The goal of the assignment is to develop an understanding for the different Verilog timing concepts in order to realize combinational circuit descriptions.

***Homework 3: PART 1***

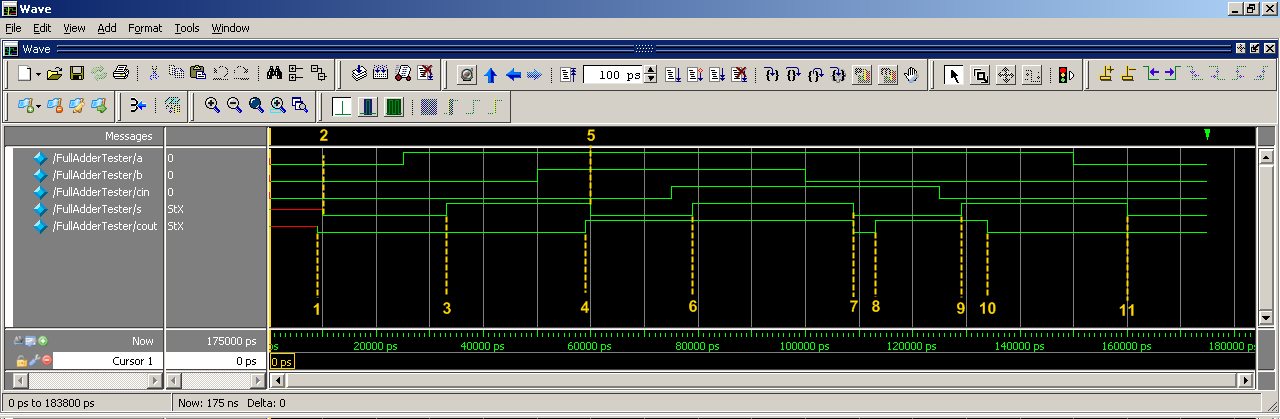
The Verilog code for this section can be found in the Pt1 folder of the ZIP file submitted:

*FullAdder.v*

*FullAdderTester.v*

In running the FullAdderTester module in ModelSim, the following waveform was generated:

**Simulation Waveform Output of FullAdderTester**



The 11 cases where both outputs, *s* and *cout*, changed in the simulation, have been labeled. The timing verification for each of these points, based on the given delays, is as follows:

*tPLH (Low to High) = 4ns*

*tPHL (High to Low) = 5ns*

In order to explain the timetables below, here is the circuit description that was implemented in Verilog, with the naming conventions used in the descriptions below:



**Timing 01** @ 9ns, *cout* goes from X to 0:

* Inputs: a = 0, b = 0, cin = 0
* +4ns delay: 2nd NAND gate sees that NAND(a & b) = 1, so use tPLH value.
* +5ns delay: 3rd NAND gates sees that NAND (X + 1) = 0, so use tPHL value.
* Total delay = 9ns, which is correct.

**Timing 02** @ 10ns, *s* goes from X to 0:

* Inputs: a = 0, b = 0, cin = 0
* +5ns delay: 1st XOR gate sees that XOR(a,b) = 0, so use tPHL value.
* +5ns delay: 2nd XOR gate sees that XOR (0,cin) = 0, so use tPHL value.
* Total delay = 10ns, which is correct.

**Timing 03** @ 33ns, *s* goes from 0 to 1:

* Inputs: a = 1 @ 25ns, b = 0, cin = 0
* +4ns delay: 1st XOR gate sees that XOR(a,b) = 1, so use tPLH value.
* +4ns delay: 2nd XOR gate sees that XOR (1,cin) = 1, so use tPLH value.
* Total delay = 8ns, which is correct since 33ns – 25ns (input change) = 8ns.

**Timing 04** @ 59ns, *cout* goes from 0 to 1:

* Inputs: a = 1, b = 1 @ 50ns, cin = 0
* +5ns delay: 2nd NAND gate sees that NAND(a,b) = 0, so use tPHL value.
  + Note: 1st NAND gate delay with XOR input doesn’t matter, since it still = 1.
* +4ns delay: 3rd NAND gates sees that NAND(NAND1, NAND2) = 1, so use tPLH value.
* Total delay = 9ns, which is correct since 59ns – 50ns (input change) = 9ns.

