Introduction to Embedded Systems
Project Report

Roni Haval Khalil - s245495
Kareem Al-Ghazali - s244822

Deadline: 06-12-2024

Table of contents:

Contribution to project work:

Hardware contribution:

We have both designed the hardware of our project, worked together on how the components needed to be configured and how we wanted the design to be. When introduced to new components that we havn't worked with during or labs, we both worked on how to implement it.

Software contribution:

Roni coded most of the methods, loop, setup and structure, and Kareem helped with implementation of the LCD, Membrane and Joystick module.

Idea of project:

We have both brainstormed several project ideas, and contributed equally to the final idea of the game mechanics.

Report contribution:

Kareem:

- Requirements
- Discussion
- Conclussion

Roni:

- Design
- implementation
- Test

Requirements:

Game requirements:

Our project is an interactive matching game where players must replicate a randomly generated output displayed on the LCD. The game must consists of several levels, with difficulty progressively increasing as the player advances.

Players will utilize the Membrane Switch Module to input their responses and replicate the random output. If time permits the will we integrate the Joystick Module, adding a new layer of complexity to the gameplay. With the directions up, down, left and right.

LCD Module Display:
This will serve as a visual interface, showing the current level, and displaying feedback (e.g., success or failure messages) at the end of each level.
Passive Buzzer:
The buzzer will act as a timer, with the frequency of the buzzing increasing as the level timer runs out, providing an auditory cue to the player.
RGB LEDs:
Light up green to indicate a win, light up red to indicate a loss. Additionally, the blue LEDs will also serve as a timer indicator, changing intensity or behavior as time runs out.

This modular approach allows for a dynamic gameplay experience, with the potential for extended functionality depending on available time and resources. By combining various hardware components, the game aims to provide an engaging and challenging experience for players.

Hardware requirements:

- (1) x Elegoo Mega 2560 R3
- (35) x M-M wires
- (4) x F-M wires
- (8) x 220 ohm resistor
- (1) x 10k ohm resistor
- (2) x 1k ohm resistor
- (3) x 5 mm red LED
- (3) x 5 mm green LED
- (2) x 5 mm blue LED
- (1) x Passive buzzer
- (1) x LCD1602 Module
- (1) x Button
- (1) x Membrane Switch Module
- (1) x Joystick Module

Software requirements:

Development tools:

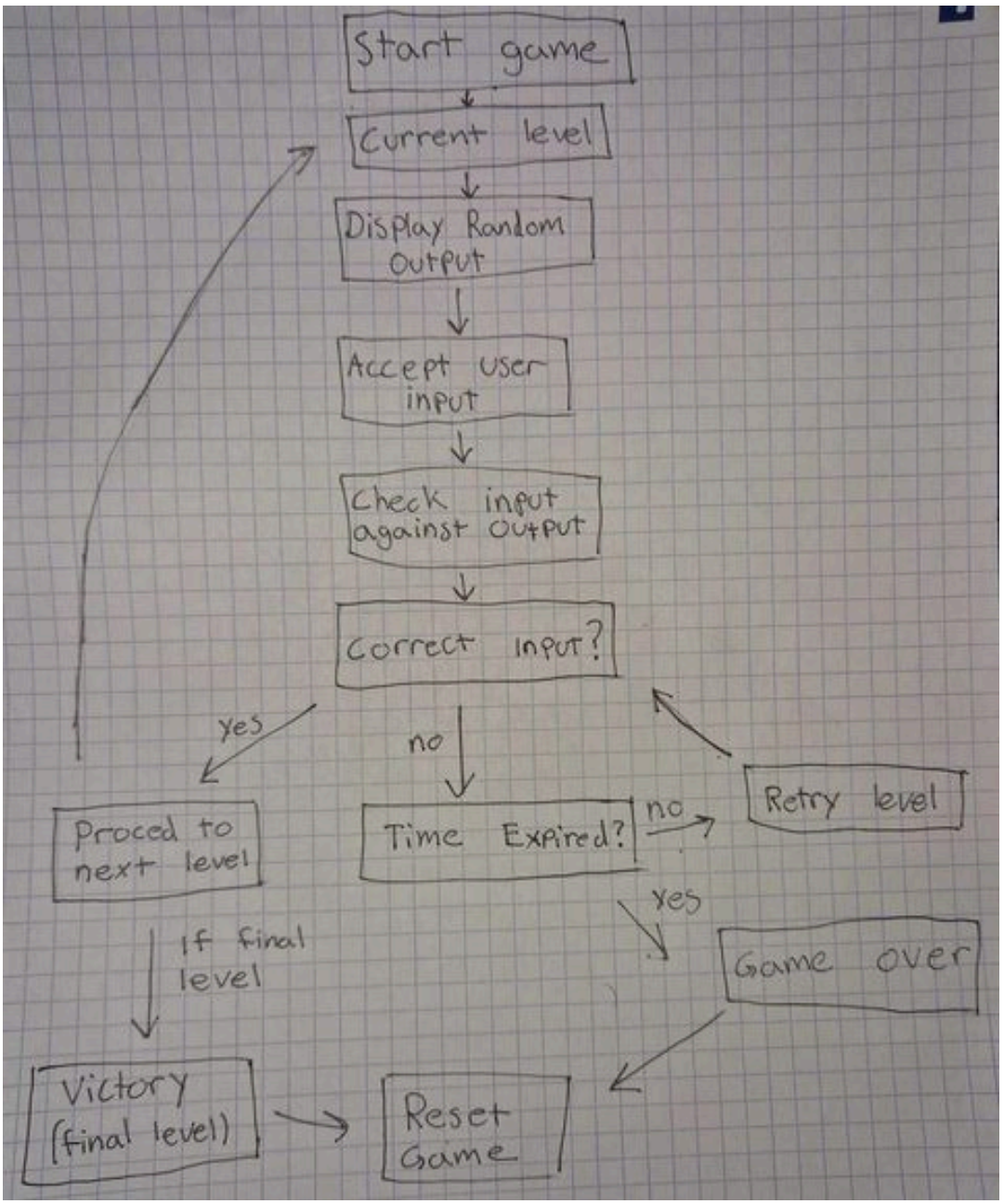
- Arduino IDE.

