CprE 381

Project Part 1 Report

Lab Partners Vishal Joel and Daniel Limanowski

Section/Lab Time B/Thurs @ 10AM

NOTE: Shane told us in class that we would NOT be docked points for a late submission as we just had to include an extra screenshot for the final part and resubmit.

a. [Part 0] 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.

- Using comments on code that may not be straightforward - Liberal comments are mandatory to maintain reusable code.
- Proper indentation to ensure readability and reuse. (1 tab indents)
- Naming convention: All lowercase names with underscores, for readability and name delimiting.
- Naming convention: Signals should be named using 's_' -> s_signal

b. [Part 1 (a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

- The purpose of sra is to support signed numbers represented in two's complement. The most significant bit, which is '1' if the value is negative, is duplicated when shifted right. MIPS does not have an sla instruction because a left-shift arithmetic would shift in zeroes from the LSB - this would be the exact same maneuver as a sll, making sla unnecessary.

c. [Part 1 (b)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

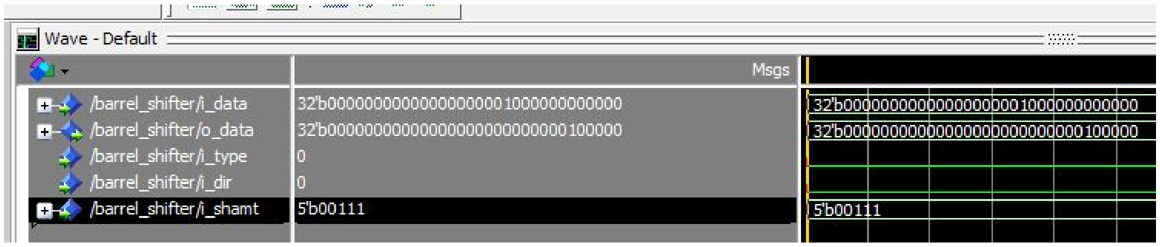
- The VHDL code uses a "with-select" operation to set a "shift bit" if the inputted "type" bit is set to "0" (logical) or "1" (arithmetic). If logical, "shift bit" is set to "0". If arithmetic, "shift bit" is set to the most-significant-bit (MSB) of the inputted data.

d. [Part 1 (c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations.

- The operation of the right barrel shifter need not be altered. A 2-1 multiplexer can be placed in front of the right shifter with a direction bit as the selector. The two inputs to the multiplexer would be 1) the normal input data to be shifted and 2) the inputted data in REVERSE order. This reversing would require another module to be built that is simply a for loop that swaps all the positions in the input vector. data(31) becomes reverse_data(0) and so on. If a left-shift is desired, the selector chooses the reversed data to be inputted to the right-shift module. After the shift is performed, the left-shift data is reversed one more time to bring it to proper order.

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

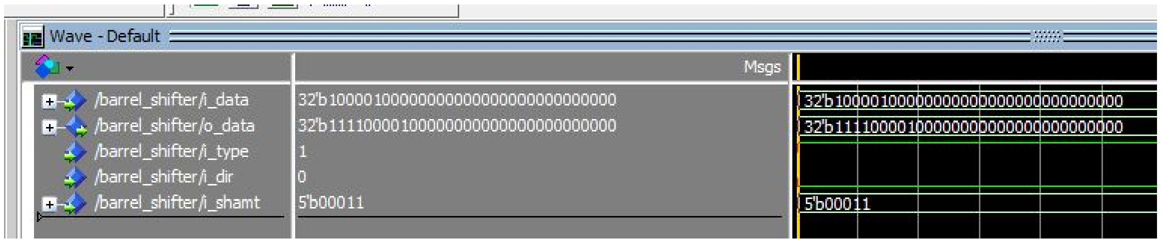
Shift-right-logical (SRL) shifts in zeroes from MSB while moving the data towards LSB.



Wave - Default

```
/barrel_shifter/i_data    32'b00000000000000000001000000000000
/barrel_shifter/o_data    32'b00000000000000000000000000100000
/barrel_shifter/i_type    0
/barrel_shifter/i_dir     0
/barrel_shifter/i_shamt   5'b00111
```

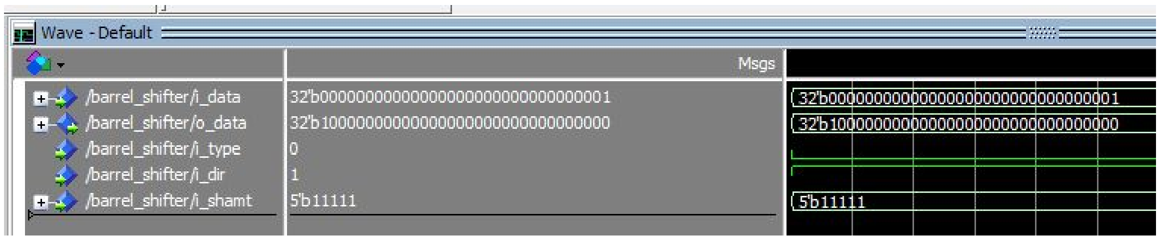
Shift-right-arithmetic (SRA) shifts in whatever the MSB is while moving data towards LSB.



Wave - Default

```
/barrel_shifter/i_data    32'b10000100000000000000000000000000
/barrel_shifter/o_data    32'b11110000100000000000000000000000
/barrel_shifter/i_type    1
/barrel_shifter/i_dir     0
/barrel_shifter/i_shamt   5'b00011
```

Shift-left-logical (SLL) shifts in zeroes from the LSB while moving data towards MSB.



