

**MICROCONTROLLERS FOR MECHATRONICS – MECA442**

Experiment 4: CONTROLLING ARDUINO PERIPHERALS

**Jana Kabrit1, Oussama Al Ahmad Matar1**

**1** Rafik Hariri University, College of Engineering, Department of Mechanical and Mechatronics

*Abstract*

*The aim of this experiment is to design and simulate a circuit that checks a security system’s key. If the user inputs the right key, which is a combination of four numbers, ‘CORRECT You may enter’ is displayed. If the key is wrong, a ‘WRONG Try again’ is displayed, and if the wrong key is entered three times in a row, a ‘INTRUDER ALERT Locked out’ will be displayed with a buzzer going off. This will be done through coding an Arduino Uno that is connected to a keypad, LCD Screen, and buzzer. The code entered on the keypad will be displayed on the screen as the user is typing it. Finally, the simulation will be done on Tinker CAD.*

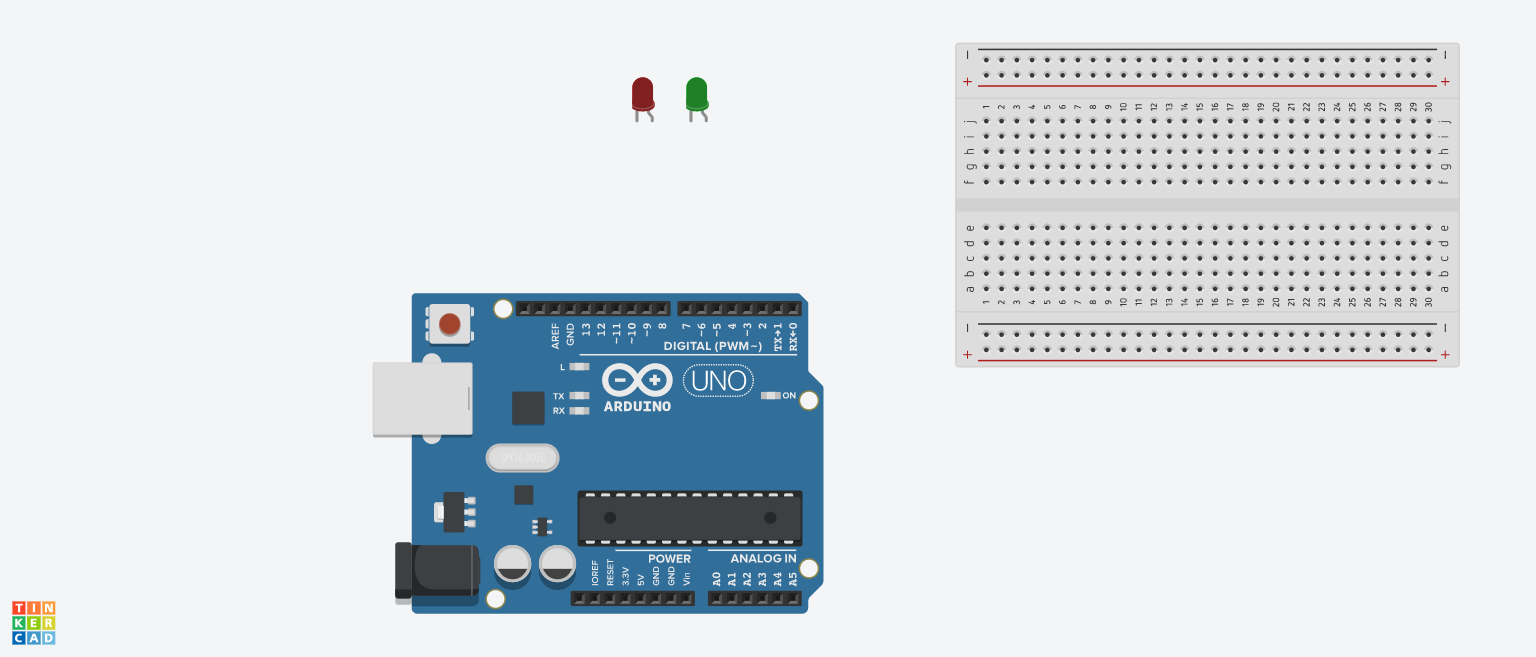
**Keywords:** Arduino Uno, Keypad, Buzzer, LCD Screen, Tinker CAD.

1. INTRODUCTION

In this experiment, we will create an alarm security system that checks if a key is correct. The key will be a combination of four numbers and entered on a keypad then displayed on an LCD screen. If the key entered is correct, the LCD screen would display a ‘CORRECT You may enter’ message, allowing the user to enter. However, if the key entered is wrong, then a ‘WRONG Try again’ message is displayed, and the user is asked to input another key on the screen. Once the number of wrong tries reaches three, a ‘INTRUDER ALERT Locked out’ message will be displayed and a buzzer would go off, locking out the user from inputting another key. This will be done through connecting the keypad and LCD screen to an Arduino Uno and coding it. The code will check which number on the keypad is pressed, debounces it, and displays it on the screen.

1. **MATERIALS AND METHODS**
   1. **Materials**
      1. **Simulated Electronics Components**
2. Arduino Uno

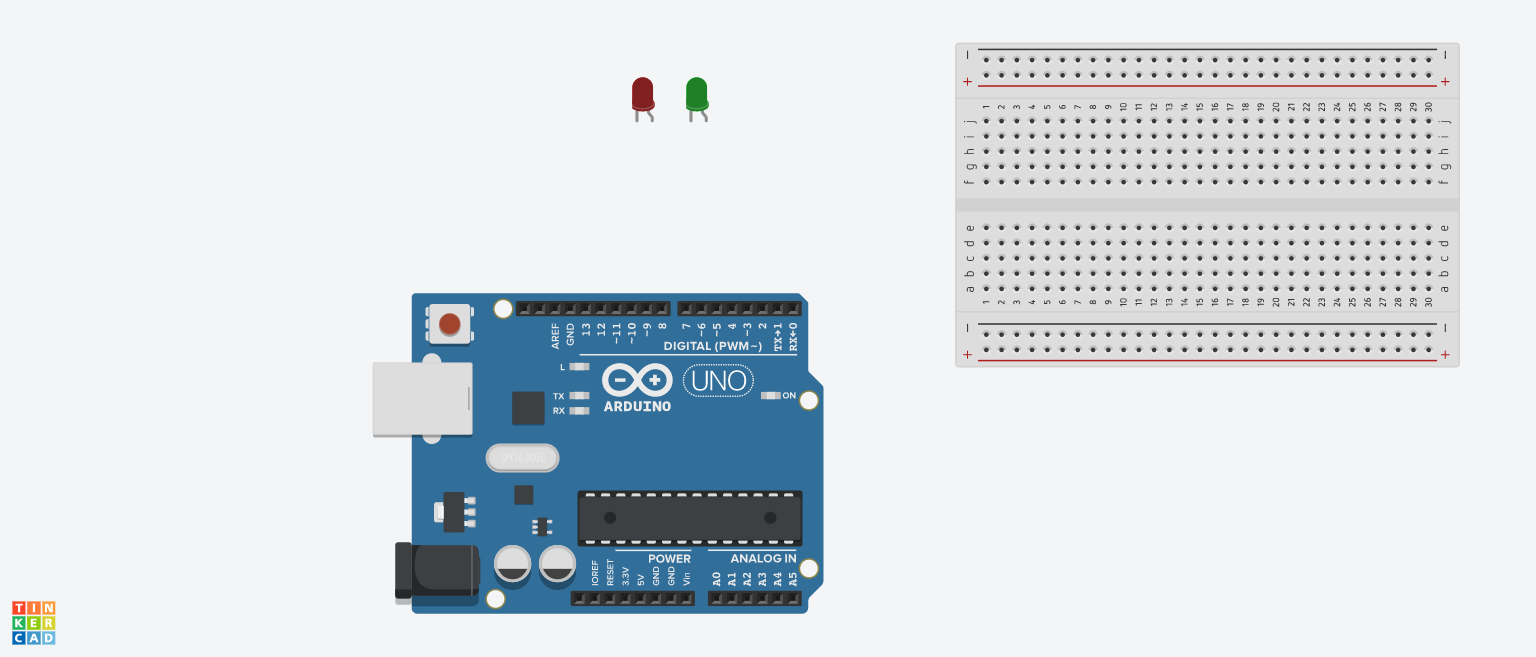
The Arduino Uno (Figure 1) is a microcontroller board based on the ATmega328. It has 20 digital input/output pins (of which 6 can be used as PWM outputs and 6 can be used as analog inputs), a 16 MHz resonator, a USB connection, a power jack, an in-circuit system programming (ICSP) header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.[1]



**FIGURE 1:** ARDUINO UNO BOARD

In this experiment, we need the Arduino so that we can program it to get number values from a keypad and display them on a screen to mimic an alarm system.

1. Breadboard:

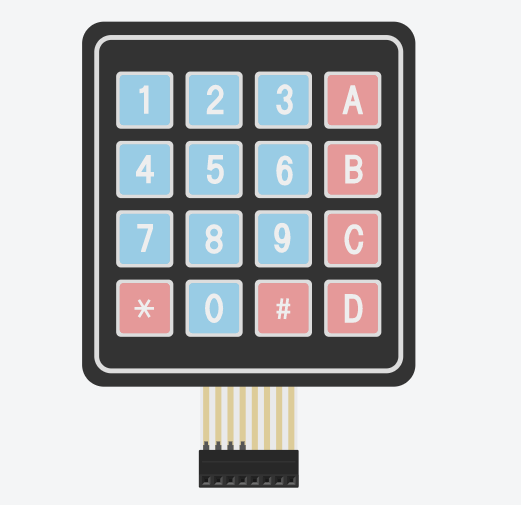


**FIGURE 2:** BREADBOARD

A breadboard (Figure 2) is a rectangular plastic board with a bunch of tiny holes in it. These holes let you easily insert electronic components to prototype (meaning to build and test an early version of) an electronic circuit.[2] In this experiment, we will use the breadboard to connect the LCD display to the Arduino through a potentiometer.

1. Keypad:

A keypad is a set of buttons which arranged in rows and columns. A 3X4 keypad has 4 rows and 3 columns, and a 4X4 keypad has 4 rows and 4 columns. Beneath each key is a membrane switch. Each switch in a row is connected to the other switches in the row by a conductive trace underneath the pad. Each switch in a column is connected the same way – one side of the switch is connected to all of the other switches in that column by a conductive trace. Each row and column is brought out to a single pin, for a total of 8 pins on a 4X4 keypad. Pressing a button closes the switch between a column and a row trace, allowing current to flow between a column pin and a row pin. [3] In this experiment, we will only use the number keys from a 4X4 keypad to be able to input the pin code on the LCD.



**FIGURE 3:** 4X4 KEYPAD

1. LCD Screen:

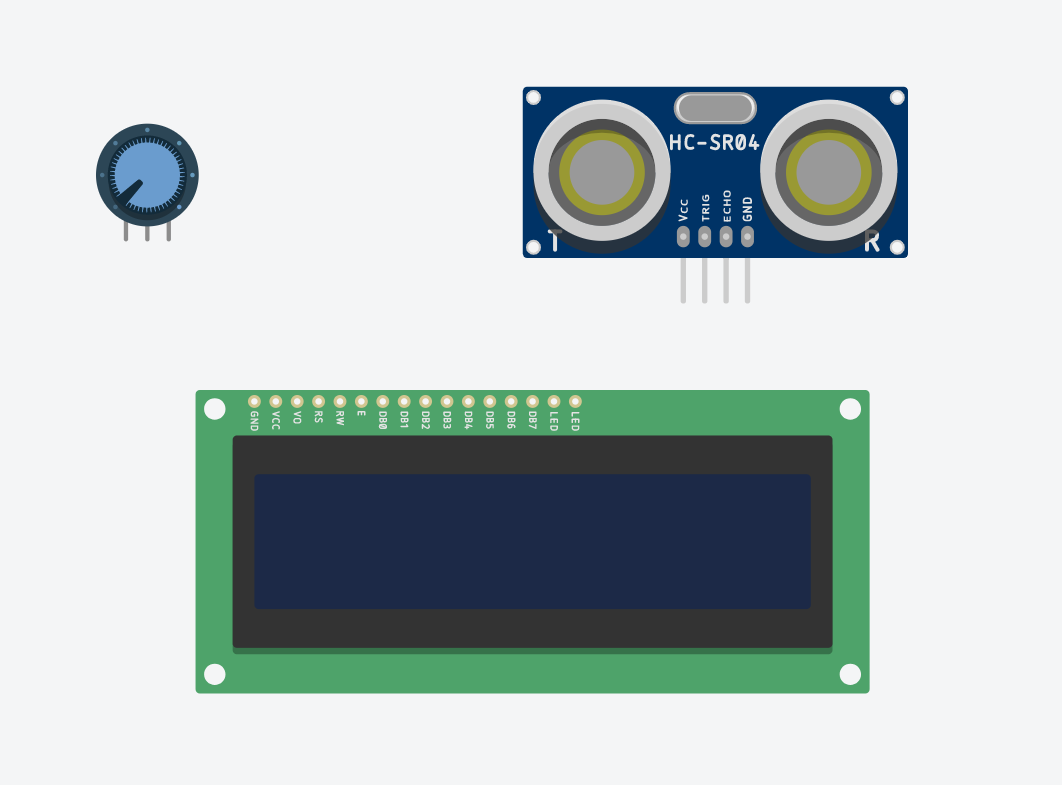
These LCDs are available in many different sizes (16×2 1602, 20×4 2004, 16×1 etc.), but they all use the same HD44780 parallel interface LCD controller chip from Hitachi. This means you can easily swap them. The LCD has 16 connection pins, numbered 1-16 from left to right. If the display does not include a resistor, you will need to add one between 5 V and pin 15. It should be safe to use a 220Ω resistor, but this value might make your display a bit dim. In this experiment, we used a potentiometer to get the best brightness. The maximum current rating of the backlight can be checked from the and used this to select an appropriate resistor value.[4]



**FIGURE 4:** LIQUID CRYSTAL DISPLAY

1. Potentiometer:

The potentiometer, commonly referred to as a “pot”, is a three-terminal mechanically operated rotary analogue device which can be found and used in a large variety of electrical and electronic circuits. They are passive devices, meaning they do not require a power supply or additional circuitry in order to perform their basic linear or rotary position function.[5] In this experiment, we will use the potentiometer to get the best brightness of the LCD screen available.



**FIGURE 5:** POTENTIOMETER

1. Piezo Buzzer:



FIGURE 6: PIEZO BUZZER

Piezo buzzers are simple devices that can generate basic beeps and tones. They work by using a piezo crystal, a special material that changes shape when voltage is applied to it. Simple change the frequency of the voltage sent to the piezo and it will start generating sounds by changing shape very quickly.[6] In this experiment, we’ll be using this buzzer as an alarm for when the pin code is written wrong.

* + 1. **Code Components**

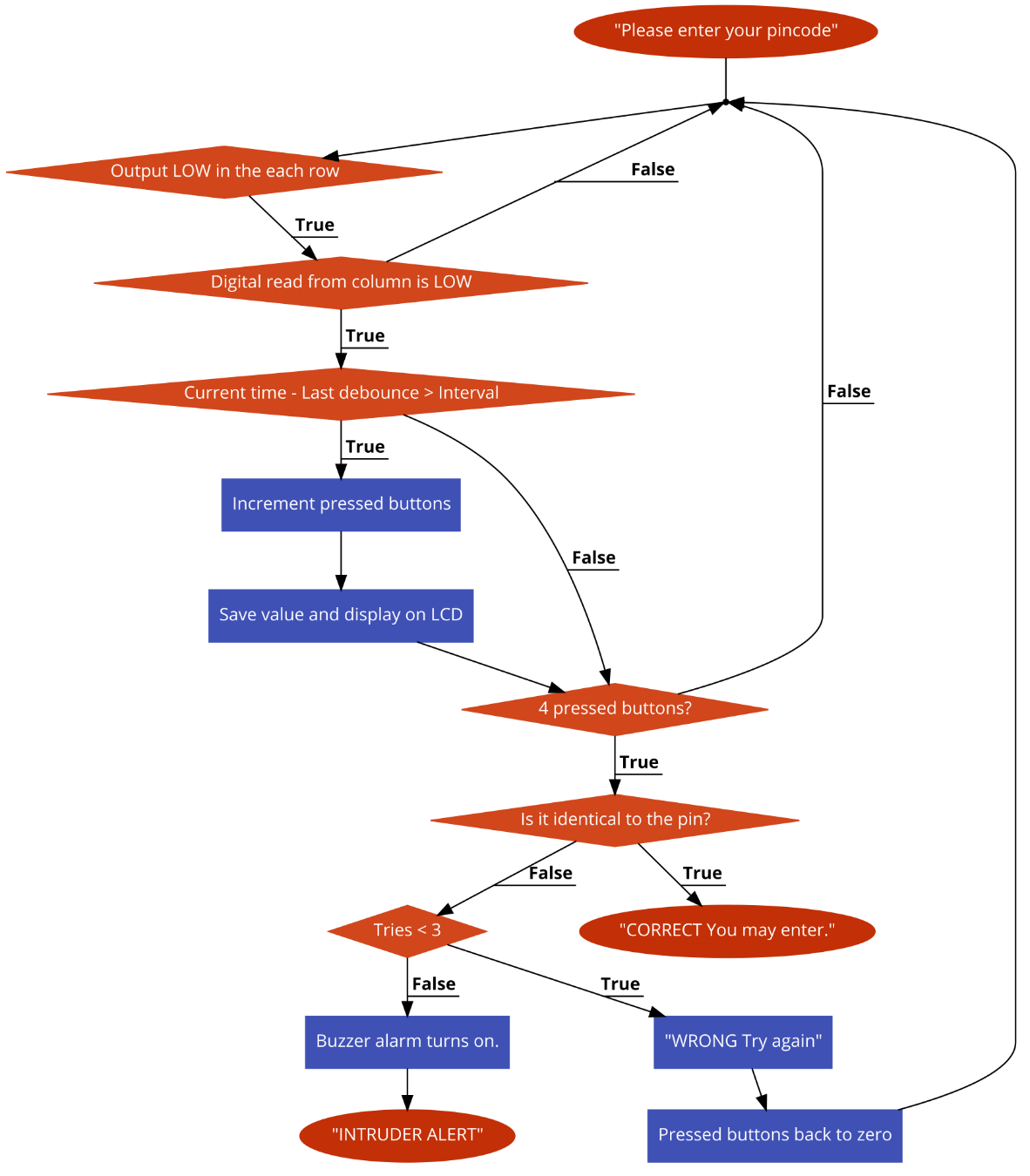
1. const: A variable qualifier that modifies the behavior of the variable, making a variable "read-only" and constant.
2. long int: Long variables are extended size variables for number storage, and store 32 bits (4 bytes) instead of the usual 16 bits (2 bytes).
3. unsigned int: Same as int in that it stores a 2-byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535.
4. void setup: The function is called whenever the program starts. It is used to initialize variables, pin modes, etc. It will only run once after each powerup or reset of the Arduino board.
5. pinMode(): A function used to configure a specific pin to behave either as an input or an output, usually in the void setup.
6. void loop: The function that holds the code inside and runs over and over as long as the board is turned on.
7. millis(): A command that returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
8. LiquidCrystal lcd: Provides a set of endpoints to manage Arduino IoT Cloud Devices, Things, Properties and Timeseries. This API can be called just with any HTTP Client, or using one of these clients
9. lcd.setCursor(): Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display
10. lcd.print(): Displays what is in the argument on the LCD Screen
11. tone(): Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.
    1. **Methods**
       1. **Code Description**

Once the Arduino Uno turns on, it sets up the pins for the LCD display (in our case pins 1, 2, 3, 4, 5, and 6 as the RS, enable, D4, D5, D6, and D7 respectively) to be able to print the output on it. Then sets up the pins for the keypad as rows and columns (in our case pins 9, 8, and 7 for columns 1 to 3, and pins 10, 11, 12, and 13 for rows 1 to 4), and pin 14 which is A0 as a digital pin for the buzzer. It also initiates the variables that keep count of how many tries the user failed at inputting the pin code, and how many times a button of the keypad has been pressed to make sure 4 digits are being compared to the code at one time. Finally, 4 constant numbers are initiated that would hold our pin code that the input is compared to.

First, in the void setup, the LCD would display the text “Please enter you pincode” in two separate lines after setting up the dimensions of the LCD, then clears it out after a second. Then the column pins of the keypad are set as INPUT PULLUP so that they’re always reading HIGH, this is necessary in this setup since the row pins are set as OUTPUT, so that when a button of the keypad is pressed, the consecutive row pin of the button would connect to the button’s consecutive column pin, which enables us to detect that this exact button has been pressed if we send a LOW signal from the row pin to the column pin. This method allows us to use as little as 7 pins instead of 12 pins for each button on the keypad. Finally, the buzzer pin is set as output for when the pin code has been wrong 3 times in a row.

In the void loop, we must consecutively send a LOW signal from each row pin to check which of the column pins detects this LOW signal incase a button was pressed. This happens in a matter of microseconds so no button press can possibly be missed. Since we have 4 rows, we could create a function for each row that would output a LOW in its respective row. Then again in each function, read from each column pin if this LOW signal was detected, if yes, then a button has been pressed and Is displayed on the LCD screen, which also increments the variable responsible for measuring how many digits are on screen at one time. So, each number is mapped to its respective column pin in its specific row function.

However, since we would rather not use delays in our code to avoid spamming the same button over and over by sending a LOW when a button is pressed, we could use a method called debouncing. Since in reality, mechanical switches don’t send a constant HIGH or LOW, but a noise signal that contains a mix of both, we could avoid delays and use the millis function. First, the debounce function would digitally read from the button, then checks if the reading was LOW (since in our case, LOW means the key is being pressed). If it was LOW, it would find the difference between the current time and the last time a key was pressed, which is initially set to zero. If this differences is greater than the interval set to, for example, 100ms, the function would increment the number of times a key was pressed by 1 and returns a reading of LOW, otherwise, HIGH would be returned. This function replaces all the digital reads of the column pins by debounce instead, so only one button can actually be inputted at an interval of 100ms at a time. The process can be shown in the flowchart below (figure 7).



**FIGURE 7:** FLOWCHART

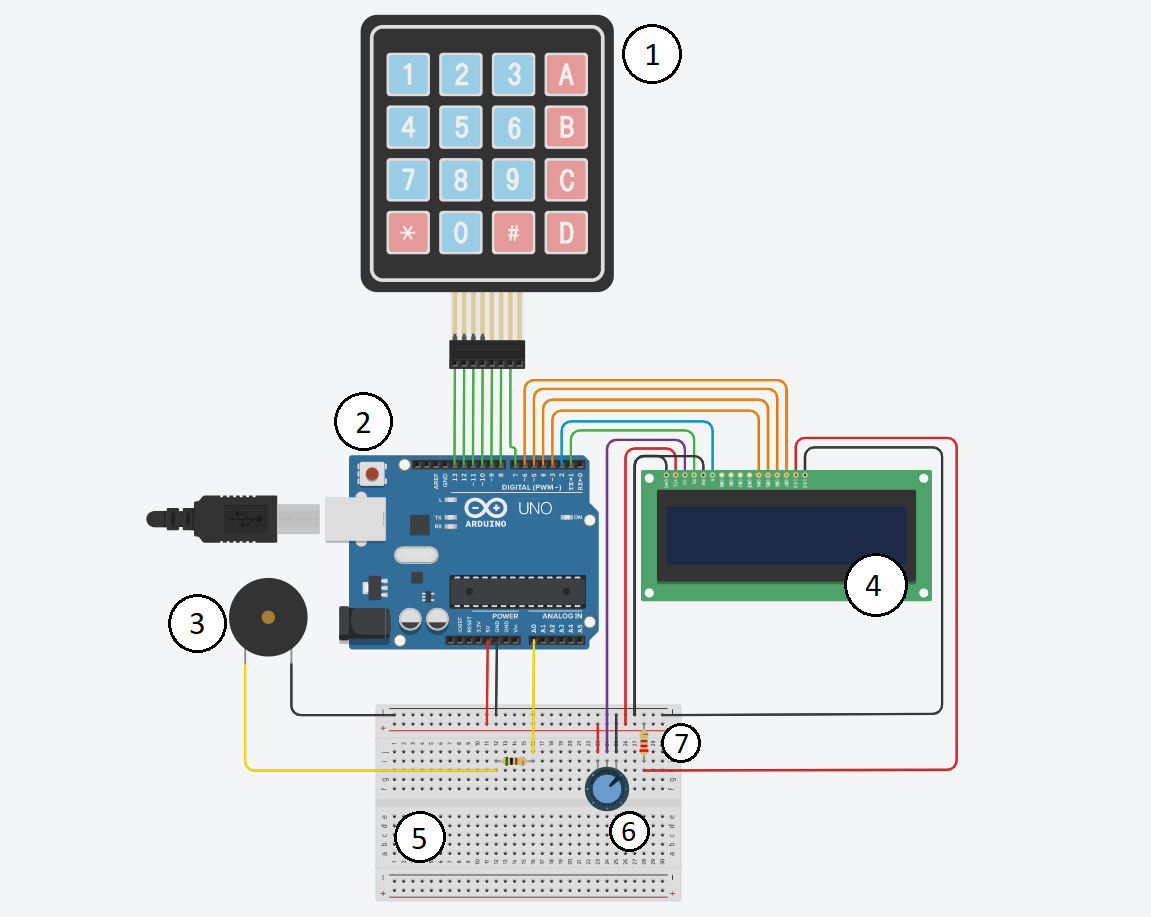
Now for the practical part, in the void loop, after calling out each of the 4 row functions, a nested if statement is responsible for checking if the written pin code matches the constant pin code we have saved. First off, when each button is pressed, the value of the button is saved to one of 4 variables which remain updated according to how many buttons have been pressed consecutively. Once all four have values, an if statement checks if they’re identical to our saved pin code, which incase it is, display “CORRECT You may enter” on the LCD for a second. If not, it displays “WRONG Try again” for a second, then resets how many keys have been pressed so the user could input a new pin code instead. However, with each wrong attempt, the tries are incremented by 1 each time. Once it reaches 3 wrong tries, the LCD would print “INTRUDER ALERT Locked out” permanently, and a buzzer would start beeping at a frequency of 500Hz for a duration of 200ms, repeating it every 300ms until the system is reset.

* + 1. **Connections on Tinker CAD**

1. First, we connected the 5V and ground pins of the Arduino Uno to the breadboard power and ground (red and black wires).
2. For the LCD, we connected the data pins D4, D5, D6, and D7 to the Arduino at pins 3, 4, 5, and 6 respectively to be able to print on the display (orange wires). The ground, R/W, and cathode pins are connected to the ground pin of the breadboard, and both the Vcc and anode of the LCD are connected to the 5V of the breadboard, however, the anode is first connected to a 220-ohm resistor to avoid damaging the screen.
3. The RS pin of the LCD is connected to pin 1 (green wire), and the Enable to pin 2 (blue wire), and finally V0 is connected to a potentiometer (purple wire) which is connected to the 5V and ground of the breadboard.
4. For the keypad, the row pins 1, 2, 3, and 4 are connected to the Arduino pins 13, 12, 11, and 10, respectively. The column pins 1, 2, and 3 are connected to 9, 8, and 7 of the Arduino (green wires).
5. For the buzzer, the positive side is connected to a 500-ohm resistor which is then connected to pin A0 of the Arduino known as digital pin 14 (yellow wire), and the negative side is connected to the ground of the Arduino through the breadboard (black wires).
6. **RESULTS AND DISCUSSION**
   1. **Simulation**

We implemented the circuit on Tinker CAD, adding the code to the Arduino Simulation, obtaining the circuit shown in Figure 8.

**TABLE 1:** LEGEND



**FIGURE 8:** CIRCUIT SCHEMATIC ON TINKER CAD



* 1. **Analysis**

After running the simulation, the circuit showed the LCD displaying each number pressed on the keypad. After we entered the first four numbers, the correctness of the combination was checked. Since we entered the right code, the ‘correct’ message was displayed. We reset the circuit through the reset button on the Arduino Uno then tried again with the wrong key. A ‘wrong, try again’ message was shown. After writing a wrong combination two more times, the system locked the user out and displayed an intruder alert, and making the buzzer go off. A video of the simulation can be shown through [this link](https://drive.google.com/file/d/1J_Iw4zuAIUvujNqnxIdi0t8Wtb7a_atY/view).

1. **CONCLUSION**

The objective of this experiment is to make a security system circuit that checks if a 4-digit key is correct. If the user enters the right pin, they are allowed to enter. Otherwise, they have 3 tries before getting locked out- making a buzzer go off. We coded an Arduino Uno and connected it to a Keypad, LCD screen, and buzzer, which allowed us to make this system functional. Also, we debounced the keypad keys, allowing us to enter the numbers in a reasonable pace and clicking time. The experiment was successful, and the simulation behaved exactly like we programmed it to.

**REFERENCES**

[1] Pololu Robotics & Electronics. *Arduino Uno*, 2020

https://www.pololu.com/product/2191#:~:text=Overview,header%2C%20and%20a%20reset%20button.