**Timing 05** @ 60ns, *s* goes from 1 to 0:

* Inputs: a = 1, b = 1 @ 50ns, cin = 0
* +5ns delay: 1st XOR gate sees that XOR(a,b) = 0, so use tPHL value.
* +5ns delay: 2nd XOR gate sees that XOR(0, cin) = 0, so use tPHL value.
* Total delay = 10ns, which is correct since 60ns – 50ns (input change) = 10ns.

**Timing 06** @ 79ns, *s* goes from 0 to 1:

* Inputs: a = 1, b = 1, cin = 1 @ 75ns
* +4ns delay: 2nd XOR gate sees that XOR(0,cin) = 1, so use tPLH value.
* Total delay = 4ns, which is correct since 79ns – 75ns (input change) = 4ns.

**Timing 07** @ 109ns, *s* and *cout* go from 1 to 0:

* Inputs: a = 1, b = 0 @ 100ns, cin = 1
* For *s*:
  + +4ns delay: 1st XOR gate sees that XOR(a,b) = 1, so use tPLH value.
  + +5ns delay: 2nd XOR gate sees that XOR(1,1) = 0, so use tPHL value.
* For *cout*:
  + +4ns delay: 2nd NAND gate sees that NAND(a,b) = 1, so use tPLH value.
    - Note that 1st NAND gate still has a value of 1 due to extra gate delay.
  + +5ns delay: 3rd NAND gate sees that NAND(1,1) = 0, so used tPHL value.
* Total delay = 9ns for BOTH *s* and *cout*, which is correct since 109ns – 100ns (input change) = 9ns.

**Timing 08** @ 113ns, *cout* goes from 0 to 1:

* Inputs: a = 1, b = 0 @ 100ns, cin = 1
* +4ns delay: 2nd NAND gate sees the change from b = 0 @ 109ns, so now NAND(1,1) = 0.
  + This will cause 3rd NAND gate, output *cout*, to change to 1 since NAND(0,1) = 1.
* Total delay = 4ns, which is correct since 113ns – 109ns (gate delay) = 4ns.

**Timing 09** @ 129ns, *s* goes from 0 to 1:

* Inputs: a = 1, b = 0, cin = 0 @ 125ns
* +4ns delay: 2nd XOR gate sees the change from cin = 0 @ 125ns, so now XOR(1,0) = 1.
  + The 1st XOR gate sees no change, since a & b remain the same.
* Total delay = 4ns, which is correct since 129ns – 125ns (gate delay) = 4ns.

**Timing 10** @ 134ns, *cout* goes from 1 to 0:

* Inputs: a = 1, b = 0, cin = 0 @ 125ns
* +4ns delay: 2nd NAND gate sees that NAND(1,cin) = 1, so used tPLH.
* +5ns delay: 3rd NAND gate sees that NAND(1,1) = 0, so use tPHL.
* Total delay = 9ns, which is correct since 134ns – 125ns = 9ns.

**Timing 11** @ 160ns, *s* goes from 1 to 0:

* Inputs: a = 0 @ 150ns, b = 0, cin = 0
* +5ns delay: 1st XOR gate sees that XOR(a,b) = 0, so used tPHL.
* +5ns delay: 2nd XOR gate sees that XOR(0,cin) = 0, so use tPHL.
* Total delay = 10ns, which is correct since 160ns – 150ns = 9ns.

The preceding timetables, along with the matching code and output waveform graph, showcase the correct functionality of the Full Adder, given the tPLH and tPHL delays of 4 and 5 ns, respectively.

This concludes the analysis for Homework 3, Part 1.

***Homework 3: PART 2***

The Verilog code for this section can be found in the Pt2 folder of the ZIP file submitted:

*FullAdder.v*

*FullAdder4bit.v*

*FullAdder4bitTester.v*

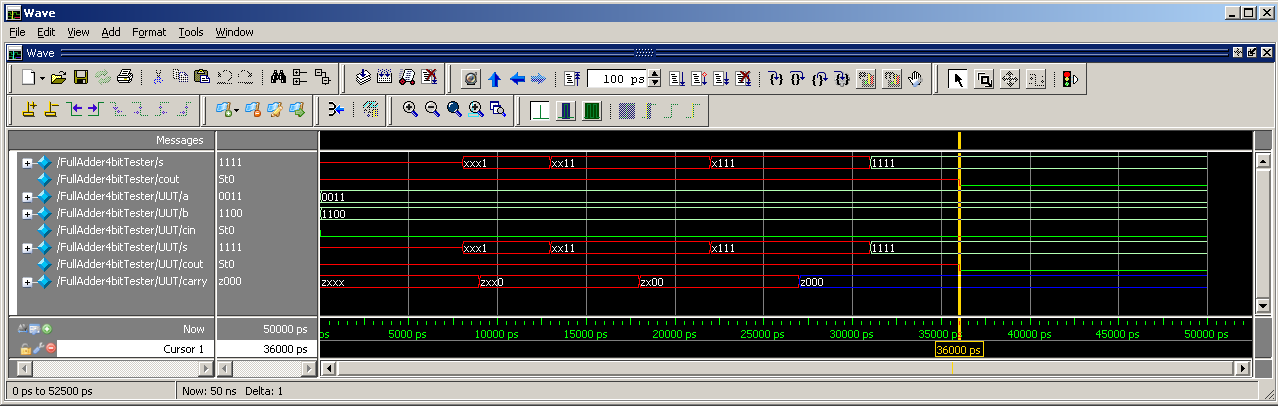
As you can see, the testbench looks at 4 different scenarios:

1. a = 0011, b = 1100, cin = 0 🡪 s = 1111, cout = 0
2. a = 0111, b = 0000, cin = 1 🡪 s = 1000, cout = 0
3. a = 1111, b = 1111, cin = 0 🡪 s = 1110, cout = 1
4. a = 0011, b = 1100, cin = 1 🡪 s = 0000, cout = 1

Below are output graphs from all simulations.

* The top two outputs are the *s* and *cout* of the 4 bit Full Adder Tester.
* The bottom outputs are of the actual 4 bit Full Adder.
  + This showcases all of the individual changing bits of *carry* and *s* within the Adder.

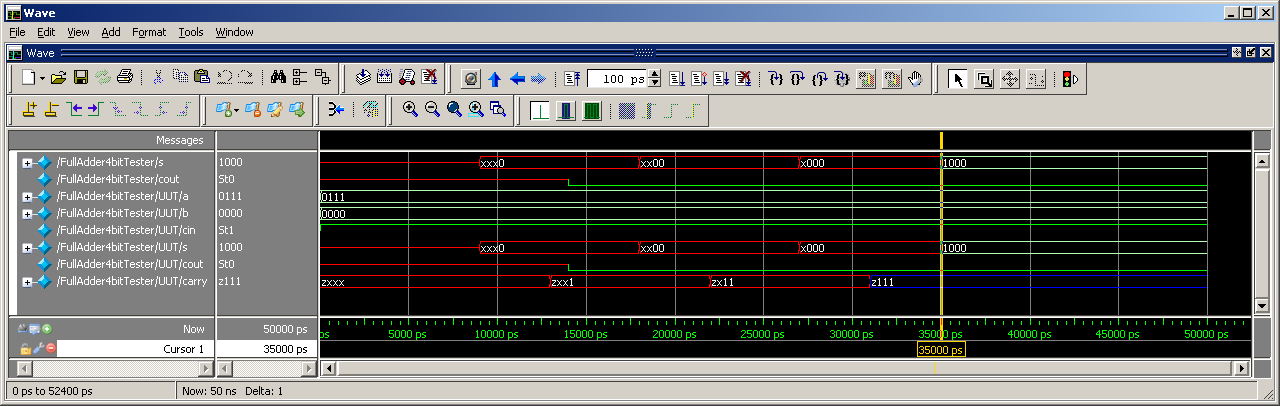
**Simulation Waveform Output of FullAdder4bitTester – Simulation 1**



In looking at this waveform output, *s* = 1111 and *cout* = 0, as expected.

The overall delay of the Adder = 36ns.