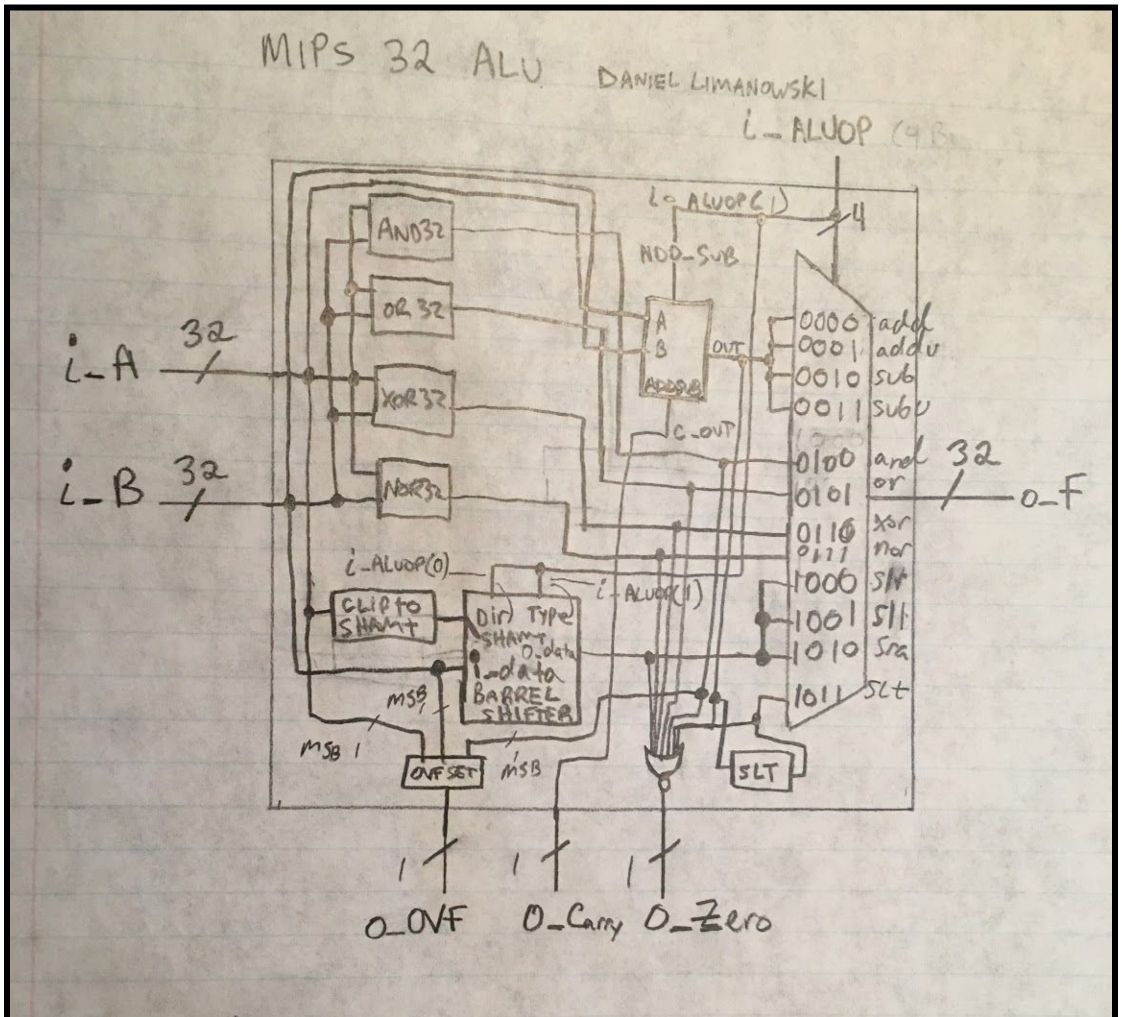
Wave - Default

```
/barrel_shifter/i_data    32'b00000000000000000000000000000001
/barrel_shifter/o_data    32'b10000000000000000000000000000000
/barrel_shifter/i_type    0
/barrel_shifter/i_dir     1
/barrel_shifter/i_shamt   5'b11111
```

f. [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?

- SLT implementation: Set-less-than instruction is implemented by checking if the result of (a - b) is negative. If negative, (a < b) is true, so we set the LSB of Result (Result(0)) to ‘1’. If (a-b) is positive, we set Result(0) = ‘0’. All other bits of Result (ie., 1..31) are always set to ‘0’ regardless. A Less input is added to each 1-bit ALU and for ALU1-30, the Less input is forced to ‘0’. ALU0’s Less is set to ALU31’s Set output.

