Liceul Teoretic "Ernő Solomon" Gheorgheni

Lucrare de atestat disciplina INFORMATICĂ

Profesor coordonator:
Zoltán Csíki

Elev:
Dear Attila-János

Gheorgheni
2025

Smart Greenhouse Controller

-IoT project/Automated system-

Content

Introduction

While thinking about my exam paper, I had several different project ideas. But I was sure that what I wanted to do should be useful and of some value. I wanted to create something that was not only interesting for me, but that could be used by anyone else.

Automation is a technology that has been around for a long time, but I think that it is not available to everyone and that few people use it, especially in the agricultural sector, where it is possible to achieve great productivity results with simple automation. But if automation can be so useful, the question arises why it is not widespread. I think this question arises because these technologies are not explicitly available to individuals and only big-boned agricultural companies can afford such improvements. In my view, open source projects can provide a solution. This will make it easier to bring the technology to users who have not had access to it before.

IoT, short for "Internet **of things"**, is the English acronym for the Internet **of things**. It short for a variety of electronic devices that can detect, possibly process, some essential information and communicate with another device on the network. This explains why it is important for an automated controller to belong to this IoT group. One of the main features of a greenhouse controller is that it can control the current parameters even in the absence of the user, something an IoT device can do.

The main value of this system lies in the processing of sensory values. I designed it to be able to measure all the parameters that characterise a greenhouse. I considered humidity, temperature, ground humidity and light intensity to be less important, but I left the measurement of carbon dioxide levels as an extensible function. I considered it important to study the external temperature and humidity to track how environmental factors affect the internal condition of the greenhouse.

In addition to web-based management, it is important that the functions can be seen and modified on the spot. Monitoring is made possible by a 0.96 inch oled display,

which displays the data broken down into menu items. A joystick module allows you to make changes.

Of course, in addition to processing and managing data, it is essential to actively and artificially influence the internal environment. This is done by a 6-channel relay module that switches on and off the light supply, humidification, irrigation and fan status.

Ultimately, I think it is important to create automation systems that are accessible to a wide range of users, both individuals and small businesses. Furthermore, it is also important that such an automation system is open source, so that it can be understood by everyone and can be further developed in the future. From this point of view, I would like to recall an idea of *Linus Torvalds*: "Intelligence comes from the ability of people to use and develop the ideas of others."

Development opportunities

The current greenhouse control system successfully implements the basic functions in many respects, such as temperature and humidity measurement and display on an OLED display, and the transmission of sensor data to an internal network server via the ESP8266 module. At the same time, the system can be further developed and optimised in several ways that not only increase performance, but also significantly improve system reliability and functionality.

1. Memory optimisation

In its current state, the system operates with redundant, repetitive text in several places, for example when writing to the OLED display. These text strings should be organized in static character arrays or in #define or const char* variables to reduce the load on SRAM. In addition, it is recommended to use C-style char[] arrays instead of the String type, as the String type may fragment memory during long runs.

2. Omission of AT commands

The ESP8266 is currently probably controlled via AT commands, which is not the most efficient solution. Instead of AT firmware, it is recommended to NodeMCU (Lua) or the ESP8266 Arduino core, and the microcontroller

direct programming. This way, you don't need to send AT commands over a serial connection, but the ESP8266 itself takes control and the Arduino Mega only transmits sensor data. This increases response time, reduces error possibilities and extends the possibilities (e.g. HTTPS connection, JSON handling).

3. Accessibility from an external network

The system is currently only available on the internal network. A next development step could be to make the web interface available via the Internet. This can be achieved by using dynamic DNS (DDNS), port forwarding in the router, or - as a more secure and scalable solution - by sending data to a cloud-based server (e.g. using MQTT, HTTPS POST REST API). The data can even be displayed in a real-time web dashboard, e.g. Node-RED.