Libraries used:

- <LiquidCrystal.h>
- <Keypad.h>
- <TimerOne.h>

Design:

Flow diagram:



This diagram represents the flow of our software, the software is designed in a way so that the game never really ends as seen on the diagram. When all the levels is completed the game resets, it even resets when you lose. In principle the software only need to compile once, and the game will loop.

Hardware design:

Our hardware design has some thought behind the specific placement of each component. We thought the LCD screen was really important, since its the main indicator of our game, and it's interactive capabilities with the user. Thats why we placed it in the middle on the breadboard, as seen on the wiring diagram under implementation.

We have split up the RGB LED's with the Red LED's on the left and the Green LED's on the right of the LCD, in order to create the effect of a "good" side and a "bad" side. The time indication, witch is the Piezo buzzer and the two Blue LED's, is on the far right of the breadboard. We would have liked a bit more space, for the time indication hardware, but we will get into that in the discussion. We choose different color LED's in order to indicate different types of mechanics.

Software design:

The thought process behind the software, is that we wanted to implement as many methods as possible, to avoid code duplication. It was important in the main loop, to only keep the hardware components that needed to be checked or active constantly. As you can see in our pseudo code under implementation.

We wanted the levels to get harder as you progress, so we set the time for each level to be constant. The only thing that got harder was the random output witch got longer by one index each level. So on level 1 we start with a length of 4 and on the final level, level 8, it has the length of 12.

The three most important methods needed in order for our game to run is:

- Levels(level)
- checkComplete(level)
- GameOver()

In the pseudo code you see generally how each of these methods work. The loop and setup method is obviously also crucial for the function of the software.

It's in our levels method we define how many levels we want in our game. It is coded so you can increase and decrease the levels, without it interfering with the functionality of the game. Here the winning conditions is also checked. The random generated output is being generated in this function, witch will get longer depending on the level. Here we use the function

random(), to randomly pick a index in an array, this array consist of all possible inputs. Finally we reset timer for each level.

The checkComplete method basically checks each index of our random output and user input, and we increment a counter every time each index in the arrays is equal to each other. We can now with the counter check if the level is completed or nor, here we compare the counter and the length of the random output. If they are equal to each other, the level completion conditions are met (trigger Green LED's), if not the you get to try again with the timer proceeding where it left off (trigger Red LED's).

The GameOver method triggers the buzzer with a continuous frequency, triggers Red LED's lightshow and resets every variable restarting the game.

We also have implemented a button with a hardware interrupt attached, this interrupt restarts the levels to level 1 and resets all the variable, it is basically a do-over button. We felt we didn't need the use of a debounce, since it isn't a main mechanic for the game. It wouldn't mean anything if the button registered a couple more inputs, since it function is to restart the game and nothing more.