- Zero is calculated by checking all inputs to our ALUOP mux. If any of these inputs are greater than zero, the Zero flag is set to off. If ALL of these inputs are zero, the zero flag is set to on. Using a NOR with the mux inputs as input, the output is our zero flag value.
- Overflow is detected by checking the values of MSB of A, B, and Result of Add/Sub. If (the MSB of A = B) AND (the MSB of Result != (A or B's MSB)), then set Overflow flag. Overflow (OVF) is ONLY looked at in SIGNED operations (add, sub).
- 32-bit ALU simple schematic (NOTE: ALUOPcodes have changed. See Mux7-1 file for current opcodes):



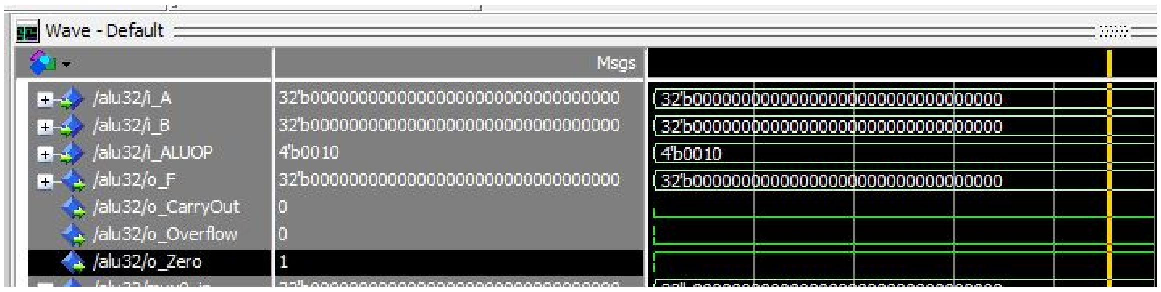
INPUTS:	i_A, i_B	= ARGUMENTS
	i_ALUOP	= MUX SELECTOR, DECIDES OPERATION
OUTPUTS:	o_F	= RESULT
	o_OVF	= OVERFLOW FLAG
	o_carry	= CARRY FLAG
	o_zero	= ZERO FLAG

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

Wrapping my head around the entire ALU unit, with all of its operations, was a challenge. However, once I drew in detail the ALU and all of the components needed to implement it, actually writing the VHDL code wasn't all that bad. Office hours were a good assist in understanding what was expected from the ALU and how to get started.

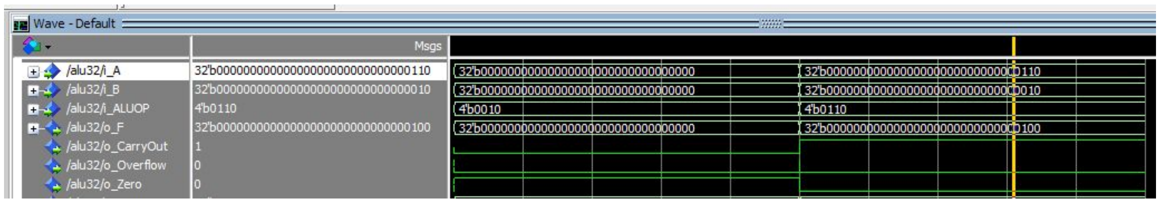
h. [Part 2 (c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

ALUOP for Add = 0b0010. We added 0+0 and the result was 0. You can see the Zero flag is properly set in the figure below.



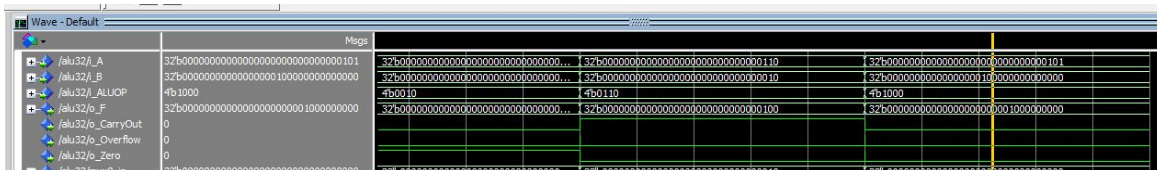
```
Wave - Default
                         Msgs
/alu32/i_A          32'b00000000000000000000000000000000
/alu32/i_B          32'b00000000000000000000000000000000
/alu32/i_ALUOP      4'b0010
/alu32/o_F          32'b00000000000000000000000000000000
/alu32/o_CarryOut   0
/alu32/o_Overflow   0
/alu32/o_Zero       1
```

ALUOP for Sub = 0b0110. We subtracted 0b010 from 0b110 and obtained 0b100.



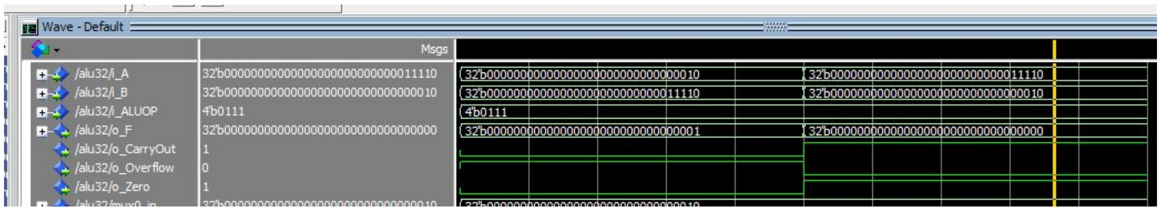
```
Wave - Default
                         Msgs
/alu32/i_A          32'b00000000000000000000000000000110
/alu32/i_B          32'b00000000000000000000000000000010
/alu32/i_ALUOP      4'b0110
/alu32/o_F          32'b00000000000000000000000000000100
/alu32/o_CarryOut   1
/alu32/o_Overflow   0
/alu32/o_Zero       0
```

ALUOP for SRL is 0b1000. SHAMT (shift amount) is i_A, set to shift right 5 bits. The register data to shift is i_B, a lonely bit in a sea of 31 zeroes. You can tell the bit is properly shifted right 5 bits.



```
Wave - Default
                         Msgs
/alu32/i_A          32'b00000000000000000000000000000101
/alu32/i_B          32'b00000000000000000100000000000000
/alu32/i_ALUOP      4'b1000
/alu32/o_F          32'b00000000000000000000001000000000
/alu32/o_CarryOut   0
/alu32/o_Overflow   0
/alu32/o_Zero       0
```

