COE 328 Final Project: 8-Bit CPU

Justin Dang (501226005)

Department of Electrical, Computer, and Biomedical Engineering, Toronto Metropolitan University

COE 328: Digital Systems, Section 18

Dr. Arghavan Asad and TA Mohammad Hossein Saliminabi

December 11th, 2024

Contents

Introduction

The purpose of this lab was to create an 8-bit central processing unit (CPU) by combining various architectures such as the register, finite state machine (FSM), 3-to-8 decoder, and arithmetic logic unit (ALU), to execute various basic operations. Two registers were used to each store an 8-bit input value in which the operations are to be performed on. An FSM was used to cycle through eight 3-bit states corresponding to the specific student ID used. This 3-bit student ID was then displayed onto a seven-segment display on the Altera board. A 3-to-8 decoder was used to convert the eight states of the FSM, to an 8-bit opcode. This and the two registers were fed into an ALU to perform a set of operations according to the opcode. A total of three different ALUs were created to perform three different sets of operations. The resulting 8-bit value produced from these ALUs was then split into two 4-bit signed numbers and displayed onto four seven-segment displays on an Altera board, two representing the absolute values and two representing the signs. By creating an 8-bit CPU with various digital components, a stronger understanding of computer architecture was developed.

Components

Register

A register is a location within a computer's processor used to store data temporarily. It is commonly made of multiple D flip-flops, each storing a single bit of data. When multiple D flip-flops are combined, a register is created which stores multiple bits of data; in this case, 8 bits. In D flip flops, the input gets copied to the output on the next clock signal. When the reset input is activated, the value of the register will be set to zero.

Table 1

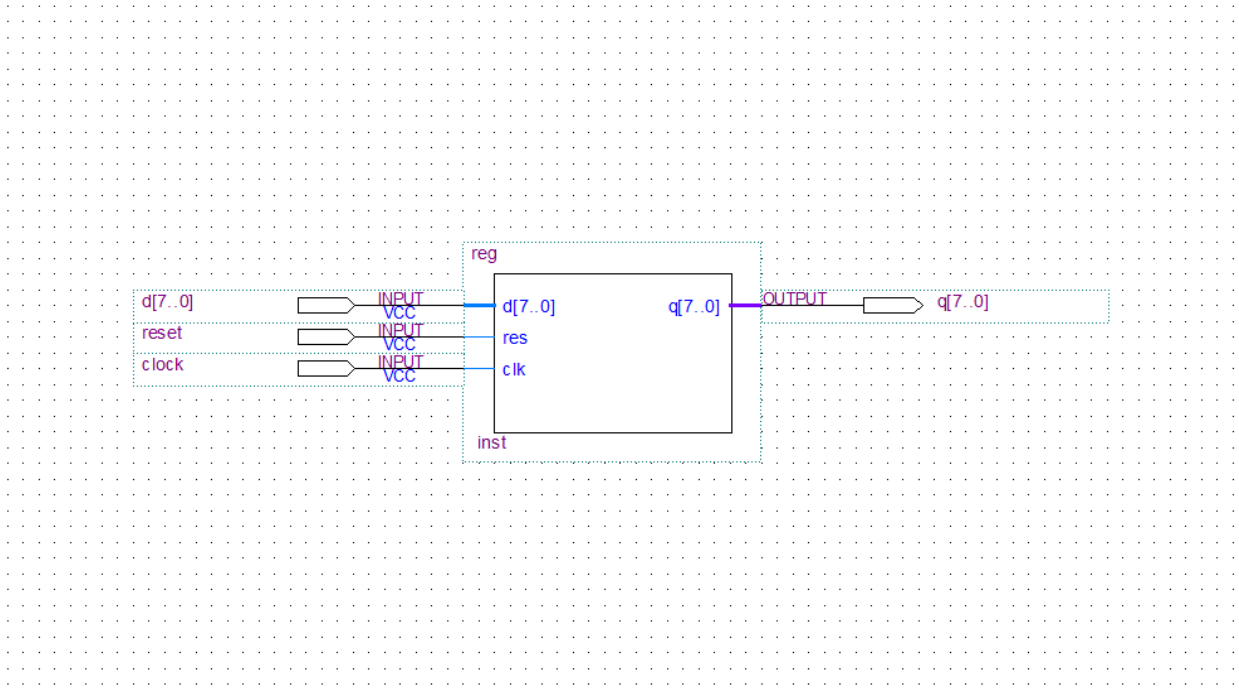
Register Truth Table

Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Q_7^+	Q_6^+	Q_5^+	Q_4^+	Q_3^+	Q_2^+	Q_1^+	Q_0^+
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Note. Truth table for the register.

