

Debugging Using Hardware Analyzer

Objectives

After completing this lab, you will be able to:

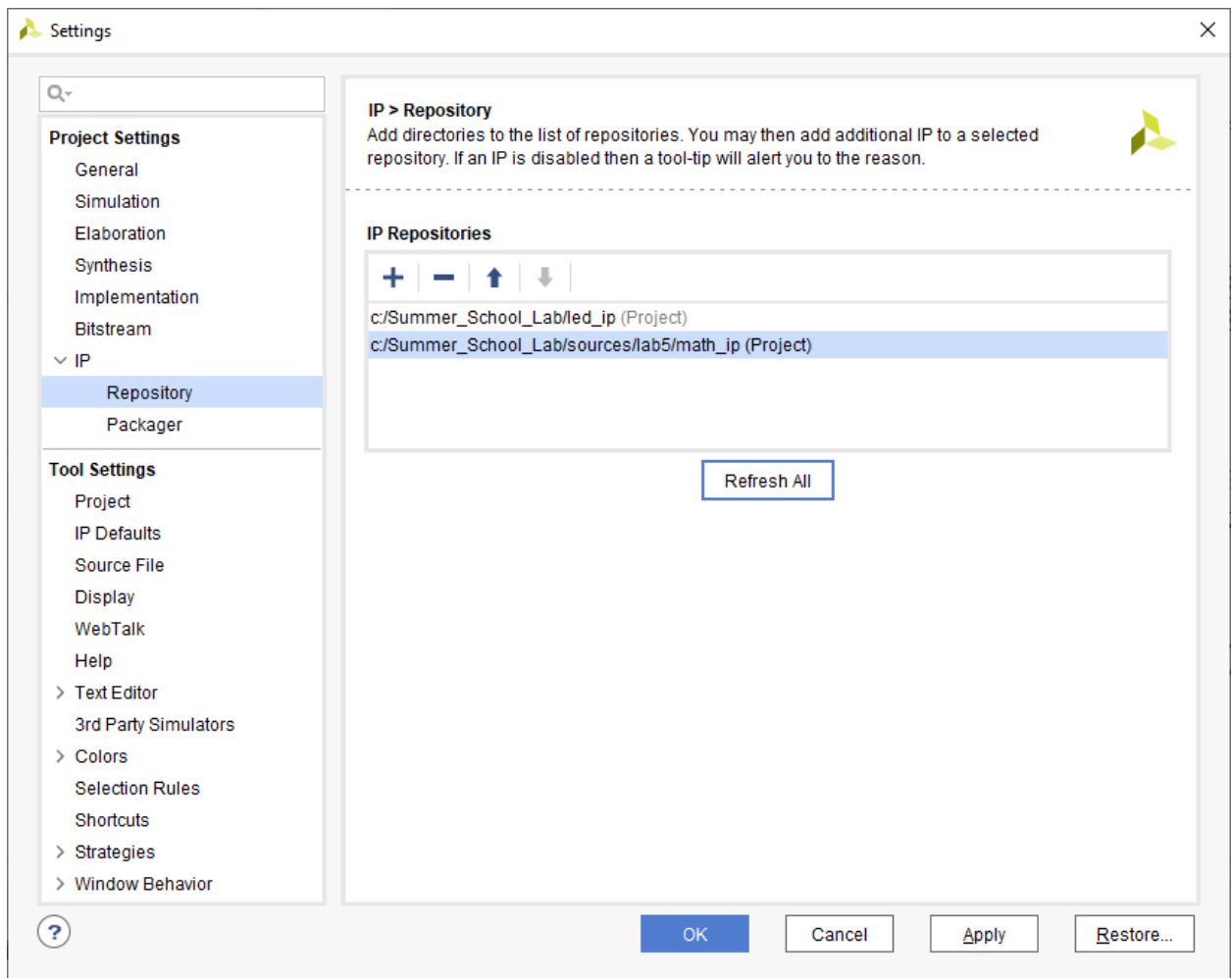
- Add a VIO core in the design
- Use a VIO core to inject stimulus to the design and monitor the response
- Mark nets as debug so AXI transactions can be monitored
- Add an ILA core in Vivado
- Perform hardware debugging using the hardware analyzer
- Perform software debugging using the SDK

Steps

Open the Project

Open the Vivado program. Open the lab1 project you created in the previous lab or use the lab3 project from the labsolution directory, and save the project as lab5. Set Project Settings to point to the IP repository provided in the sources directory.

1. Start Vivado if necessary and open either the lab3 project (lab3.xpr) you created in the previous lab or the lab3 project in the labsolutions directory using the **Open Project** link in the Getting Started page.
2. Select **File > Project > Save As ...** to open the Save Project As dialog box. Enter lab5 as the project name. Make sure that the Create Project Subdirectory option is checked, the project directory path is {labs} and click **OK**. This will create the lab5 directory and save the project and associated directory with lab5 name
3. Click **Settings** in the *Flow Navigator* pane.
4. Expand **IP** in the left pane of the *Project Settings* form and select **Repository**.
5. Click on the *plus* button of the IP Repositories panel, browse to {sources}\lab5\math_ip and click **Select**. The directory will be scanned and one IP will be detected and reported.



Specify IP Repository

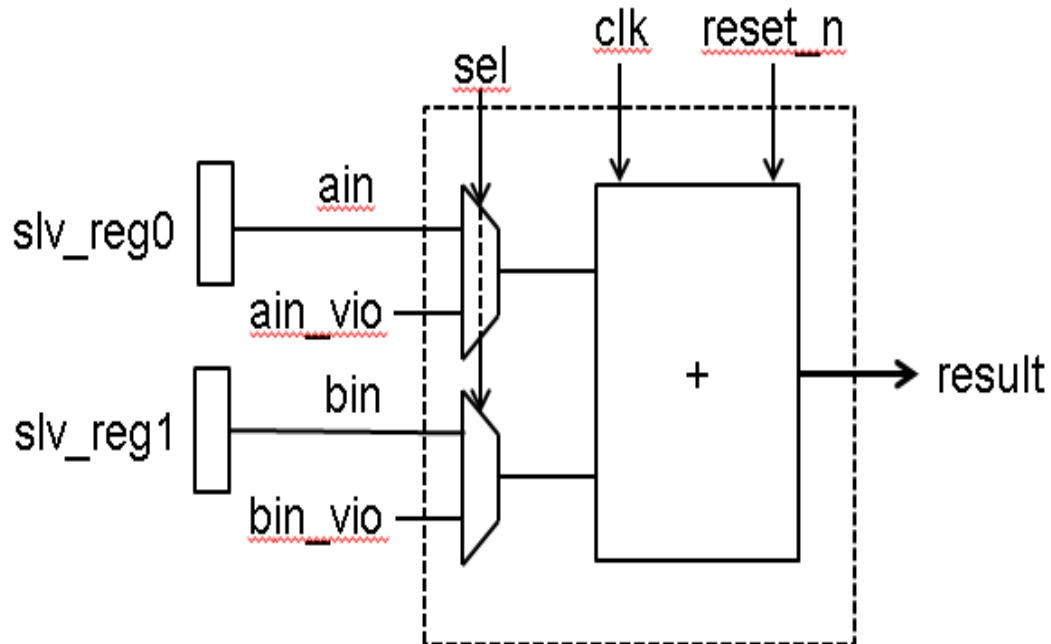
6. Click **OK** twice to close the window.

Add the Custom IP

Open the Block Design and add the custom IP to the system.

1. Click the **+** button and search for **math** in the catalog.
2. Double-click the **math_ip_v1_0** to add an instance of the core to the design.
3. Click on **Run Connection Automation**, ensure math_ip_0 and S_AXI are selected, and click **OK**.

The *Math IP* consists of a hierarchical design with the lower-level module performing the addition. The higher-level module includes the two slave registers.