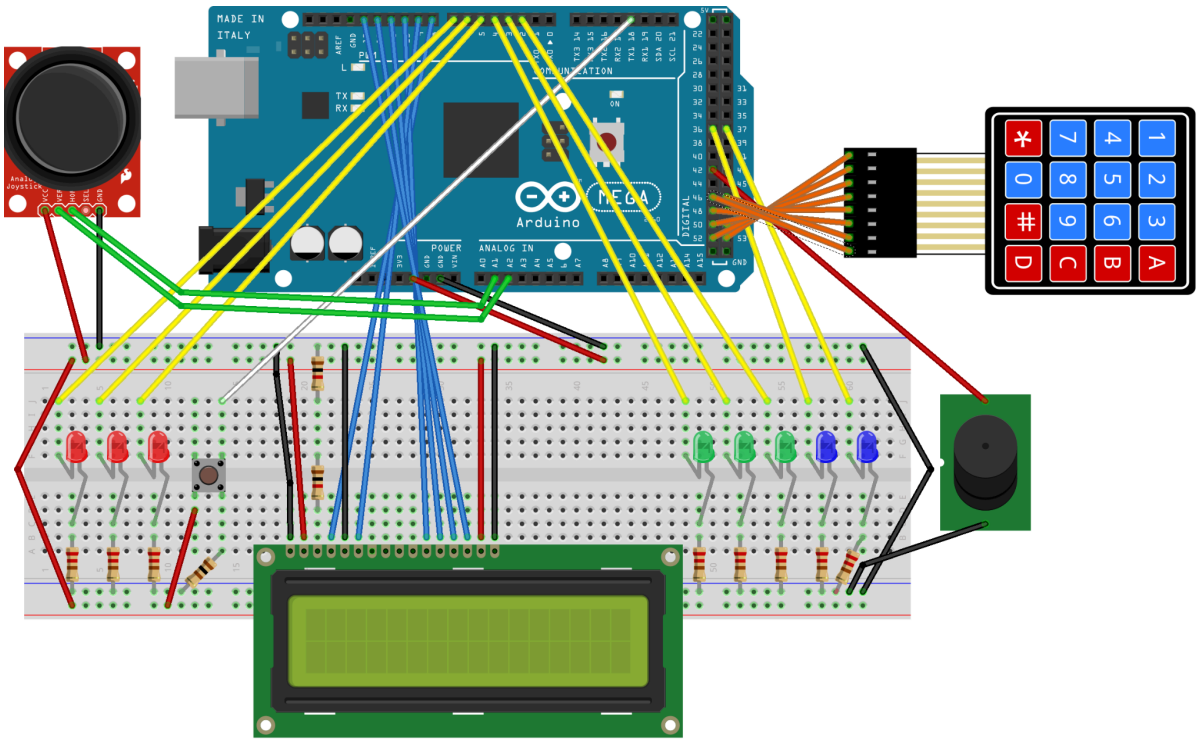
The joystick module was is quite interesting, since the use of debouncing was most important. It operates with two main values, an x-coordinate and y-coordinate going from 0-1024. In order for the software to know what left, right, up and down was, we needed a threshold and a center variable, set to 512. So when the input exceeds the threshold+center, we knew when to register the value. This is where the debounce was important, with no delay the speed at witch it would register was immensely high. So a delay on 2 ms, is just enough to register one direction and fall back to the center again.

I have used the function millis() a lot in the overall software, because it is something that always runs in the background, witch we can use to our advantage. For example we use it to check wether the timer has run out in our main loop. At first i used delay() to help operate the LED's, but found that it introduced timing, overlapping and obstructions with other components. Here the fix was to use millis() in while loops, witch helped with the timing.

The rest of the software is more intuitive consisting of if-else statements, LCD and setup.

Implementation:

Wiring diagram:



Here we see the wiring diagram of our game, witch is manually made with the program “fritzing”. It isn’t exactly a one to one copy of our wiring, since the piezo buzzer is attached onto the breadboard and not beside it. For the RGB LED’s we see that they are configured incorrectly, the but in this instance the short pin is the anode and the longer pin is the cathode.

Pseudo code:

Below you see our pseudo code, witch we have written with the setup and loop first and all the necessary methods under.

START

INITIALIZE:
- Configure LCD, keypad, LEDs, speaker, and joystick pins.
- Attach interrupt to reset button.
- Display "WELCOME" message on LCD.
- Initialize Level 1. Use method for levels

MAIN LOOP:
- Continuously check:
  1. Keypad Input:
     - If '*', clear user input.

- If '#', check if user input matches the random output. Use check complete function.
- Otherwise, store input and display it on the LCD.