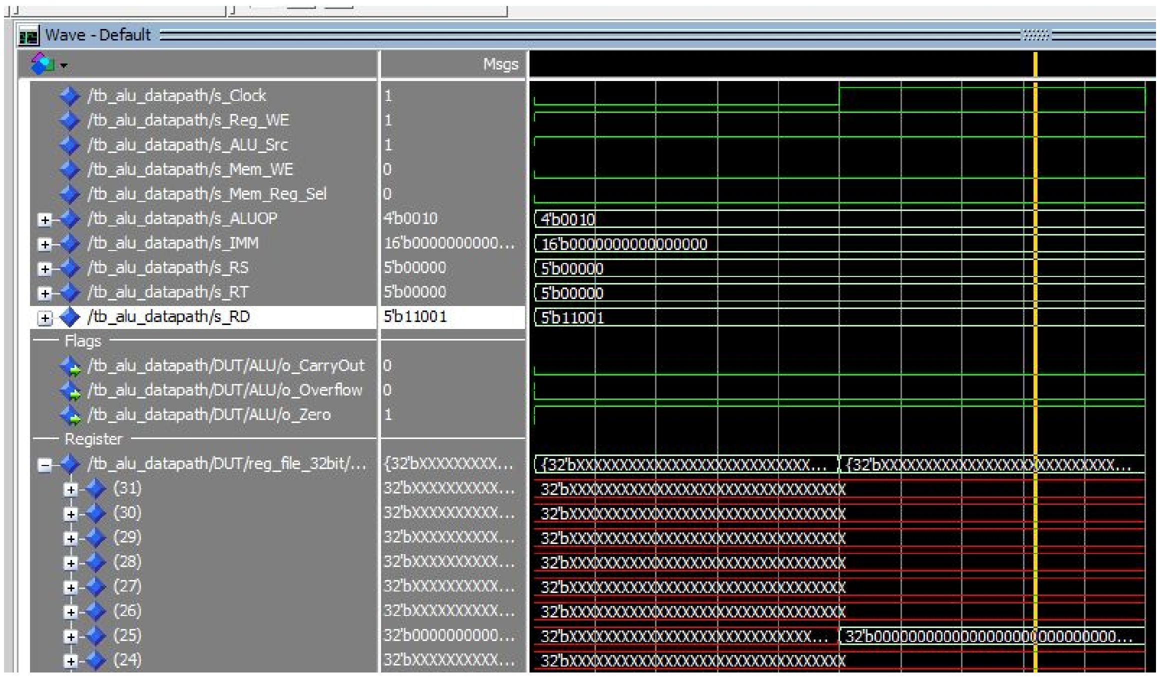
ALUOP for SLT is 0b0111. First we set A<B and see the answer is 1 (correct). Then we set A>B and see the result is also correct (= 0).



```
Wave - Default
                         Msgs
/alu32/i_A          32'b00000000000000000000000000011110
/alu32/i_B          32'b00000000000000000000000000000010
/alu32/i_ALUOP      4'b0111
/alu32/o_F          32'b00000000000000000000000000000000
/alu32/o_CarryOut   1
/alu32/o_Overflow   0
/alu32/o_Zero       1
```

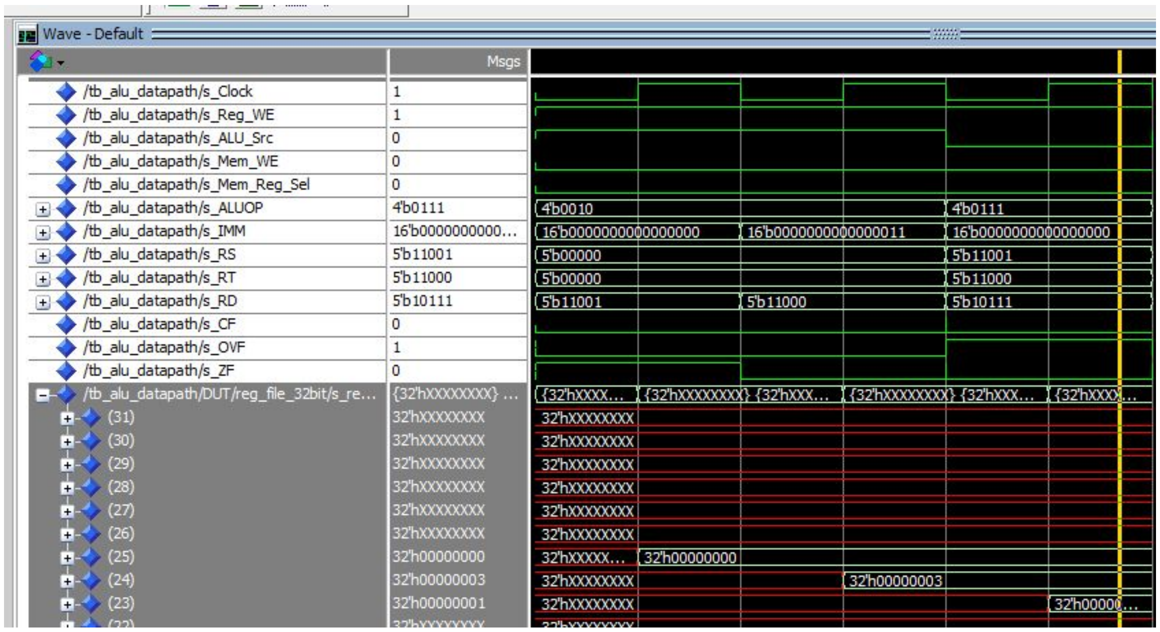
i. [Part 3(b)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