Custom Core's Main Functional Block

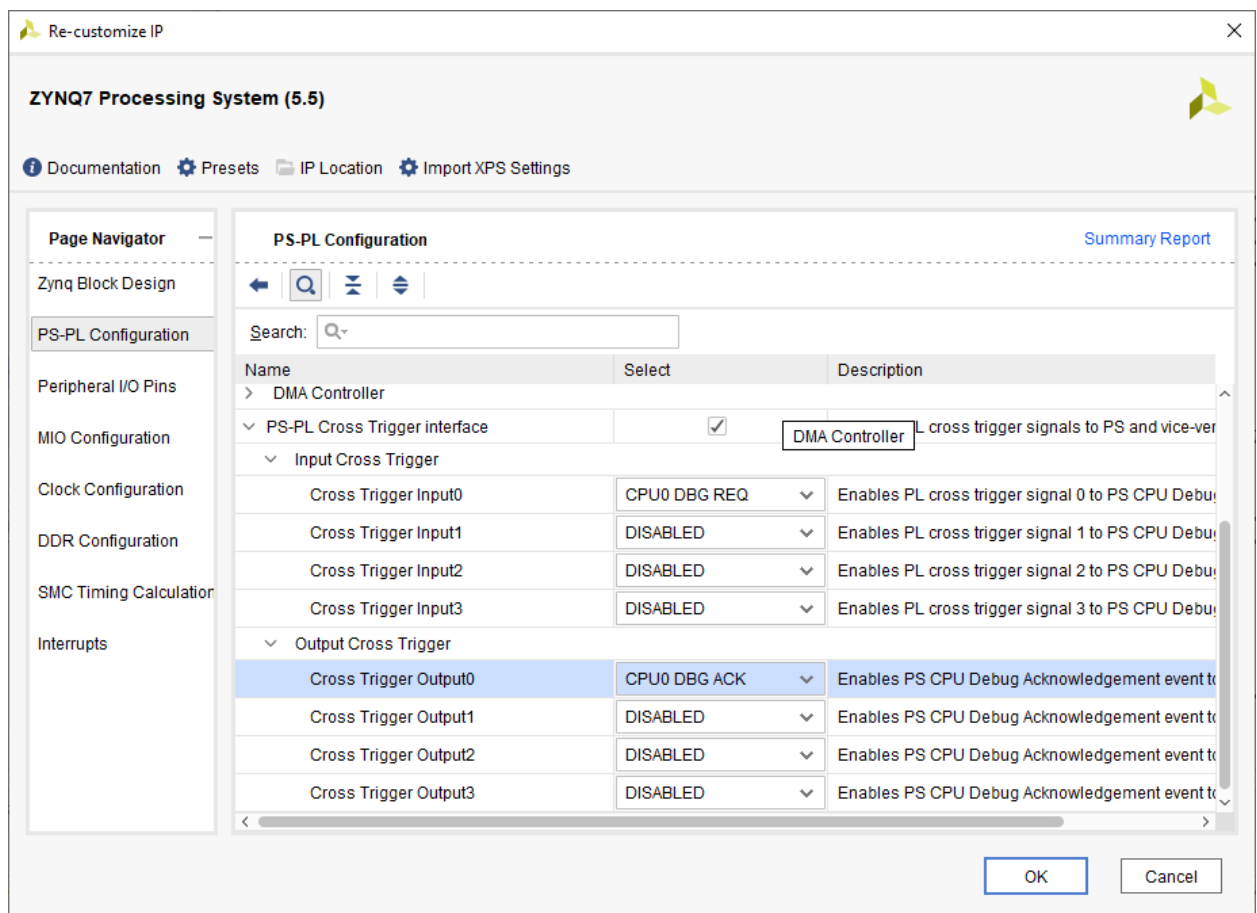
4. Delete led_ip and re-add it by click **+** and search for led_ip. Double click to add to the system.
5. Double click the led_ip_0 and configure the Led Width to 4(the same as your own board). Click OK. Re-name led_ip_0 instance to led_ip.
6. Click on **Run Connection Automation**, ensure led_ip_0 and S_AXI are selected, and click **OK**.
7. Draw LED port to LED[3:0] to make the connect.

Add the ILA and VIO Cores

We want to connect the ILA core to the LED interface. Vivado prohibits connecting ILA cores to interfaces. In order to monitor the LED output signals, we need to convert the LED interface to simple output port.

Enable cross triggering between the PL and PS

1. Double click on the *Zynq* block to open the configuration properties.
2. Click on PS-PL Configuration, and enable the *PS-PL Cross Trigger interface*.
3. Expand *PS-PL Cross Trigger interface* > *Input Cross Trigger*, and select **CPU0 DBG REQ** for *Cross Trigger Input 0*.
4. Similarly, expand *Output Cross Trigger*, and select **CPU0 DBG ACK** for *Cross Trigger Output 0* and click **OK**.



Enabling cross triggering in the Zynq processing system

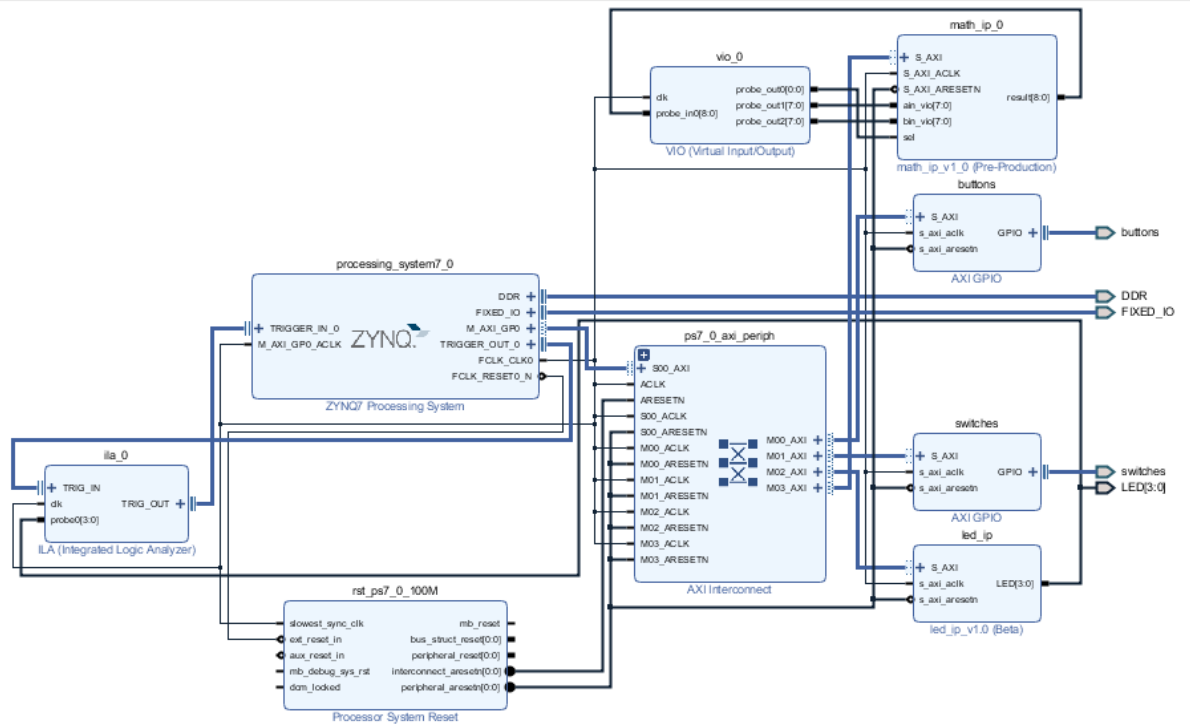
Add the ILA core and connect it to the LED output port.

1. Click the **+** button and search for **ila** in the catalog.

2. Double-click on the **ILA (Integrated Logic Analyzer)** to add an instance of it.
The *ila_0* instance will be added.
3. Double-click on the *ila_0* instance.
4. Select **Native** as the *Monitor type*.
5. Enable *Trigger Out Port*, and *Trigger In port*.
6. Select the **Probe Ports** tab, and set the **Probe Width** of *PROBE0* to **4** and click **OK**.
7. Using the drawing tool, connect the **PROBE0** port of the *ila_0* instance to the **LED** port of the *led_ip* instance.
8. Connect the **clk** port of the *ila_0* instance to the **FCLK_CLK0** port of the Zynq subsystem.
9. Connect **TRIGG_IN** of the ILA to **TRIGGER_OUT_0** of the Zynq processing system, and **TRIG_OUT** of the ILA to the **TRIGGER_IN_0**.

Add the VIO core and connect it to the math_ip ports.

1. Click the **+** button and search for **vio** in the catalog.
2. Double-click on the **VIO (Virtual Input/Output)** to add an instance of it.
3. Double-click on the *_vio_instance* to open the configuration form.
4. In the *General Options* tab, leave the *Input Probe Count* set to **1** and set the *Output Probe Count* to **3**
5. Select the *PROBE_IN Ports* tab and set the *PROBE_IN0* width to **9**.
6. Select the *PROBE_OUT Ports* tab and set *PROBE_OUT0* width to **1** , *PROBE_OUT1* width to **8** , and *PROBE_OUT2* width to **8**.
7. Click **OK**.
8. Connect the VIO ports to the math instance ports as follows:
PROBE_IN -> result
PROBE_OUT0 -> sel
PROBE_OUT1 -> ain_vio
PROBE_OUT2 -> bin_vio
9. Connect the **CLK** port of the *vio_0* to FCLK_CLK0 net.
10. The block diagram should look similar to shown below.

