Table of Contents

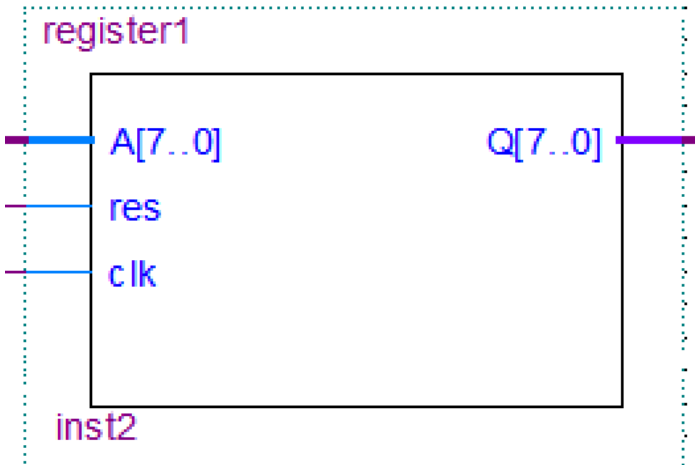
Introduction

This lab combines every component made throughout the course to create three Arithmetic Logic Units (ALU) which will perform given functions. Components two registers to store and output the inputs (register1, register2), a Finite State Machine (fsm) to output student numbers to a seven segment display decoder (sseg_neg) and provide state numbers to be processed through a decoder (dec4to16) and create operational codes for the ALU (opcode). The ALU will perform logical operations based on the given opcode and then output it to three seven segment displays with one being preserved for the sign. Additional components such as input splitters will be used to split inputs (8 bits to two 4 bits) onto dual seven segment displays which have the ability to display two numbers with a sign via single code block (sseg_dual). Finally, a state adder will also be used to display the function being performed in binary on the green leds present on the FPGA board.

Sub-Components

The storage elements register1 and register2 store the inputs A and B. The FSM follows a sequence of states which are outputted to the 4 to 16 decoder and student numbers assigned to each state displayed to seven segment displays.

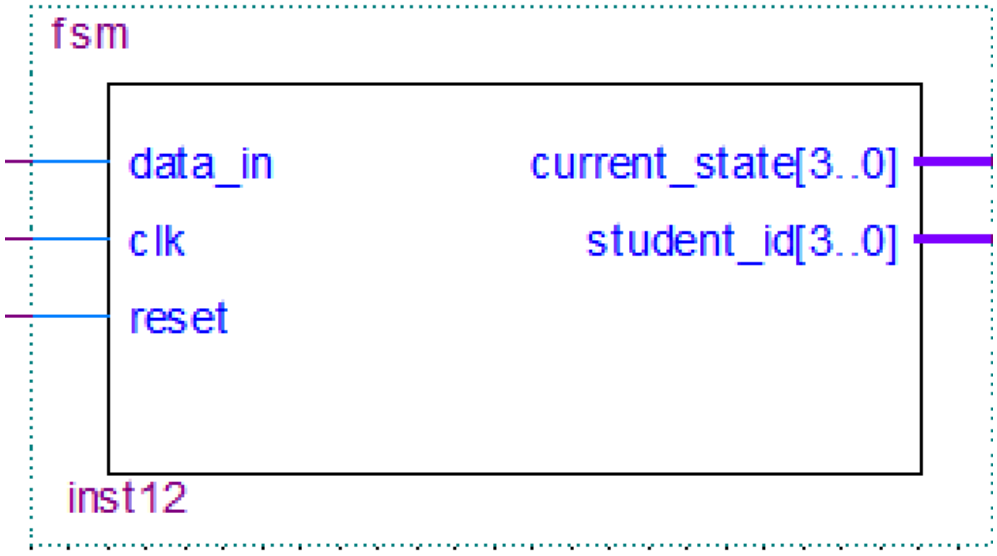
Register1, Register2:



Truth Table:

Input			Output
Clock	Reset	D	Q
0	0	0	0
0	0	1	0
1	0	0	0
1	0	1	1
1	1	1	0

fsm:

