ECE 270 Lab Verification / Evaluation Form

Experiment 12

Evaluation:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated to and verified by your lab instructor *before the end* of your scheduled lab period.

STEP	DESCRIPTION	MAX	SCORE
Pre-lab 1	Block diagram of circuit detailing interconnections	4	
Step 1	Port scrolling message display module from Lab 10	2	
Step 2	Port linear feedback shift register module from Lab 11	2	
Step 3	4-bit radix adder/subtractor module with condition codes	2	
Step 4	4-bit radix magnitude comparator module	2	
Step 5	4-bit round counter module	2	
Step 6	BCD full-adder module	2	
Step 7	2-digit BCD score adder/accumulator module	3	
Step 8	Win/lose detector module with programmable threshold	1	
Step 9	Functioning game based on “wiring” modules together	1	
Step 10	Thought questions	4	
	TOTAL	25	

Signature of Evaluator: _____

Academic Honesty Statement:

“In signing this statement, I hereby certify that the work on this experiment is my own and that I have not copied the work of any other student (past or present) while completing this experiment. I understand that if I fail to honor this agreement, I will receive a score of ZERO for this experiment and be subject to possible disciplinary action.”

Printed Name: _____ Class No. __ __ __ __ - __

Signature: _____ Date: _____

The "Non-Elite Academic Exercise" Game
-or-
The Radix Price is Right!

Instructional Objectives:
- To practice creating a variety of arithmetic circuits

Prelab Preparation:
- Read this document in its entirety
- Review the referenced Module 4 lecture material

Experiment Description:
In this experiment you will create what is indisputably an "academic exercise" with little social significance to the "cultural elites" who sit in the back row and text on their smart phones and/or check social media on their laptops during class. It *may*, however, provide a unique opportunity to impress your friends from other majors.

Imagine you are creating a new TV game show called *The Radix Price is Right!*, in which a digitally-astute contestant attempts to guess the price of *really cheap,* perhaps even *worthless* "stuff" (like first generation iPhones or Green New Deal posters). The added twist is that some of the stuff is *so* cheap that someone would have to *pay* you to take it (i.e. the price is a *negative number*). Like the "real" TV game show upon which this non-elite academic exercise is loosely fashioned, each time the contestant guesses the (randomly generated) "price" *exactly* they will earn 9 points. If the guess is "low" the contestant will earn 4 points, but if the guess is "high" the contestant will earn *zero* points. The contestant's point total will be displayed as a two-digit BCD number (on `DIS4` and `DIS3`), while the round number (starting at 1) will be displayed as a single BCD digit (on `DIS1`). A 4-bit linear feedback shift register (LFSR) will be used to generate the "radix (*two's complement*) price" for each round: here, the **radix price** will range from **-8_{10} (1000_2)** to **$+7_{10}$ (0111_2)** dollars. The contestant will enter their guess using `DIP[3:0]`. Each guess will be "clocked in" using the right pushbutton (S1BC), which will update the contestant's score, advance the round number, and advance the LFSR to its next state. After the guess for the 9th round is clocked in, the contestant's score followed by the string "**`yEAH`**" or "**`LoSEr`**" will continuously scroll across the 7-segment displays, based on whether a point total of the ***specified threshold*** (given in the `lab12_top_template.v` file) was earned. For debugging purposes, `DIP[7]` will be used to either "hide" the LFSR state and comparator results (shown on the jumbo LEDs) or display them, enabling one to play in "cheat" mode. The left pushbutton (S2BC) will be used to asynchronously reset the game.

Pre-lab Step (1):
Referring to the lab12_top_template.v file provided for this experiment on the course website, sketch a **block diagram** of the entire circuit on the page that follows, showing the interconnections among the various modules as well as all the inputs (DIP switches and pushbuttons) and outputs (LEDs).

Block Diagram:

Step (1):

Carefully examine the `lab12_top_template.v` file provided for this experiment on the course website. Rename this file *myuserid*`_lab12.v` (where *myuserid* is your 8-character login name) and insert your identifying information where indicated. Next, open ispLever and create a new Verilog project named `Lab12`, select `LC4256ZE-5TN144C` as the device to use, and import your *myuserid*`_lab12.v` source file. Next, import the "scrolling message display" Verilog modules (`msggen.v` and `dispshift.v`) you wrote for Lab 10 into your *myuserid*`_lab12.v` source file. Modify the `msggen.v` state machine so that `mSEL` is a one-bit value (instead of two), and include an enable (EN) signal. The characters denoted "#" are the contestant's final score (at the conclusion of the game), which gets scrolled along with the win/lose message string. Test your modified scrolling message display as outlined in the top template file. The two message strings should be as follows:

mSEL	*Scrolling Message Displayed*
0	*blank* → # → # → *blank* → L → o → S → E → r → *blank* → *blank* → …
1	*blank* → # → # → *blank* → y → E → A → H → *blank* → *blank* → …

Step (2):

Import the linear feedback shift register Verilog module you wrote for Lab 11 and modify it so that it is 4-bits wide. Use the feedback described on page 739 (4th Ed.) or p. 575 (5th Ed.) that causes the LFSR to cycle through all 16 possible states (including `0000`). Use the right pushbutton (S1BC) to clock the LFSR and the left pushbutton (S2BC) to provide it with an *asynchronous reset*. Route the LFSR outputs to `MIDRED[3:0]`, and use `DIP[7]` to control whether the LFSR state is "hidden" or displayed. Verify that the LFSR visits all 16 possible states in pseudo-random order.

Checkpoint: Demonstrate Steps 1 and 2 to your Lab Instructor.

