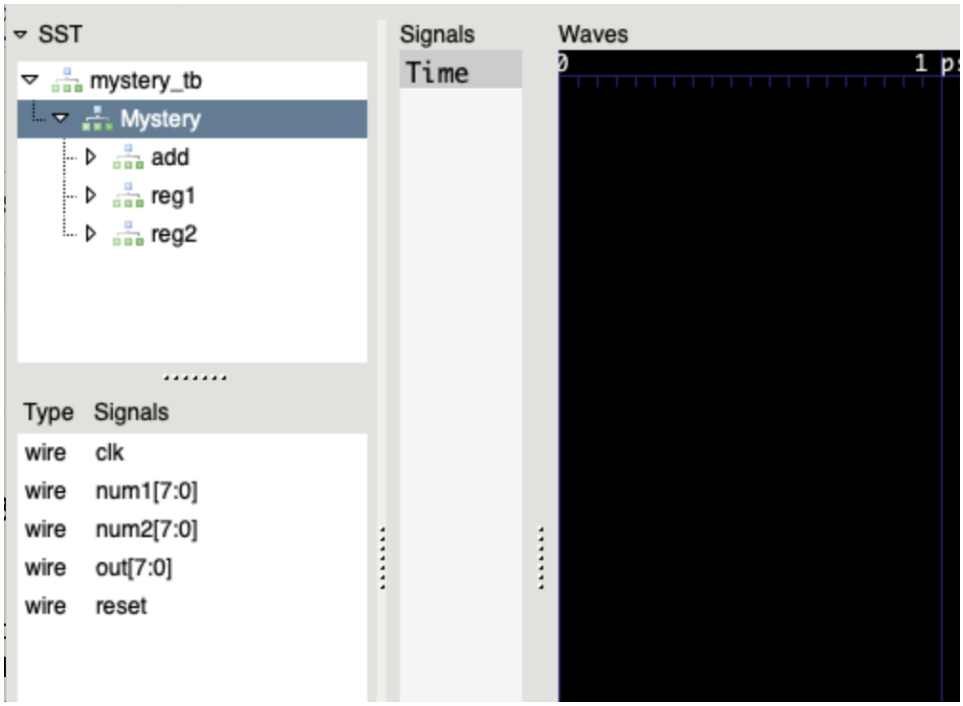
CompSci/ECE350 HW3 Due Friday 2/3/23 11:59 PM

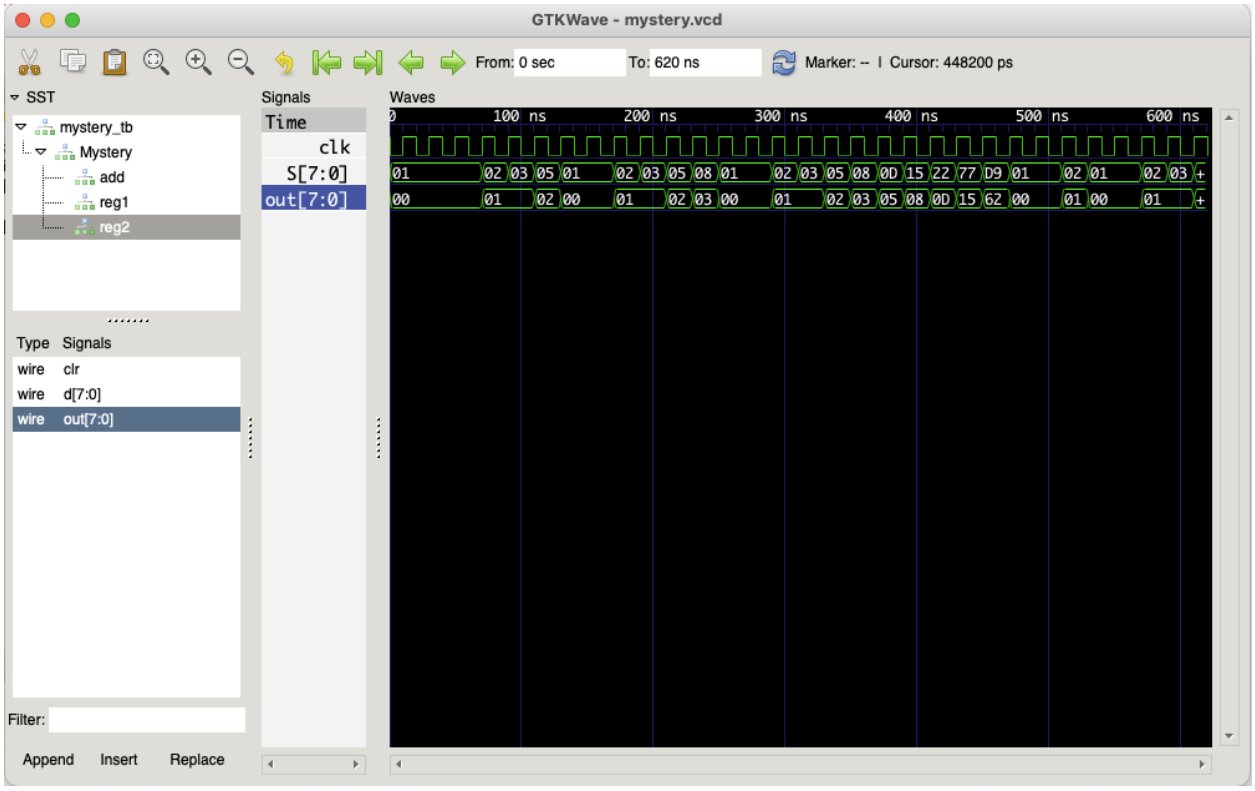
This assignment will introduce you to the basics of using Gtkwave to analyze and debug trace files. Gtkwave will become an essential tool when working on your project checkpoints. To begin download the trace file for the assignment here: mystery.vcd.