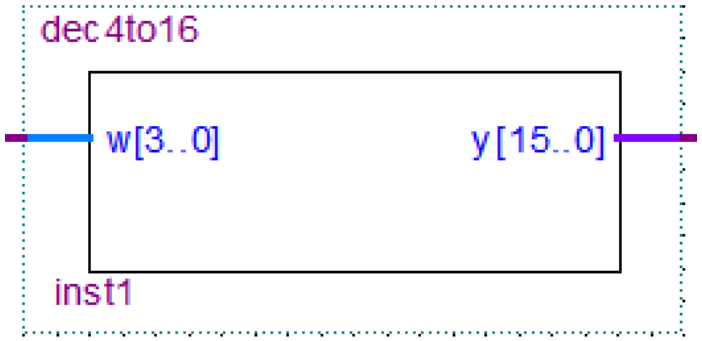


Truth Table:

The table below only considers a clock and data_in of 1, the states remain the same for a clock and data_in of 0

Input		Output	
State	Reset	current_state	student_id
S_0	0	0000	0000
S_1	0	0001	0001
S_2	0	0010	0010
S_3	0	0011	0101
S_4	0	0100	0111
S_5	0	0101	0010
S_6	0	0110	0110
S_7	0	0111	0110
S_n	1	0000	0000

dec4to6:



Truth Table:

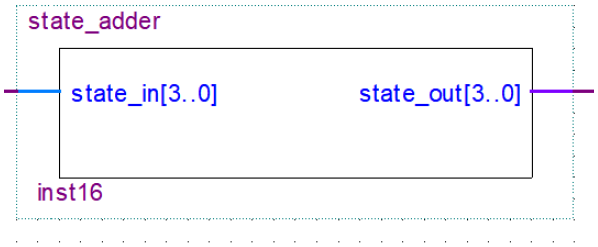
The encoder must always be 1 for an output so it is not included in the block, states beyond S_7 are not used as the only 8 states are present in the fsm, error case included in final row

Input	Output	
State	current_state	opcode
S_0	0000	0000000000000001
S_1	0001	0000000000000010
S_2	0010	0000000000000100
S_3	0011	0000000000001000
S_4	0100	0000000000010000
S_5	0101	0000000000100000
S_6	0110	0000000001000000
S_7	0111	0000000010000000
S_8	1000	0000000100000000
S_9	1001	0000001000000000
S_{10}	1010	0000010000000000
S_{11}	1011	0000100000000000
S_{12}	1100	0001000000000000
S_{13}	1101	0010000000000000
S_{14}	1110	0100000000000000
S_{15}	1111	1000000000000000
S_{15+n}	OTHERS	0000000000000000

Complementary Components

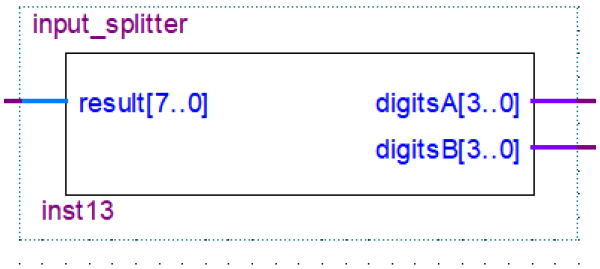
Additional components were added to keep track of the inputs and states with ease. This included an input splitter which split each input in half (4 in total, 2 for A, 2 for B), which was then displayed with a dual-sseg decoder onto 4 displays total. A state adder was used to add 1 to the output states so that the 4 green LEDs it was pinned to would indicate the function being used in binary (0 for off, 1 for on for 4 green LEDS); the states start at 0 as opposed to 1 so the function being performed would not be accurately displayed based on the given function tables. A modified sseg decoder (sseg_mod) was also made for problem set 3, it displays Y when the bcd input is "1111" and N when the input is "0000". A final note is that each block diagram included most of the same components to ensure that the unused displays are blanked out (see ALU_3, empty displays are due to one of the sseg being grounded so that the output can be seen easily). The same was done for the pin names as well so no re-pinning is required for each ALU.

state_adder:



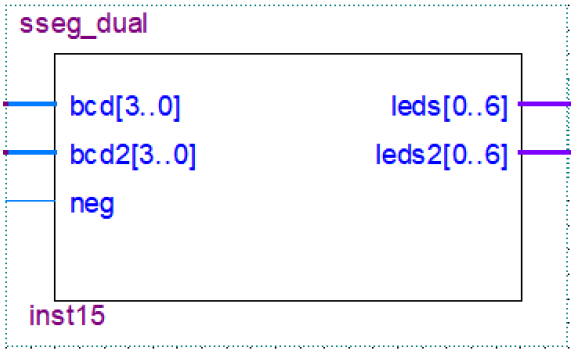
input_splitter:

Input split into two 4 bit numbers to represent the 8 bit input



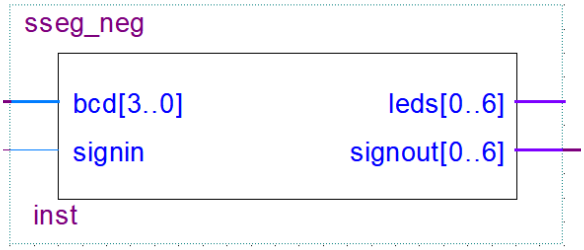
sseg_dual:

Can handle two digit numbers (in hexadecimal) and display on 2 separate ssegs



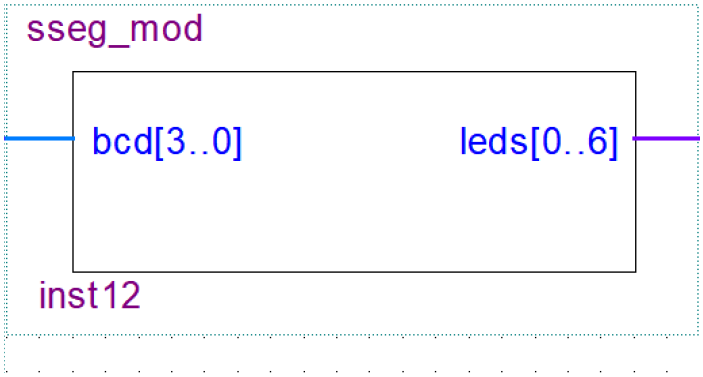
sseg_dual:

Can handle 1 digit numbers (in hexadecimal) with a sign and display on 2 ssegs



sseg_mod:

Displays Y or N based on bcd input of "1111" or "0000" respectively

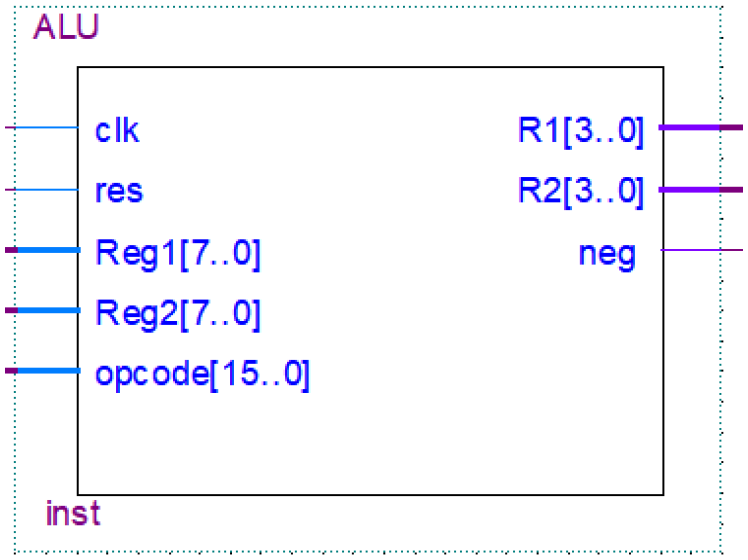


ALU_1 - Problem Set 1

The ALU in this problem set was designed to have 5 inputs and 3 outputs. The following description of the inputs is valid for all the ALUs made for this lab (Problem Set 1, 2, 3). The clock (clk) for the ALUs were initially wired with all the other clocks but remained unused in the code itself (commented out) due to synchronization issues with the opcode, the clock remains in the event it can be used for another purpose. For the overall schematic, the clock signal initiates both the changing inputs and the next opcode from the FSM. The reset (res) of the ALUs are tied to one pin to ensure every component refreshes to zero upon being reset. With a new clock signal, Reg1[7..0] (A) and Reg2[7..0] (B) are 8 bit inputs from the registers, the arithmetic operations are performed on these values. Finally, in addition with the new clock signal, the opcode[15..0] input from the decoder is used to determine which operation the ALU needs to perform. All the ALUs in this lab share these qualities for the ALU aside from the opcode, which differs for Problem Set 3. For the outputs, the ALU will perform operations based on the given opcode and split the 8 bit output into 2, 4 bit outputs as R1[3..0] and R2[3..0]; the sign, neg, will also be outputted (always zero except for subtraction to ensure errors cannot occur with the sign). The outputs are then displayed via sseg decoders. The ALU operations for this problem set are listed in the table below. Note that the subtraction operation subtracts A-B meaning if B is larger, B-A is done and the sign variable is turned on.

ALU:

Problem Set 1 ALU block diagram



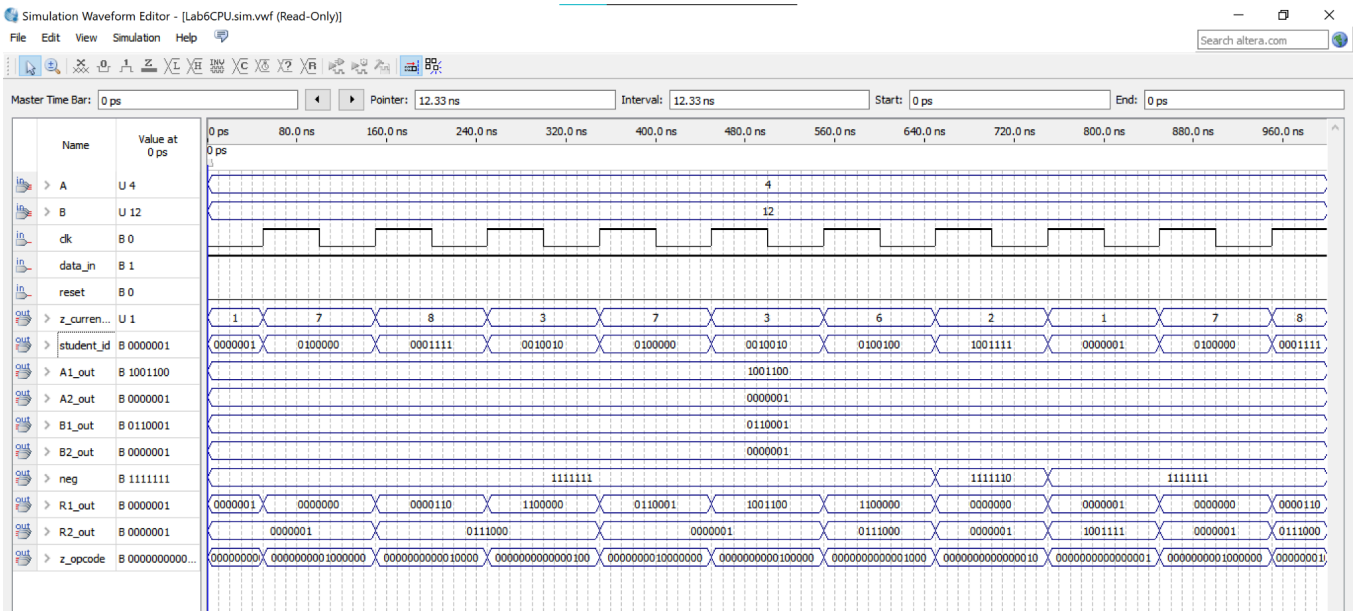
Truth Table:

The bottom-most row represents the error case in the event the microcode does not match any of the operations

Input	Output	
Function #	Microcode	Operation
1	0000000000000001	Sum(A,B)
2	0000000000000010	Diff(A,B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$
>8	OTHERS	0

