CprE 381 – Computer Organization and Assembly-Level Programming

Proj-A Report

Lab Partners ____Matthew G. Hoskins____

____Cameron Isbell____

_____

Section / Lab Time ___Section 6 / Th: 6:10pm – 8:00pm___

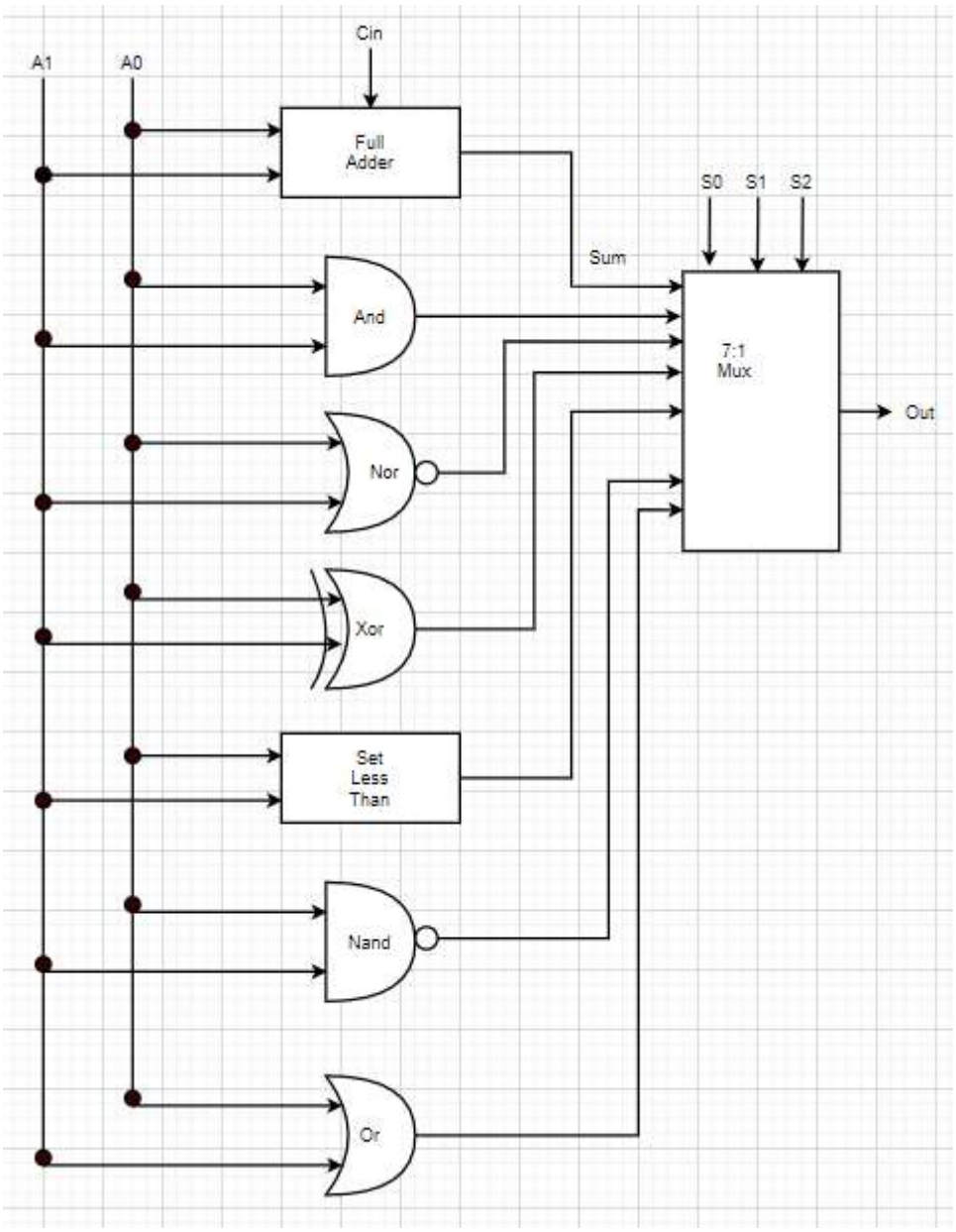
Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the Proj-A instructions for the context of the following questions.

a. [Part 0] Updated Project Team Contract (if applicable). With your project group members, create a list of best practices / tips for designing, compiling, and testing VHDL modules based on your experiences so far with these labs, both working individually and as a group.

   Team Contract Unchanged.
   Best practices / tips: When creating anything that can be based on schematics it is good to create the schematics ahead of time in order to figure out what we will be working for with the vhdl design and setup. Try to keep the naming procedures as clear as possible, so that when using components later on it will still be an easy transferring process. The testing process will be more manageable if done as after finishing the design of the vhdl so that the tests can be very easily made and errors can be found from the vhdl file that is being tested sooner for changing.

b. [Part 1 (a)] Draw a schematic for a 1-bit ALU that supports the following operations: add/sub (both signed and unsigned), slt, and, or, xor, nand, and nor. What are the inputs and outputs that are needed?