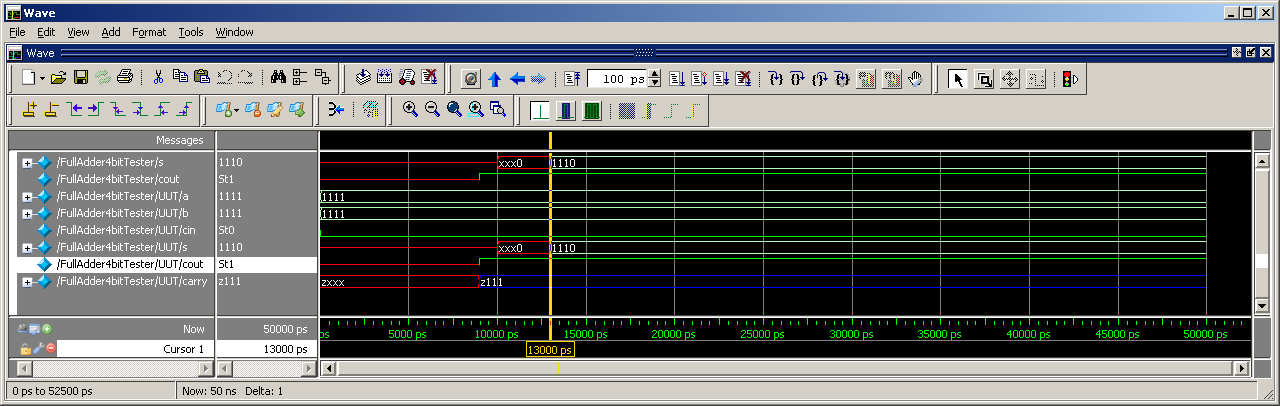
**Simulation Waveform Output of FullAdder4bitTester – Simulation 2**



In looking at this waveform output, *s* = 1000 and *cout* = 0, as expected.

The overall delay of the Adder = 35ns.

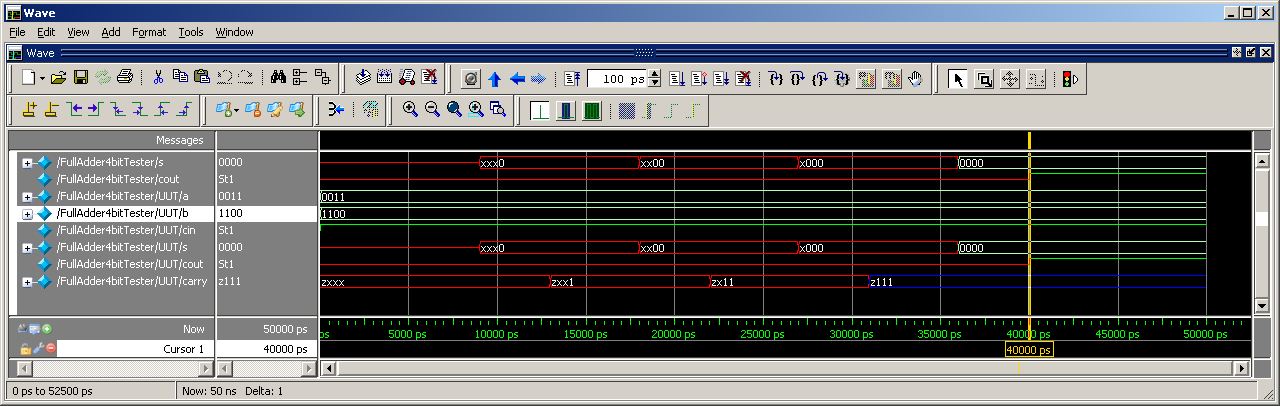
**Simulation Waveform Output of FullAdder4bitTester – Simulation 3**



In looking at this waveform output, *s* = 1110 and *cout* = 1, as expected.

The overall delay of the Adder = 13ns.

**Simulation Waveform Output of FullAdder4bitTester – Simulation 4**



In looking at this waveform output, *s* = 0000 and *cout* = 1, as expected.

The overall delay of the Adder = 40ns.

Thus, the **worst case** delay for this system is **40ns**.

This concludes the analysis for Homework 3, Part 2.

