SecureTemp Access System

PROJECT DOCUMENTATION

Subject: Embedded Systems
Authors: Mocan Vlad - Andrei & Huțuleac Diana
Faculty: Polytechnic University of Timișoara, AC, CTI-Eng
Study Year: III

Table of Contents

1. Statement and Brief Explanation of Functionalities

Statement:

The SecureTemp Access System addresses the need for secure access to sensitive environmental data, specifically temperature, in scenarios where restricted access is essential. It ensures that only authorized users can view the temperature readings, enhancing security in environments such as data centers, laboratories, or secure facilities.

System Components:

- **Arduino Uno Board**: Serves as the central processing unit, managing input from the keypad, controlling the LCD display, and reading data from the DHT22 sensor.
- **4x4 Keypad**: Allows users to input a 4-digit passcode to gain access to the temperature data.
- **Adafruit RGB LCD Pi Plate (LCD)**: Used to display messages and temperature readings to the user.
- **DHT22 Sensor**: Measures the temperature and humidity of the environment.
- **Resistor**: Used to ensure proper functioning and protection of components, such as limiting current flow to the DHT22 sensor or other parts of the circuit.

Functionality:

User Authentication: Upon powering up, the system displays the message "Enter passcode or press C to change it" on the LCD.
Users must enter a pre-recorded 4-digit passcode to access the temperature data.
If the passcode is entered incorrectly, the system counts the failed attempts and initiates a lockout period after three incorrect attempts to prevent unauthorized access.

Passcode Management: Users have the option to change the passcode by pressing 'C'.
This feature prompts the user to enter the old passcode first for verification. If correct, it allows the user to set a new 4-digit passcode, enhancing the security by enabling passcode updates.

Temperature Display: Once the correct passcode is entered, the system unlocks and reads the temperature data from the DHT22 sensor.
The temperature reading is then displayed on the LCD, providing the user with real-time environmental data.

Security Measures: If incorrect passcodes are entered consecutively three times, the system enters a lockout state for 30 seconds, displaying a countdown on the LCD. This feature deters unauthorized attempts to access the data.

2. Description of the Development Board Used in the Project

Arduino Uno Board

Description: The Arduino UNO R3 is a versatile development
board is equipped with the well-known ATmega328P Processor. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller.
"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward.

Features:

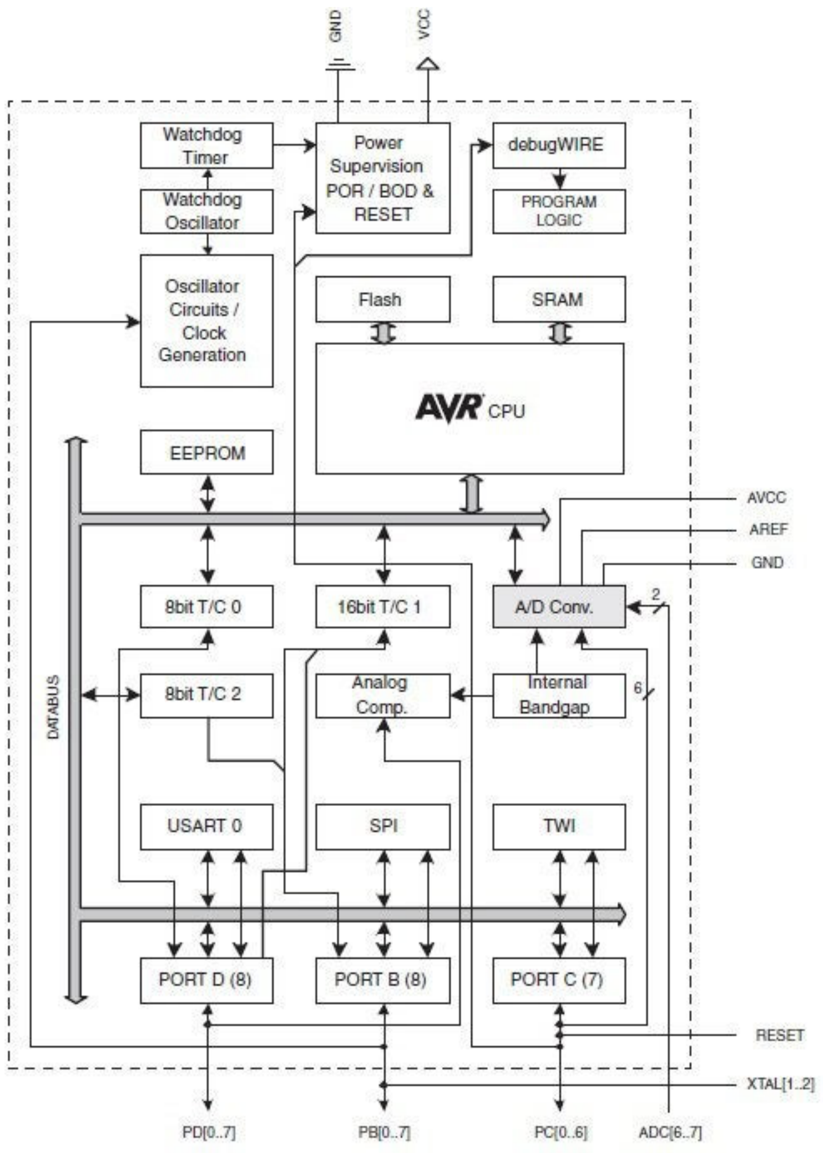
- ATMega328P Processor
- Memory:
  AVR CPU at up to 16 MHz
  32KB Flash
  2KB SRAM
  1KB EEPROM
- Security:
  Power On Reset (POR)
  Brown Out Detection (BOD)
- Peripherals:
  - 2x 8-bit Timer/Counter with a dedicated period register and compare channels
  - 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
  - 1x USART with fractional baud rate generator and start-of-frame detection
  - 1x controller/peripheral Serial Peripheral Interface (SPI)
  - 1x Dual mode controller/peripheral I2C
  - 1x Analog Comparator (AC) with a scalable reference input
  - Watchdog Timer with separate on-chip oscillator
  - Six PWM channels
  - Interrupt and wake-up on pin change

- Power:
  2.7-5.5 volts

Microcontroller: ATmega328P
The ATmega328P is an 8-bit microcontroller from the Atmel AVR family, designed for use in various embedded applications. It is known for its low power consumption and high performance.

- CPU Core: 8-bit AVR with RISC architecture
- Program Memory: 32 KB Flash memory
- Data Memory: 2 KB SRAM, 1 KB EEPROM

- Timers/Counters: Three 8-bit and one 16-bit Timer/Counter
- Interrupts: 23 interrupt vectors, including external interrupts
- I/O Ports: 23 general-purpose I/O lines
- ADC: 6-channel 10-bit ADC
- PWM: 6 PWM channels
- Communication Interfaces: USART, SPI, I2C (TWI)
- Watchdog Timer: Programmable Watchdog Timer with internal oscillator
- Oscillator: Internal calibrated RC oscillator



