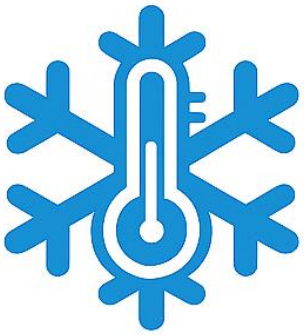


CoolTrack Pro

Smart Temperature Monitor

By: Hanna Mounir
Under the supervision of: Eng. Rawy Iskandar

Table of Contents

Chapter 1: Sensor selection, basic temperature reading circuit

1. Sensor selection:

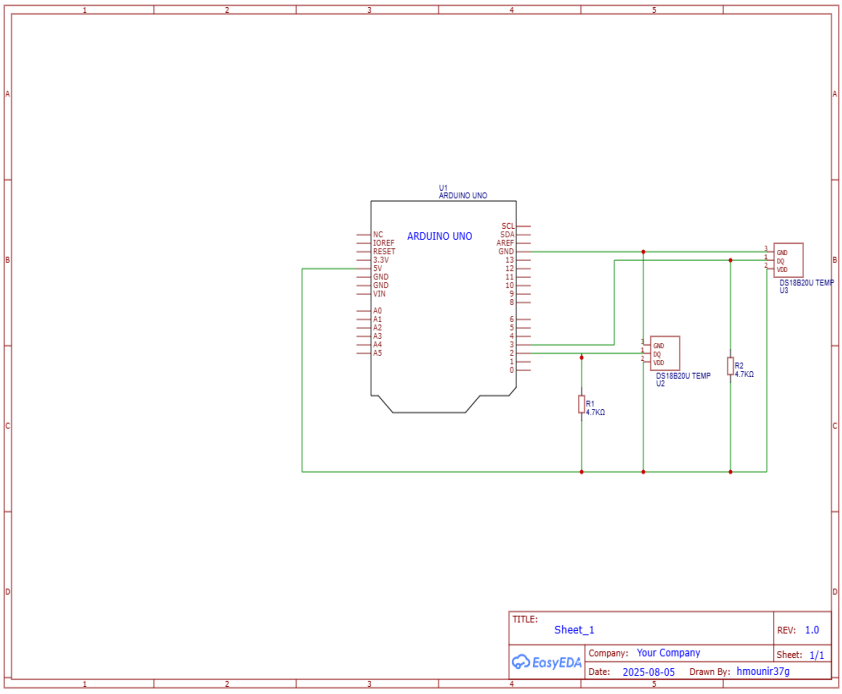
Feature / Sensor	DHT11	DHT22	DS18B20
Type	Digital temp + humidity sensor	Digital temp + humidity sensor	Digital temperature sensor only
Temperature Range	0°C to 50°C	-40°C to +80°C	-55°C to +125°C
Temperature Accuracy	±2°C	±0.5°C	±0.5°C
Response Time	1 sec	2 sec	<1 sec
Update Frequency	1 Hz (1 reading/sec)	0.5 Hz (1 reading/2 sec)	Configurable (fast)
Waterproof Option	❌ No	❌ No	✅ Yes (probe version)
Size	Small	Medium	Probe or small TO-92
Cost	Low ($)	Medium ($$)	Medium ($$)
Ease of Use	Very easy	Easy	Easy
Multiple Sensors on One Pin	❌ No	❌ No	✅ Yes (1-Wire protocol)

Why I chose the **DS18B20:**

- Waterproofing
- Ease of use
- High accuracy
- Fast response time

2. Schematic:

- Tool used EasyEDA



Description:

1. **Microcontroller**

- **Arduino Uno** is used as the main controller.
- Provides power (5 V and GND) and digital I/O pins for communication with sensors.

2. **Temperature Sensors**

- **Two DS18B20 digital temperature sensors** are used:
    - One assigned for **fridge** (connected to digital pin D2).
    - One assigned for **freezer** (connected to digital pin D3).

3. **Sensor Power Connections**

- **VDD** of both sensors is connected to Arduino **5 V**.
- **GND** of both sensors is connected to Arduino **GND**.
- This ensures both sensors share a common power and ground reference.

4. **Data Lines**

- Fridge sensor **DQ** → Arduino **D2**.
- Freezer sensor **DQ** → Arduino **D3**.
- Data lines are independent (not shared) in this version.

5. **Pull-Up Resistors**

- A **4.7 kΩ resistor** is connected between **DQ** and **5 V** for each sensor.
- This keeps the data line at a HIGH level when idle, ensuring reliable 1-Wire communication.
- Since each sensor uses its own data line, **two pull-up resistors** are required.

6. **One-Wire Communication**

- Each DS18B20 communicates over its own dedicated 1-Wire bus.
- Arduino reads temperatures independently from each sensor via separate pins.

7. **Power Source**

- Initially powered through Arduino USB (from a computer).
- Can also be powered by a 5 V source such as a power bank via the Arduino USB port in later stages

3. Assembly and execution:

1. **Component Connection**
   - Assemble the circuit according to the provided schematic diagram.
   - Ensure correct pin connections for both DS18B20 sensors, including the pull-up resistors between each data pin and 5 V.
   - Verify all power (5 V) and ground (GND) lines are securely connected.
2. **Arduino IDE Setup**

- Open the Arduino IDE on your computer.
- Install the required libraries: **OneWire** and **DallasTemperature** via *Sketch → Include Library → Manage Libraries…*.

3. **Code Preparation**

- Load the temperature reading code into the Arduino IDE.
- Ensure pin numbers in the code match the wiring in the schematic.

4. **Uploading and Testing**

- Connect the Arduino to the laptop using a USB cable.
- Select the correct board and port from the *Tools* menu.
- Upload the code to the Arduino.
- Open the Serial Monitor (baud rate: 9600) to verify temperature readings from both sensors.

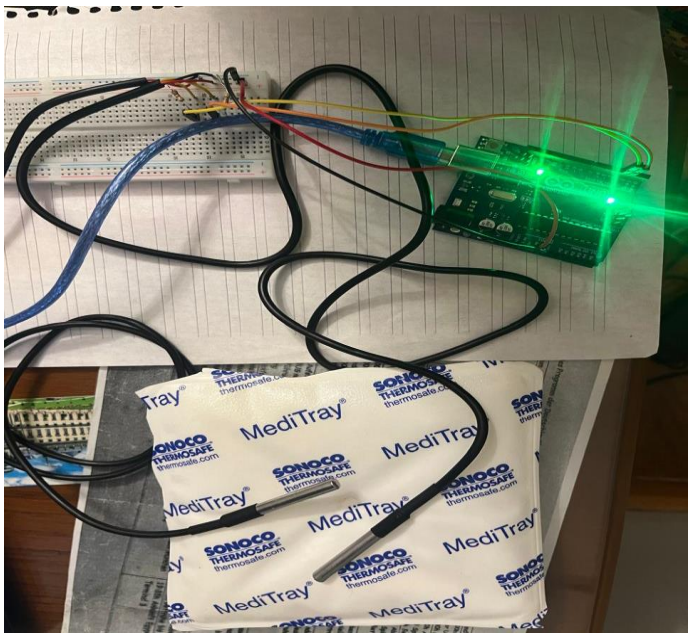
Note: An ice pack was used during testing to verify that both temperature sensors respond accurately to changes in temperature.

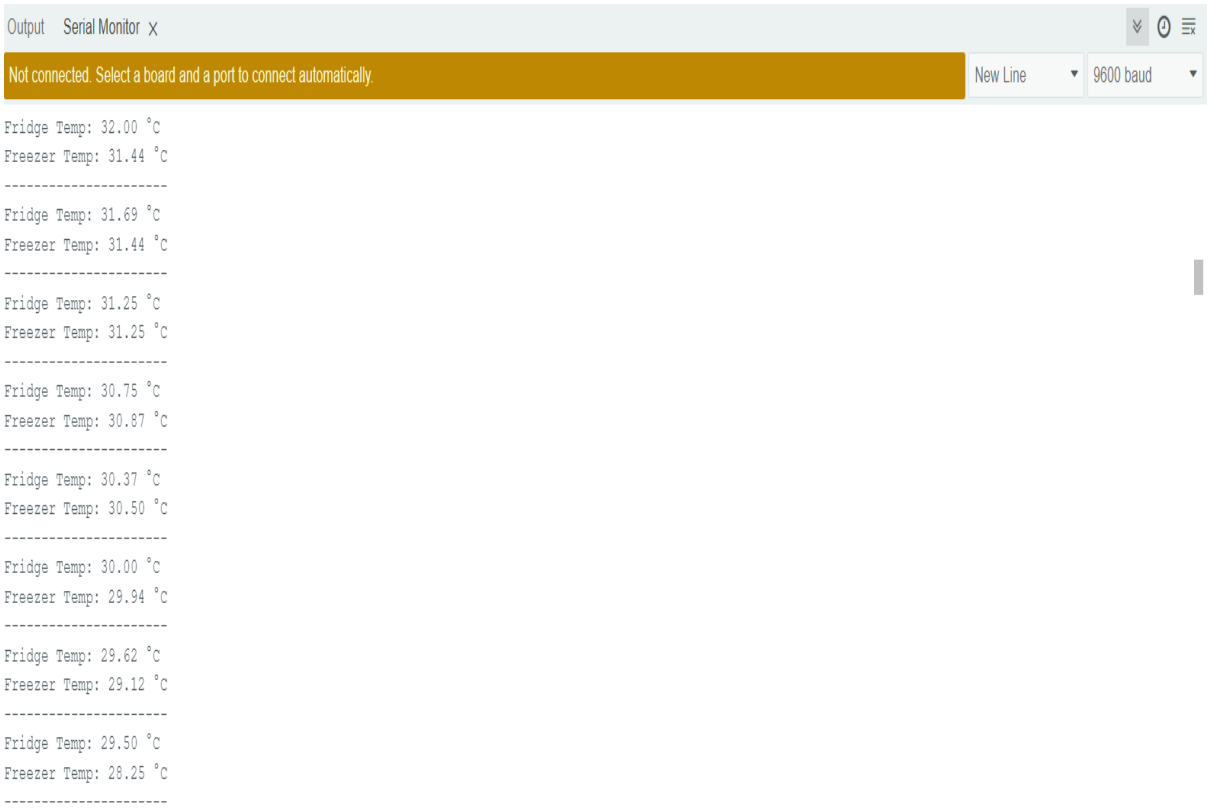
Code:

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Assign pins for each sensor
#define FRIDGE_PIN 2
#define FREEZER_PIN 3

// Create OneWire instances for each pin
OneWire oneWireFridge(FRIDGE_PIN);
OneWire oneWireFreezer(FREEZER_PIN);

// Pass the OneWire references to DallasTemperature
DallasTemperature fridgeSensor(&oneWireFridge);
DallasTemperature freezerSensor(&oneWireFreezer);

void setup() {
  Serial.begin(9600);
```

```
  // Start both sensors
  fridgeSensor.begin();
  freezerSensor.begin();
}

void loop() {
  // Request temperature readings
  fridgeSensor.requestTemperatures();
  freezerSensor.requestTemperatures();

  // Read temperatures in Celsius
  float tempFridge = fridgeSensor.getTempCByIndex(0);
  float tempFreezer = freezerSensor.getTempCByIndex(0);

  // Display in Serial Monitor
  Serial.print("Fridge Temp: ");
  Serial.print(tempFridge);
  Serial.println(" °C");

  Serial.print("Freezer Temp: ");
  Serial.print(tempFreezer);
  Serial.println(" °C");

  Serial.println("----------------------");
  delay(2000);
}
```

Pictures:





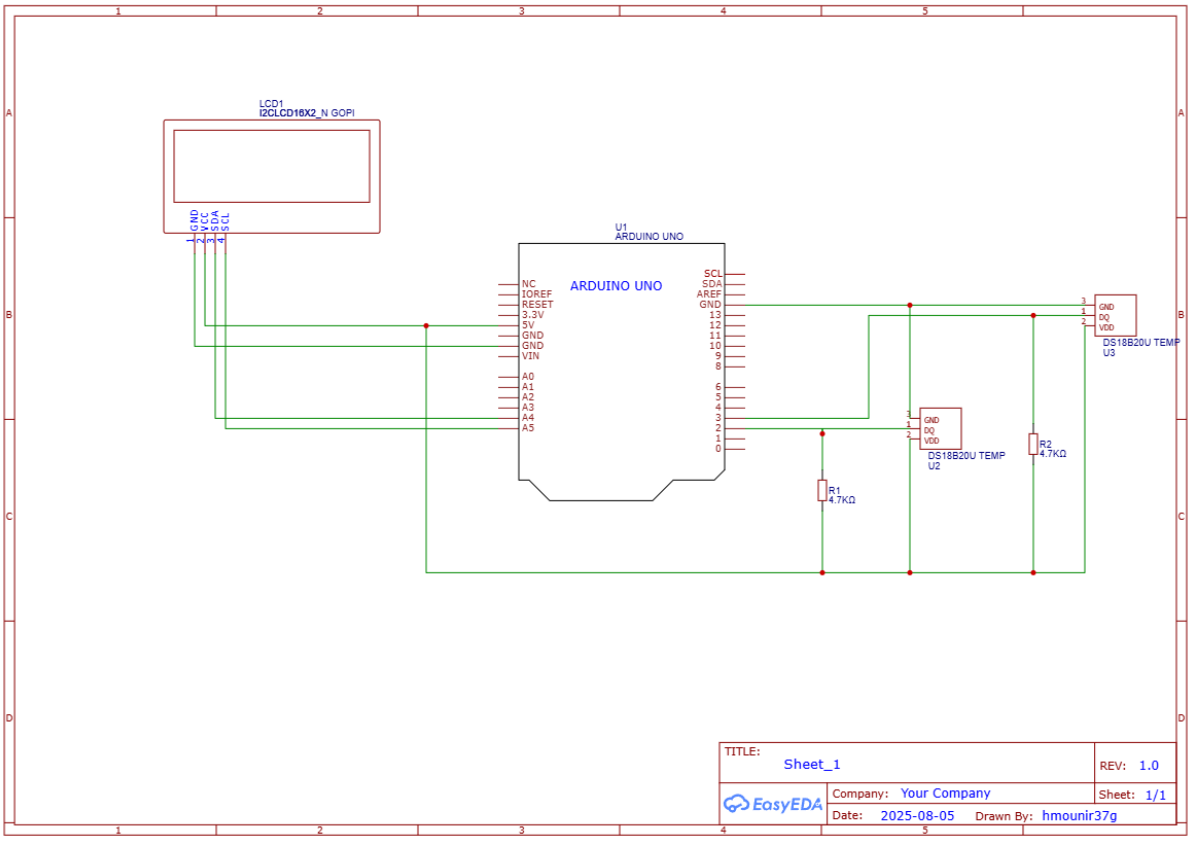
Execution: Everything works as expected

Chapter 2: Display integration and user interface

1. Objective:

The objective for Week 2 was to integrate a 16x2 LCD with I^2C interface into the CoolTrack Pro system. This allows real-time display of temperature readings from both the fridge and freezer DS18B20 sensors, enhancing usability and providing instant feedback without needing to connect to a computer.

2. Schematic:



Description:

1. 16x2 LCD with I²C:
   - VDD connected to Arduino 5 volt
   - GND connected to Arduino GND
   - LCD **SDA** → Arduino **A4**
   - LCD **SCL** → Arduino **A5**

3. Assembly and execution:

1. Soldering the I²C on the LCD

2. Component Connection

- Assemble the circuit according to the provided schematic diagram.
- Verify all power (5 V) and ground (GND) lines are securely connected.

3.Arduino IDE Setup

- Open the Arduino IDE on your computer.
- **Install the required libraries:** `LiquidCrystal_I2C` via *Sketch → Include Library → Manage Libraries*....

4.Code Preparation

- Load the temperature reading code into the Arduino IDE.
- Ensure pin numbers in the code match the wiring in the schematic

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Assign pins for each sensor
#define FRIDGE_PIN 2
#define FREEZER_PIN 3

// Create OneWire instances for each pin
OneWire oneWireFridge(FRIDGE_PIN);
OneWire oneWireFreezer(FREEZER_PIN);

// Pass the OneWire references to DallasTemperature
DallasTemperature fridgeSensor(&oneWireFridge);
DallasTemperature freezerSensor(&oneWireFreezer);

// Initialize the LCD (Address: 0x27 or 0x3F, Columns: 16, Rows: 2)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(9600);

  // Start sensors
  fridgeSensor.begin();
  freezerSensor.begin();

  // Start LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("CoolTrack Pro");
  delay(2000); // Show title for 2 sec
}

void loop() {
  // Request temperatures
  fridgeSensor.requestTemperatures();
  freezerSensor.requestTemperatures();

  // Read temperatures
  float tempFridge = fridgeSensor.getTempCByIndex(0);
  float tempFreezer = freezerSensor.getTempCByIndex(0);
```