4. Data security and encryption

Currently, the data being transmitted is likely to travel unencrypted over the network, so an important development is the introduction of encrypted data communication. This could be done using HTTPS or lower level AES encryption (either Arduino side encryption, server side decryption). This is particularly important if the device is connected to the internet or would handle sensitive data in the future.

5. Modular and reusable code

By structuring the project further, a more general, easily reusable code base can be created. It is advisable to organise the handling of individual hardware elements into separate classes or function groups (e.g. SHT sensor handler, display controller, network communication module). This not only makes the code more readable and maintainable, but also prepares the project for future expansion into a multi-module system (e.g. multi-zone greenhouse).

6. Fault tolerance and diagnostics

It would be useful to have a basic fault-tolerant system, for example using a watchdog timer, which restarts the system if it freezes or if no data is received for a certain period of time. In addition, a logging facility that saves errors or major events to an SD card or the cloud is useful.

Development environment

Arduino IDE 2.0

Programming an Arduino, like most programming languages, uses what is called an **Integrated Development Environment, or** IDE. **An integrated development environment (IDE) is** a software application that provides computer programmers with comprehensive facilities for software development. An IDE usually consists of at least a source code editor, automation tools and a debugger.

There are several IDEs to choose from for programming the arduino. Arduino officially offers two different Arduino IDEs for developers:

Arduino IDE 1.x - This is the classic version, which was the default development environment for a long time. It is simple, fast, and widely supported.

Arduino IDE 2.x - The next generation Arduino IDE, offering a modern user interface, advanced editing features (e.g. auto-complete code, dark mode) and improved debugging capabilities.

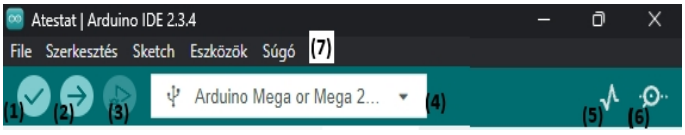
I have chosen to develop with Arduino IDE 2.0, as this version provides a more developer-friendly environment.

This IDE provides easy debugging possibilities. A so-called serial monitor is available, which allows you to keep track of variables, runtime, and correct function calls. This can be done by a simple Serial.print(Val, Format) line inserted in the code.

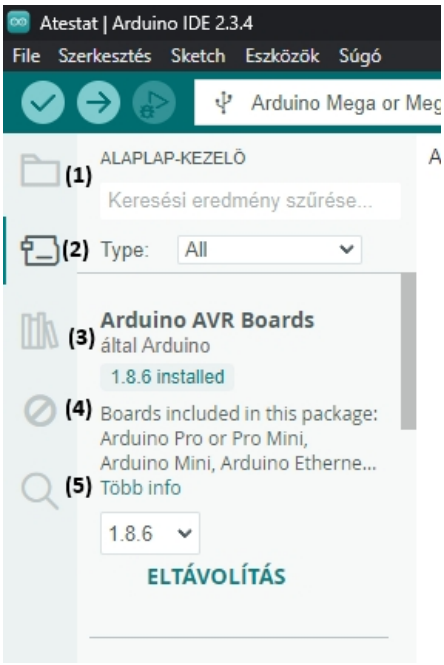
Val - The data we want to print. It can be a number, string, character or variable.

Format- This is an optional parameter that specifies the format in which the value should be displayed.

Overview of arduino IDE 2



(1) Error correction
(2) Upload, upload the program to for microcontroller
(3) Check/debug



(1.) Managing files in use
(2.) Motherboard manager, installing different motherboards
(3.) Library manager, installing libraries (4.) Debugging, debugger
(5.) Search by keyword in the project

Version control

Version management is the management of data with multiple versions. Version management systems are most commonly used in engineering and software development to manage versions of documents under development, designs, source code and other data that multiple people are working on simultaneously. Each change is tracked by version numbers or version letters.

Version Control Systems (VCS) are software tools that allow you to track changes to program code or other digital files over time. During software development, the source code is constantly changing: new features are added, bugs are fixed, or experimental changes are made by developers. A well-functioning version control system allows these changes to be documented and retrieved, and allows multiple developers to work on the same project at the same time without accidentally overwriting each other's work.

