ELECTRICAL AND ELECTRONIC ENGINEERING PROGRAMME

PROGRAMMING TECHNIQUES (EEE 60104)

Group No.	**1**	**Total Marks**	
Lab No.	**4**		
Title of Lab	**Memory Game**		

Lecturer	Dr. Chew Wei Jen	
Date	**Lab Date: November 15th, 2023**	**Submitted Date: November 29th, 2023**

Lab Report Item/total marks	**Marks**
Format/10	
Abstract and Introduction/15	
Materials and Method/15	
General Results and Discussions/50	
Conclusion and References/Appendix /10	
Total /100	

No.	**Members ID**	**Name of Group Members**
1	0360231	Muhammad Rafif Edie Wijaya
2	0364144	Law Shing Yi
3	0361508	Wong Kai Tian
4	0349907	Syazani Zikry Zahari
5	0353217	Surryaraj A/L Poobalan

Table of Contents

`

1. Abstract

The goal of this lab was to build a memory game with an Arduino Uno board. The circuit comprises 5 LEDs and 5 pushbuttons, creating a visual and interactive setup. Through programmed logic, the Arduino Uno generates random LED patterns, progressively increasing the complexity of sequences as the game advances. Players are tasked with replicating these sequences by pressing corresponding pushbuttons. A green LED is activated by successful sequences, while a red LED indicates that the game is ended when an improper input is made. The exercise places a strong emphasis on component testing for seamless functionality in addition to programming expertise.

2. Introduction

The main objective for this lab session is the practical experience of writing a C program to simulate a memory game using an Arduino Uno board. A circuit contains 5 LEDs and 5 pushbuttons that are connected to the Arduino Uno board output and input respectively. In this instance, the LEDs serve as a visual indicator that randomly lights up based on the program code, and the player was supposed to press the pushbuttons according to the LEDs that light up respectively.

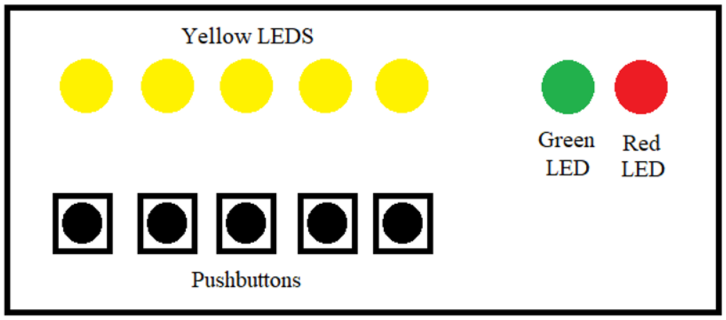


Figure 1: Memory game setup

After the player inputs the correct pushbutton sequence each time based on the LED light's random pattern, a green LED will light up and a red LED to indicate the game is over. The game will continuously output a random sequence of LED patterns each iteration and have the player repeat the same pattern. Each iteration will increase the number of LEDs that light up by one. It is important to check all of the components' usability one by one for the lab to ensure smooth workflow and only focus on the programming code part.

In the context of the lab, the students dive into the intricacies of programming the Arduino Uno to simulate the memory game effectively. Five different LED light iterations will increase the number of LEDs light up by one which is produced by the Arduino Uno programmed logic, which can emit an electrical signal to the green and red LED in response to the sequence pushbutton being pressed correctly. In addition to improving the students' grasp of C programming for microcontrollers, this exercise offers valuable knowledge about the practical uses of LED and pushbutton in C programming.

3. Methodology

a. Materials:

   i. Arduino Uno
   ii. Yellow LEDs
   iii. Red LED
   iv. Green LED
   v. 100 Ω resistors
   vi. 10 kΩ resistors
   vii. Pushbuttons

b. Methods:

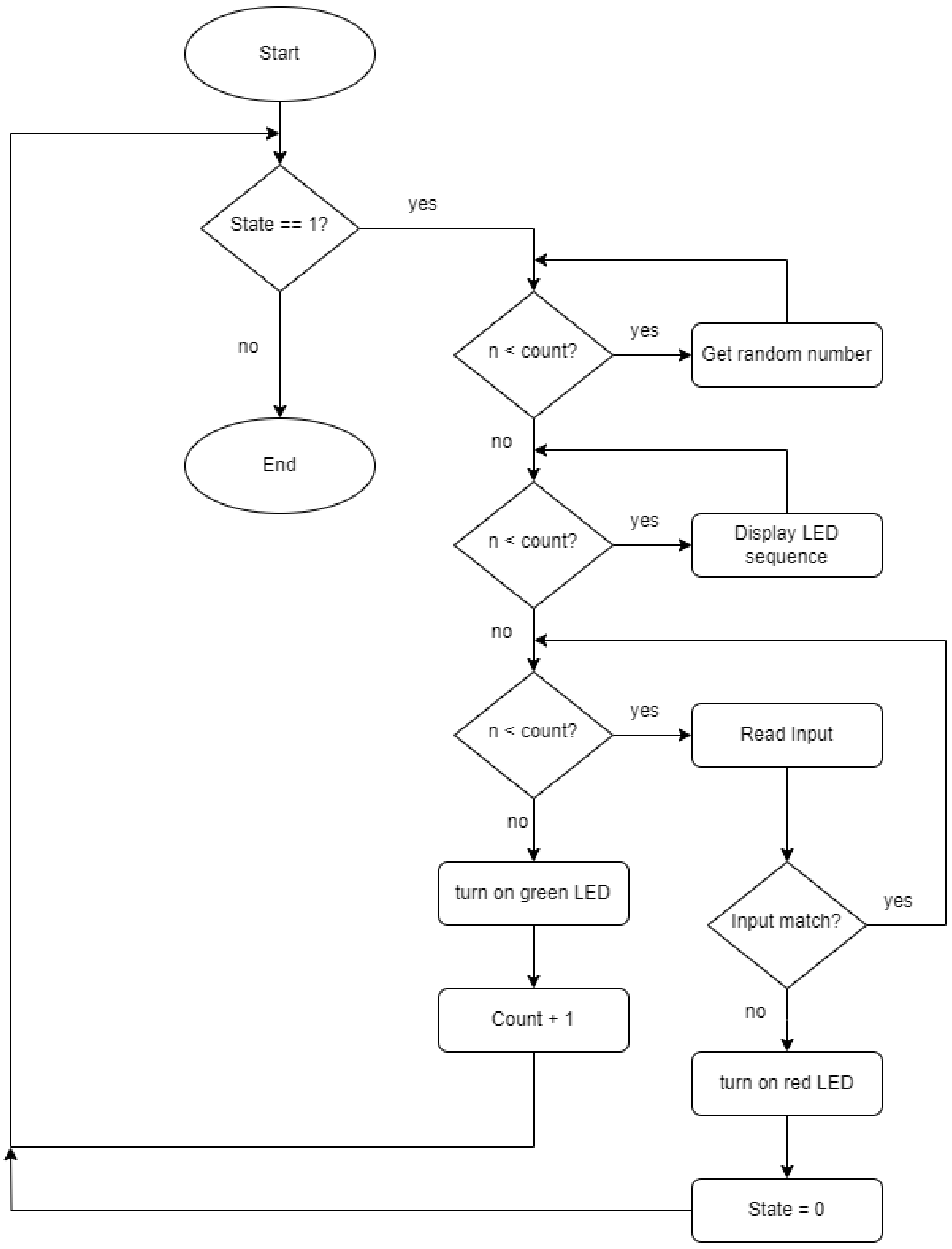


Figure 2. Flowchart of the program

c. Procedure:

i. 5 yellow LEDs are connected to pin 8 -12 through a 100Ω resistor.

ii. 5 push buttons are connected to pin A0 - A4 through a 10kΩ resistor.

iii. A green LED and red LED are connected to pin 3 and pin 4 through a 100Ω resistor.

iv. A C program is then written to continuously output a random sequence of LED patterns at each iteration and have the player repeat the same pattern. Each iteration will increase the number of LEDs that light up by one.

4. Results

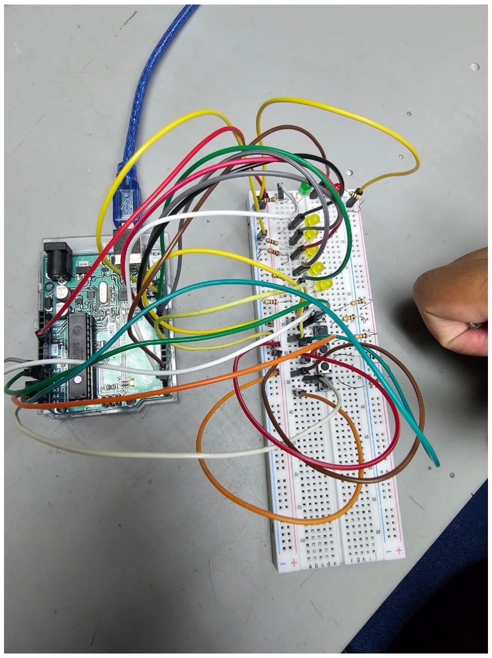


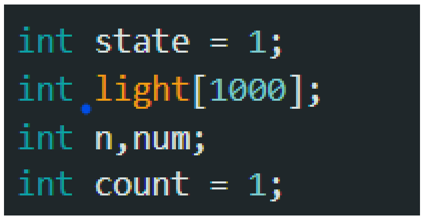
Figure 3. The complete circuit of the memory game.

5. Discussion

In this lab, the objective is to let the user turn on LEDs in the same sequence as the randomly generated sequence. In order to create the sequences, 2 new libraries were included, <*stdlib.h* > and <*time.h*>. These 2 libraries will be used to create the random sequences. The first part of the code is to set up the Arduino pins for inputs and outputs

[1]. For this lab, pins 7 to 12 were used as outputs, these pins will be connected to the LEDs that will light up in certain sequences. For the input, analog pins A0 to A4 were used. These input pins will be connected to the push buttons. Next, there are 2 LEDs that check whether the user inputs the correct sequence, these 2 pins were connected to digital pins 2 and 3.

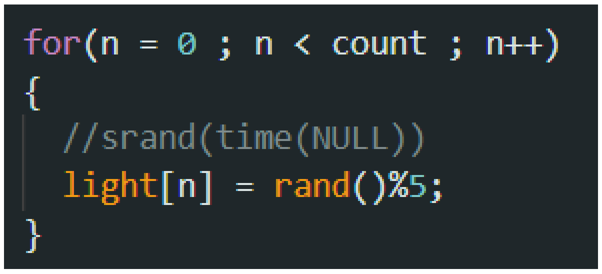
Some of the parameters used can be referred to in Figure 4. *State* is the variable that determines if the game will keep on running or not. The array *light[100]* is an array that stores the random value from 1 to 5. This array will be used to turn on the 5 LEDs in a certain sequence. The rest of the variables, *n, num, and count* are used to determine the round of the game.



```
int state = 1;
int light[1000];
int n,num;
int count = 1;
```

Figure 4. Initial variables created for the game.

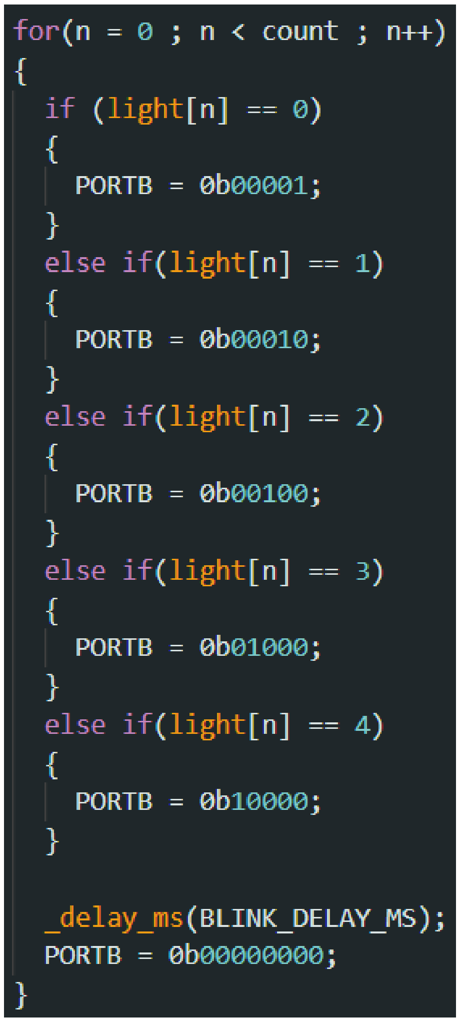
In order to make sure the game runs in a correct sequence, all parts of the game were put in a *while(state)* loop. Next, depending on which round the game is played, a random number between 0 and 5 will be created and stored in the array *light[n],* as seen in Figure 5.



```
for(n = 0 ; n < count ; n++)
{
  //srand(time(NULL))
  light[n] = rand()%5;
}
```

Figure 5. Random sequence creation.

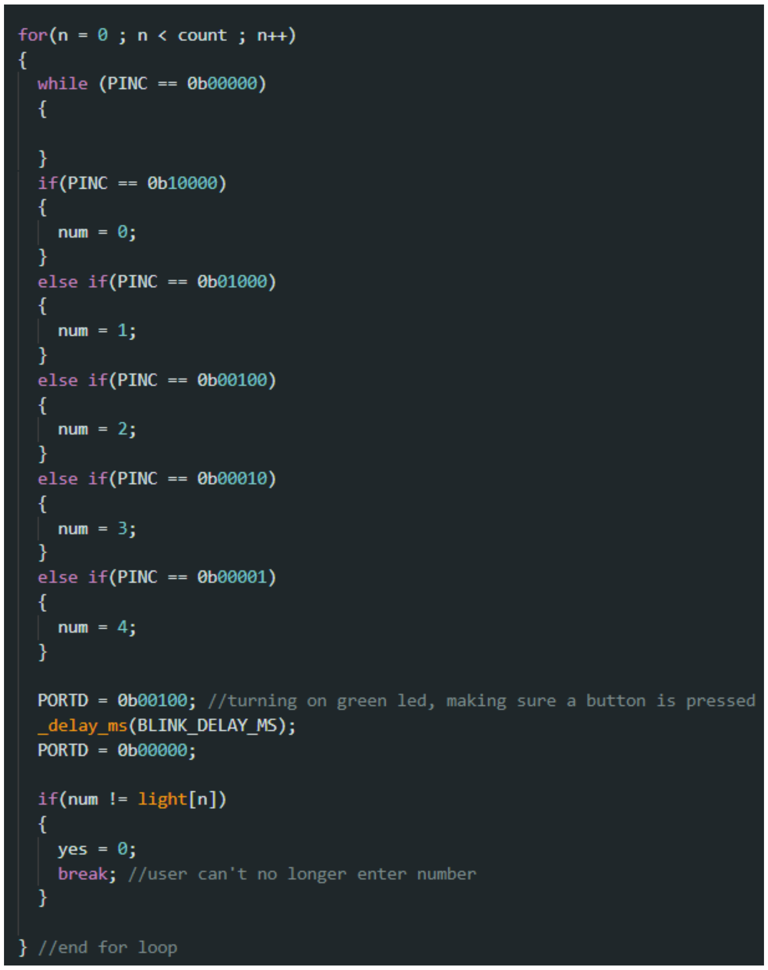
The next part of the code is to turn on the LEDs. The number of LEDs that will be turned on depends on the round of the game, as it is controlled by the variable *count*. It starts off with 1 LED and if the user presses the right button, it will keep going until 5 LEDs are turned on in the same round. In Figure 6 the LEDs are turned on using a *for loop*. It will read the array *light* and match its contents with the appropriate LED. Then, the LED will turn on for 1 second before turning off.



```
for(n = 0 ; n < count ; n++)
{
    if (light[n] == 0)
    {
        PORTB = 0b00001;
    }
    else if(light[n] == 1)
    {
        PORTB = 0b00010;
    }
    else if(light[n] == 2)
    {
        PORTB = 0b00100;
    }
    else if(light[n] == 3)
    {
        PORTB = 0b01000;
    }
    else if(light[n] == 4)
    {
        PORTB = 0b10000;
    }

    _delay_ms(BLINK_DELAY_MS);
    PORTB = 0b00000000;
}
```

