Getting Started With Signal Tap Logic Analyzer

Calvin College – Engr 304/325

Signal Tap is a Quartus tool for measuring actual circuit activity inside the FPGA. It is essentially a many-channel oscilloscope that, once triggered, records the state of every signal it is connected to every time it sees a rising edge on its clock. The clock it uses can be defined separately from the clock of the rest of the circuit. Note that some parts of your designs that are optimized away by Quartus may not be visible to the logic analyzer.

Creating and setting up the Signal Tap file

The first step after completing a compilation of your design without the logic analyzer is to create a new "SignalTap II Logic Analyzer File" using the "File/New…" menu item. The option to select is just above the "University Program VWF" option as shown in Figure 1.

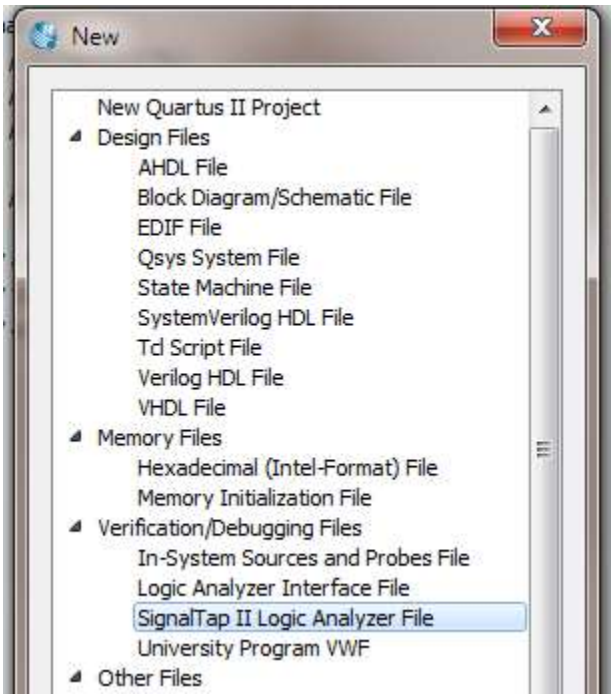

Figure 1. Choosing a new File

Quartus will open up the tool, showing the Setup tab where you select a clock and the signals to track and display. Figure 2 shows the key aspects of that view. The left hand panel allows you to select those signals to track and the right hand panel controls the clocking mechanism and the size of the data set to be collected. Note that the logic analyzer will continuously record the states of each signal every time its clock signal has a rising edge. As the data buffer fills, old data is overwritten. As soon as a trigger event takes place, a certain amount of the data collected prior to the event and a certain amount of data after the event is kept in the data buffer to be reported out to the user. The amount saved from before and after the trigger is controlled by the "trigger position" selection.