Block diagram of ATmega328p

XTAL = crystal – oscillator

ADC = Analog / Digital Convertor

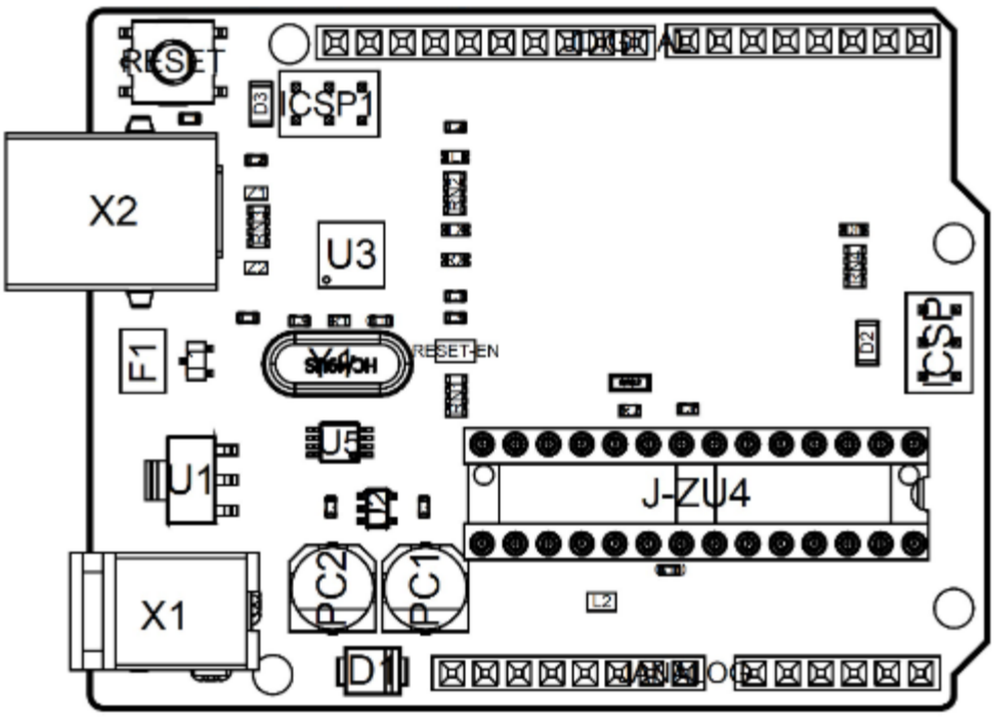
AREF = analog reference (shows the input voltage on the analog pins (default 5V)

AVCC = the pin through which the ADC can be externally supplied (because sometimes the ADC consumes more current and needs to be additionally supplied)

Peculiarities:

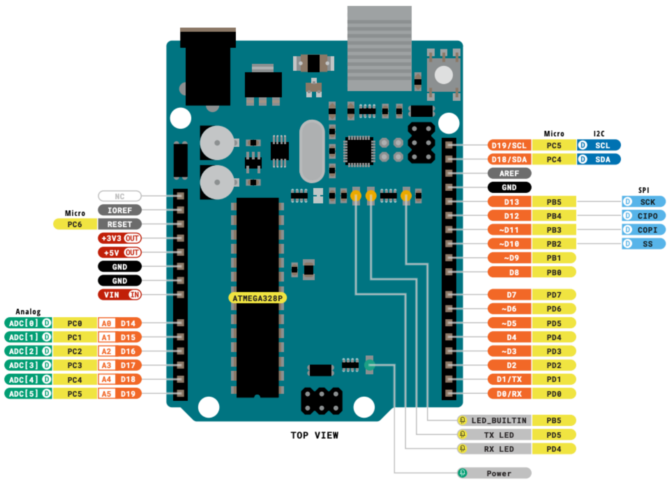
- Low Power Consumption: Features several power-saving modes.
- High Performance: Executes most instructions in a single clock cycle.
- Ease of Use: Compatible with the Arduino IDE, making it accessible for beginners and hobbyists.
- Rich Set of Peripherals: Includes a variety of communication interfaces, making it versatile for numerous applications.

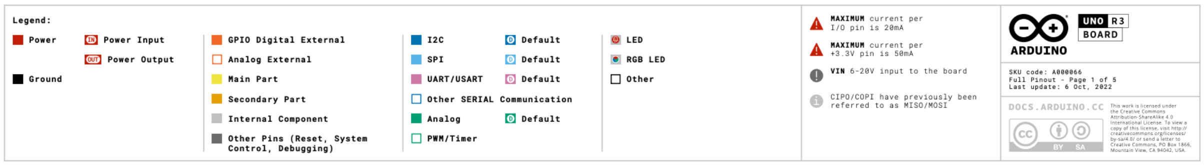
<u>Board Topology</u>



Ref.	Description	Ref.	Description
X1	Power jack 2.1x5.5mm	U1	SPX1117M3-L-5 Regulator
X2	USB B Connector	U3	ATMEGA16U2 Module
PC1	EEE-1EA470WP 25V SMD Capacitor	U5	LMV358LIST-A.9 IC
PC2	EEE-1EA470WP 25V SMD Capacitor	F1	Chip Capacitor, High Density
D1	CGRA4007-G Rectifier	ICSP	Pin header connector (through hole 6)
J-ZU4	ATMEGA328P Module	ICSP1	Pin header connector (through hole 6)
Y1	ECS-160-20-4X-DU Oscillator		

Board Pinout




Legend:

🟥 Power	IN Power Input	🟧 GPIO Digital External	🟦 I2C	Default
	OUT Power Output	⬜ Analog External	🟦 SPI	Default
⬛ Ground		🟨 Main Part	🟪 UART/USART	Default
		🟧 Secondary Part	⬜ Other SERIAL Communication	
		⬜ Internal Component	🟩 Analog	Default
		⬛ Other Pins (Reset, System Control, Debugging)	⬜ PWM/Timer	

🟥	LED
	RGB LED
⬜	Other

⚠ **MAXIMUM** current per I/O pin is 20mA

⚠ **MAXIMUM** current per +3.3V pin is 50mA

VIN 6-20V input to the board

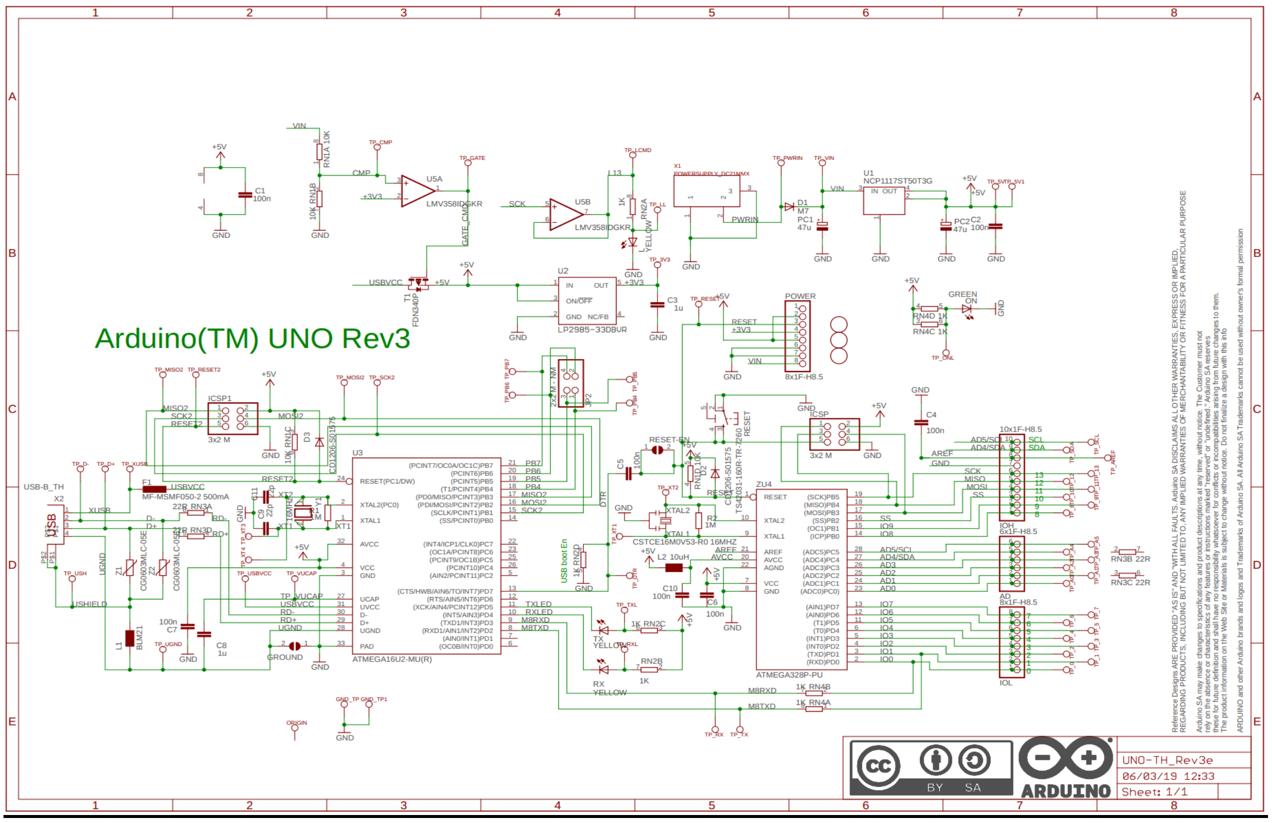
CIPO/COPI have previously been referred to as MISO/MOSI