[2] Science Buddies. *How to Use a BreadBoard*, 2020

https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard

[3] Circuit Basics, *How to Set Up a Keypad On An Arduino*, 2020

https://www.circuitbasics.com/how-to-set-up-a-keypad-on-an-arduino

[4] Maker Guides, *How to use a 16×2 character LCD with Arduino*, 2020

https://www.makerguides.com/character-lcd-arduino-tutorial/

[5] Electronics Tutorials, *Potentiometers*, 2020

https://www.electronics-tutorials.ws/resistor/potentiometer.html

[6] Adafruit, *Using Piezo Buzzers with CircuitPython & Arduino*, 2018

https://learn.adafruit.com/using-piezo-buzzers-with-circuitpython-arduino

[7] https://www.arduino.cc

**Appendix**

* Arduino Code:

#include <LiquidCrystal.h>

LiquidCrystal lcd(1, 2, 3, 4, 5, 6); // Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)

int c1 = 9, c2 = 8, c3 = 7;

int r1 = 13, r2 = 12, r3 = 11, r4 = 10;

int buzzer = 14;

unsigned long previousDebounceTime = 0;

unsigned long debounceDelay = 50;

int pressed = 0, tries = 0;

int num1, num2, num3, num4, placeholder;

const int code1 = 1, code2 = 5, code3 = 7, code4 = 9;

void setup() {

lcd.begin(16,2); // Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display

lcd.setCursor(0, 0);

lcd.print("Please enter");

lcd.setCursor(0,1);

lcd.print("your pincode.");

delay(1000);

lcd.clear();

pinMode(c1, INPUT\_PULLUP);

pinMode(c2, INPUT\_PULLUP);

pinMode(c3, INPUT\_PULLUP);

pinMode(r1, OUTPUT);

pinMode(r2, OUTPUT);

pinMode(r3, OUTPUT);

pinMode(r4, OUTPUT);

pinMode(buzzer, OUTPUT);

}

void loop() {

row1();

row2();

row3();

row4();

if(pressed == 1)

num1 = placeholder;

else if(pressed == 2)

num2 = placeholder;

else if(pressed == 3)

num3 = placeholder;

else if(pressed == 4){

num4 = placeholder;

delay(200);

if(num1 == code1 && num2 == code2 && num3 == code3 && num4 == code4){

lcd.clear();

lcd.setCursor(0,0);

lcd.print("CORRECT");

lcd.setCursor(0,1);

lcd.print("You may enter.");

delay(1000);

}

else{

tries++;

if(tries < 3){

lcd.clear();

lcd.setCursor(0,0);

lcd.print("WRONG");

lcd.setCursor(0,1);

lcd.print("Try again.");

delay(1000);

lcd.clear();

pressed = 0;

}

else{

lcd.clear();

lcd.setCursor(0,0);

lcd.print("INTRUDER ALERT");

lcd.setCursor(0,1);

lcd.print("Locked out.");

tone(buzzer, 500, 200);

delay(300);

}

}

}

}

void row1() {

digitalWrite(r1, LOW);

digitalWrite(r2, HIGH);

digitalWrite(r3, HIGH);

digitalWrite(r4, HIGH);

if (debounce(c1) == LOW) {

lcd.print("1");

placeholder = 1;

}

else if (debounce(c2) == LOW) {

lcd.print("2");

placeholder = 2;

}

else if (debounce(c3) == LOW) {

lcd.print("3");

placeholder = 3;

}

}

void row2() {

digitalWrite(r1, HIGH);

digitalWrite(r2, LOW);

digitalWrite(r3, HIGH);

digitalWrite(r4, HIGH);

if (debounce(c1) == LOW) {

lcd.print("4");

placeholder = 4;

}

else if (debounce(c2) == LOW) {

lcd.print("5");

placeholder = 5;

}

else if (debounce(c3) == LOW) {

lcd.print("6");

placeholder = 6;

}

}

void row3() {

digitalWrite(r1, HIGH);

digitalWrite(r2, HIGH);

digitalWrite(r3, LOW);

digitalWrite(r4, HIGH);

if (debounce(c1) == LOW) {

lcd.print("7");

placeholder = 7;

}

else if (debounce(c2) == LOW) {

lcd.print("8");

placeholder = 8;

}

else if (debounce(c3) == LOW) {

lcd.print("9");

placeholder = 9;

}

}

void row4() {

digitalWrite(r1, HIGH);

digitalWrite(r2, HIGH);

digitalWrite(r3, HIGH);

digitalWrite(r4, LOW);

if (debounce(c2) == LOW) {

lcd.print("0");

placeholder = 0;

}

}

int debounce(int buttonPin){

int reading = digitalRead(buttonPin);

if (reading == LOW) {

if ((millis() - previousDebounceTime) > debounceDelay) {

previousDebounceTime = millis();

pressed++;

return LOW;

}

}

return HIGH;

}