   The inputs needed are A1 and A0 as the input values. Cin as the carry in and S0, S1, S2 (all the select values) of the 7 to 1 mux (or 8 to 1 mux which ever tickles your fancy). These are the inputs needed. The outputs needed will be from the 7 to 1 mux and will be the overall output of the ALU. (Schematic pictured below)



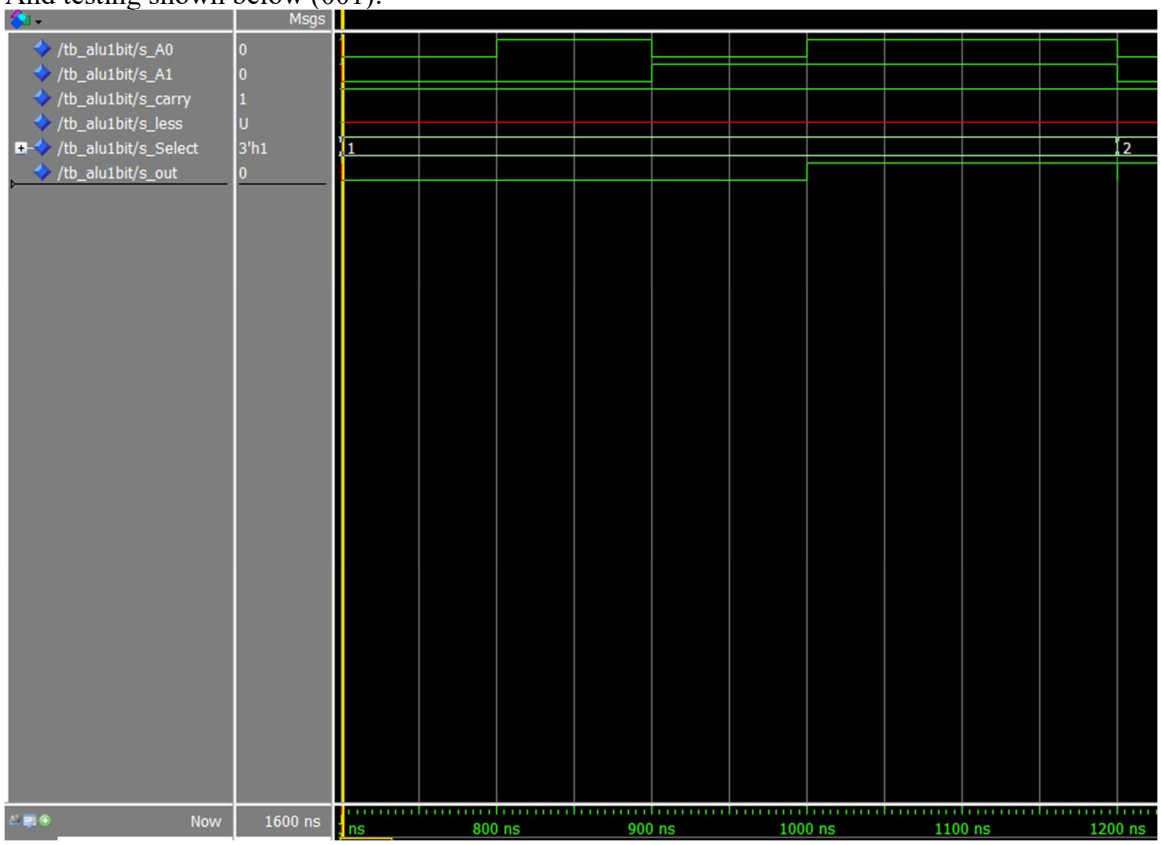
c. [Part 1 (b)] In your project writeup, describe your design in terms of the VHDL coding style you chose and the control signals that are required.

The VHDL coding style that we could use is modular structural style to the ALU. The control signals that are required is the three inputs to the mux.