This assignment will require a combination of short responses and screenshots of your own gtkwave interface. You may either make a copy of this document and attach your answers inline or create a new document and put your answers there. In either case, you should submit your solutions to Gradescope.

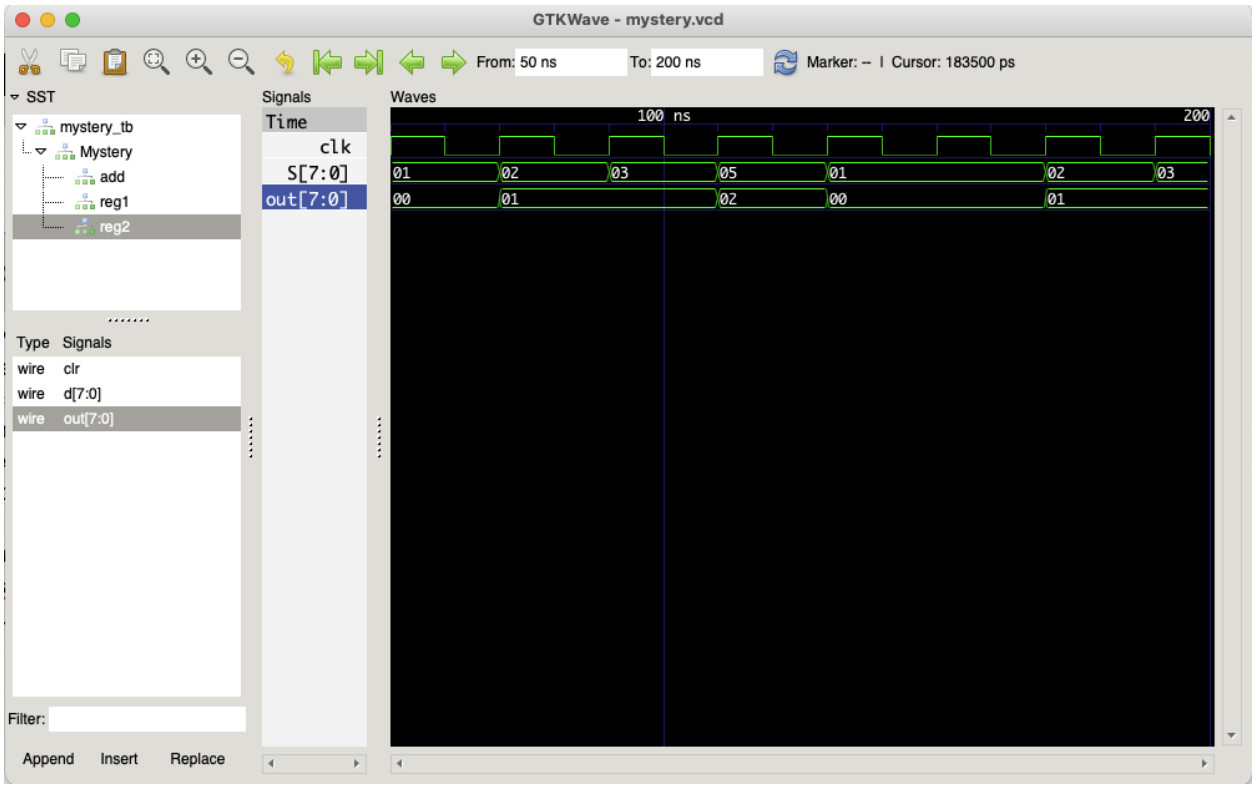
The most important panels in Gtkwave are the two on the left hand side of the screen. The panel on top displays the module tree and the panel on the bottom displays all of the wires in the selected module. The combination of these two panels allows you to understand the structure of your designs and analyze the value of any wire within any module of your design.

