

Visvesvaraya National Institute of Technology, Nagpur

Department of Electronics and Communication Engineering



Electronic Product Engineering Workshop – Report

Topic: Bluetooth based Hexadecimal-RGB Locker

Team Members:

BT18ECE077 – Kashyap Mishra

BT18ECE079 – Gauree Nadge

BT18ECE086 – Apurva R. Umredkar

BT18ECE087 – Joshin C. Remy

INDEX

1. [Introduction](#)
2. [Components Required](#)
3. [Software Used & Source Codes](#)
4. [Hardware Development Cycle](#)
5. [Conclusion](#)

Introduction

The main objective of our project is to design a locker system based on hexadecimal-RGB system.

This approach was pursued to develop a more secure locker system.




In a standard numeric (decimal) based locker system with say 18 digits, the number of possible combinations for the code are 10^{18} .

In our locker system, with 18 hexadecimal digits the number of combinations increase exponentially to 16^{18} thus increasing security.

Although the security factor increases exponentially, it would be tough for the user to remember 18 hexadecimal digits, hence we split the 18 digits into 3 groups of 6 hexadecimal digits. These 6 hex digits are used to represent an RGB color code. Thus, instead of 18 hex digits the user now must remember only 3 colors he/she wishes to keep as the locker password.

For example:

The user wishes to keep the password as FFA500228B22003366, it will be represented as:

COLOR	HEX	RGB
	FFA500	255, 165, 0
	99BB21	34, 139, 34
	003366	0, 51, 102

Now the user only must remember the colors Orange, Green and Dark Blue (forest green and midnight blue to be specific resp.)

The password can be manually entered through a keypad or using a smartphone via Bluetooth using the app custom made for this project.

The app involves a color wheel which can be used by the user to set the RGB values.

The user gets 5 attempts to feed the correct password. If the burglar runs out of attempts, then a buzzer alarm is set off and the user gets a notification on his/her smartphone about the suspicious activity.

The app has a database where it stores all the activities performed on the locker, including when the locker was opened, closed, an incorrect password was fed etc. The app also gives the user the flexibility to change the locker password, increase or decrease the number of attempts to open the safe or remotely lock/unlock the safe.

This locker system is attached to a mechanical locking system driven by a dc motor and gear train for the purpose of physically locking of the door.

Advantages and Disadvantages

- The locker system can be made more secure just by increasing the number of colors. A 6-color based locker would have 16^{36} combinations which will be way more secure.
- The smartphone app increases the convenience, all the controls are in the hands of the user. This also helps in saving time rather than manually entering 18 hexadecimal values through a keypad.
- Although the approach to the locker system is as secure as traditional decimal number system, it can be difficult for the user to precisely differentiate between shades of a color which can result in them feeding the wrong password.
- The project uses Low Energy Bluetooth module hence the user would have to stay in close range to operate the locker using the app.
- The safety of the locker can be compromised by penetration/hacking through Bluetooth since it is a quite common technology.

Components Required

1. Microcontroller: ATmega32A (Price: Rs 180)



(information source: [Datasheet for ATmega32A](#))

The ATmega32A is a low power, CMOS 8-bit microcontrollers based on the AVR® enhanced RISC architecture. The ATmega32A is a 40/44-pins device with 32 KB Flash, 2 KB SRAM and 1 KB EEPROM. By executing instructions in a single clock cycle, the devices achieve CPU throughput approaching one million instructions per second (MIPS) per megahertz, allowing the system designer to optimize power consumption versus processing speed.

Features:

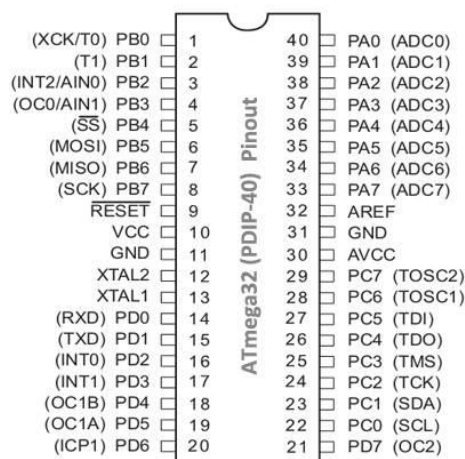
High-performance, Low-power AVR® 8-bit Microcontroller

- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 × 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16MIPS Throughput at 16MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
 - 32Kbytes of In-System Self-programmable Flash program memory
 - 1024Bytes EEPROM
 - 2Kbytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
 - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- QTouch® Library Support
 - Capacitive touch buttons, sliders, and wheels
 - QTouch and QMatrix™ acquisition
 - Up to 64 sense channels
- Peripheral Features

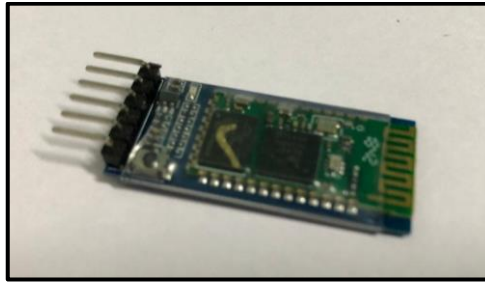
- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Four PWM Channels
- 8-channel, 10-bit ADC
- 8 Single-ended Channels
- 7 Differential Channels in TQFP Package Only
- 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
 - Byte-oriented Two-wire Serial Interface
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
 - 32 Programmable I/O Lines
 - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
 - 2.7V - 5.5V
- Speed Grades
 - 0 - 16MHz
- Power Consumption at 1MHz, 3V, 25°C
 - Active: 0.6mA
 - Idle Mode: 0.2mA
 - Power-down Mode: < 1µA

This microcontroller is the heart of our project and governs the functioning of our locker system. All the other modules and ICs are connected to the IO pins of ATmega32A. Since this microcontroller is manufactured by AVR, its source code can be easily written in Arduino IDE by importing the necessary board information and libraries.

Pin Diagram for ATmega32/ATmega32A



2. HC-05 Bluetooth Module (Price: Rs 300)



(information source: Datasheet for HC-05)

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle

Hardware Features:

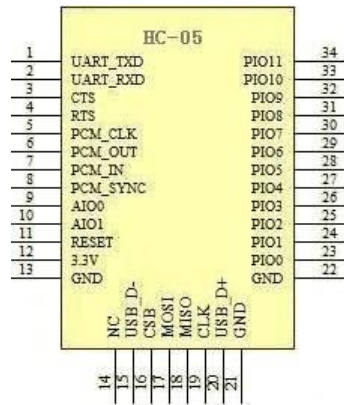
- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software Features:

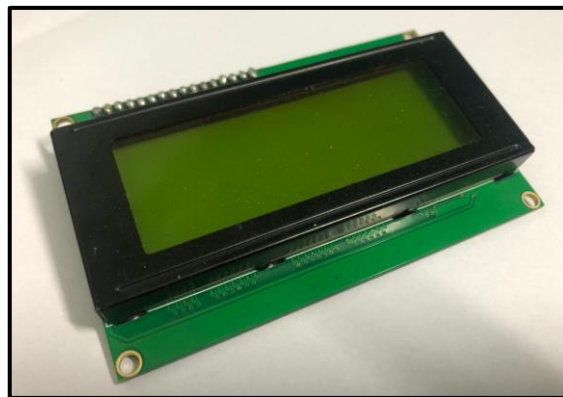
- Default Baud rate: 38400, Data bits:8, Stop bit:1, Parity: No parity, Data control: has.
- Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected.
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1 time/2s in interval, while disconnected only blue led blinks 2 times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected because of beyond the range of connection.

This HC-05 module is used for the communication between the smartphone app and our microcontroller. The app will send commands over Bluetooth which will be processed by ATmega32A with the help of the source code we developed.

