Lab 2: A Math Game

Due Friday, May 3 2019, 11:59 PM

Minimum Submission Requirements

- Ensure that your Lab2 folder contains the following files (note the capitalization convention):
    - Lab2.lgi (you may need to rename your extension from .LGI to .lgi)
    - README.txt
- Commit and push your repository

Lab Objective

The objective of this lab is to build complex combinational and sequential logic circuits.

In this lab you will implement a math game. Two random 4-bit 2SC numbers will be generated, and the user must guess what the addition or subtraction of those two numbers are.

Your design must:

1. Determine if the user entered the correct value
2. Keep track of the score
3. Modify the score according to the rules.

Game Play

1. To begin the game, reset the score to 0 by pressing the “Reset Score” button.
2. Next, generate two new numbers by pressing the “New Problem” button. The two random numbers will be displayed on two 7-segment LEDs, “Random #0” and “Random #1”.
3. The user should choose whether to add or subtract the random numbers using the “Function” switch. When the switch is low, the random numbers should be added together (r0+r1), and when it’s high, the numbers should be subtracted (r0-r1).
4. Lastly, the user should make a guess that corresponds to the result of the computation from step 3 using the keypad and press “Update Score”. If there is an overflow of the addition or subtraction, the LED labeled “Overflow” should turn on.

Scorekeeping

The score will be adjusted as such:

+1 point: user successfully guesses result
-1 point: user guesses incorrect result

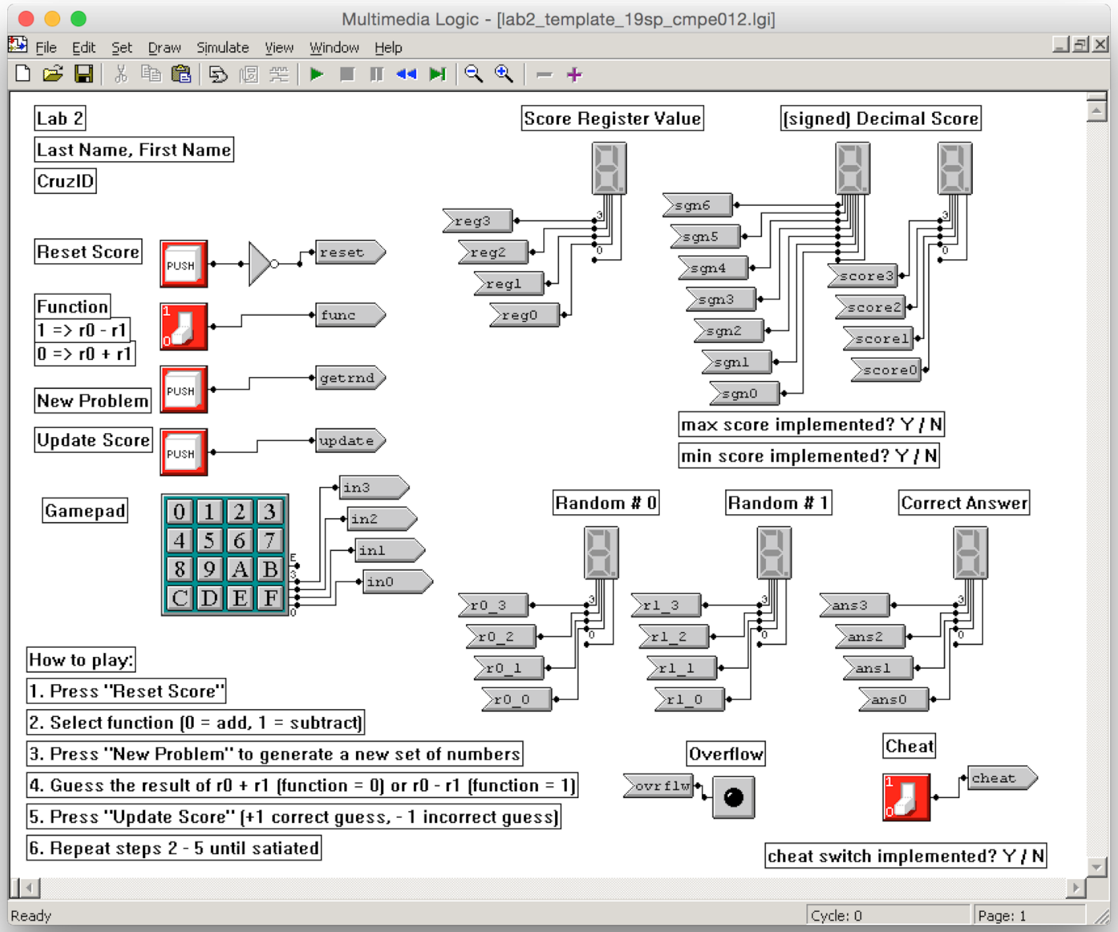
The score will be stored in a 4 bit register. This running score will be kept track of by your circuit and will allow for both negative and positive score values in the range -8 to 7 (4-bit two's complement).