Step (3):

Create a 4-bit adder/subtractor Verilog module that generates CF, NF, ZF, VF condition codes. Signed 4-bit operand `X[3:0]` will be entered using `DIP[7:4]`, while `Y[3:0]` will be entered using `DIP[3:0]`. The add/subtract mode will be controlled by the left pushbutton (S2BC): "not pressed" for add, "pressed" for subtract. Route the 4-bit sum/difference to `TOPRED[3:0]`, and route the CF, ZF, NF, and VF condition codes to `TOPRED[7:4]`, respectively. Verify adder/subtractor and condition code functionality using the sample test cases shown below (plus any other test cases you deem necessary):

```
  1 0 1 1
+ 1 0 1 0
-----------
1 0 1 0 1
```

C=1, Z=0, N=0, V=1

```
  0 0 0 1
+ 1 1 1 1
-----------
1 0 0 0 0
```

C=1, Z=1, N=0, V=0

```
  1 0 0 1
- 1 0 1 1
-----------
```

```
  1 0 0 1
  0 1 0 0
+ 0 0 0 1
-----------
  1 1 1 0
```

C=0, Z=0, N=1, V=0

Step (4):

Create a magnitude comparator Verilog module that determines if X > Y, X = Y, or X < Y based on the condition codes generated by the 4-bit adder/subtractor module realized in Step 3. For the game to be implemented in this experiment, the adder/subtractor module will be set for subtract mode, X will be the (LFSR-generated) price, and Y will be the contestant's guess. For testing purposes, if X > Y, then the jumbo yellow LED should be illuminated (indicating that the contestant's guess was too low); if X = Y, the jumbo green LED should be illuminated (guess was correct); and if X < Y, the jumbo red LED should be illuminated (guess was too high). Your magnitude comparator module should also generate the "points earned" based on each condition: 4 points if the guess was too low, 9 points if the guess was exactly right, and 0 points if the guess was too high. Verify that the magnitude comparator works as expected for various price/guess combinations (for testing purposes, values for X and Y will be entered on DIP switches, as detailed in the `lab12_top_template.v` file). *Recall that X and Y are signed 4-bit numbers.*

Checkpoint: Demonstrate Step 4 to your Lab Instructor.

Step (5):

Create a 4-bit binary "round counter" Verilog module that initializes to state `0001` (upon game reset) and increments by one each time it is clocked until it reaches state `1010`, at which point it "freezes" (i.e. advances no further). Use the left pushbutton (S2BC) to asynchronously reset the counter to its initial state, and the right pushbutton (S1BC) to clock it. Use an instantiation of the `bcd2dis` module (provided) to convert the round counter output to display code; route the output to `DIS1`. The round counter should also generate a "game over" signal that can be used to select the 7-segment display mode. Verify that the "round counter" works as expected.

Step (6):

Create a BCD full-adder Verilog module based on the design presented in the *Lecture Summary* notes (two instantiations of this module will be used to realize the 2-digit BCD adder module in Step 7). Verify that your BCD full adder works as expected for a representative set of operand combinations (for testing purposes, the two BCD operands will be entered on DIP switches, and CIN will be set to "1").

Checkpoint: Demonstrate Steps 5 and 6 to your Lab Instructor.

Step (7):

Create a two-digit BCD adder Verilog module that will successively add the points earned by the contestant in each round to their "running total" (stored in an 8-bit register). The points earned on each round are supplied by the magnitude comparator module, described in Step 4. When the game is reset, the point total should be cleared to `00`; use the left pushbutton (S2BC) to asynchronously reset the running total register to zero. Use instantiations of the `bcd2dis` module (provided) to convert the two "running total" BCD digits to display code; route the outputs to 7-segment displays `DIS4` and `DIS3`. Note that the *maximum* point total possible is **81**, based on making exact guesses in each of the 9 rounds. Verify that the point total display updates on each round as expected (and "freezes" once round 9 is completed).

Step (8):

Create a Verilog module that determines "win/lose" status based on the "winning threshold" specified in its instantiation. Its only job is to output a single bit that can be used to determine which message string to scroll once the game is over. Note that this module simply performs an *unsigned* comparison between the "fixed" (2-digit BCD) threshold indicated in the `lab12_top_template.v` file and (2-digit BCD) `myscore` earned by the contestant. Test it by following the instructions in the `lab12_top_template.v` file, and verify it works as expected.

Step (9):

You are now ready to "wire" together all the modules previously tested to create a functioning game that you can play and test your prescient skills. Verify that the final point total followed by the appropriate string is continuously scrolled on the 7-segment displays once the game is completed. Note: As a quick test, if the guess is set to `0000` for all 9 rounds, the point total should be **29**. With "cheat" mode enabled, verify your ability to both "win" and "lose" *The Radix Price is Right!*

Checkpoint: Demonstrate Step 9 to your Lab Instructor.

Step (10): Write your answers to the following Thought Questions in the space provided**.**

1. Examine the fitter report generated by ispLever and determine the total number of flip-flops utilized by your design as well as the total number of P-terms and macrocells.

   Number of flip-flops: _____

   Number of P-terms: _____

   Number of macrocells: _____

2. Write down the sequence of states (as signed decimal numbers) generated by the LFSR (these are the "prices" the contestant is attempting to guess on each round):

   _____

   _____

3. Try playing the game in "non-cheat" mode (i.e. with the LFSR state and comparator display disabled) and determine your maximum score. Write down your series of guesses.

   _____

4. Empirically determine the maximum score one can obtain with a single "fixed" guess (*i.e.* same guess used for each round).

   _____

   _____