Pin Diagram of HC-05

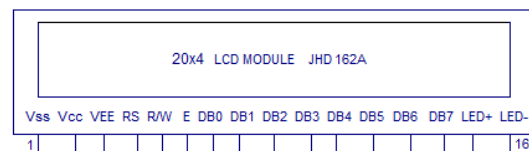


3. LCD 20x4 (Price: Rs 390)



This display module consists of 20 columns and 4 rows and each cell can display an ASCII character and will be controlled using ATmega32A. We used this module to display the password and messages such as lock/unlock, correct/incorrect password etc.

LCD Module - PIN Out Diagram



Pin 16 - Ground - Backlight LED
 Pin 15 - +5V - Connect through a current limiting resistor
 DB0 - DB7 - Data Pins - Use only 4 Data pins in 4 Bit Mode
 E - Enable Pin
 R/W - Read/Write Mode
 RS - Register Select
 VEE - Contrast Adjustment - Connect through a Potentiometer
 Vcc - +5V - Power to LCD
 Vss - Ground

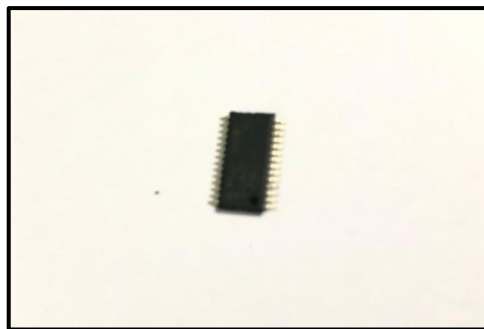
Note: This pin out diagram is same and common for many line LCD modules like 16x1, 16x2, 16x4, 8x1, 8x2, 20x1, 20x2, 20x4, 40x2, and other types of Line LCD Modules making use of the Hitachi Driver.

4. RGB LEDs (Price: Rs 10/piece *10)



This LED has 4 terminals, namely R, G, B and Ground for common cathode type. Theoretically, if each value of R, G and B are varied between 0-255 using PWM pins of ATmega32A, then it is possible to achieve a total of 16777216 ($256^3 = 16^6$) possible colors on a single LED. Full saturation of R, G & B gives white color and zero saturation will technically turn off the LED but can be referred to as black color. We require 3 of these for our project for indicating the 3 different colors.

5. IC PCA9685 (Price: 300)



(information source: [Datasheet for PCA9685](#))

The PCA9685 is an I2C-bus controlled 16-channel LED controller optimized for Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 24 Hz to 1526 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency.

Each LED output can be off or on (no PWM control) or set at its individual PWM controller value. The LED output driver is programmed to be either open drain with a 25mA current sink capability at 5 V or totem pole with a 25mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum number of discrete components for larger current or higher voltage LEDs.

The PCA9685 is in the new Fast-mode Plus (Fm+) family. Fm+ devices offer higher frequency (up to 1 MHz) and more densely populated bus operation (up to 4000 pF).

Although the PCA9635 and PCA9685 have many similar features, the PCA9685 has some unique features that make it more suitable for applications such as LCD or LED backlighting and Ambilight:

- The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels. This feature is not available in PCA9635.
- The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).
- When multiple LED controllers are incorporated in a system, the PWM pulse widths between multiple devices may differ if PCA9635s are used. The PCA9685 has a programmable prescaler to adjust the PWM pulse widths of multiple devices.
- The PCA9685 has an external clock input pin that will accept user-supplied clock (50 MHz max.) in place of the internal 25 MHz oscillator. This feature allows synchronization of multiple devices. The PCA9635 does not have external clock input feature.
- Like the PCA9635, PCA9685 also has a built-in oscillator for the PWM control. However, the frequency used for PWM control in the PCA9685 is adjustable from about 24 Hz to 1526 Hz as compared to the typical 97.6 kHz frequency of the PCA9635. This allows the use of PCA9685 with external power supply controllers. All bits are set at the same frequency.
- The Power-On Reset (POR) default state of LED n output pins is LOW in the case of PCA9685. It is HIGH for PCA9635.

The active LOW Output Enable input pin (OE) allows asynchronous control of the LED outputs and can be used to set all the outputs to a defined I2C-bus programmable logic state. The OE can also be used to externally 'pulse width modulate' the outputs, which is useful when multiple devices need to be dimmed or blinked together using software control.

Software programmable LED All Call and three Sub Call I2C-bus addresses allow all or defined groups of PCA9685 devices to respond to a common I2C-bus address, allowing for example, all red LEDs to be turned on or off at the same time or marquee chasing effect, thus minimizing I2C-bus commands. Six hardware address pins allow up to 62 devices on the same bus.

The Software Reset (SWRST) General Call allows the master to perform a reset of the PCA9685 through the I2C-bus, identical to the Power-On Reset (POR) that initializes the registers to their default state causing the outputs to be set LOW. This allows an easy and quick way to reconfigure all device registers to the same condition via software.

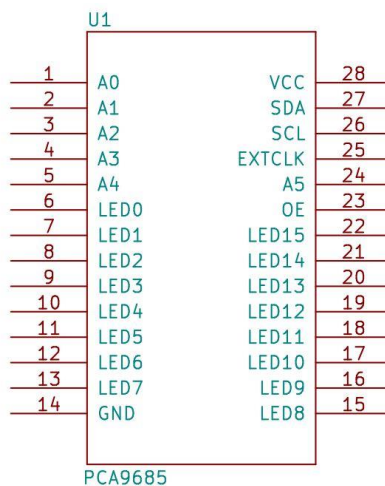
Features:

- 16 LED drivers. Each output programmable at:
 - Off
 - On
 - Programmable LED brightness
 - □ Programmable LED turn-on time to help reduce EMI
- 1 MHz Fast-mode Plus compatible I2C-bus interface with 30 mA high drive capability on SDA output for driving high capacitive buses

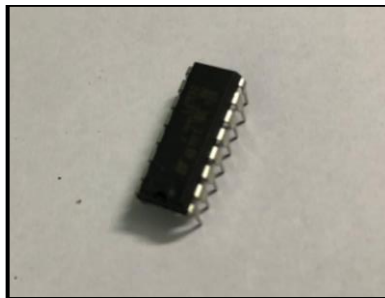
- 4096-step (12-bit) linear programmable brightness per LED output varying from fully off (default) to maximum brightness
- LED output frequency (all LEDs) typically varies from 24 Hz to 1526 Hz (Default of 1Eh in PRE_SCALE register results in a 200 Hz refresh rate with oscillator clock of 25 MHz).
- Sixteen totem pole outputs (sink 25 mA and source 10 mA at 5 V) with software programmable open-drain LED outputs selection (default at totem pole). No input functions.
- Output state change programmable on the Acknowledge or the STOP Command to update outputs byte-by-byte or all at the same time (default to 'Change on STOP').
- Active LOW Output Enable (OE) input pin. LED n outputs programmable to logic 1, logic 0 (default at power-up) or 'high-impedance' when OE is HIGH.
- 6 hardware address pins allow 62 PCA9685 devices to be connected to the same I2C-bus
- Toggling OE allows for hardware LED blinking
- 4 software programmable I2C-bus addresses (one LED All Call address and three LED Sub Call addresses) allow groups of devices to be addressed at the same time in any combination (for example, one register used for 'All Call' so that all the PCA9685s on the I2C-bus can be addressed at the same time and the second register used for three different addresses so that 1/3 of all devices on the bus can be addressed at the same time in a group). Software enable and disable for these I2C-bus address.
- Software Reset feature (SWRST General Call) allows the device to be reset through the I2C-bus
- 25 MHz typical internal oscillator requires no external components
- External 50 MHz (max.) clock input
- Internal power-on reset
- Noise filter on SDA/SCL inputs
- Edge rate control on outputs
- No output glitches on power-up
- Supports hot insertion
- Low standby current
- Operating power supply voltage range of 2.3 V to 5.5 V
- 5.5 V tolerant inputs
- 40°C to +85°C operation
- ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per JESD22-A115 and 1000 V CDM per JESD22-C101
- Latch-up testing is done to JEDEC Standard JESD78 which exceeds 100 mA
- Packages offered: TSSOP28, HVQFN28

Since our ATmega32A has only 4 PWM pins and our project requires 9 PWM outputs for controlling the RGB LEDs, this IC is used as a PWM Expander.