Our test plan is comprehensive because I developed a testbench to ensure accuracy for all the possible commands. I have included in the report a few screenshots from my tests.



```
Wave - Default
                                             Msgs
/tb_alu_datapath/s_Clock                     1
/tb_alu_datapath/s_Reg_WE                    1
/tb_alu_datapath/s_ALU_Src                   1
/tb_alu_datapath/s_Mem_WE                    0
/tb_alu_datapath/s_Mem_Reg_Sel               0
/tb_alu_datapath/s_ALUOP                     4'b0010                4'b0010
/tb_alu_datapath/s_IMM                       16'b0000000000...      16'b0000000000000000
/tb_alu_datapath/s_RS                        5'b00000               5'b00000
/tb_alu_datapath/s_RT                        5'b00000               5'b00000
/tb_alu_datapath/s_RD                        5'b11001               5'b11001
Flags
/tb_alu_datapath/DUT/ALU/o_CarryOut          0
/tb_alu_datapath/DUT/ALU/o_Overflow          0
/tb_alu_datapath/DUT/ALU/o_Zero              1
Register
/tb_alu_datapath/DUT/reg_file_32bit/...      {32'bXXXXXXXXX...      {32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...  {32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXX...
(31)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(30)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(29)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(28)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(27)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(26)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(25)                                         32'b0000000000...      32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...  32'b000000000000000000000000000...
(24)                                         32'bXXXXXXXXXXX...     32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

The figure above shows addition of the zero register with an immediate “0”. The ALUOP for addition is “0010” as seen in the waveform.



