ECE 2504: Introduction to Computer Engineering, SPRING 2015
Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board

Student Name: Bowei Zhao____

Honor Code Pledge: I have neither given nor received unauthorized assistance on this assignment.

Bowei Zhao ____

Grading: The design project will be graded on a 100 point basis, as shown below:

Manner of Presentation (30 points)

________ Completed Cover Sheet (5 points)

________ Organization: Clear, concise presentation of content; Use of appropriate, well-organized sections (15 points)

________ Mechanics: Spelling and Grammar (10 points)

Technical Merit (70 points)

________ Design Procedure Discussion (5 points)

________ Implementation Discussion: Choice of methods for implementing operations, opcode binary assignments, and values displayed on LEDs. (20 points)

________ Supporting Figures: Table of opcode assignments for each operation, Truth Tables, Karnaugh maps, Circuit Diagrams, etc. (15 points)

________ Simulation Results (15 points)

________ Validation Sheet (15 points)

________ **Project Grade**

Abstract

This project is to design and implement an Arithmetic Logic Unit based on the specification files we are given on the DE0 Nano Board. Being given a set of specified mathematical based operations, input values, output operation commands, and status bits, the project seeks to generate an output at a certain switch value that will produce a result from the Read-Only-Memory (ROM). The program is to be implemented using hexadecimal number representations in the ROM for the values of numbers A and B that they will give and specify. These values which are inputted as 8bit representations but through the mux reduced to half to take up the 4bits on the LED output, will be utilized later in arithmetic functions. Set outputs and inputs are thus able to be reproduced through an attachment to a wire that allows it a value on the switch of the multiplexer. Therefore by changing values of the switch to that of the set value of the 'port value', in the right location, you can output/simulate a LED of that port value, result, or status value for visual purpose.

Purpose

The purpose of this project was to layout, design, implement, and craft a field programmable gate array(FPGA) capable of logical arithmetic operation given a 5bit input from a ROM file. These values stored in the ROM will, through the 16 to 1 multiplexer, be 'reduced' to an 8 bit output which correlates to the LED pins on the DE0 Nano board. The project was to allow for hands-on experience in developing strategies for implementing Verilog code into modules to write good 'software' for good 'hardware'. This real world experience with Verilog will prove useful in the future.

Technical Approach

We were initially met with the objective of writing 12 modules to implement operations of which 6 were chosen to be tested by us. These 'modules', given to us on page 1 of the project specification are as follows:
Add, Sub, Inc, Dec, Neg, And, Xor, Not, Div4, Mod2, csl, csr.

The arithmetic modules and their respective opcodes we used are as follows in table 1.

Table 1 – Operation and Opcode

Opcode	0	1	2	3	4	5	6	7	8	9	10	11
Operation	Add	Sub	Inc	Dec	Neg	And	Xor	Not	Div4	Mod2	Csl	csr

Our ALU is to implement these as a set of instructions that we will be writing and defining in the form of modules. To begin with this understanding, it was found and noted that the first five of the twelve modules are exactly the same. They are essentially the utilization of the code from HW6 with the 1bit adder and the 4bit ripple carry adder. Thus we can essentially use one module adder and pass it in different values for the operations of 'Add, Sub, Inc, Dec, Neg'.

The general Addition module is invaluable to the operation of five of the modules we need to perform operations for. These modules are essentially just performing basic arithmetic similar to how Overloaded Operators work in C++ where you need to change the 'addition' operation to perform a role that is slightly different than what it is capable of innately. The addition module works by inputting the values of the first and second operand, the carryin bit, and to output the final result into the ALU output (switch value of binary 5 that the project specifies) and an output of the status bits. It gets the values of the results and carryout by utilizing a secondary sub-module called adder1bit that it will call within the main module to represent each single 1-bit adder inside the multi-bit ripple adder we are implementing.