d. [Part 1 (c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

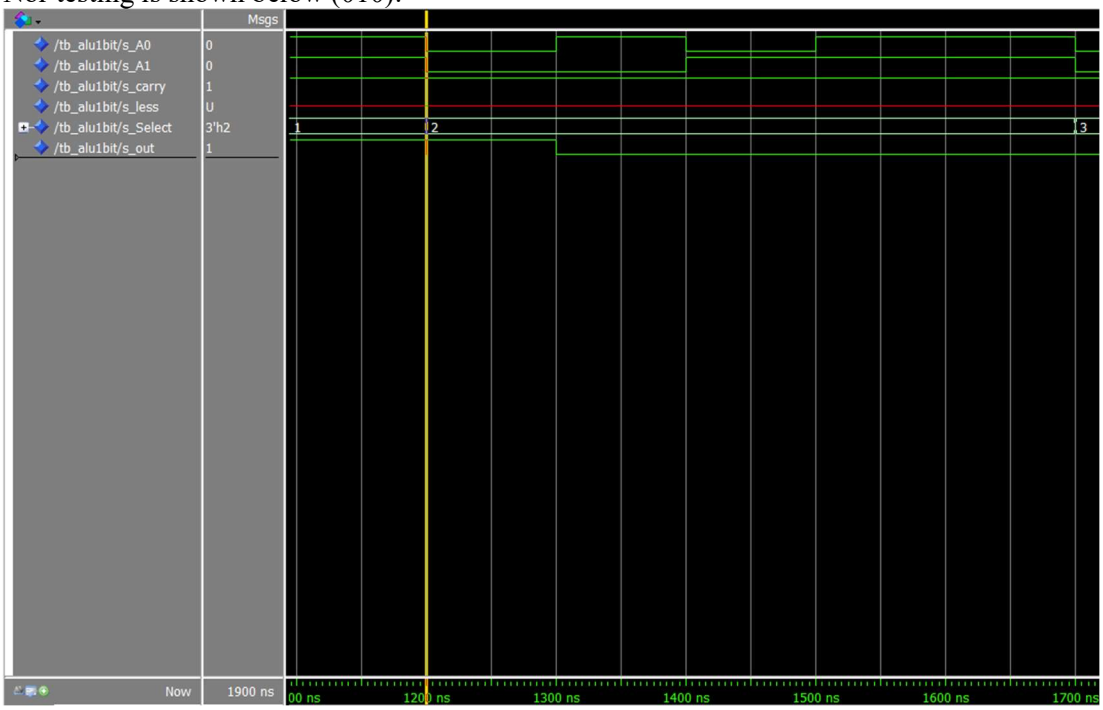
Select X (S2 S1 S0)
Select 0 (000): Adder
Select 1 (001): And
Select 2 (010): Nor
Select 3 (011): Xor
Select 4 (100): SLT
Select 5 (101): Nand
Select 6 (110): Or
Select 7 (111): Extra set to 0 when utilized
Same as the schematics below and above suggest.

And testing shown below (001):



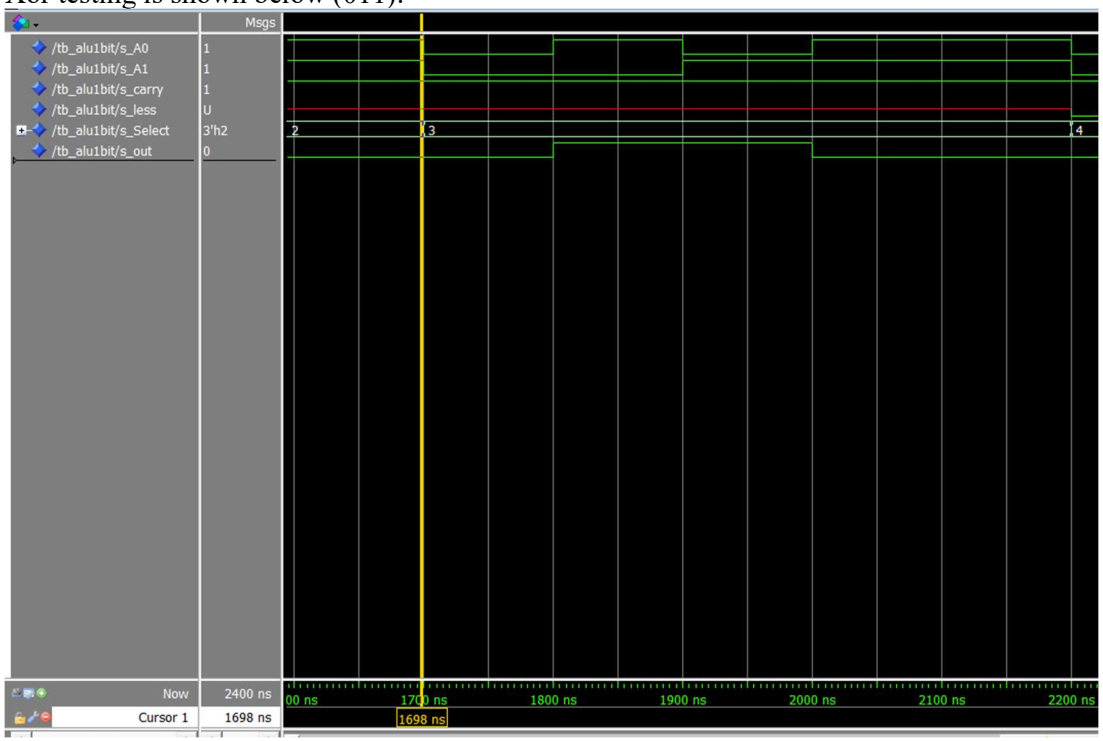
The carry value providing no influence over the results and same to the less input value. The select is at 1 (corresponding to and) and it can be seen that as the A0 and A1 inputs change the output value remains 0 except in the instance that both A0 and A1 are set to value 1; just like an and gate.

