EE2504 ESDP Project
Group-3 (Eco Clock)

Groupmates:
Anagha Balaji - E22BTECH11204
Chittampalli Ananya - EE22BTECH11206
Gargi Behera - EE22BTECH11208
Rayi Giri Varshini - EE22BTECH11215

Problem Statement:
The project envisions a sleek, compact smart clock seamlessly merging timekeeping with real-time environmental monitoring. The objective is to design a digital clock integrated with environmental sensors to display real-time temperature, humidity, and air quality readings. The primary aim is to deliver crucial environmental information concisely and in a user-friendly format. The project involves using a PCB, a microcontroller, Arduino, various sensors, and displays for integration and accurate data representation. In addition to its core functionalities, the clock features a solar battery as a power source, making the device portable; a voice controller and Bluetooth connectivity over an app can also be done. This small yet powerful device enhances timekeeping and gives users valuable insights into their environment, making it a versatile and space-efficient solution.

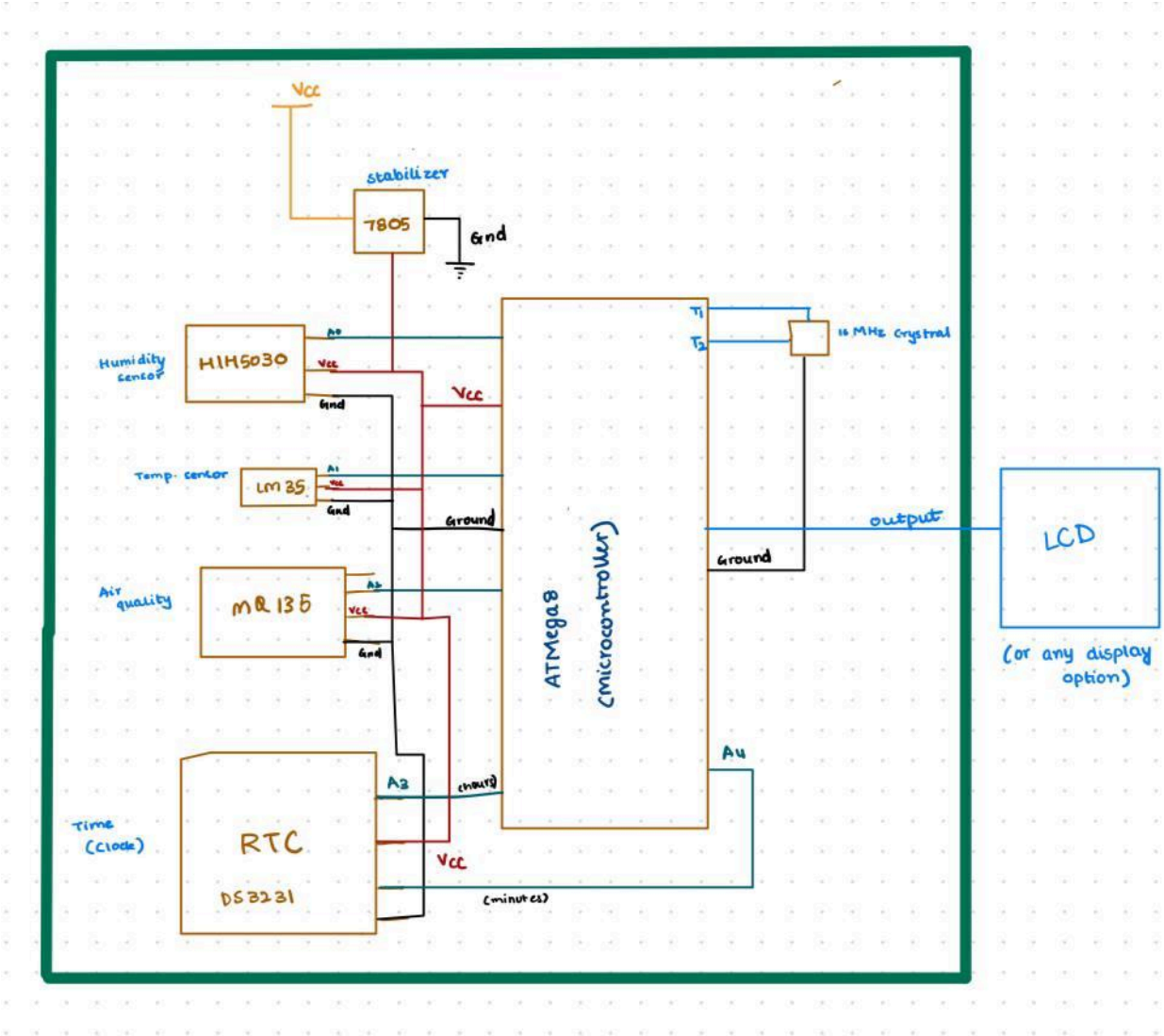
Work Division:
Anagha Balaji [EE22BTECH11204] - PCB Layout in KiCad, Air Pollution & CO_2 Sensor, Presentation, PCB Sourcing.
Chittampalli Ananya [EE22BTECH11206] - Schematic in KiCad, Humidity & CO_2 Sensor, Arduino Code, Report.
Gargi Behera [EE22BTECH11208] - PCB Layout in KiCad, Real-Time Clock (RTC), Component Sourcing, Presentation.
Rayi Giri Varshini [EE22BTECH11215] - Schematic in KiCad, Temperature Sensor, Arduino Code, Report.

Equipment Required:
1. Temperature Sensor (LM35)
2. Humidity Sensor (HIH 5030/31)
3. Air Quality Sensor (MQ135)
4. Microcontroller (ATmega8)
5. Resistors (330Ω, 65 kΩ, 20 kΩ, etc)
6. Capacitors (100uF,22pF)
7. Real-Time Clock (DS3231/DS1307)
8. 16 MHz Crystal Oscillator
9. Printed Circuit Board (PCB)
10. Display (LCD or Segment LED)
11. Arduino Board
12. Power Supply Components (Battery, Capacitors)

13. Casing

The sensors, microcontroller, clock, and other components are assembled in PCB. Proper layout, soldering, and integration are done for the device's functionality. The datasheets and guidelines of each component are attached to ensure correct interfacing and connectivity.

Block Diagram of Overall Circuit:



<u>Working Principles and Simulations</u>:

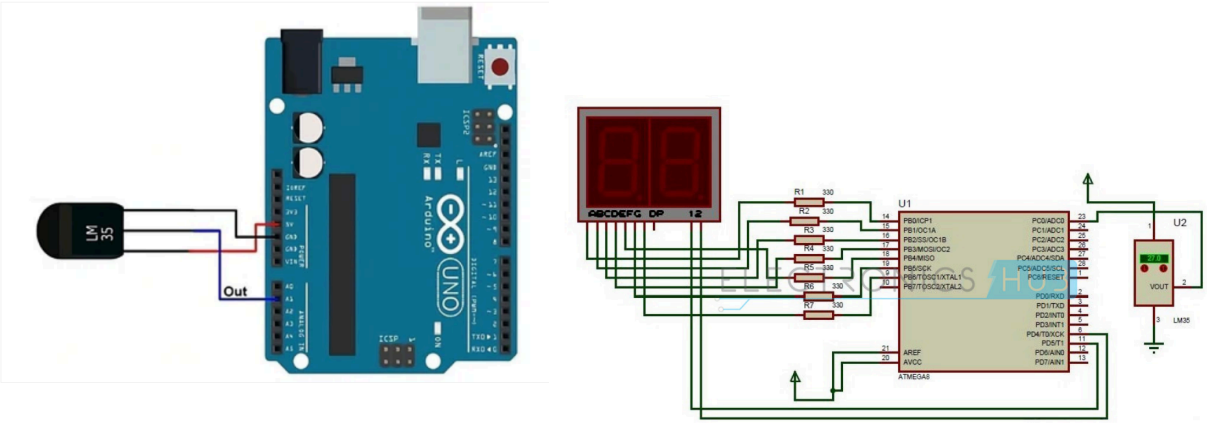
(a.) Temperature Sensor*:*

A temperature sensor is a device that measures the level of heat or cold in its surroundings. It converts the thermal data into an electrical signal, which is interpreted to measure the temperature reading. These operate in various principles like resistive, thermoelectric, infrared,

and semiconductor technologies. There are different types of temperature sensors, such as thermocouples, thermistors, RTDs, and infrared sensors.

- The temperature sensor uses electrical principles to detect the temperature value and converts it to an analog signal.
- The microcontroller picks up the analog signal and converts it into digital values. An op-amp is needed to amplify the analog temperature value before being fed to the ADC pin of the microcontroller.
- A display, LCD, or segment LED shows the current temperature value. Depending on the sensor's resolution, the display may also show temperature in decimal points.