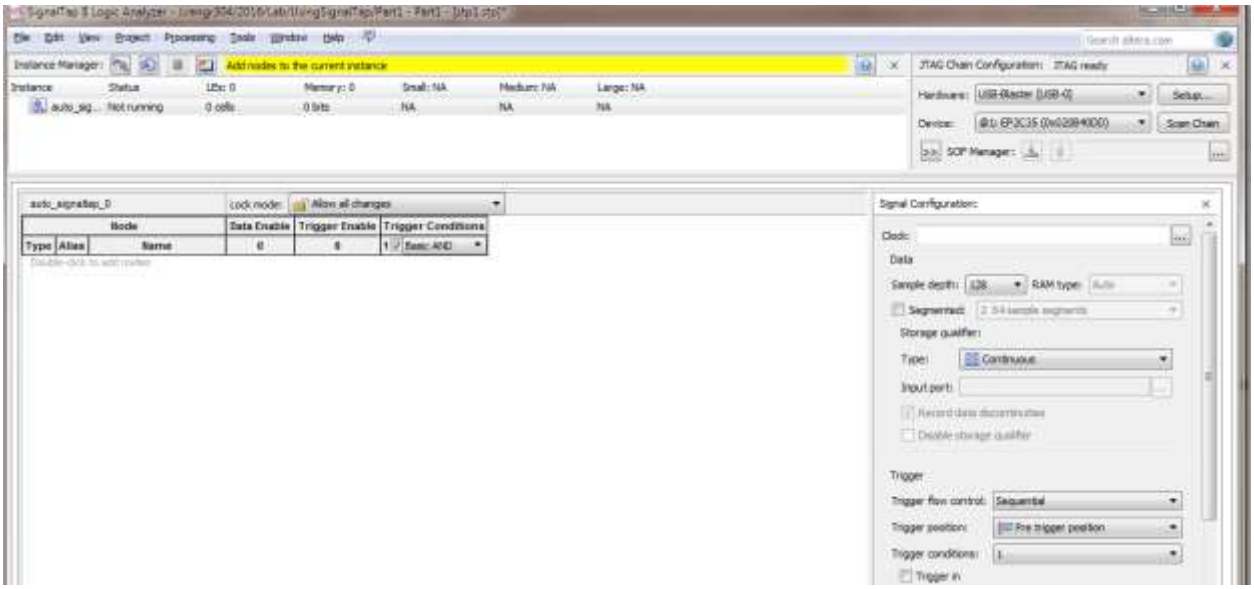

Figure 2. Signal Tap Tool

The first step in configuring the analyzer is to choose the signal that will act as the "clock." At each rising edge of this clock, the analyzer will take a snapshot of the state of each signal and then store that information in memory elements. The "Filter" choice as shown in Figure 3 allows you to look at pre- or post-compilation lists of nodes. In this case, the clock selected is the 50 MHz clock. Other slower signals can be used as the clock as long as they provide a consistent sampling rate.

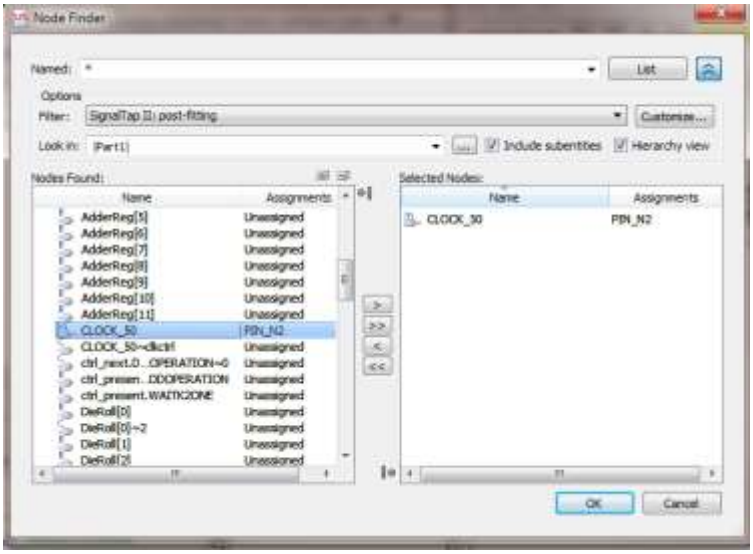

Figure 3. Using the Node Finder to select the clock

The next configuration choice is the size of the memory or "sample depth" used to store the samples (see Figure 4). As your system uses up more of the FPGA resources, less will be available for storing analyzer data. The largest resource used by the analyzer is the memory bits used to store the data. NIOS CPUs take from the same pool of memory bits to provide system RAM as the analyzer takes for string the sampled data.