The adder1bit is the sub-module of the adder module. It will be given a set of values to function, which in this case are two 'numbers' , a carry in as an input, and a result and carryout as the outputs. The variable outputs of result and carryout and intrinsic to the performance of this module and how it will work in the addition module later. Through the implementation of the basic adder1bit from HW6, we notice that this 'module' will produce a return variable of Cout(carry out) and Result. Where the carry out of the first run of the adder1bit will be the carry in of the second run of the sub-module. This continues for all eight iterations of the calling of the sub-module adder1bit within the addition module (to represent the numeric output of the ALU for the 'result' switch) until it finally gives us a final carry out and result value back. The inclusion of all the adder1bit 'return values' of result being the ALU output for the result switch, and carryout being used to represent a possible overflow status bit. Where if the last carry in bit does not equal the final carryout bit, then it means that overflow has occurred and thus the status bit of V, should be assigned a value of 1, else, a 0.

The and gate module is represented by using arithmetic operator symbol '&' as specified in Verilog to create a result of the 'ANDing' of A and B. Where we then can assign status bit Z to be 1 if the result is 0. This is because if the result is 0 from the AND operation. You can assign the negative bit by the most significant signed bit, if it is 0, it will just be 0, but if it is 1, it will change the status light.

The xor gate module is defined by the usage of the carat ^ symbol and utilizes the same idea of using a basic arithmetic operator symbol to define the result and then to assign Z a true and false statement where if the result is 0, then there is obviously a Zero status bit to be outputted and thus the value of 1.

The division gate is a representation of a shift, but done differently. Where instead of shifting it 2 with brackets >>, we are assigning positions where the result will be the two most significant signed bits followed by bits 3 to 8 of the operand of A to get a result.

The modulus gate deals with remainders of division operations. In our case, it was Modulus2 where you would divide the operand A given in by two and find if there is a remainder. If there is, we will output that there is a 'result' of 1 for there being a left over remainder.

Circle Shift Left, a relatively new idea to the coder, is fairly simple. In where the 'circular' shift can mearley be represented by the outputting the bits of 1-7 in the left hand LEDs and outputting the most signifigant bit in the right hand LED side to represent a 'circle shift' to the left.

Circle Shift Right follows a like idea to the left shifted operation but in the opposite this time. Where this time, the left handed LEDs and their represented four bit output will give the value of A in the first 'spot' and will output the rest,, bits 2 to 8, on the right hand LED.

Incrementing and decrementing the key relies on the same code but with 'previous code' already writtein in Quartus, where the result of what is given by the increment and decrement switch, seen in the project given Table 2 for the switches, will either move the value of the ROM down by 1, or up by 1.

Table 2: Switch Declaration

SW (4 bits)	**LED Value**
0000	Left two digits of the last four digits of student ID, in BCD
0001	Right two digits of the last four digits of student ID, in BCD
0010	Opcode, padded with leading 0's: {4'b0000, opcode}
0011	Address
0100	Status bits, padded with leading 0's: {4'b0000, VZCN}
0101	Result
0110	Operand A
0111	Operand B
1XXX	Available to use as you see fit

The negation relies on the Addition function but instead, it will pass in a ~operand B for the function declaration which will then produce a calculation and answer in 2s compliment. Refer to Table 3 from the project document and note the output row for the pre-given arithmetic for calculating 2s compliment.

Table 3: Output Declaration

Operation name	**Output**	**Status bits set based upon result**	**Description**
Add	A + B	V, Z, C, N	Add A and B
Sub	A + B’ + 1	V, Z, C, N	Subtract B from A
Inc	A + 1	V, Z, C, N	Increment A by 1
Dec	A – 1	V, Z, C, N	Decrement A by 1
Neg	A’ + 1	V, Z, C, N	Negative of A
And	A ∧ B	Z, N	Bitwise AND of A and B
Xor	A ⨁ B	Z, N	Bitwise XOR of A and B
Not	A’	Z, N	Bitwise complement of A
div4	A ÷ 4	Z, N	Divide A by 4
mod2	A mod 2	Z, N	Remainder of division of A by 2
csl (circular shift left)	A << 1	All status bits clear	Shift A left by 1 bit; the vacant bit positions are filled in with the bits that are shifted out of the sequence
csr (circular shift right)	A >> 1	All status bits clear	Shift A right by 1 bit; the vacant bit positions are filled in with the bits that are shifted out of the sequence

The not modulus will just output the ~A which is the negation of the operand. It is fairly simple.

Results

The results for the code was run through simulations. Please refer below to figures 1-8 for references to successful modelsim outputs for the first ROM location values as a demo. The results were such that all the outputs matched what we would have gotten on the DE0 Nano board. Where in Figure 1 and 2, I was able to output my PID successfully by setting and just changing the switches only. The clock pulse and key switch are kept the same. Only the switch value needs to be changed to output the switch bits to the LED output.

While noting that any values past decimal representation of 7 will be up to discretion and will not have a full output represented in testing on the DE0 board during validation or simulation as they are the redundant based ones.

All the lights on the DE0 board lighted up as I would expect with the right status bits in the right places performing in a order I so set defined in my declaration. This was interesting in that it showed that the logic was correct, Boolean was done correctly, and that it was a representation of a real world project that I have a correct output in.

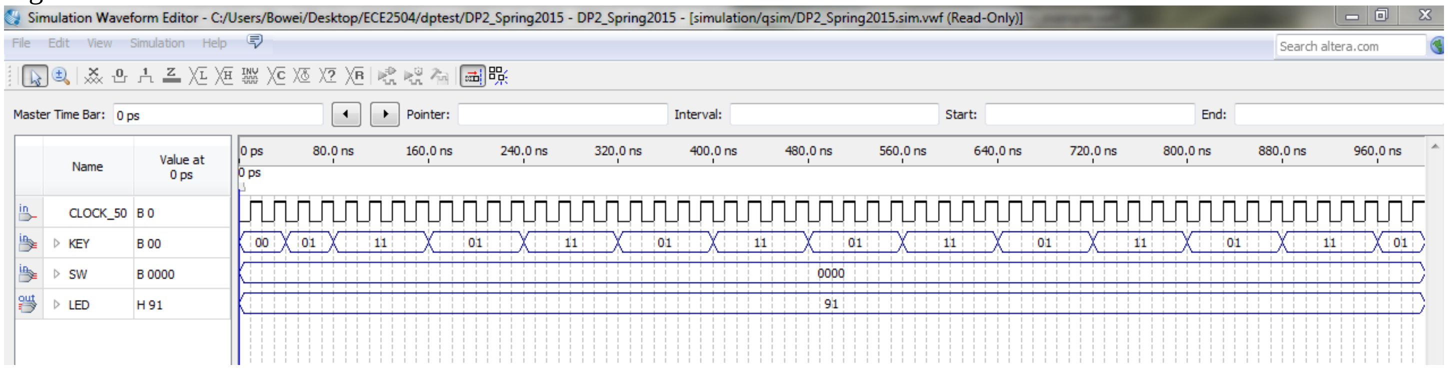
Conclusions

It is thus to be concluded that the project represented an exercise that allowed students to design, implement, and ultimately build a FPGA on a nano board. This sort of microprocessor, would deliver such outputs that are a logical setup in where our logic was proven correctly.

Personally, I felt like that the project allowed the application of skills I had developed in Intro to Computer Engineering into a real world situation. This is comparable to what you may expect out in the real world. What I had received from the project was an important toolset and proficiency in Verilog HDL that future employers may find usefu. One observation of mine however on this project would be that the Verilog and Quartus software was a learning curve yet again. We had not received as much adequate instruction as I had hoped, but I was able to ameliorate it through learning, CEL, and online tutorials. This introduces real-time problem solving skills in place of adversity and time constraints which is another factor to note on the success.