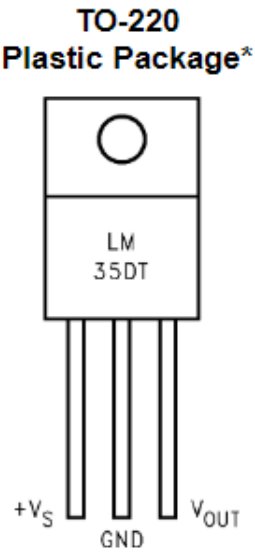
The connections are made from the sensor to the microcontroller with a resistor in the PCB, which is connected to the Arduino to code.

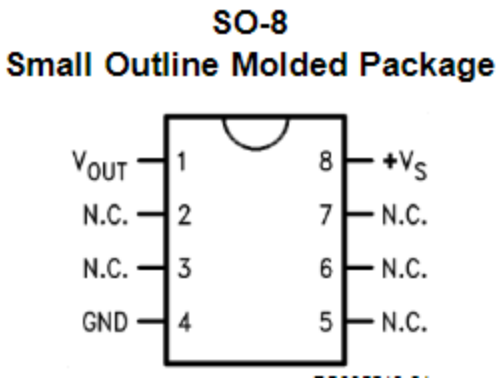


Required Materials for Measuring Temperature:

1. LM35 - Temperature Sensor

The LM35 is an integrated temperature sensor that measures the ambient temperature in degrees Celsius. Operating on a principle called proportional to absolute temperature (PTAT), the LM35 provides an output voltage directly proportional to the temperature it senses. A linear temperature-to-voltage conversion is widely used in various applications, including weather stations, temperature-controlled systems, and electronic devices.

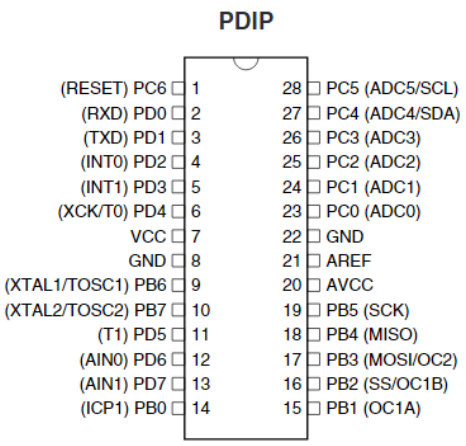


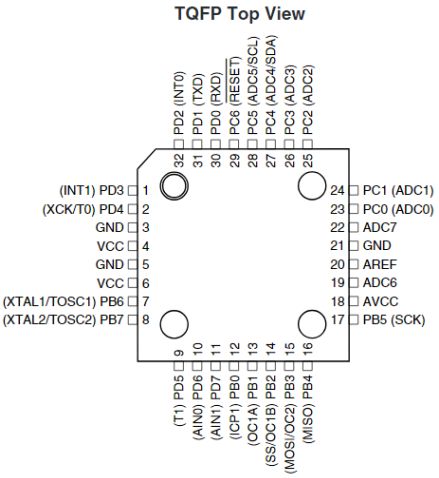


Datasheet: https://pdf1.alldatasheet.com/datasheet-pdf/view/517588/TI1/LM35.html

2. ATmega8 - Microcontroller

The ATmega8 is an 8-bit AVR RISC microcontroller developed by Atmel. It is part of the megaAVR series and is known for its versatility and low power consumption. It operates at 16 MHz, offers 8KB of in-system programmable flash memory for storing program code, is equipped with multiple input and output ports, and has multiple timers/counters. It also has a built-in ADC (Analog-to-Digital Converter) with a 10-bit resolution, allowing it to convert an analog input into a digital value from 0 to 1023.





Datasheet-https://pdf1.alldatasheet.com/datasheet-pdf/view/80247/ATMEL/ATMEGA8.html

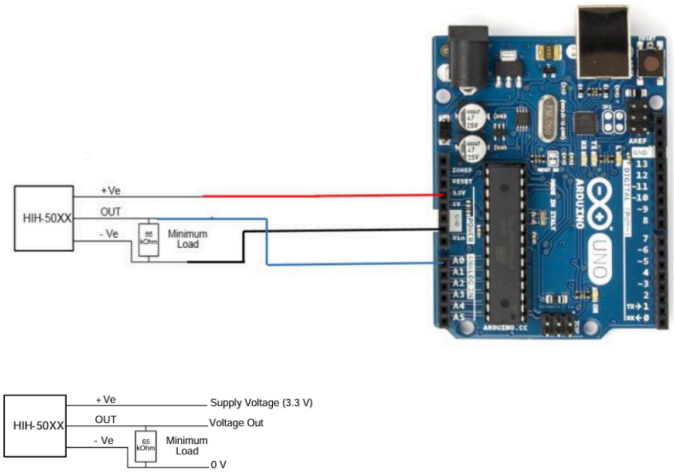
3. Resistors to connect to microcontroller and LCD.

***(b.) Humidity Sensor**:*

A humidity sensor measures the moisture level in its surroundings. It transforms the atmospheric moisture data into an electrical signal, representing the humidity level. Different

types of humidity sensors, such as hygrometers, capacitive humidity sensors, resistive humidity sensors, etc. The humidity sensors are influenced by temperature as the air's moisture-holding capacity changes with temperature.
It measures the humidity level of the immediate environment and presents the value in digital display. It involves a humidity sensor, microcontroller, resistors, and a digital display.

The connections are made from the sensor to the microcontroller with a resistor in the PCB, which is connected to the Arduino to code.



Required Materials for measuring humidity:

1. HIH 5030/31 - Low Voltage Humidity Sensor

The humidity sensor is known for its accuracy and reliability in measuring relative humidity. The sensor operates on a capacitive sensing principle, where changes in capacitance due to humidity variations are converted into electrical signals. It offers linear voltage output proportional to the humidity level.

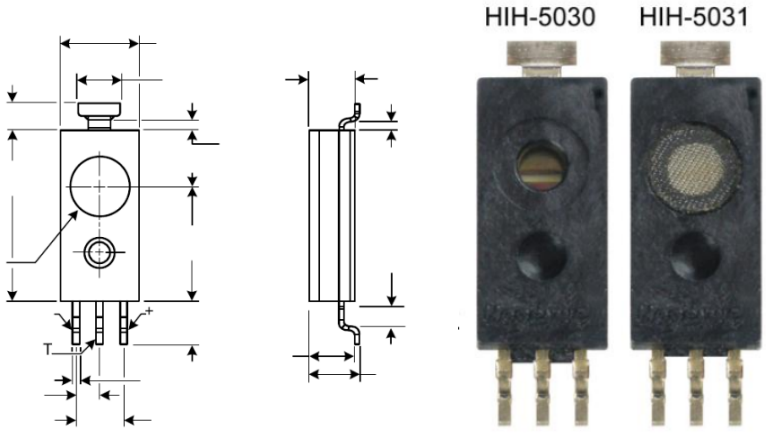


Table 1. Performance Specifications (At 3.3 Vdc supply and 25 °C [77 °F] unless otherwise noted.)

Parameter	Minimum	Typical	Maximum	Unit	Specific Note
Interchangeability (first order curve) 0% RH to 10% RH, 90% RH to 100% RH 11% RH to 89% RH	 -7 -3	 – –	 7 3	 % RH % RH	 –
Accuracy (best fit straight line) 11% RH to 89% RH	-3	–	+3	% RH	4
Hysteresis	–	2	–	% RH	–
Repeatability	–	±0.5	–	% RH	–
Settling time	–	–	70	ms	–
Response time (1/e in slow moving air)	–	5	–	s	–
Stability (at 50% RH in 5 years)	–	±1.2	–	% RH	1
Voltage supply	2.7	–	5.5	Vdc	2
Current supply	–	200	500	µA	–
Voltage output (1st order curve fit)	$V_{OUT}=(V_{SUPPLY})(0.00636(\text{sensor RH}) + 0.1515)$, typical at 25 °C				
Temperature compensation	True RH = (Sensor RH)/(1.0546 – 0.00216T), T in °C				
Output voltage temp. coefficient at 50% RH, 3.3 V	–	-2	–	mV/°C	–
Operating temperature	-40[-40]	See Figure 2.	85[185]	°C[°F]	–
Operating humidity (HIH-5030)	0	See Figure 2.	100	% RH	3
Operating humidity (HIH-5031)	0	See Figure 2.	100	% RH	–
Storage temperature	-50[-58]	–	125[257]	°C [°F]	–