ARDUINO UNO R3 BOARD

SKU code: A000066
Full Pinout - Page 1 of 5
Last update: 6 Oct, 2022

DOCS.ARDUINO.CC

Board Schematics

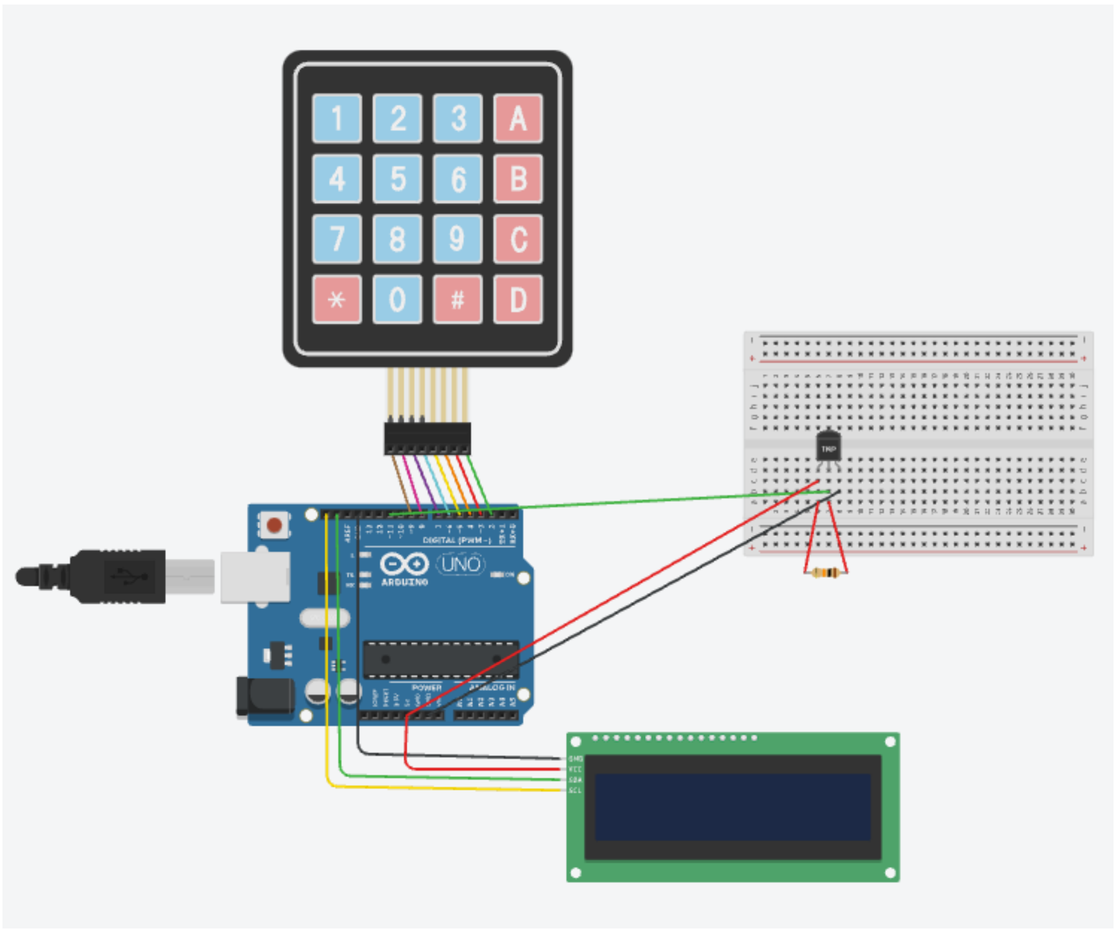


3. System Architecture – Hardware Connection

Description:
The SecureTemp Access System consists of the following main components connected to an Arduino Uno board:

- 4x4 Keypad
- Adafruit RGB LCD Pi Plate (using only the LCD)
- DHT22 Sensor
- Resistor (for pull-up on DHT22 data line)

Below is the detailed description of the connections and a schematic representation created using Tinkercad (https://www.tinkercad.com)



Hardware Connections:

1. Arduino Uno:

- Power Supply: 5V and GND
- Digital I/O Pins: Used for connecting the keypad, LCD and DHT22 sensor
- Analog Pins: Not used in this project

2. <u>4x4 Keypad:</u>

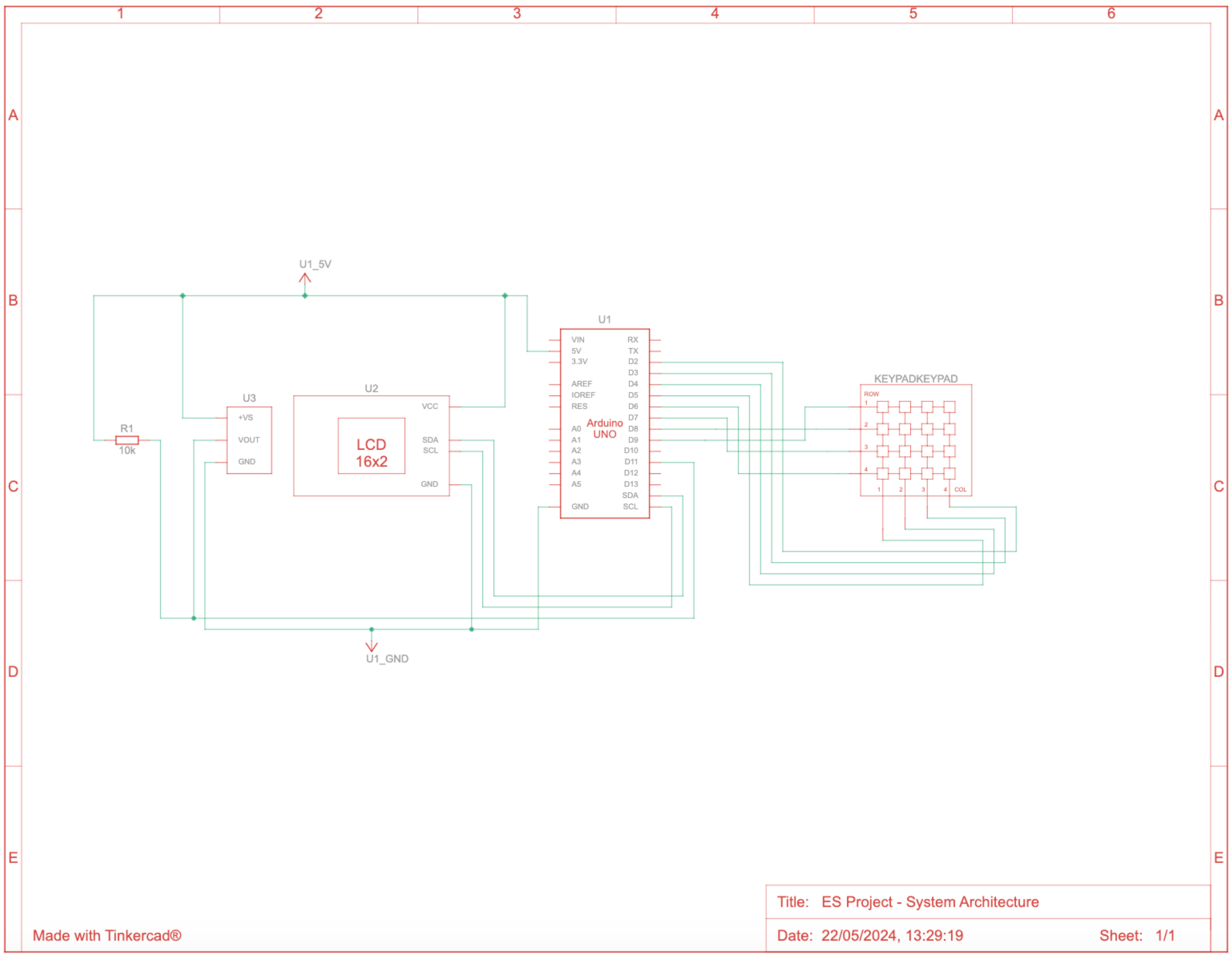
   - Row Pins (4): Connected to Arduino digital pins 6, 7, 8, and 9
   - Column Pins (4): Connected to Arduino digital pins 2, 3, 4, and 5