Appendicies

Figure 1: Waveform of Switch 0000 for PID left



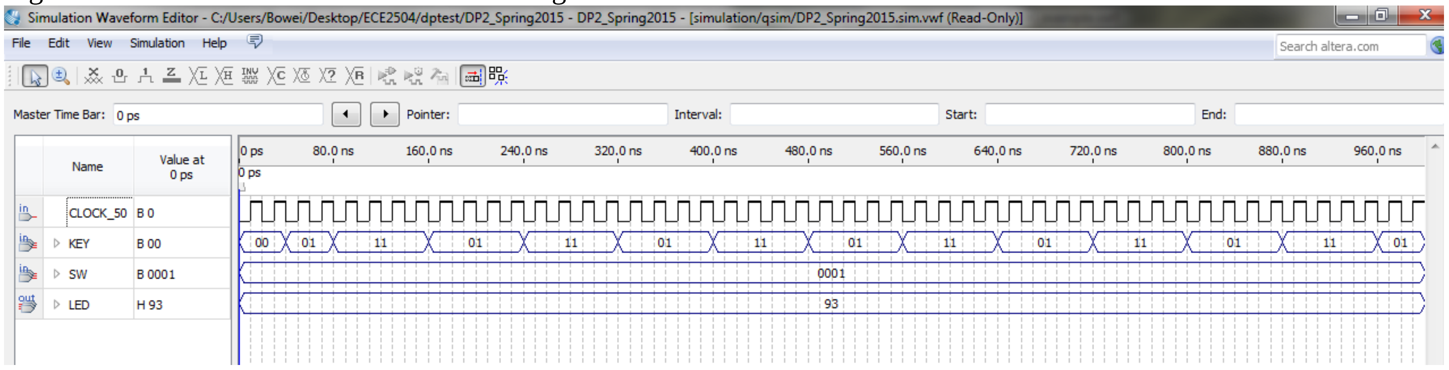
Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: Interval: Start: End:

Name	Value at 0 ps
CLOCK_50	B 0
KEY	B 00
SW	B 0000
LED	H 91

File Edit View Simulation Help
Search altera.com
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
00 01 11 01 11 01 11 01 11 01 11 01 11 01
0000
91

Figure 2: Waveform of Switch 0001 for PID right



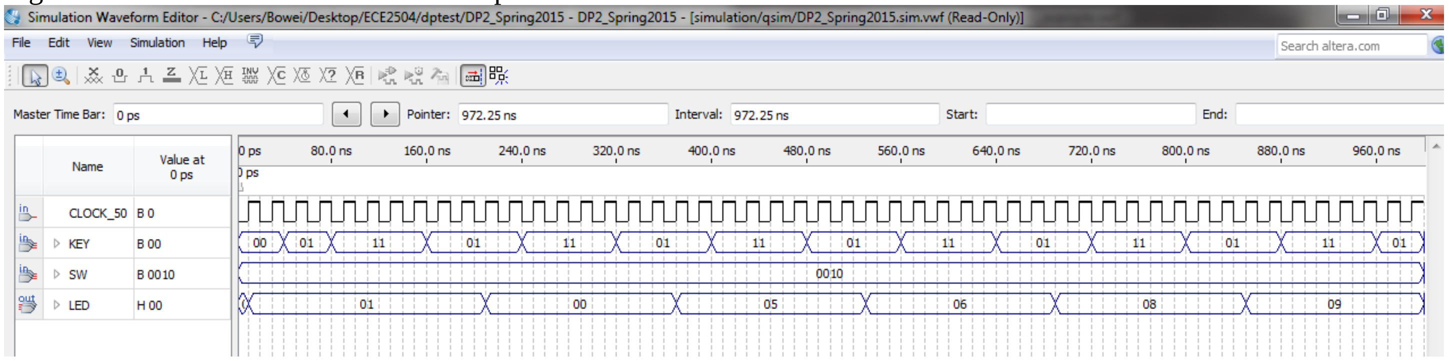
Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: Interval: Start: End:

Name	Value at 0 ps
CLOCK_50	B 0
KEY	B 00
SW	B 0001
LED	H 93

File Edit View Simulation Help
Search altera.com
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
00 01 11 01 11 01 11 01 11 01 11 01 11 01
0001
93

Figure 3: Waveform of Switch 0010 for opcode



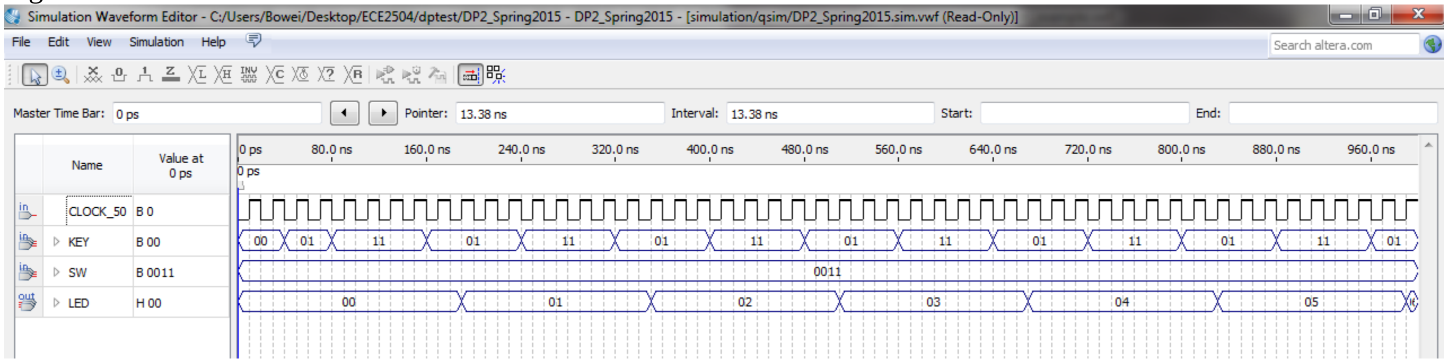
Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 972.25 ns Interval: 972.25 ns Start: End:

Name	Value at 0 ps
CLOCK_50	B 0
KEY	B 00
SW	B 0010
LED	H 00

File Edit View Simulation Help
Search altera.com
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
00 01 11 01 11 01 11 01 11 01 11 01 11 01
0010
01 00 05 06 08 09

Figure 4: Waveform of Switch 0011 for Address

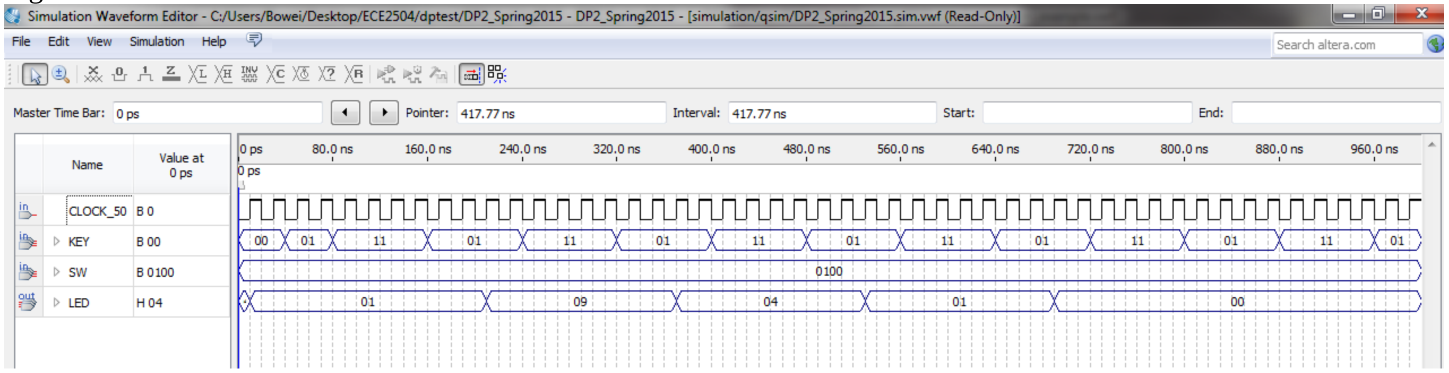


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 13.38 ns Interval: 13.38 ns Start: End:

File Edit View Simulation Help
Search altera.com
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
CLOCK_50 B 0
KEY B 00
00 01 11 01 11 01 11 01 11 01 11 01 11 01 11 01
SW B 0011
0011
LED H 00
00 01 02 03 04 05

Figure 5: Waveform of Switch 0100 for status bits

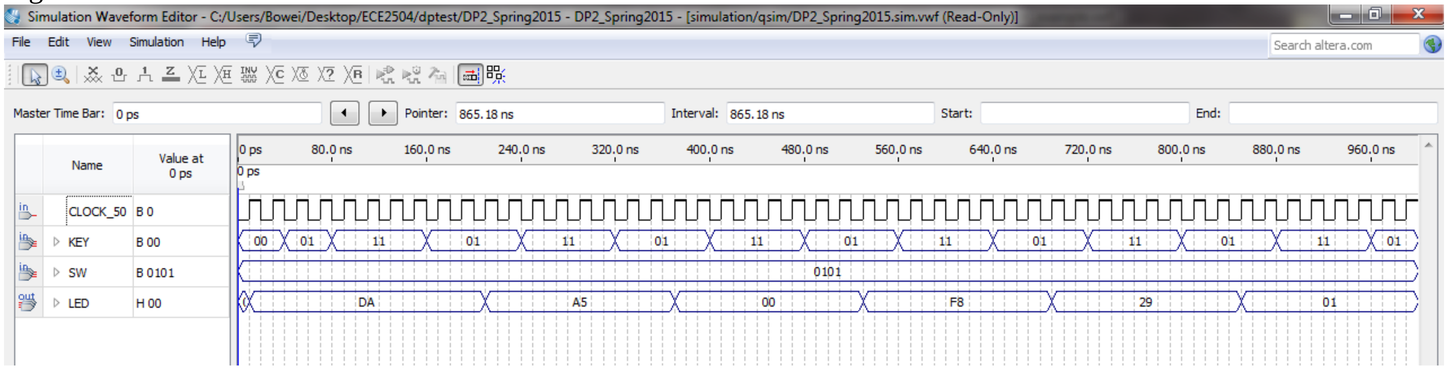


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 417.77 ns Interval: 417.77 ns Start: End:

File Edit View Simulation Help
Search altera.com
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
CLOCK_50 B 0
KEY B 00
00 01 11 01 11 01 11 01 11 01 11 01 11 01 11 01
SW B 0100
0100
LED H 04
01 09 04 01 00