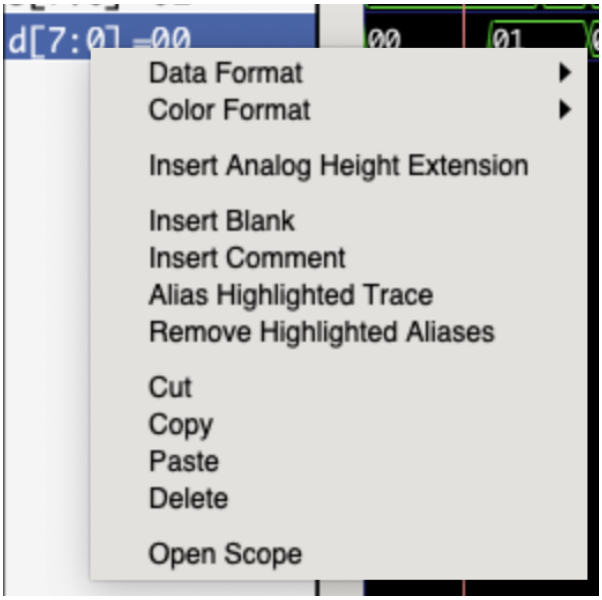


1 If you haven't already, open the mystery.vcd trace file in gtkwave. Search through the module tree and add the following signals to the analyzer: clk, S, and the data of register 2. Attach a screenshot of your gtkwave interface after you have selected the three signals.



2 The toolbar at the top provides ways to effectively navigate to different ranges of time within the trace file. The leftmost magnifying glass symbol does a zoom fit to fit the entire range of the trace file within the window. The other two zoom options let you zoom in and out. Also of note are the green arrows which let you zoom to the start and end of the trace file or zoom to the previous or next edge of the selected signal. Lastly, the input boxes let you specify a precise range of the trace file to examine by hiding the values outside of that range. To restore the entire trace, delete one of the range options you've specified and press enter without entering a new value and it should restore the range to the default value from the trace file. Play around with these tools a bit, then attach a screenshot of your gtkwave window with the range 50-200ns selected below. *(Note: the signals from question 1 should still be included)*



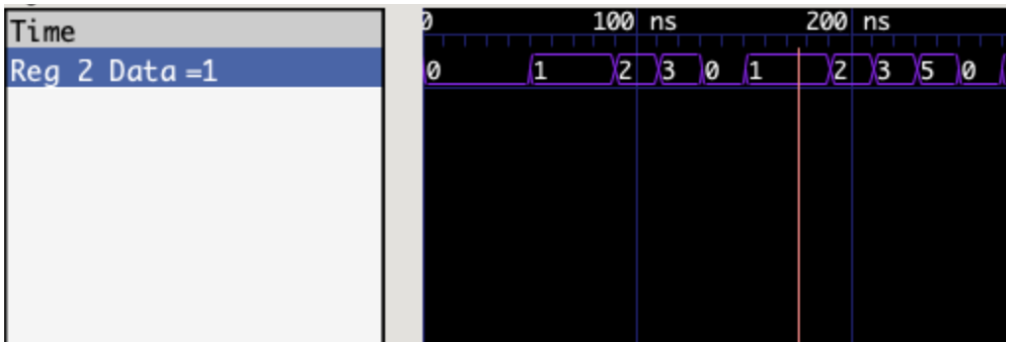


Right clicking on a trace within the analyzer opens up a number of useful tools to better organize the signals within your design. Some useful ones include:

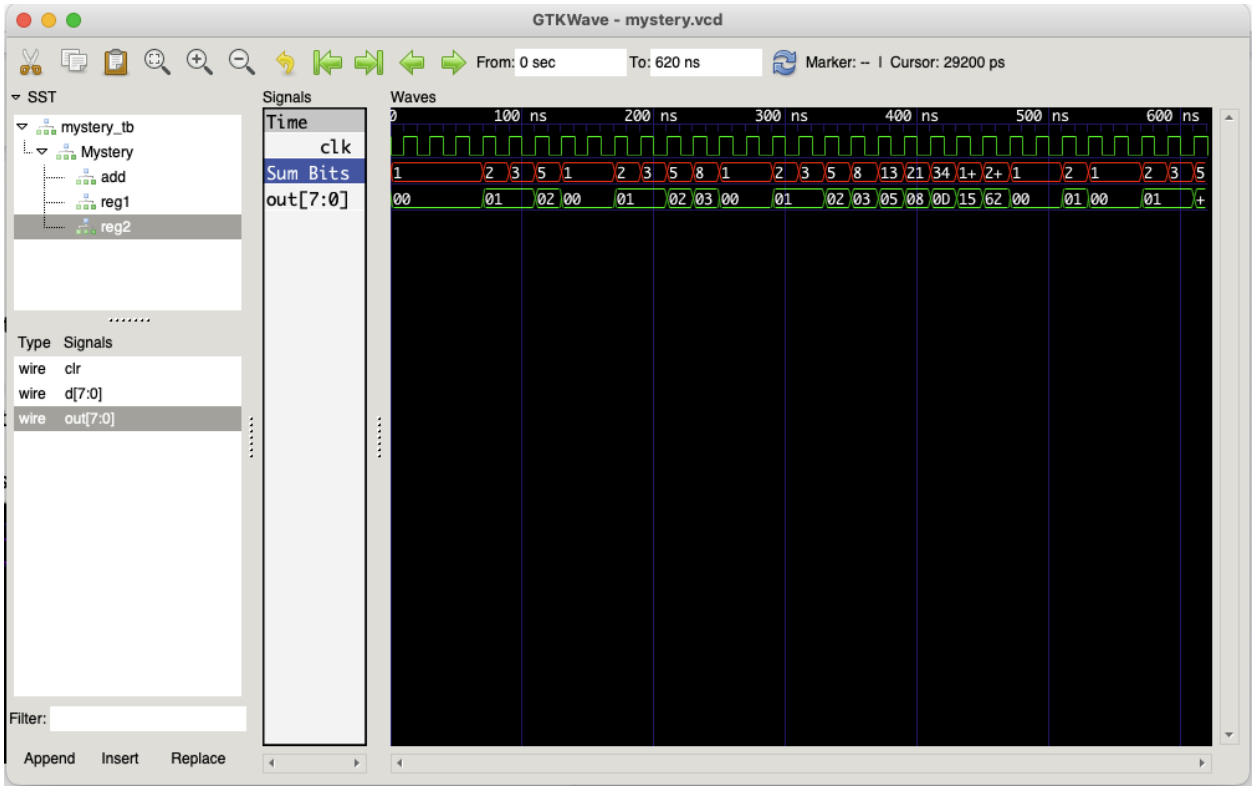
- Data Format - allows you to specify the format to display the value of the signal. Options include binary, hexadecimal, signed and unsigned decimal, ascii, etc..
- Color Format - allows you to specify the color of the signal in the analyzer
- Alias Highlighted Trace - allows you to rename the signal in the analyzer

Below is an example of the register 2 data signal being aliased to “Reg 2 Data”, with the data format changed to decimal, and color changed to violet.

Example using the register 2 data.

