**ECE 2504: Introduction to Computer Engineering (Spring 2015)**

**Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board**

**Read the entire specification before you begin working on this project!**

**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 Web Edition. You must have a DE0 Nano board.

Read this project specification in its entirety. Consult the appropriate sections of Chapter 4 and Chapter 7 of the textbook. You should also consult the DE0 Nano board user’s manual, which is on the DVD included with your board, 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** |
| 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 |

Table 1: Required ALU operations

The status bits are defined to be the following:

* V: Overflow. This bit should be set to 1 when there is an overflow for an arithmetic 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 arithmetic 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 arithmetic operation; otherwise the bit should be 0.
* N: Negative. This bit should be set to 1 when the result of an arithmetic 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.

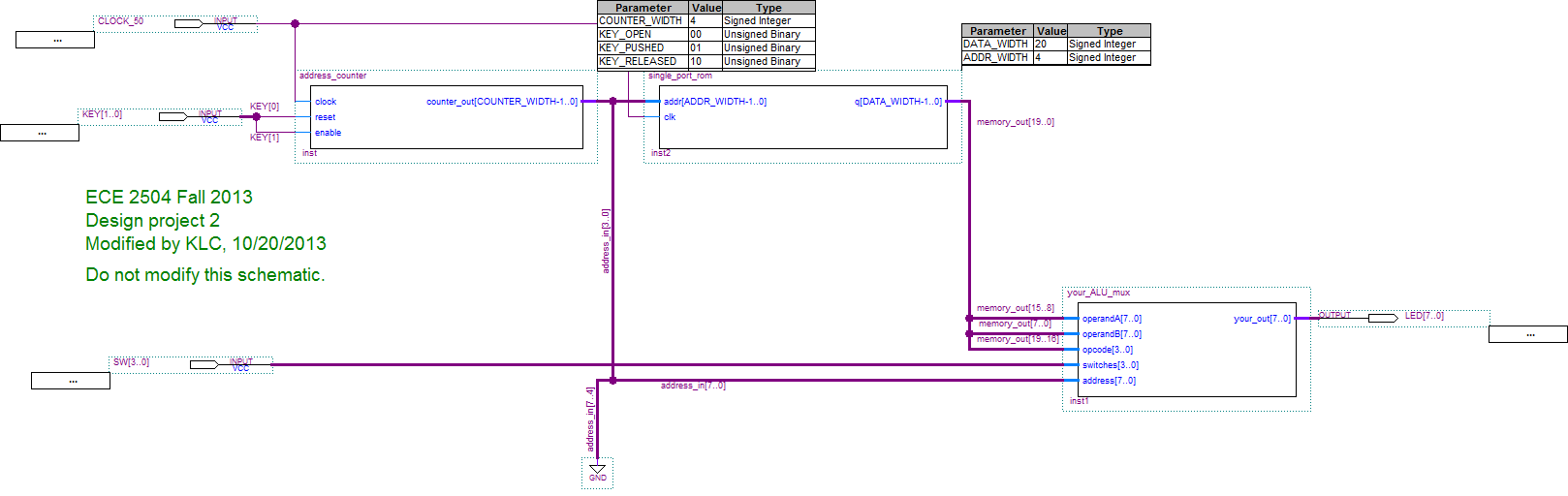
In this project you will design, simulate and implement an 8-bit ALU that performs the operations specified above. Your ALU will be part of a larger system that will supply it with inputs and display the outputs. The larger system is shown in the schematic in Figure 1. Your responsibility is to implement 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 AA16, and operand B is 1116.



You will design and implement this module.

Figure 1. Schematic of full system for project 1. 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 a 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 0000 to 0111 are specified; the values from 1000 to 1111 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.

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. As shown in Table 3, then the LEDs should display 1216 for SW[3:0] = 0000 and 3416 for SW[3:0] = 0001.

|  |  |
| --- | --- |
| **SW[3:0]** | **Value displayed on LEDs** |
| 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 |

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 a specific 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).
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\_spring2015\_input\_waveform\_example.vwf, which can be used 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 9 warnings during compilation; 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\_Spring2015.sof.