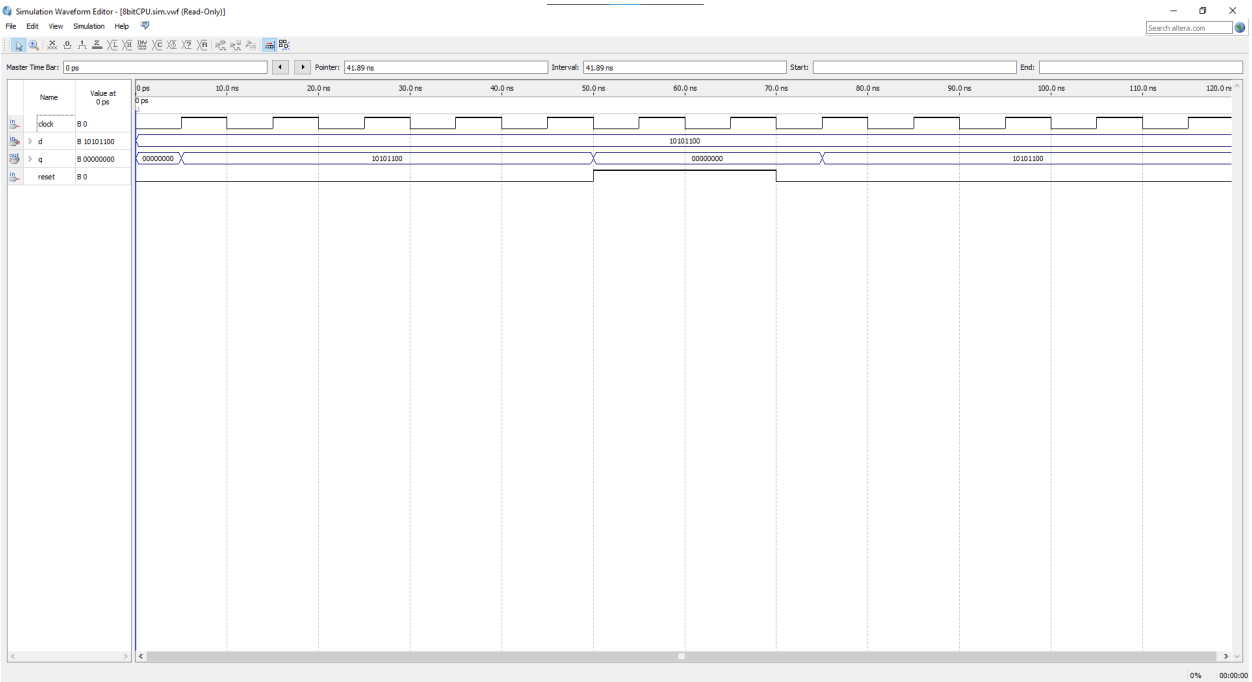
Figure 1

Register Block Diagram



Note. Block diagram for the register.

Figure 2



Note. Waveform showcasing the functionality of the register shown in Figure 1.

Finite State Machine (FSM)

A finite state machine is a computational model used to design computer pograms and sequential circuits. An FSM cycles sequentially through a finite number of states; in this case eight states which fulfills the numbers 0 to 7 used in the specific student ID. When the data in input is set to 1, the FSM moves to the next state on the following clock signal, and that state becomes the new present state. When it is set to 0, it stays at the same state on the following clock signal. When the reset input is activated, the value of the current present state will be set to zero.

