Christopher Scrosati
Advanced Digital

Lab 9 Report: ALU

Problem Summary:

Create and design an arithmetic logic unit that implements previous design files that should be able to store instructions, two numbers to memory, and be able to read those inputs from memory.

Specifications:

- Mode Operation Switch: SW[16]
  - Read Enable: Off
  - Write Enable: On
- Enable: KEY[0]
- Reset:
  - Input: SW[17]
- INST:
  - Input: SW[14-12]
    - Add = 000
    - Subtract = 001
    - Equal = 100
    - Greater than = 101
    - Less than = 110
    - A equal 0 = 111
  - Output:LEDG[2-0]
- A:
  - Input: SW[11-6]
  - Output: HEX7, HEX6
- B:
  - Input: SW[5-0]
  - Output: HEX5, HEX4
- Z:
  - Output: HEX1, HEX0
- Overflow:
  - Output: LEDG[8]
- aNeg:
  - Output: LEDR[16]
- bNeg:
  - Output: LEDR[15]
- zNeg:
  - Output: LEDR[7]
- Should store A, B and I in memory
- Should be able to read A, B and I
- Should have an asynchronous reset, asynchronous de-assert

- Should output A, B, and result on the seven segment displays
- The enable key input will be synchronized on a falling edge.
- The clock speed of the system does not need to be divided.

Design Solution:

To design the system, I needed to follow the flow of data through the system. First, the design would take in inputs for the instruction, number A and Number B, then wait for the data to be written into memory through a synchronized keypress. Once data is written into memory it should be able to be read sequentially by switching to read mode and pressing a the enable key. Once read, the instructions will be displayed via 3 LEDs and the numbers A and B will be displayed on 2 seven segment displays each, with LEDs signifying if A or B are negative. This would also cause the data to go through a combinational case statement for the ALU control. This FSM takes in the instruction set, and based on the value, perform operations using A and B, sending the result to a variable, Z, and displaying on two more seven segment displays with an LED to signify if it is negative. This design is shown in the design below.