2. Joystick Input:
   - Read joystick position.
   - Append directional input ('<', '>', '^', 'v') to user input.
3. Time Limit:
   - If time exceeds level limit, invoke GAME OVER.
4. LED and Speaker Updates:
   - Blink Blue LEDs to indicate time running out.
   - Play speaker tone with decreasing interval.

FUNCTIONS:

1. Levels(level):
   - Display level on LCD (except victory level).
   - Generate random sequence based on the level.
   - Start level timer.
   - Display random sequence on LCD.

2. readJoystickInput():
   - Read joystick position and translate into directional input.

3. blinkBlueLEDsNonBlocking():
   - Toggle Blue LEDs state at decreasing intervals.

4. blinkRedLEDsNonBlocking():
   - Blink Red LEDs non-blocking for error indication.

5. blinkGreenLEDsNonBlocking():
   - Blink Green LEDs non-blocking for success indication.

6. checkComplete(level):
   - Compare user input to random sequence.
   - If matched:
     - Display "LEVEL COMPLETE".
     - Green LED’s sequence
     - Advance to the next level.
   - If unmatched:
     - Display "TRY AGAIN".
     - Red LED’s sequence
     - Reset user input.

7. GameOver():
   - Display "GAME OVER".
   - Blink Red LEDs and play speaker sound.

- Reset to Level 1.

8. Restart():
   - Reset all game variables and restart at Level 1.

END

Test:

Component Testing:

We have testet each component in our hardware to make sure it all works as intented. Such as making sure the LCD blacklight light up with the correct amount of voltage, here we tried many different combinations of resistors, and found that two 1k ohm resistor gave us the desired outcome. Moreso we made sure the LED's light up, the buzzer work and etc.
We used tools such as the serial monitor to make sure each input for the Membrane and Joystik modules works.

Game logic testing:

Here we made sure to test each scenario, such as restarting, failing, winning conditions and completing a level, to make sure the game logic works as intended. This was also needed to test if the LED's light up when supposed to.

User testing:

Here we got help from our fellow students to test wether the time for each level was appropriate. We wanted the player to fail the first and second time, and complete it on the third time. Here we started on 20 seconds for each level, but settled for 14 seconds after some testing.

Results:

Each component and the game mechanics works as intended.

Discussion:

Reflection:

What went well:

In these past 4 weeks we have had many thoughts, ideas and failures. But it all ended up good in the end. We would say our hardware and the development of hardware had 0 to no problems, it was by far the easiest part to make, so that part went well. And the project planning went smoothly too, we both had the same route in mind, and we came to a quick agreement and started executing our project instantly. In the software part of the project, we had some difficulties but nothing we couldn't get past and complete.

Challenges:
We faced a couple of challenges, and all the challenges were in the software part. We struggled with the code for our random input display, we also struggled with the idea of where to put the interrupt part in out code, since we had forgotten we needed one until second week, and when we had an idea on what to do we also struggled with the code for our interrupt button as well.

Game performance:

The game performance goes pretty much as planned and expected, but there is one thing we didn't know the reason for. Every time the timer runs out in our game, there is a 1-2 seconds delay before the buzzer goes off. But it's a good implementation, we call it the suspense period, it gives a good feeling when you complete it in that period, like a clutch factor.

Our game has a pretty solid baseline for what makes it a fun game, a game needs difficulty to some degree so there is a happiness and feeling of achievement when you win, our games also challenge a players reaction, memory and the ability to adapt. These things are always fun to explore and try out. And all of these things make our game enjoyable.

Improvements:

There are always possibilities for improvement, and in this case too. Some improvements we could make are changing some levels, so instead of all the levels being the same concept, we could add “simons says” the later stages, or add some memory levels, where you have to remember what numbers were displayed and then press them in.
We could also add a victory melody with the buzzer after completing all levels, and a failing melody when the game over method triggers. It would also be smart to add a second breadboard to get a better overview of all the different hardware components.

Conclusion:

Our interactive matching game successfully combines hardware and software to create an engaging and progressively challenging experience. Despite initial challenges, we overcame them to implement key features like non-blocking LED behaviors, a dynamic timer system, and an interrupt-based reset.

Thorough testing ensured each component and the game mechanics worked as intended, resulting in a balanced and enjoyable gameplay experience. While improvements such as diverse level mechanics and enhanced audio-visual feedback could further refine the game, the project meets its objectives. Overall, it has been a rewarding learning experience.

Appendix:

Software:

The software can be found in the zip. file .