Table 2

FSM Truth Table

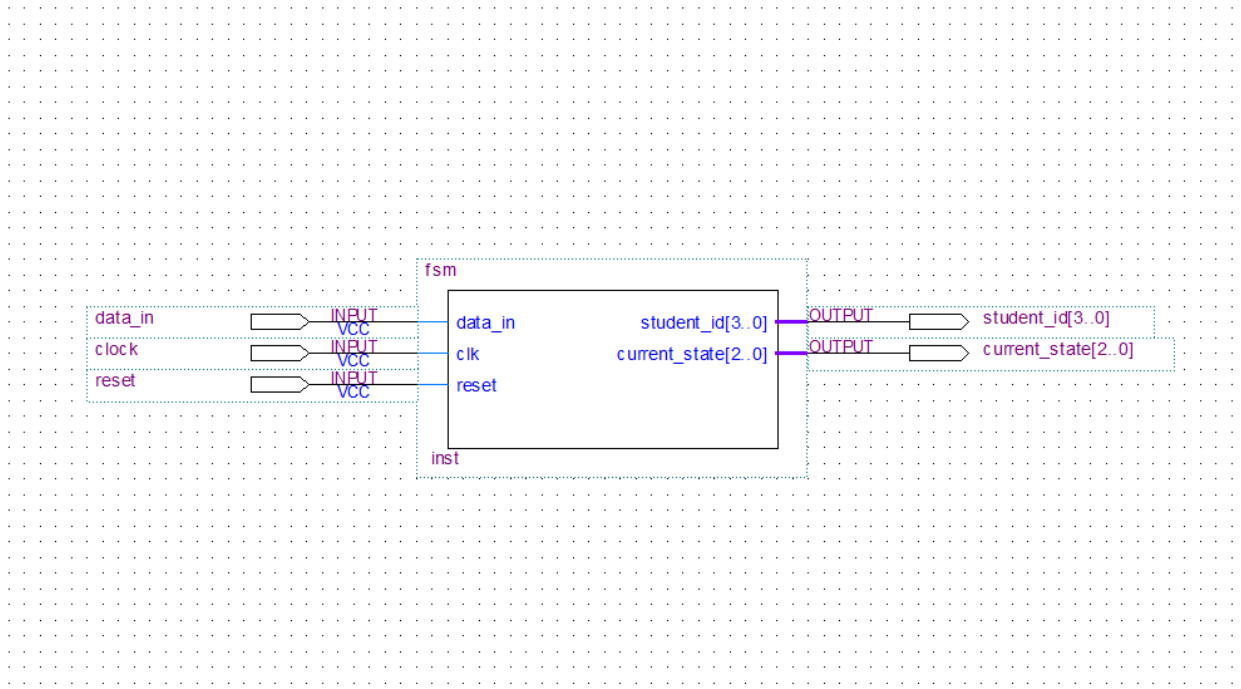
Present State	Next State

			$w = 0$			$w = 1$		
y_2	y_1	y_0	Y_2	Y_1	Y_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	1	0	0
1	0	0	1	0	0	1	0	1
1	0	1	1	0	1	1	1	0
1	1	0	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0

Note. Truth table for the FSM.

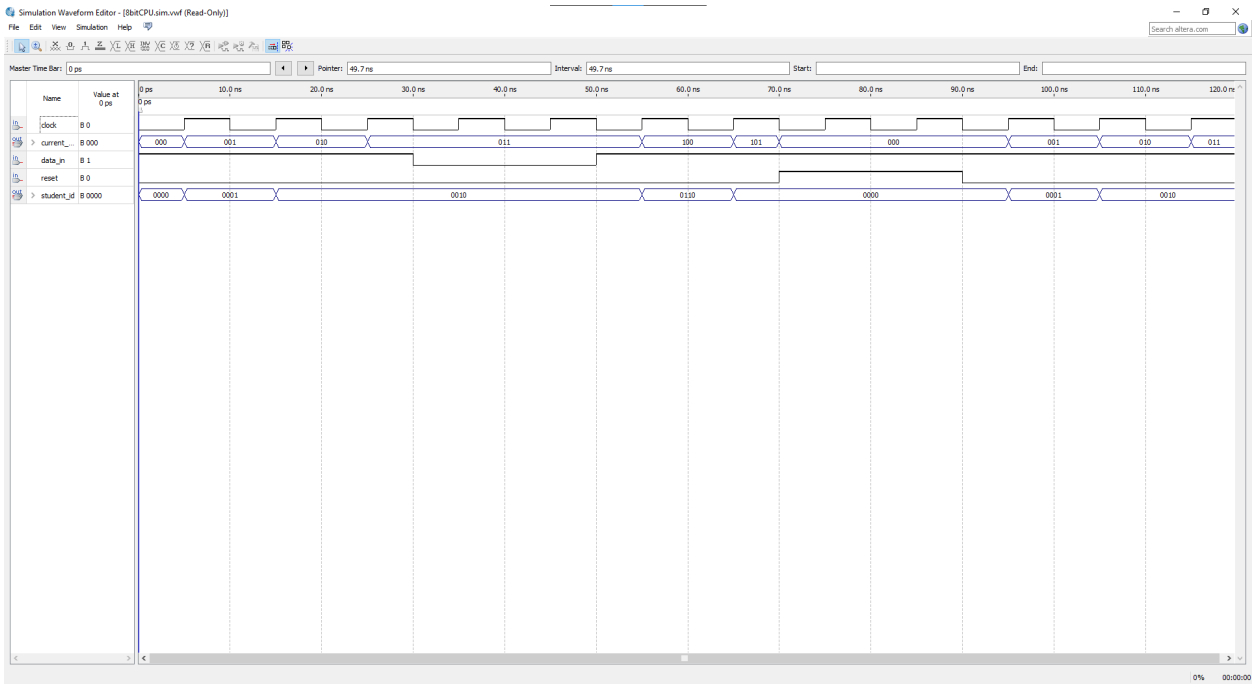
Figure 3

FSM Block Diagram



Note. Block diagram for the FSM.

Figure 4



Note. Waveform showcasing the functionality of the FSM shown in Figure 3.

3-to-8 Decoder

A 3-to-8 decoder is a digital circuit that takes a 3-bit input and converts it into an 8-bit output, each with one unique bit set to 1 while all others remain 0. Each of these values are hotcoded to each of the eight operations the ALU performs. When the enable input is set to 1, the decoder is activated on the following clock signal. When it is set to 0, the decoder is deactivated on the following clock signal.