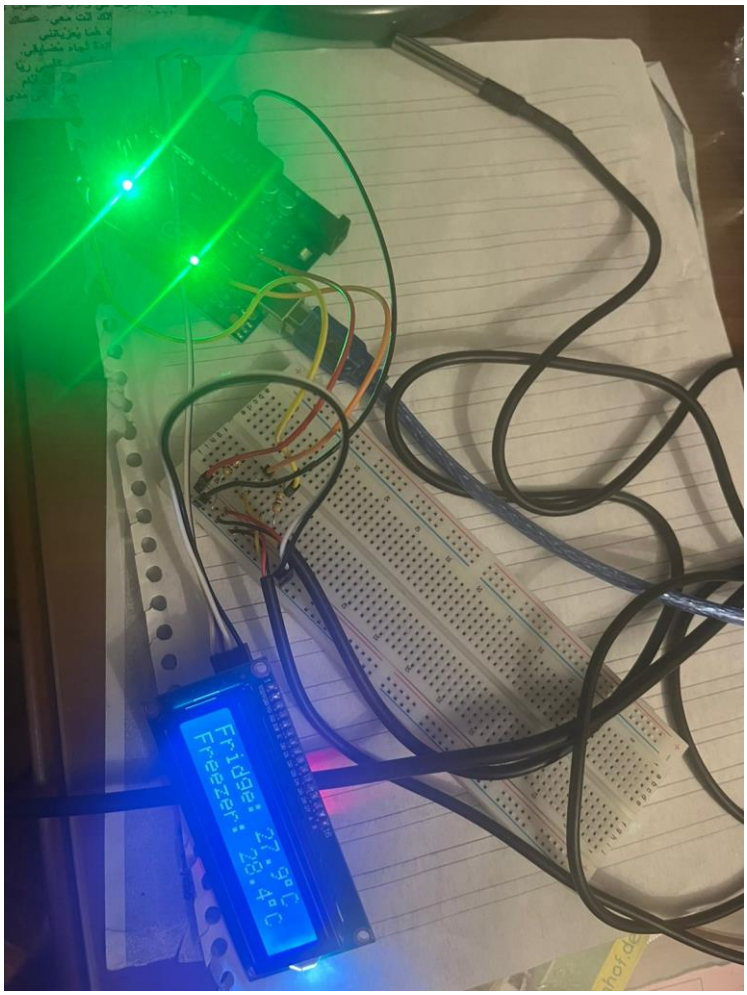
```
  // Serial Monitor output
  Serial.print("Fridge Temp: ");
  Serial.print(tempFridge);
  Serial.println(" °C");

  Serial.print("Freezer Temp: ");
  Serial.print(tempFreezer);
  Serial.println(" °C");
  Serial.println("----------------------");

  // LCD output
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Fridge: ");
  lcd.print(tempFridge, 1); // 1 decimal place
  lcd.print((char)223); // Degree symbol
  lcd.print("C");

  lcd.setCursor(0, 1);
  lcd.print("Freezer: ");
  lcd.print(tempFreezer, 1);
  lcd.print((char)223);
  lcd.print("C");

  delay(2000);
}
```

Pictures:

Boot up screen

Temperature reading

Note: The temperature readings are not relevant in this case, since the objective was not to verify the sensor accuracy but rather to confirm that the LCD display is functioning properly.



Full circuit

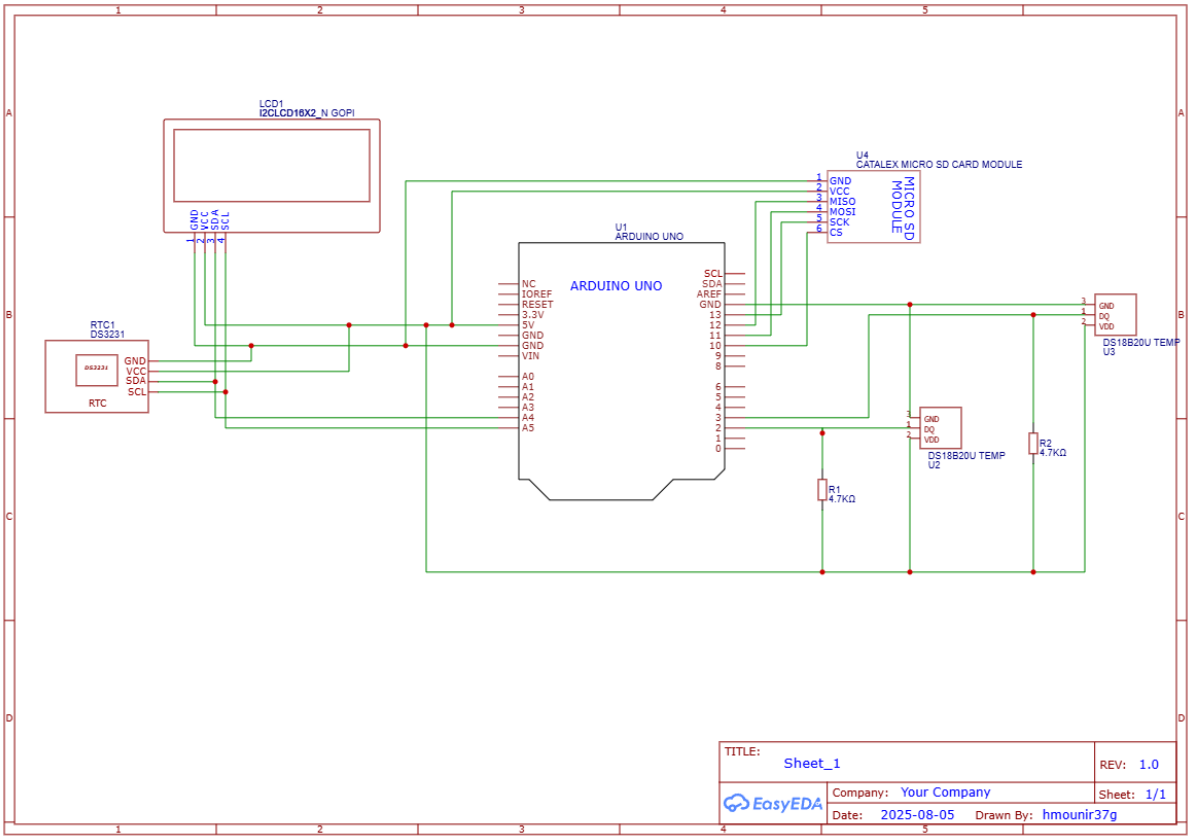
Execution: Everything works as expected

Chapter 3: Data logging and storage implementation

Objective:

The objective for Week 3 was to implement data logging and storage within the CoolTrack Pro system. This step enables temperature readings from the fridge and freezer DS18B20 sensors to be recorded over time, providing historical data that can be reviewed for performance analysis and diagnostics. By adding data storage capabilities, the system moves beyond real-time monitoring and toward comprehensive tracking of refrigerator conditions.

Schematic:



Description:

1. **Microcontroller (Arduino Uno)**

- Serves as the central controller for all components.
- Provides regulated **5 V** and **GND** rails to power the SD card module, RTC, sensors, and LCD.

- Manages digital communication via **SPI** (for SD card) and **I²C** (for RTC + LCD).

2. **SD Card Module (MicroSD)**

- Used for logging and storing temperature data.
- Communicates with the Arduino using the **SPI (Serial Peripheral Interface)** protocol:
    - **MISO (Master In Slave Out)** → Arduino **D12**
        - Sends data *from the SD card* to the Arduino.
    - **MOSI (Master Out Slave In)** → Arduino **D11**
        - Sends data *from the Arduino* to the SD card.
    - **SCK (Serial Clock)** → Arduino **D13**
        - Provides a timing clock signal from the Arduino to synchronize data transfer.
    - **CS (Chip Select)** → Arduino **D10**
        - Activates the SD card module when multiple SPI devices are present.
- Powered from Arduino **5 V** and **GND**.

3. **RTC (Real Time Clock – DS3231)**

- Provides accurate timestamps (date and time) for each logged entry.
- Communicates via the **I²C (Inter-Integrated Circuit)** bus:
    - **SDA (Serial Data Line)** → Arduino **A4**
        - Transfers the actual data between devices.
    - **SCL (Serial Clock Line)** → Arduino **A5**
        - Provides the clock signal to synchronize communication.
- Powered from Arduino **5 V** and **GND**.
- Additional pins **SQW (Square Wave)** and **32K** are not used in this project and remain unconnected.

4. **Temperature Sensors (DS18B20)**

- Two independent digital sensors are used:
    - Fridge sensor → Arduino **D2**.
    - Freezer sensor → Arduino **D3**.
- Each sensor requires a **4.7 kΩ pull-up resistor** between its data pin and 5 V to ensure stable communication.
- Provides continuous temperature readings for both compartments.

5. **LCD with I²C Interface (16x2)**

- Displays real-time fridge and freezer temperature readings.
- Shares the same **I²C bus** as the RTC:
    - **SDA → A4**, **SCL → A5**.
- Powered from Arduino **5 V** and **GND**.

6. **Power Source**

- During testing, the system is powered via Arduino's USB connection to a laptop.

- For standalone operation, a standard **5 V power bank** can be used through the Arduino USB port.

Assembly and execution:

1. **Component Connection**
- Assemble the circuit according to the provided schematic diagram.
- Verify all power (5 V) and ground (GND) lines are securely connected.
2. **Arduino IDE Setup**
- Open the Arduino IDE on your computer.
- **Install the required libraries:** `RTClib, SD` via *Sketch → Include Library → Manage Libraries*....
3. **Code Preparation**
- Load the temperature reading code into the Arduino IDE.
- Ensure pin numbers in the code match the wiring in the schematic
4. **SD Card Preparation:**
    - format the official **SD Card Formatter** utility to ensure compatibility with the Arduino SD library.
    - configured with the **FAT16/FAT32 file system**, enabling proper read/write functionality.

Note: A **2 GB microSD card** was used for data logging in this project. Cards up to **8 GB** are also supported when formatted appropriately.

Code:

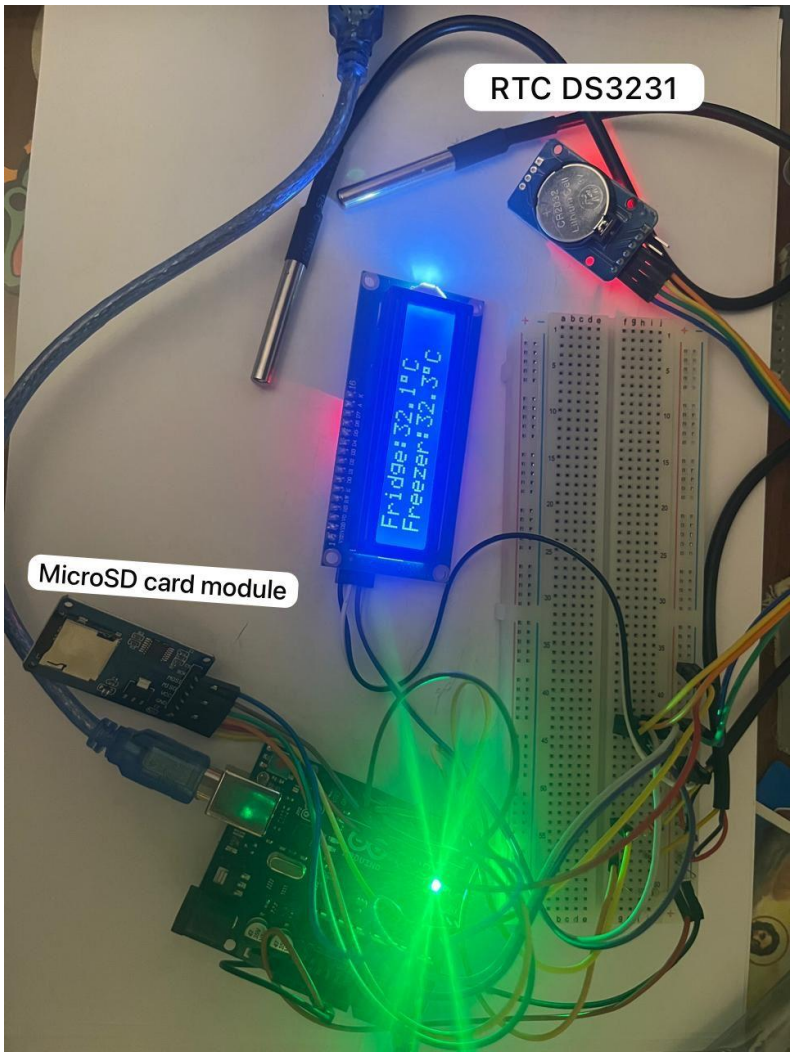
```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <RTClib.h>
#include <SD.h>

// ================== Pin Assignments ==================
#define FRIDGE_PIN   2        // DS18B20 (Fridge)
#define FREEZER_PIN  3        // DS18B20 (Freezer)

#define SD_CS        10       // Chip Select for SD card

// ================== Sensor Objects ==================
OneWire oneWireFridge(FRIDGE_PIN);
OneWire oneWireFreezer(FREEZER_PIN);
```

```
DallasTemperature fridgeSensor(&oneWireFridge);
DallasTemperature freezerSensor(&oneWireFreezer);

LiquidCrystal_I2C lcd(0x27, 16, 2);  // LCD (I²C address 0x27)
RTC_DS3231 rtc;

File logFile;

// ================== Setup ==================
void setup() {
  Serial.begin(9600);

  // Start temperature sensors
  fridgeSensor.begin();
  freezerSensor.begin();

  // Start LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("CoolTrack Pro");
  delay(2000);
  lcd.clear();

  // Start RTC
  if (!rtc.begin()) {
    Serial.println("RTC not found!");
    lcd.print("RTC Fail");
    while (1);
  }
  if (rtc.lostPower()) {
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__))); // Set to compile time
  }

  // Start SD card
  if (!SD.begin(SD_CS)) {
    Serial.println("SD Card init failed!");
    lcd.print("SD Fail");
    while (1);
  }
  Serial.println("SD Card Ready.");
  lcd.print("SD Ready");
  delay(1000);
  lcd.clear();
```

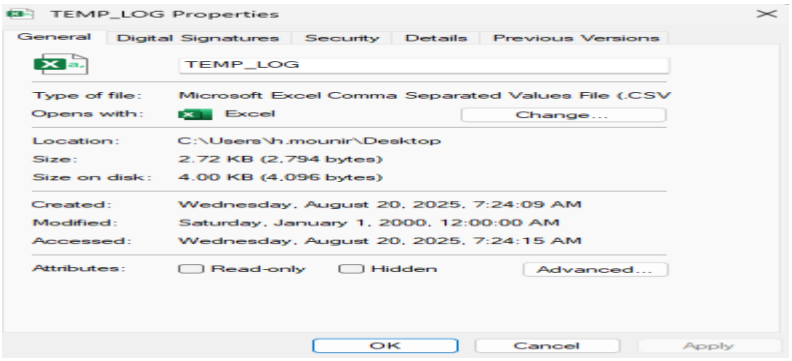
```
  // Prepare log file header
  logFile = SD.open("temp_log.csv", FILE_WRITE);
  if (logFile) {
    logFile.println("Date,Time,Fridge_C,Freezer_C");
    logFile.close();
  }
}

// ================== Main Loop ==================
void loop() {
  // ----- Time -----
  DateTime now = rtc.now();

  // ----- Temperatures -----
  fridgeSensor.requestTemperatures();
  freezerSensor.requestTemperatures();
  float tempFridge = fridgeSensor.getTempCByIndex(0);
  float tempFreezer = freezerSensor.getTempCByIndex(0);

  // ----- Serial Monitor -----
  Serial.print(now.timestamp(DateTime::TIMESTAMP_DATE));
  Serial.print(" ");
  Serial.print(now.timestamp(DateTime::TIMESTAMP_TIME));
  Serial.print(" | Fridge: ");
  Serial.print(tempFridge); Serial.print(" °C | Freezer: ");
  Serial.print(tempFreezer); Serial.println(" °C");

  // ----- LCD Output -----
  lcd.setCursor(0, 0);
  lcd.print("Fridge:");
  lcd.print(tempFridge, 1);
  lcd.print((char)223);
  lcd.print("C   "); // padding to clear leftovers

  lcd.setCursor(0, 1);
  lcd.print("Freezer:");
  lcd.print(tempFreezer, 1);
  lcd.print((char)223);
  lcd.print("C   ");

  // ----- Log to SD -----
  logFile = SD.open("temp_log.csv", FILE_WRITE);
  if (logFile) {
    logFile.print(now.year());   logFile.print("-");
    logFile.print(now.month());  logFile.print("-");
    logFile.print(now.day());    logFile.print(",");
```

```
    logFile.print(now.hour());   logFile.print(":");
    logFile.print(now.minute()); logFile.print(":");
    logFile.print(now.second()); logFile.print(",");
    logFile.print(tempFridge, 1); logFile.print(",");
    logFile.println(tempFreezer, 1);
    logFile.close();
    Serial.println("Data logged.");
  } else {
    Serial.println("Error opening log file!");
  }

  delay(2000); // update every 2 sec
}
```

Pictures:



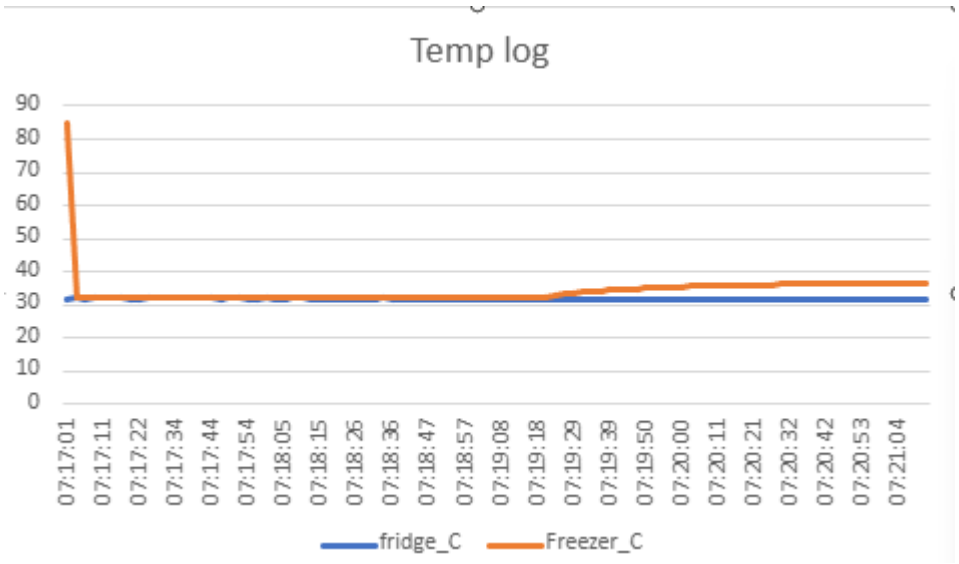
Full circuit



CSV file saved on SD card compatible with Excel

A	B	C
Time	Fridge_C	Freezer_C
07:17:01	31.9	85
07:17:03	32	32.2
07:17:06	31.9	32.2
07:17:09	32	32.2
07:17:11	32	32.2
07:17:14	32	32.2
07:17:17	32	32.2
07:17:19	31.9	32.2
07:17:22	31.9	32.2
07:17:25	32	32.2
07:17:28	32	32.2
07:17:31	32.1	32.2
07:17:34	32.1	32.2
07:17:36	32	32.2
07:17:39	32	32.2

Data log



Data in Graph form

Chapter 4: testing, and documentation

1. Testing:

Objective

To evaluate the performance of the device under practical conditions by verifying its accuracy, power consumption, and sustained operation when placed in a mini refrigerator.

Methodology

1. The device was placed in a mini refrigerator and tested for a continuous period of 3 hours.
2. Each probe was positioned in its designated location:
    a. **Fridge probe** inside the fridge compartment.
    b. **Freezer probe** inside the freezer compartment.
3. The device was powered using a **generic 5000 mAh power bank** with a 5V output.
4. A **generic room thermometer** was used as a reference instrument to validate the accuracy of the temperature readings obtained from the device.