Figure 6. Turning on LEDs according to the random sequence

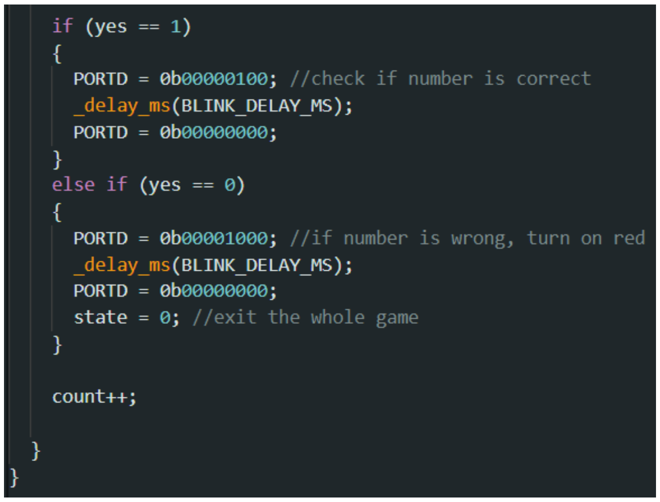
The next part of the code is to assign a number when a specific push button is pressed. In Figure 7, the process of the user pressing a push button and checking if it is the appropriate one is located inside a *for loop*. The number of iterations of this loop is controlled by the variable *count*, which indicates the round of the game. The first part of the loop is *PINC ==0b00000*, this *while loop* is empty in order to make sure the code doesn't move on unless a pushbutton is pressed. The next stage is checking which button is pressed and assigning it a number, button *A4* will be represented with 0, *A3* with 1, *A2* with 2, *A1* with 3, and *A0* with 4. To make sure that each button is pressed, the green LED will turn on for 1 second, which is represented by *PORTD = 0b00100*. After that, the code will check whether the user pressed the correct button, if not it will change the variable *yes* to 0 and break out of the *for loop* so that the user can no longer register the button they pushed.



```
for(n = 0 ; n < count ; n++)
{
  while (PINC == 0b00000)
  {

  }
  if(PINC == 0b10000)
  {
    num = 0;
  }
  else if(PINC == 0b01000)
  {
    num = 1;
  }
  else if(PINC == 0b00100)
  {
    num = 2;
  }
  else if(PINC == 0b00010)
  {
    num = 3;
  }
  else if(PINC == 0b00001)
  {
    num = 4;
  }

  PORTD = 0b00100; //turning on green led, making sure a button is pressed
  _delay_ms(BLINK_DELAY_MS);
  PORTD = 0b00000;

  if(num != light[n])
  {
    yes = 0;
    break; //user can't no longer enter number
  }

} //end for loop
```

Figure 7. Checking the button pressed by the user.