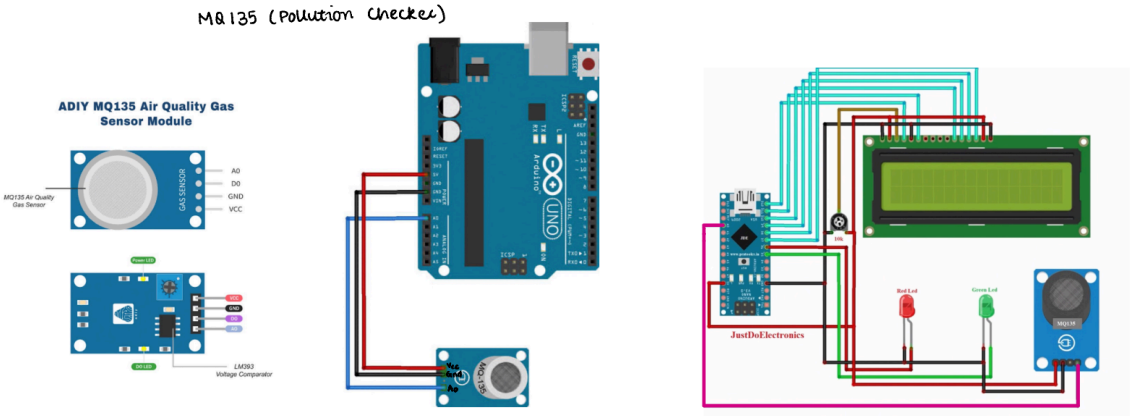
Datasheet-https://pdf1.alldatasheet.com/datasheet-pdf/view/345572/HONEYWELL/HIH-5030.html

2. Resistor (65 kΩ) to connect to the microcontroller.
3. Microcontroller (ATmega8)

***(c.) Air Quality Sensor (Pollution Sensor)**:*

It is designed to measure the composition of the air in its surroundings, assessing pollutant levels and gas concentrations. It translates complex air quality data into an electrical signal, providing the environmental conditions. Various air quality sensors exist, like gas sensors, particulate matter sensors, and volatile organic compounds (VOC) sensors. Air quality sensors are also influenced by temperature and humidity levels.

The connections are made from the sensor to the microcontroller with a resistor in the PCB, which is connected to the Arduino to code.

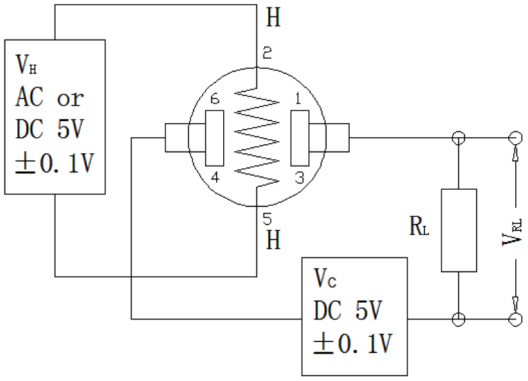


Required Materials for Measuring Air Quality:

1. MQ135 - Air Quality Gas Sensor

It is a gas sensor that detects various gases, including ammonia (NH_3), Carbon dioxide (CO_2), Sulfide, Methane, and Benzene series stream. Operating on the principle of resistance changes in the presence of specific gases, the MQ135 translates gas concentrations into electrical signals. The sensor's resistance decreases as gas concentration increases. Integrating an analog output, the MQ135 requires an analog-to-digital converter or a microcontroller to

interpret and display gas concentration levels. The detection range is 10-1000 ppm (Ammonia gas, Toluene, Hydrogen, Smoke).



Model			MQ135
Sensor Type			Semiconductor
Standard Encapsulation			Bakelite, Metal cap
Target Gas			ammonia gas, sulfide, benzene series steam
Detection range			10~1000ppm(ammonia gas, toluene, hydrogen, smoke)
Standard Circuit Conditions	Loop Voltage	V_c	≤24V DC
	Heater Voltage	V_H	5.0V±0.1V AC or DC
	Load Resistance	R_L	Adjustable
Sensor character under standard test conditions	Heater Resistance	R_H	29Ω±3Ω (room tem.)
	Heater consumption	P_H	≤950mW
	Sensitivity	S	Rs(in air)/Rs(in 400ppm H_2)≥5
	Output Voltage	Vs	2.0V~4.0V（in 400ppm H_2）
	Concentration Slope	α	≤0.6(R_{400ppm}/R_{100ppm} H_2)
Standard test conditions	Tem. Humidity		20℃±2℃；55%±5%RH
	Standard test circuit		Vc:5.0V±0.1V；V_H: 5.0V±0.1V
	Preheat time		Over 48 hours

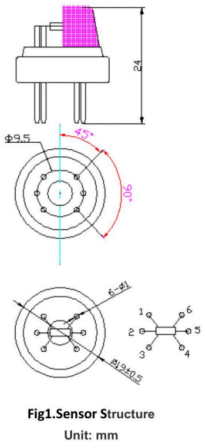


Fig1.Sensor Structure
Unit: mm

The Resistance value of MQ-135 differs for various kinds and concentrations of gases. So, sensitivity adjustment is necessary. The detector for 100 ppm NH_3 or 50 ppm Alcohol concentration in air is recommended.
Datasheet-https://pdf1.alldatasheet.com/datasheet-pdf/view/1307647/WINSEN/MQ135.html

2. Resistor about 20kΩ (10kΩ - 47kΩ) to connect to the microcontroller.
3. Microcontroller (ATmega8)

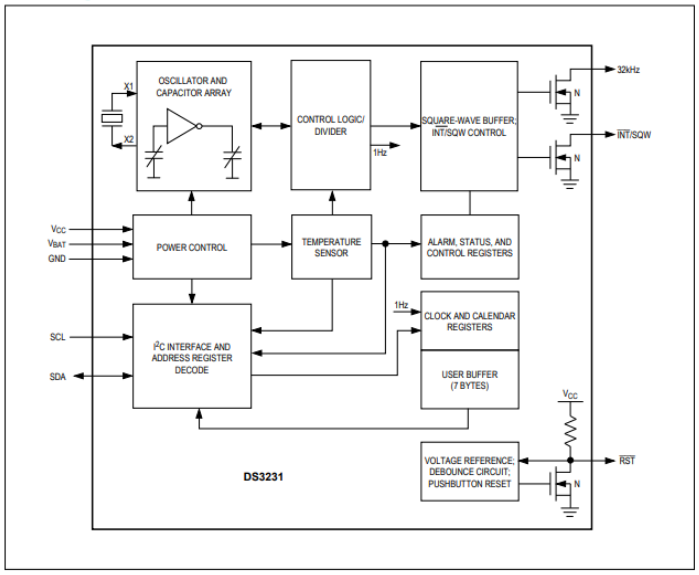
(d.) Clock:

The design aims to create a 12-hour clock using an Atmega8 microcontroller, a 4-digit 7-segment display, and an RTC (Real-Time Clock) module such as DS3231 or DS1307. The Atmega328p is programmed to control the display and synchronize time with the RTC module.

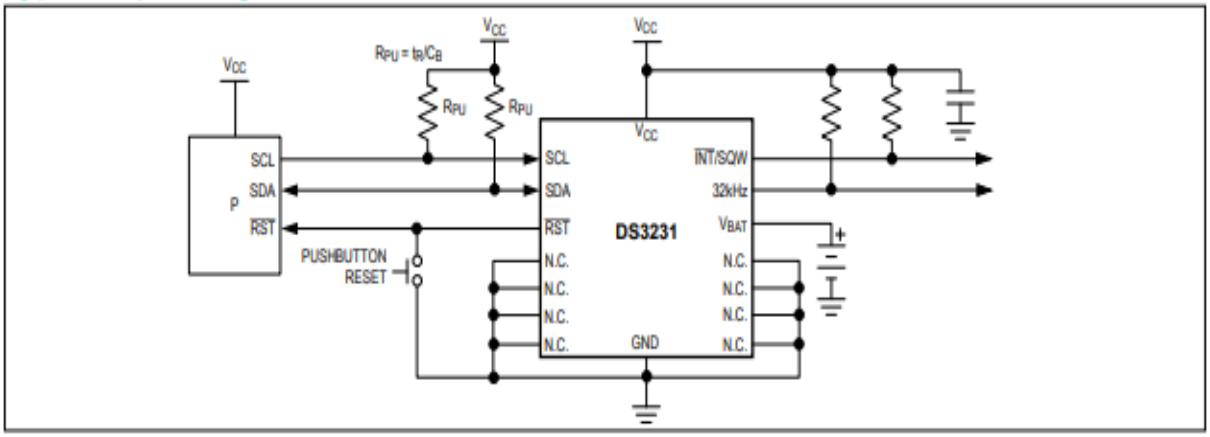
Required Materials for Clock:

1. DS3231/DS1307 RTC module.
   The Real-Time Clock (RTC) module keeps track of the current time. It provides accurate timekeeping independent of the microcontroller. The microcontroller queries the RTC module to obtain the current time and update the display. It communicates with the Atmega8 through the I2C protocol.