```
Wave - Default
                                             Msgs
/tb_alu_datapath/s_Clock                     1
/tb_alu_datapath/s_Reg_WE                    1
/tb_alu_datapath/s_ALU_Src                   0
/tb_alu_datapath/s_Mem_WE                    0
/tb_alu_datapath/s_Mem_Reg_Sel               0
/tb_alu_datapath/s_ALUOP                     4'b0111                4'b0010                                          4'b0111
/tb_alu_datapath/s_IMM                       16'b0000000000...      16'b0000000000000000  16'b0000000000000011       16'b0000000000000000
/tb_alu_datapath/s_RS                        5'b11001               5'b00000                                         5'b11001
/tb_alu_datapath/s_RT                        5'b11000               5'b00000                                         5'b11000
/tb_alu_datapath/s_RD                        5'b10111               5'b11001              5'b11000                   5'b10111
/tb_alu_datapath/s_CF                        0
/tb_alu_datapath/s_OVF                       1
/tb_alu_datapath/s_ZF                        0
/tb_alu_datapath/DUT/reg_file_32bit/s_re...  {32'hXXXXXXXX} ...     {32'hXXXX... {32'hXXXXXXXX} {32'hXXX... {32'hXXXXXXXX} {32'hXXX... {32'hXXXX...
(31)                                         32'hXXXXXXXX           32'hXXXXXXXX
(30)                                         32'hXXXXXXXX           32'hXXXXXXXX
(29)                                         32'hXXXXXXXX           32'hXXXXXXXX
(28)                                         32'hXXXXXXXX           32'hXXXXXXXX
(27)                                         32'hXXXXXXXX           32'hXXXXXXXX
(26)                                         32'hXXXXXXXX           32'hXXXXXXXX
(25)                                         32'h00000000           32'hXXXXX...  32'h00000000
(24)                                         32'h00000003           32'hXXXXXXXX                32'h00000003
(23)                                         32'h00000001           32'hXXXXXXXX                              32'h00000...
```

The above image shows the SLT operation in working condition. We compare register 25 (holding a value of 0) to register 24 (holding a value of 3). 3 is clearly greater than 0, so SLT is set to 1 (whose result is placed into $23, as seen in the waveform).

j. [Part 4(d)] Does your design meet the timing constraints that were set? Report the critical path from register to register using TimeQuest. What is the fastest clock period that your MIPS datapath can be safely run at?

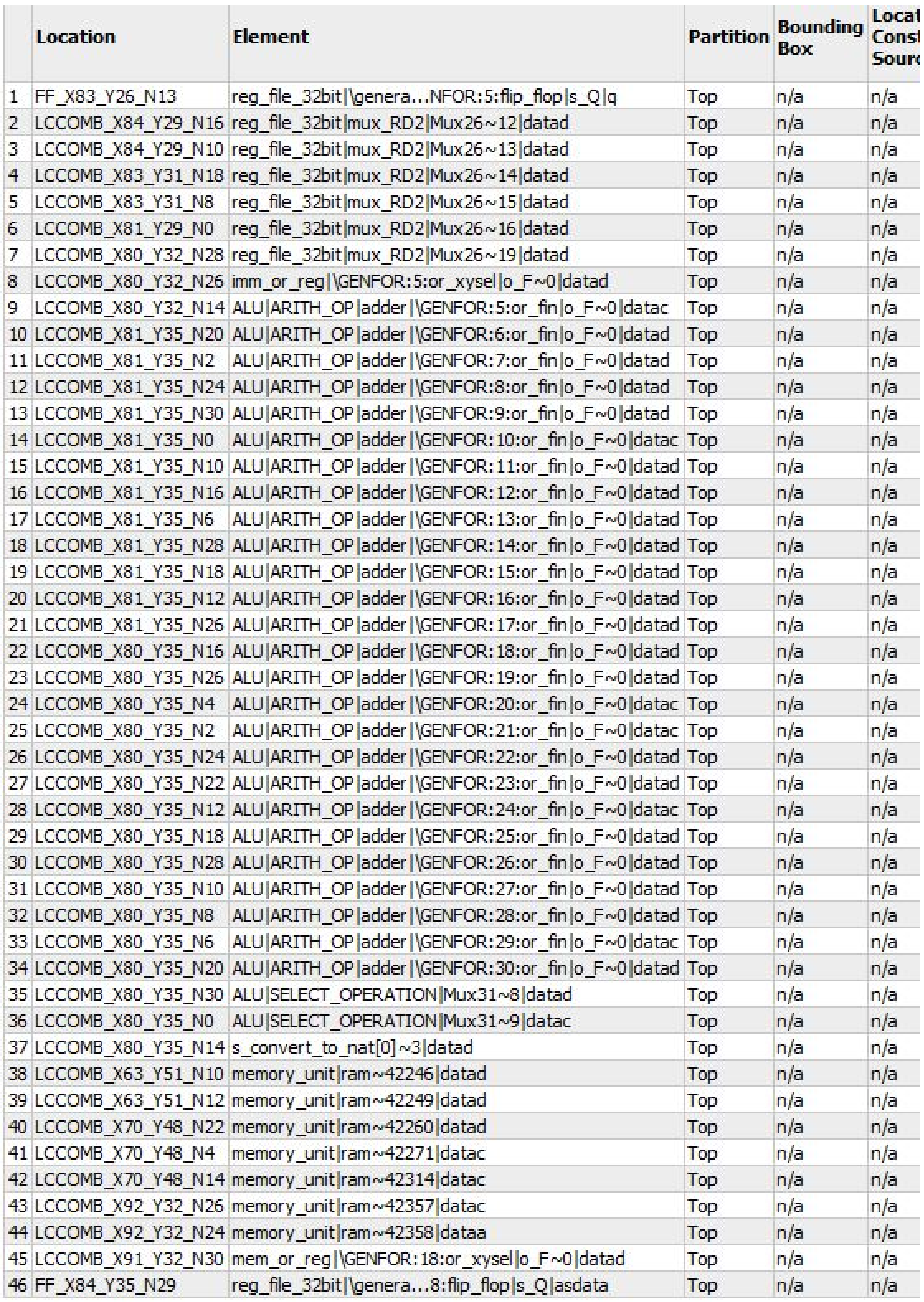
To implement a 50 MHz clock, the period was set to 20 ns and the rise was set to 10 ns.

Our design did NOT meet the timing constraints (50 MHz clock) that were set. We had a negative slack (at worst) of -11.67 ns.

Our highest "slack" value was -11.767 nanoseconds (ns) for the flip flops located inside of the register file. This is our limiting factor in terms of speed. A negative slack indicates that our slack went OVER the clock period. Instead of completing the critical path within 20 ns, we took 31.767 ns in our worst-case.

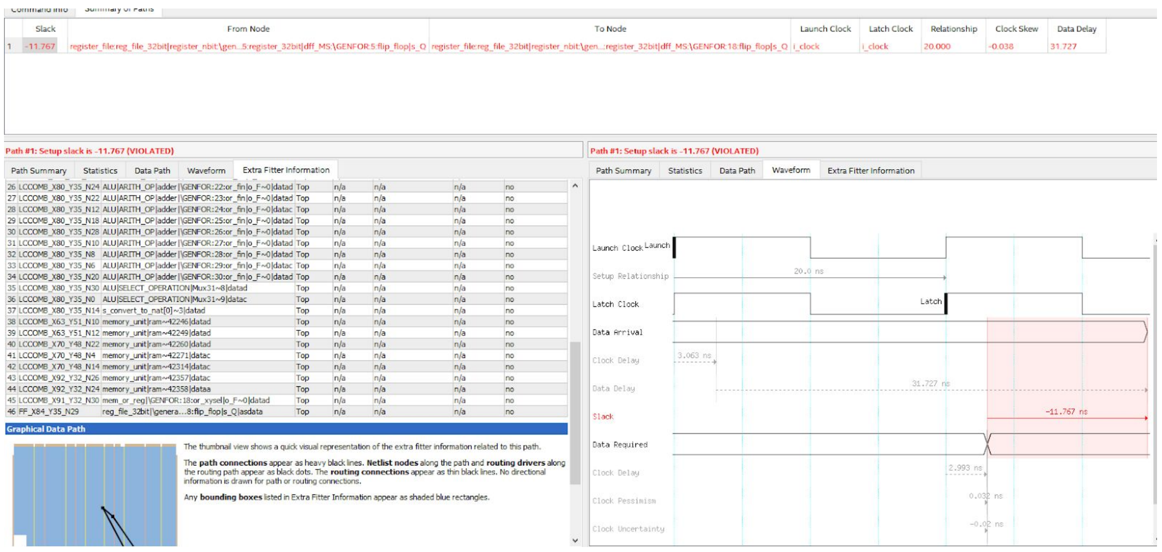
The fastest clock period we can run our datapath at is 31 MHz.