Table 3

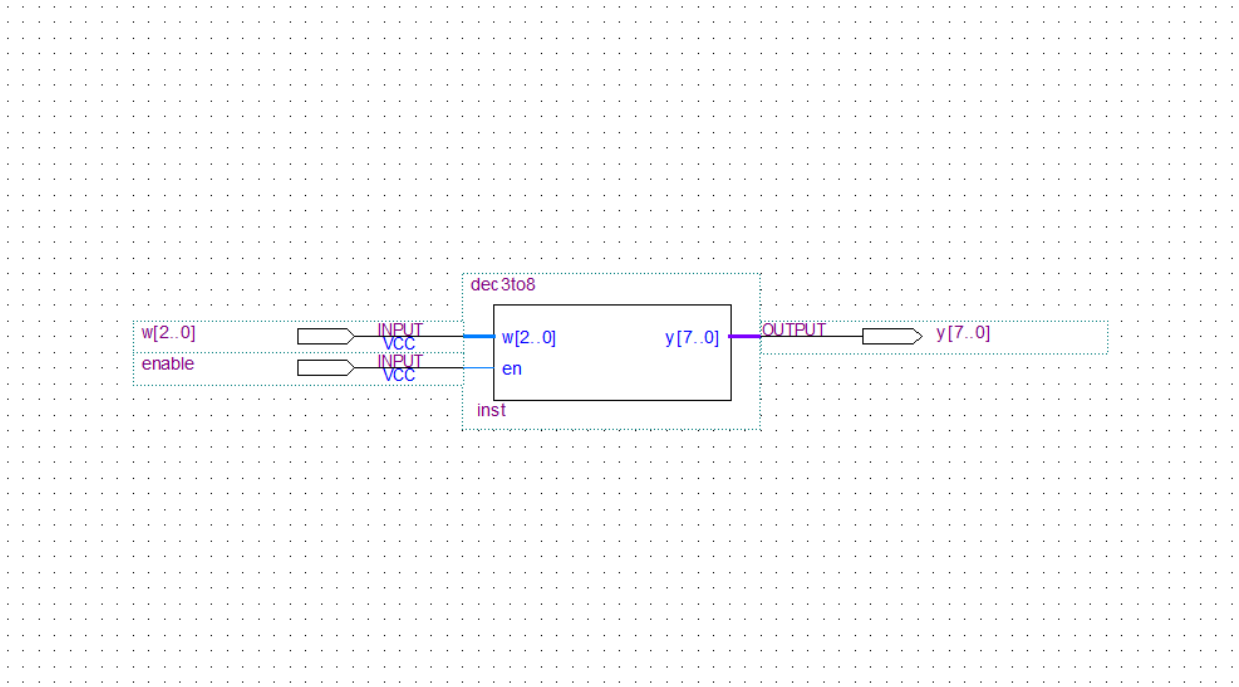
3-to-8 Decoder Truth Table

En	w_2	w_1	w_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1
0	x	x	x	0	0	0	0	0	0	0	0

Note. Truth table for the 3-to-8 decoder.

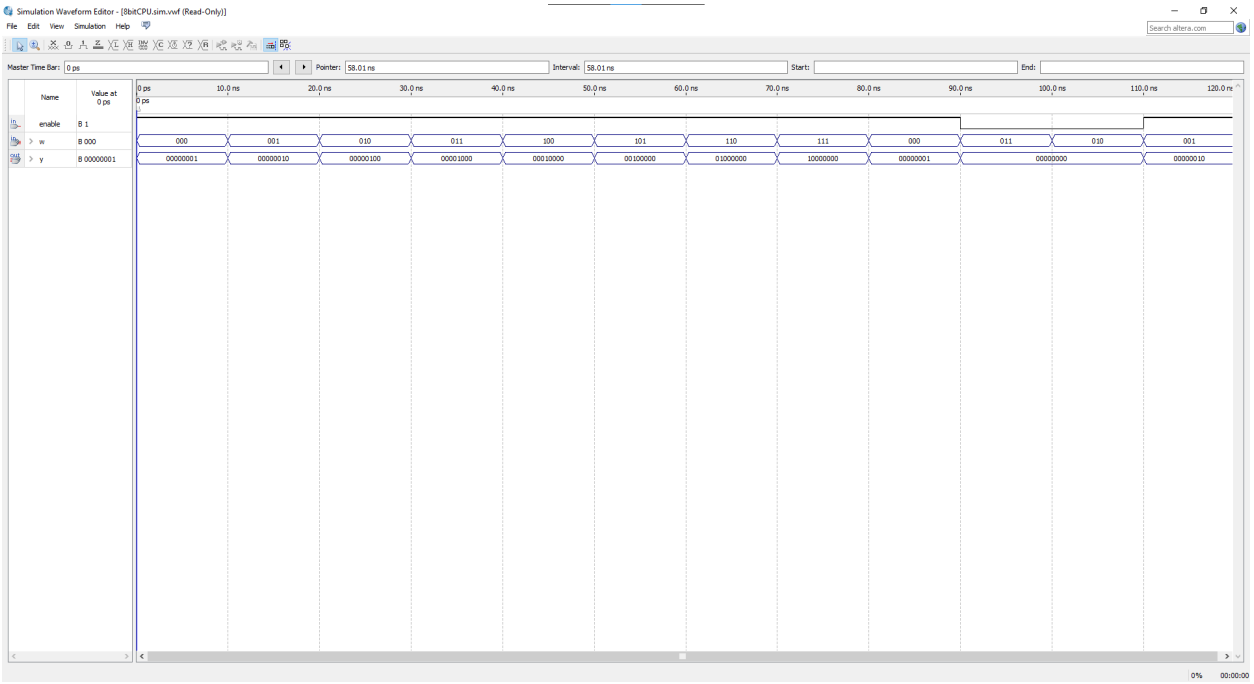
Figure 5

3-to-8 Decoder Block Diagram



Note. Block diagram for the 3-to-8 decoder.

