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 three 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 Three Provided Projects

The three 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: Tiny Encryption Algorithm Encoder/Decoder

See the grading RUBRIC below before starting with the design.

Implement an Encryptor/Decryptor for the basic tiny encryption algorithm (TEA). For more information, check here: https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm
However, you will implement TEA as follows:

```
void encrypt (uint8_t* v, uint8_t* k) {
    uint8_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint8_t delta=0xb7;                          /* a key schedule constant */
    uint8_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i < 32; i++) {                       /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint8_t* v, uint8_t* k) {
    uint8_t v0=v[0], v1=v[1], sum=0xe0, i;  /* set up */
    uint8_t delta=0xb7;                     /* a key schedule constant */
    uint8_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

Your TEA will take two 8-bit integers (V0 and V1 – really a 16-bit value) and four 8-bit keys (K0, K1, K2, and K3 – a 32-bit key) and encrypts V0 and V1 in a relatively simple and efficient manner.

This circuit should have a control FSM and a datapath. The datapath should contain a register file, an arithmetic and logical unit (ALU), and some step counters. The register file should have one write port and two read ports and contain ten 8-bit registers (V0, V1, K0, K1, K2, K3, sum, tmp0, tmp1, and tmp2) and an 8-bit ``magic constant'' (0x9e3779b9). The ALU should take two 8-bit operands and support all the necessary operations to perform the mixing step (i.e., several types of shifting, bitwise or, bitwise xor, addition, and subtraction). Use as many counters as necessary to aid in control reading in input values and writing out the output values.

Operation:

0. Load the key: If toggle switch 8 is asserted, you should be able to load K0-K3, one byte at a time by setting switches 0-7 and pushing pushbutton 0.

1. Load the values for encoding/decoding: If toggle switch 8 is not asserted, you should be able to load V0 and V1, one byte at a time by setting switches 0-7 and pushing pushbutton 0.

2. Encoding/decoding: Once the last byte of V1 is entered, encoding or decoding begins (toggle switch 9 determines whether the circuit encodes values or decodes them).

3. Display: Once encoding/decoding is complete, V0 and V1 should be displayed on the 7-segment displays one byte at a time with pushbutton 0 advancing the byte.

Examples:

If you encrypt the number $V=2810_{16}$ with key $K=B3BEEF75_{16}$, the encrypted result (cipher) should be $4C37_{16}$. This result can then be decrypted using $K=B3BEEF75_{16}$ to get the original result of 2810_{16}.

Likewise, if you encrypt the number V=0 with key K=0, the encrypted result should be $86B6_{16}$. This result can then be decrypted (provided the same key K=0 is used) to get the original result of 0.

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 and one-port write register file works by writing and reading from the register file, including reading in the same cycle. **(20 points)**

b. Design and demonstrate the ALU 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, and c and demonstrate that the complete TEA encoding and decoding works. **(20 points)**

e. Final Report **– (20 points)**

Project #2: 8-bit Booth's Multiplier

See the grading RUBRIC below before starting with the design.

As you learned earlier in the semester, multiplication is effectively repeated shifting and addition. This understanding can be used to reduce the area of a hardware multiplier by implementing a sequential multiplication circuit. Unfortunately, this means that for each bit of the multiplier there is both an add and a shift, even for cases like *0 where no adds are necessary. Therefore, you are tasked with designing an 8-bit sequential multiplier based on Booth's algorithm (https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm). Your Booth's algorithm multiplier should have an FSM for controlling the algorithm and a datapath that includes a register file, a modified adder-subtractor circuit (needs selective sign extension), and some single flip-flops to represent P[-1]/carries/signs.

The register file has two read ports, a single write port, and contains just four 8-bit registers (multiplicand--m, lower result--P[7:0], upper result--P[15:8], and count).

Behavior:

Step 1: Initialization

* Load the multiplicand from toggle switches 0-7.
* Load the multiplier into the lower result register from toggle switches 8-15.
* P[-1] is initialized to 0.

Step 2: Multiplication

* Read P lower and check the value of p[0:-1]:

  If 01 add a sign-extension of m to P (this is a multi-step process since P is a 16-bit value).

  If 10 add a sign-extension of -m to P (this is a multi-step process since P is a 16-bit value).

  If 00 or 11 don't add or subtract.
* Right shift P (use an arithmetic shift where P's sign doesn't change).
* Repeat these steps until they have been done 8 times.