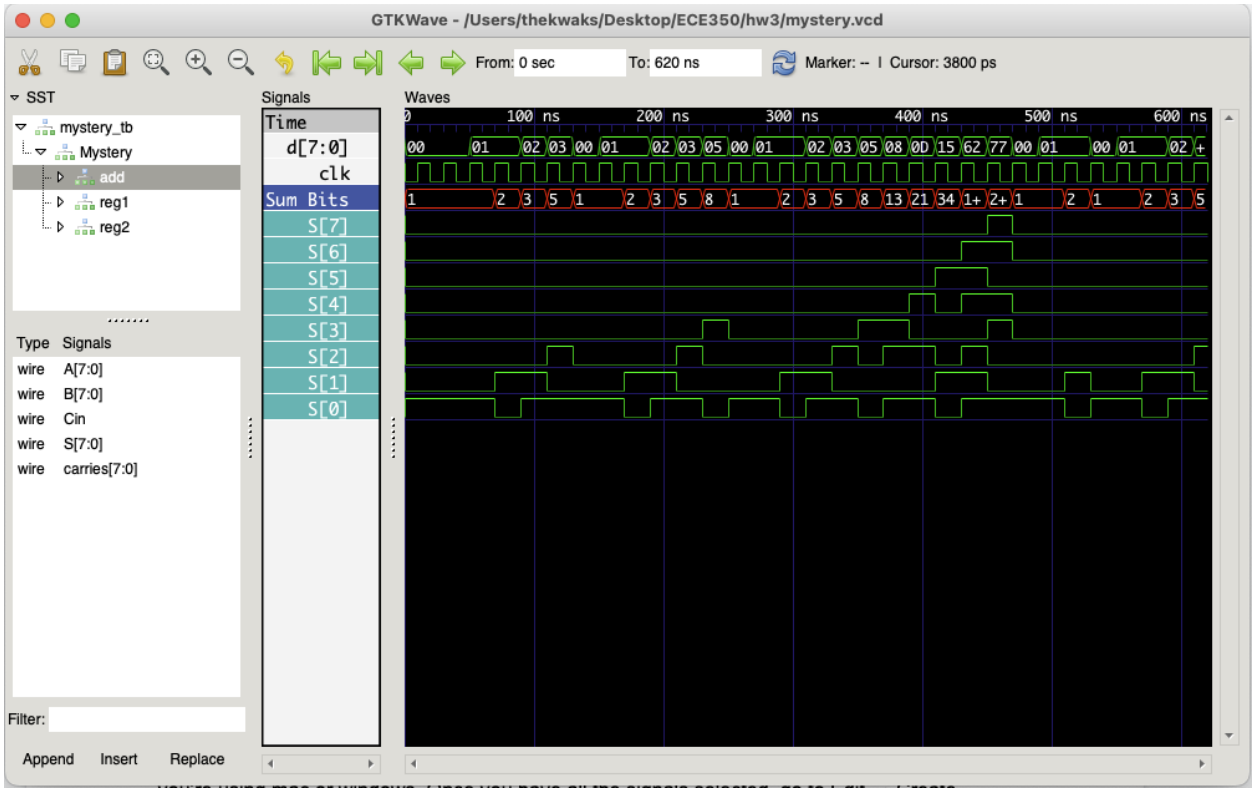


3 Use each of the three tools mentioned above on the signal S. Attach a screenshot of your analyzer when complete. *(Note: If you haven’t already reset the range to the trace file defaults do so now. Refer back to question 2 if you are confused)*



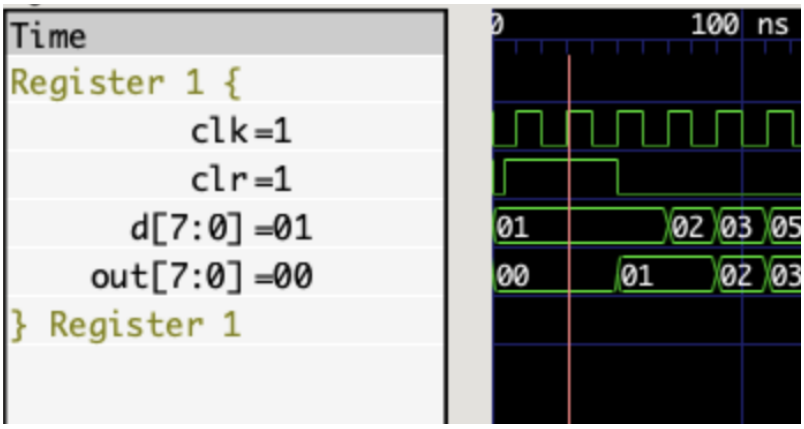
4 If you have a signal that is made up of a bus of wires rather than a single wire (i.e. S), you can double click on the signal name in the analyzer to expand that signal into each of its individual

