ECE 3544: Digital Design I
Project 5 – Datapath and Finite State Machine Design

Read this specification in its entirety before you begin working on the project!
Read the Assignment page on Canvas where you downloaded the project materials!

Honor Code Requirements
You must do this assignment individually. The following represent unauthorized aids, as the term is used to define cheating in the Virginia Tech Honor System Constitution:

- Discussing *any aspect or result* of your design with *any person* other than your instructor and the ECE 3544 GTAs. This includes but is not limited to the implementation of Verilog code, as well as the supporting details for that code – state diagrams, state tables, block diagrams, *etc.*
- Using *any design element* – including Verilog code – from *any printed or electronic source* other than your course textbook and those sources posted on the course Canvas site.

It is permissible for you to re-use any of your own original Verilog code. It is also permissible to use the keypressed *module that was provided to you in Project 3B.*

All code submitted is subject to plagiarism checking by MOSS (theory.stanford.edu/~aiken/moss/). Any copying flagged by MOSS will be treated as Honor Code violations and submitted to the Virginia Tech Undergraduate Honor System.

Project Objective
In this project, you will design and implement a datapath, along with a finite state machine to model the controller for the datapath. The core of the FSMD will center on two interacting state machines: one to supply values from the inputs to the operand registers and datapath, and one to perform operations.

While the majority of the instructions can be structured as micro-operations, the FSMD will contain a multiply instruction that it will perform over multiple cycles.

Requirements
The DE1-SoC board IS REQUIRED for this project.

You must have the current version of ModelSim ALTERA STARTER EDITION and Quartus II Web Edition 14.1 installed on your computer. The instructions are done using these versions of ModelSim and Quartus. While the directions may be consistent with other versions of ModelSim and Quartus, you must use the versions indicated above.

Datapath Operations

Operation	**Result of the Operation**	**Operation code**
XOR Your ID[1]	Bitwise-XOR the value {A, B} with the last four digits of your student number, expressed in BCD.	0000
(1's) Complement[2]	Bitwise-complement the value of A.	0001
AND[2]	Bitwise-AND the values of A and B.	0010

The Operation table continues on the next page.

OR[2]	Bitwise-OR the values of A and B.	0011
Negate[3]	Take the 2's complement of A. Take A as an 8-bit signed 2's complement operand, sign-extended to 16 bits.	0100
Add[3]	Add A and B. Take A and B as 8-bit signed 2's complement operands, sign-extended to 16 bits.	0101
Subtract[3]	Subtract B from A. Take A and B as 8-bit signed 2's complement operands, sign-extended to 16 bits.	0110
Logical Shift Left[4]	Logically shift A to the left by B positions. Take B as a 4-bit unsigned operand.	1000
Arithmetic Shift Right[4]	Arithmetically shift A to the right by B positions. Take B as a 4-bit unsigned operand.	1001
Rotate Right[5]	Rotate A to the right by B positions. Take B as a 4-bit unsigned operand.	1010
Rotate Left[5]	Rotate A to the left by B positions. Take B as a 4-bit unsigned operand.	1011
Add Accumulate[3]	Add A to the previous result. Take A as an 8-bit signed 2's complement operand, sign-extended to 16 bits	1100
Multiply[6]	Multiply A and B. Take A as the multiplicand, and B as the multiplier.	1101
Debug[7]	Display the values of internal signals in your design. You may choose these values as needed.	Any unused opcode

Notes

1. Suppose that your student ID ends with the digits 1854. For A = 11111010 and B = 01010011, the result of this operation would be $(1854)_{16}$ XOR $(FA53)_{16}$ = $(E207)_{16}$
2. For the logical operations, the result is an 8-bit value zero-filled to 16-bits.
3. For the arithmetic operations, the 16-bit result should reflect an operation performed upon 16-bit sign-extended versions of operands A and B.
4. For the arithmetic shift operations, operand A represents a signed 2's complement value. The result should reflect a 16-bit sign-extended version of operand A shifted by a number of positions equal to the value of operand B. Do not use the Arithmetic Shift dataflow operators to implement your arithmetic shift operations
5. For the rotate operations, you are rotating an 8-bit value (A) through what amounts to a 16-bit register. Imagine that operand A is *zero-filled* to 16 bits before being rotated.
6. For the multiply operation, A and B are unsigned values. The 16-bit result represents an unsigned value. Do not use the multiply dataflow operator to implement your multiplier.
7. For debug, the operation codes are to be defined by you as part of your design.