Step 3: Display result

* Display the result (i.e., P[15:0]) on the 7-segment display using pushbutton 0 to toggle between the upper and lower half.

Example of Booth's Multiplication of two 5-bit numbers:

Let us multiply 12 by 11. The result should come out to be 00 1000 0100 (132). Let's verify. First, place 01100 in as the multiplier, then fill the receptacle and Prev with 0s.

Receptacle					Multiplier					Prev	Comments (This is 12×11)
P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	P-1	Multiplicand N = 11 = 01011_2 , -N = 10101_2
0	0	0	0	0	0	1	1	0	0	0	P0 = P-1 → only shift to the right.
0	0	0	0	0	0	0	1	1	0	0	P0 = P-1 → only shift to the right.
0	0	0	0	0	0	0	0	1	1	0	[P0 P-1] = 10, so add –N to the receptacle (10101_2)
1	0	1	0	1	0	0	0	1	1	0	Then shift everything to the right.
1	1	0	1	0	1	0	0	0	1	1	P0 = P-1 → only shift to the right.
1	1	1	0	1	0	1	0	0	0	1	[P0 P-1] = 01, so add +M to the receptacle (01011_2) 01011 + 11101 = 01000 (11 – 3 = 8).
0	1	0	0	0	0	1	0	0	0	1	Then shift everything to the right.
0	0	1	0	0	0	0	1	0	0	0	We've shifted five times, so read off the result in Receptacle and Multiplier.
0	0	1	0	0	0	0	1	0	0	0	You can also just compare with the previous result to see that the answer is identical.

Suppose we wanted to multiply 8 and -13. Booth's Multiplication Algorithm can also handle this case with negative numbers to multiply. Let's start with multiplier M as 8 and the multiplicand N = -13. Once again, the table will be setup as follows:

Receptacle					Multiplier					Prev	Comments (This is 8×-13)
P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	P-1	Multiplicand N = -13 = 10011_2 , -N = 01101_2
0	0	0	0	0	0	1	0	0	0	0	P0 = P-1→ only shift right
0	0	0	0	0	0	0	1	0	0	0	P0 = P-1→ only shift right
0	0	0	0	0	0	0	0	1	0	0	P0 = P-1→ only shift right
0	0	0	0	0	0	0	0	0	1	0	[P0 P-1] = 10, so add –N to the receptacle (01101_2)
0	1	1	0	1	0	0	0	0	1	0	Then shift everything to the right.
0	0	1	1	0	1	0	0	0	0	1	[P0 P-1] = 01, so add +N to the receptacle (10011_2) 10011 + 00110 = 11001 (-13 + 6 = -7).
1	1	0	0	1	1	0	0	0	0	1	Then shift everything to the right.
1	1	1	0	0	1	1	0	0	0	0	We've shifted five times, so read off the result in Receptacle and Multiplier.
1	1	1	0	0	1	1	0	0	0	0	You'll notice that the result is negative, as we expected. Specifically, 1110011000, we have a large negative number in 2's complement, from which we can find the magnitude by inverting the bits to the left of the rightmost 1 to get 00 0110 1000_2 (104), which means our result is -104.

On further inspection, you'll notice that the result is merely -13 in binary (10011_2), but shifted three places to the left. Note that if we want to select a negative number such as -13 as the multiplier, the receptacle does **not** take on the sign extension of the multiplier (it may be tempting to fill it in, but this is handled with Booth's algorithm); we fill the receptacle with zeros as we always do.

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 and one-port write register file works by writing and reading from the register file, including reading in the same cycle. **(20 points)**

b. Design and demonstrate the modified adder/subtractor 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, and c and demonstrate that the complete multiplier works. **(20 points)**

e. Final Report. **(20 points)**

Project #3: Traffic Light Fairness System

See the grading RUBRIC below before starting with the design.

In this project, you will design a 4-way intersection management and traffic control system. The control system will function as follows: there will be four lanes of traffic, each with a capacity limit on the number of cars at the intersection. The number of cars will be maintained with four specialized counters. The traffic light system will greenlight one of the four lanes of traffic, which will decrement the number of cars at the intersection by one for each clock cycle. The traffic will be provided to one of the four lanes as indicated by the position of four toggle switches. One car will be added to the specified lane for each clock cycle. If the addition of a new vehicle will exceed the vehicle capacity for this lane, then the vehicle will not be added. Again, the vehicle count will be maintained by a specialized counter for each lane.