Pin Diagram for PCA9685



6. L293D Motor Driver IC (Price: Rs 30)



(information source: [Datasheet for L293D](#))

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC, and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

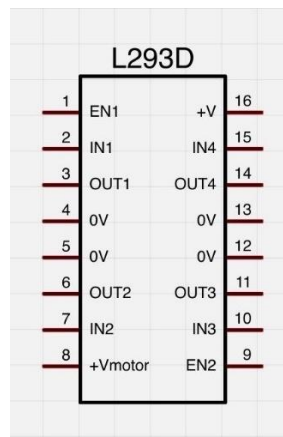
Features:

- Featuring Unitrode L293 and L293D
- Products Now from Texas Instruments
- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply

- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functionally Similar to SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

This motor driver IC will control the direction of the standard DC motor which we have applied in the mechanical locking system.

Pin Diagram of L293D



7. Standard 5V DC Motor (Price: Rs 20)

This component is required for driving the gears and locking mechanism.

8. Gears: Rack & Pinion (Price: Rs 100)

These gears in combinations provides linear motion which is required for our locking mechanism.

9. Resistors (Price: Re 1/piece)

- 82Ω x2
- 330Ω x9

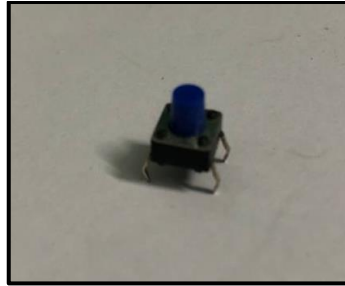
10. Capacitors (Price: Re 1/piece)

- 22pF x2

11. 16 MHz Crystal

For the external clock.

12. Tactile Push Buttons *(Price Rs 3/piece)*



For the keypad and reset buttons. It has two channels and a momentary push mechanical action.

13. Arduino Uno *(Spare)*

This development board uses ATmega328p as its microcontroller. We used this device for initial prototyping and as a substitute for burning the code to our main microcontroller ATmega32A.

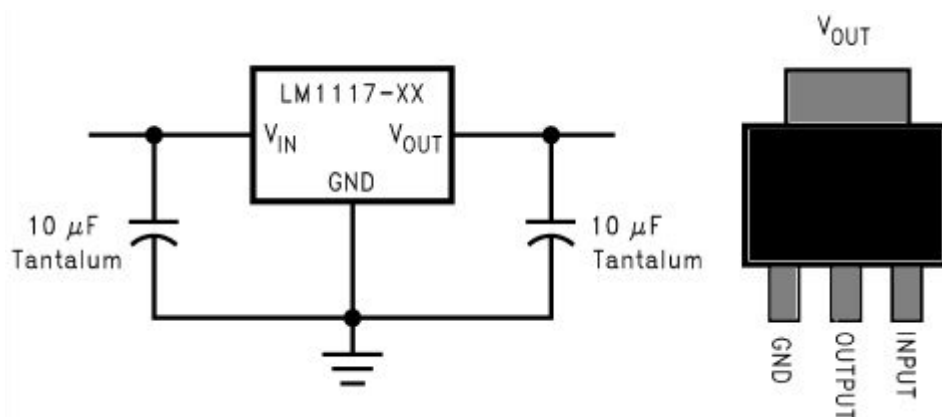
14. 12V DC Adapter *(Price: Rs 180)*

For supplying power to our circuit.

15. IC LM1117 *(Price: Rs 10)*

Since ATmega32A can have a maximum supply voltage as 5.5V, we need to use a voltage regulator. LM1117 is a 12V to 3.3V voltage regulator IC.

The 10 μ F capacitors are used for smoothening the output voltage, to avoid ripples. These are optional for our project since it is a low voltage application.



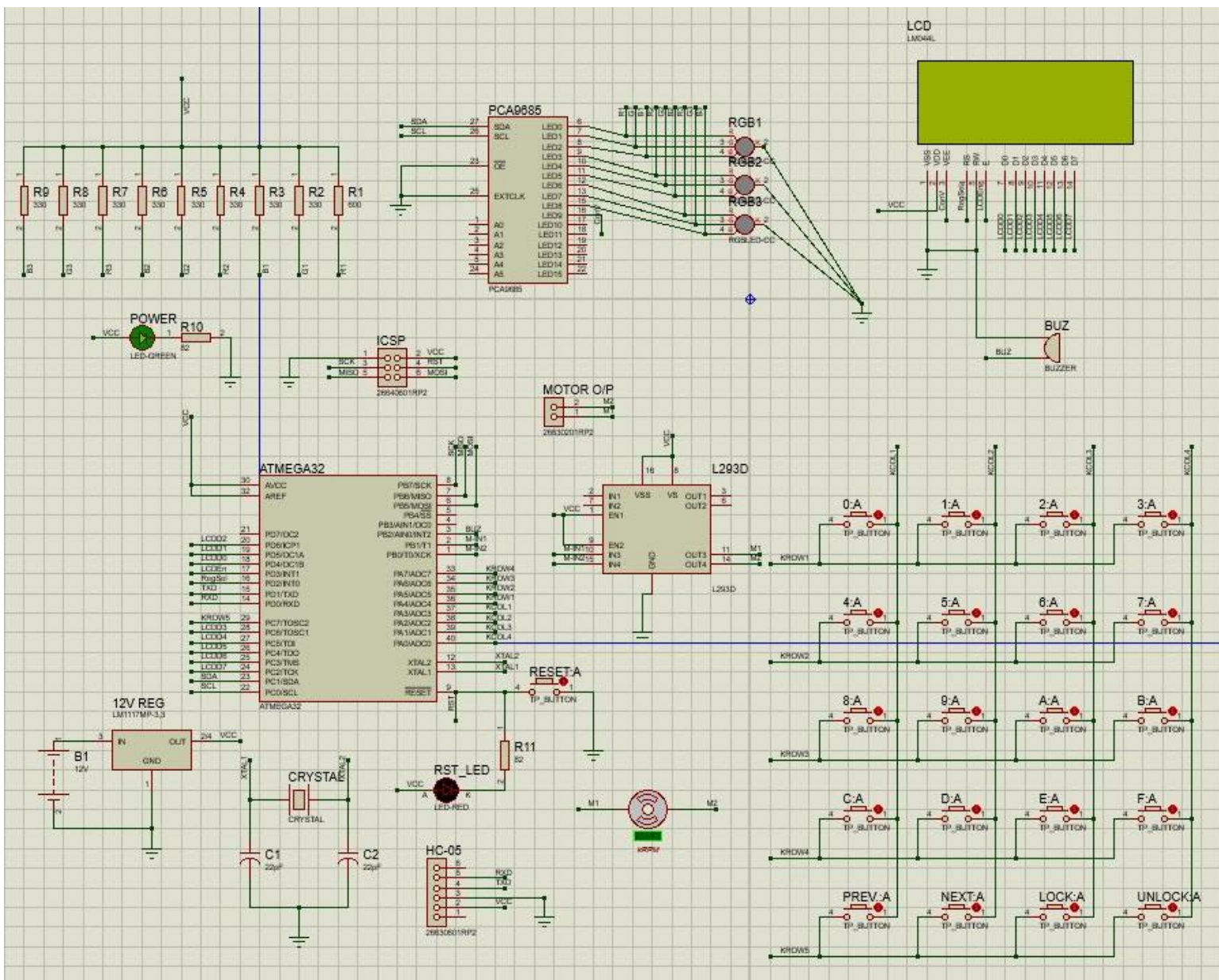
Software Used & Source Codes

Various software, IDEs were used in the development and interfacing of this project.