Waveform:

The figure below is an example of inputs given to the ALU

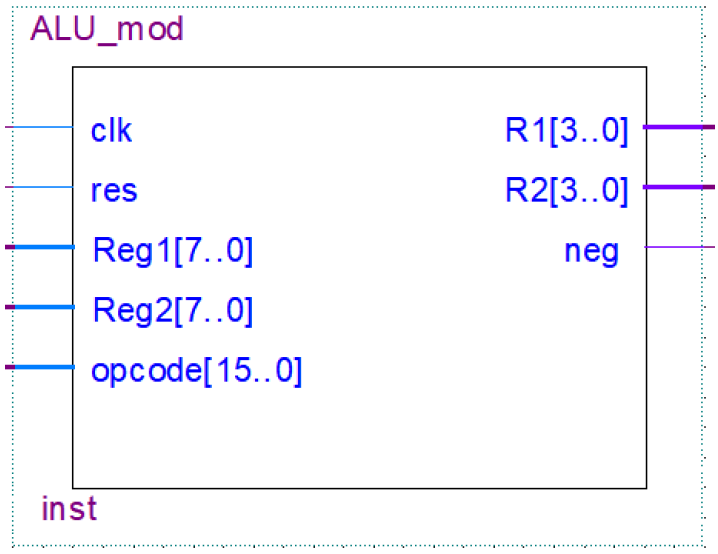


ALU_2 - Problem Set 2

This ALU shares all the input attributes from Problem Set 1 and only differs by the operations performed on the inputs - functionally the inputs behave the same as Problem Set 1, so the description section for ALU_mod (ALU_2) inputs will be skipped. Functionally Problem Set 1 and 2 do not differ aside from the operations so the outputs also split in the same way. Specific operations are done for either each input or both inputs together, the following table outlines the different operations done.

ALU_mod:

Problem Set 2 ALU block diagram



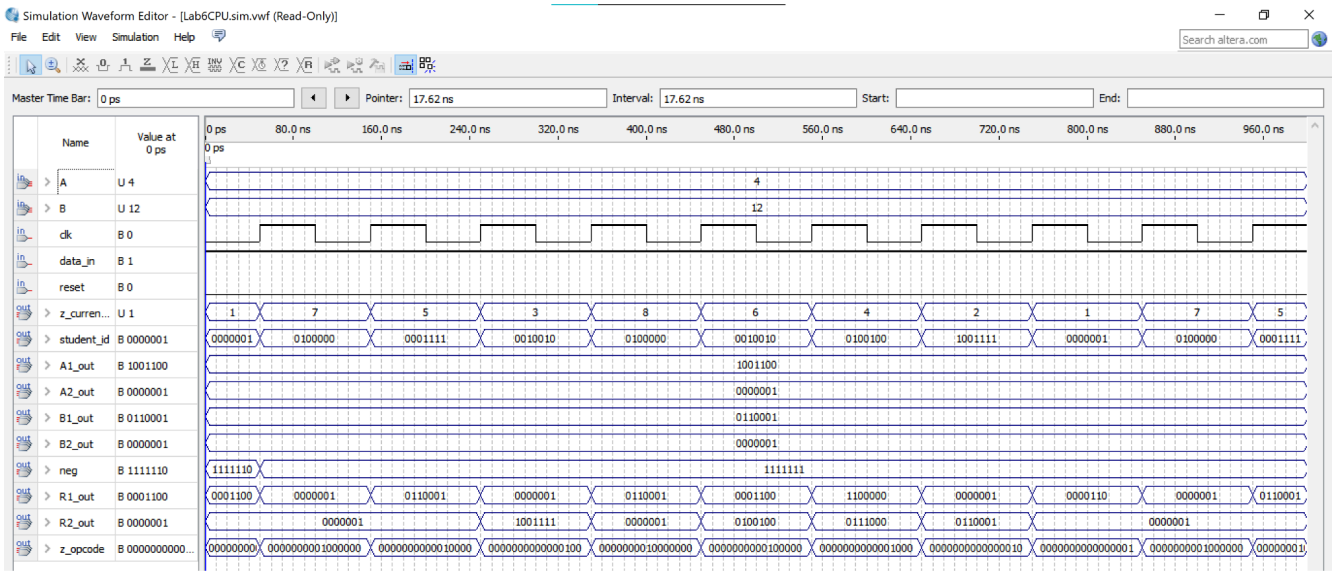
Truth Table:

Note that Reg1=A, Reg2=B

Input	Output	
Function #	Microcode	Operation
1	0000000000000001	Decrement B by 9
2	0000000000000010	Swap upper and lower 4 bits of B
3	0000000000000100	Shift A left by 2 bits, input bit=0 (SHL)
4	0000000000001000	Produce NAND A and B
5	0000000000010000	Find and output the greater value between A and B (Max(A,B))
6	0000000000100000	Invert even bits of B
7	0000000001000000	Produce output of null
8	0000000010000000	Replace upper four bits of B with upper four bits of A
>8	OTHERS	0

Waveform:

The figure below is an example of inputs given to the ALU_mod, note the pin at the bottom is to indicate the opcode given to the ALU

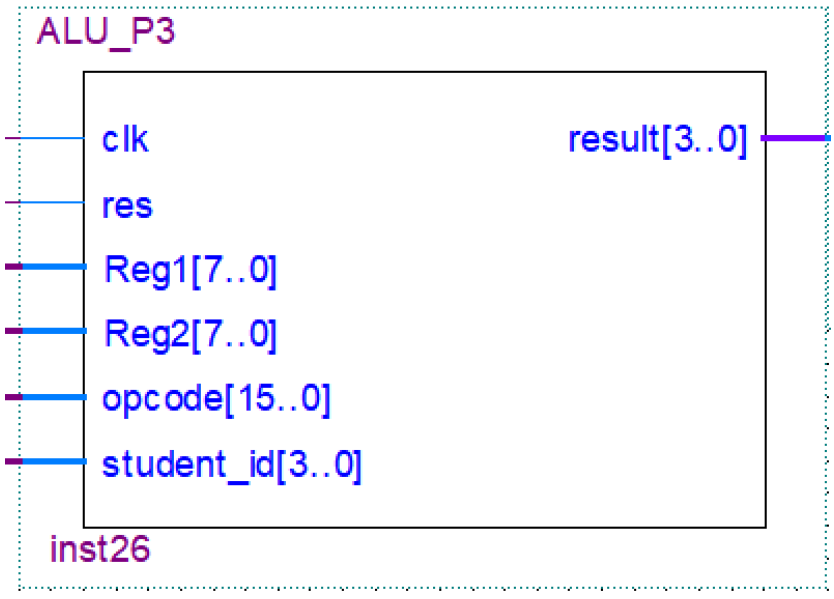


ALU_3 - Problem Set 3

This ALU shares most of the input attributes from Problem Set 1 but differs greatly in how each attribute is used. A new input of student_id[3..0] is added to be used for comparison with the inputs from Reg1[7..0]. Reg1 is split into two 4 bit numbers which are then compared to the student number given from the FSM. If any of the two 4 bit numbers are less than the student number, the output is "Y" when decoded by the sseg, otherwise the "N" is displayed. The output result[3..0] submits "1111" to the sseg decoder for "Y" and "0000" for "N". The modified sseg decoder (sseg_mod). Functionally, the opcode is not used at all due to this but is kept in the block diagram to remain consistent with the previous ALUs with regards to pinning so the opcode input can essentially be ignored. To summarize, the only difference inputs with ALU_P3 and the previous ALUs are the student_id[3..0] input receiving the student number, the opcode not being used due to no involvement in the operations, and output result being 4 bits.