Block Diagram



Typical Operating Circuit



Datasheet- https://www.analog.com/media/en/technical-documentation/data-sheets/ds3231.pdf

2. Atmega8p microcontroller
   The Atmega8 is the brain of the clock. It executes the program, manages the display, and communicates with the RTC module. It controls the multiplexing of the 7-segment display, reads time from the RTC, and updates the display accordingly.

3. 16MHz crystal.
   Provides a stable clock source for the Atmega328p microcontroller. It also ensures accurate and reliable execution of the program on the microcontroller.

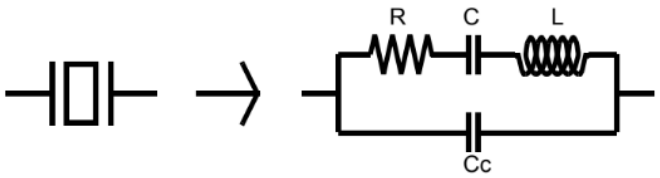
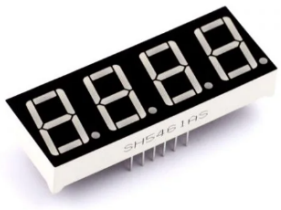


Figure 2. Crystal Electrical Equivalent Schematic

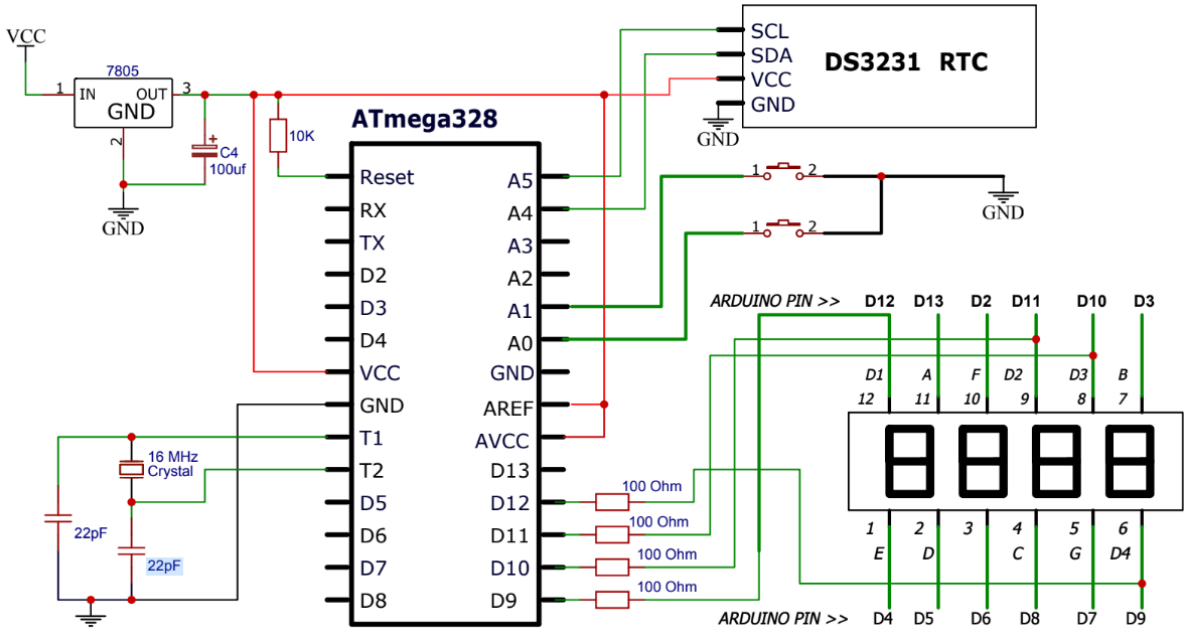
Datasheet- https://www.nxp.com/docs/en/application-note/AN2500.pdf

4. 4-digit 7-segment display.

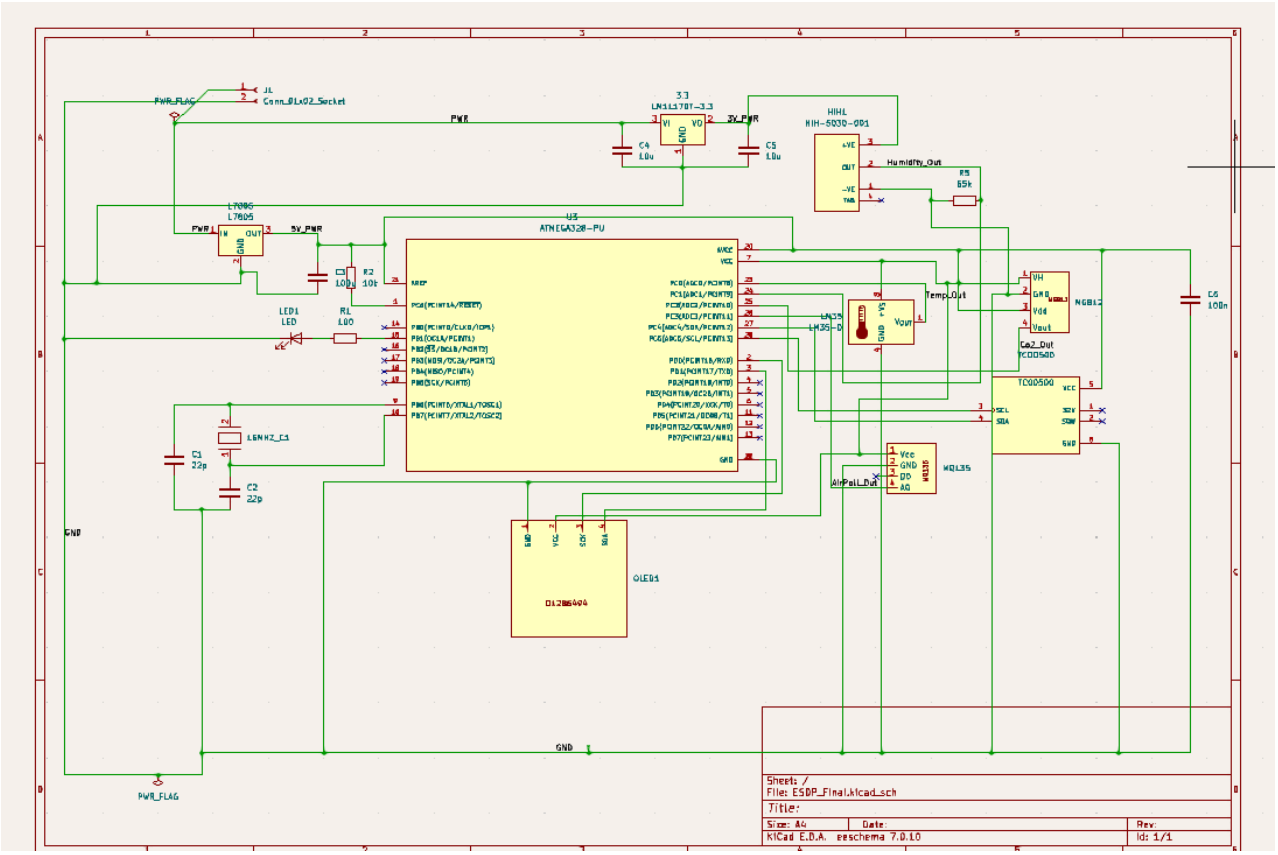


5. Resistors and wiring components.
6. Arduino IDE for coding and uploading the program.

The Atmega8 is connected to the 7-segment display and the RTC module. The 7-segment display is multiplexed to display hours and minutes, and the RTC module is used to keep track of real-time. The crystal oscillator ensures accurate timekeeping.