***Homework 3: PART 3***

The Verilog code for this section can be found in the Pt3 folder of the ZIP file submitted:

*FullAdder.v*

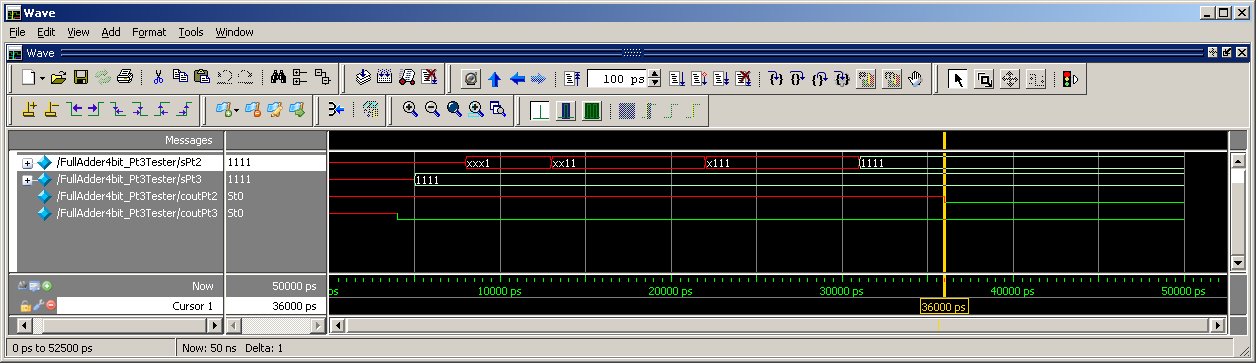
*FullAdder4bit.v*

*FullAdder4bit\_Pt3.v*

*FullAdder4bit\_Pt3Tester.v*

In instantiating both simulations, it can be seen that the assign statement (with concatenation) causes an instantaneous calculation of the *cout* and *s* bits, so there is effectively no computation time. The only delays that were present were the initial pin-to-pin delays that were assigned to the tPHL and tPLH values from Part #2. Here is an example output from Simulation 1:

**Simulation Waveform Output of Pt2 & Pt3 Full Adders – Simulation 1**



As state previously, it takes 4ns for *cout* to receive the correct value for the Pt3 Adder, and 5ns for *s* to receive the correct value for the Pt3 Adder. Below is a table that summarizes all of the timing differences between all four Simulations and the two different Adder instantiations:



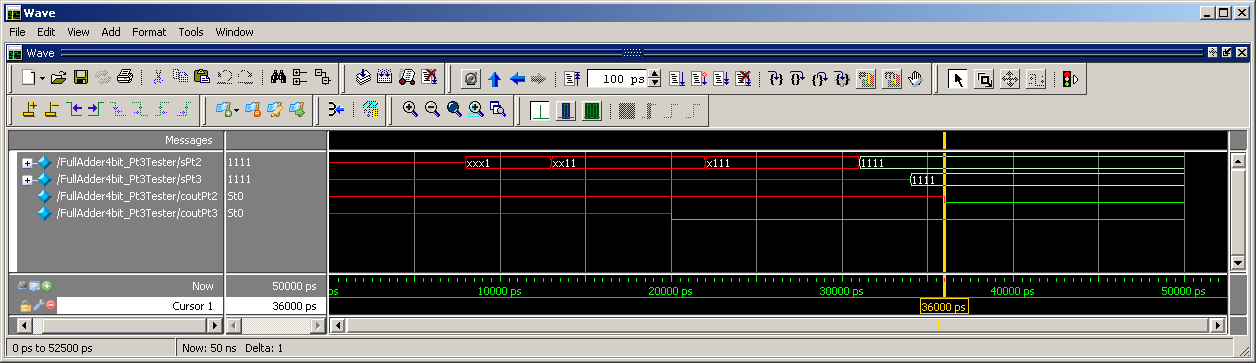
Based on this data, then pin-to-pin delays were adjusted accordingly:

*cout = 4 + 20.75 = 25ns*

*s = 5 + 23.75 = 29ns*

Here is the new model, with the applied pin-to-pin delays:

**Simulation Waveform Output of Pt2 & Pt3 Full Adders – Simulation 1**



Below is the new table that summarizes all of the timing differences between all four Simulations and the two different Adder instantiations, now with the adjusted pin-to-pin delays:



The averages across all four simulations are good, but can still be further optimized. In playing around with the deltas and the averages, the following pin-to-pin delays were chosen:



Therefore, the final pin-to-pin delays for the FullAdder4bit\_Pt3 are as follows:

specify

(a, b, cin \*> cout) = 20; 🡺 20ns for *cout*

(a, b, cin \*> s ) = 34; 🡺 34ns for *s*

endspecify

This concludes the analysis for Homework 3, Part 3.