With version management, all changes are logged: who changed what, when and what. This not only makes development more transparent and secure, but also allows you to restore a stable state to a previous state if, for example, a new change causes a bug. Version management not only supports teamwork, but also organises and archives the work of individual developers.

The best-known and most widely used version control system today is Git, developed by Linus Torvalds, the creator of the Linux operating system. Git is a distributed version control system, so each developer has the history of the entire project on his own machine. This is particularly useful when you are temporarily disconnected from the Internet or need to back up. The most popular online repository for Git is GitHub, which not only allows version control, but also offers a full community platform where other developers' projects are available and you can collaborate.

Libraries

The development of an efficient and fast

we use different **libraries** for programming.

The term "library" simply refers to a file that contains compiled object code (and data) that can later be edited (linked) by a program. Libraries allow an application to be more modular, faster to recompile and easier to update. Libraries can be classified into three types: static libraries, shared libraries and dynamic load (DL) libraries.

In the Arduino ecosystem, libraries can be installed via the **built-in interface** and can be included in code with a single **#include** statement. Each library is used to support a specific hardware or function, for example.

- **#include <Wire.h>** - Handles **I2C (Inter-Integrated Circuit)** communication.
- #include <Adafruit_GFX.h> - The Adafruit Graphics Library, which draws graphical objects (text, lines, shapes) on the display.
- #include <dht.h> - A simple library for managing DHT11/DHT22 temperature and humidity sensors.

Tools

For the project, I used various hardware tools to provide control, data collection and display. These devices work together to implement the functionality of the system.

Microcontroller: Arduino and ESP8266

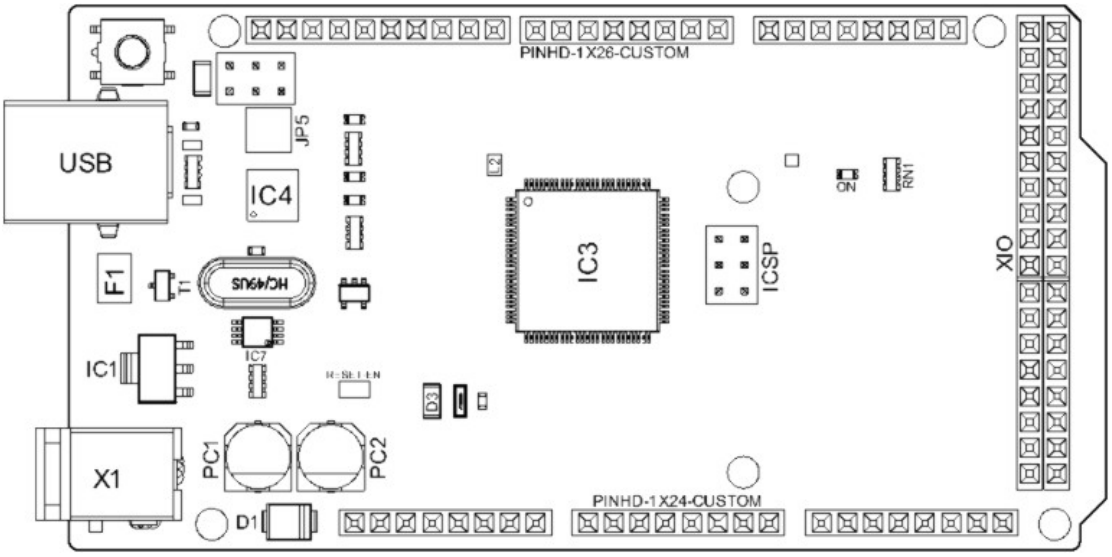
The project is based on an Arduino Mega, which is responsible for managing the sensors and displays. In addition, the ESP8266 Wi-Fi module allows the system to connect to a network and send data to a server.