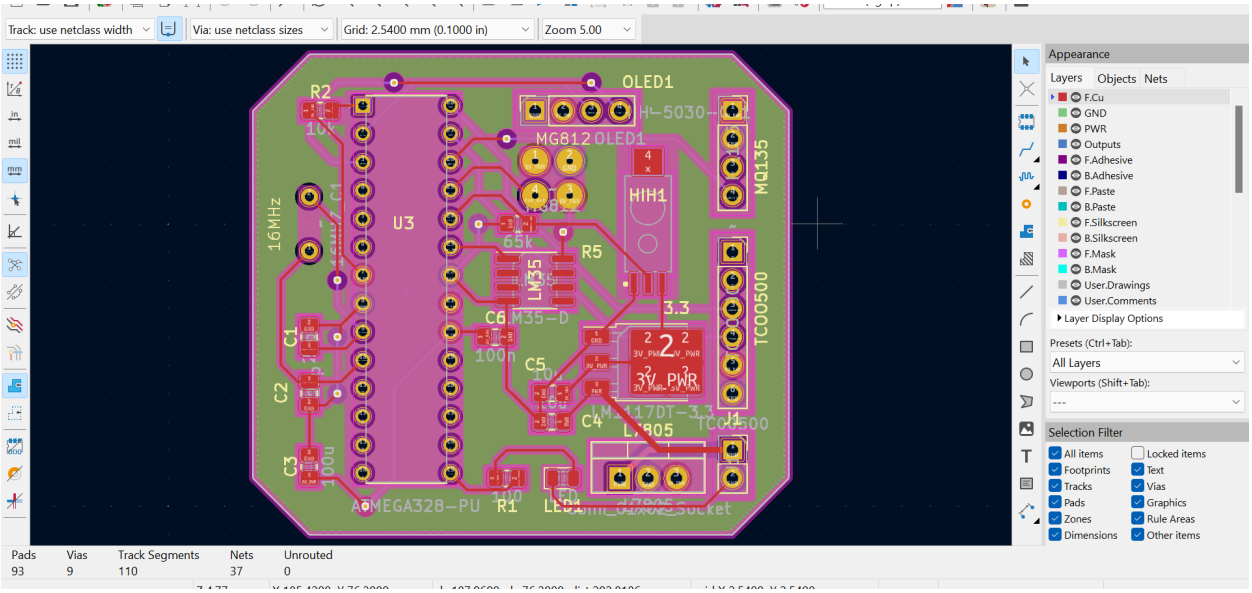


Schematic Snapshot:

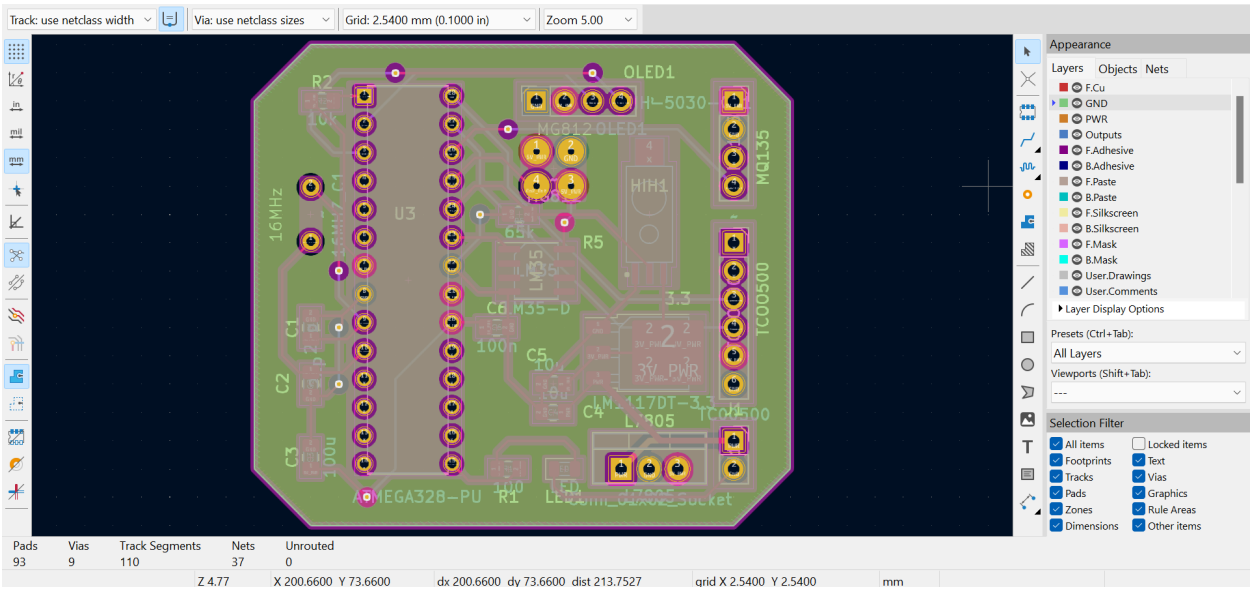


PCB Snapshot:

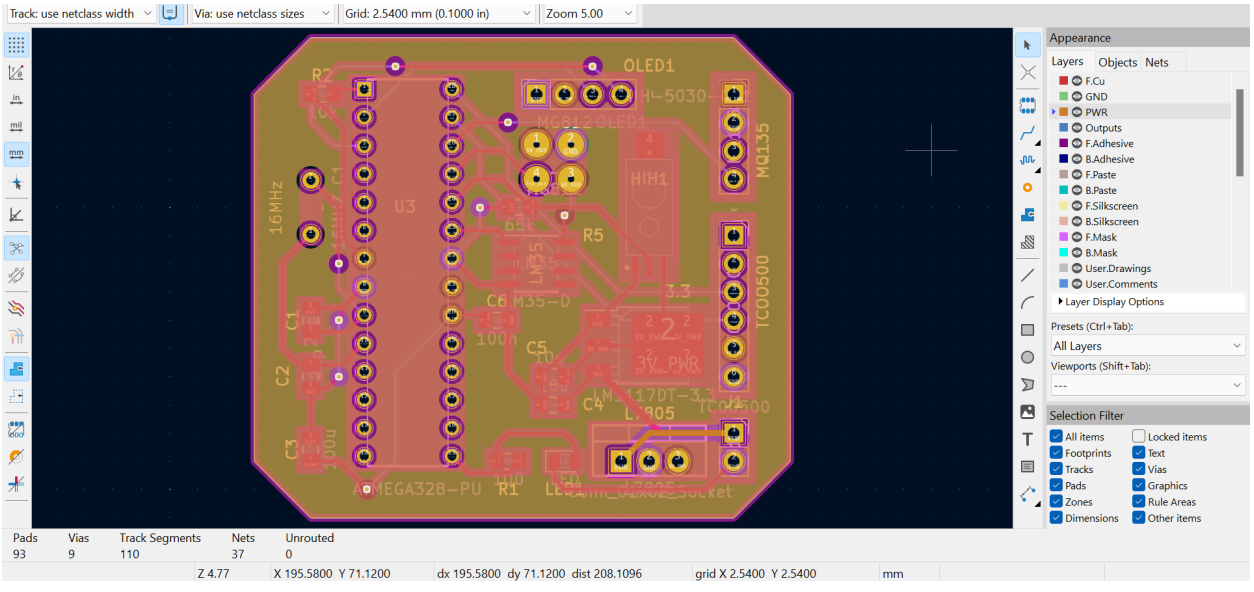
Layer-1:



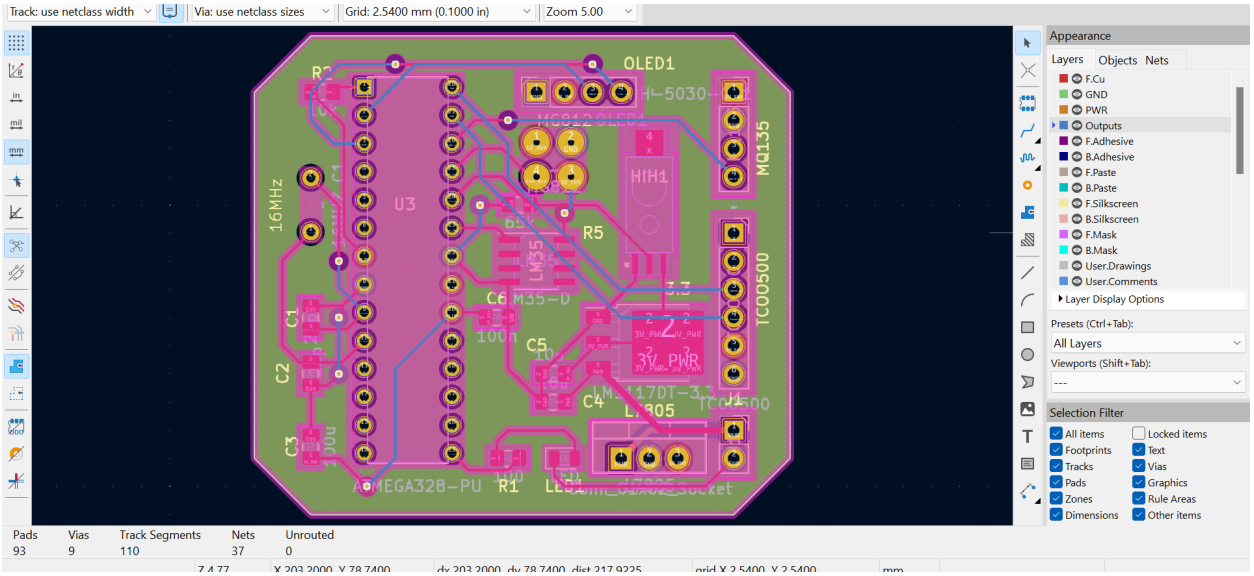
Layer-2: GND



Layer-3:PWR

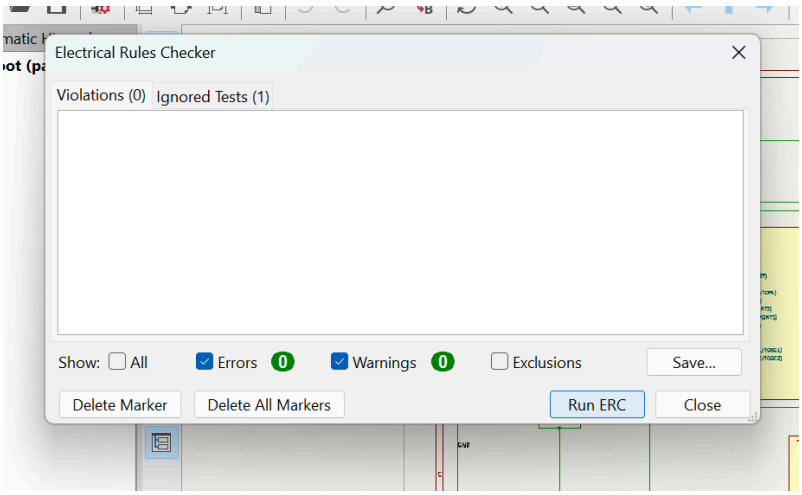


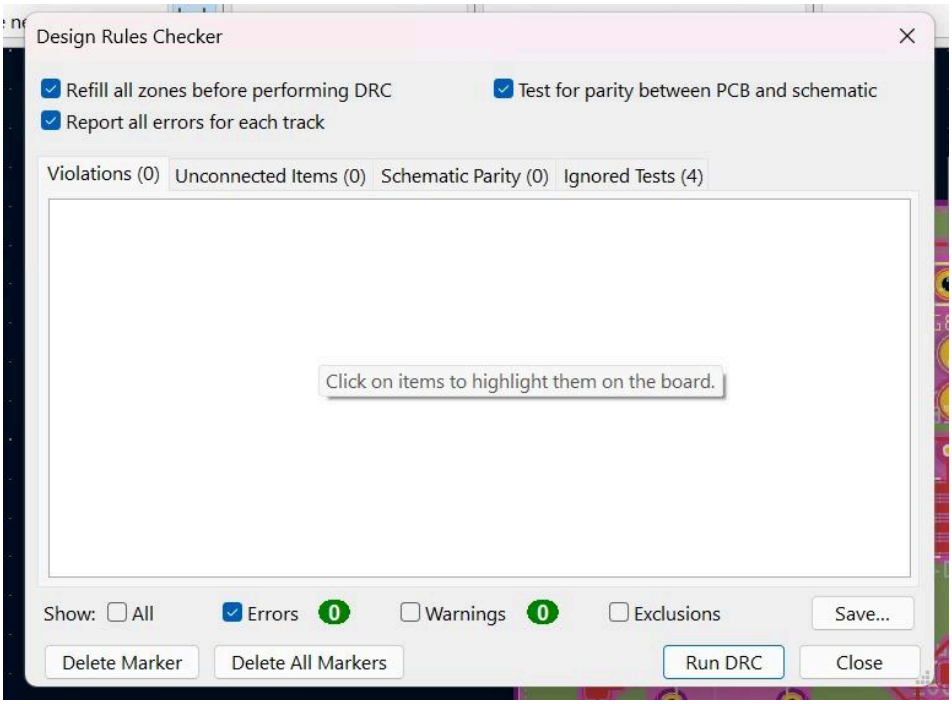
Layer-4:

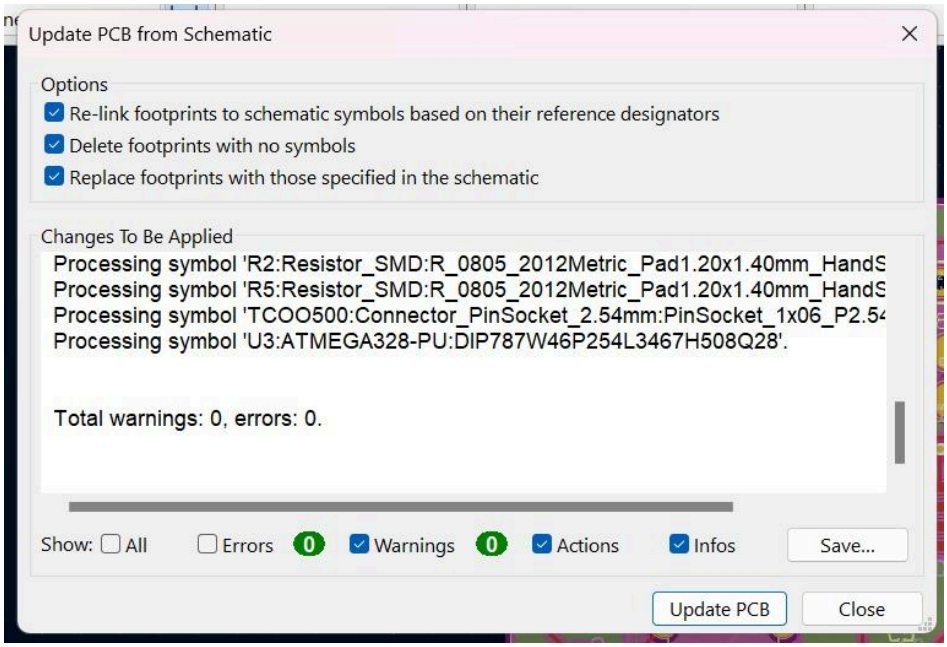


PCB Picture:

LVC, ERC & DRC Report:

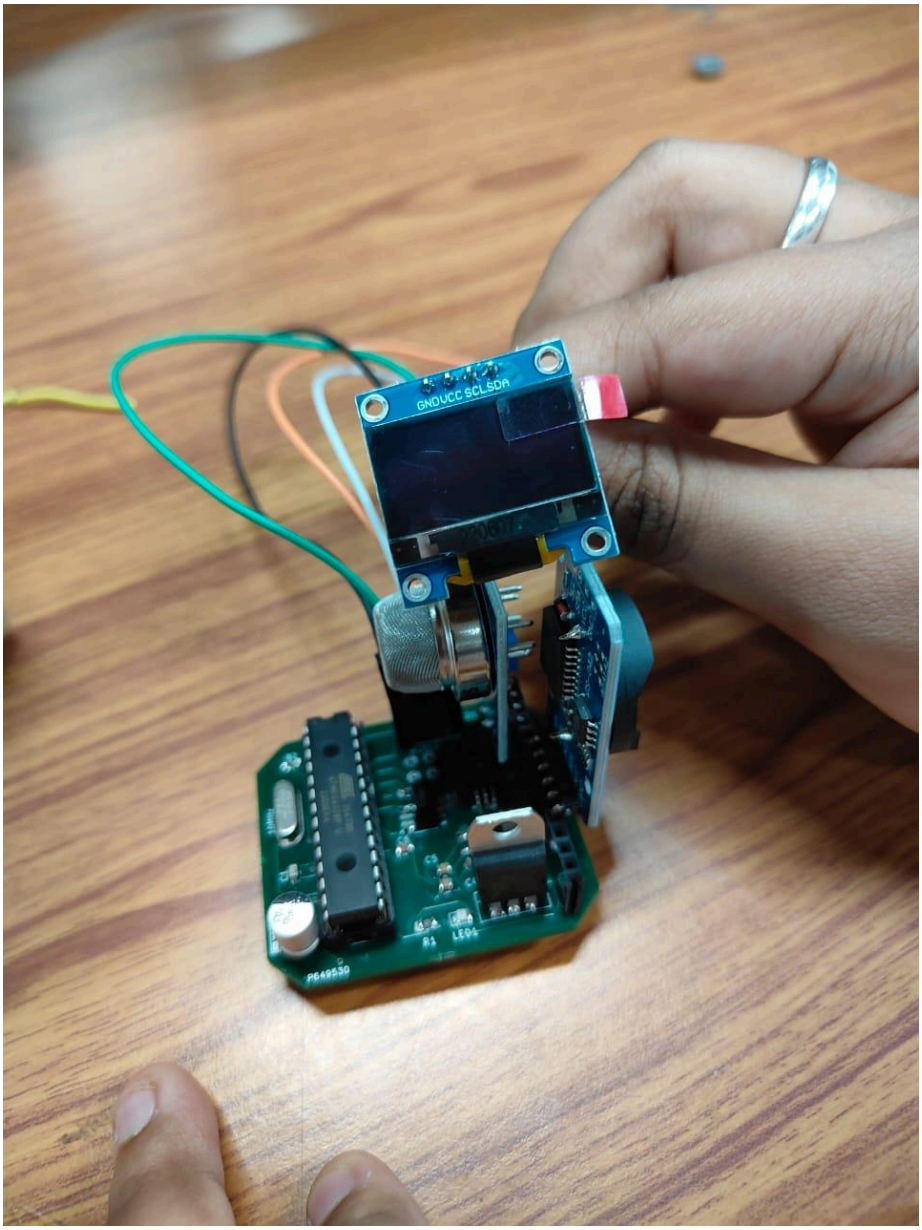






Presentation Link: [Eco Clock]