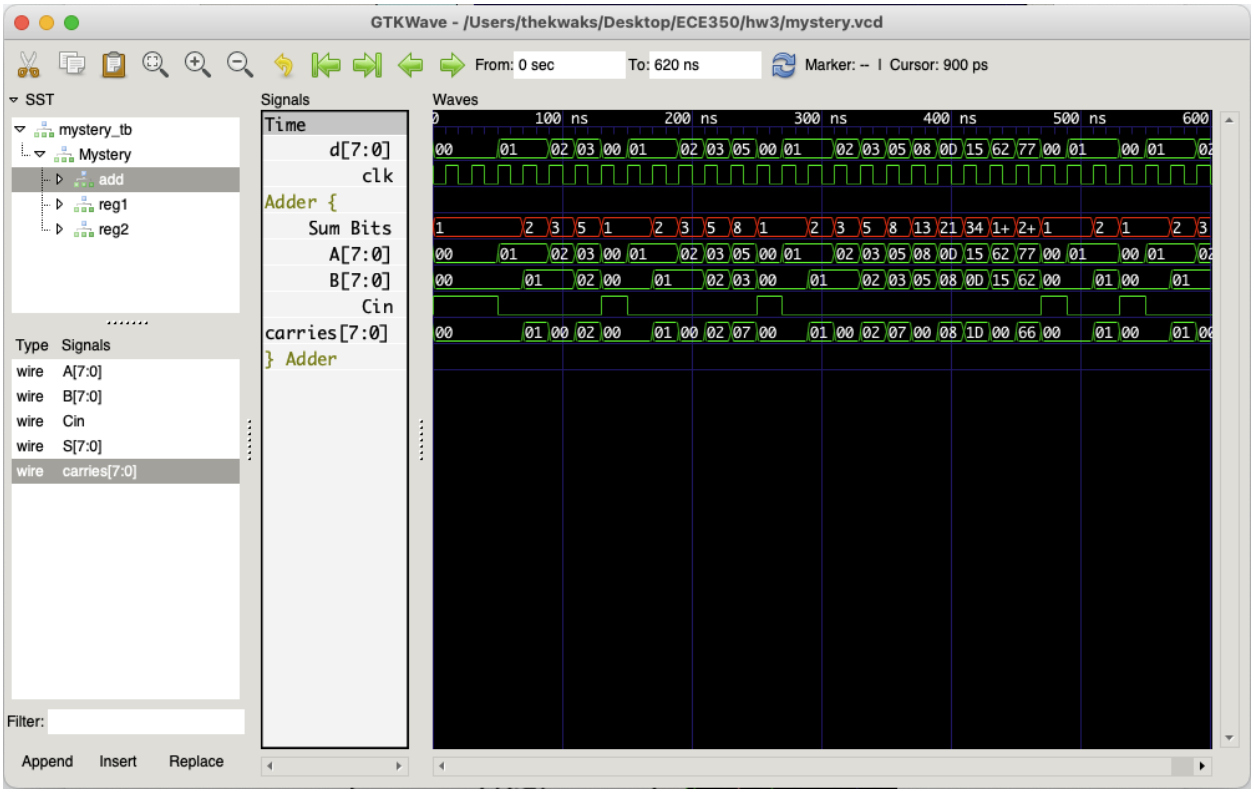
component wires for closer analysis. Attach a screenshot of the S signal expanded to show all 8 of its component wires.



5 You can also create groups to organize and easily show/hide signals within the analyzer. First, add all of the signals from the *add* module to the analyzer. Then, select all of the names of the signals in the analyzer by either clicking the first then holding shift and selecting the last in the range, or by clicking each signal individually while also holding Cmd or Ctrl depending on if you're using mac or windows. Once you have all the signals selected, go to Edit → Create Group and give your group a descriptive name. Within the analyzer your signals are nested within the group and the group can be closed or expanded by double clicking the name of the group within the analyzer. Attach a screenshot of the group you created for the signals in the adder module.

Example using the signals in register 1.





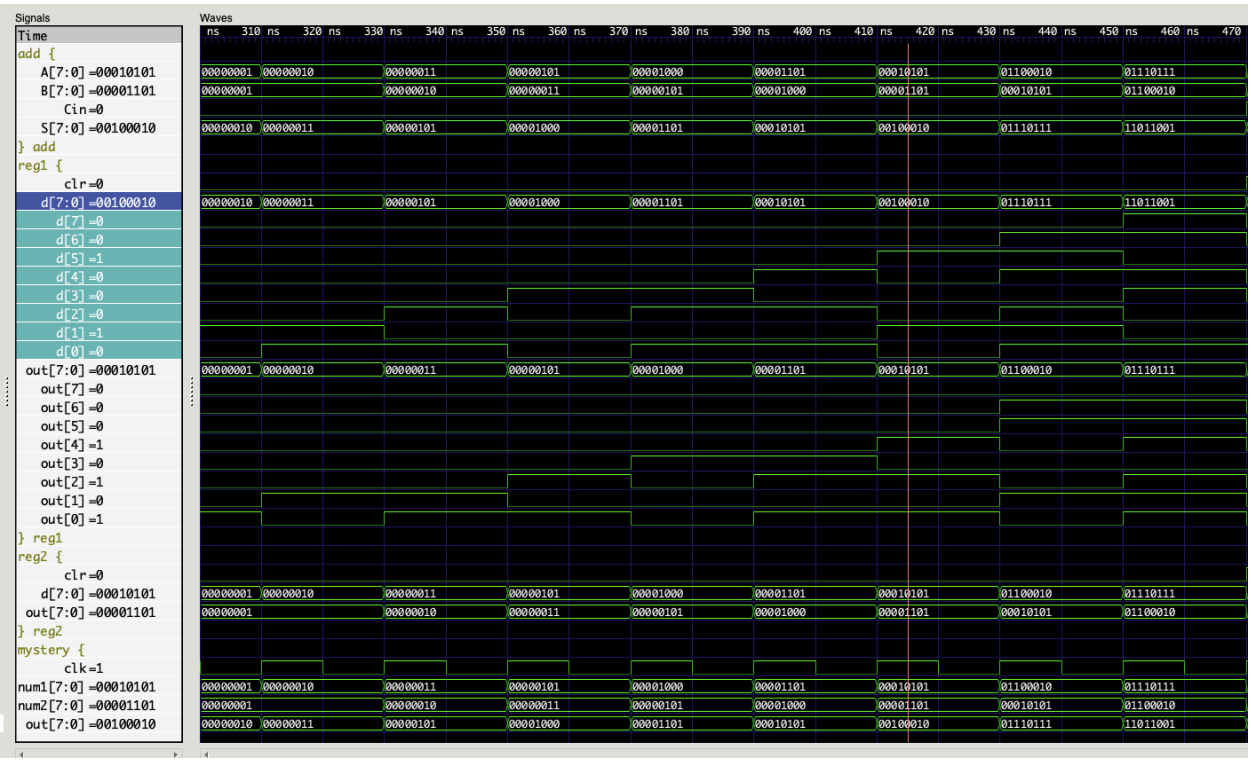
6 Use the analyzer to determine the purpose of the *Mystery* module. In a few sentences describe what the *Mystery* module does and the roles of each of its child modules.