Nor testing is shown below (010):



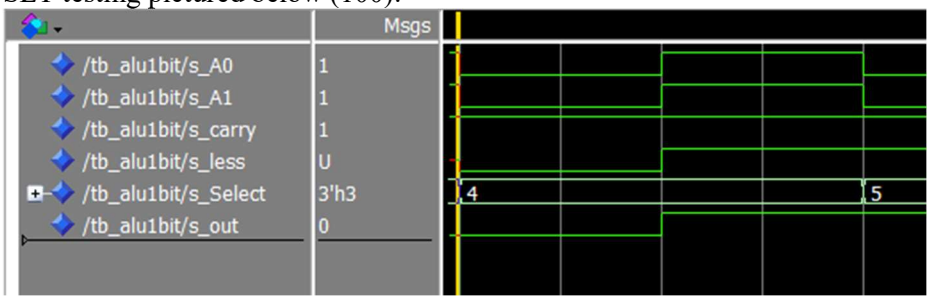
Same parameter in terms of inputs and outputs to that of the and test from above. Again it shows that the out value is 1 only in the instance that a nor gate would set the result to 1 which happens to be when both values are equal to 0.

Xor testing is shown below (011):



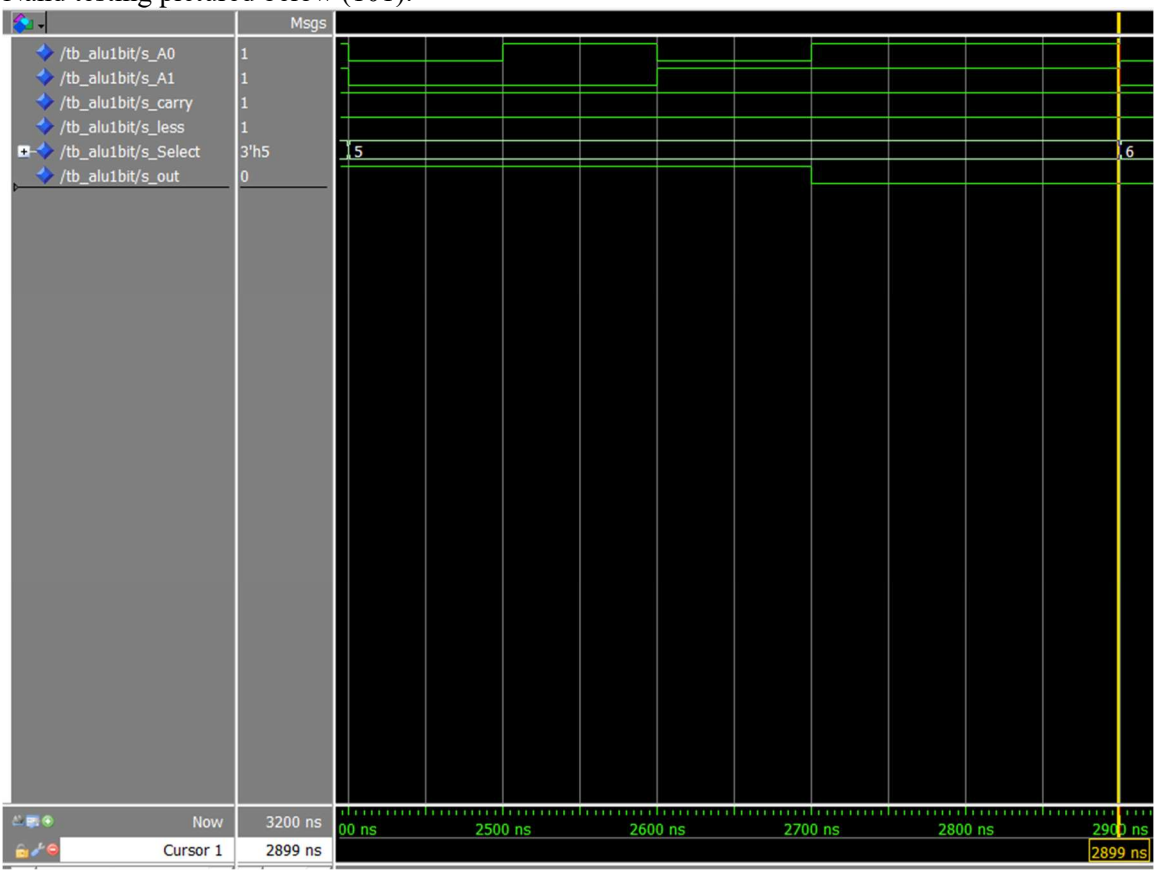
Same parameters as before for input and output. It acts as a xor gate when only one input value is set to 1 thus making the output 1 otherwise it is 0.