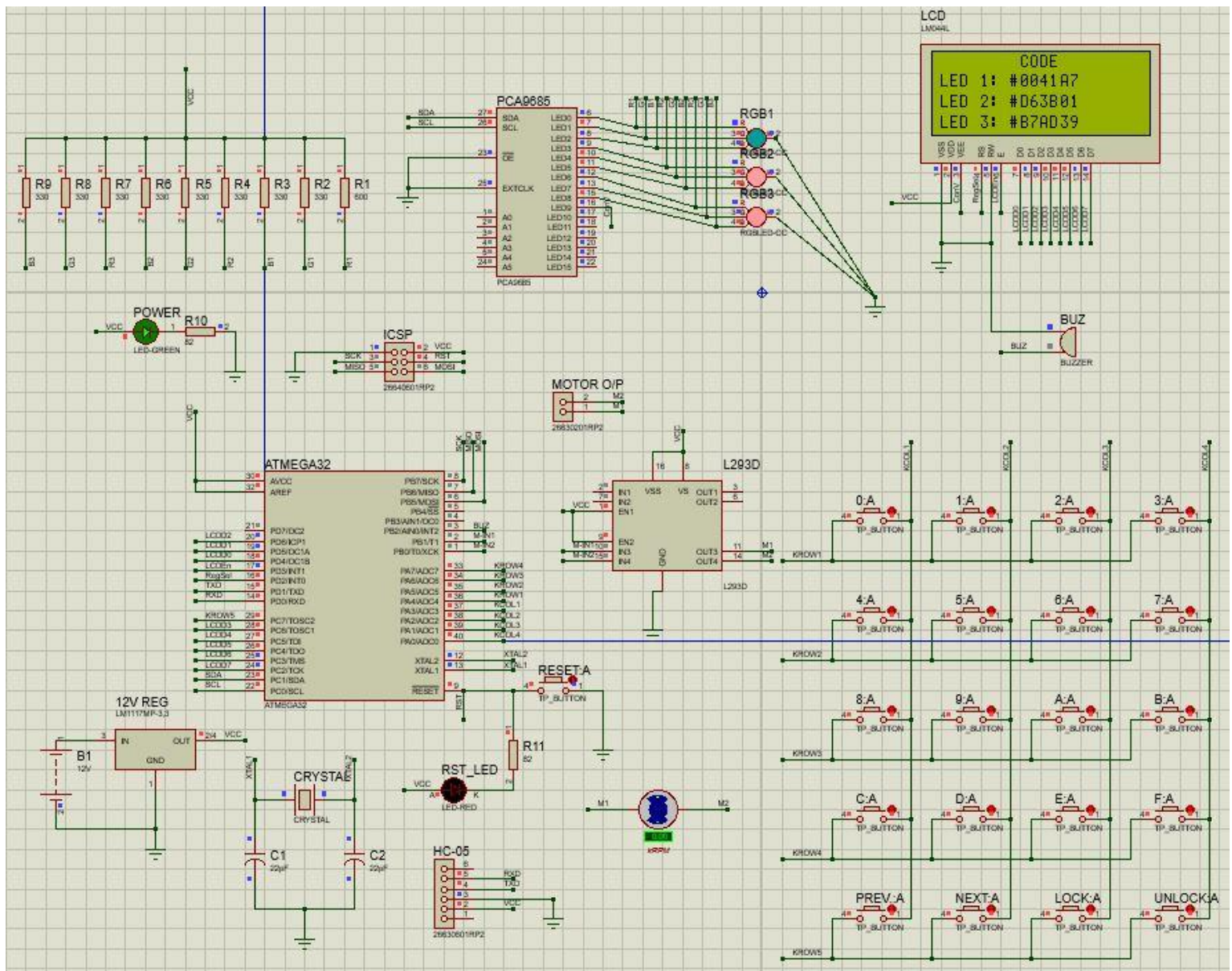
1. Proteus 8 Professional

This software was used for the virtual simulation of the circuit and designing the PCB for our project.

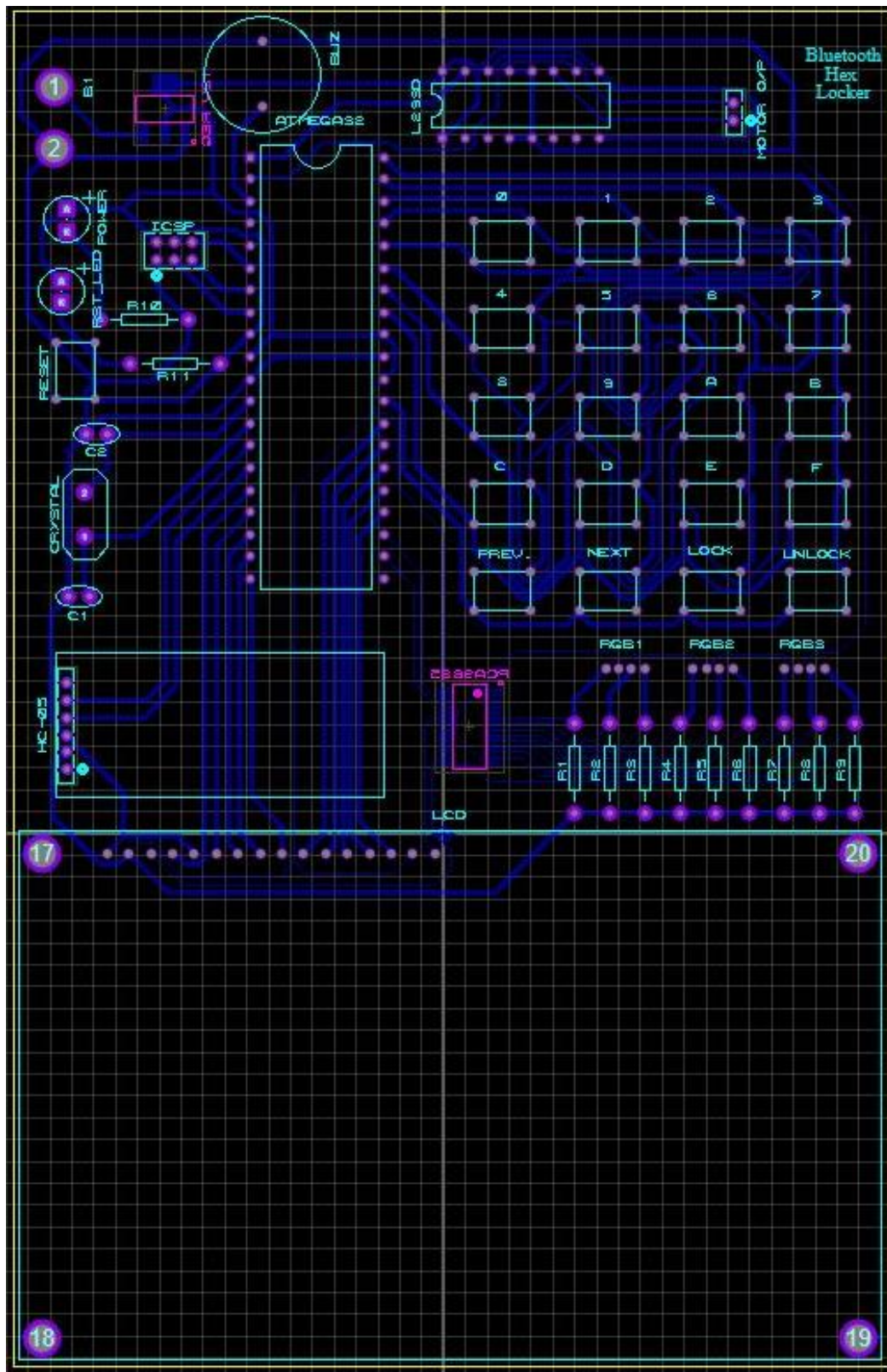
Circuit Diagram



Simulation



PCB Design



2. Arduino IDE

Although there is no option to write the code for ATmega32/ATmega32A in Arduino IDE, we can import the board under the “Board Manager” tab. The board information is imported from the GitHub repository created by MightyCoreMaster. We imported the libraries such as LiquidCrystal.h, SoftwareSerial.h, Adafruit_PWMServoDriver.h to interface with various modules we used in the project.