After fabricating PCB:

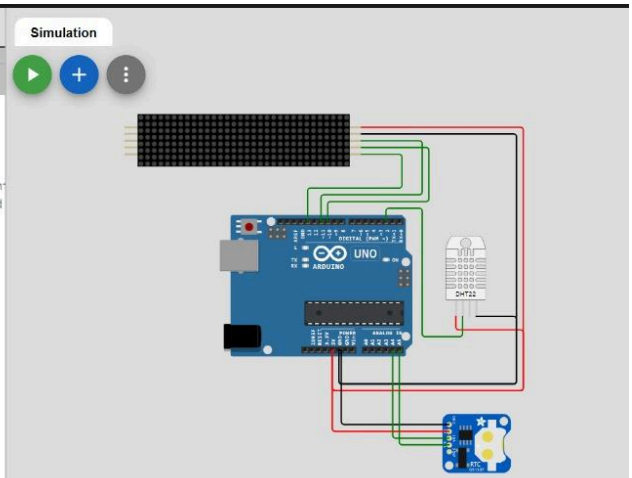


Arduino Code with online simulation:

sketch.ino | diagram.json | Font7Seg.h | libraries.txt ● | Library Manager

```ino
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <DHT.h>
#include <SPI.h>
#include <Wire.h>
#include "Font7Seg.h"

// Define the number of devices we have in the chain and the hardware int
// NOTE: These pin numbers will probably not work with your hardware and
// need to be adapted
#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW
#define MAX_DEVICES 4 // Define the number of displays connected
#define CLK_PIN   13 // CLK or SCK
#define DATA_PIN  11 // DATA or MOSI
#define CS_PIN    10 // CS or SS
#define SPEED_TIME 75 // Speed of the transition
#define PAUSE_TIME  0
#define MAX_MESG  20

// These are for the clock
#define DS1307_ADDRESS 0x68

// These are for the temperature
#define DHTPIN 2
#define DHTTYPE DHT22
#define TIMEDHT 1000
```

Simulation

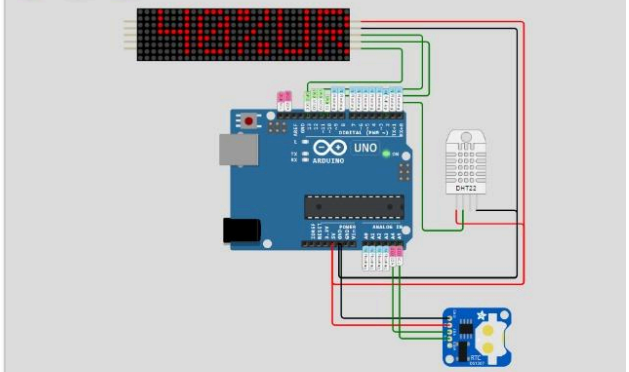


sketch.ino | diagram.json | Font7Seg.h | libraries.txt ● | Library Manager

```ino

// Global variables
uint8_t wday, mday, month, year;
uint8_t hours, minutes, seconds;

char szTime[9];    // mm:ss\0
char szMesg[MAX_MESG + 1] = "";

float humidity, celsius, fahrenheit;

uint8_t degC[] = { 6, 3, 3, 56, 68, 68, 68 }; // Deg C
uint8_t degF[] = { 6, 3, 3, 124, 20, 20, 4 }; // Deg F

uint8_t clear = 0x00;

uint32_t timerDHT = TIMEDHT;

DHT dht(DHTPIN, DHTTYPE);

// Hardware SPI connection
MD_Parola P = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

void beginDS1307()
{
  // Read the values (date and time) of the DS1307 module
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire.write(clear);
```

Simulation

00:07.096 84%



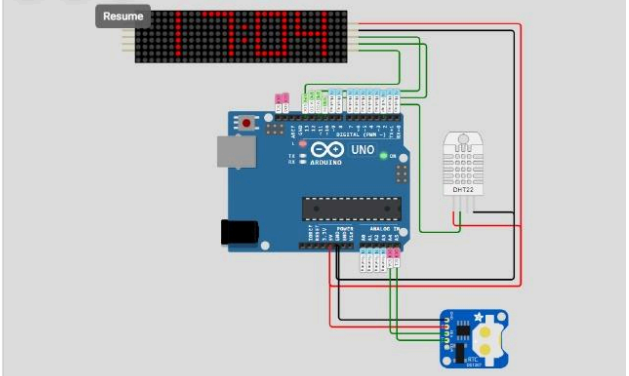
sketch.ino | diagram.json | Font7Seg.h | libraries.txt ● | Library Manager

```ino
{

  seconds = bcdToDec(Wire.read());
  minutes = bcdToDec(Wire.read());
  hours = bcdToDec(Wire.read() & 0xff);
  wday = bcdToDec(Wire.read());
  mday = bcdToDec(Wire.read());
  month = bcdToDec(Wire.read());
  year = bcdToDec(Wire.read());
}

uint8_t decToBcd(uint8_t value)
{
  return ((value / 10 * 16) + (value % 10));
}

uint8_t bcdToDec(uint8_t value)
{
  return ((value / 16 * 10) + (value % 16));
}

// Code for reading clock time
void getTime(char *psz, bool f = true)
{
  sprintf(psz, "%02d%c%02d", hours, (f ? ':' : ' '), minutes);
}
```

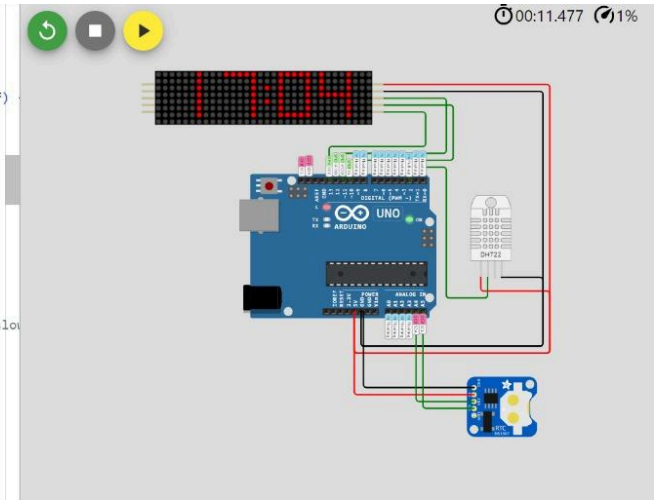
Simulation

00:11.477 1%

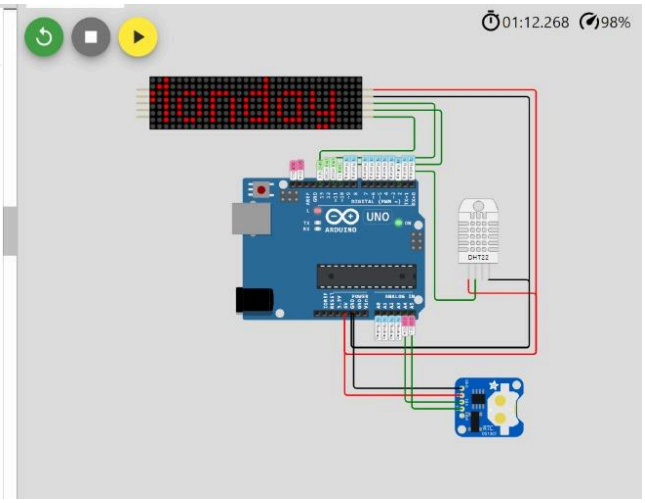
Resume



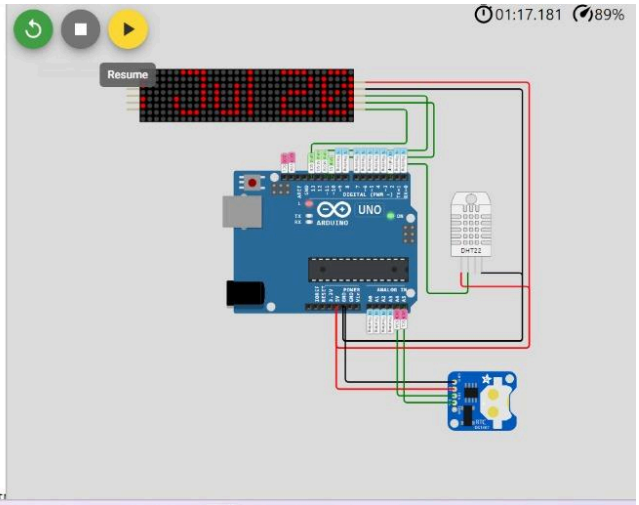
```
// Code for reading clock date
void getDate(char *psz)
{
  char  szBuf[10];
  sprintf(psz, "%d %s %04d", mday , mon2str(month, szBuf, sizeof(szBuf)) 
}

// Code for get Temperature
void getTemperature()
{
  // Wait for a time between measurements
  if ((millis() - timerDHT) > TIMEDHT) {
    // Update the timer
    timerDHT = millis();

    // Reading temperature or humidity takes about 250 milliseconds!
    // Sensor readings may also be up to 2 seconds 'old' (its a very slo
    humidity = dht.readHumidity();

    // Read temperature as Celsius (the default)
    celsius = dht.readTemperature();

    // Read temperature as Fahrenheit (isFahrenheit = true)
    fahrenheit = dht.readTemperature(true);

    // Check if any reads failed and exit early (to try again)
```