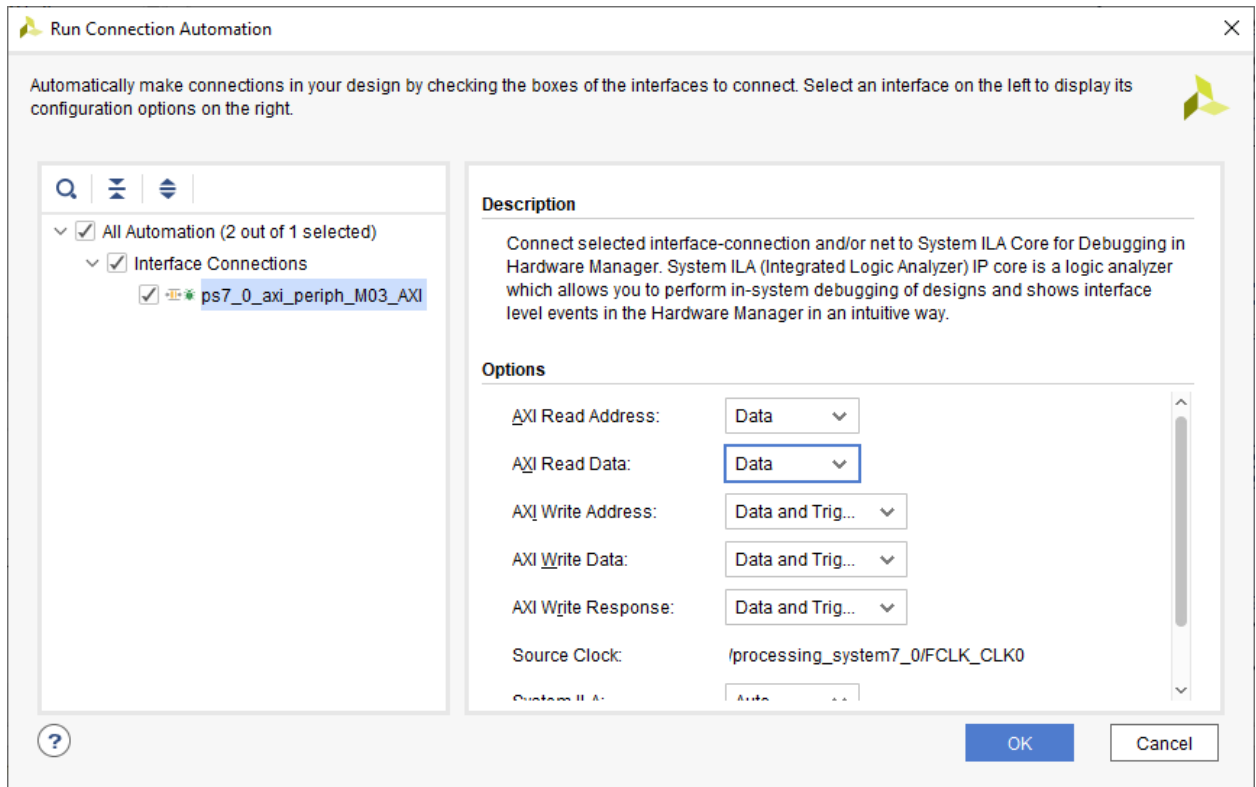


VIO added and connections made

Mark Debug the S_AXI connection between the AXI Interconnect and math_0 instance. Validate the design.

1. Select the **S_AXI** connection between the AXI Interconnect and the *math_ip_0* instance .
2. Right-click and select **Debug** to monitor the AXI4Lite transactions.
3. Click the **Run Connection Automation** link to see the form where you can select the desired channels to monitor.
4. Change *AXI Read Address* and *AXI Read Data* channels to **Data** since we will not trigger any signals of those channels. And click OK.

This saves resources being used by the design.



Selecting channels for debugging

Notice that a system_ila IP instance got added and the M03_AXI <-> S_AXI connection is connected to its SLOT_0_AXI interface.

The block diagram should look as shown below.

2. Click **OK** to export and **Yes** to overwrite the previous project created by lab1.
3. Launch SDK by clicking **File > Launch SDK** and click **OK**.
4. Right-click on the **lab3** and **lab3_bsp_0** and **system_wrapper_hw_platform_2** projects in the Project Explorer view and select **close project**. The name may be different.

Create an empty application project named lab5, and import the provided lab5.c file.

1. Select **File > New > Application Project**.
2. In the *Project Name* field, enter **lab5** as the project name, leave all other settings to their default's and click **Next** (a new BSP will be created) .
3. Select the **Empty Application** template and click **Finish**.

The lab5 project will be created in the Project Explorer window of the SDK.

4. Select **lab5 > src** in the project view, right-click, and select **Import**.
5. Expand the **General** category and double-click on **File System**.
6. Browse to the **{sources}\lab5** folder.
7. Select **lab5.c** and click **Finish**.

A snippet of the part of the source code is shown in the following figure. It shows that two operands are written to the custom core, the result is read, and printed out. The write transaction will be used as a trigger condition in the Vivado Logic Analyzer.

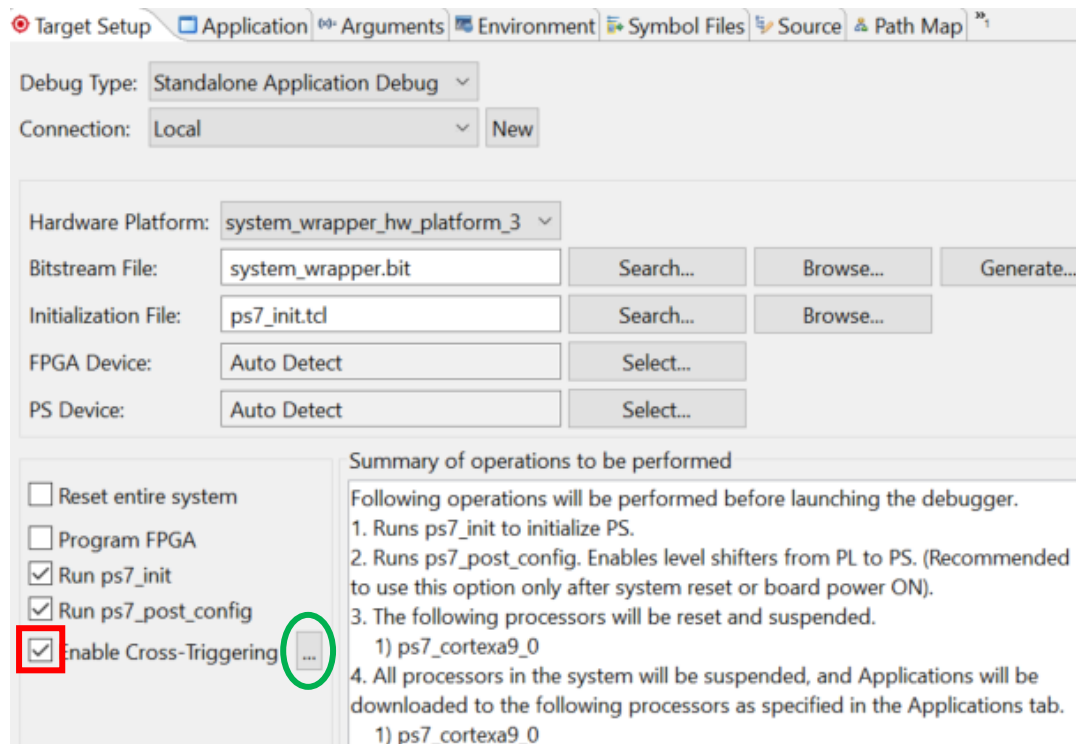
```

6= int main (void)
7 {
8     XGpio sws, btns;
9     int i, sws_check, btns_check;
10
11     xil_printf("-- Start of the Program --\r\n");
12     xil_printf("-- Press any of BTN0-BTN3 to see corresponding output on LEDs --\r\n");
13     xil_printf("-- Set slides switches to 0x03 to exit the program --\r\n");
14
15     // AXI GPIO switches Initialization
16     XGpio_Initialize(&sws, XPAR_SWITCHES_DEVICE_ID);
17     XGpio_SetDataDirection(&sws, 1, 0xffffffff); // input
18     // AXI GPIO buttons Initialization
19     XGpio_Initialize(&btns, XPAR_BUTTONS_DEVICE_ID);
20     XGpio_SetDataDirection(&btns, 1, 0xffffffff); // input
21
22     Xil_Out32(XPAR_MATH_IP_0_BASEADDR, 0x12);
23     Xil_Out32(XPAR_MATH_IP_0_BASEADDR+4, 0x34);
24     i=Xil_In32(XPAR_MATH_IP_0_BASEADDR);
25     xil_printf("result=%x\r\n",i);
26
27     while (1)
28     {
29         btns_check = XGpio_DiscreteRead(&btns, 1);
30         LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, btns_check);
31
32         sws_check = XGpio_DiscreteRead(&sws,1);
33         xil_printf("DIP_switches: %d\r\n",sws_check);
34         if((sws_check & 0x03)==0x03)
35             break;
36         for (i=0; i<99999999; i++); // delay loop
37     }
38     xil_printf("-- End of Program --\r\n");
39
40     return 0;
41 }

