following:

// File: fsm.v

// This is the top level design for EE178 Lab #5.

// The `timescale directive specifies what the

// simulation time units are (1 ns here) and what

// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is

// using Verilog-2001 syntax.

module fsm (

input wire restart\_from\_pin,

input wire pause\_from\_pin,

input wire clk,

output reg even,

output reg odd,

output reg terminal

);

localparam [1:0] FIRST = 2'b11;

localparam [1:0] SECOND = 2'b10;

localparam [1:0] THIRD = 2'b01;

reg [1:0] state = FIRST;

wire restart\_from\_vio, restart;

wire pause\_from\_vio, pause;

// Begin INSTANTIATION Template from VEO or VHO File

vio\_0 my\_vio (

.clk(clk), // input wire clk

.probe\_in0(restart), // input wire [0 : 0] probe\_in0

.probe\_in1(pause), // input wire [0 : 0] probe\_in1

.probe\_in2(state), // input wire [1 : 0] probe\_in2

.probe\_out0(restart\_from\_vio), // output wire [0 : 0] probe\_out0

.probe\_out1(pause\_from\_vio) // output wire [0 : 0] probe\_out1

);

// End INSTANTIATION Template from VEO or VHO File

assign restart = restart\_from\_pin || restart\_from\_vio;

assign pause = pause\_from\_pin || pause\_from\_vio;

always @(posedge clk)

begin

case(state)

FIRST: if (restart || pause) state <= FIRST;

else state <= SECOND;

SECOND: if (restart) state <= FIRST;

else if (pause) state <= SECOND;

else state <= THIRD;

THIRD: if (!restart && pause) state <= THIRD;

else state <= FIRST;

default: state <= FIRST;

endcase

end

always @\*

begin

even = (state == SECOND);

odd = (state == FIRST) || (state == THIRD);

terminal = (state == THIRD) && (restart || !pause);

end

// Begin INSTANTIATION Template from VEO or VHO File

ila\_0 my\_ila (

.clk(clk), // input wire clk

.probe0(restart), // input wire [0 : 0] probe0

.probe1(pause), // input wire [0 : 0] probe1

.probe2(state), // input wire [1 : 0] probe2

.probe3(even), // input wire [0 : 0] probe3

.probe4(odd), // input wire [0 : 0] probe4

.probe5(terminal) // input wire [0 : 0] probe5

);

// End INSTANTIATION Template from VEO or VHO File

endmodule

You will notice this is a version of the FSM example used in the lecture material. Its value in this assignment is the relative simplicity it provides.

After you have saved the new design source, create a new constraint source named fsm.xdc and enter the constraints provided below then save the file. Take a moment to read the constraints to understand how the FSM inputs are assigned to switches SW0 and SW1, and how the FSM outputs are assigned to indicators LD0, LD1, and LD2.

# Constraints for CLK

set\_property PACKAGE\_PIN H16 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -period 10.000 -name my\_only\_clock -waveform {0.000 5.000} [get\_ports clk]

# Constraints for SW0

set\_property PACKAGE\_PIN R17 [get\_ports restart\_from\_pin]

set\_property IOSTANDARD LVCMOS33 [get\_ports restart\_from\_pin]

# Constraints for SW1

set\_property PACKAGE\_PIN U20 [get\_ports pause\_from\_pin]

set\_property IOSTANDARD LVCMOS33 [get\_ports pause\_from\_pin]

# Constraints for LD0

set\_property PACKAGE\_PIN N20 [get\_ports even]

set\_property IOSTANDARD LVCMOS33 [get\_ports even]

# Constraints for LD1

set\_property PACKAGE\_PIN P20 [get\_ports odd]

set\_property IOSTANDARD LVCMOS33 [get\_ports odd]

# Constraints for LD2

set\_property PACKAGE\_PIN R19 [get\_ports terminal]

set\_property IOSTANDARD LVCMOS33 [get\_ports terminal]

Although it may not have occurred to you previously, when you add or edit source files, the Vivado Project Manager analyzes the source files to gain understanding of the design hierarchy. In this case, Vivado will determine from the available source files in the project, that there are instantiated modules for which it has no source files. This status is designated with a “?” icon next to the modules in the design sources tree, as shown in Figure 1.

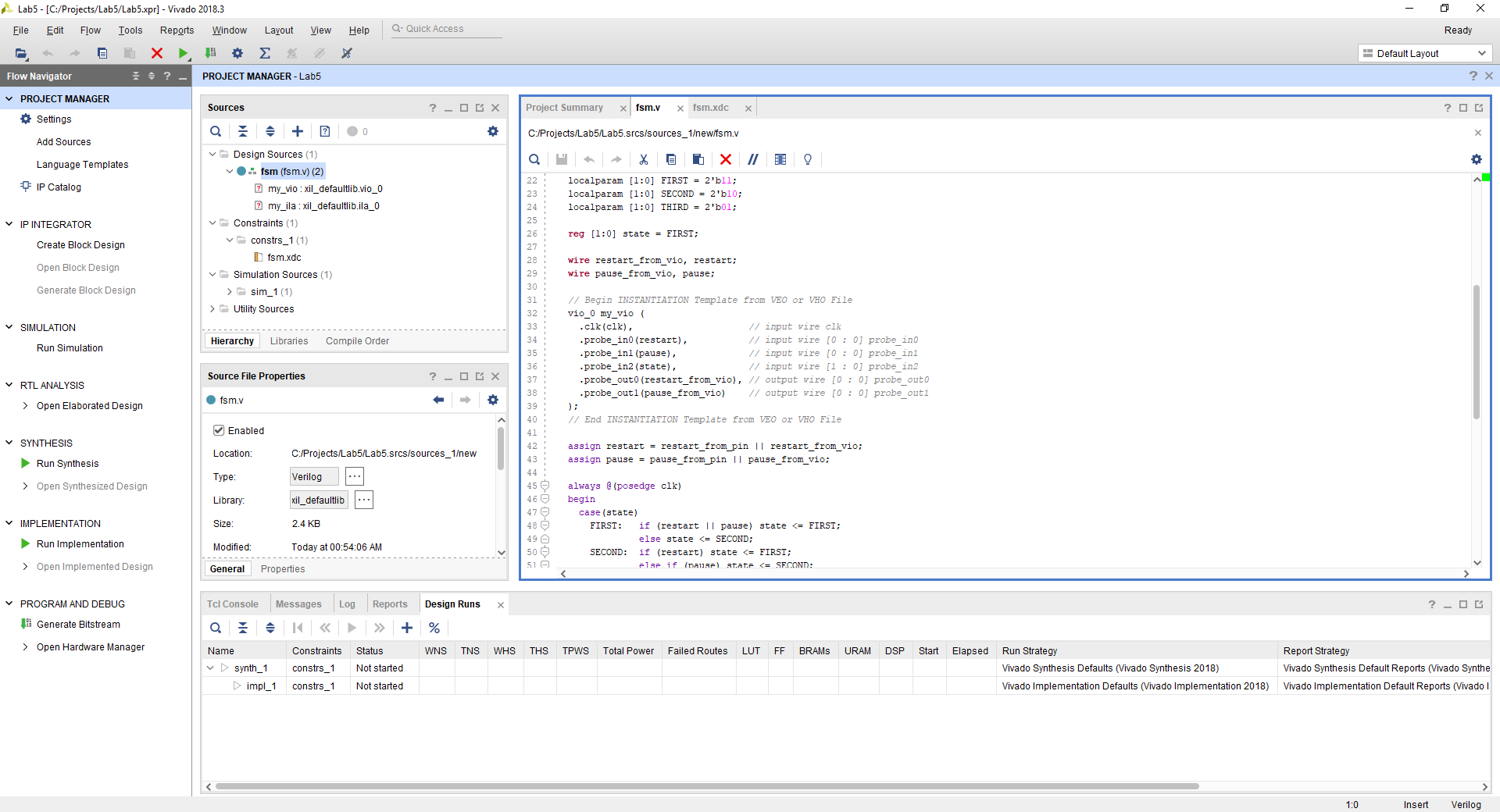


Figure 1: Project Missing Source Files for Instantiated Sub-Modules

You might encounter the “?” icon transiently while setting up a project, while you are in the process of adding or creating source files, or if you have syntax errors in your code. These scenarios can result in Vivado being unable to identify source code for all of the modules it may be expecting to find.

In this project, source files are indeed missing. The missing items are two IP cores for debug. IP is short for Intellectual Property, a widely used term. In the context of digital logic design, IP cores are essentially pre-made sub-modules intended for re-use. Xilinx provides many at no cost through the Vivado IP Catalog.

In the Flow Navigator, under Project Manager, click on the IP Catalog. The IP Catalog will open, showing an organized list of IP cores which are available. Browse through the list to find the IP cores for debug. The two you will need are identified with red boxes in Figure 2.