Details:
Your program will start off by initializing the capacity of each of the four lanes, one by one. This will be done by specifying a **non-zero** capacity for each lane and storing the number into a register file. The register file will contain four 4-bit registers with one input port and one output port. If the user specifies a zero capacity, your circuit should reject this capacity and instead prompt again for a **non-zero** capacity.
Once all capacities have been loaded into the register, traffic management and control can begin. There are two modes that your system will operate in:
Mode A (No Traffic Can Pass): In this mode, all lanes at the intersection will receive a red light, meaning that no lane of traffic will allow any cars through. Traffic can be added to the lanes at the intersection, but the number of cars in each lane will not decrease as all lanes have red lights. Traffic will be added to each intersection by one of four toggle switches. Each switch, when set to one, can add one car to its lane for each clock cycle. However, you will implement priority on these lanes such that, if traffic is requested to be added to two lanes, you should prioritize one lane over the other. The choice of lane priority is up to you, but should be specified. (Example: if lane 2 and 3 have their toggle switches set to 1, you will, for instance, only add one car to lane 3 for each clock cycle while lane 3's toggle switch is at 1).
In addition, as cars can be added to one of the lanes at a time, your system should also avoid exceeding the lane capacity for the lane to which you are adding cars. If the lane capacity is set to 9 for a given lane, then the counter for the number of cars in the lane should not exceed 9. The counter is allowed to equal the capacity, but it cannot be greater than the capacity at any time. The capacity is stored in the register file, so your circuit must read the appropriate capacity and use it to prevent the appropriate counter from exceeding its capacity.
Mode B (Traffic Can Pass from One Lane Only): In this mode, your system will greenlight one of the traffic lanes. While a particular lane has a green light, the number of cars in this lane will decrement by one. This vehicle is considered to have passed through the intersection. Your traffic light will cycle to

other lanes that have cars and reduce the number of cars in lanes, as appropriate. The means of greenlighting the lanes of traffic is up to you, with the following restrictions:

1. The traffic light should not be green for one lane for more than five consecutive clock cycles when traffic exists in another lane. This is to prevent your traffic light from only servicing one lane of traffic over other lanes. Correspondingly, if traffic only exists in one lane, then your traffic light can remain green for this one lane. It should not remain green for one lane indefinitely if even a single car exists in another lane.
2. The traffic light should not be green for a lane with no cars in it when traffic exists in another lane. This is to prevent your traffic light from cycling blindly through each lane. Your light must assess the values of the counter (the number of cars in each lane) to determine if a lane has cars in it and should be skipped.

In addition, traffic in Mode B can be added to each lane the same way that it was in Mode A. Again, the number of cars in each lane should not exceed the lane's capacity. This means that traffic can be added to a lane that has a greenlight, which will have a net effect that leaves the number of cars in this lane unchanged. The restrictions for the traffic light still apply, and the lane can only have a greenlight for as many as five consecutive cycles while traffic exists in another lane.

Components:

- Four input toggle switches for adding traffic to each lane. These will, when set to one, add one car to the appropriate lane for every clock cycle. If more than one of these toggle switches are one, a lane must be prioritized from the four lanes selected to add a car for each cycle.
- A register file with four 4-bit registers to hold the capacity for each of the four lanes. The register file will have one write port and one read port to store and load the lane capacity.
- Four counters which will store the number of cars actually waiting in each of the four lanes. The counter should increment if indicated by the traffic input switch and should decrement if the lane has a green light. If both are present or if neither are present, then the counter value should remain unchanged. The counter also should not increment beyond the capacity for the lane to which it is assigned. The counter should also not decrement while at zero.
- Circuitry that will ensure the counters do not exceed their capacity.
- Circuity that will prioritize one of the four toggle switches for inputting traffic into each lane.
- A state machine that will store the capacities in the register file one by one, then provide green lights for a lane (if in Mode B) or prevent traffic from passing through the intersection (if in Mode A) in accordance with the previously specified rules.

RUBRIC

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

a. Demonstrate a functional register file. **(20 points)**
b. Demonstrate functional counters that will increment, decrement, and operate within the specified bounds. **(20 points)**
c. Demonstrate a functional FSM that will operate the components specified above in accordance with the restrictions and requirements specified above. **(20 points)**
d. Demonstrate a working circuit that meets all of the specified criteria. **(20 points)**
e. Final Report. **(20 points)**

Project #4: 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)**