ECE 2504 – Introduction to Computer Engineering (Fall 2017)
Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board

Honor Code Requirements
Each student **must** complete this project and the associated report individually. Do not discuss any aspect of your solution or approach with anyone except for your instructor or a CEL GTA. Consider all information that you derive from your design process to be proprietary. Among other things, this includes the manner in which you implement your operations, and the number of chips that you use. Copying or using any other person's design is a violation of the Virginia Tech Honor Code, and will be prosecuted as such. You may discuss general features of Quartus and the DE0 Nano board. Direct all other questions to your GTA or to your instructor.

Objectives
- Design, simulate, and implement an Arithmetic Logic Unit from a specification.
- Write a project report describing the design process and its results.

Preparation
You must have access to a computer that can run Quartus. You must have a DE0 Nano board.

Read this project specification in its entirety. Consult the appropriate sections of Chapter 3 and Chapter 8 of the textbook. You should also consult the DE0 Nano board user's manual, particularly Chapter 3 and 6. This lab follows a simplified version of the steps described in Chapter 6 of the user's manual.

Project Description
An arithmetic logic unit (ALU) is a combinational circuit that performs a variety of common arithmetic, logic, and shift operations. An ALU has a set of control inputs that determine which of the operations will be carried out on a set of operands. For this project, the ALU will take as input two 8-bit operands, A and B, and perform the operation on those operands specified by a 4-bit operation code (opcode). The ALU will generate the 8-bit result of the operation as well as four status bits. The set of operations that you are required to implement is specified in Table 1.

Operation name	Output	Status bits set based upon result	Description
not	A'	Z, N	Bitwise complement of A
dec	A – 1	V, Z, C, N	Decrement A by 1
inc	A + 1	V, Z, C, N	Increment A by 1
xnor	$(A \oplus B)'$	Z, N	Bitwise XNOR of A and B
neg	A' + 1	V, Z, C, N	Negative of A
and	$A \wedge B$	Z, N	Bitwise AND of A and B
add	A + B	V, Z, C, N	Add A and B
sub	A + B' + 1	V, Z, C, N	Subtract B from A
lsl (logical shift left)	B << 1	All status bits clear	Shift B left by 1 bit; the vacant bit positions are filled in with "0".
asr (arithmetic shift right)	B_{6-0} >> 1 B_7 no change	All status bits clear	Shift B right by 1 bit; the MSB bit position does not change.
mult8	B x 8	Z, N	Multiply B by 8
mod4	A mod 4	Z, N	Modulus of A by 4; Note if A is negative, the result has to be positive.

Table 1: Required ALU operations

The status bits are defined to be the following (see Table 1 for which operations are affected):

- V: Overflow. This bit should be set to 1 when there is an overflow for an operation assuming the operands are in two's complement; otherwise the bit should be 0.
- Z: Zero. This bit should be set to 1 when the result of an operation is equal to decimal 0; otherwise the bit should be 0.
- C: Carry out. This bit should be set to 1 when there is a carry out from the most significant bit of the result of an operation; otherwise the bit should be 0.
- N: Negative. This bit should be set to 1 when the result of an operation is negative assuming the result is in two's complement; otherwise the bit should be 0.

The status bits should be set only for the operations as shown in Table 1. For other operations, they should be cleared.

In this project you will design, simulate and implement an 8-bit ALU that performs the operations specified above. Your ALU will be part of a larger system that will supply it with inputs and display the outputs. The larger system is shown in the schematic in Figure 1. Your responsibility is to implement the block labeled "your_ALU_mux". The rest of the system has been provided for you. The files for the project are in the Quartus archived project posted with this description.

The system takes as input the four DIP switches on the DE0 Nano board (SW[3:0]) and the two pushbuttons (KEY[1:0]). The pushbuttons control a counter that generates addresses for a small Read Only Memory (ROM). Each time KEY[1] is pushed and released, the counter advances to the next address in the ROM, and a new set of operands and opcode is applied to the inputs of the your_ALU_mux module. KEY[0] is the reset button; after the board has been programmed, it should be pushed and released before performing any other operations to put the design into a known state. The ROM contains has 16 words (addresses 0 through 15), each of which is 20 bits wide. Each word contains two 8-bit operands and a 4-bit opcode, arranged as shown in Figure 2.