SLT testing pictured below (100):



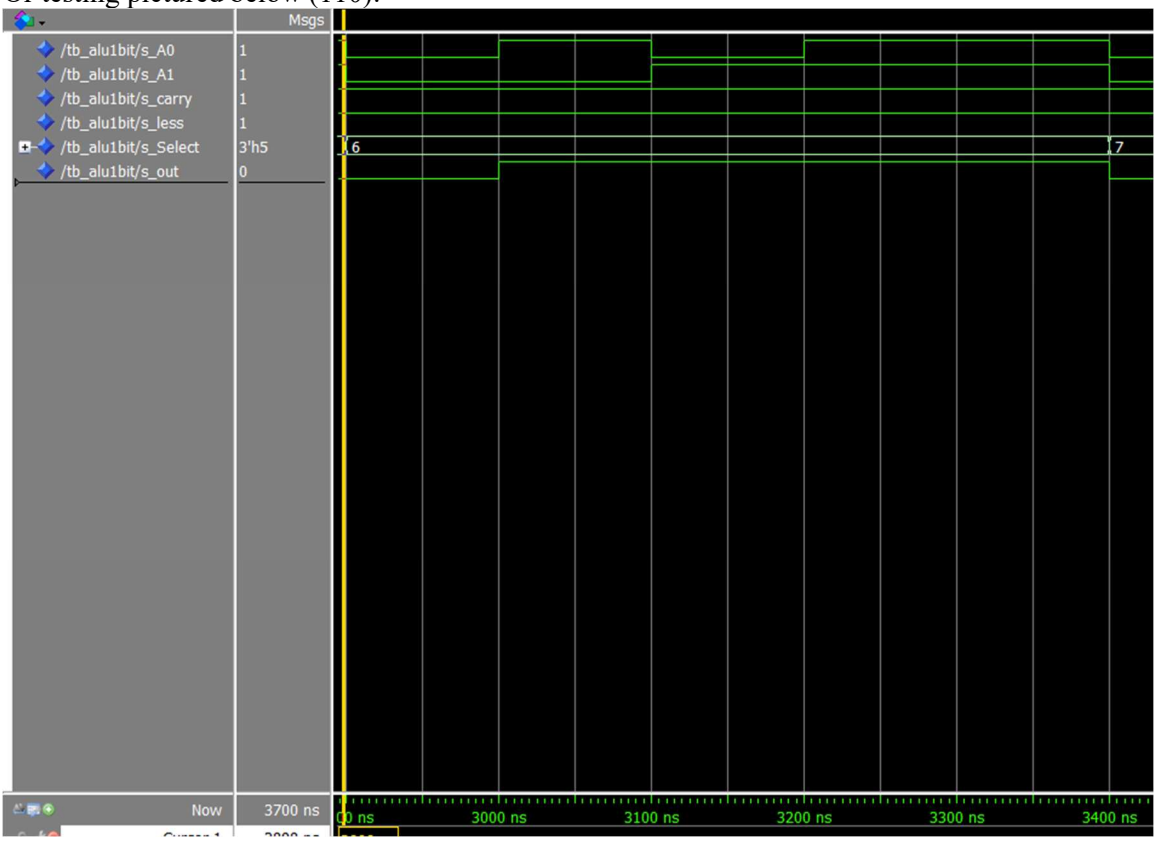
In this the less input matters as an input and so does A0 and A1 like before with the output still being s_out. Here it sets the output to 1 if the less input is set to 1 and that the corresponding value from the input is true.

Nand testing pictured below (101):



Back to the settings from the previous simple gates where the output corresponds to s_out and the only inputs that have an effect here are A0 and A1. It shows the functionality of a Nand gate where the output is 1 as long as the two input values are not both 1.

Or testing pictured below (110):