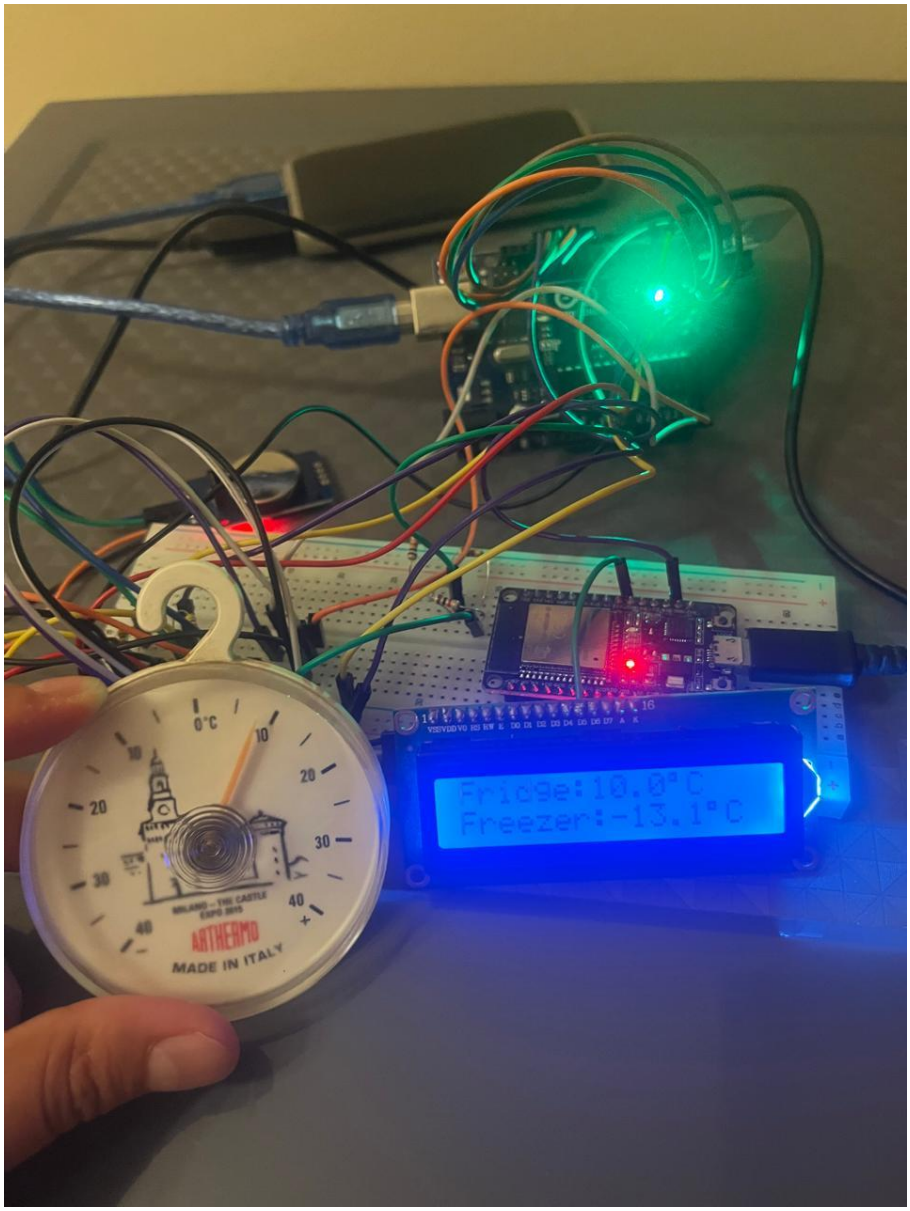
Results

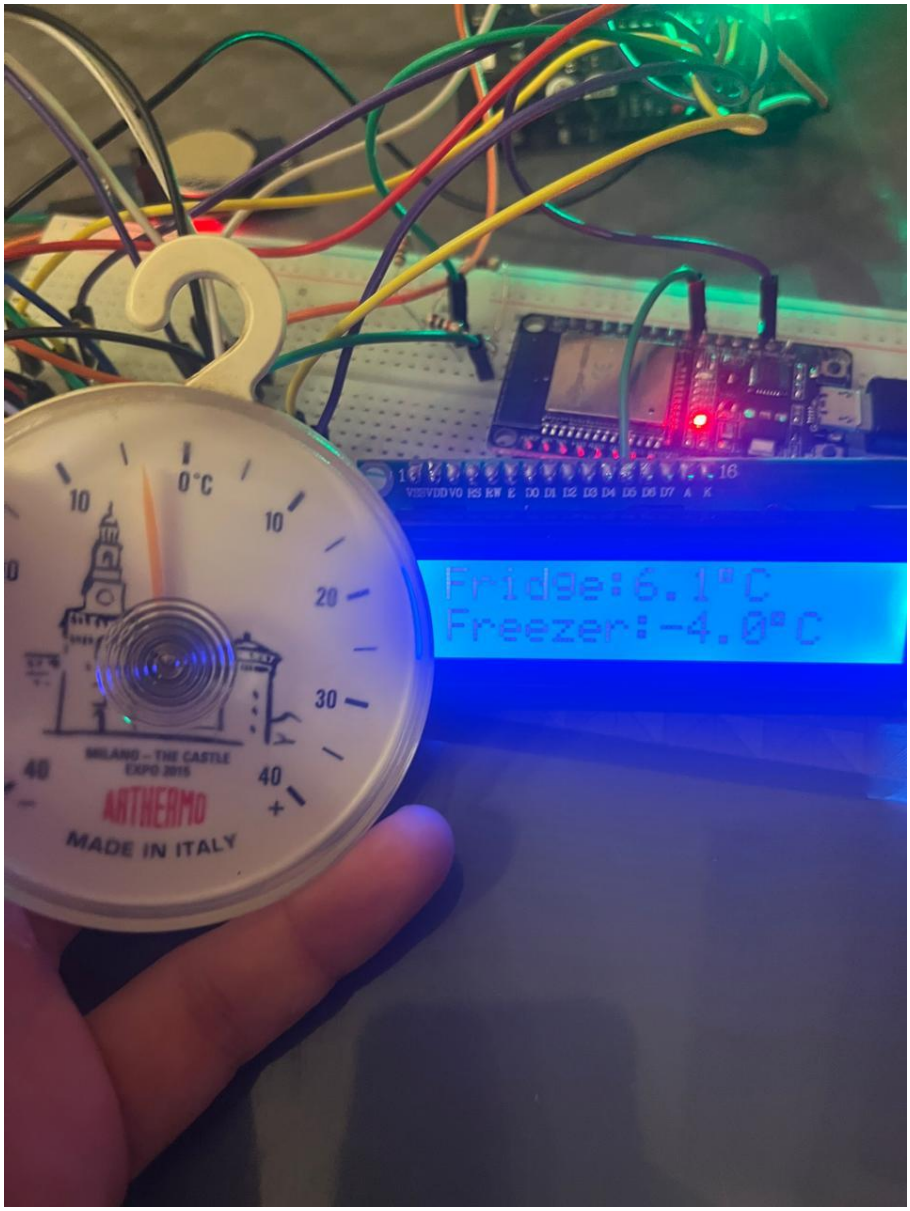
1. The device exhibited **low power consumption**, demonstrating that it can be reliably sustained by the 5000 mAh power bank over extended operation.
2. Temperature readings recorded by the device were **accurate and consistent** with the reference thermometer. A variation of approximately **±1°C** was observed, attributed to the limited accuracy of the reference thermometer.
3. The **SD card module** had been verified in a previous milestone and was confirmed to be functioning correctly.

Conclusion

The test results confirm that the device performs reliably in real-world conditions. It demonstrates excellent power efficiency, sustained operation on a portable power source, and accurate temperature measurements within acceptable limits.

Test setup


Fridge:10.0°C
Freezer:-13.1°C
0°C
10
20
30
40
ARTHERMO
MADE IN ITALY
Fridge test



Freezer test

2.Documentation

Arduino: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf
DS18B20 (temperature sensor): https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf?utm_source=chatgpt.com
16x2 LCD: https://www.vishay.com/docs/37484/lcd016n002bcfhet.pdf
I2C: https://components101.com/modules/i2c-serial-interface-adapter-module
DS3231 RTC Module: https://www.analog.com/media/en/technical-documentation/data-sheets/ds3231.pdf?utm_source=chatgpt.com

Micro-SD Card Module: https://cdn.awsli.com.br/945/945993/arquivos/Datasheet-MicroSD-Module.pdf?utm_source=chatgpt.com