19 16	15 8	7 0
Opcode	Operand A	Operand B

Table 2. Contents of each ROM address

The contents of the ROM are contained in the file called "rom.txt" in the Quartus project. Each line of the text file has the hex value for one memory location. Since each word is 20 bits wide, each line of the file has a 5 digit hex number. For example, address 0 in the "rom.txt" provided with this project is "4AA11," which means that the opcode for address 0 is 4, operand A is AA_{16}, and operand B is 11_{16}.

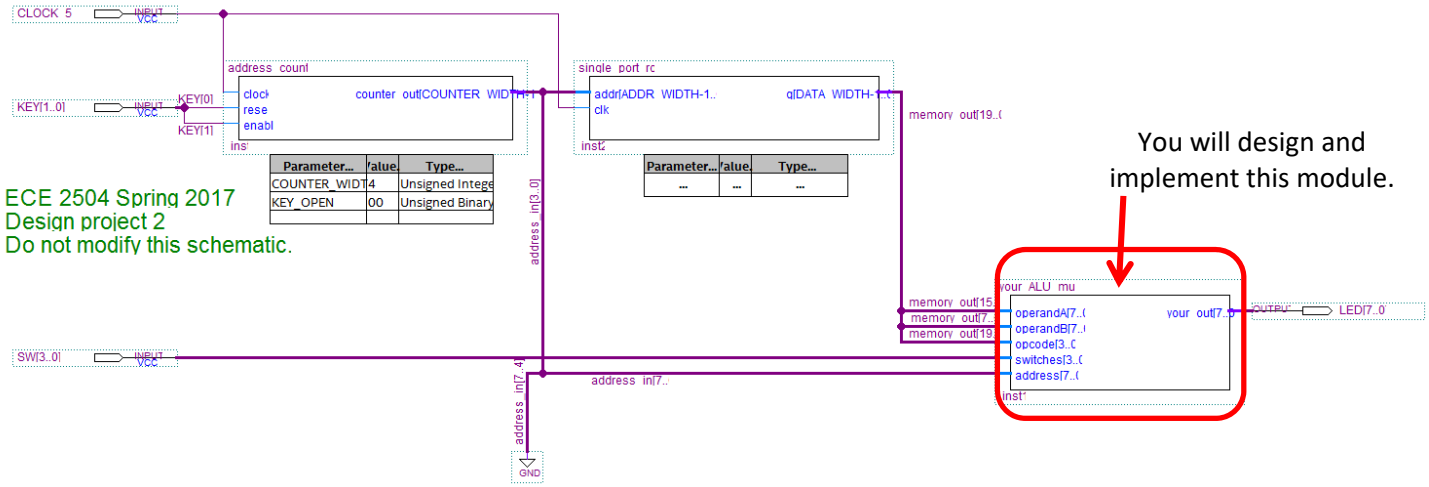

Figure 1. Schematic of full system for DP2. Your responsibility is to implement the "your_ALU_mux" module.

The module your_ALU_mux should contain your ALU as well as a 16-to-1 mux that controls the value displayed on the LEDs. The port declaration for the your_ALU_mux module is as follows:

```
module your_ALU_mux(your_out, operandA, operandB, opcode, switches, address);
     input [7:0] operandA, operandB, address;
     input [3:0] opcode, switches;
     output [7:0] your_out;
```

As shown on the schematic, the inputs operandA, operandB, and opcode are supplied from the ROM output, while the address input is supplied by the counter. The switches input is connected to the DIP switches, SW[3:0]. The output, your_out, is connected to the LEDs, with LED[7] being the most significant bit.

The DIP switches control the 16-to-1 mux in the your_ALU_mux module; the mux is used to select which value is displayed on the LEDs. The Verilog model provided for the your_ALU_mux module already has an 8 bit wide, 16 to 1 mux instantiated in it with the DIP switches connected to the select lines. You must modify the instantiation of the mux to provide the outputs shown in Table 3. Only the binary values from 1000 to 1111 are specified; the values from 0000 to 0111 are up to you. For example, you might use them to display values from the ALU other than the result and status bits to help with debugging. Your report should include a table similar to Table 3 that lists how the LEDs were used for each combination of the switches, *including what you assigned to switch values 0000 to 0111*.