3. <u>Adafruit RGB LCD Pi Plate (LCD):</u>

   - I2C Interface: Connected to Arduino SDA (A4) and SCL (A5) pins
   - Power Supply: 5V and GND from Arduino

4. <u>DHT22 Sensor:</u>

   - VCC: Connected to 5V pin on Arduino
   - GND: Connected to GND pin on Arduino
   - Data: Connected to digital pin 11 on Arduino
   - Resistor: A 10kΩ resistor connected between VCC and the data pin of the DHT22 (pull-up resistor)



Schematic View

The schematic diagram depicts the hardware connections for the SecureTemp Access System. The Arduino Uno (U1) serves as the main controller, interfacing with a 4x4 keypad connected to its digital pins 2 through 9. The LCD (U2) is connected via the I2C interface (a serial communication protocol) on pins A4 (SDA - Serial Data Line) and A5 (SCL - Serial Clock Line) of U1, used to display messages and temperature readings. The DHT22 sensor (U3) connects to digital pin 11 of U1, with a 10kΩ resistor (R1) providing a pull-up on the data line to ensure reliable communication. This configuration enables secure access to temperature data via the keypad input.

4. Detailed Description of the Microcontroller Modules Involved in the Project (ATmega328/P)

The ATmega328P microcontroller, used in the Arduino Uno, is a versatile and powerful 8-bit microcontroller from Atmel (now Microchip). It includes several important modules that facilitate the functionality required for the SecureTemp Access System.

1. I2C Interface

The I2C interface is a two-wire serial communication protocol used to connect low-speed peripherals to microcontrollers. In the ATmega328P, the I2C interface is implemented using the following pins:

SDA (Serial Data Line, pin A4 on U1): This line is used for sending and receiving data between the master (Arduino Uno) and the slave devices (LCD in this project).

SCL (Serial Clock Line, pin A5 on U1): This line provides the clock signal, which synchronizes data transfer between devices on the I2C bus.

In this project, the I2C interface is used to connect the LCD (U2) to the Arduino Uno (U1), allowing the microcontroller to send commands and data to be displayed.

2. GPIO (General-Purpose Input/Output) Pins

The ATmega328P microcontroller has several GPIO pins used for various functions in the project:

Keypad Interface:
Digital Pins 2-9 (on U1): These pins are connected to the rows and columns of the 4x4 keypad. The microcontroller scans these pins to detect which key is pressed.
Rows (R1-R4): Connected to pins 6, 7, 8, and 9.
Columns (C1-C4): Connected to pins 2, 3, 4, and 5.

DHT22 Sensor:
Digital Pin 11 (on U1): This pin reads data from the DHT22 sensor. A 10kΩ resistor (R1) is connected between the VCC and data line of the DHT22 sensor to act as a pull-up resistor, ensuring reliable data transmission.

3. Timers

The ATmega328P microcontroller has three timers used for generating delays, PWM signals, and managing time-related tasks:

Timer0: An 8-bit timer used for generating the system time base (millis() and delay() functions) and for PWM on digital pins 5 and 6.
Timer1: A 16-bit timer used for precise timing operations, such as managing debounce time for keypad input and delay intervals for LCD updates.
Timer2: An 8-bit timer used for additional timing operations, including delays for sensor readings.

4. Interrupt System

The ATmega328P microcontroller supports an interrupt system that allows it to respond to asynchronous events. This feature is essential for handling events without continuously polling:

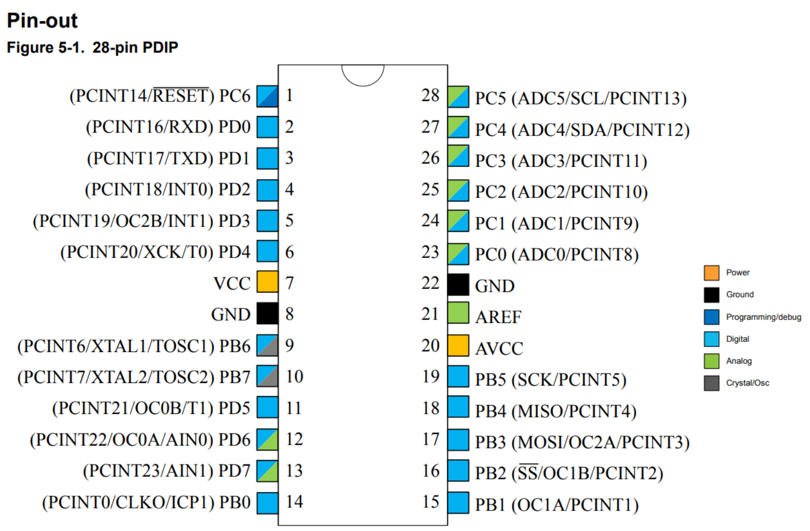
External Interrupts: The microcontroller can be configured to trigger an interrupt when a specific pin changes state (e.g., detecting a button press on the keypad).
Timer Interrupts: Used to execute code at specific time intervals, which can help manage tasks like checking for new keypad input or updating the display.

5. Serial Communication (USART)

The ATmega328P includes a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module for serial communication, which is useful for debugging and communication with other devices:

RX (Digital Pin 0 on U1): Receive pin for serial communication.
TX (Digital Pin 1 on U1): Transmit pin for serial communication.

In this project, the USART module is not explicitly used but can be employed for serial debugging to monitor the system's status and troubleshoot issues.



ATmega328P pinout for the 28-pin PDIP - Plastic Dual In-line package

5. Explanation of the Chosen Display System's Operation

The LCD RGB KEPAD for Rpi

In the SecureTemp Access System, the display module used is the Adafruit RGB LCD Pi Plate. This display system consists of an LCD screen mounted on a plate, which also includes an MCP23017/MCP23S17 I/O expander chip. Although the plate includes RGB backlighting capabilities, only the LCD functionality is utilized in this project for displaying text.

The LCD RGB KEYPAD for RPi comes with following features:

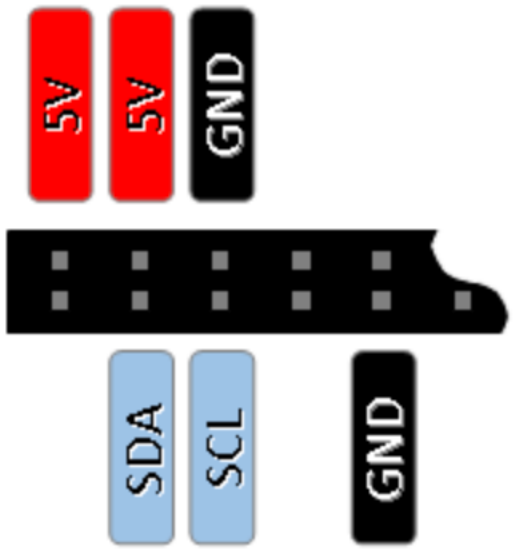
- A 16bit I2C port expander based on a MCP23017
- 5 push buttons
- a RGB LED
- a 1602 LCD with white characters and blue backlight

The I2C Expander MCP23017 for the LCD

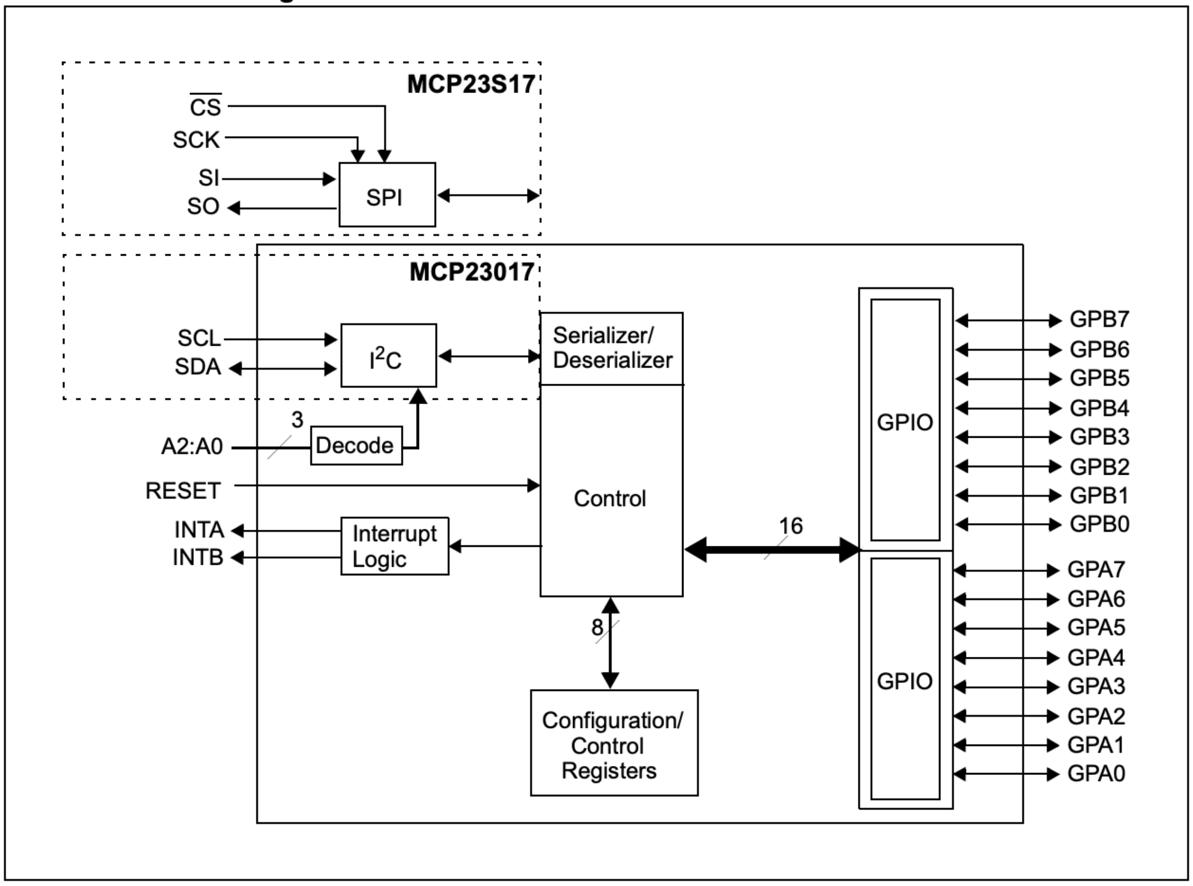
As port expander a MCP23017 is used. The MCP23017 has 16 bits (channels), 7 channels are connected to the LCD in 4 bit mode, 3 channels are used for the RGB LED and 5 channels are inputs connected for the 5 buttons. One channel (GPA5) remains unused.

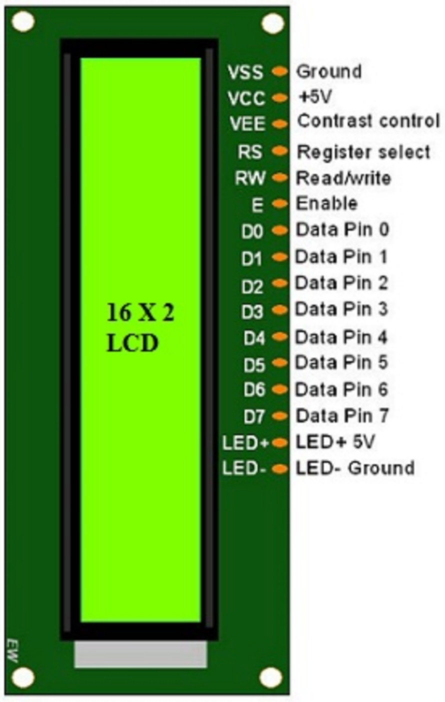
Physical Pin	Pin Name	Pin ID Adafruit	Usage
21	GPA0	0	Button Select
22	GPA1	1	Button right
23	GPA2	2	Button Down
24	GPA3	3	Button up
25	GPA4	4	Button left
26	GPA5	5	-
27	GPA6	6	RGB red
28	GPA7	7	RGB blue
1	GPB0	8	RGB green
2	GPB1	9	D7
3	GPB2	10	D6
4	GPB3	11	D5
5	GPB4	12	D4
6	GPB5	13	EN
7	GPB6	14	RW
8	GPB7	15	RS

The pin layout on the RPi header looks like following:



Functional Block Diagram





LCD-16x2-pin-diagram

Technical Details

LCD Operation:

The LCD used in the project is a standard 16x2 character display, which means it can show up to 16 characters on each of its two lines. The LCD receives data and commands from the Arduino Uno via the I2C interface. This interface simplifies wiring by using only two pins (SDA and SCL) for communication.

MCP23017 I/O Expander:

The MCP23017 chip expands the number of I/O pins available to the Arduino Uno. It communicates with the microcontroller via the I2C bus. It translates I2C commands into parallel data and control signals required to operate the LCD.
This chip helps in reducing the number of pins needed on the microcontroller, as controlling the LCD directly would require many more I/O pins.

I2C Communication:

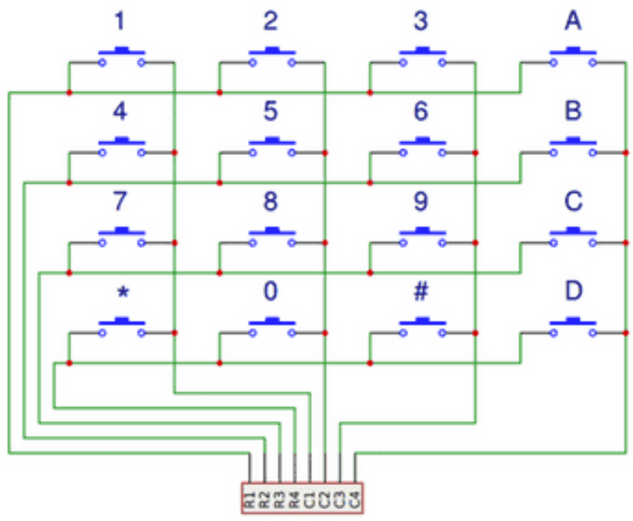
The I2C protocol involves two main lines: the Serial Data Line (SDA) and the Serial Clock Line (SCL).
SDA carries the data, while SCL carries the clock signal to synchronize data transfer.
Each device on the I2C bus has a unique address. The MCP23017's address can be set using hardware jumpers, allowing multiple devices to coexist on the same bus without conflicts.

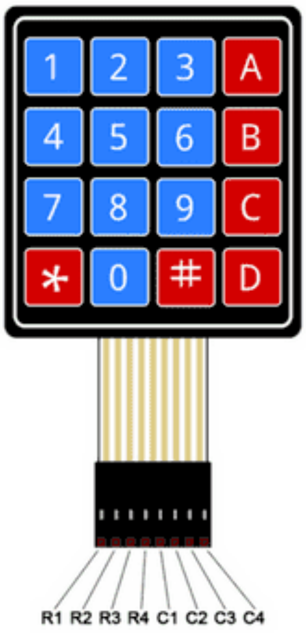
Display Functionality in the Project:

The LCD displays messages such as "Enter passcode" or temperature readings from the DHT22 sensor.
The Arduino sends commands and data to the MCP23017, which then translates these into the appropriate signals for the LCD.
Commands include clearing the display, setting the cursor position, and writing characters.