Figure 6



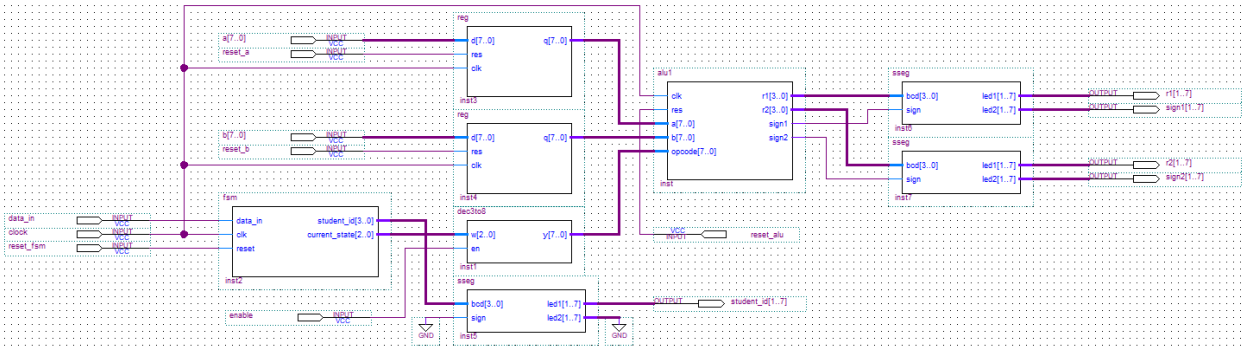
Note. Waveform showcasing the functionality of the 3-to-8 decoder shown in Figure 5.

Arithmetic Logic Unit (ALU) for Problem Set 1

The first ALU performs the operations shown in Table 4 on the two 8-bit values fed from each register. A total of eight operations are executed corresponding to the eight opcodes fed from the 3-to-8 decoder. The resulting 8-bit value from the operation then splits into two 4-bit signed numbers and each is fed into a BCD-to-7-segment decoder in order to display the two numbers and their respective signs onto an Altera board.

Figure 7

CPU Block Diagram for Problem Set 1



Note. Block diagram for the CPU using an ALU for problem set 1.

The first ALU has five inputs (clock, reset, first 8-bit number, second 8-bit number, opcode) and four outputs (first four bits of the 8-bit result, last four bits of the 8-bit result, sign of the first four bits, sign of the last four bits). When the clock input, denoted as ‘clk’ in Figure 7, switches from low to high, it tells the ALU to execute the current instance of the opcode, which begins at 00000001. For every clock signal, the opcode will increment to the next. For example: 00000001 → 00000010 → 00000100. This continues until all eight opcodes are run through, then repeats. This opcode input, denoted as ‘opcode[7..0]’, comes from the 3-to-8 decoder and is based on the current state of the FSM. Its purpose is to carry out the operations on the two 8-bit inputs coming from the registers, denoted as ‘a[7..0]’ and ‘b[7..0]’, abiding by the clock signal as previously mentioned. Adding on to the example, the corresponding first three clock signals will thus execute: the sum of the two inputs, the difference of the two inputs, the first input inverted, and so on for every following opcode. This only works when the reset input, denoted as ‘res’, is set to an active low. If it is set to an active high, the ALU will be reset to a default value of 00000000. The ALU itself splits the resulting 8-bit value from the operation performed into two 4-bit signed values. The first four bits, last four bits and their respective signs, denoted as ‘r1[3..0]’, ‘r2[3..0]’, ‘sign1’, and ‘sign2’, are then fed into two BCD-to-7-segment decoders

which display the results of the two numbers and their signs onto four different seven segment displays.