You must include the last four digits of your student ID number (not PID, but student ID number) in BCD in the model using the last_four_ID_digits bus that is defined in the your_ALU_mux module. For example, if the last four digits of your ID number were "1234," then last_four_ID_digits would be set to 16'h1234 or 16'b0001001000110100. As shown in Table 3, the LEDs should then display 12_{16} for SW[3:0] = 1110 and 34_{16} for SW[3:0] = 1111.

SW[3:0]	**Value displayed on LEDs**
1000	Address
1001	Opcode, padded with leading 0's: {4'b0000, opcode}
1010	Operand A
1011	Operand B
1100	Result
1101	Status bits, padded with leading 0's: {4'b0000, VZCN}
1110	Left two digits of the last four digits of student ID, in BCD
1111	Right two digits of the last four digits of student ID, in BCD
0XXX	Available to use as you see fit

Table 3: DIP switch select lines and value displayed on LEDs

Finally, the port declaration of the ALU is not specified. At a minimum, the ports should include the two operands and the opcode as inputs, and the result and status bits as output. You can include other inputs and outputs to help with debugging the hardware. As part of your design process, you will have to assign specific 4-bit opcode values for each operation. Do not simply assign an arbitrary opcode to a particular operation: You will find that groups of operations have similar structures such that a careful choice of the opcodes will simplify the logic for controlling the operations. Your project report must include a table showing the opcode values that you chose for each operation.

Requirements and constraints

1. You are permitted to modify only the Verilog file your_ALU_mux.v, and the ROM contents file, rom.txt. You must not modify any other file or schematic in the project. Any additional modules that you might need to create to implement your design should be included in the your_ALU_mux.v file.
2. You are not permitted to modify the port declaration of the your_ALU_mux module.
3. You must implement your design using structural and dataflow Verilog constructs (gate primitives, assign statements with operators). You are not permitted to create any schematics or to use behavioral Verilog constructs (e.g., case statements, for loops). *Note that if you use dataflow Verilog it will be more difficult to determine your gate count and propagation delay.*
4. Your design must be completely combinational.
5. Your design must display the last four digits of your student ID number as shown in Table 3.

Procedure

The Quartus archive project file posted for this project includes the schematic and Verilog files necessary to build the system. Do not modify any files except for the `rom.txt` file and the `your_ALU_mux.v` file. In particular, if you change the top level schematic or modify the pin assignments, there is a chance you could damage your DE0 Nano board.

While the provided Quartus project does not fully implement the project, it is functional and can be synthesized and then used to program Field Programmable Gate Array (FPGA) on the DE0 Nano board using the programming instructions provided below. Before modifying the `your_ALU_mux.v` file, you should program the FPGA on the DE0 Nano board with the provided Quartus project to be sure you understand the programming procedure and the behavior of the pushbuttons and switches. You should confirm that the switch settings behave as described in the model. As stated above, the version provided to you does not correctly implement the project specification.

Develop a Verilog model of the ALU by itself and test it in simulation to make sure it behaves correctly. You will want to create a new project in Quartus just for your ALU. (The instructions for creating a new project and simulating a model have been provided in earlier assignments; please refer to those instructions if you have any questions about those tasks.) Rather than building the whole ALU and then simulating it, you should build and simulate the ALU one operation at a time. Depending upon your design, you might create smaller functional blocks that will be used to implement the ALU; if so, you should simulate each of those blocks individually to make sure they behave correctly before you connect them together into larger blocks. Include simulation results of your ALU for each operation for multiple operand values in your report. Label the results to show which operation is being tested. You should carefully consider which sets of inputs are used to test your ALU so that a wide variety of cases are covered.

Once you are confident that your ALU is working correctly, copy it into the `your_ALU_mux.v` file in the provided Quartus project and instantiate it. You can then simulate the whole system. A sample simulation input file is included with the project, `dp2_Fall2017_input_waveform_example.vwf`, to give you an idea how to simulate the full project as shown in Figure 2. The counter and ROM in the schematic are sequential circuits that are positive edge triggered, so the KEY and SW inputs should not change on the positive edge of clock. Furthermore, the counter only changes the address value when KEY[1] is pushed and then released, so KEY[1] must go from 1 to 0 and back to 1 for the counter to advance. Finally, the KEY and SW values should remain steady for several clock cycles after any change in value to mimic the real hardware: the clock is running at 50 MHz, which is much (much!) faster than you can change the pushbuttons and DIP switches. Include simulation results of the whole system in your report, showing that the output behaves correctly for several combinations of switch settings and ALU operations.