Below shows the critical path:



	Location	Element	Partition	Bounding Box	Locat Const Sourc
1	FF_X83_Y26_N13	reg_file_32bit|\genera...NFOR:5:flip_flop|s_Q|q	Top	n/a	n/a
2	LCCOMB_X84_Y29_N16	reg_file_32bit|mux_RD2|Mux26~12|datad	Top	n/a	n/a
3	LCCOMB_X84_Y29_N10	reg_file_32bit|mux_RD2|Mux26~13|datad	Top	n/a	n/a
4	LCCOMB_X83_Y31_N18	reg_file_32bit|mux_RD2|Mux26~14|datad	Top	n/a	n/a
5	LCCOMB_X83_Y31_N8	reg_file_32bit|mux_RD2|Mux26~15|datad	Top	n/a	n/a
6	LCCOMB_X81_Y29_N0	reg_file_32bit|mux_RD2|Mux26~16|datad	Top	n/a	n/a
7	LCCOMB_X80_Y32_N28	reg_file_32bit|mux_RD2|Mux26~19|datad	Top	n/a	n/a
8	LCCOMB_X80_Y32_N26	imm_or_reg|\GENFOR:5:or_xysel|o_F~0|datad	Top	n/a	n/a
9	LCCOMB_X80_Y32_N14	ALU|ARITH_OP|adder|\GENFOR:5:or_fin|o_F~0|datac	Top	n/a	n/a
10	LCCOMB_X81_Y35_N20	ALU|ARITH_OP|adder|\GENFOR:6:or_fin|o_F~0|datad	Top	n/a	n/a
11	LCCOMB_X81_Y35_N2	ALU|ARITH_OP|adder|\GENFOR:7:or_fin|o_F~0|datad	Top	n/a	n/a
12	LCCOMB_X81_Y35_N24	ALU|ARITH_OP|adder|\GENFOR:8:or_fin|o_F~0|datad	Top	n/a	n/a
13	LCCOMB_X81_Y35_N30	ALU|ARITH_OP|adder|\GENFOR:9:or_fin|o_F~0|datad	Top	n/a	n/a
14	LCCOMB_X81_Y35_N0	ALU|ARITH_OP|adder|\GENFOR:10:or_fin|o_F~0|datac	Top	n/a	n/a
15	LCCOMB_X81_Y35_N10	ALU|ARITH_OP|adder|\GENFOR:11:or_fin|o_F~0|datad	Top	n/a	n/a
16	LCCOMB_X81_Y35_N16	ALU|ARITH_OP|adder|\GENFOR:12:or_fin|o_F~0|datad	Top	n/a	n/a
17	LCCOMB_X81_Y35_N6	ALU|ARITH_OP|adder|\GENFOR:13:or_fin|o_F~0|datad	Top	n/a	n/a
18	LCCOMB_X81_Y35_N28	ALU|ARITH_OP|adder|\GENFOR:14:or_fin|o_F~0|datad	Top	n/a	n/a
19	LCCOMB_X81_Y35_N18	ALU|ARITH_OP|adder|\GENFOR:15:or_fin|o_F~0|datad	Top	n/a	n/a
20	LCCOMB_X81_Y35_N12	ALU|ARITH_OP|adder|\GENFOR:16:or_fin|o_F~0|datad	Top	n/a	n/a
21	LCCOMB_X81_Y35_N26	ALU|ARITH_OP|adder|\GENFOR:17:or_fin|o_F~0|datad	Top	n/a	n/a
22	LCCOMB_X80_Y35_N16	ALU|ARITH_OP|adder|\GENFOR:18:or_fin|o_F~0|datad	Top	n/a	n/a
23	LCCOMB_X80_Y35_N26	ALU|ARITH_OP|adder|\GENFOR:19:or_fin|o_F~0|datad	Top	n/a	n/a
24	LCCOMB_X80_Y35_N4	ALU|ARITH_OP|adder|\GENFOR:20:or_fin|o_F~0|datac	Top	n/a	n/a
25	LCCOMB_X80_Y35_N2	ALU|ARITH_OP|adder|\GENFOR:21:or_fin|o_F~0|datac	Top	n/a	n/a
26	LCCOMB_X80_Y35_N24	ALU|ARITH_OP|adder|\GENFOR:22:or_fin|o_F~0|datad	Top	n/a	n/a
27	LCCOMB_X80_Y35_N22	ALU|ARITH_OP|adder|\GENFOR:23:or_fin|o_F~0|datad	Top	n/a	n/a
28	LCCOMB_X80_Y35_N12	ALU|ARITH_OP|adder|\GENFOR:24:or_fin|o_F~0|datac	Top	n/a	n/a
29	LCCOMB_X80_Y35_N18	ALU|ARITH_OP|adder|\GENFOR:25:or_fin|o_F~0|datad	Top	n/a	n/a
30	LCCOMB_X80_Y35_N28	ALU|ARITH_OP|adder|\GENFOR:26:or_fin|o_F~0|datad	Top	n/a	n/a
31	LCCOMB_X80_Y35_N10	ALU|ARITH_OP|adder|\GENFOR:27:or_fin|o_F~0|datad	Top	n/a	n/a
32	LCCOMB_X80_Y35_N8	ALU|ARITH_OP|adder|\GENFOR:28:or_fin|o_F~0|datad	Top	n/a	n/a
33	LCCOMB_X80_Y35_N6	ALU|ARITH_OP|adder|\GENFOR:29:or_fin|o_F~0|datac	Top	n/a	n/a
34	LCCOMB_X80_Y35_N20	ALU|ARITH_OP|adder|\GENFOR:30:or_fin|o_F~0|datad	Top	n/a	n/a
35	LCCOMB_X80_Y35_N30	ALU|SELECT_OPERATION|Mux31~8|datad	Top	n/a	n/a
36	LCCOMB_X80_Y35_N0	ALU|SELECT_OPERATION|Mux31~9|datac	Top	n/a	n/a
37	LCCOMB_X80_Y35_N14	s_convert_to_nat[0]~3|datad	Top	n/a	n/a
38	LCCOMB_X63_Y51_N10	memory_unit|ram~42246|datad	Top	n/a	n/a
39	LCCOMB_X63_Y51_N12	memory_unit|ram~42249|datad	Top	n/a	n/a
40	LCCOMB_X70_Y48_N22	memory_unit|ram~42260|datad	Top	n/a	n/a