Arduino Mega

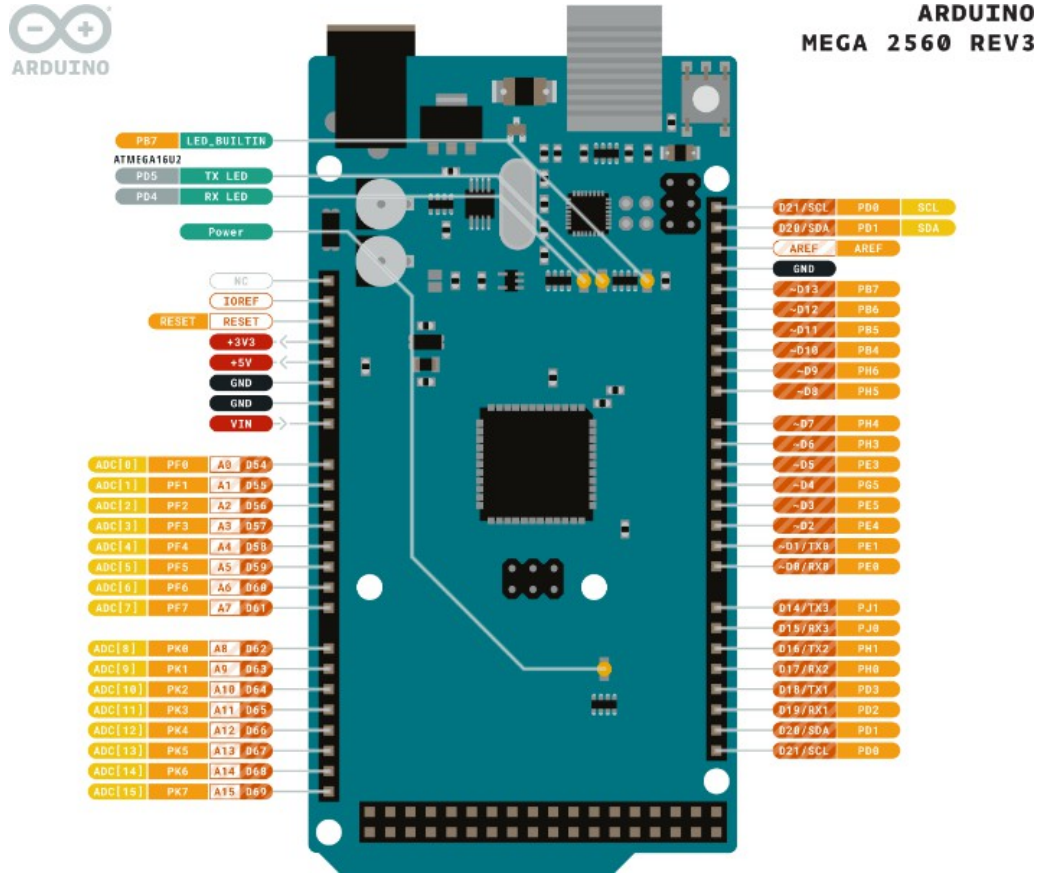
The Arduino microcontroller processes the sensor data and controls the output devices.

The Arduino Mega 2560 is an ATmega2560 microcontroller board. It has 54 digital input and output pins (15 of which can be used as PWM outputs), 16 analog inputs, 4 UART (hardware serial port), 16 MHz crystal oscillator, USB connector, power connector, ICSP header and reset button.

Structure of the arduino mega



Footprint of the arduino mega



ESP8266:

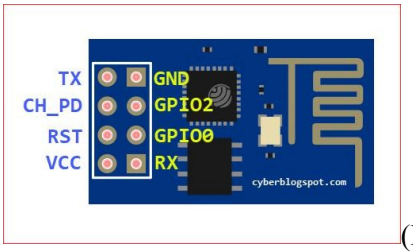
A microcontroller that can also act as a standalone Wi-Fi module, communicating with the Arduino via AT commands over a serial connection and allowing data to be transmitted to the Internet.