Source Code:

```
/* Bluetooth Hex Locker Code
   Author: Apurva Umredkar*/

//Defining all the pins and importing all the required libraries

#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver rgb = Adafruit_PWMServoDriver();
#define pwmR1 0
#define pwmG1 1
#define pwmB1 2
#define pwmR2 3
#define pwmG2 4
#define pwmB2 5
#define pwmR3 6
#define pwmG3 7
#define pwmB3 8
int r1 = 0, r2 = 0, r3 = 0, g1 = 0, g2 = 0, g3 = 0, b1 = 0, b2 = 0, b3 = 0;

//LCD Pins

#define L0 12
#define L1 13
#define L2 14
#define L3 22
#define L4 21
#define L5 20
#define L6 19
#define L7 18
#define RS 10
#define LCDEN 11

#include <LiquidCrystal.h>
LiquidCrystal lcd(RS, LCDEN, L0, L1, L2, L3, L4, L5, L6, L7);

//Keypad Pins

#define KR1 27
#define KR2 26
#define KR3 25
#define KR4 24
```

```

#define KR5 23
#define KC1 28
#define KC2 29
#define KC3 30
#define KC4 31

int i, j, key;
int krow[] = {KR1, KR2, KR3, KR4, KR5};
int kcol[] = {KC1, KC2, KC3, KC4};

//Buzzer pins
#define BUZ 2

//PASSWORD
String p1 = "222222";
String p2 = "222222";
String p3 = "222222";

//Current Code
String c1, c2, c3;
int digitcount = -1, attempts = 5, stat = 0;

void setup() {
  // put your setup code here, to run once:
  lcd.begin(20, 4);
  lcd.clear();
  for (i = 0; i < 5; i++)
    pinMode(krow[i], OUTPUT);
  for (i = 0; i < 4; i++)
  {
    pinMode(kcol[i], INPUT);
    digitalWrite(kcol[i], HIGH);
  }

  //initial random password
  randomSeed(analogRead(30));
  randomizePass();
  rgb.begin();
  rgb.setPWMFreq(1600);
  dispRGB();
}

void loop() {
  // put your main code here, to run repeatedly:

  for (i = 0; i < 5; i++)
  {
    digitalWrite(krow[0], HIGH);
    digitalWrite(krow[1], HIGH);
    digitalWrite(krow[2], HIGH);

```

```

digitalWrite(krow[3], HIGH);
digitalWrite(krow[4], HIGH);
digitalWrite(krow[i], LOW);
for (j = 0; j < 4; j++)
{
    key = digitalRead(kcol[j]);
    if (key == LOW)
    {
        if (digitcount == 0)
            c1 = "000000";
        else if (digitcount == 6)
            c2 = "000000";
        else if (digitcount == 12)
            c3 = "000000";
        keypress(i, j);
        digitcount == 18 ? digitcount = -1 : digitcount++;
        lcdText();
        delay(50);
    }
}
}
}
}

```

```

void keypress(int x, int y)
{
    if (i == 0 && j == 0)
    {
        if (digitcount < 6)
        {
            c1[digitcount] = '0';
        }
        else if (digitcount < 12 && digitcount > 5)
        {
            c2[digitcount - 6] = '0';
        }
        if (digitcount < 18 && digitcount > 11)
        {
            c3[digitcount - 12] = '0';
        }
        lcdText();
        dispRGB();
    }
    if (i == 0 && j == 1)
    {
        if (digitcount < 6)
        {
            c1[digitcount] = '1';
        }
        else if (digitcount < 12 && digitcount > 5)
        {
            c2[digitcount - 6] = '1';
        }
    }
}

```

```

}
if (digitcount < 18 && digitcount > 11)
{
    c3[digitcount - 12] = '1';
}
lcdText();
dispRGB();
}
if (i == 0 && j == 2)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '2';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '2';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '2';
    }
    lcdText();
    dispRGB();
}
if (i == 0 && j == 3)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '3';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '3';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '3';
    }
    lcdText();
}
if (i == 1 && j == 0)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '4';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '4';
    }
}

```

```

if (digitcount < 18 && digitcount > 11)
{
    c3[digitcount - 12] = '4';
}
lcdText();
dispRGB();
}
if (i == 1 && j == 1)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '5';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '5';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '5';
    }
    lcdText();
    dispRGB();
}
if (i == 1 && j == 2)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '6';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '6';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '6';
    }
    lcdText();
    dispRGB();
}
if (i == 1 && j == 3)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '7';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '7';
    }
}

```

```

if (digitcount < 18 && digitcount > 11)
{
    c3[digitcount - 12] = '7';
}
lcdText();
dispRGB();
}
if (i == 2 && j == 0)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '8';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '8';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '8';
    }
    lcdText();
    dispRGB();
}
if (i == 2 && j == 1)
{
    if (digitcount < 6)
    {
        c1[digitcount] = '9';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = '9';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = '9';
    }
    lcdText();
    dispRGB();
}
if (i == 2 && j == 2)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'A';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'A';
    }
}

```

```

if (digitcount < 18 && digitcount > 11)
{
    c3[digitcount - 12] = 'A';
}
lcdText();
dispRGB();
}
if (i == 2 && j == 3)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'B';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'B';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = 'B';
    }
    lcdText();
    dispRGB();
}
if (i == 3 && j == 0)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'C';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'C';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = 'C';
    }

    lcdText();
    dispRGB();
} if (i == 3 && j == 1)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'D';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'D';
    }
}

```



```

if (digitcount < 18 && digitcount > 11)
{
    c3[digitcount - 12] = 'D';
}
lcdText();
dispRGB();
}
if (i == 3 && j == 2)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'E';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'E';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = 'E';
    }
    lcdText();
    dispRGB();
}
if (i == 3 && j == 3)
{
    if (digitcount < 6)
    {
        c1[digitcount] = 'F';
    }
    else if (digitcount < 12 && digitcount > 5)
    {
        c2[digitcount - 6] = 'F';
    }
    if (digitcount < 18 && digitcount > 11)
    {
        c3[digitcount - 12] = 'F';
    }
    lcdText();
    dispRGB();
}
if (i == 4 && j == 0)
{
    if (digitcount < 6)
    {
        digitcount = -1;
        c1 = "000000";
        lcdText();
        dispRGB();
    }
    else if (digitcount > 5 && digitcount < 12)

```

```

{
    digitcount = -1;
    c1 = "000000";
    lcdText();
    dispRGB();
}
else if (digitcount > 11 && digitcount < 18)
{
    digitcount = 5;
    c2 = "000000";
    lcdText();
    dispRGB();
}
}
if (i == 4 && j == 1)
{
    if (digitcount < 6)
    {
        digitcount = 5;
        c2 = "000000";
        lcdText();
        dispRGB();
    }
    else if (digitcount > 5 && digitcount < 12)
    {
        digitcount = 11;
        c3 = "000000";
        lcdText();
        dispRGB();
    }
    else if (digitcount > 11 && digitcount < 18)
    {
        digitcount = 11;
        c3 = "000000";
        lcdText();
        dispRGB();
    }
}
if (i == 4 && j == 2)
{
    lcd.clear();
    lcd.print("Status: LOCKED");
    rgb.setPWM(pwmR1, 4096, 0);
    rgb.setPWM(pwmG1, 0, 0);
    rgb.setPWM(pwmB1, 0, 0);
    rgb.setPWM(pwmR2, 4096, 0);
    rgb.setPWM(pwmG2, 0, 0);
    rgb.setPWM(pwmB2, 0, 0);
    rgb.setPWM(pwmR3, 4096, 0);
    rgb.setPWM(pwmG3, 0, 0);
    rgb.setPWM(pwmB3, 0, 0);
}