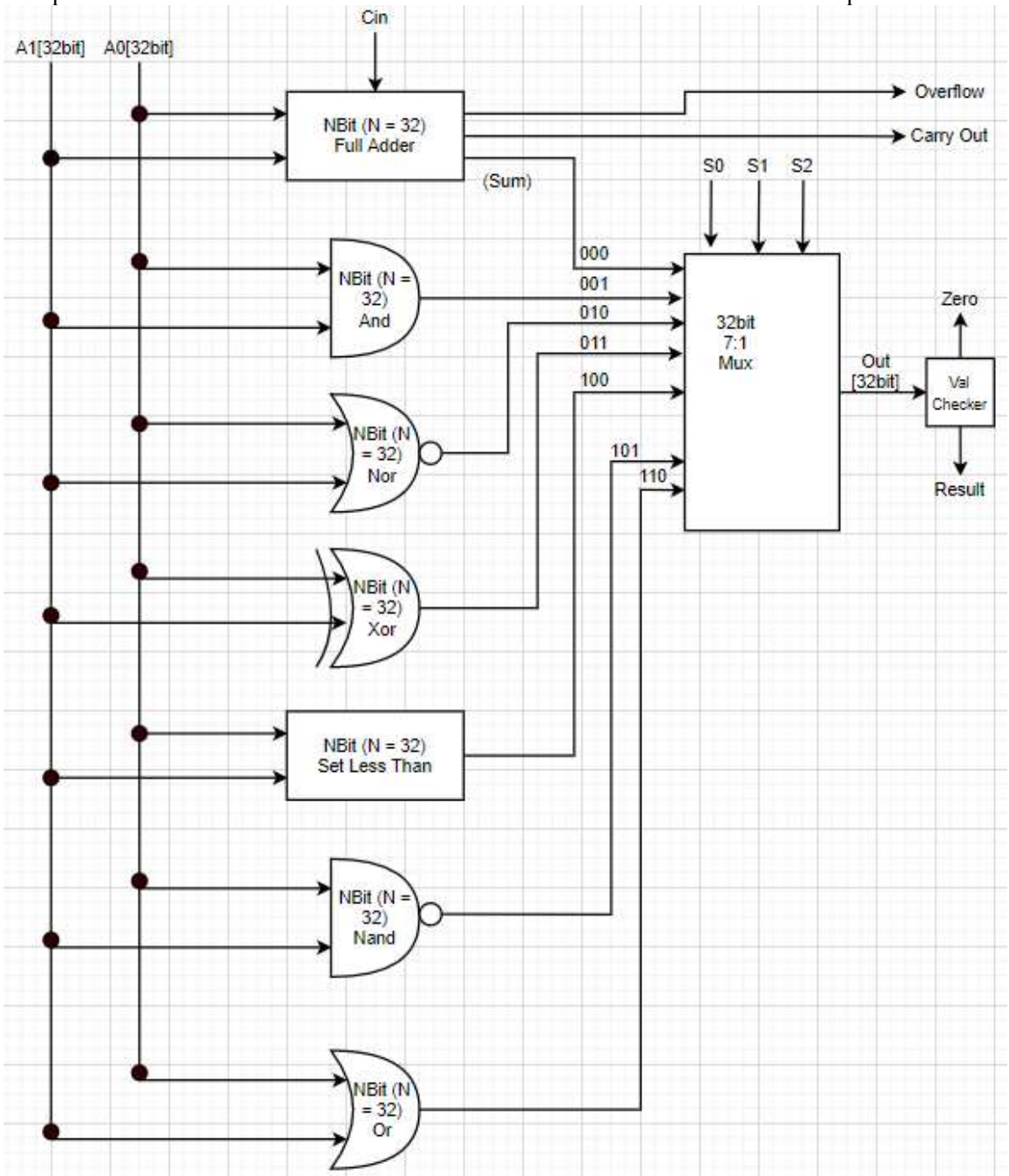


Same setup for inputs and output as above. It show the effect creation of an or gate for the ALU where the output is 1 as long as at least one of the input values are set to 1.

Spare Mux Port testing pictured below (111):

Here is the value of the 8th slot (when you count starting at 1) where there is not another logic unit to place into the ALU, so instead it is an automatic return of 0 if this selection is entered into the mux of the ALU as can be seen from above.

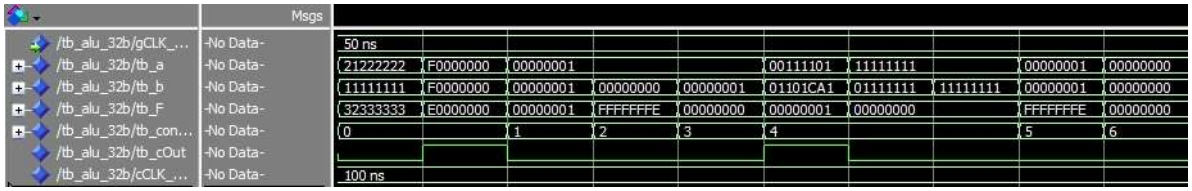
e. [Part 2 (a)] Draw a simplified schematic for this 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is slt implemented?



f. [Part 2 (b)] In your writeup, describe what challenges (if any) you faced in implementing this module.

It openly showed what needed to be changed and or fixed which is beneficial, but it did cause the need to have to go back and review what needed to be fixed. As well as, the need to rethink some logic when involving the continued passing of the 32 bit entities.

g. [Part 2 (c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.



Control signal 0x0 : adder, Cycle 1, 2

Cycle 1 is an adder, as shown by the control signal of 0x0. The point of this test is to show that the adder works properly under normal circumstances.

Cycle 2 is an adder with the purpose of showing how the overflow component works

Control signal 0x1: and, Cycle 3

Cycle 3 shows the "and" function of the ALU. As can be seen, the only 1 bit in the output is the bit where both tb_a and tb_b are 1.

Control signal 0x2: nor, Cycle 4

Cycle 4 shows the "nor" functionality by setting all bits, apart from those where a = 0 and b = = 0, to a1.

Control signal 0x3: xor, Cycle 5

Cycle 5 shows the xor functionality by showing only bits 1 and 0 equate to a 1 output.

Control signal 0x4: slt, Cycle 6, 7, 8

Cycle 6 shows both that slt can determine when multi-bit input A is less than multi-bit input B and that slt sets only the least-significant-bit when A < B.

Cycle 7 shows that slt does not produce set the output bit to 1 if A is not less than B.

Cycle 8 exists for a similar reason as Cycle 7, to show that the output does not equate to a 1 if the values are equivalent.

Control signal 0x6: nand, Cycle 9

Cycle 9 shows that the ALU is capable of nand-ing each bit and producing the correct output.