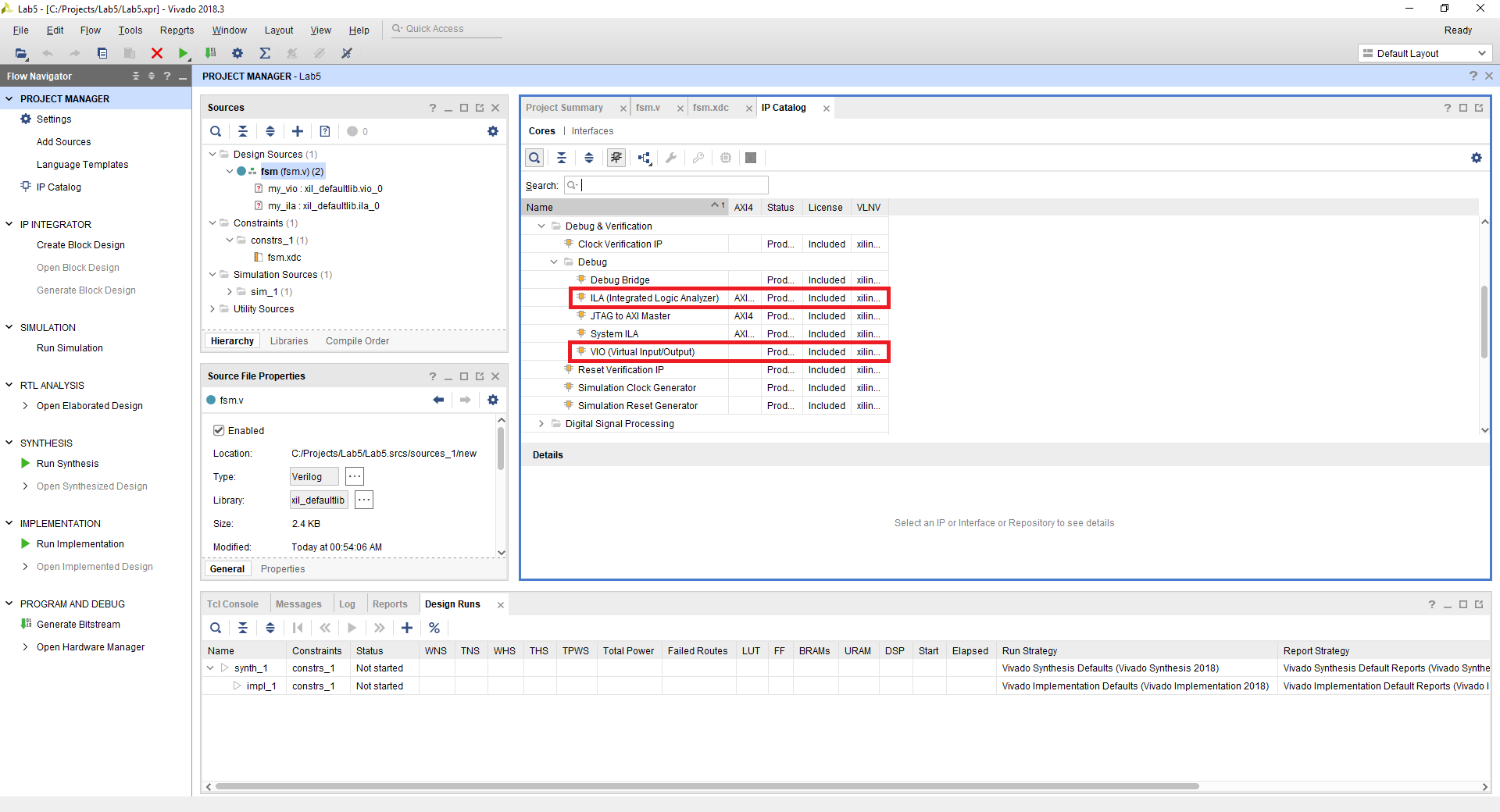


Figure 2: IP Cores for Debug

Begin by double clicking the Integrated Logic Analyzer (ILA) core. This will launch an IP configuration window. This core is highly flexible, and many variations of it are possible.

Copy the settings shown in both Figure 3 and Figure 4 exactly, before you click OK at the bottom of the window.

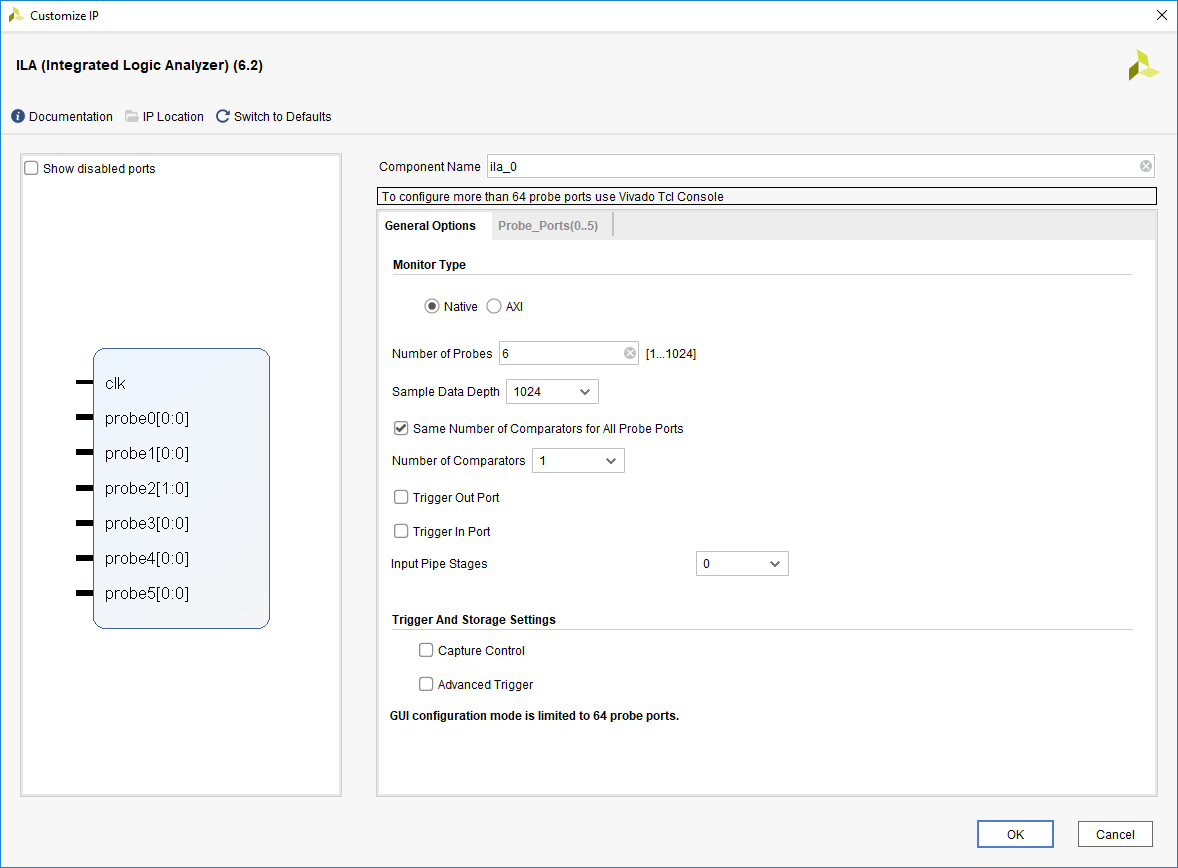


Figure 3: ILA General Options

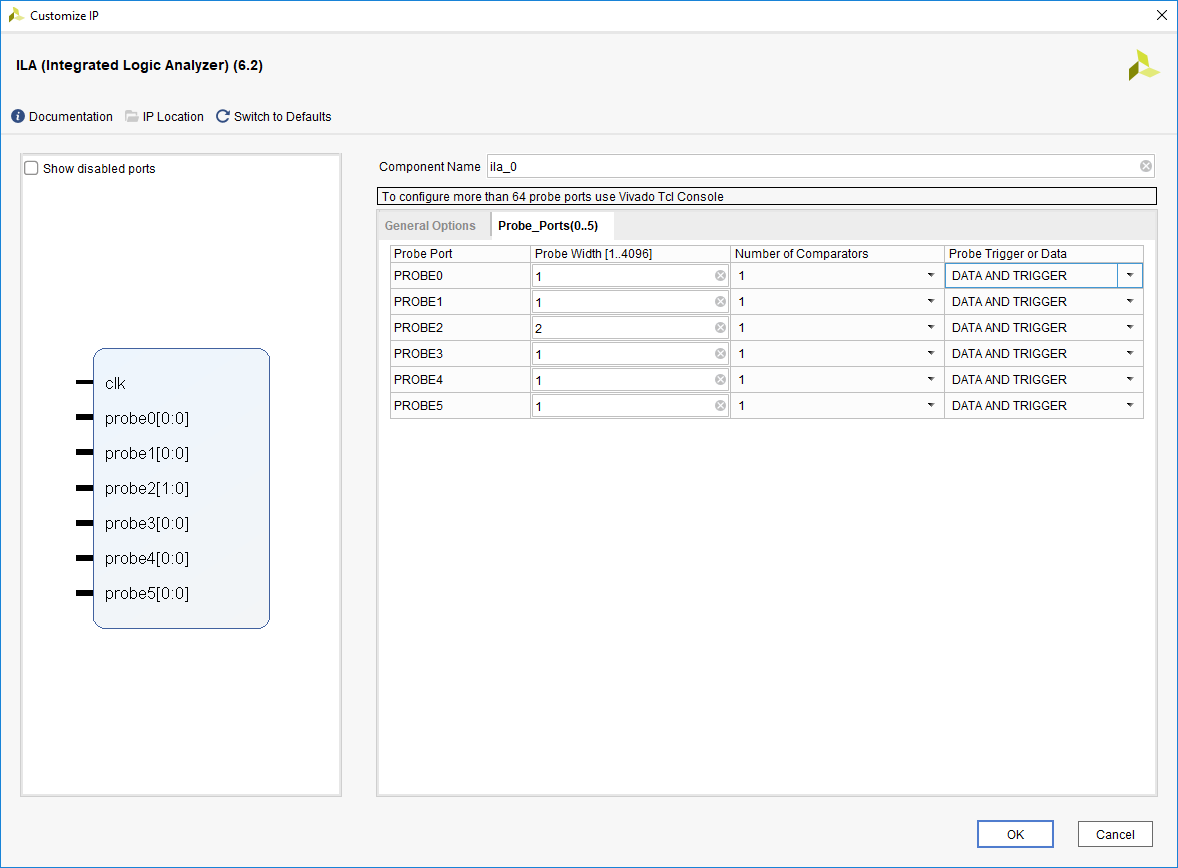


Figure 4: ILA Probe Ports

When you click OK, Vivado may ask for your permission to create a project subdirectory for storing IP core source files. Then, Vivado will confirm what it will do next, a process called output product generation, in essence – the creation of the IP core source files. This dialog is shown in Figure 5, and you should accept the defaults as shown.

These defaults result in Vivado creating the IP core source files and then additionally running synthesis on the IP core immediately. This immediate synthesis of the IP core by itself is called “out of context” synthesis and although you will spend a few moments waiting for it to complete, it is a one-time process that eliminates the need for it to be done each time you implement your project – a significant time saver.

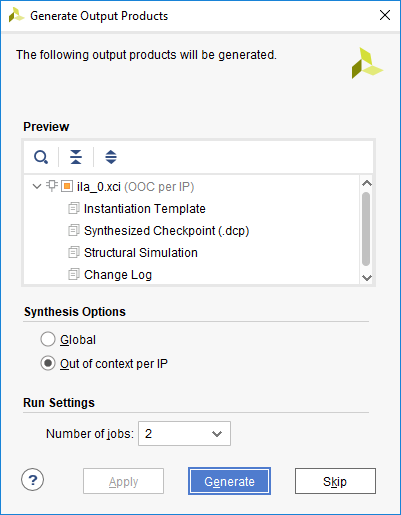


Figure 5: Generate ILA Output Products

Continue by double clicking the Virtual Input / Output (VIO) core. This will launch an IP configuration window. This core is also highly flexible, and many variations of it are possible.

Copy the settings shown in both Figure 6, Figure 7, and Figure 8 exactly, before you click OK at the bottom of the window.