***Homework 3: PART 4***

The Verilog code for this section can be found in the Pt4 folder of the ZIP file submitted:

*FullAdder.v*

*FullAdder4bit\_Pt3.v*

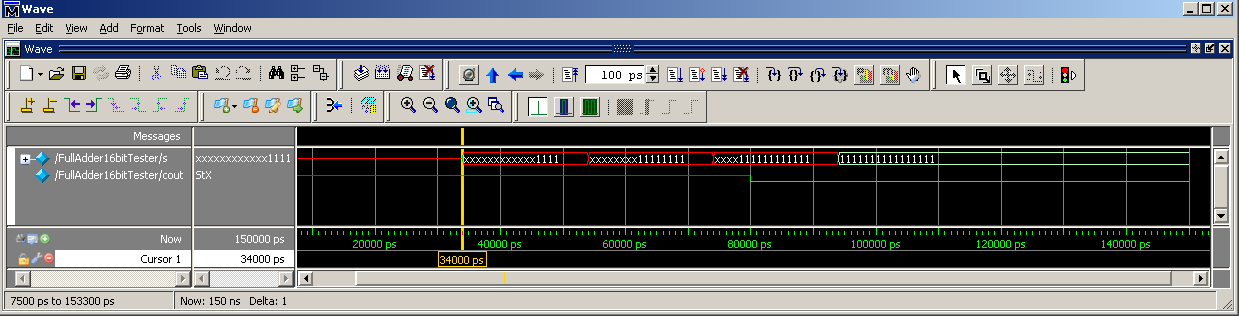
*FullAdder16bit.v*

*FullAdder16bitTester.v*

From Pt3 of Homework 3, the pin-to-pin delays were set to: (a, b, cin \*> cout) = 20; (a, b, cin \*> s ) = 34.

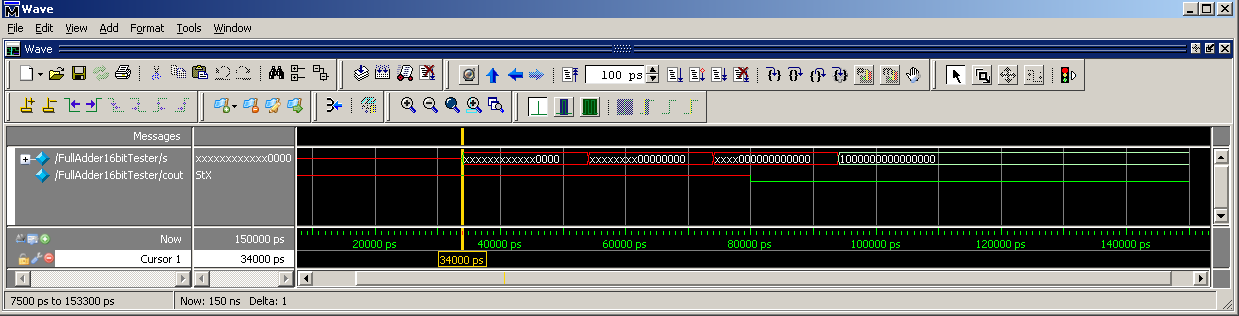
In order to verify operation and timing delays, here are the outputs for all four simulations:

**Simulation Waveform Output of 16bit Full Adder – Simulation 1**



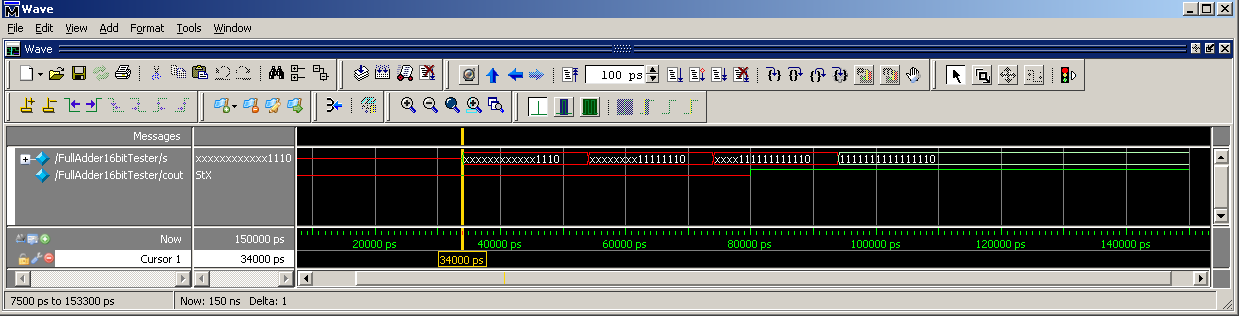
The 1st timing delay on *s* happens at 34ns, which makes sense given the pin-to-pin delay. Subsequent delays are in multiples of 20ns or 34ns. The output *cout* occurs at 80ns, which is expected since its pin-to-pin delay is 20ns, and we are doing four separate adder computations. According to the simulation and what is being tested, the sum and carryout outputs are also correct.

**Simulation Waveform Output of 16bit Full Adder – Simulation 2**



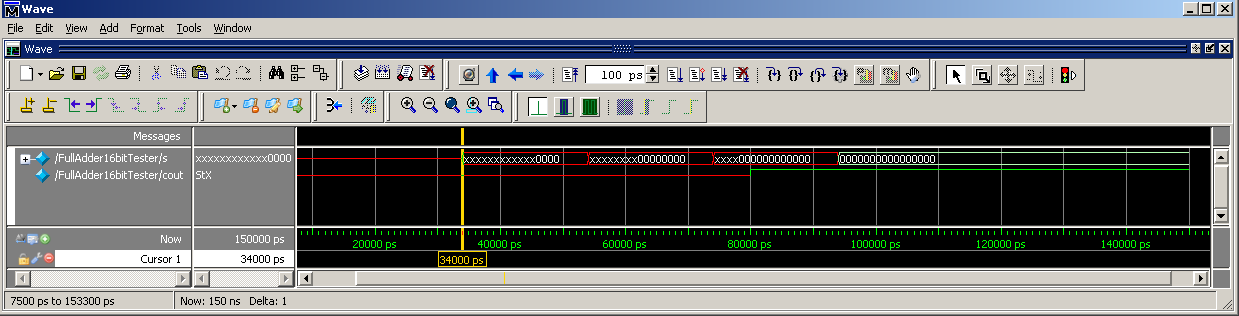
The 1st timing delay on *s* happens at 34ns, which makes sense given the pin-to-pin delay. Subsequent delays are in multiples of 20ns or 34ns. The output *cout* occurs at 80ns, which is expected since its pin-to-pin delay is 20ns, and we are doing four separate adder computations. According to the simulation and what is being tested, the sum and carryout outputs are also correct.

**Simulation Waveform Output of 16bit Full Adder – Simulation 3**



The 1st timing delay on *s* happens at 34ns, which makes sense given the pin-to-pin delay. Subsequent delays are in multiples of 20ns or 34ns. The output *cout* occurs at 80ns, which is expected since its pin-to-pin delay is 20ns, and we are doing four separate adder computations. According to the simulation and what is being tested, the sum and carryout outputs are also correct.

**Simulation Waveform Output of 16bit Full Adder – Simulation 4**



The 1st timing delay on *s* happens at 34ns, which makes sense given the pin-to-pin delay. Subsequent delays are in multiples of 20ns or 34ns. The output *cout* occurs at 80ns, which is expected since its pin-to-pin delay is 20ns, and we are doing four separate adder computations. According to the simulation and what is being tested, the sum and carryout outputs are also correct.

This concludes the analysis for Homework 3, Part 4.