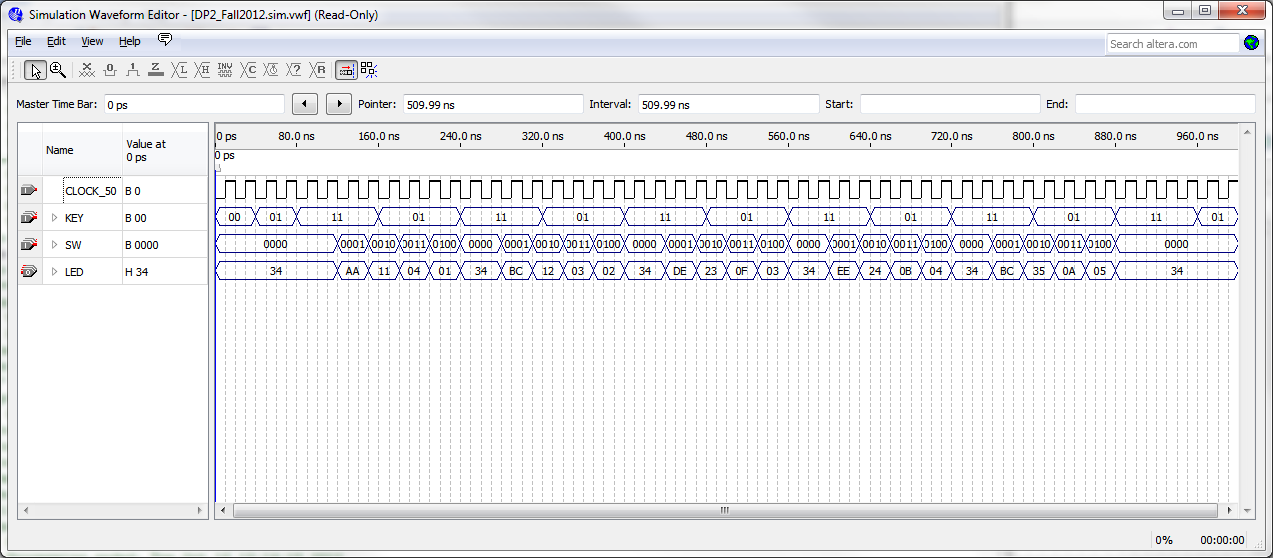


Figure 3. Sample input simulation file for testing the whole system.

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\_Spring2015.sof file is shown as the file to be downloaded to the board. If it does not appear, select “Add File” and then select the DP2\_Spring2015.sof file from the project directory.

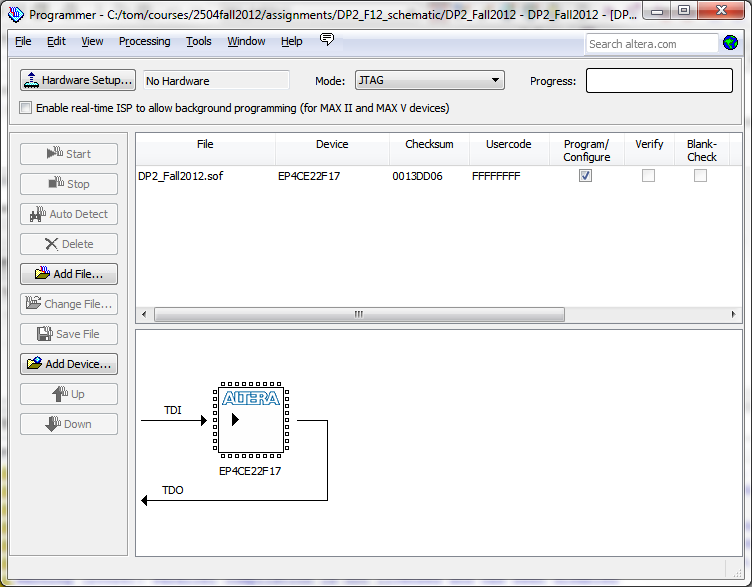


Figure 4. Programmer window.

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 5. Then select close and return to the programmer window.

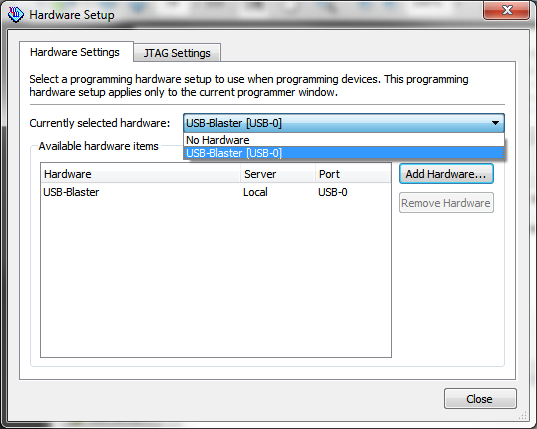


Figure 5. Choosing the USB-Blaster option on the Hardware Setup window

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 = 27, B = 65 |
| 1 | add, A = 76, B = 89 |
| 2 | and, A = 208, B = 47 |
| 3 | xor, A = 200, B = 48 |
| 4 | div4, A = 165, B = 15 |
| 5 | mod2, A = 167, B = 15 |
| 6 | csl, A = 178, B = 43 |

Table 4. ROM contents of addresses 0-6 to be used for validation (all values are in decimal)

Q: But wait a minute! 8-bit 2cm numbers have a range of -128 to +127. Why does the validation sequence refer to numbers larger than that?

A: This is actually intended as a timesaver for you. Just because the instruction sequence refers to some numbers as being positive doesn't make them so. Numbers don't get stored with any "meaning" when they end up in a binary representation. We've discussed this fact in class - numbers receive the meaning that we impose upon them when we look at them. So it's perfectly legitimate to call a number one thing in one context and recognize that it will be viewed differently in a different context.

The number 192 has binary representation 11000000 if viewed as unsigned.

The number -64 had binary representation 11000000 if viewed as a signed 2's complement number.

(Aside 1: note that the "sign" bit for both of these is a 1 for status purposes.)