```

```

delay(250);
if(stat!=0)
    attempts = 5;
randomizePass();
}

if (i == 4 && j == 3)
{
    if (p1 == c1 && p2 == c2 && p3 == c3)
    {
        lcd.clear();
        lcd.print("Status: UNLOCKED");
        rgb.setPWM(pwmR1, 0, 0);
        rgb.setPWM(pwmG1, 4096, 0);
        rgb.setPWM(pwmB1, 0, 0);
        rgb.setPWM(pwmR2, 0, 0);
        rgb.setPWM(pwmG2, 4096, 0);
        rgb.setPWM(pwmB2, 0, 0);
        rgb.setPWM(pwmR3, 0, 0);
        rgb.setPWM(pwmG3, 4096, 0);
        rgb.setPWM(pwmB3, 0, 0);
    }
    else
    {
        rgb.setPWM(pwmR1, 4096, 0);
        rgb.setPWM(pwmG1, 0, 0);
        rgb.setPWM(pwmB1, 0, 0);
        rgb.setPWM(pwmR2, 4096, 0);
        rgb.setPWM(pwmG2, 0, 0);
        rgb.setPWM(pwmB2, 0, 0);
        rgb.setPWM(pwmR3, 4096, 0);
        rgb.setPWM(pwmG3, 0, 0);
        rgb.setPWM(pwmB3, 0, 0);

        lcd.clear();
        lcd.print("INCORRECT CODE!");
        attempts--;
        if (attempts > 0)
        {
            lcd.setCursor(0,1);
            lcd.print(String(attempts) + " ATTEMPTS REMAINING");
            digitalWrite(BUZ, HIGH);
        }
        else
        {
            lcd.clear();
            lcd.print("INTRUSION DETECTED");
            lcd.setCursor(0, 1);
            lcd.print("DISABLING LOCKER FOR 2 MINUTES");
            for (int i = 0; i <= 240; i++)
            {

```

```

        digitalWrite(BUZ, HIGH);
        delay(250);
        digitalWrite(BUZ, LOW);
        delay(250);
    }
}
}
delay(250);
randomizePass();
lcdText();
dispRGB();
digitalWrite(BUZ, LOW);
}
}

```

```

void hex2rgb(String s, int *r, int *g, int *b) {
    int n[6];
    for (int i = 0; i < 6; i++) {
        switch (s[i]) {
            case '0': n[i] = 0;
                break;
            case '1': n[i] = 1;
                break;
            case '2': n[i] = 2;
                break;
            case '3': n[i] = 3;
                break;
            case '4': n[i] = 4;
                break;
            case '5': n[i] = 5;
                break;
            case '6': n[i] = 6;
                break;
            case '7': n[i] = 7;
                break;
            case '8': n[i] = 8;
                break;
            case '9': n[i] = 9;
                break;
            case 'A': n[i] = 10;
                break;
            case 'B': n[i] = 11;
                break;
            case 'C': n[i] = 12;
                break;
            case 'D': n[i] = 13;
                break;
            case 'E': n[i] = 14;
                break;
            case 'F': n[i] = 15;
                break;
        }
    }
}

```

```

    }
}
*r = 16 * n[0] + n[1];

*g = 16 * n[2] + n[3];

*b = 16 * n[4] + n[5];
}

```

```

String dec2hex(long int n) {
    if (n == 0)
        return "0";
    String h = "";
    while (n > 0) {
        int x = n % 16;
        switch (x) {
            case 10: h.concat("A");
                break;
            case 11: h.concat("B");
                break;
            case 12: h.concat("C");
                break;
            case 13: h.concat("D");
                break;
            case 14: h.concat("E");
                break;
            case 15: h.concat("F");
                break;
            default: h.concat(x);
        }
        n /= 16;
    }
}

```

```

    if (h.length() < 6)
        while (h.length() < 6)
            h.concat("0");
    String x = "";
    int f = h.length() - 1;
    while (f >= 0) {
        x.concat(h[f]);
        f--;
    }
}

```

```

    return x;
}

```

```

void lcdText()
{
    lcd.setCursor(8, 0);
    lcd.print("CODE");
    lcd.setCursor(0, 1);
}

```

```

    lcd.print("LED 1: #" + c1);
    lcd.setCursor(0, 2);
    lcd.print("LED 2: #" + c2);
    lcd.setCursor(0, 3);
    lcd.print("LED 3: #" + c3);
}

void dispRGB()
{
    hex2rgb(c1, &r1, &g1, &b1);
    hex2rgb(c2, &r2, &g2, &b2);
    hex2rgb(c3, &r3, &g3, &b3);
    rgb.setPWM(pwmR1, r1, 0);
    rgb.setPWM(pwmG1, g1, 0);
    rgb.setPWM(pwmB1, b1, 0);
    rgb.setPWM(pwmR2, r2, 0);
    rgb.setPWM(pwmG2, g2, 0);
    rgb.setPWM(pwmB2, b2, 0);
    rgb.setPWM(pwmR3, r3, 0);
    rgb.setPWM(pwmG3, g3, 0);
    rgb.setPWM(pwmB3, b3, 0);
}

void randomizePass()
{
    digitcount = -1;
    lcd.clear();
    c1 = dec2hex(random(0, 16777215));
    c2 = dec2hex(random(0, 16777215));
    c3 = dec2hex(random(0, 16777215));
    lcdText();
    dispRGB();
}

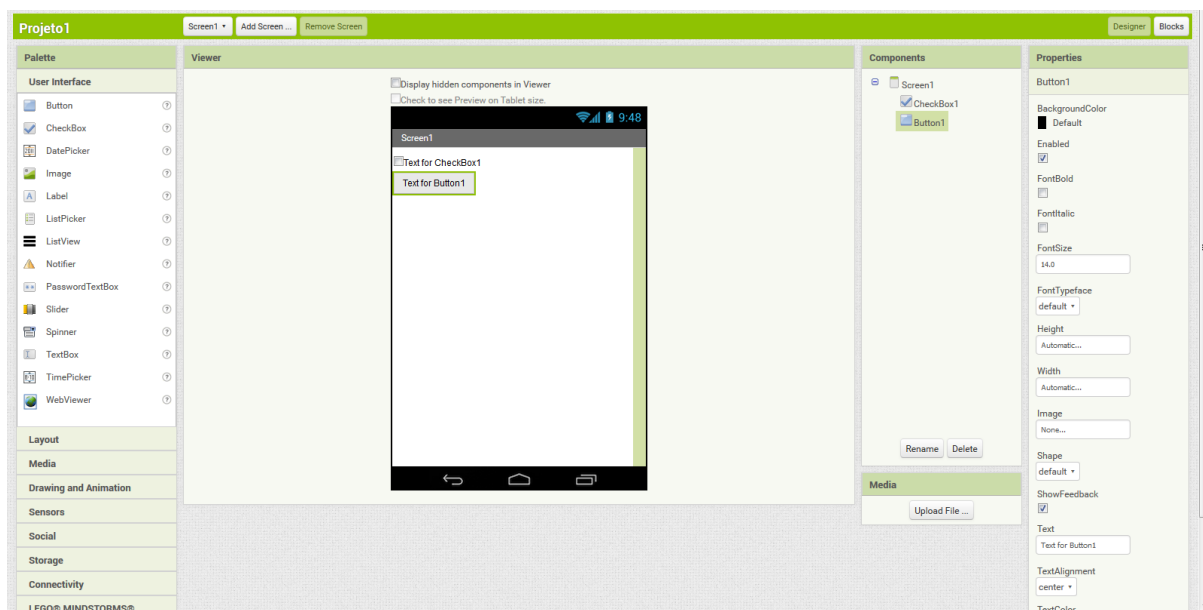
```

3. MIT App Inventor

The app is designed using MIT App Inventor. MIT App Inventor is a web application integrated development environment originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). It allows newcomers to computer programming to create application software (apps).

It uses a graphical user interface (GUI) very similar to the programming languages Scratch (programming language) and the StarLogo, which allows users to drag and drop visual objects to create an application that can run on android devices.