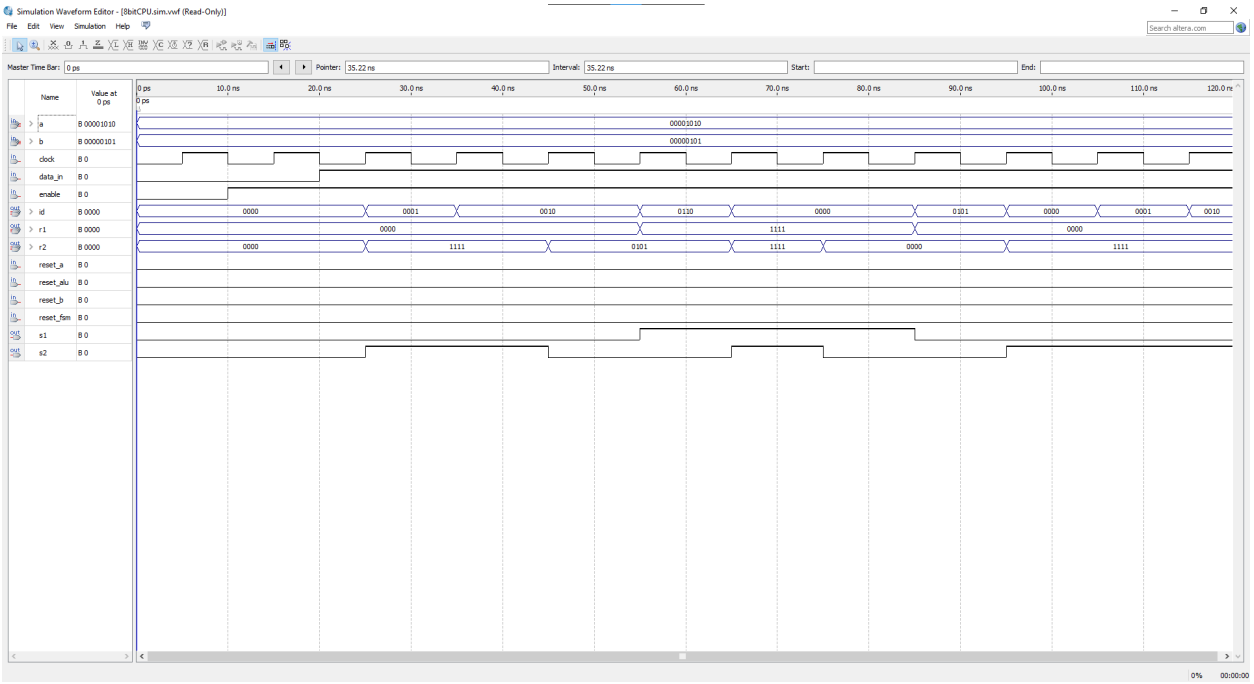
Table 4

Microcodes for Problem Set 1

Function #	Opcode	Function
1	00000001	$sum(a,\ b)$
2	00000010	$dif(a,\ b)$
3	00000100	$\overline{a}$
4	00001000	$\overline{a \cdot b}$
5	00010000	$\overline{a + b}$
6	00100000	$a \cdot b$
7	01000000	$a \oplus b$
8	10000000	$a + b$

Note. Microcodes generated by the 3-to-8 decoder for the ALU used for problem set 1.

Figure 8

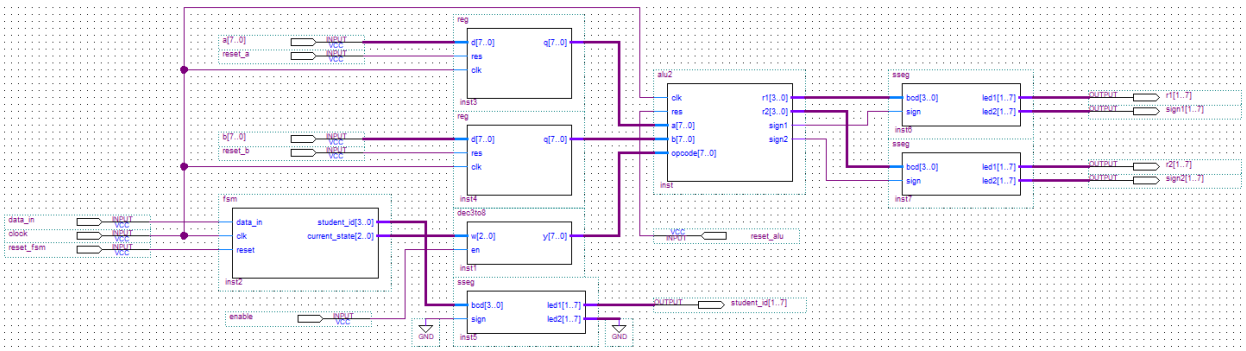


Note. Waveform for problem 1 using student ID = 5(01226005), register$_1$ = 10, and register$_2$ = 5. Seven segment outputs were removed for clarity. The student ID begins at the second clock signal. The operations begin at the third clock signal.

Arithmetic Logic Unit (ALU) for Problem Set 2

The second ALU performs the operations shown in Table 5 on the two 8-bit values fed from each register. A total of eight operations are executed corresponding to the eight opcodes fed from the 3-to-8 decoder. The resulting 8-bit value from the operation then splits into two 4-bit signed numbers and each is fed into a BCD-to-7-segment decoder in order to display the two numbers and their respective signs onto an Altera board.

Figure 9



Note. Block diagram for the CPU using an ALU for problem set 2.

The second ALU has the same five inputs and outputs as the first ALU, and its functionality is exactly the same. The only difference is that the opcodes carry out a different set of operations. Continuing from the example of the first three opcodes (00000001 → 00000010 → 00000100), the corresponding first three clock signals for the second ALU will rather: increment the first input by 2, shift the second input to the right by two with an input bit of 0, shift the first input to the right by four with an input bit of 1, and so on for every following opcode. The result of course gets split into two 4-bit signed values and are fed to the BCD-to-7-segment decoders to be displayed on the seven segment displays on the Altera Board, just like for the first ALU.