The last part of the code checks if the user can keep playing the game. As seen in Figure 8, if the variable *yes* is equal to 1, it means that the user pressed the correct sequence for that round, and it will turn on the green LED for 1 second. Next, the variable *count* will increase by one which indicates the user going on to the next round. However, if the user presses the wrong sequence, the variable *yes* will have a value of 0, as seen in Figure 7. Then, the red LED will turn on for 1 second, and the variable *state* will have a value of 0, which will break out of the *while loop* at the beginning, which ends the game.



```
    if (yes == 1)
    {
      PORTD = 0b00000100; //check if number is correct
      _delay_ms(BLINK_DELAY_MS);
      PORTD = 0b00000000;
    }
    else if (yes == 0)
    {
      PORTD = 0b00001000; //if number is wrong, turn on red
      _delay_ms(BLINK_DELAY_MS);
      PORTD = 0b00000000;
      state = 0; //exit the whole game
    }

    count++;

  }
}
```

Figure 8. Checking whether to end the game.

6. Conclusion

In conclusion, this experiment successfully met its objective of using Arduino Uno and C software to create a functional memory game. We created an interesting task by adding pushbuttons and LEDs to the circuit. The C program generated random LED patterns, requiring players to replicate the sequence using corresponding pushbuttons. The addition of a green LED for correct responses and a red LED for game-over enhanced the interactive experience. The progressive increase in the number of lit LEDs heightened the game's difficulty, showcasing the versatility of Arduino Uno in creating dynamic projects. This experiment provided practical insights into microcontroller applications, strengthening participants' programming skills.

7. Reference

[1] Arduino. "Port Manipulation," [Online], Available: https://docs.arduino.cc/hacking/software/PortManipulation, Accessed: November 1, 2023.

[2] Arduino. "Pin Mapping for Arduino Uno," [Online], Available: https://docs.arduino.cc/hacking/hardware/PinMapping168, Accessed: November 1, 2023.

8. Appendix

```
#include <avr/io.h> // header file for input output pins
#include <util/delay.h> // header file for delay
#include <stdlib.h>
#include <time.h>
#define BLINK_DELAY_MS 1000

int main(void)
{
 DDRB = 0b11111;
 DDRC = 0b00000;
 DDRD = 0b01100;

 int state = 1;
 int light[1000];
 int n,num;
 int count = 1;

 while(state)
 {
  int yes = 1;

  for(n = 0 ; n < count ; n++)
  {
   //srand(time(NULL))
   light[n] = rand()%5;
  }

  for(n = 0 ; n < count ; n++)
  {
   if (light[n] == 0)
   {
    PORTB = 0b00001;
   }
   else if(light[n] == 1)
   {
    PORTB = 0b00010;
   }
   else if(light[n] == 2)
   {
    PORTB = 0b00100;
   }
   else if(light[n] == 3)
   {
    PORTB = 0b01000;
   }
   else if(light[n] == 4)
   {
    PORTB = 0b10000;
   }

   _delay_ms(BLINK_DELAY_MS);
   PORTB = 0b00000000;
  }

  for(n = 0 ; n < count ; n++)
  {
   while (PINC == 0b00000)
   {

   }
   if(PINC == 0b10000)
   {
    num = 0;
   }
   else if(PINC == 0b01000)
   {
    num = 1;
   }
   else if(PINC == 0b00100)
   {
    num = 2;
   }
   else if(PINC == 0b00010)
   {
    num = 3;
   }
   else if(PINC == 0b00001)
   {
    num = 4;
   }
```

```
      PORTD = 0b00100; //turning on the green LED, making sure a button is pressed
      _delay_ms(BLINK_DELAY_MS);
      PORTD = 0b00000;

      if(num != light[n])
      {
        yes = 0;
        break; //user no longer can enter a number
      }

    } //end for loop

    if (yes == 1)
    {
      PORTD = 0b00000100; //check if number is correct
      _delay_ms(BLINK_DELAY_MS);
      PORTD = 0b00000000;
    }
    else if (yes == 0)
    {
      PORTD = 0b00001000; //if number is wrong, turn on red
      _delay_ms(BLINK_DELAY_MS);
      PORTD = 0b00000000;
      state = 0; //exit the whole game
    }

    count++;

  }
}
```