The purpose of the Mystery module is to output the fibonacci sequence. It does this by using the add module to add the outputs of reg1 and reg2, which contain both the previous and current numbers in the sequence to produce the next number in the fibonacci sequence. The add module starts off with a Cin of 1, which produces a sum of 1. The sum is always routed to the data input of reg1, which gets latched to reg1's output after a clock cycle. The data input of reg2 follows the output of reg1 so that reg2 takes on the value of reg1 after a clock cycle. This

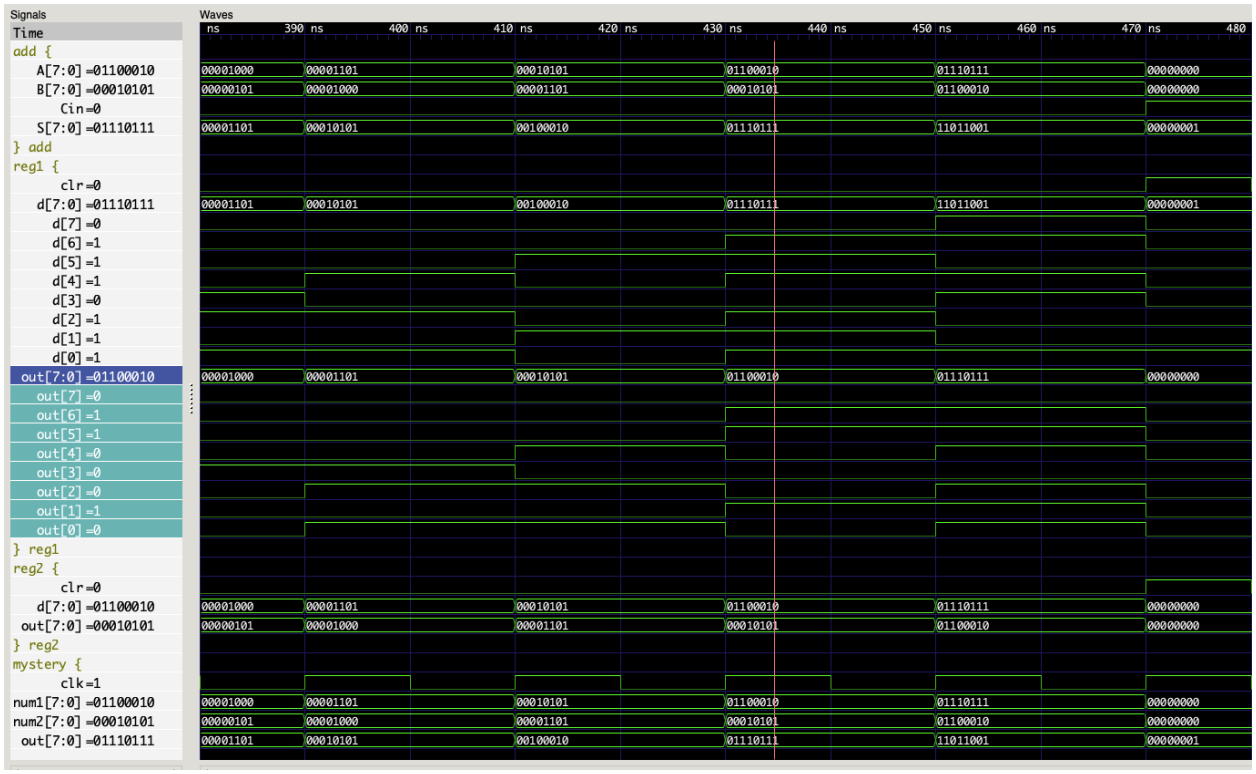
produces the effect of reg2 always lagging behind by one clock cycle, and the sum from the add module accurately producing the next number in the sequence which gets put back into reg1.

7 There is a bug somewhere in the design. Use the trace file to find and identify the root cause of the bug. Describe what the bug in the design is and attach screenshot(s) showing your work.

The bug in the design is that at 430 ns into the run, the data input of reg1 does not properly get latched into the output of reg1. In the previous clock cycle (shown below), the data input of reg1 is correctly equal to 34 (00100010 in binary).



However, in the following clock cycle, the output of reg1 gets set to 98 (01100010 in binary), which ends up producing a sum of 98 + 21 = 119 (01110111 in binary) when really the output of reg1 should be 34 and the next number in the fibonacci sequence outputted by the Mystery module should be 34 + 21 = 55 (00110111 in binary). The screenshot below shows this next clock cycle where the output of reg1 is incorrectly set to 98.



The cause of this bug seems to be that the wires representing the bits out[6] and out[5] are connected, so that when out[5] is set to 1 during the latching of value of 34 (00100010), out[6] also gets set to 1 to produce 98 (01100010).