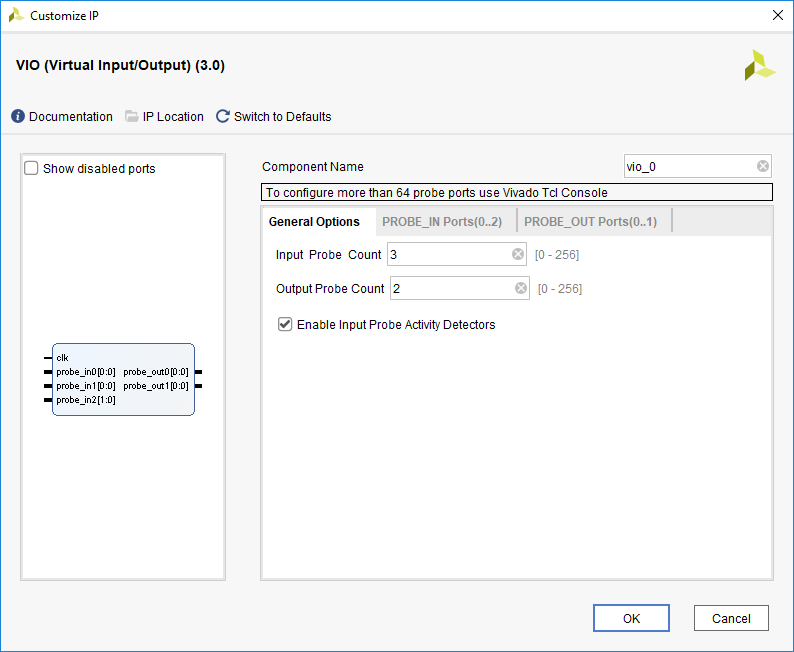


Figure 6: VIO General Options

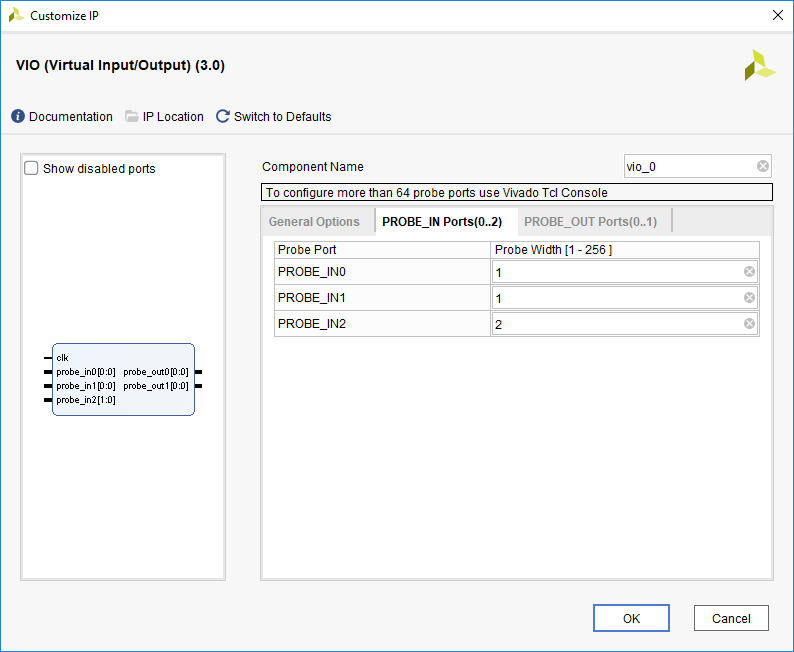


Figure 7: VIO Probe Input Options

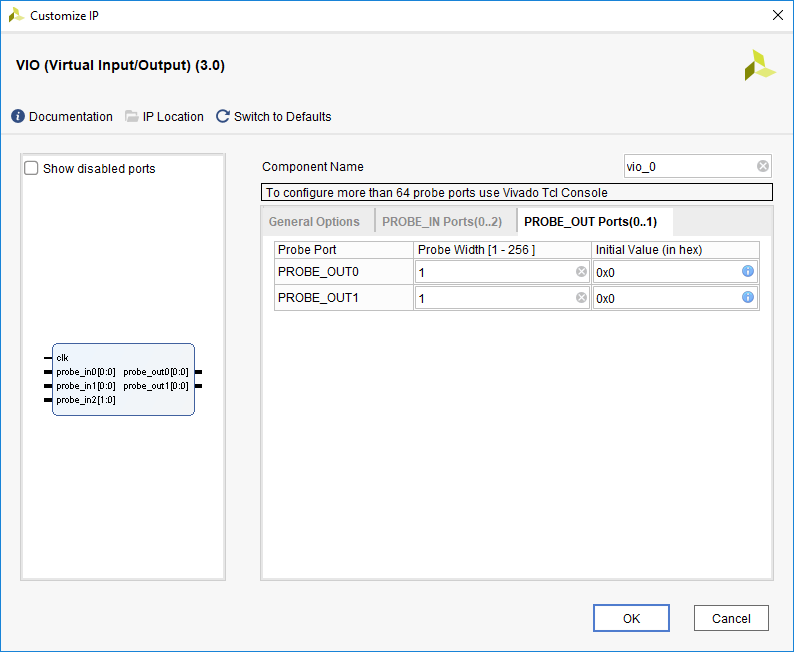


Figure 8: VIO Probe Output Options

As with the previous IP core, after you click OK, continue to accept the default output product generation settings as shown in Figure 9.

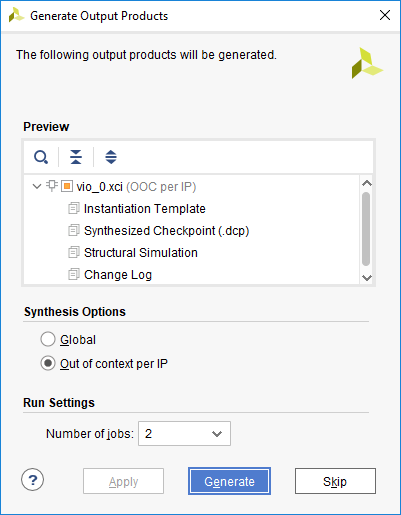


Figure 9: Generate VIO Output Products

Assuming you have copied the settings for ILA and VIO exactly, after both IP cores have been generated, the Vivado Project Manager will no longer show “?” for these modules as shown in Figure 10.

In the event you have made an error during configuration of one or both of these IP cores, you can investigate further in the IP Sources tab, and right click on an IP core for a context menu, which provides options to re-configure it, or you might consider another option to remove it from the project and re-create it. This assignment does not cover all possible IP core management operations, so please ask the instructor for assistance if necessary.

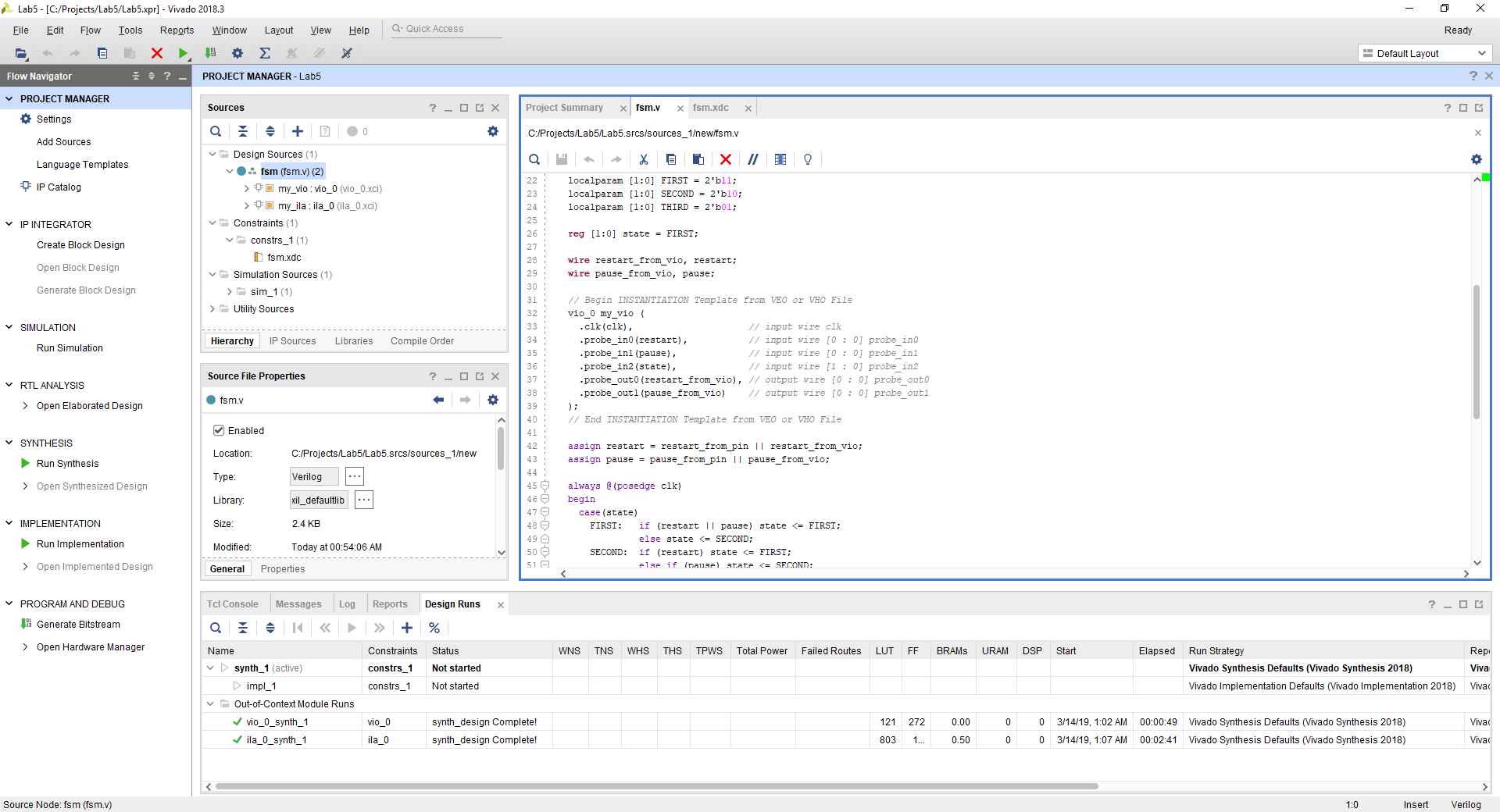


Figure 10: Project with All Source Files Present

In a typical debug scenario, you will make choices about what signals you would like to control, observe, and record. The natural flow of using these IP cores is that you first decide what you want, generate the IP cores, and then instantiate and connect them in your design as you wish.

In this exercise, I made quasi-meaningful but, truthfully, arbitrary choices for the purposes of exhibition of the debug tools. I wanted a method to control the FSM inputs, so I decided to logically OR two virtual outputs with signals coming from pins. Then, to observe what is actually input to the FSM, I used two virtual inputs. Additionally, I wanted to observe the FSM state so that I could answer some basic questions such as, “Is the FSM actively changing state?” and “If the FSM is not actively changing state, what state is it parked in?” For this, I used an additional virtual input, one that was 2-bit wide, based on the width of the FSM state element. Although I could have also created more virtual inputs to observe the FSM outputs, I arbitrarily decided I did not want to do that, in favor of recording those instead.

To gain cycle-by-cycle understanding of what is taking place in hardware, I wanted a method to record all signals coming in and out of the FSM. Assuming the integrated logic analyzer is operating on the same clock as the FSM, by extension it becomes a part of the synchronous logic design and the clock activity does not need to be explicitly recorded. I decided to record the two inputs, the FSM state element, and the three outputs.

After making these arbitrary choices, I instantiated and connected the IP cores accordingly, to avoid need for students to edit the source files (thereby, eliminating risk of typos and syntax errors). This is the reason you must copy ILA and VIO settings exactly, to mirror what I did, resulting in identical module names and ports (quantity, names, widths, and directions). The resulting IP cores that are generated by students “should just drop into the project and work” which is a famous-last-words kind of statement.