Figure 6: Waveform of Switch 0101 for Result

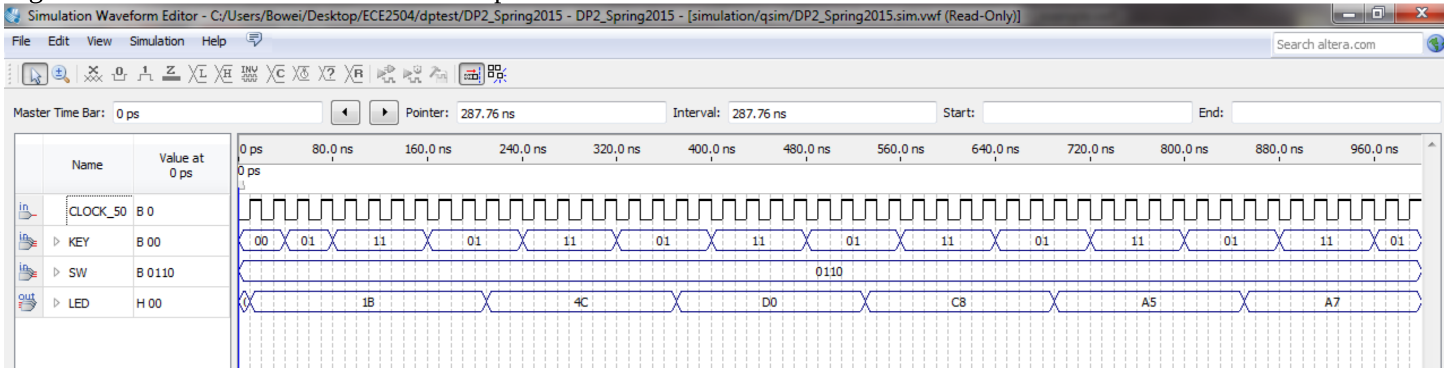


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 865.18 ns Interval: 865.18 ns Start: End:

File Edit View Simulation Help
Search altera.com
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
CLOCK_50 B 0
KEY B 00
00 01 11 01 11 01 11 01 11 01 11 01 11 01 11 01
SW B 0101
0101
LED H 00
DA A5 00 F8 29 01

Figure 7: Waveform of Switch 0110 for operand A

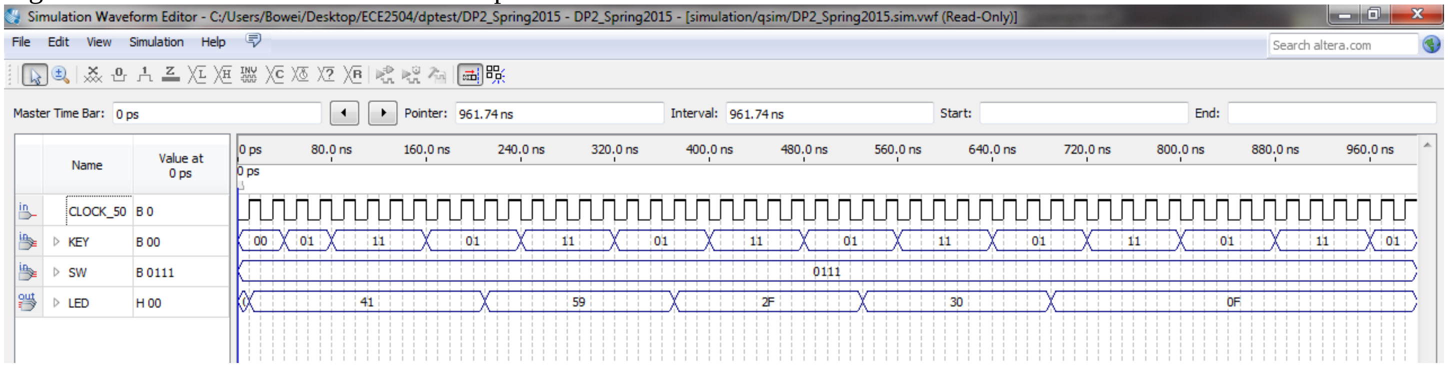


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 287.76 ns Interval: 287.76 ns Start: End:

File Edit View Simulation Help
Search altera.com
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
CLOCK_50 B 0
KEY B 00
00 01 11 01 11 01 11 01 11 01 11 01 11 01 11 01
SW B 0110
0110
LED H 00
1B 4C D0 C8 A5 A7

Figure 8: Waveform of Switch 0111 for operand B


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/dptest/DP2_Spring2015 - DP2_Spring2015 - [simulation/qsim/DP2_Spring2015.sim.vwf (Read-Only)]
File
Edit
View
Simulation
Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 961.74 ns
Interval: 961.74 ns
Start:
End:
Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
CLOCK_50
B 0
KEY
B 00
00
01
11
SW
B 0111
0111
LED
H 00
41
59
2F
30
0F