Table 5

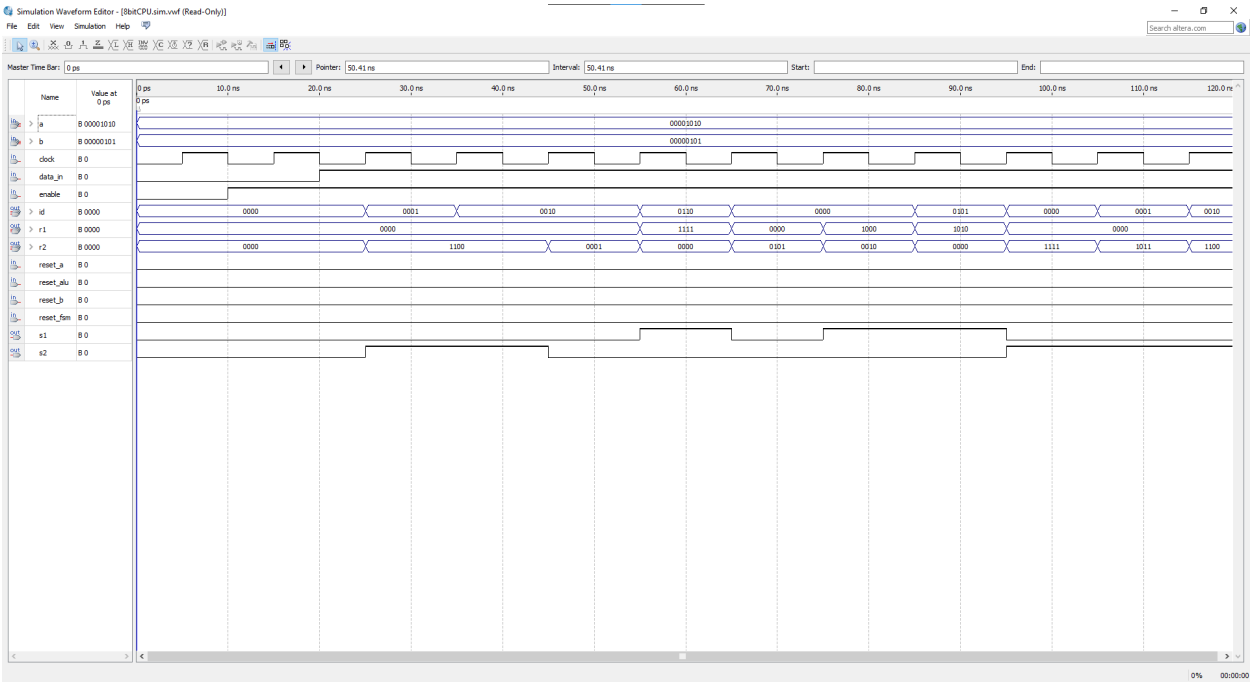
Microcodes for Problem Set 2

Function #	Opcode	Function
1	00000001	Increment *a* by 2
2	00000010	Shift *b* to the right by two bits, input bit = 0

3	00000100	Shift a to the right by four bits, input bit = 1
4	00001000	Find the smaller value of a and b, and produce the results
5	00010000	Rotate a to the right by two bits
6	00100000	Invert the bit-significance order of b
7	01000000	Produce the result of XORing a and b
8	10000000	Produce the summation of a and b, then decrease it by 4

Note. Microcodes generated by the 3-to-8 decoder for the ALU used for problem set 2.

Figure 10

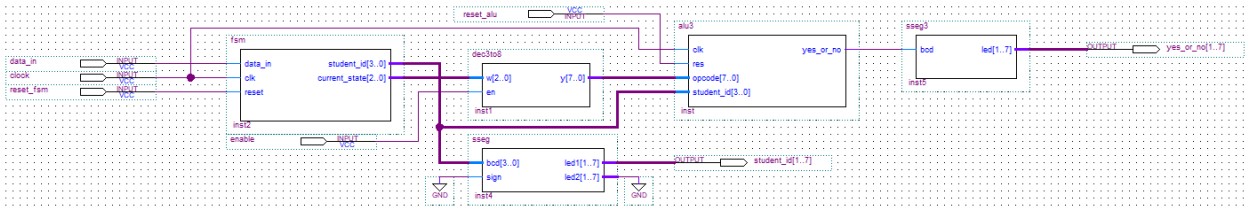


Note. Waveform for problem 2 using student ID = 5(01226005), register$_1$ = 10, and register$_2$ = 5. Seven segment outputs were removed for clarity. The student ID begins at the second clock signal. The operations begin at the third clock signal.

Arithmetic Logic Unit (ALU) for Problem Set 3

The third ALU performs the operations shown in Table 6 on the last eight digits of the student ID, one at a time. Since each opcode fed from the 3-to-8 decoder carries the same operation, a total of one operation is executed; that being, to display ‘y’ if the student ID digit is odd and ‘n’ otherwise. The resulting 8-bit value from the operation then splits into two 4-bit signed numbers and each is fed into a BCD-to-7-segment decoder in order to display the two numbers and their respective signs onto an Altera board.

Figure 11



Note. Block diagram for the CPU using an ALU for problem set 3.

Unlike the first and second ALUs, this ALU has four inputs and (clock, reset, opcode, student ID) and one output (yes/no). The clock, opcode, and reset inputs of the third ALU work exactly the same as those of the first and second ALUs. However, they work in tandem with the new student ID input, denoted as ‘student_id[3..0]’, which is the main difference between the third ALU, and the first and second ALUs. The third ALU instead, has the student ID fed from the FSM and thus, it is controlled by the current state of the FSM. Continuing from the example of the first three opcodes (00000001 → 00000010 → 00000100), the corresponding first three clock signals for the second ALU will rather produce a 1-bit yes (1) or no (0) output, denoted as ‘yes_or_no’, for: the first digit of the student ID, the second digit of the student ID, the third digit of the student ID, and so on for every following opcode. This is then fed into a single BCD-to-7-segment decoder in order to display ‘y’ or ‘n’ onto a seven segment display on the Altera board. It is important to note that the yes or no outputs are delayed by one clock signal. This is due to the nature of FSMs as the next state is based on the current state. So although the current state obtains a yes or no result at that instance, it is displayed at the subsequent instance.

Table 6

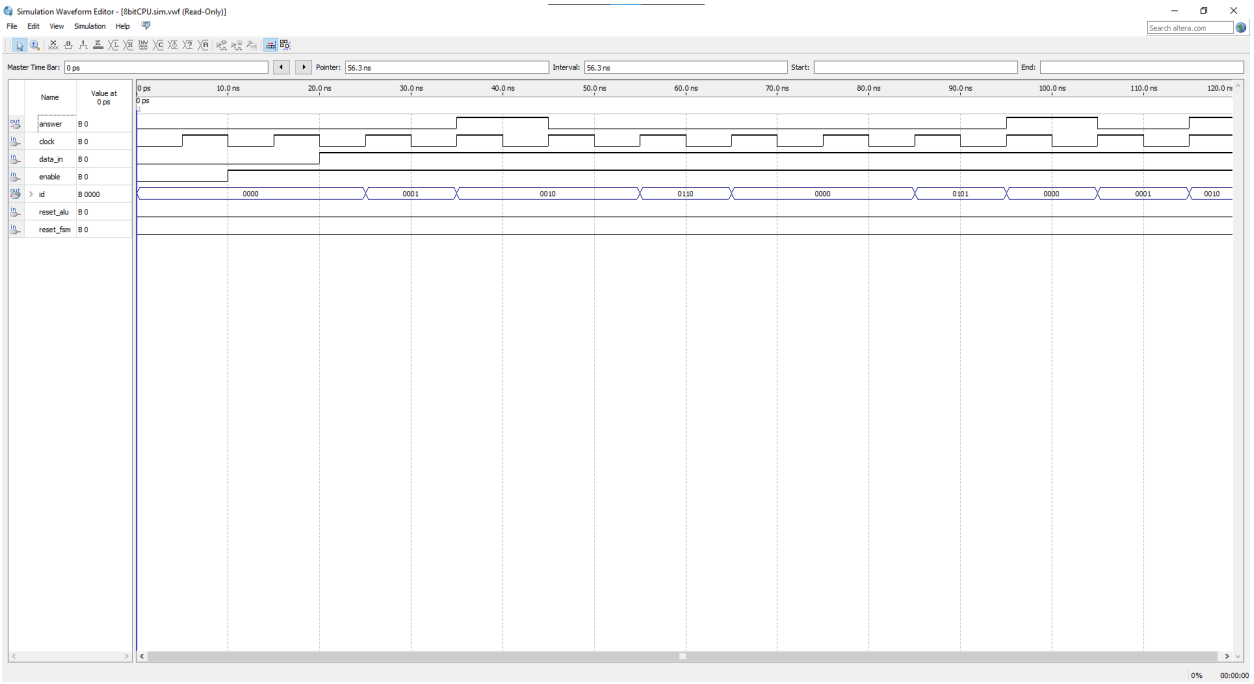
Microcodes for Problem Set 3

Function #	Opcode	Function
1	00000001	Display ‘y’ if the student ID

		digit is odd and ‘n’ otherwise
2	00000010	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
3	00000100	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
4	00001000	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
5	00010000	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
6	00100000	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
7	01000000	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise
8	10000000	Display ‘y’ if the student ID digit is odd and ‘n’ otherwise

Note. Microcodes generated by the 3-to-8 decoder for the ALU used for problem set 3.

Figure 12



Note. Waveform for problem 3 using student ID = 5(01226005). Seven segment outputs were removed for clarity. The student ID begins at the second clock signal. The operations begin at the third clock signal.

Conclusion

In conclusion, this lab successfully demonstrated the creation and operation of an 8-bit CPU by integrating key digital components such as the register, finite state machine (FSM), 3-to-8 decoder and ALU. The operations for all three ALUs outputted the correct result as shown in Figures 8, 10, and 12. This was further checked during the lab session, in which the student ID digit and numbers displayed on the seven segment displays for the first and second ALUs at each clock signal, corresponded with the signed 4-bit binary representations of the waveform outputs. The third ALU also performed as expected, with the yes or no result displaying on the seven segment displays on the subsequent clock signal, matching the waveform. Overall, this lab was a success, as it helped students develop a basic understanding of computer architecture, laying a strong foundation for larger future projects in digital systems design.

References

Brown, S. D., & Vranesic, Z. G. (2009). *Fundamentals of digital logic with VHDL Design*. McGraw-Hill.

Toronto Metropolitan University. (n.d.). Lab 6 - Design of a Simple Central Processing Unit. Toronto.