Control signal 0x7: or, Cycle 10

Cycle 10 shows that the ALU can or each bit from the input and produce the correct output.

h. [Part 3 (a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

SRL being Shift Right Logical and SRA being Shift Right Arithmetic.
SRL treats the number as bunch of bits and shifts *in* zeros (the >> operator in C). While SRA treats the number as a signed integer (in 2's complement) and effectively retains the topmost bit, and then shifts in zeros if the topmost value was 0 but 1's value if the topmost value was 1.
SLA, or Shift Left Arithmetic, is equivalently multiplication by a power of 2 for binary numbers.

i. [Part 3 (b)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

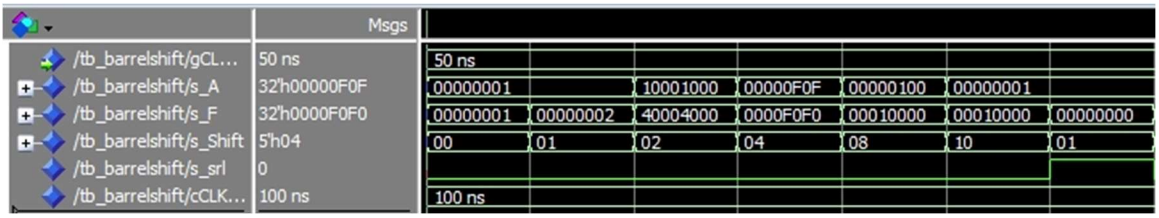
The code takes in a select value as an input that then goes through a multiplexor where if it is a 0 value it keeps it relatively the same as a change to the data is not needed, but if the value is a 1 then it will reverse the input data appropriately.

j. [Part 3 (c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations.

In order to support left shifting operations what would need to be done is the inversion of the data and then to invert the resulting data once the actual shifting process has been completed. This would be done by setting up inverters at both ends that activate via mux selection from something like an input variable.

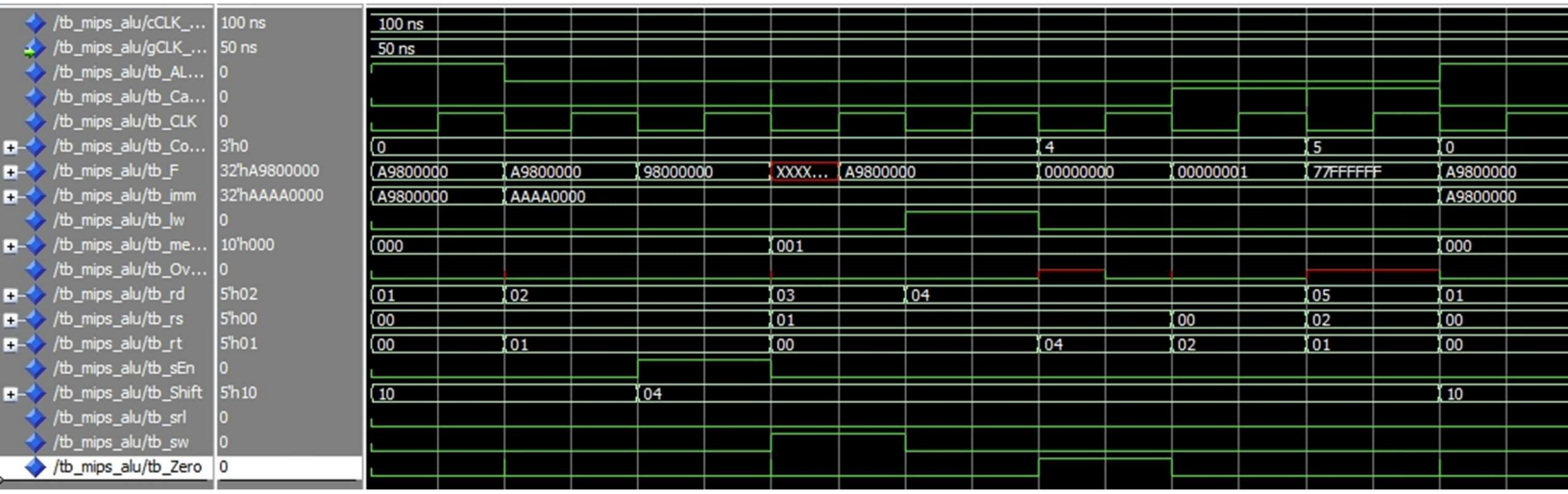
k. [Part 3 (d)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

We implemented two ways, one per partner, for the barrel shifter for full understanding and to be sure we were able to create a functioning barrel shifter. The one Hoskins did was one that focused more on the muxes like the diagram link provided by the lab. Whereas, the one Isbell implemented was created by looping through arrays with the right shifting parameters. The waveform pictured below is a testbench based around Isbell's barrel shifter.