That said, if you were reusing what you learn here, to debug a different project, you may want a different mix of ILA and VIO, and each of them may have a unique set of ports. This is all at your discretion. But, how do you know the correct way to instantiate each of the IP cores you generate?

The files created in the output product generation step typically include instantiation templates. You don’t see them in the design sources tree, but they are added to the project – go see for yourself. Referring to Figure 10, if you click on the IP Sources tab, Vivado will show you a list of the IP cores that have been generated. If you expand the tree, you will find the instantiation templates for each IP core. The file extensions for these templates are VEO (Verilog) and VHO (VHDL). It is usually the case that both of these files are generated. Since we are working in Verilog, if a VEO file is available, that is the easiest; simply copy and paste the instantiation code from the VEO file into your code. However, you can achieve the same result by inspecting the VHO file to understand the IP core details, and then translate those into your code as an instantiation.

Run the complete implementation flow to generate a bitstream, and when that has completed, connect your board, power it on, and open the Hardware Manager. Proceed to program the device. You will notice in Figure 11 that not only is a BIT file listed, but also an LTX file.

The LTX file contains information extracted from the project, regarding what signals you have connected to the ILA and VIO. This enables the Hardware Manager to display these to you by name, rather than as numbered “channels” or “probes” and in some cases may even enable the Hardware Manager to display signal values using names rather than binary – for example, wouldn’t it be nice to see the current FSM state represented by names in your code, e.g. FIRST, SECOND, THIRD?

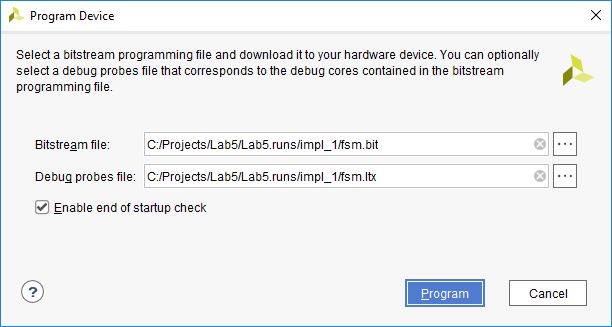


Figure 11: Program Device

With the device programmed, spend a little time exploring how the two switches affect what you can observe on the three indicators. There is no way for you to control or observe the FSM at speed, but you could try some quick sanity experiments.

For example, if you set both inputs high on the switches, that should park the FSM in the FIRST state, and the indicators ought to match the expected outputs from that state. Another experiment you could try is to set both inputs low on the switches, allowing the FSM to quickly cycle through its complete state sequence, and the indicators may exhibit brightness related to the duty cycle of each output. You should be able to rationalize what you see, but we can gain much better insight with the integrated debug tools!

Returning to the Hardware manager, expect that the Hardware Manager has identified the debug IP cores in your design and has connected to them so you can use them. In the main window, you will see two tabs, one for the ILA and the other for the VIO. Select the VIO tab first, as shown in Figure 12.

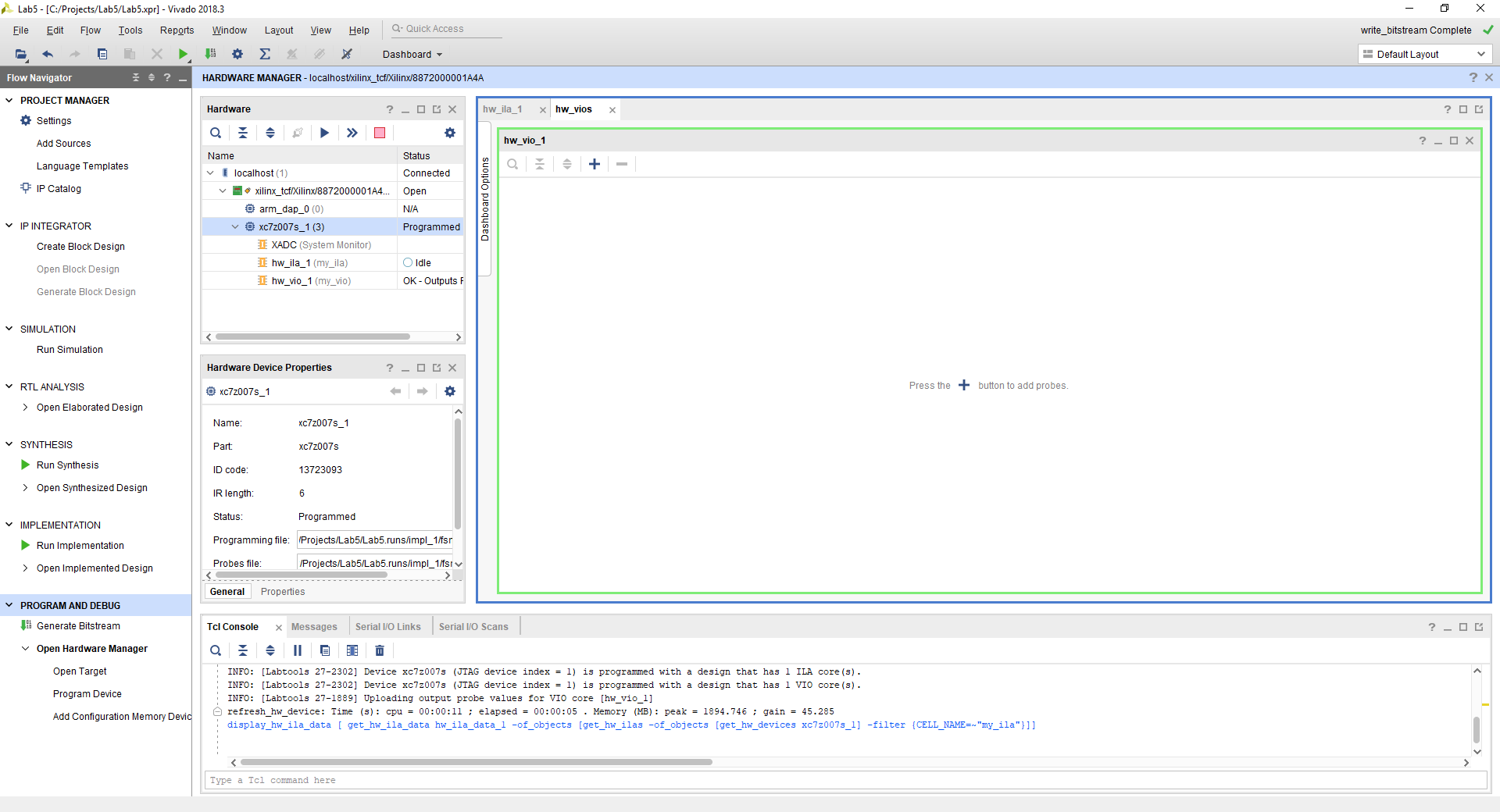


Figure 12: VIO Tab in Hardware Manager

To start using the VIO, you need to add probes to the VIO tab. One way to do this is to click on the “+” which is offered in two places, in the middle of the empty VIO tab, as well as the top left of the VIO tab. When you click on the “+” you will be offered a list of available probes as shown in Figure 13.

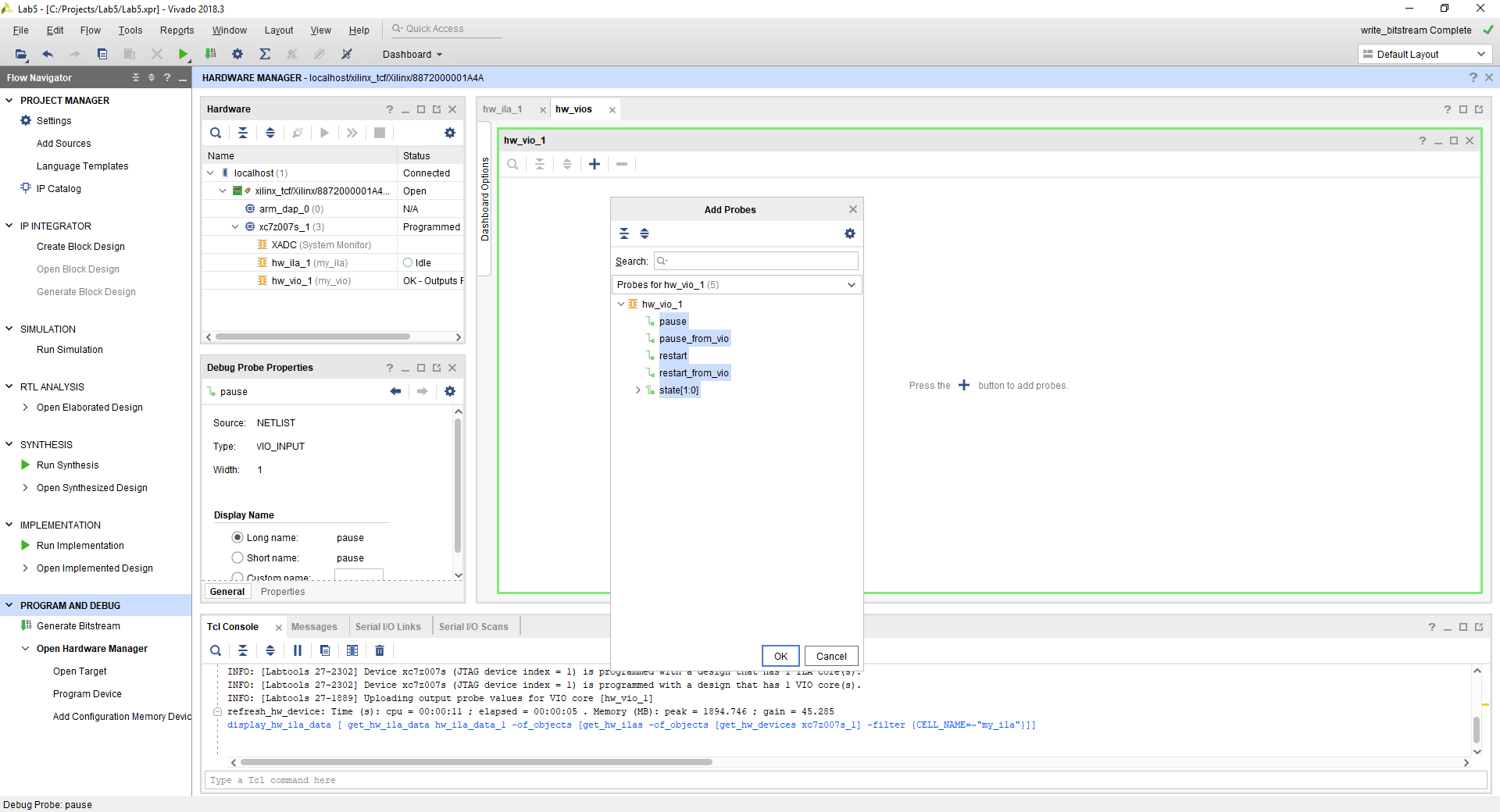


Figure 13: Adding Probes to VIO Tab

For our purposes, we want to add all of the available probes. You can select all the probes individually or you can simply click on the VIO instance which selects all the probes connected to it. After you accept the selections, the screen should look similar to Figure 14. You may need to adjust the column widths of the VIO probe table for best viewing.