41	LCCOMB_X70_Y48_N4	memory_unit|ram~42271|datac	Top	n/a	n/a
42	LCCOMB_X70_Y48_N14	memory_unit|ram~42314|datac	Top	n/a	n/a
43	LCCOMB_X92_Y32_N26	memory_unit|ram~42357|datac	Top	n/a	n/a
44	LCCOMB_X92_Y32_N24	memory_unit|ram~42358|dataa	Top	n/a	n/a
45	LCCOMB_X91_Y32_N30	mem_or_reg|\GENFOR:18:or_xysel|o_F~0|datad	Top	n/a	n/a
46	FF_X84_Y35_N29	reg_file_32bit|\genera...8:flip_flop|s_Q|asdata	Top	n/a	n/a

A general screenshot from the timing window after analysis:



Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-11.767	register_file:reg_file_32bit|register_nbit:\gen...5:register_32bit|dff_MS:\GENFOR:5:flip_flop|s_Q	register_file:reg_file_32bit|register_nbit:\gen...:register_32bit|dff_MS:\GENFOR:18:flip_flop|s_Q	i_clock	i_clock	20.000	-0.038	31.727

Path #1: Setup slack is -11.767 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

26	LCCOMB_X80_Y35_N24	ALU|ARITH_OP|adder|\GENFOR:22:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
27	LCCOMB_X80_Y35_N22	ALU|ARITH_OP|adder|\GENFOR:23:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
28	LCCOMB_X80_Y35_N12	ALU|ARITH_OP|adder|\GENFOR:24:or_fin|o_F~0|datac	Top	n/a	n/a	n/a	no
29	LCCOMB_X80_Y35_N18	ALU|ARITH_OP|adder|\GENFOR:25:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
30	LCCOMB_X80_Y35_N28	ALU|ARITH_OP|adder|\GENFOR:26:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
31	LCCOMB_X80_Y35_N10	ALU|ARITH_OP|adder|\GENFOR:27:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
32	LCCOMB_X80_Y35_N8	ALU|ARITH_OP|adder|\GENFOR:28:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
33	LCCOMB_X80_Y35_N6	ALU|ARITH_OP|adder|\GENFOR:29:or_fin|o_F~0|datac	Top	n/a	n/a	n/a	no
34	LCCOMB_X80_Y35_N20	ALU|ARITH_OP|adder|\GENFOR:30:or_fin|o_F~0|datad	Top	n/a	n/a	n/a	no
35	LCCOMB_X80_Y35_N30	ALU|SELECT_OPERATION|Mux31~8|datad	Top	n/a	n/a	n/a	no
36	LCCOMB_X80_Y35_N0	ALU|SELECT_OPERATION|Mux31~9|datac	Top	n/a	n/a	n/a	no
37	LCCOMB_X80_Y35_N14	s_convert_to_nat[0]~3|datad	Top	n/a	n/a	n/a	no
38	LCCOMB_X63_Y51_N10	memory_unit|ram~42246|datad	Top	n/a	n/a	n/a	no
39	LCCOMB_X63_Y51_N12	memory_unit|ram~42249|datad	Top	n/a	n/a	n/a	no
40	LCCOMB_X70_Y48_N22	memory_unit|ram~42260|datad	Top	n/a	n/a	n/a	no
41	LCCOMB_X70_Y48_N4	memory_unit|ram~42271|datac	Top	n/a	n/a	n/a	no
42	LCCOMB_X70_Y48_N14	memory_unit|ram~42314|datac	Top	n/a	n/a	n/a	no
43	LCCOMB_X92_Y32_N26	memory_unit|ram~42357|datac	Top	n/a	n/a	n/a	no
44	LCCOMB_X92_Y32_N24	memory_unit|ram~42358|dataa	Top	n/a	n/a	n/a	no
45	LCCOMB_X91_Y32_N30	mem_or_reg|\GENFOR:18:or_xysel|o_F~0|datad	Top	n/a	n/a	n/a	no
46	FF_X84_Y35_N29	reg_file_32bit|\genera...8:flip_flop|s_Q|asdata	Top	n/a	n/a	n/a	no

Graphical Data Path

The thumbnail view shows a quick visual representation of the extra fitter information related to this path.

The **path connections** appear as heavy black lines. **Netlist nodes** along the path and **routing drivers** along the routing path appear as black dots. The **routing connections** appear as thin black lines. No directional information is drawn for path or routing connections.

Any **bounding boxes** listed in Extra Fitter Information appear as shaded blue rectangles.

Path #1: Setup slack is -11.767 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information