Specification

Template

Build your lab starting with the template file provided on Canvas. The template file contains the user interface (page 1) and logic to generate the random numbers (page 2). **DO NOT MODIFY THE FIRST PAGE** except for your name, CruzID, and placeholder comments indicating if you implemented the extra credit. Do not modify the random number circuits on the second page.

Replace "Y/N" with Y if you implemented the extra credit option, and N if you didn't. There are placeholder signal senders and receivers on the second page that you should use. You may remove these senders and receivers from the second page as you use them in your design. Additional wires and logic circuits shall be drawn on subsequent pages. **Remember to rename the template file** to Lab2.lgi before committing to your repository.



max score implemented? Y / N

min score implemented? Y / N

cheat switch implemented? Y / N

How to play:

1. Press "Reset Score"
2. Select function (0 = add, 1 = subtract)
3. Press "New Problem" to generate a new set of numbers
4. Guess the result of r0 + r1 (function = 0) or r0 - r1 (function = 1)
5. Press "Update Score" (+1 correct guess, - 1 incorrect guess)
6. Repeat steps 2 - 5 until satiated

Figure: Lab 2 Template

User Interface

Inputs

Reset Score	Resets the register and score to 0, normally off
Function	Used to select between addition (switch = 0) and subtraction (switch = 1)
New Problem	Gives a new set of random numbers, normally off
Update Score	Updates the register and score, normally off
Gamepad	Used to make a guess of the resulting computation
Cheat	Allows user to hide the correct answer to make it a game (extra credit) 0 = don't cheat => make "correct answer" display "0" 1 = cheat => display correct answer

Outputs

Register Value	Raw value from the register storing the score, displayed as 4-bit two's complement
Signed Decimal Score	Human readable score, i.e. two's complement score converted to a sign and magnitude. E.g. if register value is F (1111 as binary or -1), the sgn display would have the middle segment illuminated to show "-" and the magnitude would display "1". (Hint: If the number is negative, display the additive inverse and a negative sign.)
Random #0	4-bit two's complement number Range of possible values: -8 to 7. If function = 0, Random #0 will be added to Random #1 If function = 1, Random #1 will be subtracted from Random #0 Note: If F is displayed this is equivalent to -1
Random #1	4-bit two's complement number Range of possible values: -8 to 7. If function = 0, Random #1 will be added to Random #0 If function = 1, Random #1 will be subtracted from Random #0
Correct Answer	The 4-bit two's complement result of the addition or subtraction of Random #0 and Random #1
Overflow LED	Light signifying there was an overflow in the computation

Design Layout

The blank pages of the template are labeled with suggestions for the components that should be drawn on that page, e.g. “Score Register,” and “Display Logic.” You may use as many pages as needed to keep your circuits looking legible. For ease of grading, please follow this order of components:

1 - Function logic	Used to implement the correct operation (add or subtract)
2 - Random Number Adder	Used to add (or subtract) two numbers together
3 - Overflow Logic	Determines if computation overflowed
4 - 4-bit Comparator	Compares user guess to correct answer
5 - Multiplexors	Used to select between +1 and -1 as a score modifier
6 - Score Adder	Modifies score
7 - Score Register	Stores game score
8 - Display Logic	Converts game score to human readable sign and magnitude decimal score

Top Level

Here is a top-level block diagram of the game.

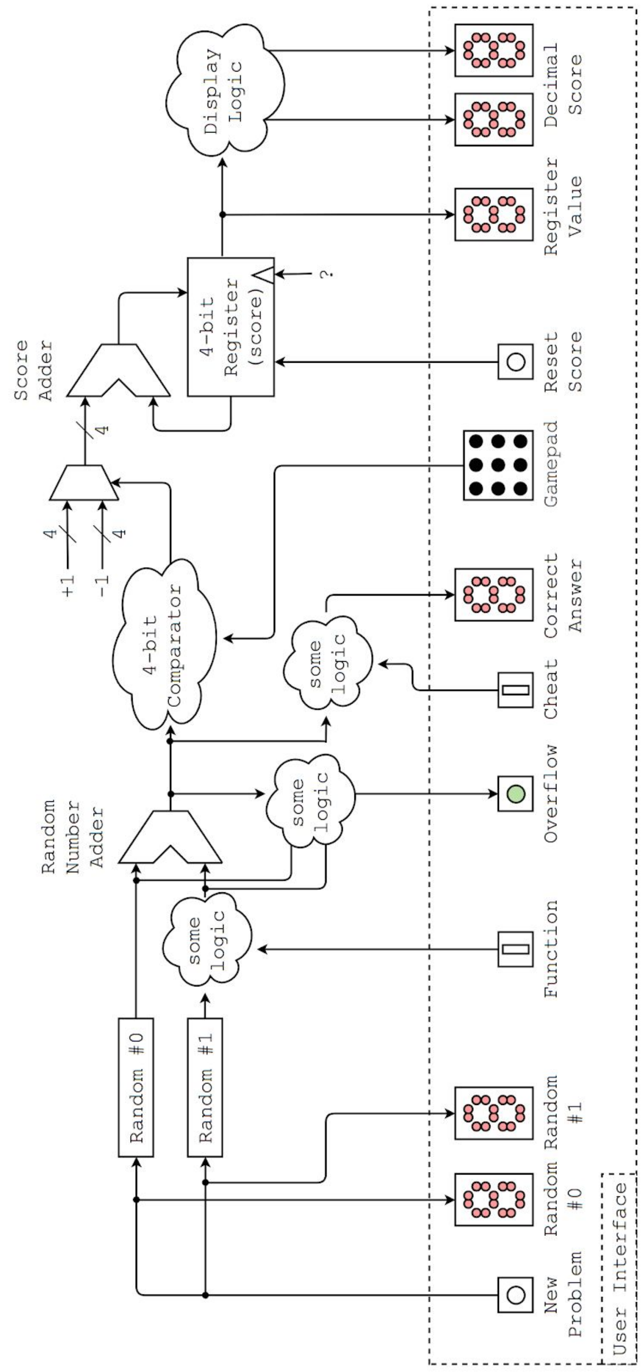


Figure: Top Level Diagram

Overflow

Overflow (or underflow) occurs when the outcome of an arithmetic operation is incorrect due to the limitation of how the value is stored. In this game, if overflow or underflow occurs in the computation of the two random numbers, The Overflow LED should light up.

For example, the 4-bit 2SC computation +7 + +1 = +8 (01000), would cause overflow. The register can only store the least significant 4 bits (1000). In 4-bit 2SC, 1000 is -8.

This can also happen for negative values: -8 + -1 = -9 (10111), but with only 4-bits to represent this value, it appears as +7 (0111).

README.txt

This file must be a plain text (.txt) file. For full credit, it should contain your first and last name (as it appears on Canvas) and your CruzID. Your answers to the questions should total at least 150 words. Your README should adhere to the following template:

```
------------------------
Lab 2: A Math Game
CMPE 012 Spring 2019

Last Name, First Name
CruzID
-------------------------

What did you learn in this lab?
Write the answer here.

What worked well? Did you encounter any issues?
Write the answer here.

How would you redesign this lab to make it better?
Write the answer here.

What external resources did you use to complete this lab?
(Not including course materials)
Write the answer here.

Did you work with anyone on the labs? Describe the level of collaboration.
Write the answer here.
```

Extra Credit

There are three extra credit options for this lab:

1. Implement a cheat switch
2. Add functionality that prevents the score from overflowing (going over +7)
3. Add functionality that prevents the score from underflowing (going beyond -8)

Extra Credit Option 1: Cheat Switch

Connect the cheat switch to control the output of the Correct Answer display. If the cheat switch is on, the correct answer should be displayed. If the cheat switch is off, the Correct Answer display should show a "0"

Extra Credit Options 2 and 3: Max and Min Score

With the current implementation, the score will not stop at the upper score boundary of +7, or the lower score boundary of -8. For instance, if the score is currently +7, and the score is incremented by 1, the result overflows to 0b1000 which is -8 in 4-bit 2SC.

For extra credit option 2, add functionality to your circuit that keeps the score from overflowing.

For extra credit option 3, add functionality to your circuit that keeps the score from underflowing.

Important Stuff

For the register, use D flip-flops, and make sure they are edge triggered with a clear line. You may NOT use the counter, mux or ALU objects provided in MML.

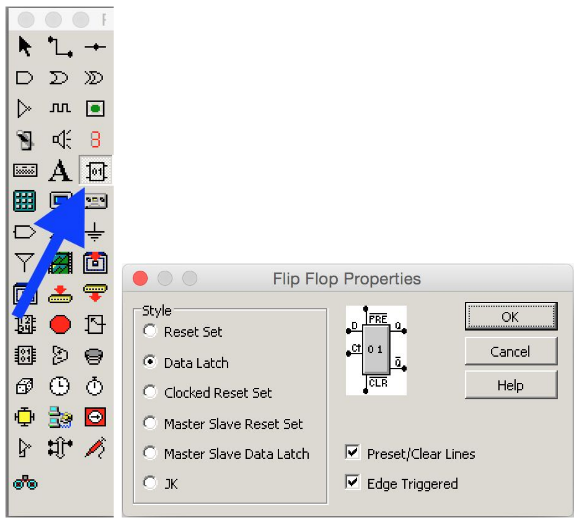
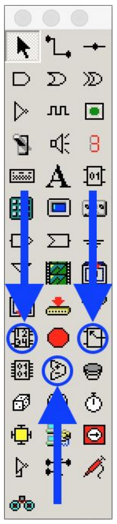
USE	DON'T USE
 Flip Flop Properties Style Reset Set [✓ Data Latch] Clocked Reset Set Master Slave Reset Set Master Slave Data Latch JK [✓ Preset/Clear Lines] [✓ Edge Triggered] OK Cancel Help	

Figure 2: MML Palette

For your convenience, [here]() is a tabular description of how flip-flop with clear line works. You may also find practice_flipflop.lgi handy for understanding how flip-flops set and reset.

Examples of how to hook up the D flip-flop is included in practice_flipflop.lgi.

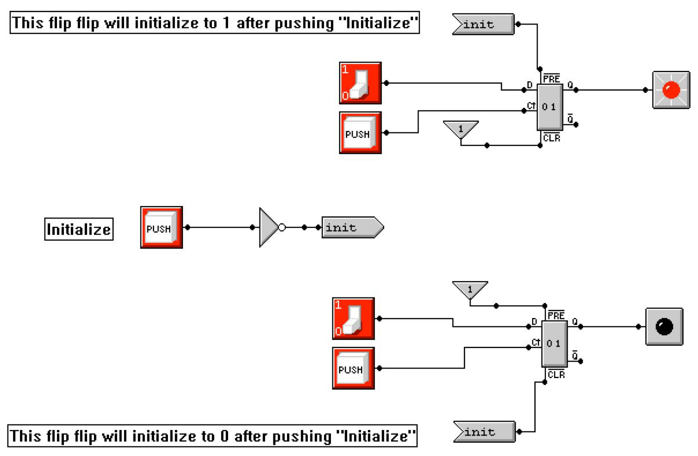

This flip flip will initialize to 1 after pushing "Initialize"
init
PUSH
Initialize
PUSH
init
PUSH
This flip flip will initialize to 0 after pushing "Initialize"
init

Figure 4: Flip-flop Usage Example

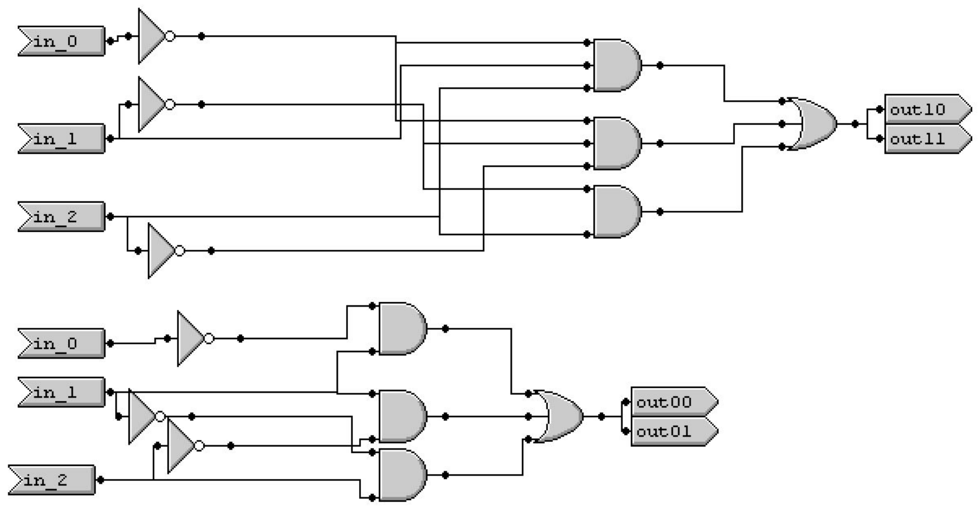
Comments

Each page should be labeled with your last name, first name, and CruzID (the name used in your UCSC email address). Label each circuit with a description of the functionality and the part of the lab that they are for.

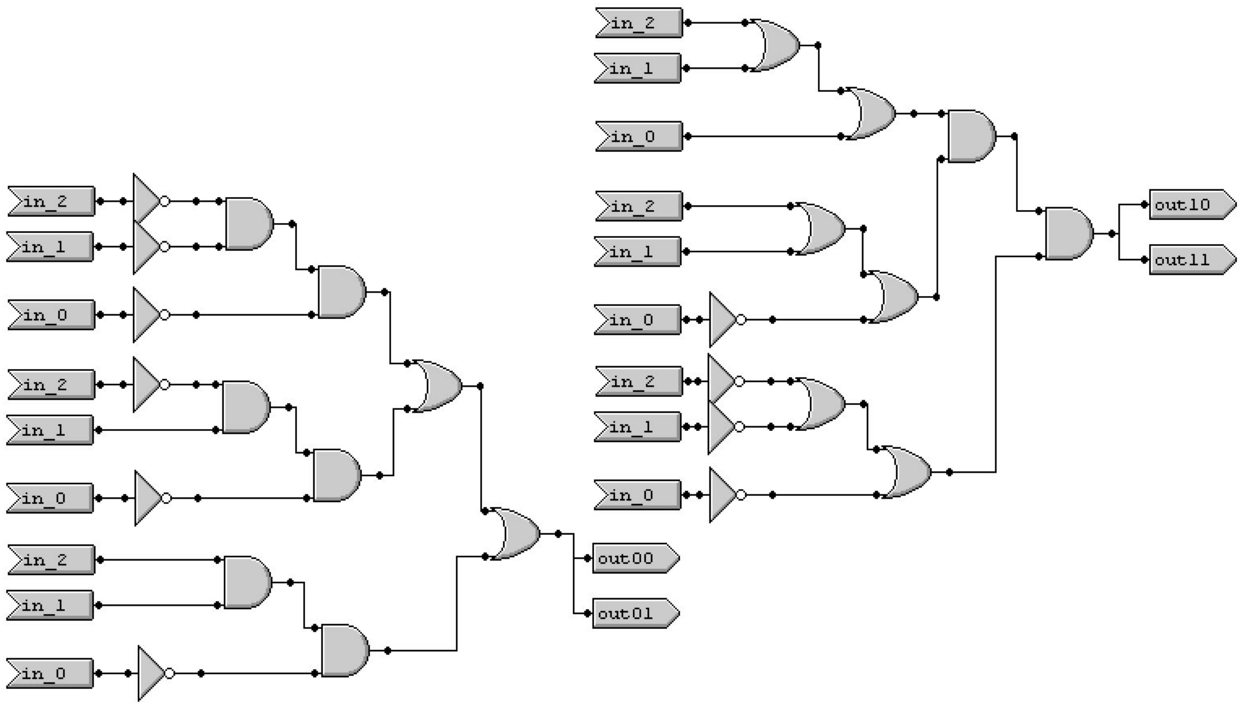
Visual Structure

Presentation of information is an important part of deliverables. Clean documentation is easy to comprehend and looks professional. Your circuits should be structured in an organized method that is easy to read and interpret. Using the "Snap to Grid" setting under the View menu makes it easy to line up components. A clean circuit uses many senders and receivers with meaningful names, and has no wires crossing over each other. Note that there may be multiple receivers for one sender. See below for examples of messy and clean circuits.

Messy Circuit Example



Clean Circuit Example



Missing Wire Best Practices

MML has a known bug which causes some wires to disappear after reopening the file. To reduce the likelihood of this occurring, DO NOT use the “Node” tool (it’s a tiny black dot located at the top-right of the tool palette). This tool is particularly vulnerable to the bug.

If this bug occurs, the grader will attempt to repair the missing wire in your file. This is only possible if your circuit is very readable. Make sure that wires do not cross whenever possible. Wire paths should be short and direct. Use receivers very liberally.

Grading Rubric

TBD