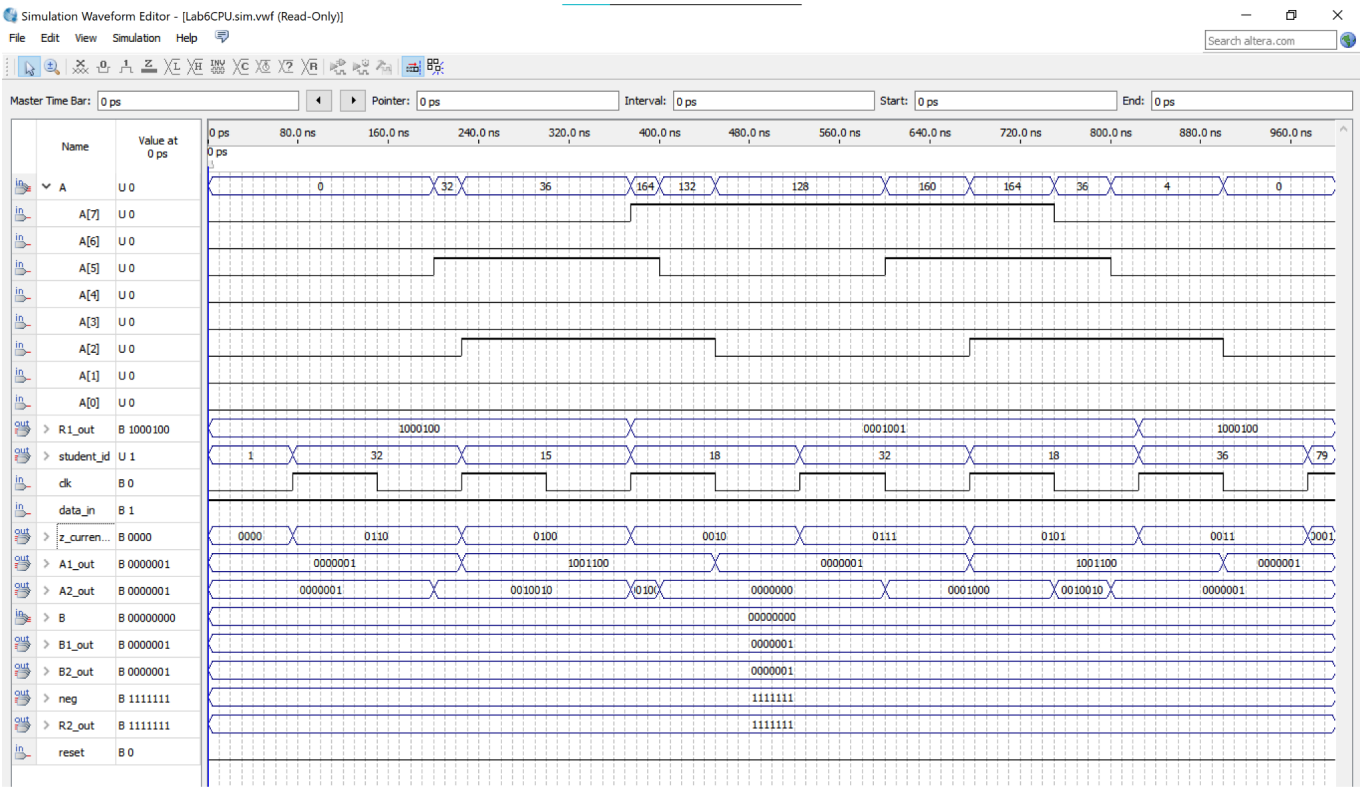
ALU_P3:

Problem Set 3 ALU block diagram



Waveform:

The figure below is an example of inputs given to the ALU_P3, note that 1000100 is Y and 0001001 is N, the stuff below z_current_state are unused



Conclusion

In conclusion, the combination of the registers, ALU, and decoder expose the innumerable amount of possible operations which can be done. This combination can be implemented into various block diagrams to allow for specific user-chosen operations. For example, ALU allows for simple arithmetic operations such as addition, subtraction, and various boolean algebra operations to specific operations done on specific inputs (ALU_mod) such as replacing the upper 4 bits of one input with the upper 4 bits of another input. ALU_P3 showed that the ALU can also be used to output simple boolean expressions for a single input as seen when the output was either Y or N on the sseg displays depending on what student number was sent to the ALU. The implementation of the ALU in this lab was shown in 3 different ways to prove how versatile programming an ALU using VHDL is for applications on an FPGA board and opened up the possibilities and it being extended far beyond just FPGA boards.