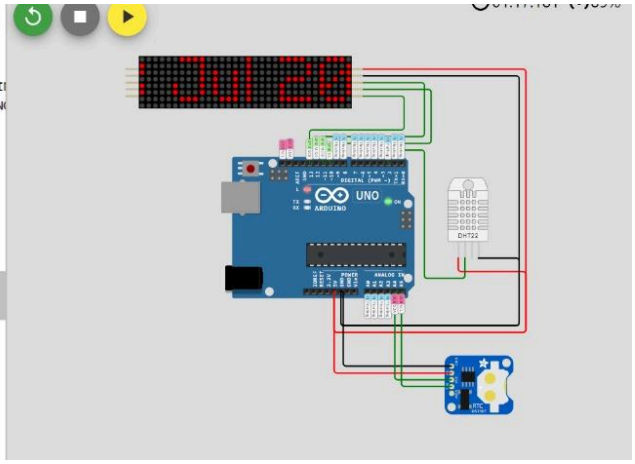

```
  {
    if ((millis() - timerDHT) > TIMEDHT) {
      if (isnan(humidity) || isnan(celsius) || isnan(fahrenheit)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
      }
    }
  }

  // Get a label from PROGMEM into a char array
  char *mon2str(uint8_t mon, char *psz, uint8_t len)
  {
    static const __FlashStringHelper* str[] =
    {
      F("Jan"), F("Feb"), F("Mar"), F("Apr"),
      F("May"), F("Jun"), F("Jul"), F("Aug"),
      F("Sep"), F("Oct"), F("Nov"), F("Dec")
    };

    strncpy_P(psz, (const char PROGMEM *)str[mon - 1], len);
    psz[len] = '\0';

    return (psz);
  }

  char *dow2str(uint8_t code, char *psz, uint8_t len)
```



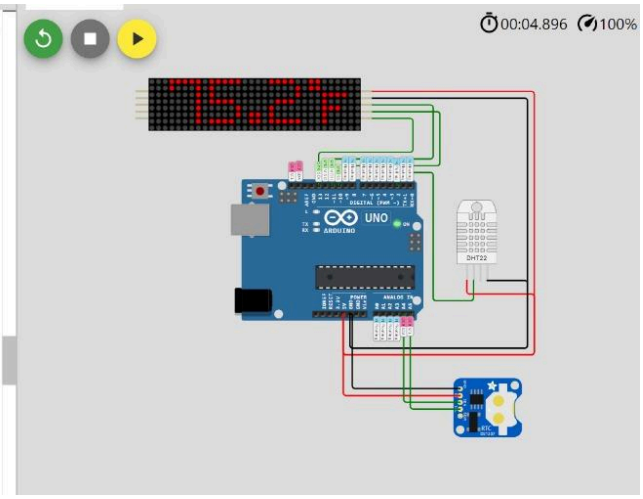

```
char *dow2str(uint8_t code, char *psz, uint8_t len)
{
  static const __FlashStringHelper* str[] =
  {
    F("Sunday"), F("Monday"), F("Tuesday"),
    F("Wednesday"), F("Thursday"), F("Friday"),
    F("Saturday")
  };

  strncpy_P(psz, (const char PROGMEM *)str[code - 1], len);

  psz[len] = '\0';

  return (psz);
}

void setup(void)
{
  Wire.begin();

  P.begin(2);
  P.setInvert(false);

  P.setZone(0,  MAX_DEVICES - 4, MAX_DEVICES - 1);
  P.setZone(1, MAX_DEVICES - 4, MAX_DEVICES - 1);
```



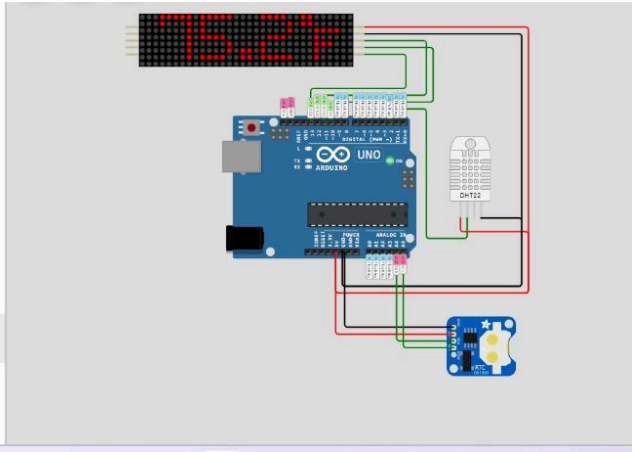
```
153
154   P.setZone(0,  MAX_DEVICES - 4, MAX_DEVICES - 1);
155   P.setZone(1, MAX_DEVICES - 4, MAX_DEVICES - 1);
156
157   P.displayZoneText(1, szTime, PA_CENTER, SPEED_TIME, PAUSE_TIME, PA_PRI
158   P.displayZoneText(0, szMesg, PA_CENTER, SPEED_TIME, 0, PA_PRINT , PA_N
159
160   P.addChar('$', degC);
161   P.addChar('&', degF);
162
163   dht.begin();
164 }
165
166 void loop(void)
167 {
168   static uint32_t lastTime = 0; // Memory (ms)
169   static uint8_t  display = 0;  // Current display mode
170   static bool flasher = false;  // Seconds passing flasher
171
172   beginDS1307();
173   getTemperature();
174
175   P.displayAnimate();
176
177   if (P.getZoneStatus(0))
```




```
167 {
176
177   if (P.getZoneStatus(0))
178   {
179     switch (display)
180     {
181     case 0: // Temperature deg Celsius
182       P.setPause(0, 1000);
183       P.setTextEffect(0, PA_SCROLL_LEFT, PA_SCROLL_UP);
184       display++;
185       dtostrf(celsius, 3, 1, szMesg);
186       strcat(szMesg, "$");
187
188       break;
189     case 1: // Temperature deg Fahrenheit
190       P.setTextEffect(0, PA_SCROLL_UP, PA_SCROLL_DOWN);
191       display++;
192       dtostrf(fahrenheit, 3, 1, szMesg);
193       strcat(szMesg, "&");
194
195       break;
196     case 2: // Humidity
197       P.setTextEffect(0, PA_SCROLL_DOWN, PA_SCROLL_LEFT);
198       display++;
199       dtostrf(humidity, 3, 0, szMesg);
200       strcat(szMesg, "%UR");
```




```
199       dtostrf(humidity, 3, 0, szMesg);
200       strcat(szMesg, "%UR");
201
202       break;
203     case 3: // Clock
204       P.setFont(0, numeric7Seg);
205       P.setTextEffect(0, PA_PRINT, PA_NO_EFFECT);
206       P.setPause(0, 0);
207
208       if ((millis() - lastTime) >= 1000)
209       {
210         lastTime = millis();
211         getTime(szMesg, flasher);
212         flasher = !flasher;
213       }
214
215       if ((seconds == 00) && (seconds <= 30)) {
216         display++;
217         P.setTextEffect(0, PA_PRINT, PA_WIPE_CURSOR);
218       }
219
220       break;
221     case 4: // Day of week
```



```
      break;
    case 4: // Day of week
      P.setFont(0, nullptr);
      P.setTextEffect(0, PA_SCROLL_LEFT, PA_SCROLL_LEFT);
      display++;
      dow2str(wday, szMesg, MAX_MESG);

      break;
    default: // Calendar
      P.setTextEffect(0, PA_SCROLL_LEFT, PA_SCROLL_LEFT);
      display = 0;
      getDate(szMesg);

      break;
    }

    P.displayReset(0); // Rest zone zero
  }
}

```