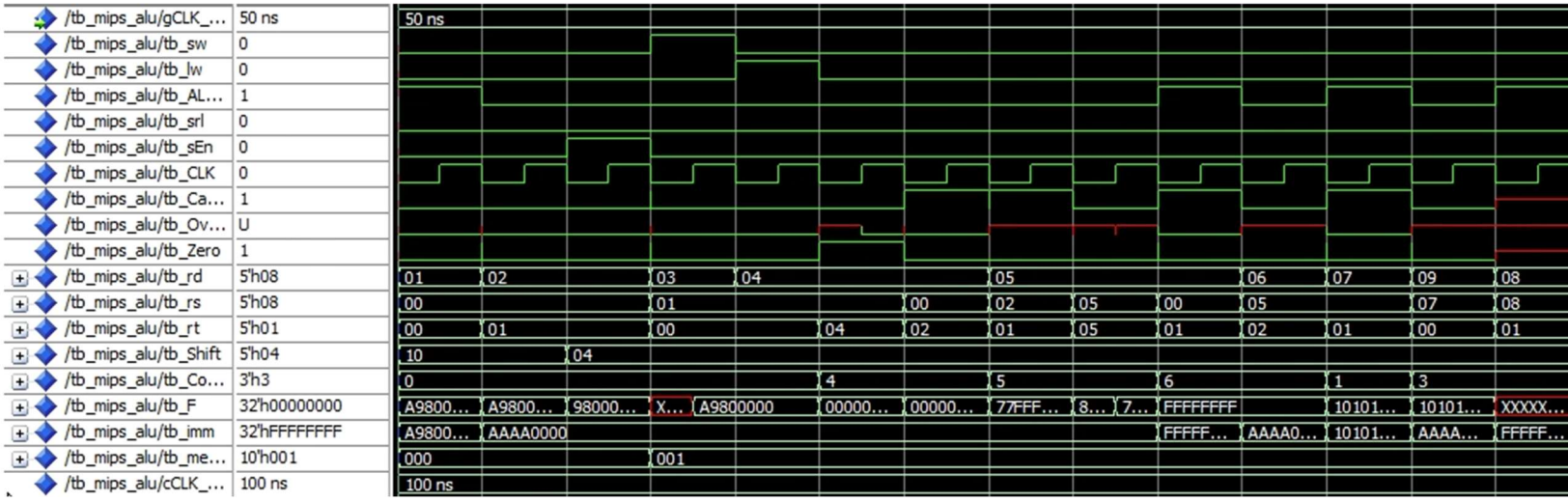


This includes a variety of testing that goes through testing a range of potential instances that could be used. The first test has the shift value set to 0 meaning that the value should not be altered any and that is what comes out of the final of s_F when the s_A is the input value. After that is a test of a single bit change of the value of 00000001 (in hex) and it does successfully shift the bit by one to the right. This continues and proves to be effective and working after that.

l. [Part 4(b)] Justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.



This waveform is the results of the test bench of testing the combination of both the new ALU with the new barrel shifter component all combined with the previous parts that we had (Memory and registers).
The outputs of F, Overflow, memOut, and carry out.
The imm being an initial feeding of values.
The first series of lines is the test going through the process of loading in the value of A9800000 and accomplishing the equivalent of loading it into the first register slot. After this is the addition of AAAA0000 and then the corresponding shift of values to it thus resulting in 98000000. From this point on the imm value is left unchanged as there are now values that we can manipulate loaded into the memory path. By the third series of clocks an effective sw of value 1 to the first register is called. And afterward the lw of that value is then utilize to quickly show its effectiveness. In this series slt is then utilized on the register be effected by the lw and the first one from the earlier steps still. After this is a nanding process to show the altering value and then this specific test bench repeats.



Pictured above is another series of tests that were done on the MIPS ALU that was created for

this lab. This image a more zoomed out version than that of the previous test bench image to show the more expansive and interwovenness of the values that are being worked. Aside from the output the most actively changing values are that of the rd, rd, rt as that is a main focus of this testing series.

m. [Part 5(c) BONUS] Justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.

n. [Feedback] You must complete this section for your lab to be graded. Please complete each column **separately** for each team member; I expect it to take roughly 10 minutes (do not take more than 20 minutes).

i. How many hours did you spend on this lab?

Task	**During lab time**			**Outside of lab time**		
Team Initials	CI	MH		CI	MH	
Reading lab	0.5	.5 h		1 hr	.75 h	
Pencil/paper design	0.5 h	0 h		0.5 h	1 h	
VHDL design	1 h	.5 h		6 hr	2 h	
Assembly coding	0h	0 h		0 h	0 h	
Simulation	0.5 h	1.5 h		2 h	1.5h	
Debugging	0.5 h	1 h		3 h	1.75h	
Report writing	0 h	.5 h		0.5 h	2.5h	
Other:	0 h	0 h		0 h	0 h	
Total	4 h	4 h		13 h	9.5 h	

ii. If you could change one thing about the lab experience, what would it be? Why?

It is more of a change on our end, but it would be to have most of the lab finished by the second lab, so that the time in that session can be spent confirming what we have created during that time.

iii. What was the most interesting part of the lab?

One of the interesting parts was learning about the barrel shifter and having to go through attempting to implement it based off of the provided schematic. It was a very interesting way, but probably the most obvious way to do something like that considering the solution is typically muxes in the gate and architecture world of computers.