(Aside 2: if you divide -64 by 4, the result should still be a negative number.)

Since both numbers have the same representation in binary, we can discuss them as though they were the same in terms of register contents (contents = 0xC0), since the register isn't attached to our viewpoint in the first place.

In the places where the validation instruction sequence is referring to numbers greater than 127, it's referring to negative numbers. They're only being referred to as numbers that are positive as a time saver for you. The reason you had it recommended to you that you try numbers over 200 as sample operands is to make certain that you test your datapath with numbers that are sure to be (or at least should be) interpreted as being negative by your datapath

After you have built your circuit and are satisfied with its operation for these operations, take your computer and DE0 Nano board to the CEL and have the GTA validate the circuit by completing the included validation form. The waiting lines in the CEL can be very long as the due date approaches, so validate as early as you can. You should have your Quartus project open on your computer before validation for the GTA so that he can examine your design if necessary.

**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.

**Your lab report should be submitted on Scholar. Your validation sheet should be submitted as a hard copy.**

Refer to Section 6 of the course Lab Manual for details on preparing your report. Prepare your report on your word processor. Do not include hand-written items. Proofread your report to ensure that it is free of spelling and grammar errors.

Use the cover sheet included with this specification as the first page of your report. Do not use any other cover page. Please submit your report with completed cover sheet on Scholar. You must also submit your rom.txt file and your your\_ALU\_mux.v file on Scholar.

**Grading**

The design project will be graded on a 100 point basis, as shown on the cover sheet.

**Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board**

**Validation Sheet – Page 1**

**The Validation Sheet for Project 2 is two pages long. Make sure that you take both pages of this Validation Sheet and your student ID card to the CEL when you go to have your design validated.** Before you begin validation, you should have your Quartus project open on your laptop with your DE0 Nano board connected, and the programmer window open and ready to program the board. Your rom.txt file should be open in the Quartus window for the GTA to see.

**No GTA or student should discuss any aspect of a student’s design with another student. Among other things, this includes the manner in which a student implements specific operations.**

**It is the student’s responsibility to make sure that this validation sheet is completed correctly. If there are any questions about the validation, the student should check with the instructor.**

**All sections of this Validation Sheet must be completed in INK. Failure to use ink will result in a validation grade of 0.**

Student Name (printed in ink):

GTA Validation Instructions

Do not validate this lab if the student has not printed their name in ink above and on the next page. Before continuing, verify that the name printed in both places is the name on the student’s ID card.

1. Program the FPGA on the DE0 Nano board using the Start button on the programmer window.
2. When the programming has successfully completed, reset the design by pressing and releasing the KEY0 pushbutton.
3. Set the DIP switches to 0000 and record the value of the LEDs as two hex digits in Table 1 on the next page. (The DIP switches are 0 on the side labeled “ON”.)
4. Set the DIP switches to 0001 and record the value of the LEDs as two hex digits in Table 1.
5. Compare the four digits from steps 3 and 4 to the last four digits of the student’s ID number on their ID card. **If the four digits do not match the last four digits of the student’s ID number on their ID card, STOP THE VALIDATION. DO NOT CONTINUE.**

For the remaining steps of the validation, the values of the switch settings in Gray code order will allow the requested items to be checked in the order in the table more quickly. **All values should be recorded in the table as two hexadecimal digits.** For each address, verify that the opcode, operand A and operand B digits match the value in the corresponding address in the rom.txt file on the student’s computer.

1. Fill in the row of Table 2 for the sub operation. Verify that the address is 0.
2. Press and release KEY1.
3. Fill in the row of Table 2 for the add operation. Verify that the address is 1.
4. Press and release KEY1.
5. Fill in the row of Table 2 for the and operation. Verify that the address is 2.
6. Press and release KEY1.
7. Fill in the row of Table 2 for the xor operation. Verify that the address is 3.
8. Press and release KEY1.
9. Fill in the row of Table 2 for the div4 operation. Verify that the address is 4.
10. Press and release KEY1.
11. Fill in the row of Table 2 for the mod2 operation. Verify that the address is 5.
12. Press and release KEY1.
13. Fill in the row of Table 2 for the csl operation. Verify that the address is 6.
14. Sign and date the validation sheet below.

**Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board**

**Validation Sheet – Page 2**

Table 1: Checking BCD equivalent to student ID

|  |  |  |
| --- | --- | --- |
|  | **Switch setting, SW[3:0]** | |
|  | **0000** | **0001** |
| LED value in hexadecimal | \_\_\_\_ \_\_\_\_ | \_\_\_\_ \_\_\_\_ |

Table 2. Checking the operation of the ALU. All values of the LEDs should be recorded as two digit hexadecimal numbers.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SW[3:0] →** | **0011** | **0010** | **0110** | **0111** | **0101** | **0100** |
| **operation** | **address** | **opcode** | **operandA** | **operandB** | **result** | **status** |
| sub | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| add | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| and | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| xor | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| div4 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| mod2 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| csl | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |

Comments (only required if something is unusual or wrong):

GTA Printed Name and Signature:

Date and Time of Validation: