CS 21 Project 1: Bulls and Cows Circuit Simulation

Department of Computer Science
College of Engineering
University of the Philippines - Diliman

1 Introduction

For this project, you are to create a *Logisim* circuit simulation of the *Bulls and Cows* game.

1.1 Bulls and Cows

Bulls and Cows is a logic game wherein the player has to guess a predetermined number in at most some number of attempts. For each guess made, the player is given the number of *bulls* and *cows* of the guess as a clue.

A digit of the guess can be classified as being a bull, a cow, or neither. A *bull* is a digit of the guess G that is present in the answer A and is found in the same position in both G and A. A *cow* is a digit of G found in A but is found in different positions in G and A.

As a four-digit example, $A = 1234$ and $G = 2034$ has 2 bulls (digits 3 and 4 of G are in the right positions) and 1 cow (digit 2 of G is in the wrong position)—note that the digit 0 of G is not in A, so it is neither a bull nor a cow.

1.2 Logisim

Logisim[1] is a free and open-source digital logic circuit simulator. The simulator is written in Java (which allows it to be cross-platform) and produces OS-agnostic output files (`.circ`).

2 Project Specifications

Create a Logisim circuit simulation of the Bulls and Cows game implemented as described in the subsections below.

2.1 Circuit labels

For the **main circuit**, text corresponding to the names and class numbers of each group member must be visible (and the group name if it exists).

Define C to be a set $C_0, C_1, \ldots, C_{G-1}$ containing the class numbers of each of the G group members sorted in ascending order (i.e., C_0 is the smallest class number in the

[1] `http://www.cburch.com/logisim/`

group). N is defined as follows:

$$N = \sum_{i=0}^{G-1} 51^i C_i$$

.

For **all** other user-defined **subcircuits**, a small textbox with content equal to N must be present in the lower-right-hand corner when at **50% zoom level**.

A 5% deduction will be incurred for noncompliance or incorrect execution of any of the above.

2.2 Input/output components

All input and output components must be in the *main circuit*—**the circuit is not valid otherwise and will not be eligible for checking**.

2.2.1 Input components

The circuit must have at least the following input components:

- Numpad (one button for each number from `0` to `9` labeled accordingly)
- Left/Up button (labeled `<` or ∧)
- Right/Down button (labeled `>` or `v`)
- Enter button (labeled `x`)

2.2.2 Output components

The circuit must have at least the following output components:

- 7-segment LEDs (8 in total)
  - Displayed four-digit number; one for each digit
  - `Bulls` (labeled)
  - `Cows` (labeled)
  - `Lives` (labeled)
  - `Wins` (labeled)
- LEDs (9 in total)
  - Digit selector for four-digit number; one for each digit (position beside each digit accordingly)
  - `Start` LED (labeled)
  - `Game` LED (labeled)
  - `Wait` LED (labeled)
  - `Win` LED (labeled)
  - `Lose` LED (labeled)

2.3 Phases

The circuit should have the following phases with behavior defined in the succeeding subsections:

1. Setup Phase
2. Wait Phase 1
3. Wait Phase 2
4. Guess Phase
5. Compute Phase
6. Done Phase
7. Hold Phase
8. Reset Phase

2.3.1 Setup Phase

During this phase, the user can enter the four-digit answer. Only one digit may be selected at any given time; this is shown using one of the four indicator LEDs beside each 7-segment LED representing the digits. Pressing the directional buttons while on a triggering edge changes the selected digit exactly once; the selector LEDs must simultaneously reflect the change as well. Only the selected digit must be modified by a numpad key press.

Once the digits of the intended answer are set, the `Enter` button may be continuously pressed until the next triggering edge to transition into **Wait Phase 1**.

Note that among the LEDs, only the selector LEDs and the `Start` LED are active. For the 7-segment LEDs, only the `Bulls`, `Cows`, and `Lives` are inactive.

2.3.2 Wait Phase 1

During this phase, all display components are inactive except the `Wins` display and both the `Wait` and `Start` LEDs. A transition into **Wait Phase 2** occurs automatically after the next triggering edge.

2.3.3 Wait Phase 2

During this phase, all display components except both the `Wins` display and the `Setup` LED (note that the `Wait` LED is inactive as well). A transition into the **Guess Phase** occurs automatically after the next triggering edge.

2.3.4 Guess Phase

During this phase, the player can input digits in a manner similar to how input is done during the **Setup Phase** (this implies that the selection LEDs and relevant 7-segment LEDs are active). The `Game` LED along with the `Bulls`, `Cows`, `Lives`, and `Wins` 7-segment LEDs must all be active.

Once the digits of the intended guess are set, the `Enter` button may be continuously pressed until the next triggering edge to transition into the **Compute Phase**.

2.3.5 Compute Phase

During this phase, all display components are inactive except the `Wins` display and both the `Game` and `Wait` LEDs.

If the player has either guessed the number correctly (win) or has **zero** lives left after an incorrect guess (lose), a transition into the **Done Phase** occurs automatically after the next triggering edge. Otherwise, an automatic transition back into the **Guess Phase** occurs during the said edge.

Note that once the transition into the **Guess Phase** is complete, the value shown in the `Lives` display must have been decremented by one to reflect that an attempt has been made. The `Bulls` and `Cows` display should be updated according to their definitions.

2.3.6 Done Phase

During this phase, only the `Lives`, `Wins`, and four-digit number displays are active along with either the `Win` or `Lose` LEDs reflecting proper game state.

Note that the four-digit display must show the correct answer. The `Lives` display should show its previous value decremented by one to reflect that an attempt has been made. The value shown in the `Wins` display must be incremented by one if the player has won during this round.

The `Enter` button may be continuously pressed until the next triggering edge to transition into the **Hold Phase**.

2.3.7 Hold Phase

During this phase, all display component states are the same as with the **Done Phase** with the exception of the `Wait` LED being active as well.

If the `Enter` button is pressed during the next triggering edge, a transition into the **Reset Phase** occurs. Otherwise, a transition back into the **Done Phase** occurs. In practice, this means that the `Enter` button must be held for **two** triggering edges to transition from the **Done Phase** into the **Reset Phase**.

2.3.8 Reset Phase

During this phase, only the `Lives` display is active. An automatic transition back into the **Setup Phase** occurs after the next triggering edge. This phase serves as a pause which separates the current round from the next.

2.4 Input/output behavior

2.4.1 Input behavior

- All buttons are expected to be pressed for at least one clock cycle
- Buttons must be activated **exactly once** per user-defined triggering edge
- Holding a button continuously for two clock cycles must register as two button presses
- Only one button is pressed at any given point in time

	S	W1	W2	G	C	D	H	R
4-digit number	✓			✓		✓	✓	
`Bulls`				✓				
`Cows`				✓				
`Lives`				✓		✓	✓	
`Wins`	✓	✓	✓	✓	✓	✓	✓	✓
Digit selectors	✓*			✓*				
`Start` LED	✓	✓	✓					
`Game` LED				✓	✓			
`Wait` LED		✓			✓		✓	
`Win` LED						✓*	✓*	
`Lose` LED						✓*	✓*	

Table 1: I/O active state matrix (* indicates certain conditions have to be met)

2.4.2 Output behavior

A matrix showing which components must be active during which phases is given in Table 1 for your convenience (inactive 7-segment LEDs must be blank). Listed below are details specific to each component.

- 7-segment LEDs for four-digit number
  - Shows *answer digits* during the **Setup Phase**, **Done Phase**, and **Hold Phase**
  - Shows *guess digits* during the **Guess Phase**
- `Wins` 7-segment LED
  - Has `0` as its initial value during power on
  - Maximum supported value is `15` (show `10` to `15` as hex)
  - Value overflows back to `0` after exceeding `15` wins
  - Value is incremented once during the **Done Phase** if the player has won during the current round
- `Lives` 7-segment LED
  - Has `7` as its initial value per round
  - Value is decremented after each **Compute Phase** regardless of correctness of guess
- `Bulls` and `Cows` 7-segment LEDs
  - Proper value is updated and displayed during the **Guess Phase**
  - Duplicate digit cases need not be handled
- `Win` and `Lose` LEDs (labeled)
  - Active only if the player has won/lost during the current round
  - Player wins if the guess is equal to the answer even if the player has zero lives
  - Player loses if the guess is not equal to the answer and the player has zero lives

2.5 Tips

- Divide the circuit design into self-contained **subcircuits** with **well-defined input and output pins** planned beforehand (see next section for suggested subcircuits); this allows both an easier time with debugging and a way to delegate tasks
- Imported subcircuits may not be modified directly; however, you may open subcircuits in a separate Logisim window and select the `Reload Library` option on the main circuit after the subcircuit edits have been saved
- Subcircuit display symbols may be manually edited via the **Edit Circuit Appearance** which allows changing of the component shape, reordering of I/O ports, and adding of on-screen labels
- Go through at least the first four sections of Logisim's **User Guide/documentation** and the **Library Reference**; the sections **Subcircuits** and **Wire bundles** are must-reads before laying out your circuit design
- You will likely use all the gates in the **Plexers library**—ensure that you know what each component does and what other components are available in Logisim
- You may need to use tri-state buffers (**Controlled Buffer/Inverter** in Logisim) for better I/O control
- Make extensive use of Logisim's **Wire Bundle**, **Splitter**, and **Data Bits** features to save space—it is of note that the model solution has 19 well-defined subcircuits
- Logisim allows both **negating gate inputs** to avoid adding separate inverters and **resizing gate symbols** to save space and make your design cleaner
- Use **read-only memory (ROM) components** to quickly form combinational circuits from truth tables instead of deriving expressions via K-maps and mapping out logic gates
- Simple combinational logic manipulating the clock signal inputs of each flip-flop/register (i.e., **allowing/preventing flip-flop value changes based on state**) may be needed
- **Manually driving the clock** via the `Simulate > Tick Once` feature (use the shortcut!) may help with debugging
- You may use third-party libraries such as `https://github.com/marceloboeira/logisim-7-segment-display-driver`; use **floating values** (displayed as blue lines) as inputs to the said decoder to have 7-segment LEDs appear blank
- Start early, save often, and make backups (using version control systems such as *Git* is a good habit to form); no special consideration will be given to make up for accidental file erasures and the like

2.6 Subcircuit suggestions

- **Game state controller** – takes in the current state and user inputs and provides the next state as output lines (ideally derived via sequential design techniques)
- **Display controller** – has output lines directly connected to output components; input lines are processed/filtered via gate logic (multiplexers, decoders, etc.) depending on the game state

- **Memory unit** – contains most/all the flip-flops/registers of the circuit with input lines providing new values for each and output lines supplying current values to other circuits; this allows centralized management of data changes and tracking
- **Input controller** – takes in button states as inputs and converts them into logical values (e.g., which register will receive the current numpad input); this decouples how input is retrieved from the user and what values are expected by the other subcircuits (i.e., you may have a simple number input interface at first and refactor it as a numpad cleanly later on)
- **Game logic controller** – performs all computations needed during game proper (e.g., bulls/cows/lives/wins computation, win/lose signalling)

3 Grading

3.1 Extra credit

Two opportunities are given to earn at most **30 bonus points** with details given below. Note that bonus points will only be granted if the cumulative score without bonuses amount to **at least 60%**.

3.1.1 Generate random answer

Add an extra button labeled `R` that, when continuously pressed during the next triggering edge, randomizes the current input for the answer during the **Setup Phase** once. Note that this button must **not** be functional for any other phase. The new value must be displayed immediately after the triggering edge. Holding the button for multiple triggering edges should result in a new number after each triggering edge.

The four-digit value must **not** have any repeating digits and must be generated with an *acceptable level of randomness* (consult your instructor if you have questions regarding this definition).

You are allowed to use the **Random Generator** component of Logisim. A seed value of `21140145` with `16` data bits **must** be used **without skipping any valid number generated in the sequence**—no credit will be given otherwise.

Proper implementation of the said extension will merit **15 bonus points**.

3.1.2 User input validation

The project specifications allows players to input answers and/or guesses with duplicate digits; the implementation is not required to handle numbers with duplicate digits. A logical extension to the project is to disallow inputs leading to duplicate digits.

Add an extra LED labeled `E` (for *error*) which should activate while the user is pressing down on a numpad button which will result in a number with duplicate digits. Note that this LED should only activate during either the **Setup Phase** or the **Guess Phase** (phases allowing digit input).

Finally, the default value for the answer that is set during the **Setup Phase** of each round should be `1234`. The default guess set during the first **Guess Phase** of each

round should be `9876`.

Proper implementation of the said extension will merit **15 bonus points**.

3.2 Scoring rubric

Criterion	Points
Ten-key numpad provides input as expected	10
Selector buttons and LEDs are correct with respect to digit input	5
`Enter` button works as expected	5
`Start`, `Game`, and `Wait` are activated during the right phases	5
All non-selector LEDs are activated in the right sequence	5
All 7-segment LEDs are deactivated at the right instances	5
Four-digit display shows the correct values for each phase	10
Number of bulls is properly computed	5
Number of cows is properly computed	10
Number of lives is properly computed	15
Number of wins is properly computed	15
`Win` and `Lose` are activated under the proper win conditions	10
Bonus 1: Valid random answer button	15
Bonus 2: User input validation	15
	130/100

3.3 Testing

- Expect all test input values to have unique digits (i.e., no duplicate digits) unless *user input validation* has been implemented
- The clock will be running at either `2 Hz` or `4 Hz`
- Earned points may be withdrawn for particular items if answers to related questions are deemed insufficient or outright wrong

4 Submission Details

Submit an archive `cs21172project1.zip` (or `.tar.gz`) containing all relevant *Logisim circuit files* (`.circ`) through the UVLê submission module. Ensure that your archive contains **ALL** `.circ` files needed to run properly on a fresh system with Logisim (either verify this yourself or ask your instructor before the deadline to check if the submission is valid). **Projects with missing and/or corrupted files will not be checked. No leeway will be given for this.**

Note: non-working or late submissions will NOT be credited.

Warning: Logisim allows a multitude of ways to detect plagiarism; members of all groups suspected of plagiarism will be given an `INC` and **must** undergo a lengthy process during the **midyear break** to prove otherwise.

All group members must be present during project checking.

Note that project submission and checking may be done **earlier** than the specified dates below in case of scheduling concerns.

Deadline: 2018 May 23, 11:59pm (before Thursday)
Project demo/checking: 2018 May 24 (Thursday; time to be announced)