System Interface

The system has the following inputs:

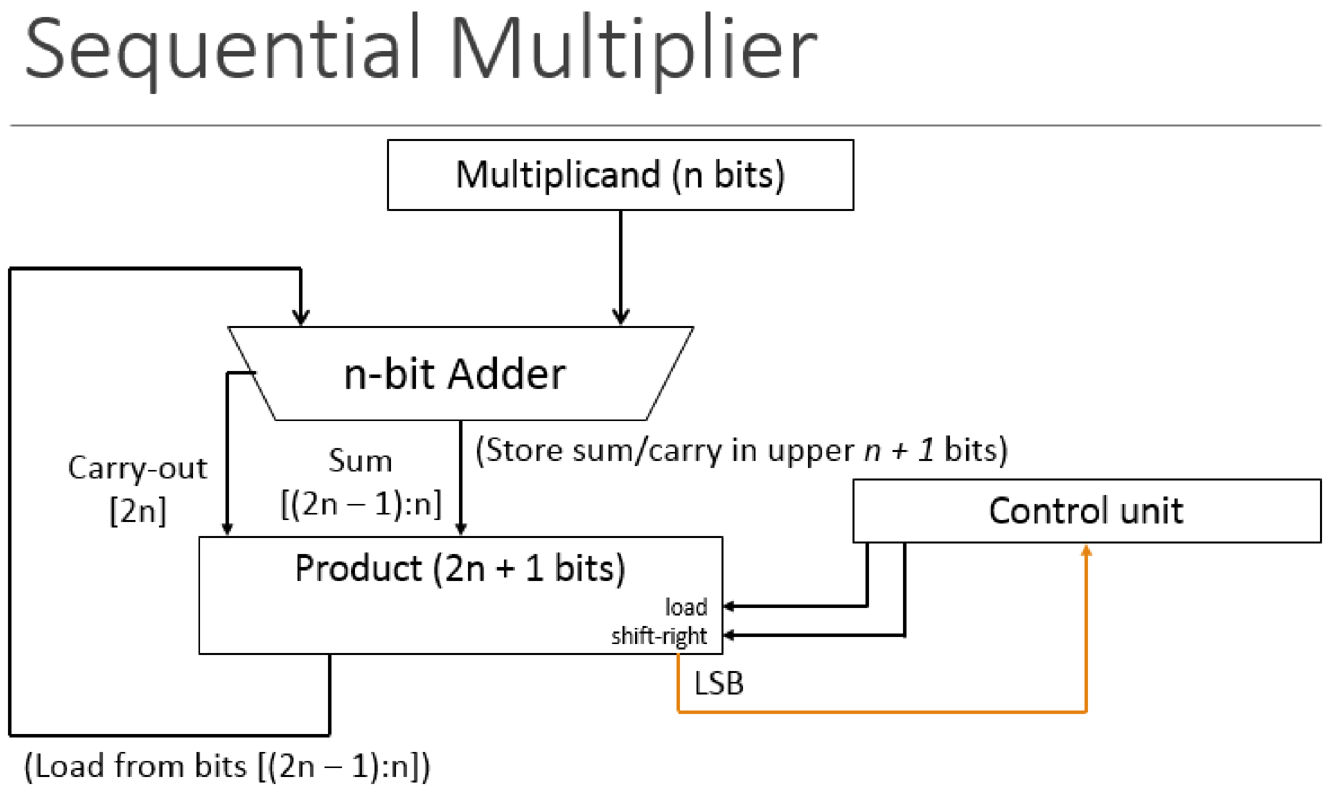
- Clock / CLOCK_50 – The DE1-SoC board has four 50 MHz clock signals. Use CLOCK_50 as the clock for the sequential elements of your design. Do not use the switches or pushbuttons as clock signals.
- Enter / KEY[1] – The Enter button is used to enter values from the Switches as described in the **Method of Operation**, and to advance the multiplication operation to its next partial product.
- Reset / KEY[0] – The Reset button is an active-low system reset. Pressing and releasing the Reset button should return the system to its initial state.
- Switches / SW[7:0] – The system uses the Switches to provide operand values. Operation codes are 4-bit values that use SW[3:0]. In general, A and B are 8-bit values, but in cases where B is a 4-bit value, it should use SW[3:0].

The system has the following outputs:

- Hex Displays / HEX[3:0]: For regular operations, the hex displays should show the current value of the result register in hexadecimal. HEX[0] is the least significant digit. Regular operations should use all four displays to show a 16-bit result, with leading zeroes as needed. The result should appear as a 16-bit signed 2's complement value as needed. For example, $(-1)_{10} = (FFFF)_{16}$. For debug operations, provide documentation in your report that describes the significance of each displayed value and the manner in which the value is displayed.