```

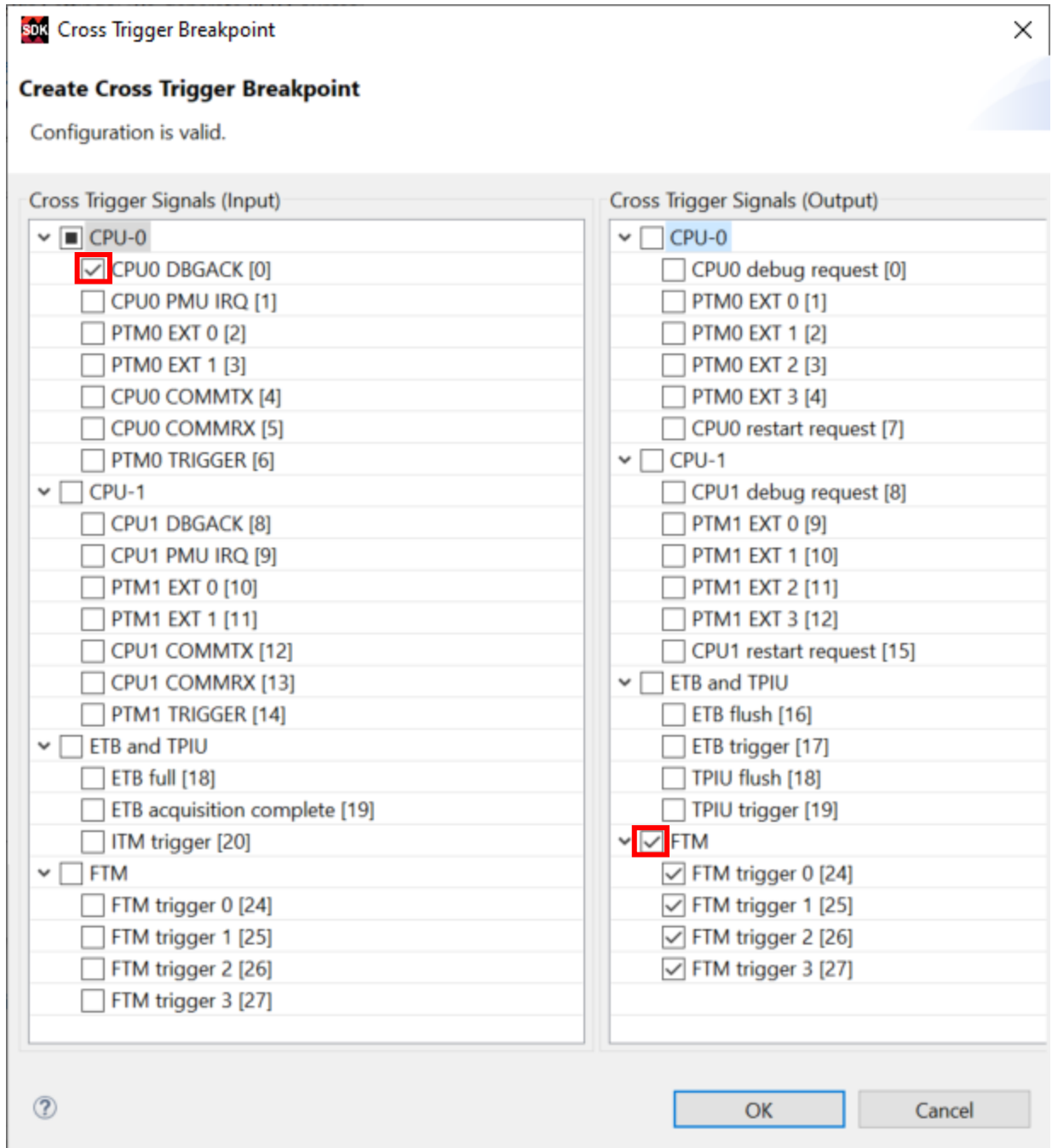
Source Code snippet

8. Right click on *lab5*, and select **Debug As > Debug Configurations**
9. Double click on Xilinx C/C++ application (System Debugger) to create a new configuration (*lab5 Debug will be created*), and in the *Target_Setup* tab, check the **Enable Cross-Triggering** option , and click the Browse button.



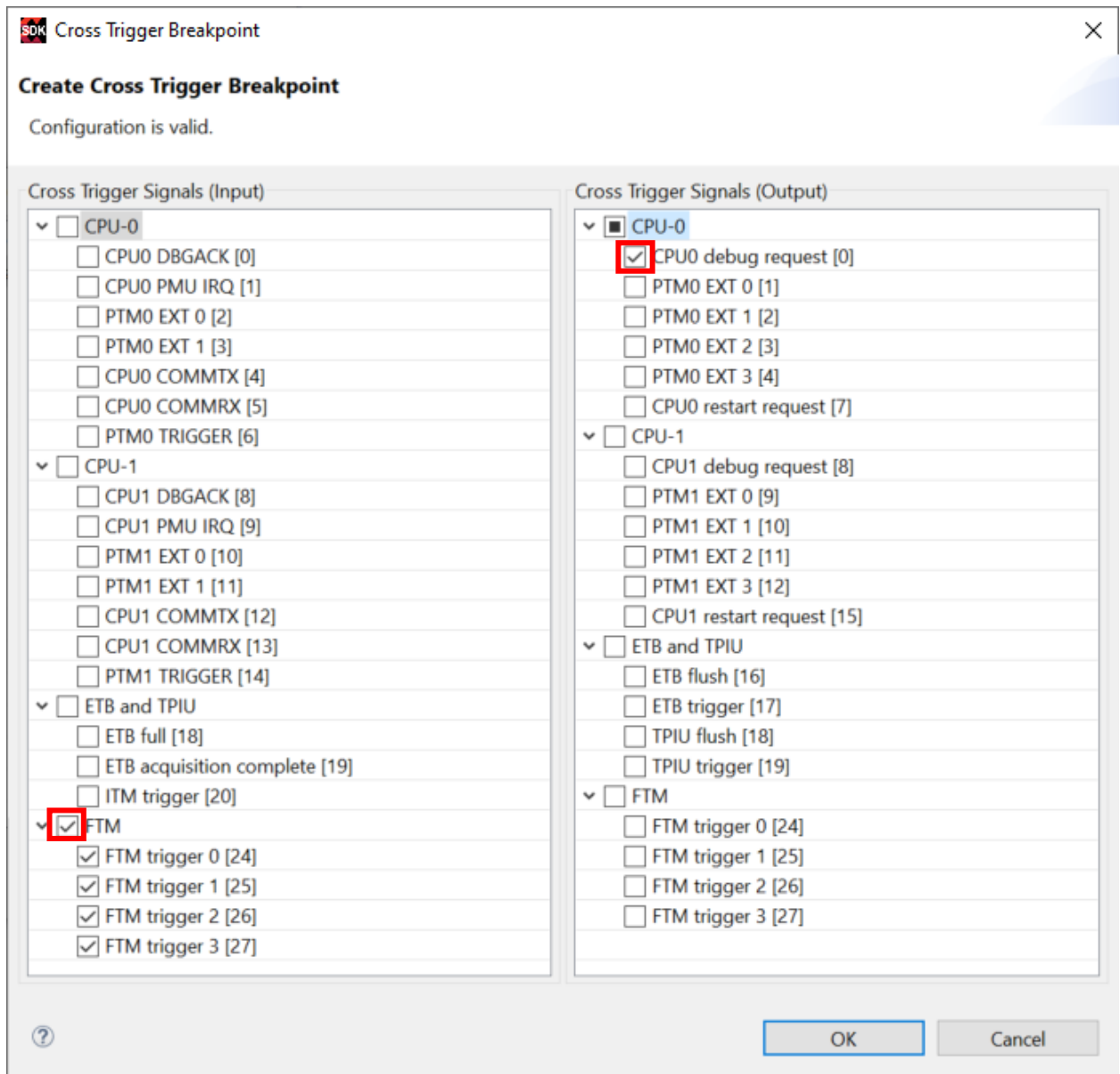
Enable cross triggering in the software environment

10. When the *Cross Trigger Breakpoints* dialog box opens, click **Create**
11. Select the options as shown below and click **OK** to set up the cross-trigger condition for *Processor to Fabric*.



Enabling CPU0 for request from PL

12. In the *Cross Trigger Breakpoints* dialog box click **Create** again.
13. Select the options as shown below and click **OK** to set up the cross trigger condition for *Fabric to Processor*.





