Cpr E 281 FINAL PROJECT

ELECTRICAL AND COMPUTER ENGINEERING
IOWA STATE UNIVERSITY

FINAL Project

Objectives

The main objective of the final project is to teach you how to put together all of the class material that you have learned so far in order to program the Altera DE2 board to carry out an independent activity. You will have to design a circuit to realize one digital machine that accepts input from switches and outputs to LEDs and 7-segment displays.

Project Selection

Feel free to choose any one of the following four projects. They are all of approximately the same difficulty. Your grade will not depend on the project that you pick, as long as you implement the entire project.

If you don't like any of the provided projects, then you also have the option of specifying your own project. If there was something that you always wanted to do in digital design, now is your chance. However, your proposed project MUST be of appropriate complexity as determined by your lab TA. The minimum constraints are that y*our project MUST: 1) include a state machine, 2) include a register file, 3) a combinational logic, and 4) be demonstrated on the ALTERA boards provided in the lab.*

Notes on the Four Provided Projects

The four projects descriptions outline all necessary details for the projects. If you choose to implement one of these projects, **you must follow** all details listed in the project description. Your implementation, at a minimum, **should satisfy** all implementation features and constraints as specified in this document. This means that if you do not implement a certain feature or reduce the complexity of the project you will lose points as indicated in the grading **RUBRIC**. You are allowed to work beyond the basic functionality as long as the implementation improves upon the specified design. It would be helpful to discuss any such changes with your TA.

Think about the final project as a really long homework. You may have to go to the lab outside of your regularly scheduled lab time in order to complete it. To get a grade for this assignment you need to present your final project to your TA during dead week (in your regular lab time). No presentation, no grade!

Also, there will be ZERO credit for any design step in the grading rubric that does not work or produces the wrong output or is not of the given specifications. All components of your design MUST WORK and produce the output as described in the specs to receive credit.

Project #1: Stack Arithmetic

See the grading RUBRIC below before starting with the design.

For this circuit, you will need to implement a simple finite state machine to control a stack of 4-bit unsigned integers. If you are not familiar with the stack abstract data type, check http://en.wikipedia.org/wiki/Stack_(abstract_data_type) . You have to design this stack so that its contents can be used to perform some simple arithmetic. The user must be able to perform four operations:

- **Push** - A value is added to the top of the stack.
- **Pop** - The top-most value is removed from the stack.
- **Pop with Add** - The top two values on the stack are popped from the stack, added together, and the result is pushed back onto the top of the stack.
- **Pop with Subtract** - The top two values on the stack are popped from the stack, used to perform subtraction, and the result is pushed back onto the top of the stack. (The first value popped is subtracted from the second value popped.)
- **Pop and exchange –** The top two values on the stack are popped and inserted back on the stack in reverse order.

To keep things simple, the maximum stack depth will be four. Your circuit should display the complete contents of the stack on HEX3 through HEX0. To be clear, HEX3 should display the first value pushed onto the stack, HEX2 should display the second value, and so on. The seven-segment display should have no lit LEDs if the stack is empty.

Some notes about error handling:

- **Stack Overflow** - If a push is performed, but there is no more room on the top of the stack, the contents of the stack should not be changed and the *stack overflow register* should be set. Hook this register up to an LED to notify the user. This register and LED should remain set and lit until the board is reset.
- **Stack Underflow** - If any of the three pop operations are performed without enough values on the stack to perform that operation, the contents of the stack should not be changed and the *stack underflow register* should be set. Hook this register up to an LED to notify the user. This register and LED should remain set and lit until the board is reset.

Some hints:

- You may find it convenient to implement the stack using a 4-bit register file with 4 registers, two read ports, and one write port.

- If there is only one value on the stack, only HEX3 should display a value. The other HEX displays should not have any lit LEDs.

RUBRIC

Each of the following steps must be in its **own design file**. Demonstrate each step individually to receive credit.

a. Display complete stack (4-bit wide 4-register file) contents on HEX displays using the **PUSH** operation. You may have to push some random initial values into the register file. **(20 points)**
b. Demonstrate the **POP** operation by popping the values pushed in Part a. **(20 points)**
c. Design and demonstrate **POP with ADD/SUBTRACT/EXCHANGE**. **(20 points)**
d. Put together all of the individual components in Parts a,b, and c and demonstrate the complete stack machine with **error handling**. **(20 points)**
e. **Final Report – 20 points**

Project #2: Sorting Machine

See the grading RUBRIC below before starting with the design.

