

Education Objective

The educational objective of this demonstration is to become familiar with the Signal Tap logic analysis tool within the Quartus II toolset.

Background

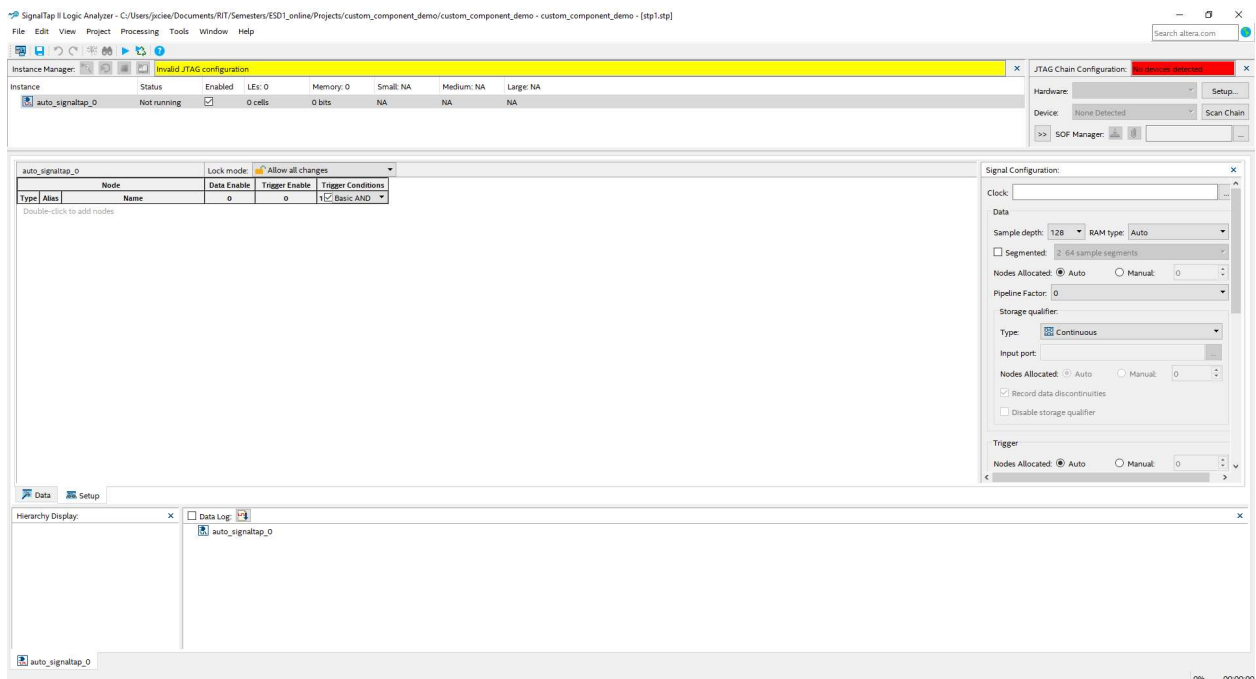
Signal Tap is a logic analysis tool (a virtual logic analyzer) that allows you to observe design signals while the code is executing on the FPGA. This tool is extremely useful for debugging embedded systems because analysis occurs while the NIOS II is interfacing with the hardware. Additionally, NIOS II/hardware interfaces can be observed. While Modelsim should still be used to simulate hardware modules before they are integrated into the system, without a NIOS II Modelsim model, we are unable to simulate real processor/peripheral interactions.

A logic analyzer (physical or virtual) takes a “snapshot” of signal values at a regular interval and displays them. It is important to remember that, unlike an oscilloscope, a logic analyzer does not measure any timing data. For example, if a signal is low at sample time 1 and high at sample time 2, all we know is that the signal changed sometime between the two samples. We don’t know when or how many times it changed.

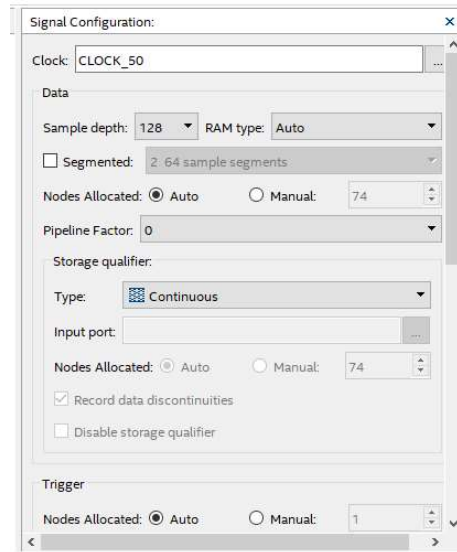
For this demonstration we will use the custom_component_demo project from last week.

Demonstration Procedure

1. Open your custom_component_demo project in Quartus II.
2. Select **file > new > SignalTap II Logic Analyzer File** and click okay.
3. You should see the following window pop up.



4. In the area entitled [Signal Configuration](#) you will set up the parameters for sampling. A logic analyzer uses a signal in the system to determine when to sample and take a “snapshot”. Most often this is a system clock signal. We will use the 50Mhz clock, which will give us a sample period of 20 ns.
5. In the [Signal Configuration](#) area, click on the ... next to the blank named [Clock](#). In the popup node finder window, pull down the box next to [Fitter](#) and choose [Pins:all](#) and then click on [List](#). There should be four entries in the table. Click on [CLOCK_50](#) and then > to move it to the other side of the table. Then click [ok](#). You should see the following:



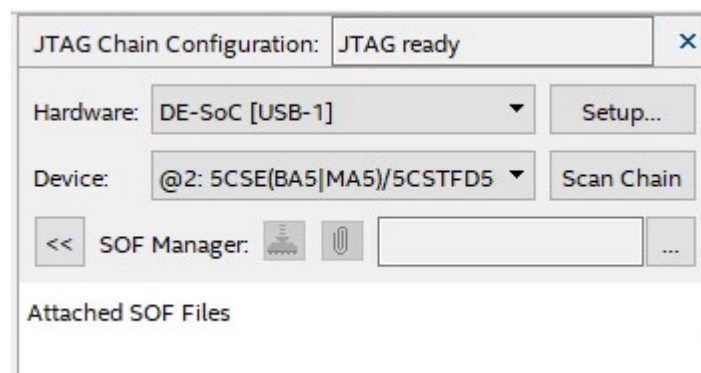
6. Now we need to add the internal signals that we would like to look at. Double click in the [setup](#) window (the large blank area) and a node finder window will pop up. Pull down the box next to [Fitter](#) and choose [Design Entity \(all names\)](#). Click on [List](#).
7. There will be 8 entries in the table, one of which is [nios_system:u0](#). Click to expand this and scroll down to find [custom_ip:my_custom_ip_0](#). Note that all of the ip from the system, including the NIOS II, are listed here. Because these are soft ip, you can monitor their signals and registers if you would like. For this demo we are going to focus on the custom_component.
8. Click to expand the [custom_ip:my_custom_ip_0](#) and move the following signals to the [Nodes Found](#) area:
 - a. Invalid_export
 - b. Irq
 - c. Registers
 - d. write
 - e. Address
 - f. Ext_addr_export
 - g. Ext_data_export
 - h. Writedata


Click [insert](#) and then [close](#).

9. The next step is to set up a trigger event. There are a lot of options for triggering, but we will start with a simple one. First click in the **Trigger Enable** boxes to disable all but the write signal. In the Trigger Conditions column, pull down and choose **Comparison**. Right click in the cell for **write** and choose rising edge (since write is an active high signal). You should have the following:

auto_signaltap_0			Lock mode: Allow all changes		
Type	Alias	Node	Data Enable	Trigger Enable	Trigger Conditions
		74	1	1	1 Comparisor
		...m_ip:my_custom_ip_0 invalid_export	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		...m:u0 custom_ip:my_custom_ip_0 irq	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		...custom_ip:my_custom_ip_0 Registers	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		...u0 custom_ip:my_custom_ip_0 write	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		+ ..._ip:my_custom_ip_0 address[2..0]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		+ ...custom_ip_0 ext_addr_export[2..0]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		+ ...ustom_ip_0 ext_data_export[31..0]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		+ ...my_custom_ip_0 writedata[31..0]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

10. Choose **File > save** and assign a name. Click **Save** and then **Yes**.
11. Since SignalTap adds signals to your design, you must compile at this point. Click on compile. When the compile is done, program the board
12. Open the NIOS II Software Build Tools for Eclipse and select the correct workspace for the Custom_component_demo. Right click on the bsp folder, choose **NIOS II > Generate BSP**.
13. Right click on the App folder and choose **Build Project**. On the pull down menu next to the bug icon, choose Debug as > NIOS II Hardware.
14. Return to the SignalTap window. Make sure the hardware is recognized and choose the correct **Device** as shown below



15. Click on the  Run Analysis icon. Now SignalTap is waiting for the trigger event. Return to NIOS II Software Build Tools for Eclipse and click on the resume button.
16. Now return to the SignalTap window. You will see that the data has triggered. Looking at the trigger point, you will see that there was a write pulse and the ext_data is written to the register

at address 0. This corresponds to line 131 in the C program where the internal registers are loaded. If you scroll to the right in the data window of SignalTap, you will see that another write pulse has not been captured, even though we are expecting four writes.

17. Return to the [Setup](#) tab and change the [Sample depth](#) to 256 and recompile. Click on the little red square in the Software Build Tools for Eclipse to disconnect (you have to do this every time you want to re-program the board). Program the board. Click on [Run Analysis](#) in SignalTap and rerun the C program.
18. This time if you scroll to the right after the data is captured, you can see that 2710H (10,000) is written to the register with address 1.
19. Since there are 8 writes to the internal register, this is not the most efficient way to check them. Another useful debugging tool is to use the breakpoint in the Software Build Tools. In SBT, right click in the space next to the for loop and choose [add breakpoint](#). Now run the program and it will stop at the breakpoint. In SignalTap, enable the analysis and in SBT, click on [step over](#) twice. In SignalTap, the write will be captured. By repeating this seven more times, you can verify all of the writes to the IP.
20. Return to [SignalTap](#) and click on the [Setup](#) tab. Click to disable write as a trigger and to enable `invalid_export` as a trigger. Set the trigger for rising edge. You will have to recompile the design and reprogram the board after changing the trigger. Disable the breakpoint in SBT. In SignalTap, enable the analysis and in SBT, run the program. Notice that in SignalTap it still says "Acquisition in Progress". This is because you have not entered an invalid address. Now raise SW3 on the board. In SignalTap you can see that as soon as the invalid signal went high, an interrupt was generated.

Although this tutorial may have been tedious, SignalTap is extremely useful in debugging your code. The more you use it, the more intuitive it will become.

Deliverables: Submit a document with the following captures. Be sure to explain what each capture is demonstrating in your document.

1. The original write capture.
2. A different write capture.
3. An error generation capture.