After your model simulates satisfactorily, you must compile it and then program the DE0 Nano board with it. Please refer to sections 6.8 and 6.9 of the DE0 Nano user's manual for full instructions. Briefly, to compile your design, double-click **Compile Design** in the Tasks window, or select **Processing**->**Start Compilation** from the toolbar. The project files provided to you will generate 5 warnings during compilation (9 if you are running multiple processors); these can be ignored. If you receive critical warnings or errors, however, you should correct them before continuing. When the compilation completes with no errors, it will generate an "SRAM Object File", .sof, which you will use to program the FPGA on the DE0 Nano board. The *.sof file will have the same name as the project, in this case, `DP2_Fall2017.sof`.

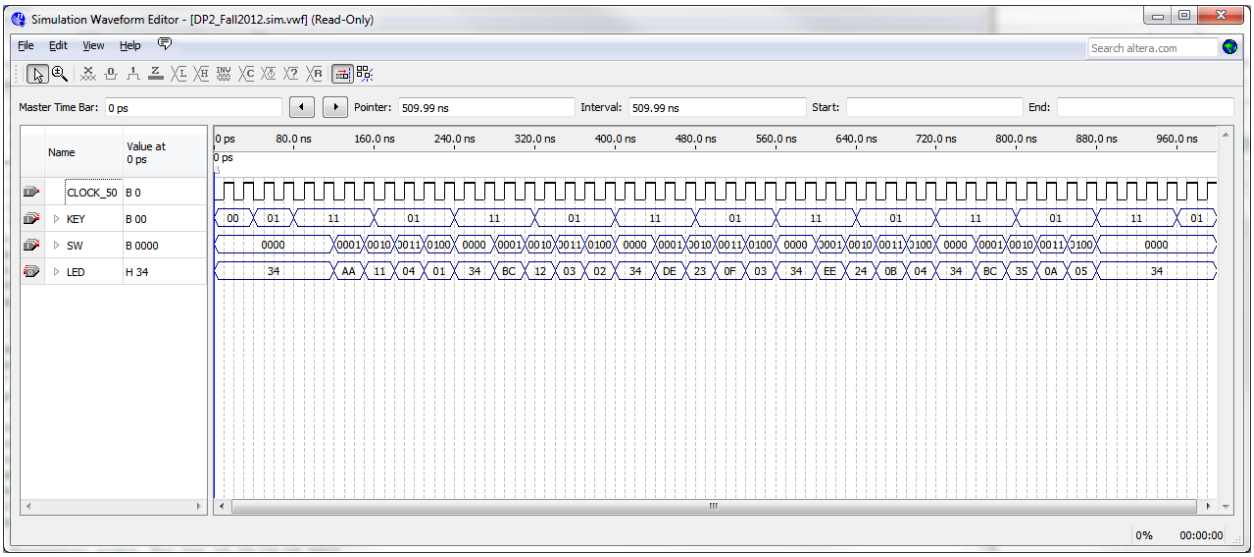

Figure 2. Sample output simulation file for testing the whole system.

When the project is successfully compiled, you can program the DE0 Nano board. Connect the board to your computer using the USB cable provided with the board. Then double-click on **Program Device (Open Programmer)** in the Tasks window, or select **Tools**->**Programmer** from the toolbar. The programmer window should open, as shown in Figure 3. The `DP2_Fall2017.sof` file should be shown as the file to be downloaded to the board. If it does not appear, select **Add File** and then select the `DP2_Fall2017.sof` file from the project directory.