Design a sorting machine that sorts numbers in either ascending or descending order. The idea is to design a two-port read two-port write register file with k registers. You must choose a value for k in your design and it must be at least 4. The data are stored in registers using some input switches (address and data are specified by switches). Then there are two counters, C1 and C2. A four state machine sorts the numbers using the bubble sort algorithm as follows.

Load the registers with initial values. Start the machine in state S0.

State S0:

C1 is initialized to 0. Go to state S1.

State S1:

C2 is initialized to C1 + 1. Go to State S2.

State S2:

Read the contents of the two registers from the two addresses specified by C1 and C2. Call these values D1 and D2.

Feed D1 and D2 into a maximizer/minimizer circuit, which will yield MAX and MIN on two ports.

At the clock edge, MAX is written into register C1 and MIN is written into register C2. This will swap the values if they were out of order or keep them the same if they were in order.

If C1 = k-2 then
    Go to State S3
Else if C2 = k-1 then
    Increment C1 and Go to State S1
Else
    Increment C2 and Go to State S2

State S3:

Registers are sorted and displayed on 7-segment displays. Done.

RUBRIC

Each of the following steps must be in its **own design file**. Demonstrate each step individually to receive credit.

a. Design and demonstrate that the two-port read two-port write register file with k registers works by writing and reading from the register file. You may have to use 7-seg LED to show the contents of the sorting register file. **(20 points)**
b. Design and demonstrate the maximizer/minimizer circuit**. (20 points)**
c. Design and demonstrate that the state machine works. **(20 points)**
d. Put together all the individual components in Parts a,b, an c and demonstrate that the complete Sorting machine works. **(20 points)**
e. **Final Report (20 points)**

Project #3: Circular Queue Processor

See the grading RUBRIC below before starting with the design.

A circular queue is an abstract storage structure that uses FIFO policy to enqueue and dequeue data. The locations to enqueue and dequeue data are maintained in two pointer indexes – *head* (also known as read) and *tail* (also known as write). Head points to the first non-empty data in the queue. Tail points to the next free location to write in the queue. Data is always removed (dequeued) from the head of the queue. Data is always written (enqueued) into the queue at the tail.

Refer to https://en.wikipedia.org/wiki/Queue_(abstract_data_type) and http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/8-List/array-queue2.html for details of circular queue implementation in software.

The Queue storage structure is implemented using a 4-bit two-read port, one-write port register file of ***K*** registers (***K*** MUST be at least 8 and power of 2 for simplicity).

The circular queue processor should function as follows:

a. Initially the queue is empty.
b. On *enq* input signal a 4-bit data is written into the queue at the location pointed by tail, at the next active clock edge. The tail pointer is adjusted accordingly.
c. On *deq* input signal a 4-bit data is removed from the queue at the location pointed by head, at the next active clock edge. The head pointer is adjusted accordingly.
d. The *enq* and *deq* signals should not be both active in the same clock cycle.
e. A *qprocess* input signal, at an active clock edge, will dequeue the first TWO entries in the queue. These TWO entries are added and the result of the addition is enqueued back into the queue. The Head and Tail pointers are adjusted accordingly.

Error Handling

a. The qprocess signal should not have any effect if there is only one or zero data in the queue.
b. An *error* signal should be active if an attempt is made to dequeue from an empty queue.
c. An *error* signal should be active if an attempt is made to enqueue into a full queue.

Sample Demo

a. Enqueue the 4-bit numbers 1,2,3,4,5 into the queue.
b. Use the qprocess input to add all the numbers. The result should be 15.
c. Dequeue the sum (15) from the queue.

RUBRIC

Each of the following steps must be in its **own design file**. Demonstrate each step individually to receive credit.

a. Design and demonstrate that the two-port read one-port write register file with k registers works, by writing and reading from the register file. You may have to use 7-seg LED to show the contents of the register file. **(20 points)**
b. Design and demonstrate the control FSM for the circular queue**.** **(20 points)**
c. Design and demonstrate the logic to retrieve two numbers from the head of the queue, add them and display the result on LED. **(20 points)**
d. Put together all of the individual components in Parts a,b, and c and demonstrate that the complete queue processor works. **(20 points)**
e. **Final Report – 20 points**

Project #4: Priority queues

See the grading RUBRIC below before starting with the design.