Enabling CPU0 for request to PL

14. Click **OK** , then click **Apply**, then **Close**

Test in Hardware

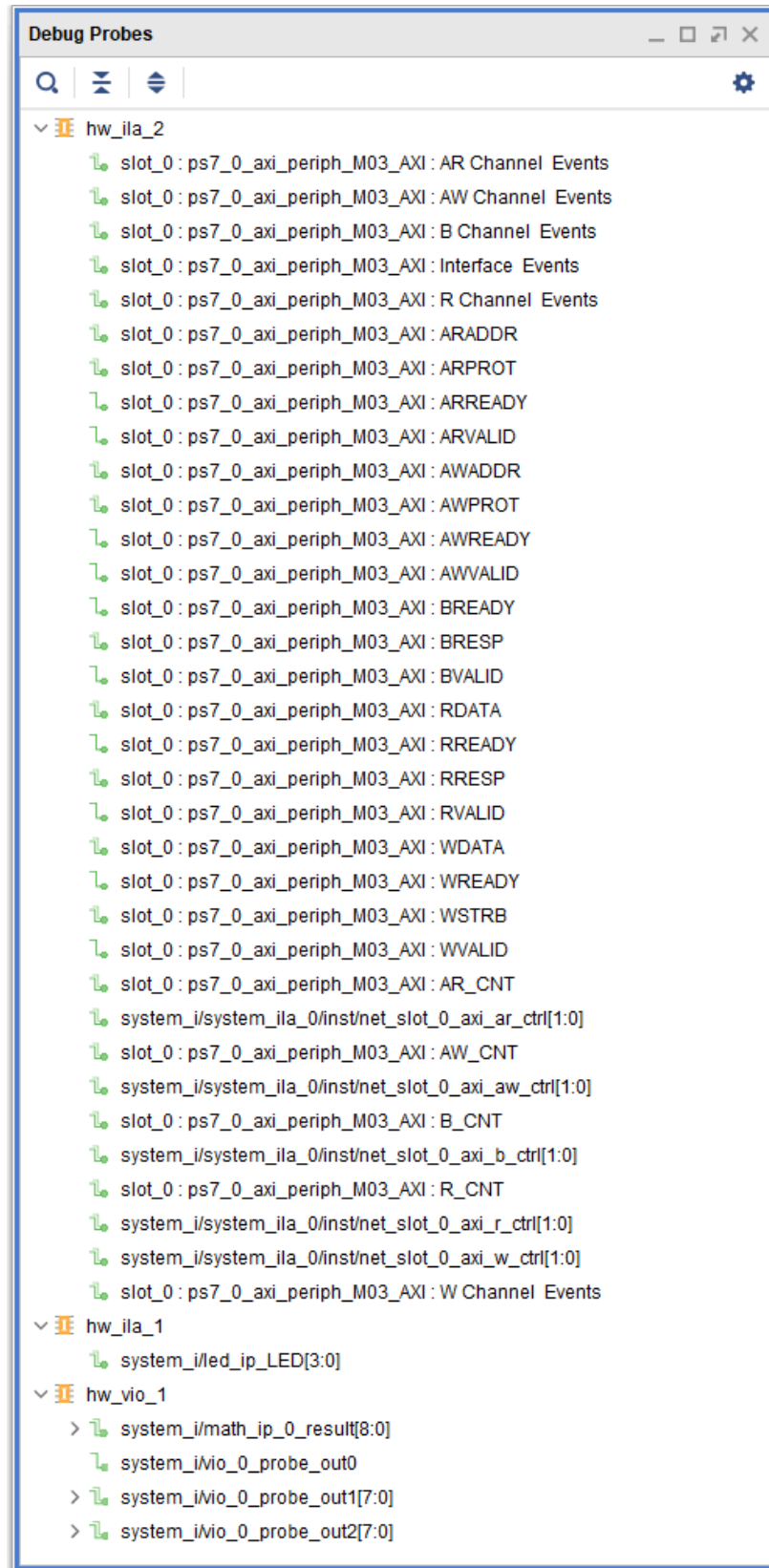
Connect and power up the board. Download the bitstream into the target device. Start the debug session on lab5 project. Switch to the Debug perspective and establish serial communication.

1. Connect and power up the board.
2. Select **Xilinx > Program FPGA** and click **Program**
3. Select the **lab5** project in *Project Explorer*, right-click and select **Debug As > Launch on Hardware** (System Debugger) to download the application, execute ps7_init. (If prompted, click **Yes** to switch to the Debug perspective.) The program execution starts and suspends at the entry point.
4. Select the  **Terminal** tab. If it is not visible then select **Window > Show view > Terminal** tab.
5. Click on  and select the appropriate COM port (depending on your computer), and configure it as previous.

Start the hardware session from Vivado.

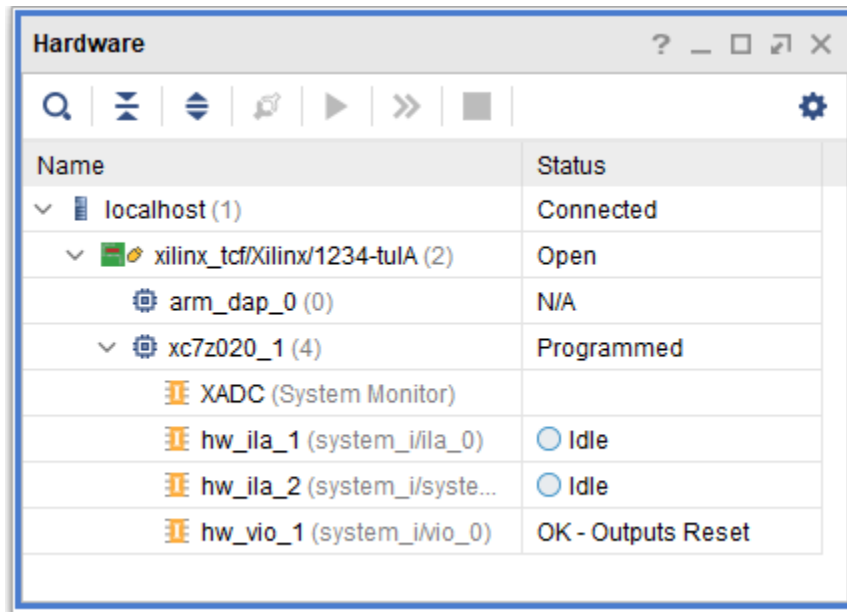
1. Switch to Vivado.
2. Click on **Open Hardware Manager** from the *Program and Debug* group of the *Flow Navigator* pane to invoke the analyzer.
3. Click on the **Open Target > Auto connect** to establish the connection with the board.
4. Select **Window > Debug Probes**

The hardware session will open showing the **Debug Probes** tab in the **Console** view.



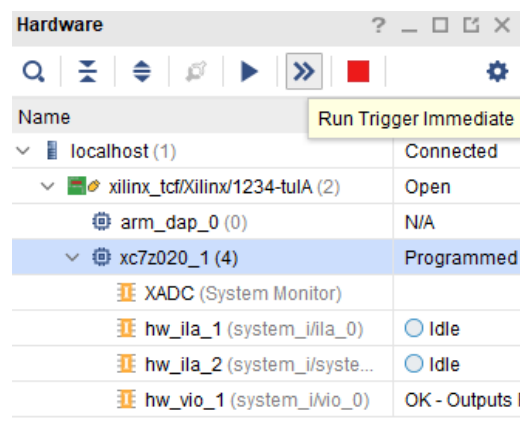
Debug probes

The hardware session status window also opens showing that the FPGA is programmed (we did it in SDK), there are three cores out of which the two ila cores are in the idle state.



Hardware session status

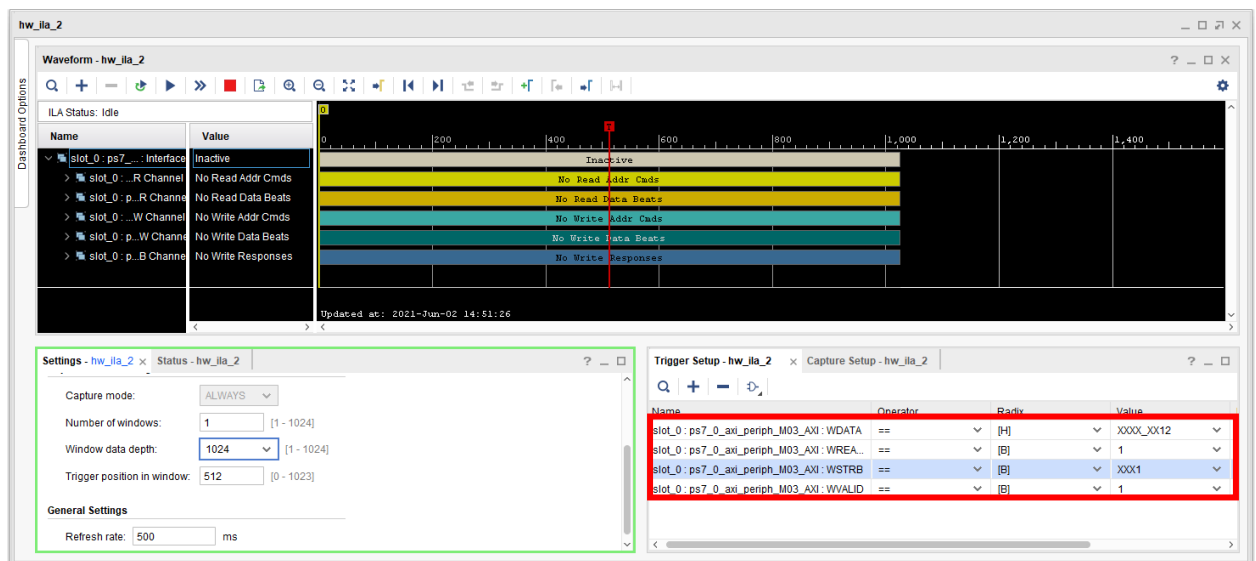
5. Select the XC7Z020, and click on the **Run Trigger Immediate** button to see the signals in the waveform window.



Opening the waveform window

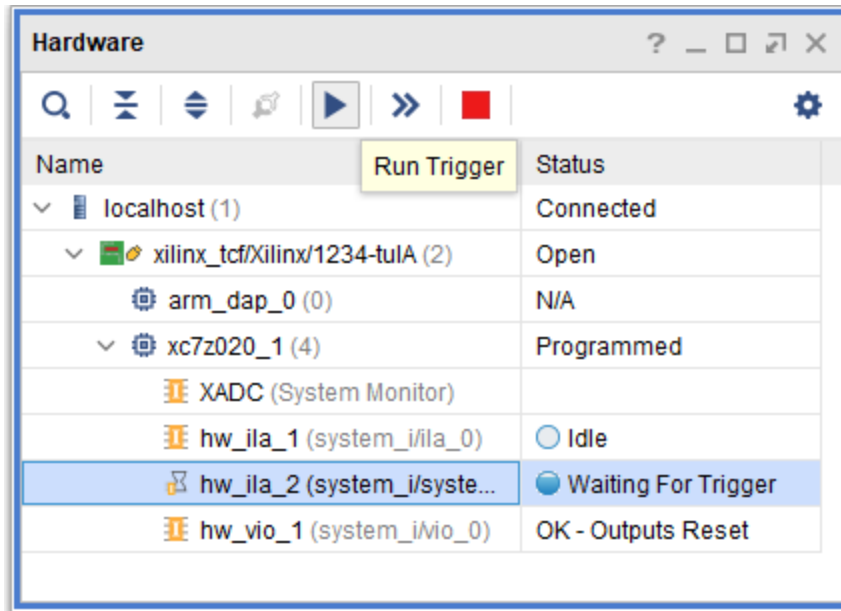
Setup trigger conditions to trigger on a write transaction (WSTRB) when the desired data (WDATA) of XXXX_XX12 is written. The transaction takes place when WVALID and WREADY are equal to 1.

1. Click on the *hw_ila_2* tab to select it. In the **Debug Probes** window, under *hw_ila_2*, drag and drop the **WDATA** signal to the *ILA Basic Trigger setup* window .
2. Set the value to **XXXX_XX12** (HEX) (the value written to the math_0 instance at line 24 of the program).
3. Similarly, add **WREADY**,** WSTRB, and WVALID** signals to the *ILA Basic Trigger setup* window.
4. Change the radix to binary for WSTRB, and change the value from **xxxx** to **xxx1**
5. Change the value of **WVALID** and **WREADY** to 1.
6. Set the trigger position of the *hw_ila_2* to **512** in the *Settings – hw_ila_2*



Setting up the ILA

7. Similarly, set the trigger position in the *Settings – hw_ila_1* tab to **512**.
8. Select **hw_ila_2** in the *Hardware* window and click on the **Run Trigger** button and observe that the *hw_ila_2* core is armed and showing the status as **Waiting For Trigger**.



Hardware analyzer running and in capture mode

9. Switch to SDK.
10. Near line 25 (right click in the margin and select *Show Line Numbers* if necessary), double click on the left border on the line where `xil_printf` statement is (before the `while (1)` statement) is defined in the `lab5.c` window to set a breakpoint.

```

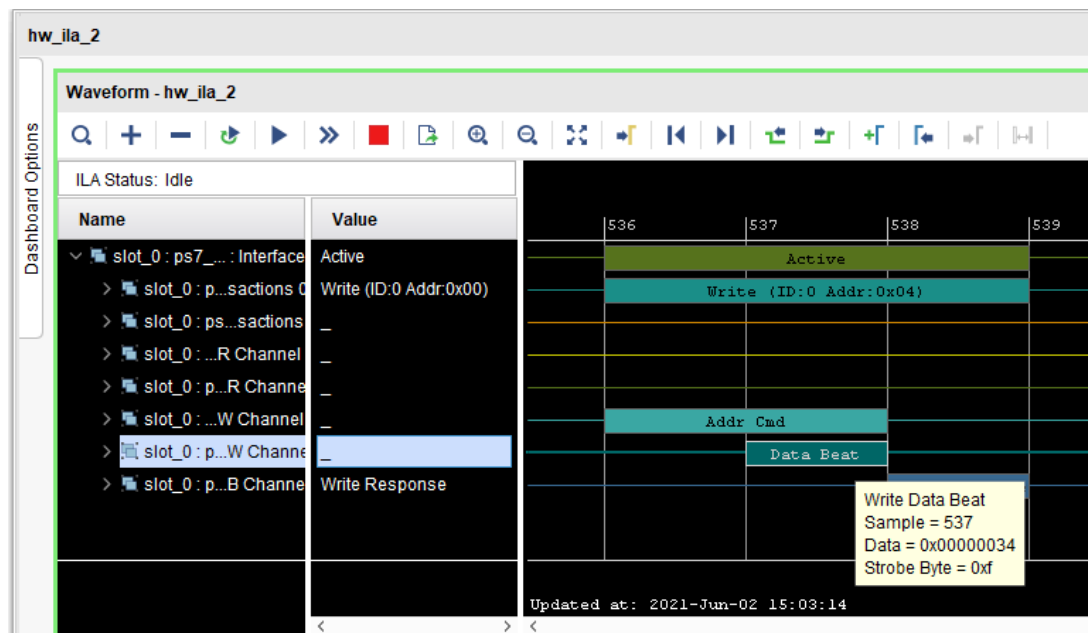
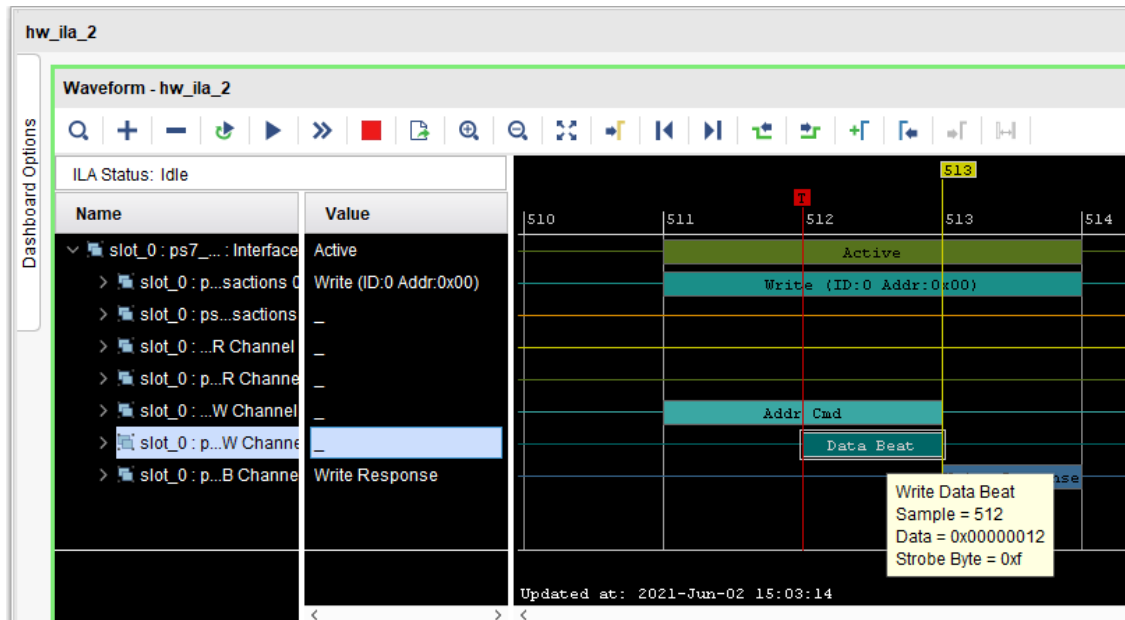
21
22     Xil_Out32(XPAR_MATH_IP_0_BASEADDR, 0x12);
23     Xil_Out32(XPAR_MATH_IP_0_BASEADDR+4, 0x34);
24     i=Xil_In32(XPAR_MATH_IP_0_BASEADDR);
25     xil_printf("result=%x\r\n",i);
26

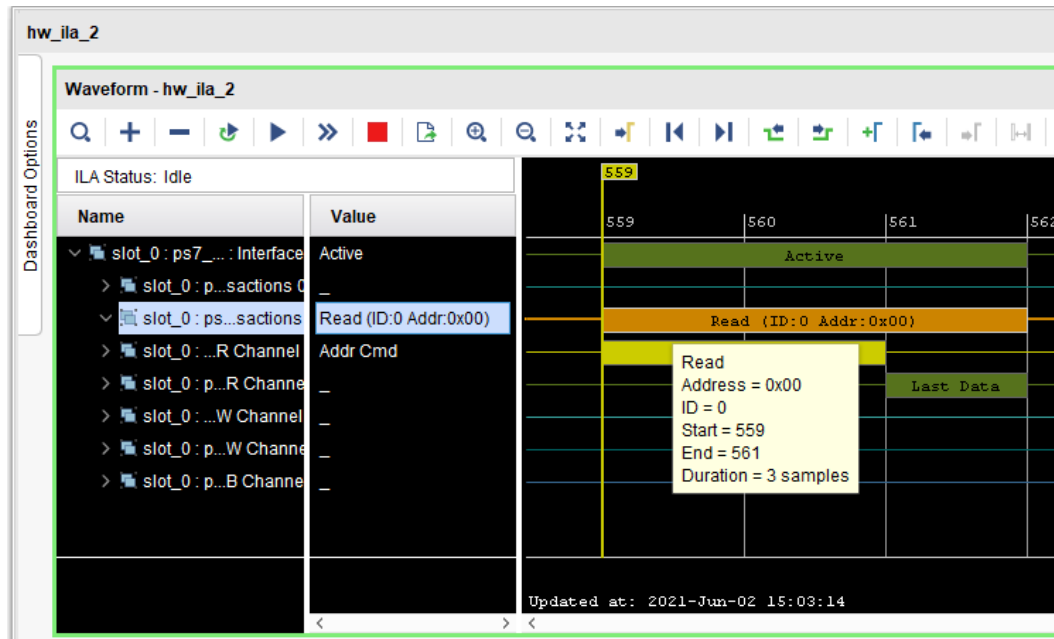
```

Setting a breakpoint

11. Click on the **Resume** (🟢) button to execute the program and stop at the breakpoint.
12. In the Vivado program, notice that the **hw_ila_2** status changed from *Waiting for Trigger* to *Idle*, and the waveform window shows the triggered output (select the `hw_ila_data_2.wcfg` tab if necessary).
13. Move the cursor to closer to the trigger point and then click on the 🔍 button to zoom at the cursor. Click on the **Zoom In** button couple of times to see the

activity near the trigger point. Similarly, you can see other activities by scrolling to right as needed.





Zoomed waveform view of the three AXI transactions

Observe the following:

Around the 512th sample WDATA being written is 0x012 at offset 0 (AWADDR=0x0). At the 537th sample, offset is 0x4 (AWADDR), and the data being written is 0x034. At the 559th sample, data is being read from the IP at the offset 0x0 (ARADDR), and at 561st mark the result (0x46) is on the RDATA bus.

14. You also should see the following output in the Terminal console.

```

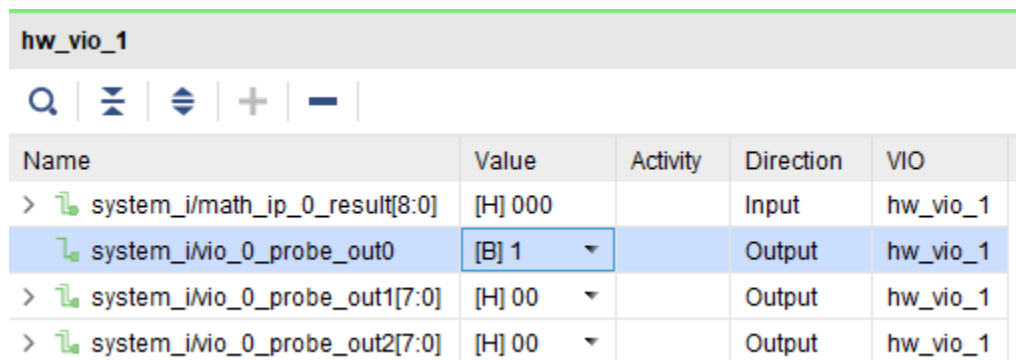
SDK Log  Memory  Terminal 1
Serial: (COM5, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
-- Start of the Program --
-- Press any of BTN0-BTN3 to see corresponding output on LEDs --
-- Set slides switches to 0x03 to exit the program --





```

Terminal Output

In Vivado, select the VIO Cores related from the Dashboard Options windows, set the `vio_1_probe_out0` so `math_ip`'s input can be controlled manually through the VIO core. Try entering various values for the two operands and observe the output on the `math_ip_1_result` port in the Console pane.

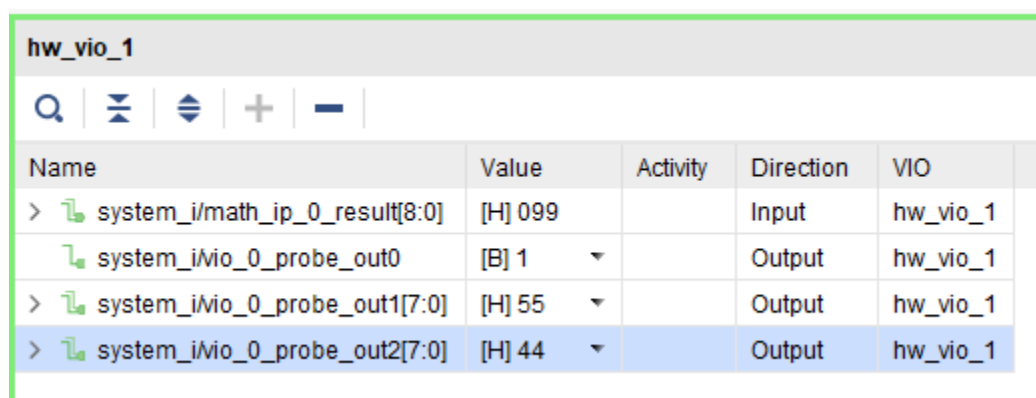
1. Select the **hw_vio_1** core in the *Dashboard Options* panel.
2. Click on the **+** button and select all signals to stimulate and monitoring. Change the **vio_0_probe_out0** value to **1** so the `math_ip` core input can be controlled via the VIO core.







Name	Value	Activity	Direction	VIO
>  system_i/math_ip_0_result[8:0]	[H] 000		Input	hw_vio_1
 system_i/vio_0_probe_out0	[B] 1		Output	hw_vio_1
>  system_i/vio_0_probe_out1[7:0]	[H] 00		Output	hw_vio_1
>  system_i/vio_0_probe_out2[7:0]	[H] 00		Output	hw_vio_1

VIO probes

3. Change **vio_0_probe_out1** value to **55** (in Hex), and similarly, **vio_0_probe_out2** value to **44** (in Hex). Notice that for a brief moment a blue-colored up-arrow will appear in the Activity column and the result value changes to **099** (in Hex).