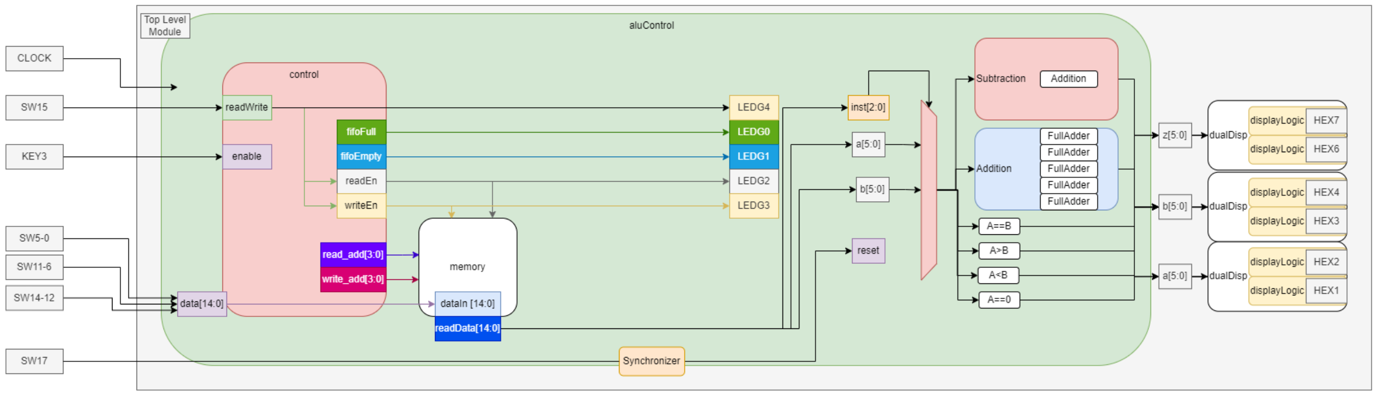


Figure 1: Block Diagram

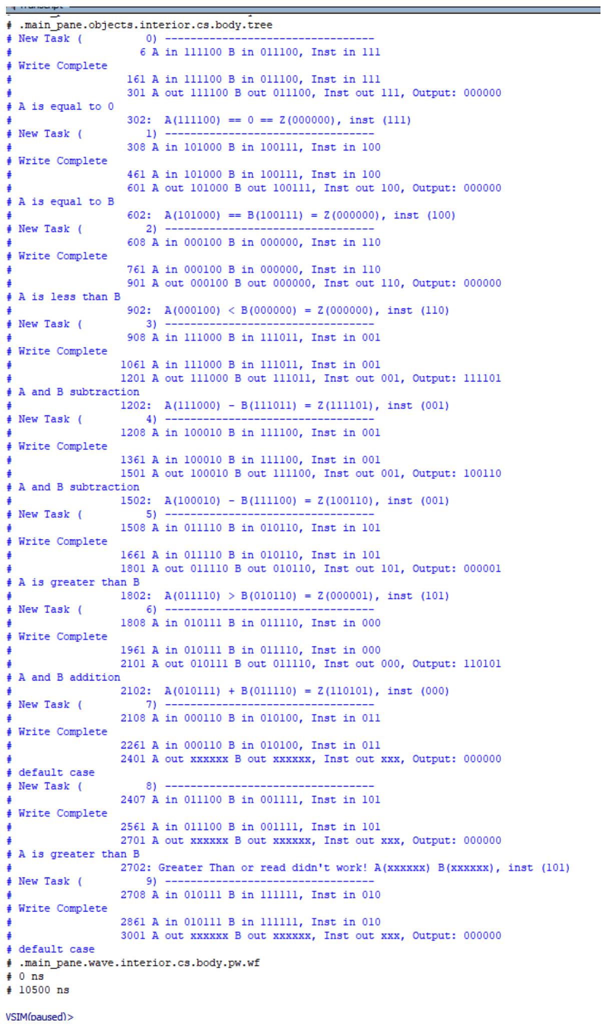
I implemented modules that were previously designed for 2's complement addition, 2's complement subtraction, memory, synchronizers, resets, displaying numbers in decimal, and displaying numbers on the seven-segment display. All modules had been tested previously. The top level included the ALU control module and Dual Display modules. The ALU control module contains the memory, the memory control, subtraction, and addition as submodules.

Testing Approach:

Since every module besides the ALU control module had been tested previously, I only needed to ensure that the case statement was functioning properly, and that the appropriate values were being sent to the display drivers. To do this I implemented the ALU control and display modules in the top level. The test bench randomizes the instruction and values being sent to memory, then utilizes a case statement to verify the values being read from memory into the

ALU. The output is then self-verified in the testbench. TO make my testbench function I just needed to change the internal logic for a, b, and z to output logic.

Results:



```
# .main_pane.objects.interior.cs.body.tree
# New Task (          0) ---------------------------------
#                      6 A in 111100 B in 011100, Inst in 111
# Write Complete
#                    161 A in 111100 B in 011100, Inst in 111
#                    301 A out 111100 B out 011100, Inst out 111, Output: 000000
# A is equal to 0
#                    302:  A(111100) == 0 == Z(000000), inst (111)
# New Task (          1) ---------------------------------
#                    308 A in 101000 B in 100111, Inst in 100
# Write Complete
#                    461 A in 101000 B in 100111, Inst in 100
#                    601 A out 101000 B out 100111, Inst out 100, Output: 000000
# A is equal to B
#                    602:  A(101000) == B(100111) = Z(000000), inst (100)
# New Task (          2) ---------------------------------
#                    608 A in 000100 B in 000000, Inst in 110
# Write Complete
#                    761 A in 000100 B in 000000, Inst in 110
#                    901 A out 000100 B out 000000, Inst out 110, Output: 000000
# A is less than B
#                    902:  A(000100) < B(000000) = Z(000000), inst (110)
# New Task (          3) ---------------------------------
#                    908 A in 111000 B in 111011, Inst in 001
# Write Complete
#                   1061 A in 111000 B in 111011, Inst in 001
#                   1201 A out 111000 B out 111011, Inst out 001, Output: 111101
# A and B subtraction
#                   1202:  A(111000) - B(111011) = Z(111101), inst (001)
# New Task (          4) ---------------------------------
#                   1208 A in 100010 B in 111100, Inst in 001
# Write Complete
#                   1361 A in 100010 B in 111100, Inst in 001
#                   1501 A out 100010 B out 111100, Inst out 001, Output: 100110
# A and B subtraction
#                   1502:  A(100010) - B(111100) = Z(100110), inst (001)
# New Task (          5) ---------------------------------
#                   1508 A in 011110 B in 010110, Inst in 101
# Write Complete
#                   1661 A in 011110 B in 010110, Inst in 101
#                   1801 A out 011110 B out 010110, Inst out 101, Output: 000001
# A is greater than B
#                   1802:  A(011110) > B(010110) = Z(000001), inst (101)
# New Task (          6) ---------------------------------
#                   1808 A in 010111 B in 011110, Inst in 000
# Write Complete
#                   1961 A in 010111 B in 011110, Inst in 000
#                   2101 A out 010111 B out 011110, Inst out 000, Output: 110101
# A and B addition
#                   2102:  A(010111) + B(011110) = Z(110101), inst (000)
# New Task (          7) ---------------------------------
#                   2108 A in 000110 B in 010100, Inst in 011
# Write Complete
#                   2261 A in 000110 B in 010100, Inst in 011
#                   2401 A out xxxxxx B out xxxxxx, Inst out xxx, Output: 000000
# default case
# New Task (          8) ---------------------------------
#                   2407 A in 011100 B in 001111, Inst in 101
# Write Complete
#                   2561 A in 011100 B in 001111, Inst in 101
#                   2701 A out xxxxxx B out xxxxxx, Inst out xxx, Output: 000000
# A is greater than B
#                   2702: Greater Than or read didn't work! A(xxxxxx) B(xxxxxx), inst (101)
# New Task (          9) ---------------------------------
#                   2708 A in 010111 B in 111111, Inst in 010
# Write Complete
#                   2861 A in 010111 B in 111111, Inst in 010
#                   3001 A out xxxxxx B out xxxxxx, Inst out xxx, Output: 000000
# default case
# .main_pane.wave.interior.cs.body.pw.wf
# 0 ns
# 10500 ns

VSIM(paused)>
```