The reason for stating “look similar” vs. “look exact” is that what you see will depend on the current switch settings on your board. To make it “look exact” set both inputs low on the switches. Then, you should see exactly what is shown in Figure 14 – namely, that all the signals are low, except for the FSM state, which is changing state frequently. The VIO has activity detectors and will indicate recent low-to-high transitions, recent high-to-low transitions, and recent toggling activity using arrows in the activity column of the VIO probe table.

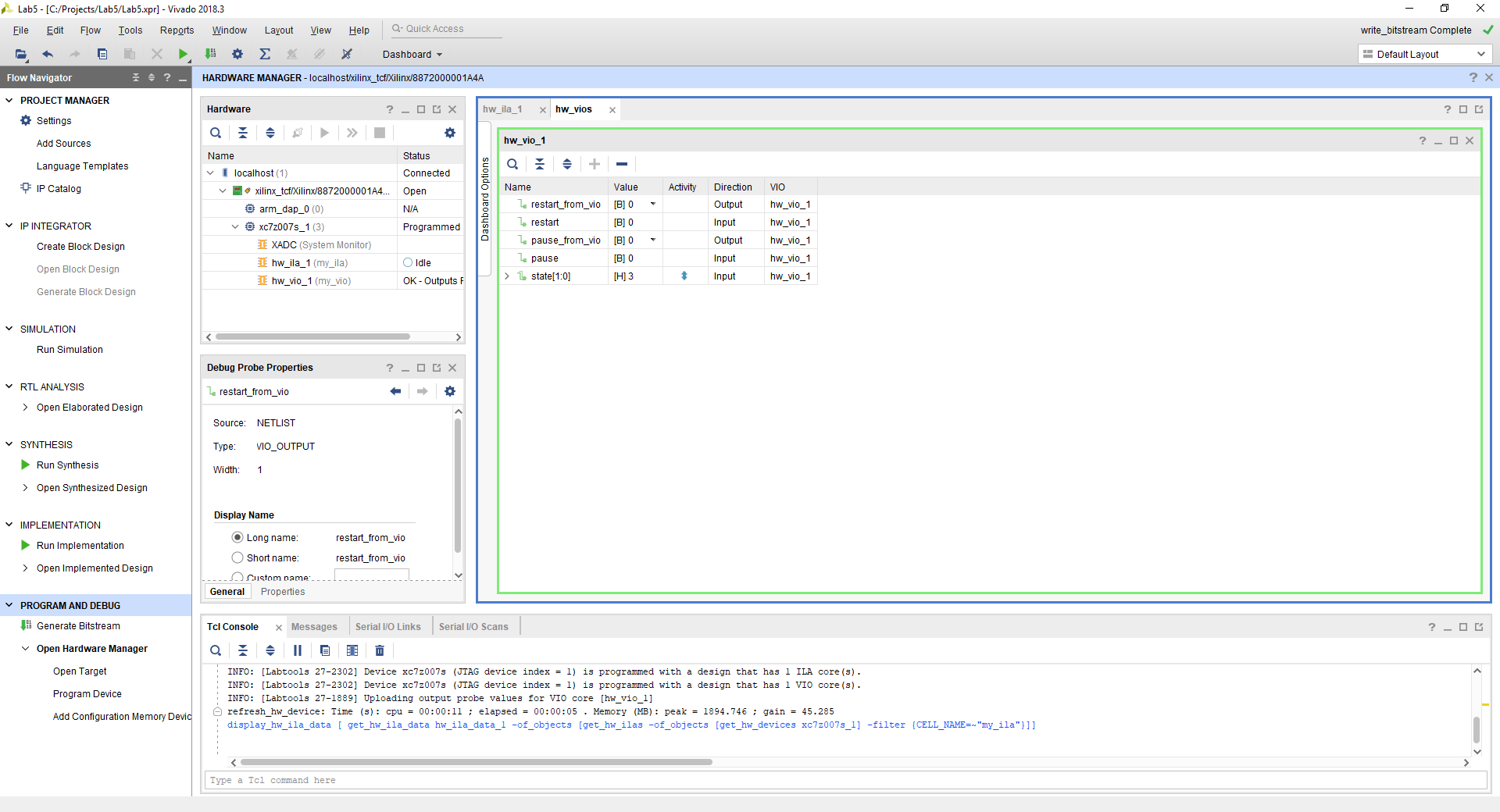


Figure 14: VIO Tab with Probes, All Inputs Low

Now, keeping both inputs low on the switches, use the Hardware Manager to force the FSM inputs high. You do this in the VIO tab by setting the value of those signals high (these will be the only two signals in the table where you are allowed to control the value). The result is shown in Figure 15. The FSM should be parked in the FIRST state – convince yourself this is true by referring to the FSM code where the state encodings are established through local parameter values. Leave the FSM parked in this condition – both inputs low on the switches, both inputs to the FSM forced high, and the FSM parked in the FIRST state.

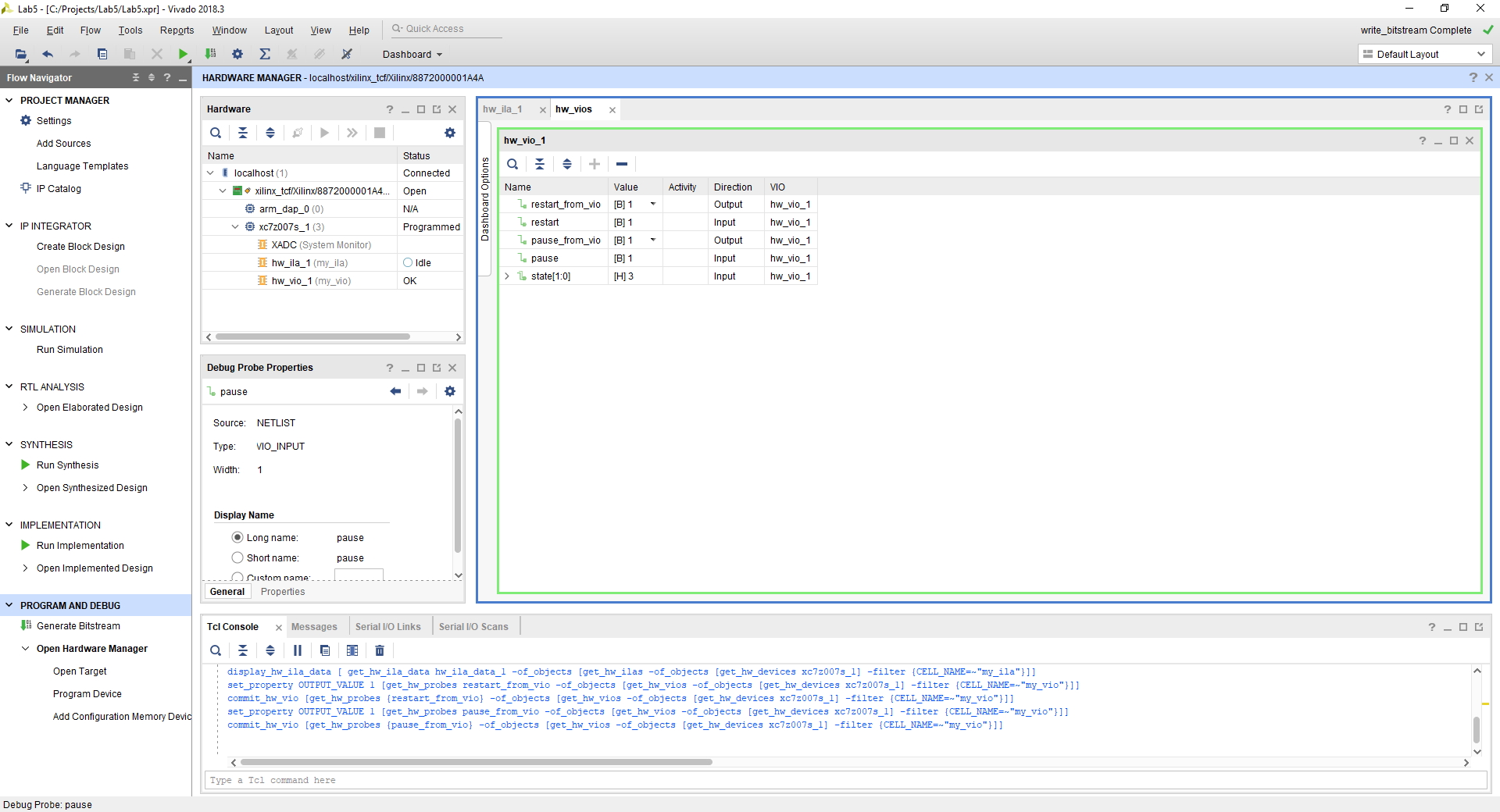


Figure 15: VIO Tab with Probes, All Inputs High

Switch to the ILA tab in the Hardware Manager, as shown in Figure 16. This interface has similarities to the Vivado simulator. It pre-populates signals, but you can add signals using the “+”, and remove or change signals by right clicking their names for a context menu. The Hardware Manager will save the setup of your waveform display between sessions.

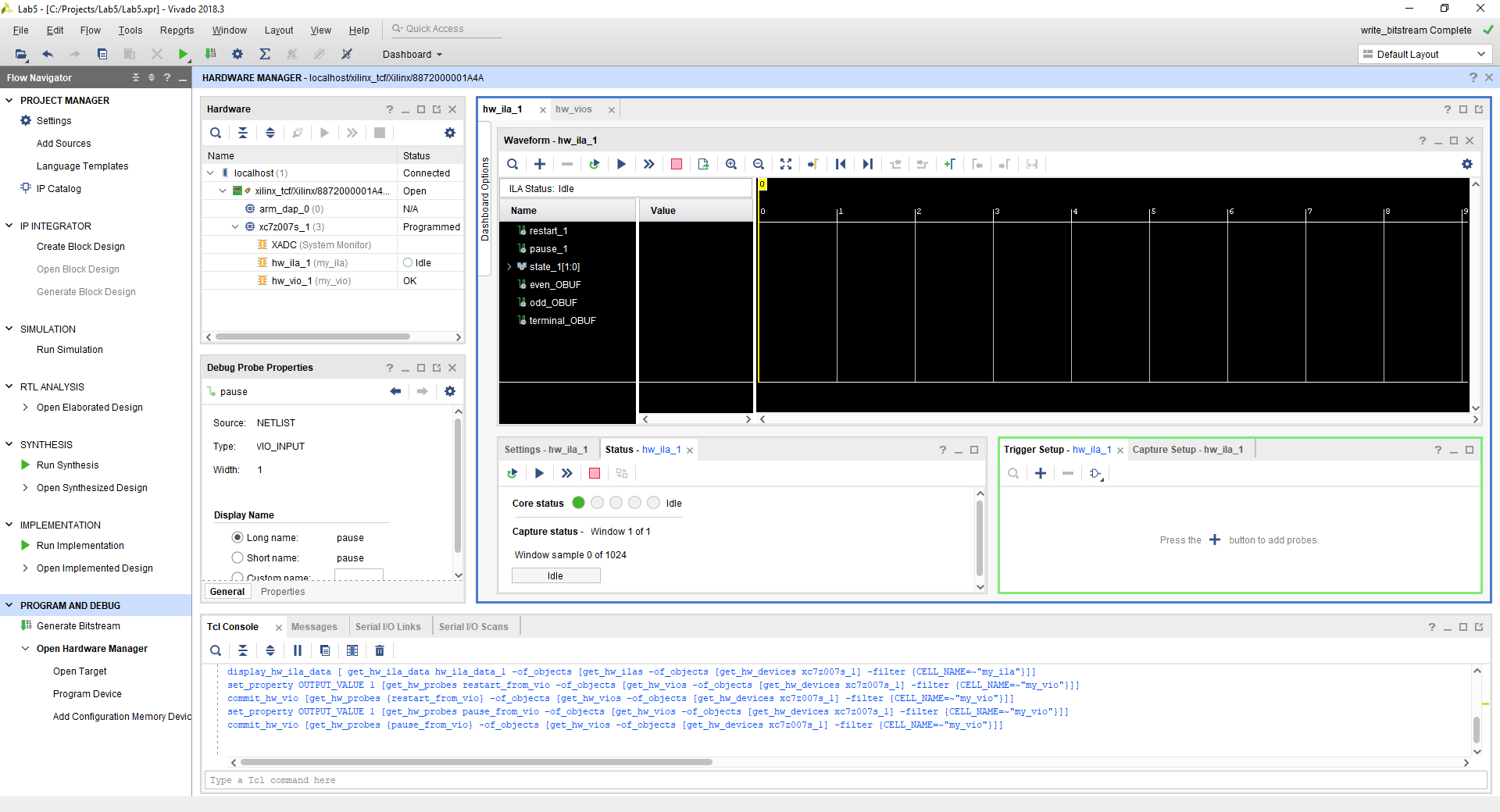


Figure 16: ILA Tab in Hardware Manager

There are several ways to initiate a recording, but a common and very helpful method is to set a trigger event. A trigger event is a condition for which the ILA will monitor, and when the condition is satisfied, the ILA will record until its recording buffer is full.

Recording buffer size is one of the configuration options you can modify when configuring the ILA core from the IP catalog. Also note there are options for pre-store in advance of the trigger event, so that the ILA will record not only what occurs at the trigger event forward, but also record what has happened leading up to the trigger event. Often, the most important information for debugging is what happened before the trigger event!

For our practice use of the ILA, we will specify a trigger based on the high-to-low transition of the pause input to the FSM. As shown in Figure 17, add a trigger signal using the “+” in the trigger setup pane of the ILA window. Select the pause input to the FSM.

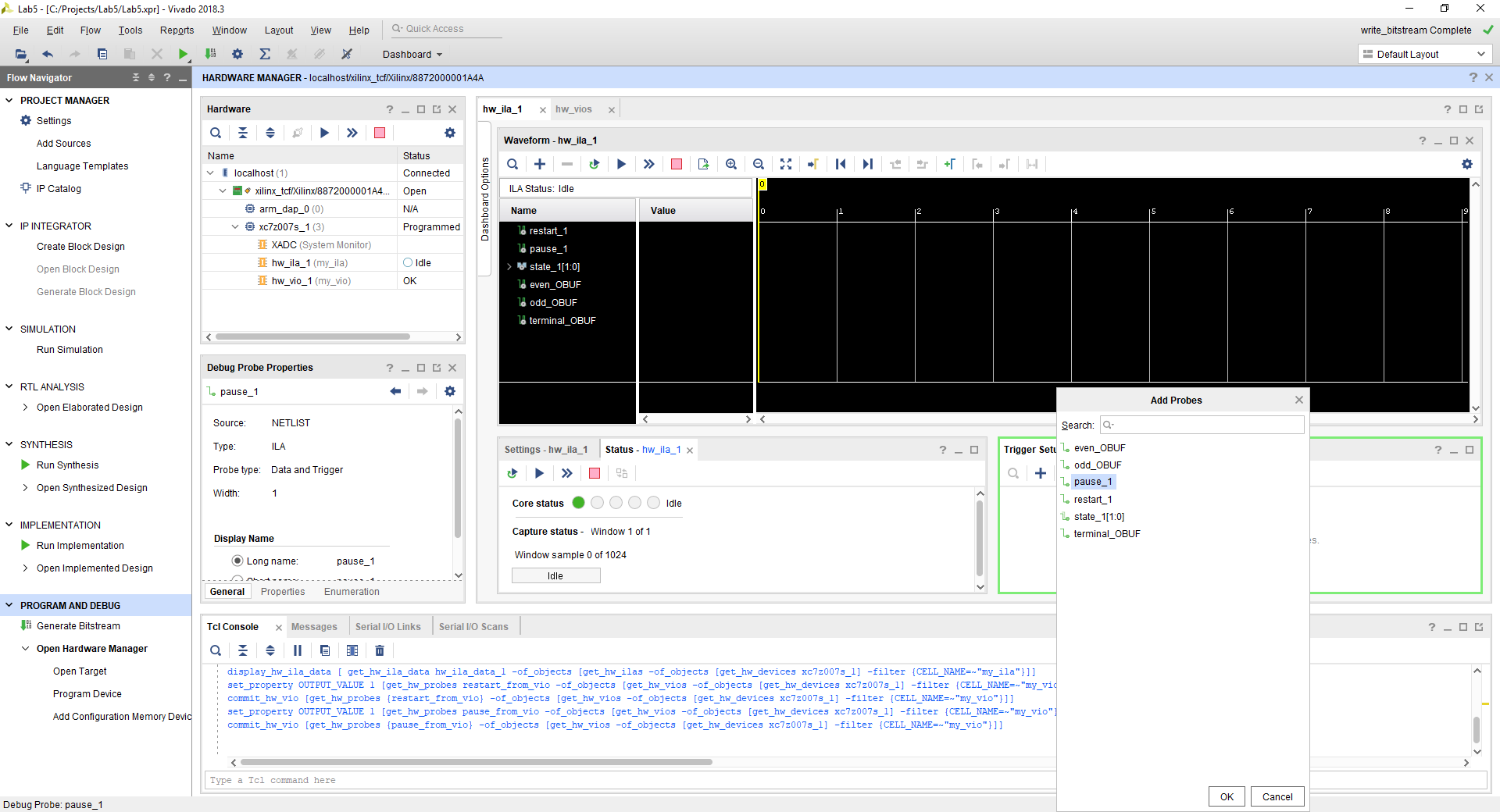


Figure 17: Adding Triggers to ILA Tab – Signal Selection

After selecting the desired signal, it will be listed in the trigger setup pane. Figure 18 shows how to specify the desired condition for that signal. As described, we will use high-to-low (a falling transition). This completes the setup of a simple trigger event. The ILA can support the expression of very complex triggers, a capability that has high value for debugging complex issues in hardware. We’ve only scratched the surface of what is possible.

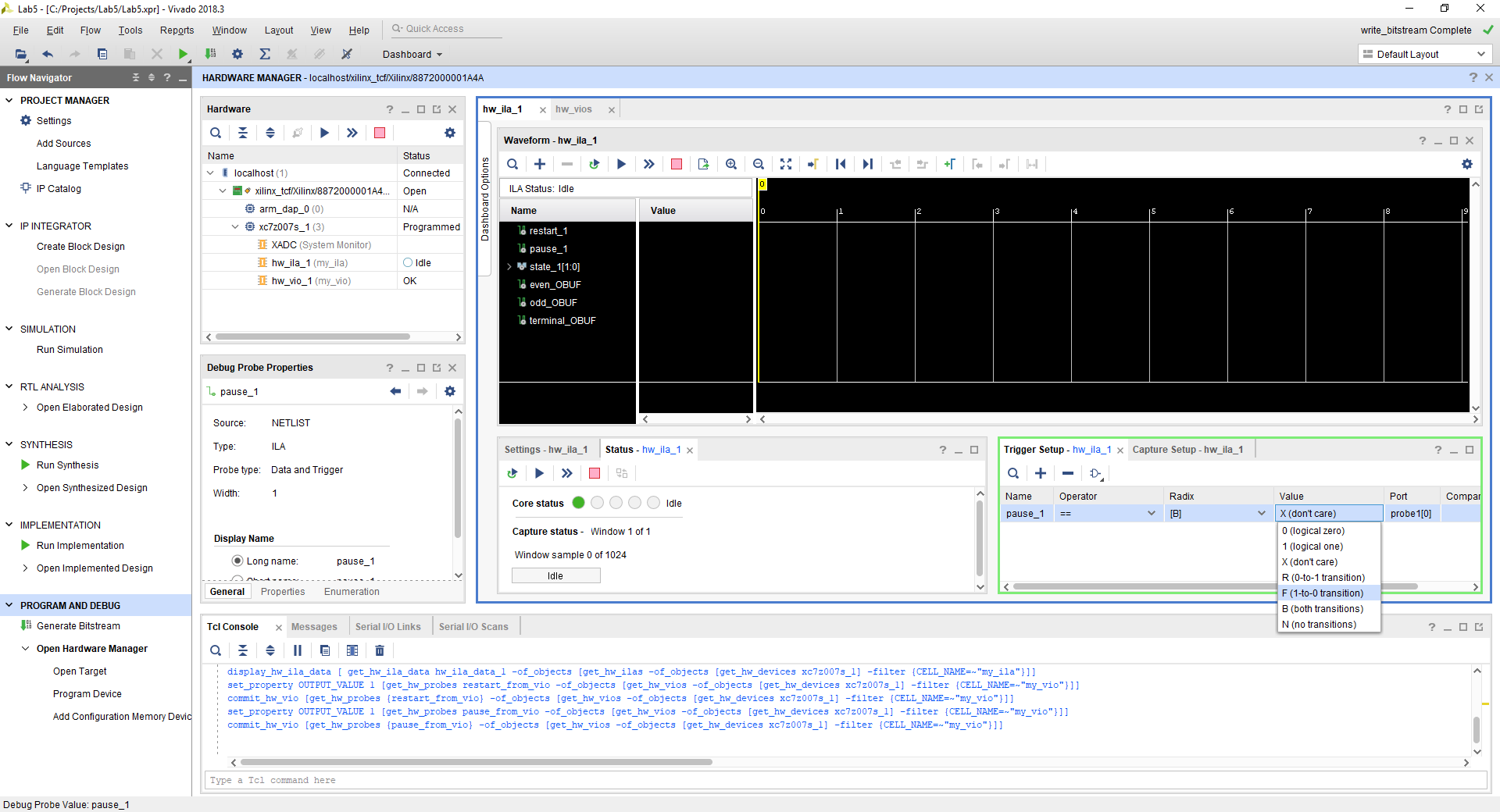


Figure 18: Adding Triggers to ILA Tab – Condition Selection

The next step is to arm the ILA trigger. You can arm it for a one-time capture, or you can arm it for repeated capture. Using the button highlighted in Figure 19, arm the trigger for a one-time capture. In passing, notice the Hardware pane of the window, it shows the hardware contains an ILA and a VIO, with their statuses. In Figure 19, the ILA status is IDLE and the VIO status is OK. When you arm the ILA trigger, it should change from IDLE status into WAITING for trigger status.

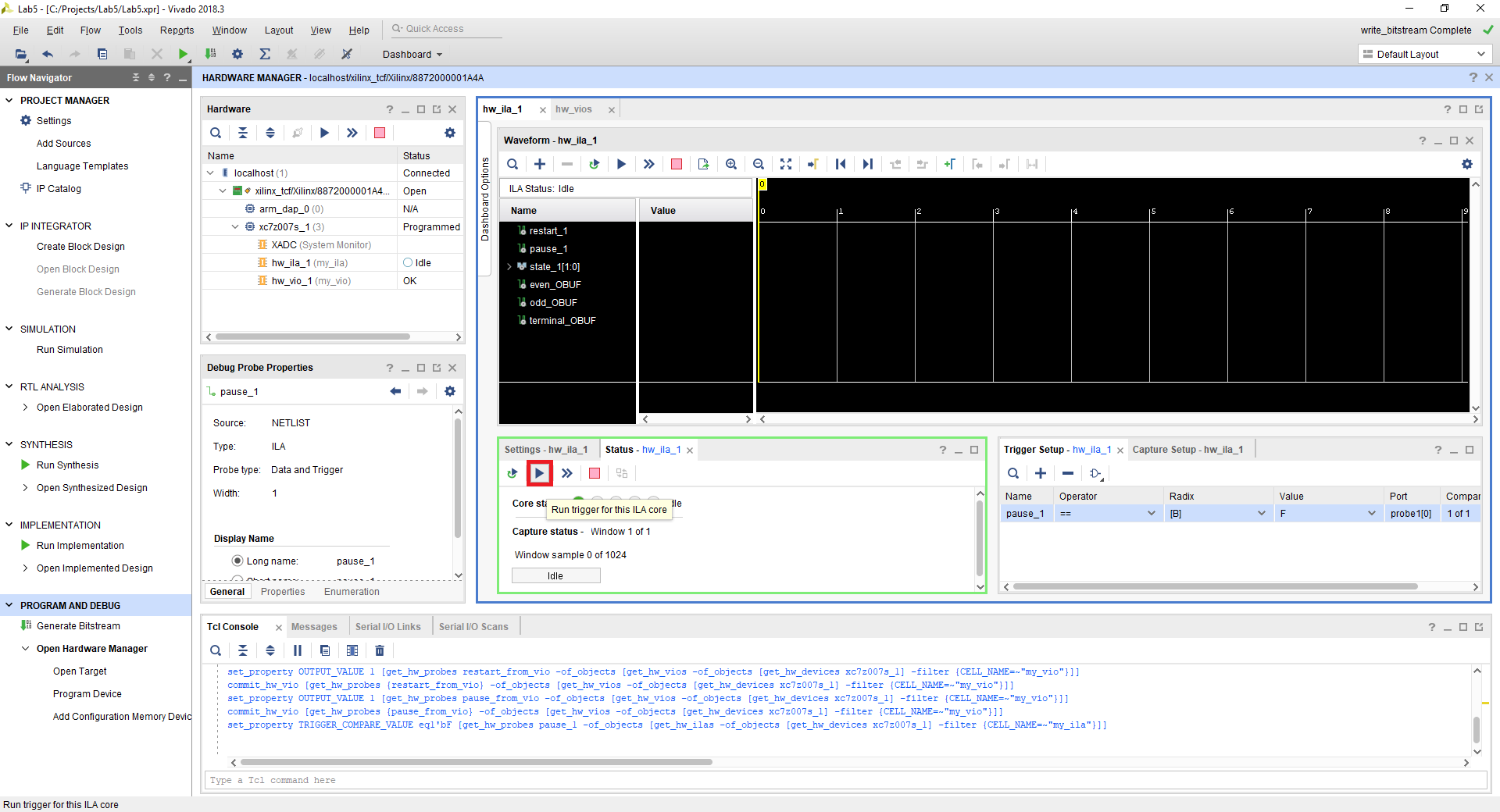


Figure 19: Arming the ILA Trigger

Return to the VIO tab as shown in Figure 20. It should continue to be in the state you left it earlier. Additionally, note how in Figure 20 you can see that the ILA status is WAITING for trigger.

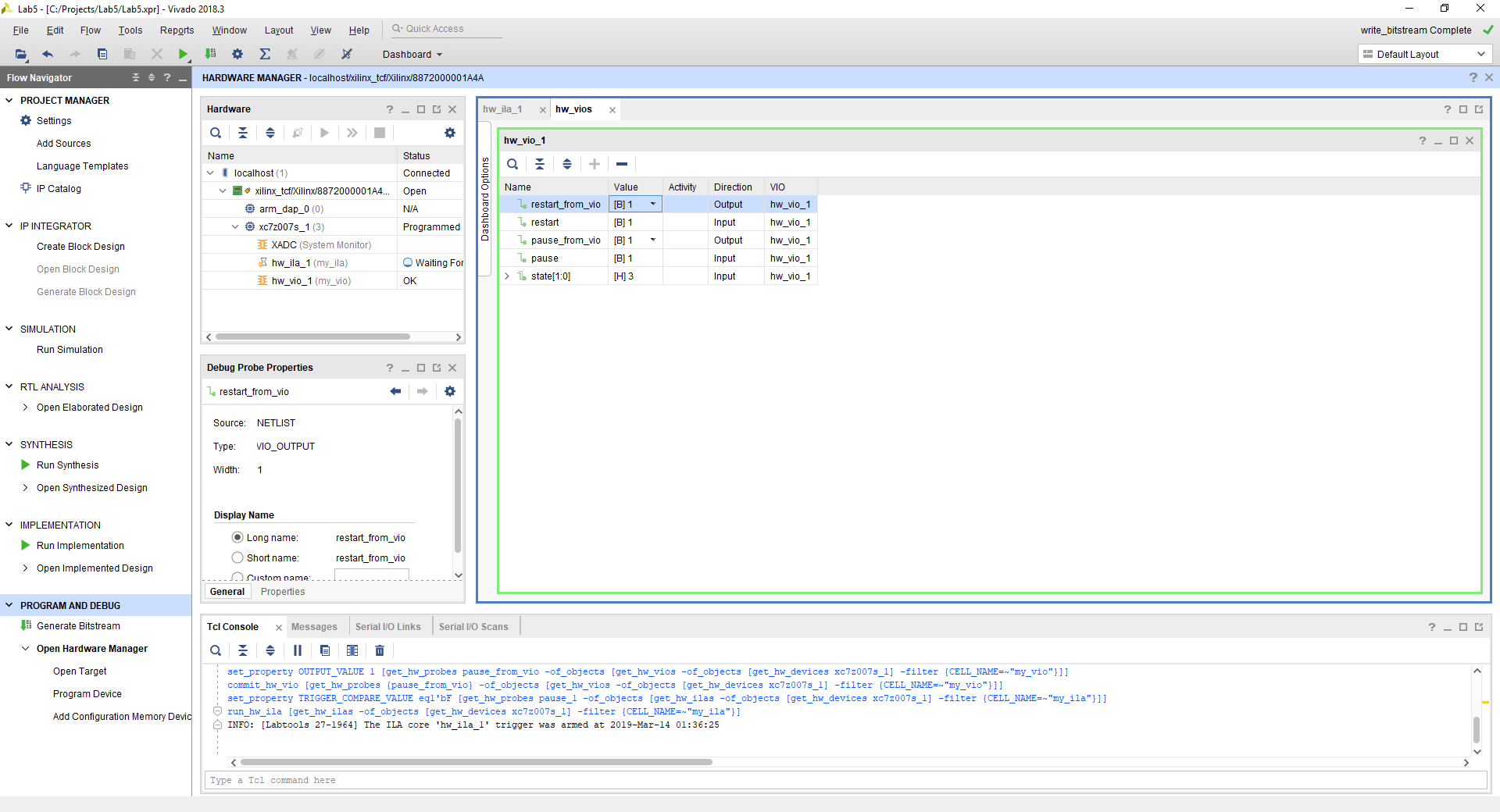


Figure 20: Return to VIO Tab with ILA Waiting for Trigger

First, release the FSM so that it is not being held in its initial state. This is done by changing the virtual output value forcing the restart signal – set it low. The FSM will still be in the initial state without any state transitions, since it continues to be paused, as shown in Figure 21.

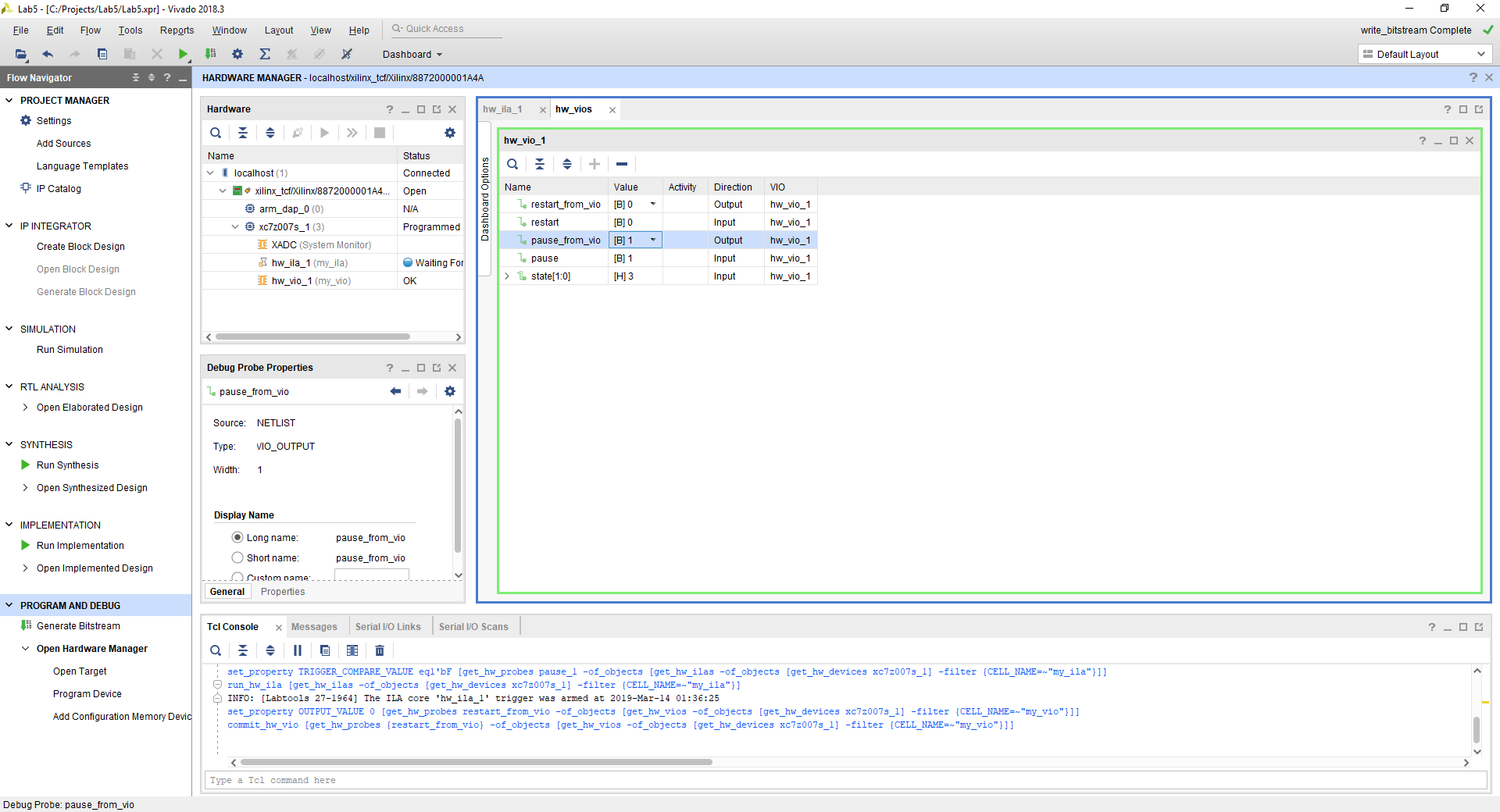


Figure 21: Get Ready for Action

Now, release the FSM so that it is not being paused in its initial state. This is done by changing the virtual output value forcing the pause signal – set it low. As you may realize, this change results in a high-to-low transition on the pause signal, which also happens to be the ILA trigger condition. Upon making this change to the pause signal in the VIO, the ILA should trigger and record what is happening in hardware.

If you watch carefully, at the moment you change the pause signal in the VIO, the ILA status will make several quick transitions from WAITING, eventually arriving at IDLE, and in the VIO the FSM state should indicate on-going transitions. This result is shown in Figure 22.

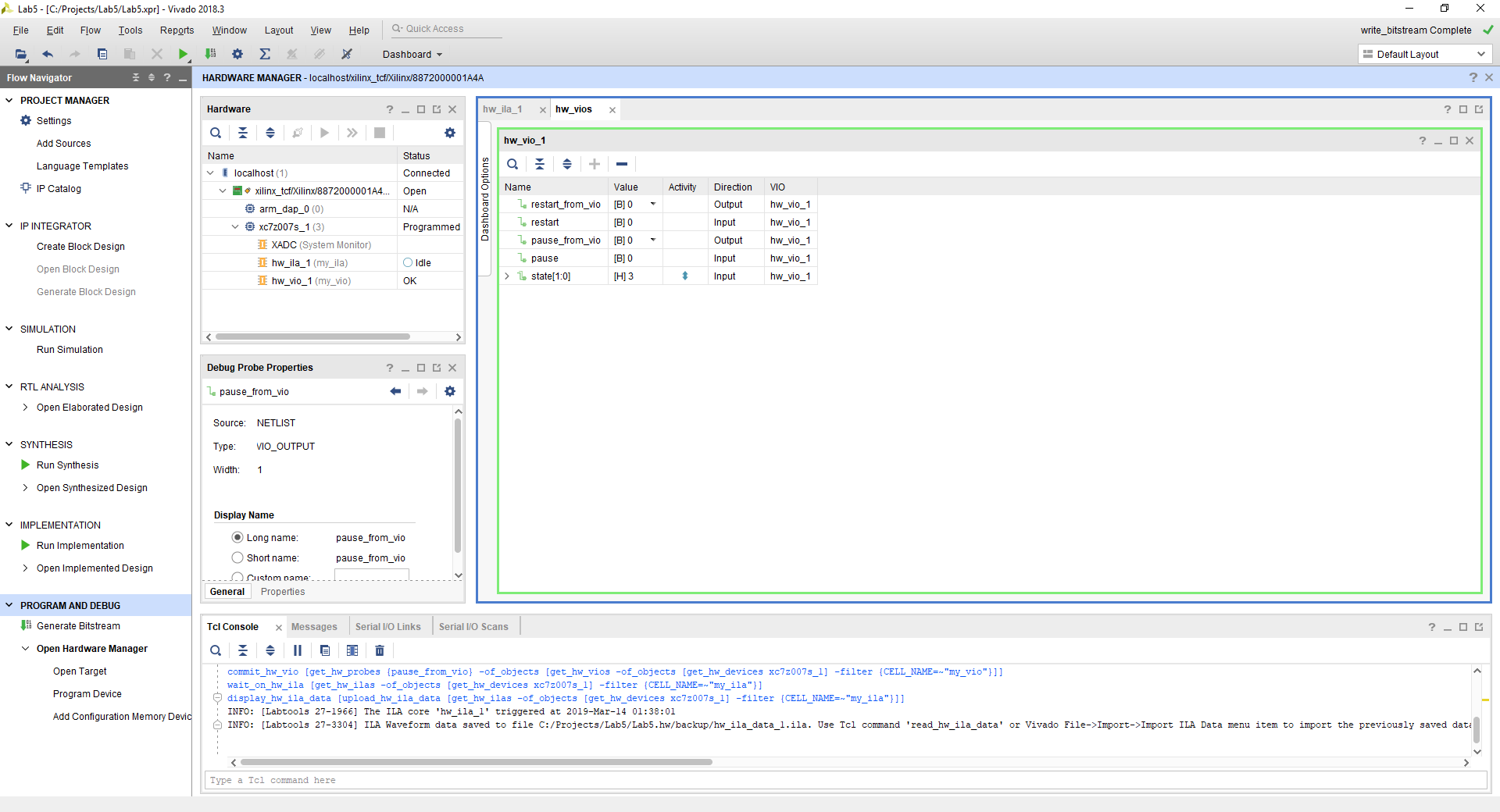


Figure 22: Action Completed

Return to the ILA tab as shown in Figure 23, where the ILA capture is now visible. Zoom in around the red cursor (which marks the trigger event) so that you can see cycle-by-cycle activity. Does it match your expectation based on your understanding of the FSM design? If the results match your expectations, you have now reached the end of the exercise. Reach out to the instructor for assistance if you do not get the expected results.

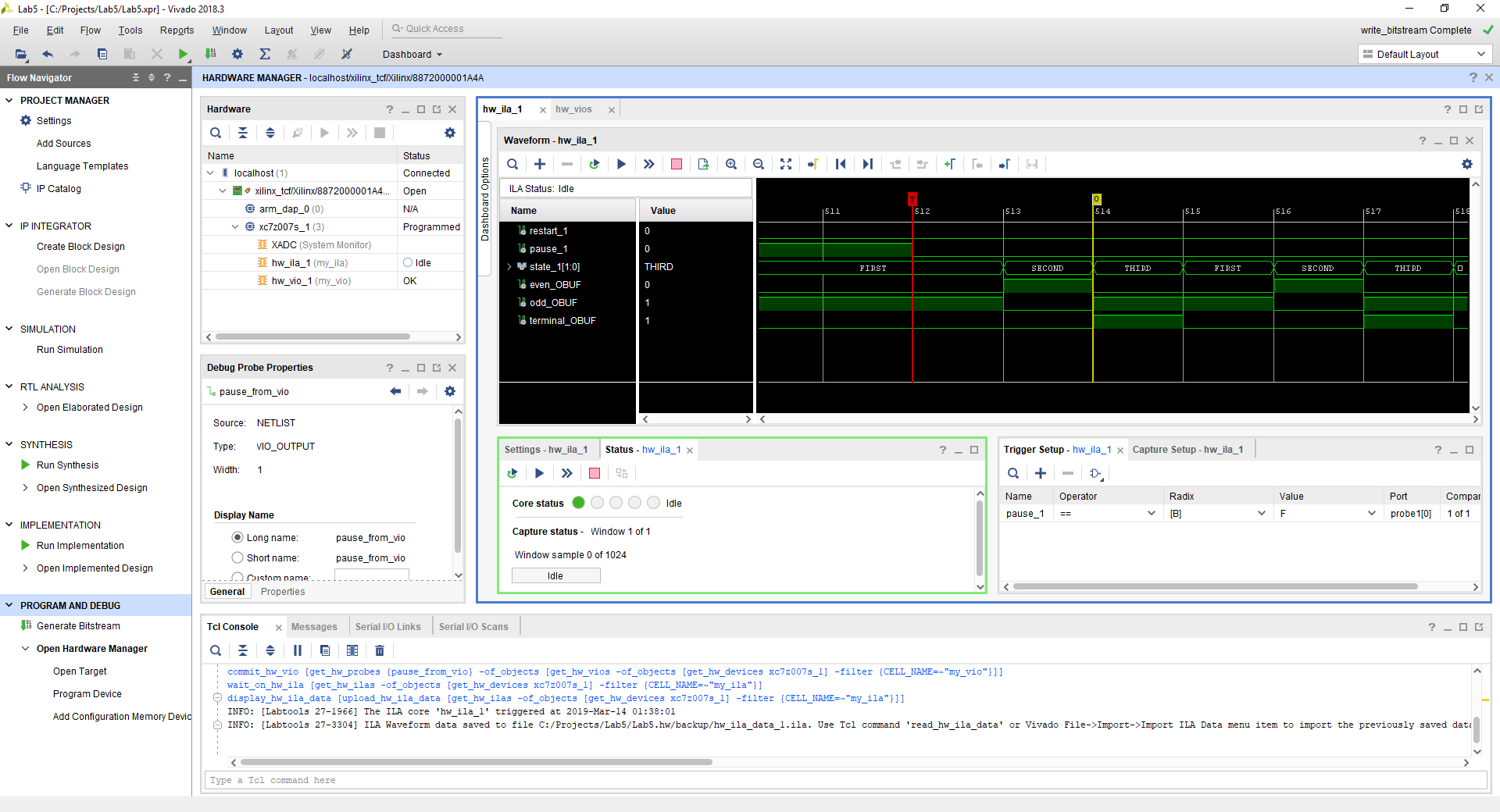


Figure 23: Expected Results

## Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. The followings are required for your demonstration: Videos and waveform screenshots showing:

1. The triggers of FIRST state remaining in the FIRST state or transitioning into the SECOND state by arming the *restart* or *pause* conditions.
2. The triggers of SECOND state remaining in the SECOND state or transitioning into the THIRD state by arming the *restart* or *pause* conditions.
3. The triggers of THIRD state remaining in the THIRD state or transitioning into the FIRST state by arming the *restart* or *pause* conditions.

You are also required to submit an archive of your project in the form of a ZIP file. Use the Vivado “File” → “Project” → “Archive...” option to create the ZIP file. Name the archive lab5\_yourlastname\_yourfirstname.zip.

You will submit your archive to the instructor through Canvas by the due date and time along with a narrated video of your demonstration. If your circuit is not completely functional by the due date, you should demonstrate and turn in what you have accomplished to receive partial credit. See the syllabus for the late penalty guideline.