  For all operations except the multiply operations, the result should be displayed immediately after all operands have been entered. You must implement the multiply operation using the method for serial multiplication described in the lecture slides. Each time the user presses and releases Enter, the result should show the value of the product register after each shift.

For reference, here is a block diagram for the serial multiplier implementation that your datapath must use:



Switches SW[9:8] are available as optional inputs for you to use in debugging your design. LEDs LED[9:0] and hex displays HEX[5:4] are available as debugging outputs. If you choose to use these inputs and outputs for debugging, document your methodology for using them in your project report. If you choose not to use the optional outputs, they should be turned off.

Method of Operation

After being reset, the system should wait for an operation. To enter an operation, set the Switches to an operation code, then press and release Enter. To enter operand A, set the Switches to the value of A, then press and release Enter. For operations that use operand B, set the switches to the value of B, then press and release Enter.

For all operations except for the multiply operations, the result should be displayed as soon as the last operand has been entered. For example, the result of the addition operation should be displayed as soon as Enter is released when entering B.

For the multiply operation, after multiplier B is entered, the user must press and release Enter eight more times – once for each step in generating the product. After B is entered, the result should display the value of the product register with its initial value. Since we have eight bit operands, the register must be shifted eight times. For each of the next eight presses of Enter, the result should display the product register after shift is performed.

After each operation is completed, the system should go back to waiting for an operation. While waiting for an operation, the seven segment displays should display the result of the operation that was just completed. Changing the switches should not affect the value displayed by the seven segment displays.

For your debug operations, you should consider internal values of your design to display on the LEDS and HEX5/HEX4 displays. You should briefly describe any design decisions you made about the debug operations in your report.

In summary, the steps for operating the device are:

0. Between operations, the system is waiting for the user to enter an operation code.
1. Set SW[3:0] to the desired operation code. Press and release Enter.
2. Set SW[7:0] to the value of operand A. Press and release Enter. *For operations requiring only operand A, go to Step 4.*
3. For operations requiring operand B, set SW[7:0] to the value of operand B. Press and release Enter.
4. For all operations except the multiply operation, HEX[3:0] should now display the result. *Return to Step 0.* For the multiply operation, the displays should now show {8'b0, B}. *This is the initial value of the product register in the multiplier described in the lecture notes. Continue to Step 5.*
5. For the multiply operation, press and release Enter eight more times to show the result. After each press and release, HEX[3:0] should show the value the product register. After the eighth press, HEX[3:0] should display the result of the multiply operation. *Return to Step 0.*

Design Tips

- Before writing any modules, work several examples by hand for each operation to make sure you understand how they should work.
- Design smaller aspects of this system individually. Implement and test them before moving on to the next aspect. You will be more successful taking this approach, as opposed to trying to implement the whole system before testing any of it.
- Break the design into communicating finite state machines, e.g., a state machine for the multiply operations and another for entering operation codes and operands.
- Test each of your components in simulation using a test bench before trying them on the DE0 Nano.

Project Submission

Write a report describing your design and implementation process.

- Discuss the decisions you made about implementing the elements of your design.
- Include a detailed block diagram of the interacting units in your system.
- Include a state diagram for the system controller, and the state diagrams of other units that might require them.
- Include waveforms showing the correct behavior of your design. Provide a waveform for each of the operations.
- Document any additions you make for debugging purposes, and describe your motivation for each debug operation.

I have included the validation sheet that the GTAs will use to test your design after the submission deadline. You do not have to go to the CEL to have your project validated. Instead, you should use the information in the validation sheet as one basis for testing your design before you submit your project.

Your project submission on Canvas should include the following items:

1. Project report in Word or PDF.
2. A Quartus Archive containing the source files for your top-level module, any modules that the top-level module requires to function, and your test benches for the top-level module. To create the Quartus Archive, choose **Project > Archive Project** after you complete your implementation. When prompted for a name for your archive, the default archive name will be the same as the original archive. *Append your Virginia Tech PID to the end of the filename*. Make certain that you submit the archive that you create – the one containing your solution to the project – and not the one that I provided to you – the one that only contains the initial files.

Put your report and Quartus Archive into a single .zip file. *The Canvas Assignment page where you downloaded the project materials may list additional submission requirements. Follow all submission instructions regardless of their source or placement.*

Your top-level module may be tested with our own secret test bench, so it is important that you use the module declaration provided with the archived project. You must also include the source files for every module required for your top-level module. Failure to do so will result in a grade of 0 for that portion of the project.

Grading for your submission will be as described on the cover sheet included with this description.