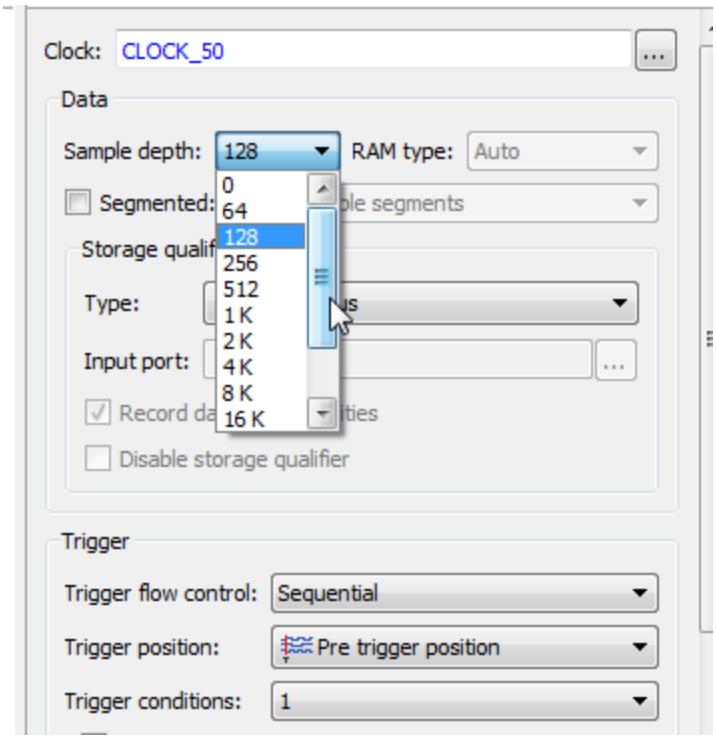

Figure 4. Adjusting the sample record size

After configuring the analyzer, the nodes to be analyzed need to be added to the left side of the window. This is done in a similar way as the Quartus simulator added nodes and in the way the clock signal was selected. After choosing the nodes, it is important to decide whether or not each particular signal (or bit of a bus) should be used to establish a trigger condition. The options are shown in Figure 5.

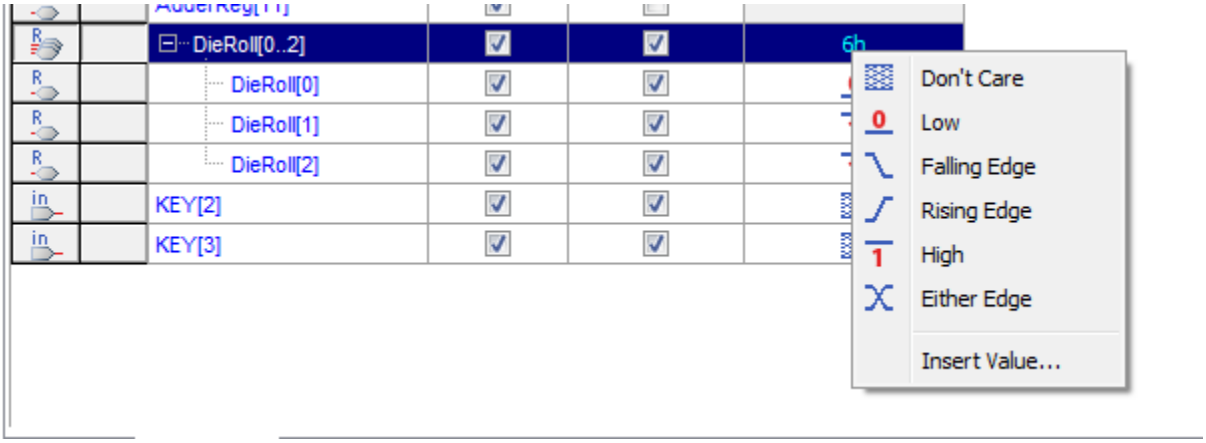

Figure 5. Choosing which signals cause a trigger event

Notice that as you add signals, you can add internal signals by navigating deeper into the Insert Nodes listings. In addition, if you left-click in the add-nodes area of the data window, you can select the "Add State Machine Nodes…" menu item. If you have any state machines in the design, the states can be displayed as well. Figure 6 shows the state machine names found.

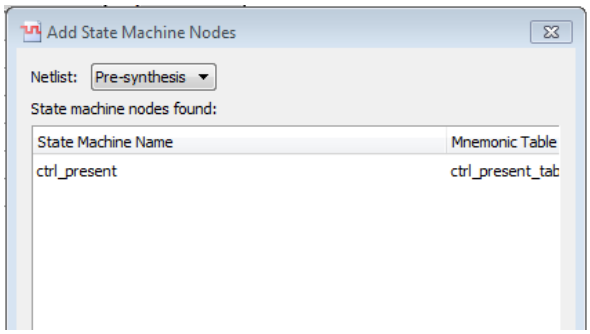

Figure 6. Adding state machine nodes to data windo

You may notice that as changes are made to the configuration, the status bar at the top of the window may change. Figure 7 shows what happens when the changes require a recompilation of the analyzer and the entire FPGA design. Changing trigger types will likely not require a compilation change, but other changes might.

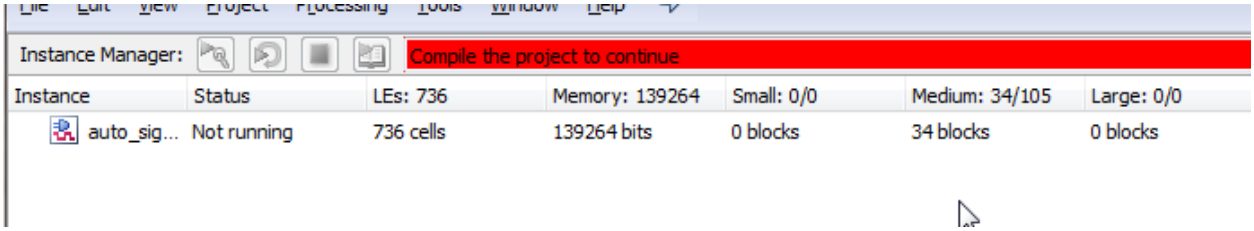

Figure 7. Notice of re-compilation required

Be sure to save the Signal Tap information in a *.stp file as shown in Figure 8.

Figure 8. Saving the Signal Tap file

After the save operation completes the first time, you will be asked if the Signal Tap component should be added to your current project, as shown in Figure 9. You should select "Yes" or you will need to manually add the *.stp file to your project.

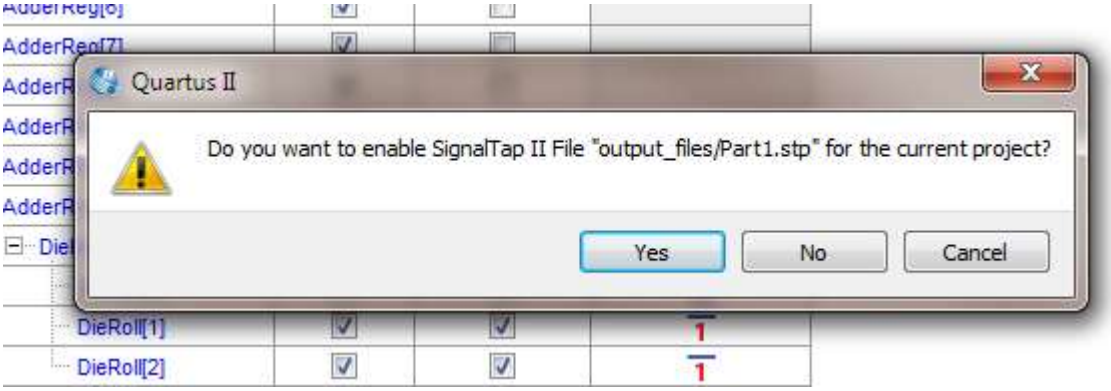

Figure 9. Prompt to add the analyzer to your project

After the project is fully compiled and programmed on the FPGA, you should see the status bar indicate that the analyzer is ready to acquire data. This is shown in Figure 10.

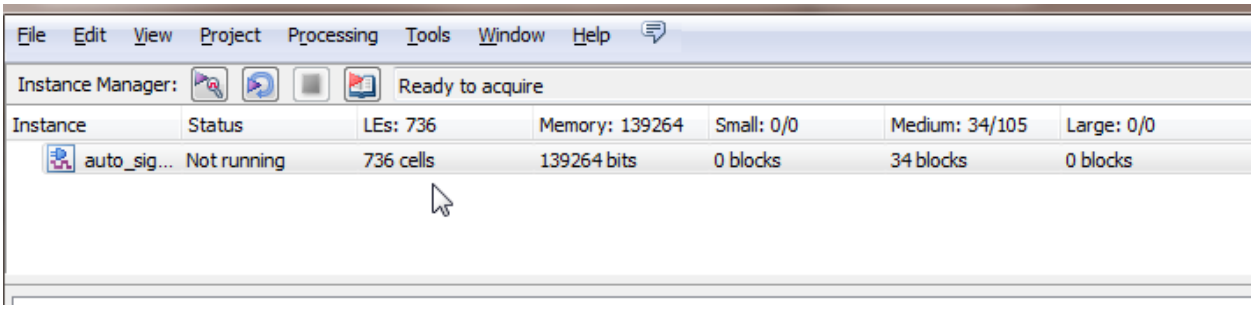

Figure 10. Ready to acquire and status info

Running the data logging system

In order to start an actual data logging session, the "Run Analysis" button shown in Figure 11 will need to be selected. The analyzer will then look for a triggering signal. If one doesn't arrive soon, then the status will show that the acquisition is in progress and it is waiting for the trigger signal. This is shown in Figure 12.

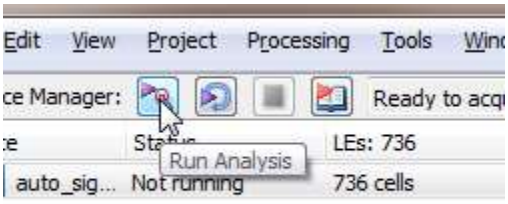

Figure 11. Button to start the analysis

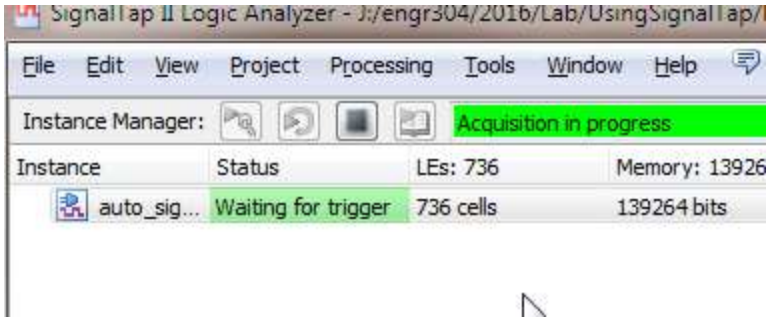

Figure 12. Status of analyzer when waiting for trigger event

A look at the resulting waveforms

Figure 13 shows a die-rolling and accumulating register design as it responds to the trigger event defined as "DieRoll = 0x6." The signal values before and after the event are shown. Figure 14 shows the signals in response to a trigger event defined as the rising edge of KEY(2). Notice in the figure that the states of a state machine are also shown as signals.

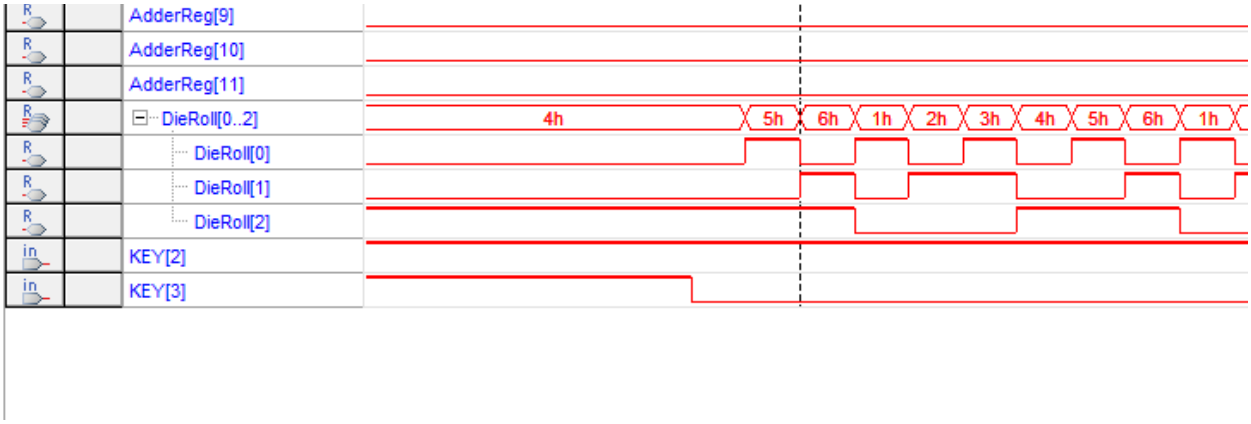

Figure 13. Waveforms when triggered by DieRoll = 0x6

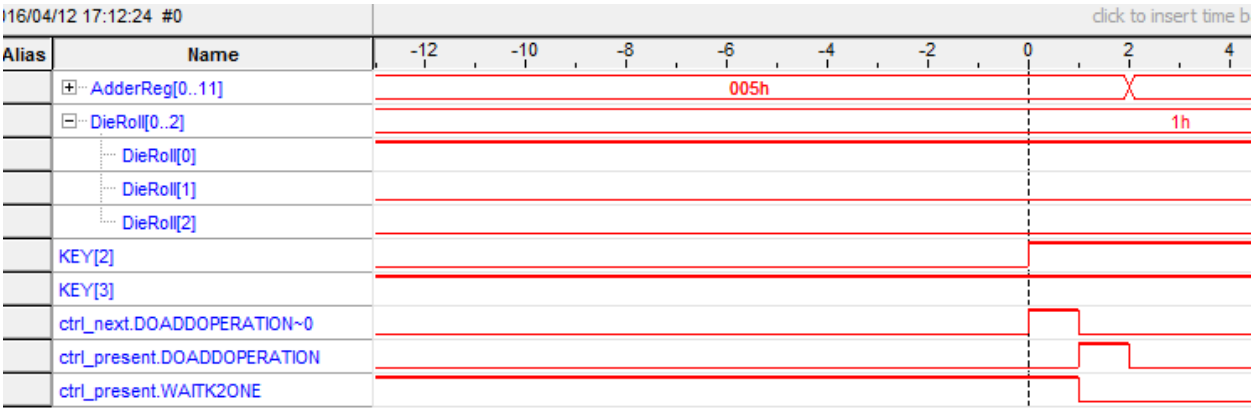

Figure 14. Waveforms when triggered by KEY(2) rising edge

Preventing signals and registers from being optimized away during compilation

It is possible to set an attribute on a VHDL signal such that the compiler with preserve or keep the signal during compilation and optimization. Figure 15 shows two VHDL statements that set certain attributes of signals. These attributes control the way the VHDL is compiled. For signals intended to be registers, you would use the "preserve" attribute and for signals intended to be simple wires from combinational logic blocks, you would use the "keep" attribute. Keep in mind that after debugging is done, these attribute assignments should be removed or set to "false."

```
SIGNAL AdderReg : STD_LOGIC_VECTOR(11 downto 0);
attribute preserve: boolean;  -- keeps reg bits from optimization
attribute preserve of AdderReg: signal is true;
SIGNAL Sum : STD_LOGIC_VECTOR(11 downto 0);
attribute keep: boolean;  -- keeps wires from optimization
attribute keep of sum: signal is true;
```

Figure 15. Preserving signals during optimization