Name	Value	Activity	Direction	VIO
>  system_i/math_ip_0_result[8:0]	[H] 099		Input	hw_vio_1
 system_i/vio_0_probe_out0	[B] 1		Output	hw_vio_1
>  system_i/vio_0_probe_out1[7:0]	[H] 55		Output	hw_vio_1
>  system_i/vio_0_probe_out2[7:0]	[H] 44		Output	hw_vio_1

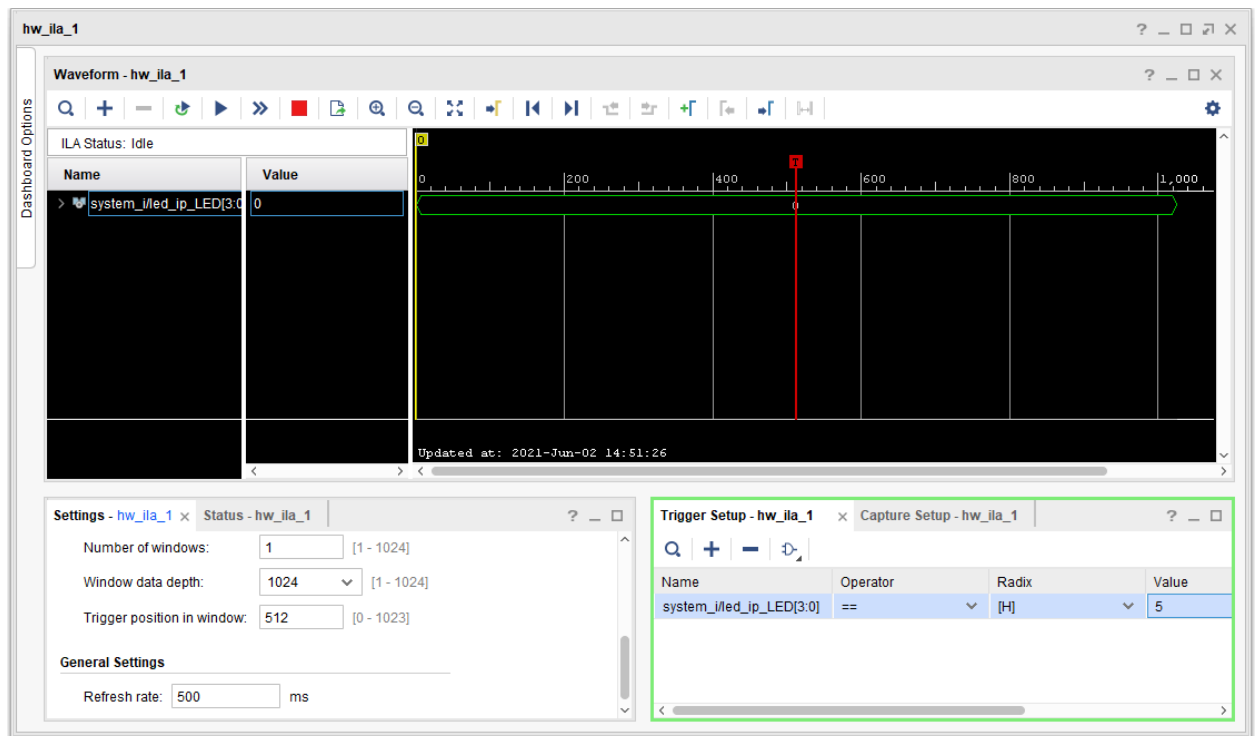
Input stimuli through the VIO core's probes

4. Try a few other inputs and observe the outputs.

5. Once done, set the `vio_0_probe_out0` to **0** to isolate the vio interactions with the math_ip core.

Setup the ILA core (`hw_ila_1`) trigger condition to `0x2` for the PYNQ-Z1/PYNQ-Z2. Make sure that the switches on the board are not set at `x3` (for PYNQ-Z1/PYNQ-Z2). Set the trigger equation to be `==`, and arm the trigger. Click on the Resume button in the SDK to continue executing the program. Change the switches and observe that the hardware core triggers when the preset condition is met.

1. Select the **hw_ila_1** in the *Dashboard Options* panel.
2. Add the LEDs to the *Basic Trigger Setup*, and set the trigger condition of the `hw_ila_1` to trigger at LED output value equal to **0x5** for the PYNQ-Z1/PYNQ-Z2.



Setting up Trigger for `hw_ila_1`

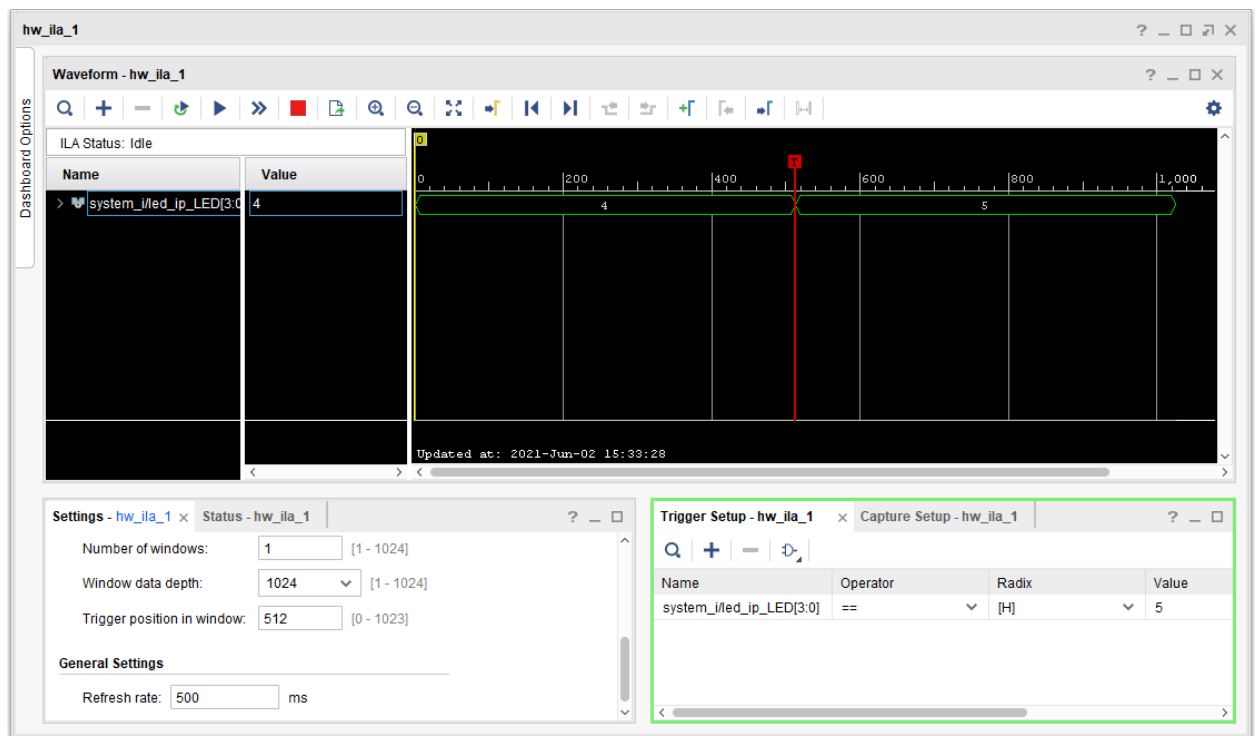
3. Ensure that the trigger position for the `hw_ila_1` is set to **512**.

Make sure that the switches are not set to 11 (PYNQ-Z1/PYNQ-Z2) as this is the exit pattern.

4. Right-click on the *hw_ila_1* in the *hardware* window, and arm the trigger by selecting **Run Trigger**.

The hardware analyzer should be waiting for the trigger condition to occur.

5. In the SDK window, click on the *Resume* button.
6. Press the push-buttons and see the corresponding LED turning ON and OFF.
7. When the condition is met(Press 5), the waveform will be displayed.



ILA waveform window after Trigger

Cross trigger a debug session between the hardware and software

1. In Vivado, select *hw_ila_1*
2. In the ILA properties, set the *Trigger mode* to **BASIC_OR_TRIGG_IN** , and the *TRIG_OUT* mode to **TRIGGER_OR_TRIG_IN**

3. In SDK, in the C/C++ view, relaunch the software by right clicking on the lab5 project, and selecting *Debug As > Launch on Hardware (System Debugger)*.

Click **OK** if prompted to reset the processor. The program will be loaded and the execution will suspend at the entry point


4. Arm the *hw_ila_1* trigger.
5. In SDK continue execution of the software to the next breakpoint (line 25).

When the next breakpoint in SDK is reached, return to Vivado and notice the ILA has triggered.

Trigger the ILA and cause the software to halt

1. Click Step Over (F6) button twice to pass the current breakpoint
2. Arm the *hw_ila_1* trigger
3. Resume the software (F8) until it enters the while loop
4. Verify it is executing by toggling the dip switches
5. In Vivado, arm the *hw_ila_1* trigger
6. Press the push-buttons to 0x5, and notice that the application in SDK will break at some point (This point will be somewhere within the while loop)
7. Click on the **Resume** button

The program will continue execution. Flip switches until it is 0x03.

8. Click the Disconnect button () in the SDK to terminate the execution.
9. Close the SDK by selecting **File > Exit**.
10. Close the hardware session by selecting **File > Close Hardware Manager**. Click **OK**.
11. Close Vivado program by selecting **File > Exit**.
12. Turn OFF the power on the board.

Conclusion

In this lab, you added a custom core with extra ports so you can debug the design using the VIO core. You instantiated the ILA and the VIO cores into the design. You used Mark Debug feature of Vivado to debug the AXI transactions on the custom peripheral. You then opened the hardware session from Vivado, setup various cores, and verified the design and core functionality using SDK and the hardware analyzer.