6. Technical Description of the Sensors and Components Used

1. **4x4 Keypad (Input Device)**
   The buttons on a keypad are arranged in rows and columns. 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.
   The Arduino detects which button is pressed by detecting the row and column pin that’s connected to the button.



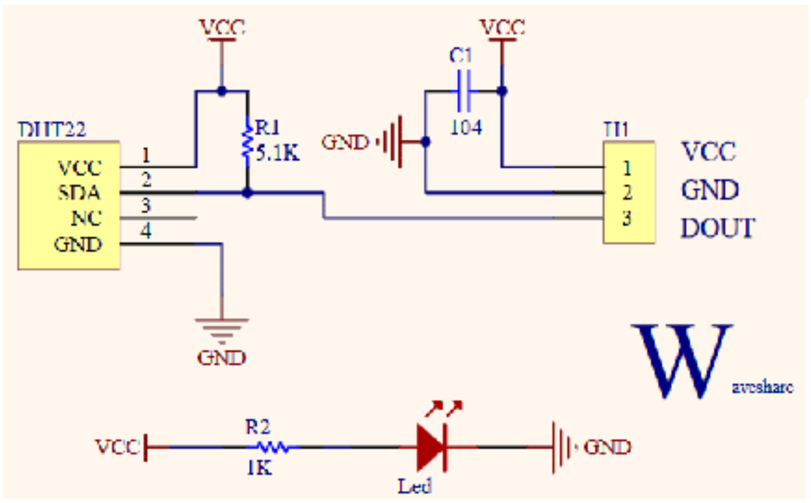


2. DHT22 Temperature and Humidity Sensor

Operating Principle: The DHT22 sensor measures temperature and humidity using a capacitive humidity sensor and a thermistor. It outputs a digital signal containing the temperature and humidity data.

Interface Type: The sensor connects to the Arduino via a digital I/O pin (D11) and requires a 5V power supply along with a ground connection. A 10kΩ pull-up resistor is placed between the data line and VCC to ensure stable data transmission.

Data Interpretation Mode: The sensor transmits data as a digital signal composed of 40 bits: 16 bits for humidity, 16 bits for temperature, and 8 bits for a checksum. The Arduino initiates communication by sending a start signal and then waits for the sensor to respond with the data. Once received, the Arduino reads and processes the data to extract the temperature and humidity values, verifying the checksum to ensure data integrity.