Here is a preview of the app:



The purpose of making an app was so that the user can reset the password of the locker incase if he/she forgets the same using a user-friendly color wheel that corresponds to its respective hexadecimal value. The app has a database where it stores all the activities performed on the locker, including when the locker was opened, closed, an incorrect password was fed etc.

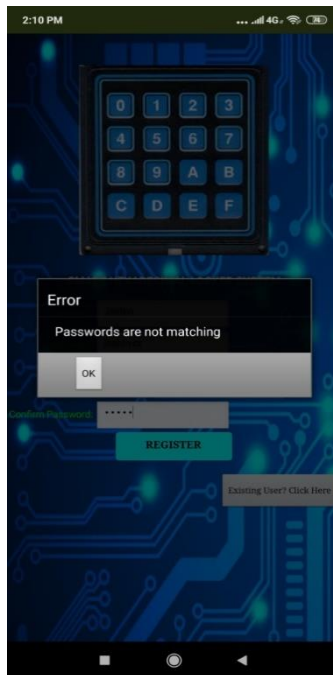
The app contains 4 screens in total.

- Screen 1: User Registration Page
- Screen 2: Login Page
- Screen 3: Pop up Menu Bar
- Screen 4: Resetting Password using RGB Color Wheel

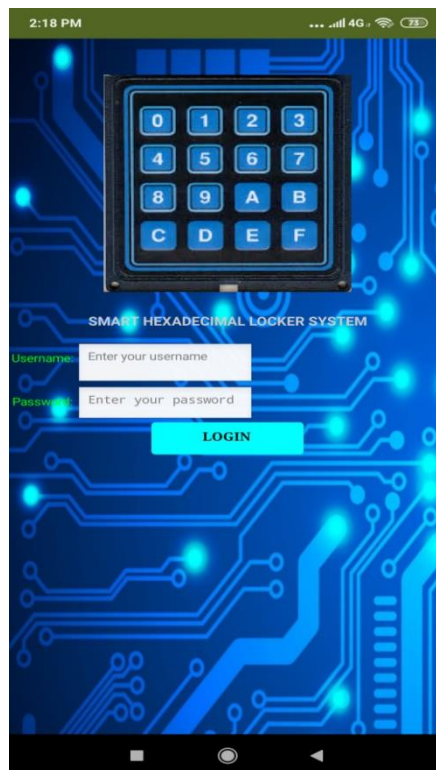
Screen 1



The system allows a one-time User Registration containing basic requirements for suitable username, password etc. and verifying the same. Example:



Screen 2



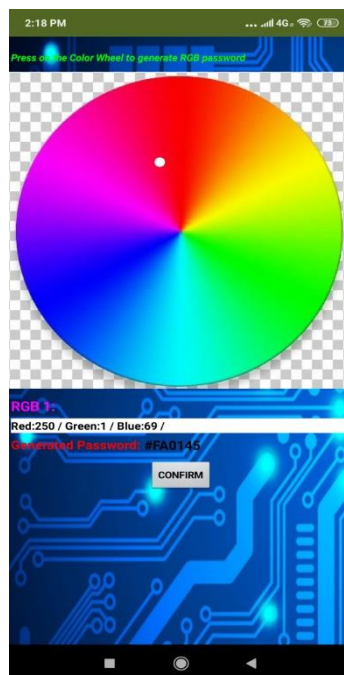
Login page verifies the credentials entered during the Registration thus allowing the user to make changes regarding to locker system.

Screen 3



Pop up Menu Bar to choose respective task for the locker.

Screen 4

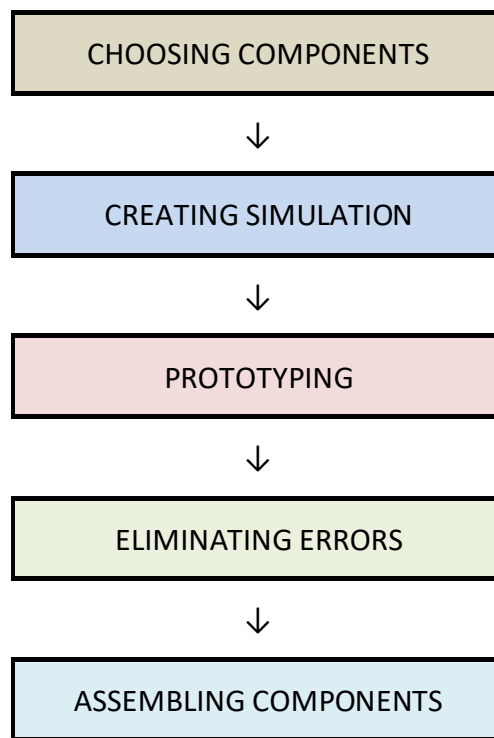


User friendly color wheel where user can change password of the locker using simple touches on the wheel. The pixel value is converted to its respective hexadecimal value and this process is repeated for all 3 RGB's.

HARDWARE DEVELOPMENT CYCLE

Here we shall break down the journey of developing this project from the scratch to the product i.e. from breadboard to PCB.

FLOWCHART



1. CHOOSING THE COMPONENTS

The components must be chosen such that they satisfy all the requirements for the project and at the same time are readily available in the market at a reasonable price. While most of the components were purchased from the local electronics market, few components were purchased online such as the PCA9685 IC.

2. CREATING A SIMULATION

Before assembling the project on a breadboard or soldering them directly to the PCB, it is important to simulate the working of the circuits so that a basic idea is derived. This helps in preventing basic mistakes and also allows us to test different scenarios which can help us to choose/replace components according to our need.

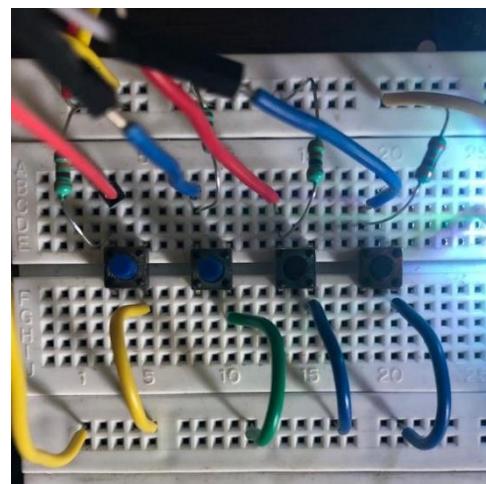
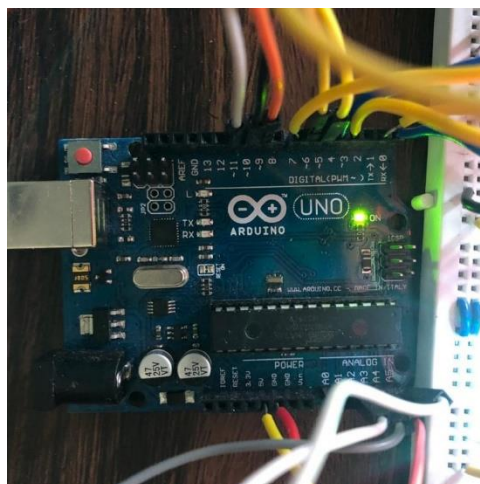
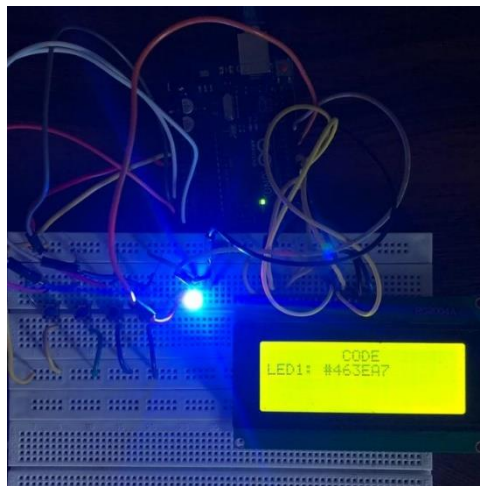
For simulation purposes, the software Proteus 8 Professional was used. It also has a vast library which helped us in choosing the required components. The simulation and circuit diagram are attached above (under the chapter [Software Used and Source Codes](#)).