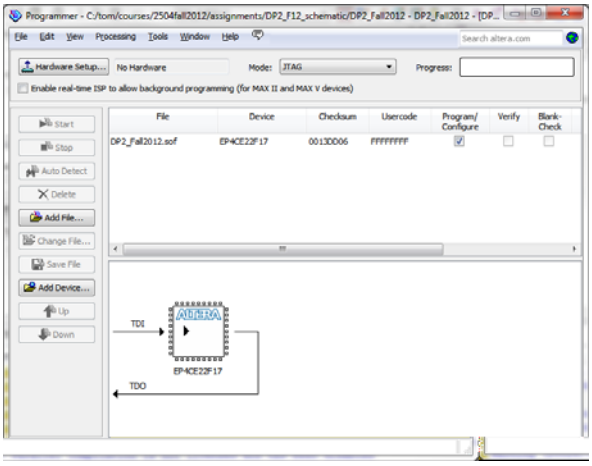

Figure 3. Programmer window.

Important note: The first time that you open the programmer window, you must click Hardware Setup, and turn on the **USB-Blaster [USB-0]** option under the **Currently selected hardware** pull down menu as shown in Figure 4. Then select close and return to the programmer window.

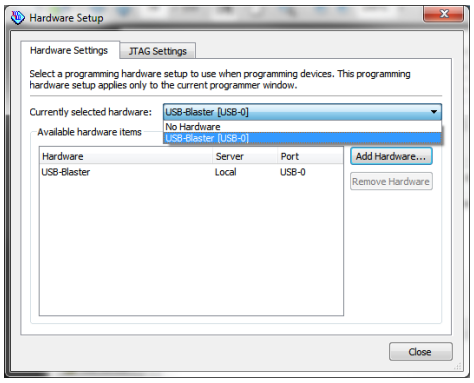

Figure 4. Choosing the USB-Blaster option on the Hardware Setup window

In the programmer window, click on the **Start** button. The start button will gray out, and the LEDs on the DE0 Nano board will dim. In a few seconds, the Progress indicator on the programmer window should turn green and say 100% (Successful). Your design has now been downloaded to the board.

To test your design, you should begin by resetting the counter by pressing and releasing the KEY0 pushbutton. You can now operate the design by using the DIP switches to control what is displayed on the LEDs and using the KEY1 pushbutton to advance to the next location in the ROM.

Once you are confident that your design is working on the DE0 Nano board, you should change the `rom.txt` file so that the operations are as shown in Table 4 (all values are in decimal, but you must use hexadecimal in the `rom.txt` file), which will be used to validate your project in the CEL.

ROM address	**operation, operand A, operand B**
0	sub, A = 17, B = -87
1	add, A = 119, B = -11
2	and, A = -57, B = 118
3	xnor, A = 63, B = -64
4	not, A = 73, B = -57
5	mult8, A = 118, B = -9
6	inc, A = -82, B = 63
7	neg, A = -47, B = 17
8	dec, A = -107, B = -57
9	mod4, A = -47, B = -11
A	asr, A = -9, B = -103
B	lsl, A = 63, B = 92

Table 4. ROM contents of addresses 0-B to be used for validation (all values are in decimal)

Using your final ALU design, determine the number of 2-input gates utilized and the propagation delay of the entire ALU. Include this information in your report. You must include an explanation of your calculations and any assumptions.

Circuit Validation

This project does not require CEL validation. Instead, you will provide the source files that will allow the GTA to compile the same files that you used to implement your ALU on the DE0 Nano Board. Create an archive of your work by choosing Project > Archive Project after you complete the implementation. When you create the archive (*.qar) , it should appear

in the same folder that was created when you opened the original archive. Upload the archive to Canvas. Make certain that you upload the completed archive that you created, and not the one that was provided to you.

Project Report

After you have validated your logic circuit, prepare and submit a written lab report that presents a detailed discussion of the project. It should include the design approach you followed, the final design you implement, the design decisions that you made and the alternatives you considered, your simulation results, your observations, and your conclusions. Subdivide your report into logical sections and label them as appropriate. The last two pages of your report must be the validation sheet. Be sure to include all the required elements. Refer to the Assignment rubric to see how this project will be graded.

Submission Requirements:

Submit the following files on Canvas. **Do not put them in a zip file!**

1. Your Archived Project **DP2_Fall2017_ .QAR**
2. Your completed Verilog file **your_ALU_mux.v**
3. Your modified memory file **rom.txt**
4. Your project report as a pdf or doc. Remember the _____ should be the validation sheet. Remember to complete page 1 of the validation sheet!

Grading

The design project will be graded on a 100 point basis, as shown in the Assignment rubric.