7. Program, Comments and Description

This code implements a SecureTemp Access System using an Arduino Uno, a 4x4 keypad, an Adafruit RGB LCD Pi Plate (using only the LCD), and a DHT22 temperature and humidity sensor. The system allows the user to enter a passcode using the keypad to unlock access to temperature and humidity data displayed on the LCD. The system also provides functionalities to change the passcode and includes a lockout mechanism after several incorrect attempts.

```
#include <Wire.h> // Include the Wire library for I2C communication

// MCP23017 I2C address
#define MCP23017_ADDR 0x20 // Define the I2C address for the MCP23017 chip

// MCP23017 registers
#define IODIRA 0x00 // Register address for setting the direction of GPIOA
#define IODIRB 0x01 // Register address for setting the direction of GPIOB
#define GPIOA 0x12 // Register address for the data of GPIOA
#define GPIOB 0x13 // Register address for the data of GPIOB
#define OLATA 0x14 // Register address for the output latches of GPIOA
#define OLATB 0x15 // Register address for the output latches of GPIOB

// LCD commands
#define LCD_CLEARDISPLAY 0x01 // Command to clear the LCD display
#define LCD_RETURNHOME 0x02 // Command to return the cursor to the home position
#define LCD_ENTRYMODESET 0x04 // Command to set the entry mode
#define LCD_DISPLAYCONTROL 0x08 // Command to control the display
#define LCD_FUNCTIONSET 0x20 // Command to set the function
#define LCD_SETDDRAMADDR 0x80 // Command to set the DDRAM address

// LCD flags for function set
#define LCD_4BITMODE 0x00 // Flag to set 4-bit mode
#define LCD_2LINE 0x08 // Flag to set 2-line display
#define LCD_5x8DOTS 0x00 // Flag to set 5x8 character font

// LCD flags for display on/off control
#define LCD_DISPLAYON 0x04 // Flag to turn on the display
#define LCD_DISPLAYOFF 0x00 // Flag to turn off the display
#define LCD_CURSOROFF 0x00 // Flag to turn off the cursor
#define LCD_BLINKOFF 0x00 // Flag to turn off cursor blinking

#define LCD_WIDTH 16  // Define the width of the LCD (16 characters)

// Keypad configuration
const byte ROWS = 4; // Define the number of rows in the keypad
const byte COLS = 4; // Define the number of columns in the keypad

// Define the characters on the keypad
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

// Define the row and column pins of the keypad
byte rowPins[ROWS] = {9, 8, 7, 6}; // Pins connected to the rows of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; // Pins connected to the columns of the keypad

char preRecordedPassword[5] = "1234"; // Define the default passcode
const int ledPin = 13; // Define the pin for the LED
bool systemLocked = true; // Variable to track if the system is locked
bool scrolling = false; // Variable to track if a message is scrolling

char enteredPassword[5]; // Buffer to store the entered password
int index = 0; // Index for the entered password

#define NO_KEY '\0' // Define NO_KEY as a null character to represent no key press

int incorrectAttempts = 0; // Counter for incorrect password attempts
const int maxAttempts = 3; // Maximum allowed incorrect attempts
unsigned long lockoutTime = 30000; // Lockout duration in milliseconds (30 seconds)
unsigned long lockoutStartTime = 0; // Variable to store the start time of the
lockout period
bool isLockout = false; // Variable to track if the system is in lockout mode

#define DHTPIN 11  // Define the digital pin connected to the DHT22 sensor

void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate
  Wire.begin(); // Initialize I2C communication

  // Initialize MCP23017 I/O expander
  writeRegister(MCP23017_ADDR, IODIRA, 0x00); // Set all pins of GPIOA as outputs
  writeRegister(MCP23017_ADDR, IODIRB, 0x00); // Set all pins of GPIOB as outputs

  // Initialize the LCD
  delay(50); // Wait for the LCD to power up

  // Follow the recommended initialization sequence for the LCD
  lcdCommand(0x03); // Set the LCD to 8-bit mode
  delayMicroseconds(4500); // Wait for more than 4.1ms
  lcdCommand(0x03); // Set the LCD to 8-bit mode again
  delayMicroseconds(4500); // Wait for more than 4.1ms
  lcdCommand(0x03); // Set the LCD to 8-bit mode again
  delayMicroseconds(150); // Wait for more than 100µs
  lcdCommand(0x02); // Set the LCD to 4-bit mode

  // Set the function of the LCD: 4-bit mode, 2 lines, 5x8 font
  lcdCommand(LCD_FUNCTIONSET | LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS);
  // Set the display control: display on, cursor off, blink off
  lcdCommand(LCD_DISPLAYCONTROL | LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF);

  // Clear the LCD display
  lcdCommand(LCD_CLEARDISPLAY);
  delayMicroseconds(2000); // Wait for the command to execute

  // Set the entry mode: increment automatically, no display shift
  lcdCommand(LCD_ENTRYMODESET | 0x02);

  // Initialize the DHT22 sensor
  pinMode(DHTPIN, INPUT); // Set the DHT22 pin as an input
}

void loop() {
  if (isLockout) { // Check if the system is in lockout mode
    unsigned long elapsedTime = millis() - lockoutStartTime; // Calculate the
elapsed time since lockout started
    if (elapsedTime >= lockoutTime) { // Check if the lockout time has passed
      isLockout = false; // Reset lockout status
      incorrectAttempts = 0; // Reset incorrect attempts counter
      lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
      lcdPrint("Enter passcode:"); // Prompt the user to enter the passcode
    } else {
      lcdSetCursor(0, 1); // Set the cursor to the second line
      lcdPrint("Retry in: "); // Display retry message
      lcdPrint(String((lockoutTime - elapsedTime) / 1000).c_str()); // Display the
remaining lockout time in seconds
      lcdPrint(" sec "); // Display "sec" to indicate seconds
    }
    return; // Exit the loop
  }

  if (systemLocked) { // Check if the system is locked
    static bool scrollingMessage = true; // Static variable to track if the
scrolling message is active
    static bool changePasswordMode = false; // Static variable to track if the
change password mode is active
    static bool newPasswordMode = false; // Static variable to track if the new
password mode is active
    static bool oldPasswordMode = false; // Static variable to track if the old
password mode is active

    if (scrollingMessage) { // Check if the scrolling message is active
      scrollText("  Enter passcode or press C to change it"); // Scroll the prompt
message on the LCD
      scrollingMessage = false; // Deactivate the scrolling message
    }

    char customKey = getKey(); // Get the pressed key from the keypad
    if (customKey != NO_KEY) { // Check if a key was pressed
```

```
      scrollingMessage = false; // Deactivate the scrolling message
      if (customKey == 'C') { // Check if the 'C' key was pressed to change the
password
        changePasswordMode = true; // Activate change password mode
        lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
        lcdPrint("Old passcode:"); // Prompt the user to enter the old passcode
      } else {
        if (changePasswordMode) { // Check if change password mode is active
          changePasswordMode = false; // Deactivate change password mode
          oldPasswordMode = true; // Activate old password mode
          lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
          lcdPrint("Old passcode"); // Prompt the user to enter the old passcode
        } else if (oldPasswordMode) { // Check if old password mode is active
          if (index < 4) { // Check if the entered password length is less than 4
            enteredPassword[index++] = customKey; // Add the pressed key to the
entered password
            lcdSetCursor(index - 1, 1); // Set the cursor to the next position
            lcdPrint("*"); // Display an asterisk for each entered character
          }

          if (index == 4) { // Check if the entered password length is 4
            enteredPassword[index] = '\0'; // Null-terminate the entered password
            delay(500); // Wait for 500 milliseconds
            if (strcmp(enteredPassword, preRecordedPassword) == 0) { // Compare the
entered password with the stored password
              oldPasswordMode = false; // Deactivate old password mode
              newPasswordMode = true; // Activate new password mode
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              lcdPrint("New passcode"); // Prompt the user to enter the new
passcode
              index = 0; // Reset the index for the new password
            } else { // If the entered password is incorrect
              incorrectAttempts++; // Increment the incorrect attempts counter
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              scrollText("Wrong passcode"); // Scroll the wrong passcode message
              delay(2000); // Wait for 2 seconds
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              lcdPrint("Old passcode:"); // Prompt the user to enter the old
passcode again
              index = 0; // Reset the index for the next attempt
              checkLockout(); // Check if the system should enter lockout mode
            }
          }
        } else if (newPasswordMode) { // Check if new password mode is active
          if (index < 4) { // Check if the entered password length is less than 4
            enteredPassword[index++] = customKey; // Add the pressed key to the
entered password
            lcdSetCursor(index - 1, 1); // Set the cursor to the next position
            lcdPrint("*"); // Display an asterisk for each entered character
          }
          if (index == 4) { // Check if the entered password length is 4
            enteredPassword[index] = '\0'; // Null-terminate the entered password
            delay(500); // Wait for 500 milliseconds
            // Update the stored password with the new one
            strcpy(preRecordedPassword, enteredPassword); // Copy the new password
to the stored password
            newPasswordMode = false; // Deactivate new password mode
            systemLocked = false; // Unlock the system
            lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
            lcdPrint("Passcode changed"); // Display passcode changed message
            digitalWrite(ledPin, HIGH); // Turn on the LED
            systemLocked = false; // Ensure the system is unlocked
            delay(2000); // Wait for 2 seconds
            lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
            lcdPrint("See temperature"); // Prompt the user to see the temperature
            delay(2000); // Wait for 2 seconds

            lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
            lcdPrint("See temperature"); // Prompt the user to see the temperature
            index = 0; // Reset the index for the next use
          }
        } else {
          // Normal mode for checking the passcode
          if (index < 4) { // Check if the entered password length is less than 4
            enteredPassword[index++] = customKey; // Add the pressed key to the
entered password
            lcdSetCursor(index - 1, 1); // Set the cursor to the next position
            lcdPrint("*"); // Display an asterisk for each entered character
          }

          if (index == 4) { // Check if the entered password length is 4
            enteredPassword[index] = '\0'; // Null-terminate the entered password
            delay(500); // Wait for 500 milliseconds
            if (strcmp(enteredPassword, preRecordedPassword) == 0) { // Compare the
entered password with the stored password
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              lcdPrint("Passcode correct"); // Display passcode correct message
              digitalWrite(ledPin, HIGH); // Turn on the LED
              systemLocked = false; // Unlock the system
              delay(2000); // Wait for 2 seconds
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              lcdPrint("See temperature"); // Prompt the user to see the
temperature
              delay(2000); // Wait for 2 seconds
              incorrectAttempts = 0; // Reset the incorrect attempts counter
            } else { // If the entered password is incorrect
              incorrectAttempts++; // Increment the incorrect attempts counter
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              scrollText("Wrong passcode"); // Scroll the wrong passcode message
              delay(2000); // Wait for 2 seconds
              lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
              lcdPrint("Enter passcode:"); // Prompt the user to enter the passcode
again
              checkLockout(); // Check if the system should enter lockout mode
            }
            index = 0; // Reset the index for the next attempt
          }
        }
      }
    }
  } else {
    // If the system is unlocked, read data from the DHT22 sensor
    float humidity, temperature; // Variables to store the temperature and humidity
values
    if (readDHT22(temperature, humidity)) { // Read data from the DHT22 sensor
      Serial.print("Humidity: "); // Print humidity label to serial monitor
      Serial.print(humidity); // Print humidity value to serial monitor
      Serial.print("%  Temperature: "); // Print temperature label to serial
monitor
      Serial.print(temperature); // Print temperature value to serial monitor
      Serial.println("C"); // Print unit of temperature to serial monitor

      // Display temperature and humidity on the LCD
      lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
      lcdSetCursor(0, 0); // Set the cursor to the first line
      lcdPrint("Temp: "); // Display temperature label
      lcdPrint(String(temperature).c_str()); // Display temperature value
      lcdPrint("C"); // Display unit of temperature

      lcdSetCursor(0, 1); // Set the cursor to the second line
      lcdPrint("Humidity: "); // Display humidity label
      lcdPrint(String(humidity).c_str()); // Display humidity value
      lcdPrint("%"); // Display unit of humidity

      delay(2000);  // Wait for 2 seconds before the next reading
    } else {
      Serial.println("Failed to read from DHT22 sensor!"); // Print error message
to serial monitor if reading fails
    }
  }
}

void lcdCommand(uint8_t command) {
  // Send a command to the LCD
  write4bits((command & 0xF0) >> 4, false); // Send the higher nibble
  write4bits(command & 0x0F, false); // Send the lower nibble
  delayMicroseconds(2000); // Wait for the command to settle
}

void lcdPrint(const char *str) {
  // Print a string to the LCD
  while (*str) { // Loop through each character in the string
```

```
    writeChar(*str++); // Write each character to the LCD
  }
}

void writeChar(uint8_t value) {
  // Write a single character to the LCD
  write4bits((value & 0xF0) >> 4, true); // Send the higher nibble
  write4bits(value & 0x0F, true); // Send the lower nibble
}

void write4bits(uint8_t value, bool isData) {
  // Write 4 bits to the LCD
  uint8_t out = 0; // Initialize the output variable

  // Map the bits to the corresponding GPB pins
  if (value & 0x01) out |= (1 << 4); // D4 -> GPB4
  if (value & 0x02) out |= (1 << 3); // D5 -> GPB3
  if (value & 0x04) out |= (1 << 2); // D6 -> GPB2
  if (value & 0x08) out |= (1 << 1); // D7 -> GPB1

  if (isData) {
    out |= 0x80; // RS -> GPB7 (1 for data)
  } else {
    out &= ~0x80; // RS -> GPB7 (0 for command)
  }

  // RW is always 0 (write mode)
  out &= ~0x40; // RW -> GPB6 (0 for write)

  // Send data to LCD
  writeRegister(MCP23017_ADDR, GPIOB, out | 0x20); // EN high -> GPB5
  delayMicroseconds(1); // Ensure the EN pin is high for a short period
  writeRegister(MCP23017_ADDR, GPIOB, out & ~0x20); // EN low -> GPB5
  delayMicroseconds(100); // Commands need > 37us to settle
}

void writeRegister(uint8_t addr, uint8_t reg, uint8_t value) {
  // Write a value to a register on the MCP23017
  Wire.beginTransmission(addr); // Begin transmission to the I2C address
  Wire.write(reg); // Write the register address
  Wire.write(value); // Write the value to the register
  Wire.endTransmission(); // End transmission
}

void scrollText(const char* message) {
  // Scroll a message across the LCD
  int len = strlen(message); // Get the length of the message
  if (len <= 16) { // If the message length is less than or equal to 16 characters
    lcdSetCursor(0, 0); // Set the cursor to the first line
    lcdPrint(message); // Print the message
    delay(2000); // Wait for 2 seconds
    return; // Exit the function
  }

  int position = 0; // Initialize the position variable
  while (position <= len - 16) { // Loop through the message to scroll it
    lcdSetCursor(0, 0); // Set the cursor to the first line
    lcdPrint(message + position); // Print a part of the message starting from the
current position
    delay(500); // Adjust scrolling speed as needed
    position++; // Move to the next position
  }
}

char getKey() {
  // Get the key pressed on the keypad
  for (byte c = 0; c < COLS; c++) { // Loop through each column
    pinMode(colPins[c], OUTPUT); // Set the current column pin as output
    digitalWrite(colPins[c], LOW); // Set the current column pin to LOW
    for (byte r = 0; r < ROWS; r++) { // Loop through each row
      pinMode(rowPins[r], INPUT_PULLUP); // Set the current row pin as input with
pull-up
      if (digitalRead(rowPins[r]) == LOW) { // Check if the key is pressed
        while (digitalRead(rowPins[r]) == LOW); // Wait for the key to be released
        pinMode(colPins[c], INPUT); // Set the current column pin as input
        pinMode(rowPins[r], INPUT); // Set the current row pin as input
        return hexaKeys[r][c]; // Return the key value
      }
      pinMode(rowPins[r], INPUT); // Set the current row pin as input
    }
    pinMode(colPins[c], INPUT); // Set the current column pin as input
  }
  return NO_KEY; // Return NO_KEY if no key is pressed
}

void lcdSetCursor(uint8_t col, uint8_t row) {
  // Set the cursor position on the LCD
  const uint8_t row_offsets[] = { 0x00, 0x40, 0x14, 0x54 }; // Define the row
offsets
  lcdCommand(LCD_SETDDRAMADDR | (col + row_offsets[row])); // Set the DDRAM address
with the row and column offset
}

void checkLockout() {
  // Check if the system should enter lockout mode
  if (incorrectAttempts >= maxAttempts) { // Check if the incorrect attempts exceed
the maximum allowed attempts
    isLockout = true; // Set the lockout status to true
    lockoutStartTime = millis(); // Record the lockout start time
    lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
    scrollText("Too many attempts"); // Scroll the too many attempts message
    delay(2000); // Wait for 2 seconds
    lcdCommand(LCD_CLEARDISPLAY); // Clear the LCD display
  }
}

// DHT22 Functions
void startSignal() {
  // Send start signal to the DHT22 sensor
  pinMode(DHTPIN, OUTPUT); // Set the DHT22 pin as output
  digitalWrite(DHTPIN, LOW); // Pull the pin low for 18 milliseconds
  delay(18);
  digitalWrite(DHTPIN, HIGH); // Pull the pin high for 40 microseconds
  delayMicroseconds(40);
  pinMode(DHTPIN, INPUT); // Set the DHT22 pin as input
}

bool readResponse() {
  // Read the response from DHT22 sensor
  unsigned int loopCnt = 10000;
  while(digitalRead(DHTPIN) == HIGH) { // Wait for the pin to go LOW
    if (loopCnt-- == 0) return false; // Timeout after 10000 cycles
  }

  loopCnt = 10000;
  while(digitalRead(DHTPIN) == LOW) { // Wait for the pin to go HIGH
    if (loopCnt-- == 0) return false; // Timeout after 10000 cycles
  }

  loopCnt = 10000;
  while(digitalRead(DHTPIN) == HIGH) { // Wait for the pin to go LOW
    if (loopCnt-- == 0) return false; // Timeout after 10000 cycles
  }
  return true; // Return true if response is received
}

int readByte() {
  // Read a byte of data from the DHT22 sensor
  int byte = 0;
  for (int i = 0; i < 8; i++) {
    unsigned int loopCnt = 10000;
    while(digitalRead(DHTPIN) == LOW) { // Wait for the pin to go HIGH
      if (loopCnt-- == 0) return -1; // Timeout after 10000 cycles
    }

    unsigned long length = micros(); // Record the time the pin went HIGH

    loopCnt = 10000;
    while(digitalRead(DHTPIN) == HIGH) { // Wait for the pin to go LOW
      if (loopCnt-- == 0) return -1; // Timeout after 10000 cycles
    }
    length = micros() - length; // Calculate the duration the pin was HIGH
    if (length > 40) { // If the duration was greater than 40 microseconds
      byte |= (1 << (7 - i));  // Set the corresponding bit in the byte
    }
  }
  return byte; // Return the byte read from the sensor
}

bool readDHT22(float& temperature, float& humidity) {
  // Read temperature and humidity data from the DHT22 sensor
  uint8_t data[5] = {0, 0, 0, 0, 0}; // Array to store the 40 bits of data from the
sensor

  startSignal(); // Send start signal to the DHT22 sensor
  if (!readResponse()) { // Wait for and check the response from the sensor
    return false; // If the response is not received, return false
  }

  // Read 5 bytes (40 bits) of data from the sensor
  for (int i = 0; i < 5; i++) {
    int byte = readByte(); // Read each byte
    if (byte == -1) { // If reading a byte fails, return false
      return false;
    }
    data[i] = byte; // Store the byte in the data array
  }

  // Verify the checksum to ensure data integrity
  if (data[4] != ((data[0] + data[1] + data[2] + data[3]) & 0xFF)) {
    return false; // If checksum doesn't match, return false
  }

  // Convert the data
  humidity = ((data[0] << 8) + data[1]) * 0.1; // Combine the first two bytes for
humidity and scale
  temperature = (((data[2] & 0x7F) << 8) + data[3]) * 0.1; // Combine the next two
bytes for temperature and scale
  if (data[2] & 0x80) { // Check if the temperature is negative
    temperature = -temperature; // If negative, convert to negative value
  }

  return true; // If everything is successful, return true
}
```

8. Bibliography and sources of information

- 4x4 Keypad:
  https://www.circuitbasics.com/how-to-set-up-a-keypad-on-an-arduino/
- LCD RGB Keypad:
  https://werner.rothschopf.net/microcontroller/202105_arduino_liquid_crystal_mcp23017_en.htm
  https://www.instructables.com/Use-your-Adafruit-rgb-lcd-Pi-Plate-for-Raspberry-P/
- DHT22 Sensor:
  https://forum.arduino.cc/t/reading-dht22-sensor-with-direct-register-access/1128914
  http://www.ocfreaks.com/basics-interfacing-dht11-dht22-humidity-temperature-sensor-mcu/
- DHT22 Data Sheet:
  https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf
- MCP23017/MCP23S17 Data Sheet:
  https://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf
- LCD Data Sheet:
  https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf
- Arduino Board:
  https://docs.arduino.cc/hardware/uno-rev3/
- Arduino Data Sheet:
  https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf
- ATmega328/P Data Sheet:
  https://datasheet.octopart.com/ATMEGA328P-MU-Microchip-datasheet-65729177.pdf
- Embedded Systems Arduino Course
  https://docs.google.com/presentation/d/1jOxJLk7RgOnQ1FzNQzcq9hWwffLBDiVF/edit#slide=id.p25