A priority queue is a regular queue structure with a weight (priority) assigned to indicate the relative importance of the elements in the queue among multiple queues.

Refer to https://en.wikipedia.org/wiki/Queue_(abstract_data_type) for more details about queues.

The Queue storage structure is implemented using a 4-bit one-read port, one-write port register file of ***K*** registers (***K*** MUST be at least 8 and power of 2 for simplicity).

This project will use two queues – Q1 (High priority) and Q2(Low priority).

The system of two queues will operate in two modes – independent and cascaded.

Independent mode – Data can be enqueued and dequeued independently in each queue. In this mode, Q1 and Q2 operate as individual queues without any priority order.

Cascaded mode – Data can be independently enqueued in each queue. However, dequeuing imposes the priority order. If Q1 is not empty, data is dequeued ONLY from Q1. If and only if, Q1 is empty, data can be dequeued from Q2. Further, dequeued data from Q1 is enqueued in Q2 in the next clock cycle and, similarly, dequeued data from Q2 is enqueued in Q1 in the next clock cycle. This the cascaded operation mode.

The priority queue should function as follows:

a. Initially Q1 and Q2 are empty.
b. An *enq1(enq2)* input signal will write a 4-bit data into Q1(Q2) at the next active clock edge.
c. A *deq1(deq2)* input signal will remove a 4-bit data from Q1(Q2) at the next active clock edge.
d. enq1/2 and deq1/2 signals should not be both active in the same clock cycle.
e. A *qmode* input signal, at an active clock edge, will determine the operating mode of the priority queue.
f. qmode = 0 for independent mode, qmode = 1 for cascaded mode.

Error Handling

a. The qmode signal should not have any effect if there is zero data in both Q1 and Q2.
b. An *error* signal should be active if an attempt is made to dequeue from an empty queue.
c. An *error* signal should be active if an attempt is made to enqueue into a full queue.

Sample Demo

a. Enqueue two 4-bit numbers 0xA, 0xF into Q1. Set qmode = 0. Show dequeue operation in Q1**.**
b. Enqueue two 4-bit numbers 0xA, 0xF into Q1. Set qmode = 1. Show that both elements move to Q2 while dequeuing from Q1. Both elements should move back to Q1 while dequeuing from Q2.

RUBRIC
Each of the following steps must be in its **own design file**. Demonstrate each step individually to receive credit.

a. Design and demonstrate that the one-port read one-port write register file with k registers works, by writing and reading from the register file. You may have to use 7-seg LED to show the contents of the register file. **(20 points)**
b. Design and demonstrate the control FSM for the priority queue and show that the independent mode works**. (20 points)**
c. Design and demonstrate that the cascaded mode works. **(20 points)**
d. Put together all of the individual components in Parts a,b, and c and demonstrate that the complete priority queue with both modes works. **(20 points)**
e. **Final Report – 20 points**

Project #5: Specify Your Own Project

See the grading RUBRIC below before starting with the design.

Propose a project of your own design. Once again, the minimum constraints are that your project MUST: 1) include a state machine, 2) include a register file, 3) a combinational logic, and 4) be demonstrated on the ALTERA boards provided in the lab.

If you choose this option, you **MUST** discuss your intended design with your lab TA to see if the project would be appropriate for this class. This discussion can be over e-mail. Please contact your lab TA.

If the lab TA or Instructor evaluates the project and recommends that it is not of appropriate difficulty, then the project will be denied and you'll have to modify it or propose a new one.

RUBRIC

Each of the following steps must be in its **own design file**. Demonstrate each step individually to receive credit.

a. Design and demonstrate the state machine for your design **(20 points)**
b. Design and demonstrate the register file for your design **(20 points)**
c. Design and demonstrate the combinational circuit. **(20 points)**
d. Put together all of the individual components in Parts a,b, and c and demonstrate the complete project. **(20 points)**
e. **Final Report – 20 points**