***Homework 3: PART 5***

The Verilog code for this section can be found in the Pt5 folder of the ZIP file submitted:

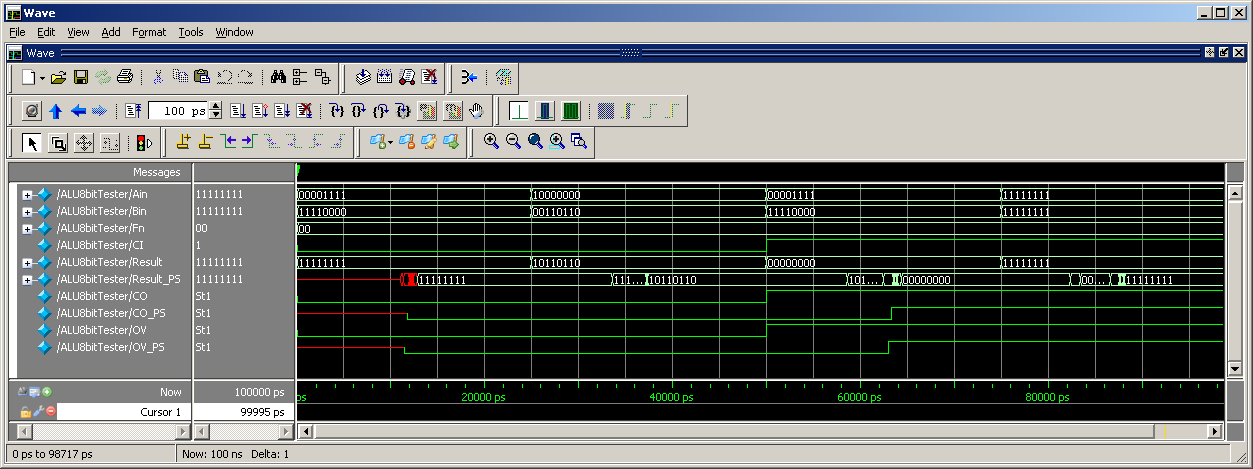
*ALU8bit.v*

*ALU8bit.vo*

*ALU8bitTester.v*

In running the first simulation for the *Ain* + *Bin* functionality, it’s clear that there’s a distinguishable delay between the pre-synthesis outputs and the post-synthesis outputs:

**Simulation Waveform Output – Pre & Post Synthesis Timing Differences**



In the waveform generator, the following delays were measured:

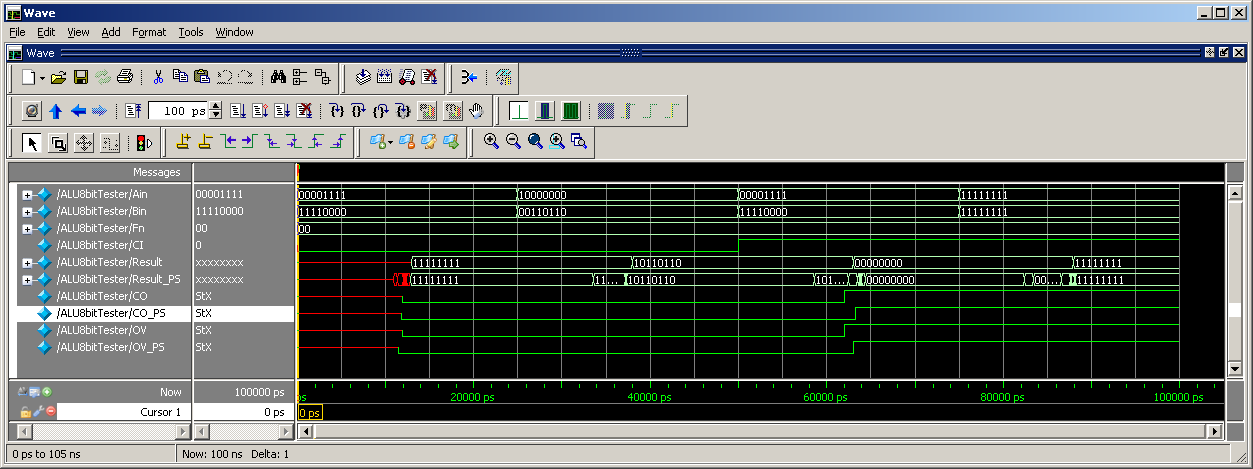
*Result* vs. *Result\_PS* = 13ns

*CO* vs. *CO\_PS* = 12ns

*OV* vs. *OV\_PS* = 12ns

These delays were back-annotated into the ALU8bit.v module, and now the delays between the pre-synthesis and post-synthesis instantiations match very closely, from an output-timing perspective:

**Simulation Waveform Output – Pin-to-Pin Delay Correction**



All other functions in the ALU showcase the same results when run in the testbench.

This concludes the analysis for Homework 3, Part 5.