Figure 2: Testbench Transcript

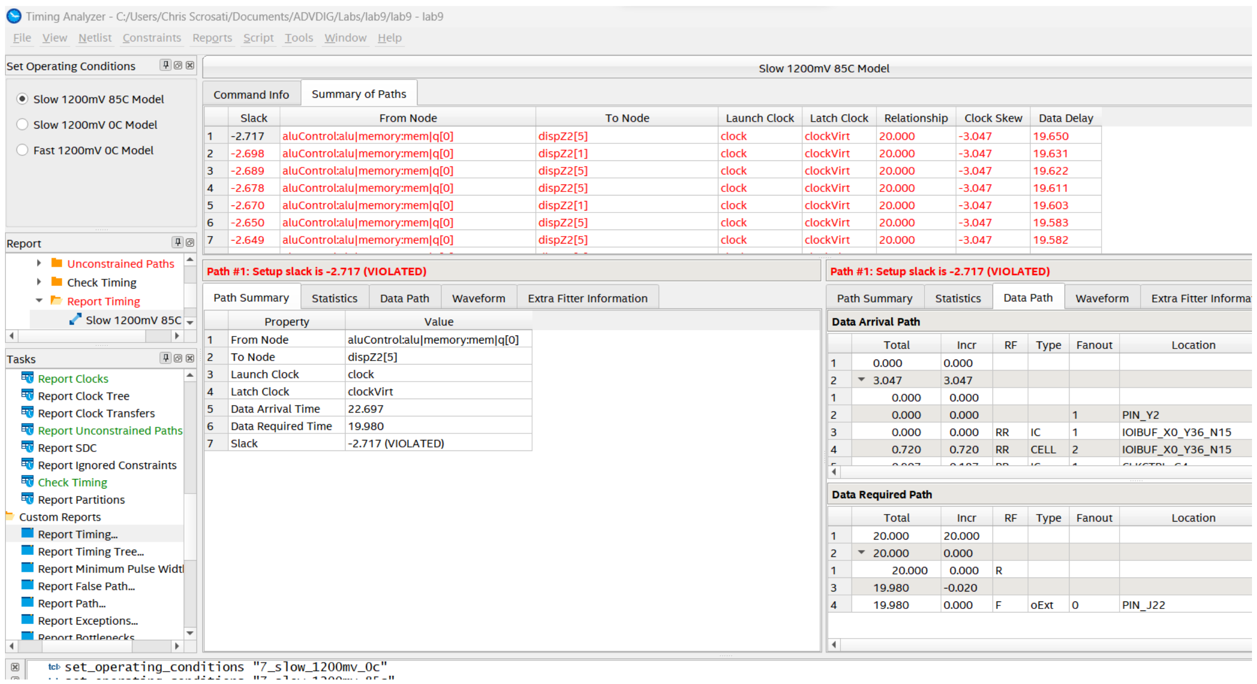
As shown in the figure above, the testbench outputs verification that the ALU instruction worked. Each task writes in values to memory, then reads them out.

After implementing my design on the board, I demonstrated for check off. The one issue my design has is that the comparisons only work with unsigned binary, not signed 2’s complement.

Christopher Scrosati
Advanced Digital

Post-lab Analysis:

I set the clock speed to 50MHz, which is the base speed of the clock on the FPGA. The longest path, as shown in figure 3, is from the memory output q[0] bit to the display Z number 2 [5] bit.



Timing Analyzer - C:/Users/Chris Scrosati/Documents/ADVDIG/Labs/lab9/lab9 - lab9

Set Operating Conditions
- (●) Slow 1200mV 85C Model
- ( ) Slow 1200mV 0C Model
- ( ) Fast 1200mV 0C Model

Report
- Unconstrained Paths
- Check Timing
- Report Timing
  - Slow 1200mV 85C

Tasks
- Report Clocks
- Report Clock Tree
- Report Clock Transfers
- Report Unconstrained Paths
- Report SDC
- Report Ignored Constraints
- Check Timing
- Report Partitions
- Custom Reports
- Report Timing...
- Report Timing Tree...
- Report Minimum Pulse Widt
- Report False Path...
- Report Path...
- Report Exceptions...
- Report Bottlenecks

Slow 1200mV 85C Model

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-2.717	aluControl:alu|memory:mem|q[0]	dispZ2[5]	clock	clockVirt	20.000	-3.047	19.650
2	-2.698	aluControl:alu|memory:mem|q[0]	dispZ2[1]	clock	clockVirt	20.000	-3.047	19.631
3	-2.689	aluControl:alu|memory:mem|q[0]	dispZ2[5]	clock	clockVirt	20.000	-3.047	19.622
4	-2.678	aluControl:alu|memory:mem|q[0]	dispZ2[5]	clock	clockVirt	20.000	-3.047	19.611
5	-2.670	aluControl:alu|memory:mem|q[0]	dispZ2[1]	clock	clockVirt	20.000	-3.047	19.603
6	-2.650	aluControl:alu|memory:mem|q[0]	dispZ2[5]	clock	clockVirt	20.000	-3.047	19.583
7	-2.649	aluControl:alu|memory:mem|q[0]	dispZ2[5]	clock	clockVirt	20.000	-3.047	19.582

Path #1: Setup slack is -2.717 (VIOLATED)

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

	Property	Value
1	From Node	aluControl:alu|memory:mem|q[0]
2	To Node	dispZ2[5]
3	Launch Clock	clock
4	Latch Clock	clockVirt
5	Data Arrival Time	22.697
6	Data Required Time	19.980
7	Slack	-2.717 (VIOLATED)

Path #1: Setup slack is -2.717 (VIOLATED)

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

Data Arrival Path

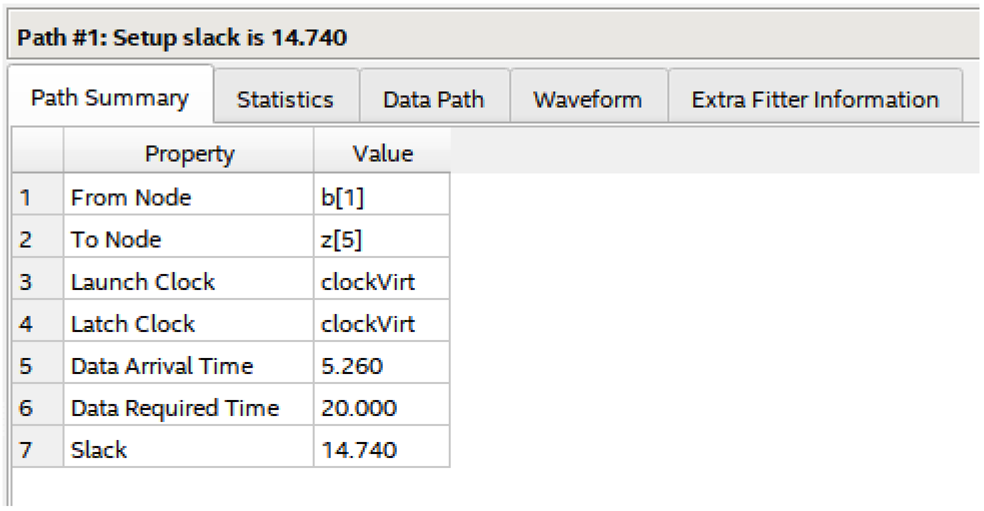
	Total	Incr	RF	Type	Fanout	Location
1	0.000	0.000				
2	3.047	3.047				
1	0.000	0.000				
2	0.000	0.000			1	PIN_Y2
3	0.000	0.000	RR	IC	1	IOIBUF_X0_Y36_N15
4	0.720	0.720	RR	CELL	2	IOIBUF_X0_Y36_N15

Data Required Path

	Total	Incr	RF	Type	Fanout	Location
1	20.000	20.000				
2	20.000	0.000				
1	20.000	0.000	R			
3	19.980	-0.020				
4	19.980	0.000	F	oExt	0	PIN_J22

```
set_operating_conditions "7_slow_1200mv_0c"
```

Figure 3: Timing of overall design



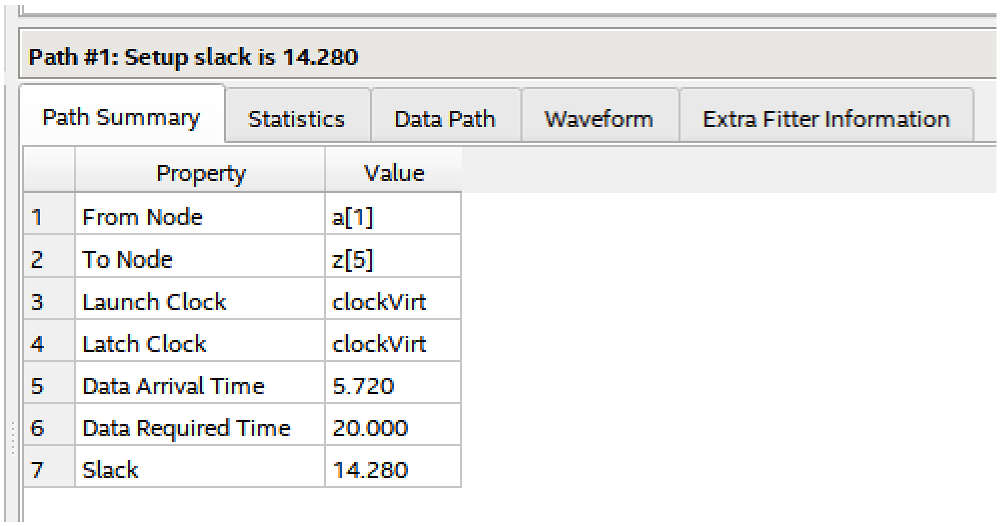
Path #1: Setup slack is 14.740

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

	Property	Value
1	From Node	b[1]
2	To Node	z[5]
3	Launch Clock	clockVirt
4	Latch Clock	clockVirt
5	Data Arrival Time	5.260
6	Data Required Time	20.000
7	Slack	14.740

Figure 4: Propagation Delay of Addition

The propagation delay through the addition module is 5.808ns as shown in Figure 4 under "Data Arrival Time".



Path #1: Setup slack is 14.280

	Property	Value
1	From Node	a[1]
2	To Node	z[5]
3	Launch Clock	clockVirt
4	Latch Clock	clockVirt
5	Data Arrival Time	5.720
6	Data Required Time	20.000
7	Slack	14.280

Figure 5: Propagation Delay of Subtraction

The propagation delay through the subtraction module is 5.720ns Figure 5 under "Data Arrival Time".

The propagation delay through the logic unit itself is 22.717ns as shown in Figure 3 under "Data Arrival Time". 1/22.717ns=44MHz as the max clock rate.

<u>Post-Lab:</u>

My test procedure worked by writing randomized data into memory and then reading that data for use in the ALU. Once the output has been assigned, the testbench checks the output based on a case statement that includes separately calculated outputs with the A and B provided from the read with the output from the ALU. If this is correct it states the instruction worked, otherwise it states that it did not work. Instruction values that fall outside of the defined values in the ALU cause a print denoting it is in the default case.

Addition required 14 logic elements, and 0 registers; The maximum clock rate is 1/5.26ns=190MHz. Subtraction required 16 logic elements and 0 registers; The maximum clock rate is 1/5.72ns=174Mhz. The logic unit required 559 logic elements and 153 registers.

The logic unit was not possible to isolate, so these values are indicative of my top-level module. Subtraction seemed to have the largest space requirements, and this makes sense seeing as it calls the addition module after using combinational logic to invert the bits of B. Subtraction also has the longest setup time. The hold time for both addition and subtraction are the same since there is no flipflop in use.

To design the system, I assumed that there were no delay, latency, or space requirements to meet. I think the addition of a flipflop would greatly reduce the delay created by the immense combinational logic.

Based off the work I did for worksheet 23, I could improve the speed of my longest path in my ALU control by using a flipflop instead of combinational logic that operates on the clock.

Christopher Scrosati
Advanced Digital

All flipflops in my design are necessary, meaning latency can only be increased, not reduced. Most logic in my design is combinational, resulting in a high delay, and a low latency. Adding a flipflop would have the added benefit of increasing my clock frequency.