3. PROTOTYPING

Before we assembled the final PCB, it was important to know whether we get the output in real life hardware since if there is any error it would be very difficult to desolder the components from the PCB and redesign and reprint the PCB.

Hence, we initially tested our project on a breadboard. The project is too big and requires a lot of connections, so we made the connections for only 1 RGB LED, meaning a password of only 6 hex digits. We also did the initial testing without a keypad and instead used only two buttons as a means of incrementing and decrementing the hex values.

This possessed a lot of challenges. Even on a single RGB LED, without the keypad the user would have to cycle through 16 million+ values for feeding the password which is time consuming.



The Locking Mechanism:

Our initial approach was a physical lock design using a tower lock. The components required were a pair of nut and bolt, tower lock, standard dc motor and a frame designed using ice cream sticks to support the lock



(concept of initial locking mechanism, source: YouTube @TheWrench)

Design: The bolt is stuck to tower lock's handle. The nut is fitted in bolt which is attached to motor.

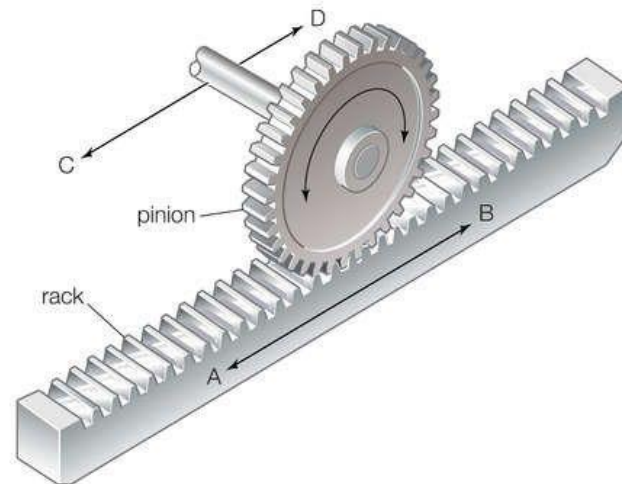
Mechanism:

The bolt is rotated using motor. The direction of the motor is controlled by the L293D IC. Since position of bolt is fixed, the nut moves over bolt depending on direction of rotation of bolt. By controlling direction of rotation of bolt (attached to motor) the lock can be moved back and forth. Since tower lock is attached to moving bolt, lock can be locked and unlocked.

Drawbacks:

- The mechanism wasn't working smoothly
- There was a time lag
- The design was not stable
- Most of the times an initial kick was required

For our final approach, we used a combination of gears, a Rack and Pinion model.



We replaced the tower lock with the rack and pinion and attached a small hook at the end of the rack which acts as a lock. The pinion is controlled by the standard DC motor.

Mechanism:

A rack and pinion are a type of linear actuator that comprises a circular gear engaging a linear gear, which operate to translate rotational motion into linear motion. Driving the pinion into rotation causes the rack to be driven linearly.

By changing direction of rotation of motor (attached to pinion), Rack can be moved back and forth to lock and unlock.

Advantages over previous approach:

- Smooth working
- More Stable
- Simpler Design
- No initial kick required

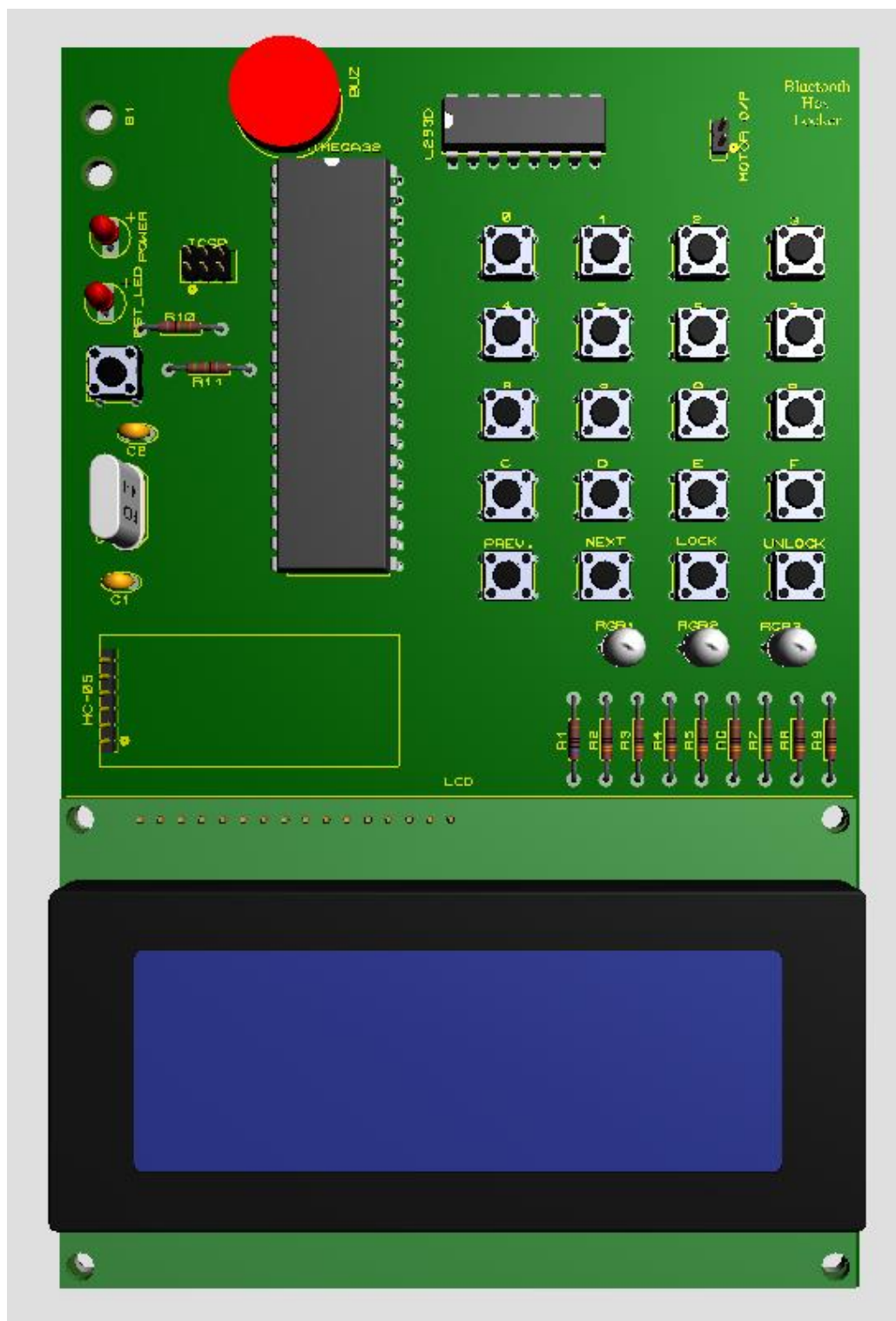
4. ELIMINATING ERRORS

All the errors and problems faced in the prototyping were later corrected by simulation and hit and trial methods.

5. ASSEMBLING COMPONENTS

After all the errors were eliminated, the PCB design was submitted, and we received the processed PCB. While most of the components were based on Through-Hole Technology (THT), there were a few Surface-Mounted Devices (SMD). All the components were soldered manually, and the final product was tested again.

3D VISUALIZATION OF FINAL PCB



CONCLUSION

This project was completed over a period of 3 months under the given budget. We have successfully designed the Bluetooth-based Hexadecimal-RGB Locker as per our objective without any errors.

The development of this project involved a lot of research on various microcontrollers, the mechanical aspects of the above designed locking mechanism and making an Android application.

It helped us gain various industry level skills such as Arduino IDE programming, PCB Designing and Android App Development.

We have documented the components used, their datasheets and pricing, software used and the development cycle of the hardware in this report.