The ESP8266 is a very popular, low-cost Wi-Fi enabled microcontroller module developed by Espressif Systems. The module has an integrated TCP/IP protocol stack and a 32-bit RISC CPU that can operate at clock speeds of up to 80-160 MHz. Thanks to its built-in Wi-Fi functionality, it is an ideal choice for IoT (Internet of Things) projects, as it can connect to wireless networks independently or act as a Wi-Fi modem in addition to another microcontroller.

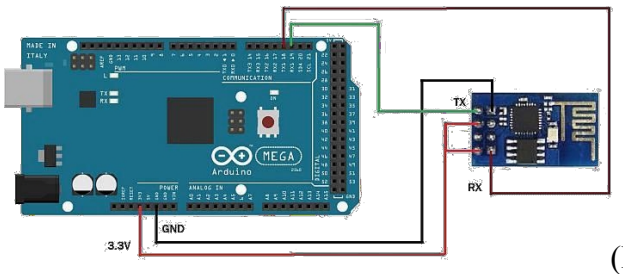
The ESP-01 has only 8 legs, two of which can be used as GPIO ports (GPIO0, GPIO2), the rest for power, serial communication (TX/RX) and control.

It can be programmed with AT commands via another microcontroller (e.g. Arduino UNO) or directly if a suitable USB to serial converter and the appropriate firmware in the Arduino IDE are available.

The figure below shows the function of the microcontroller's pins (Figure 1) and an example of how to connect it to an arduino mega (Figure 2).



(Figure 1.)



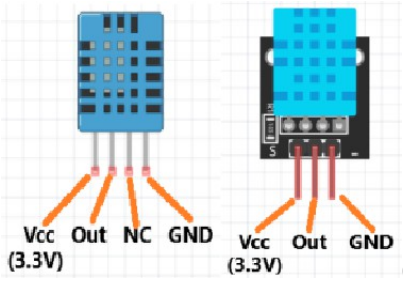
(Figure 2,)

Sensors (sensors)

To operate the system, I used various sensors that allow the measurement and processing of environmental data.

DHT11 - Temperature and humidity sensor

- It allows you to measure air temperature and humidity.
- It provides digital output that can be easily processed by the Arduino.



(Output of the sensor spikes)

Soil moisture sensor

- It measures the soil moisture content, which can help to control automatic irrigation.
- It gives an analog output, which can be received by one of the analog pins of the arduino.

(Function of the sensor spikes)

Light level detector LDR

- Photo-resistance, the amount of resistance depends on the intensity of the light incident on it.
- The output is read by one of the analog inputs of the arduino.

Display: OLED SSD1306

The 0.96-inch OLED display (SSD1306) allows you to visualize the measured data in menu-driven format.Using the Adafruit SSD1306 and GFX libraries, you can easily plot text and graphical elements (e.g. icons or diagrams).

The **SSD1306** type display using **OLED (Organic Light Emitting Diode)** technology is a low-power, high-contrast display often used in embedded systems such as Arduino or ESP8266 microcontrollers. The most common version of the display is **0.96 inches in** diameter, has a resolution of **128x64 pixels**, and contains a control chip, **SSD1306**, which is responsible for driving the OLED matrix. This controller also supports **I2C** and **SPI** communication protocols, allowing easy connectivity and fast data transfer.

Displays have the advantage of a high brightness, high contrast image, as each pixel light independently, eliminating the need for backlighting. Their small size and low power consumption make them ideal for mobile and IoT applications such as displaying sensor data or simple menus.

The following figures show the connections needed to communicate with the Arduino. The first figure (Figure 1) shows a connection established with an arduino uno using the I2C communication protocol. The second figure (Figure 2) shows the function of the display spikes (also using I2C communication protocol). The third figure (Figure 3) also shows the function of the spikes, but with a display using SCA communication.

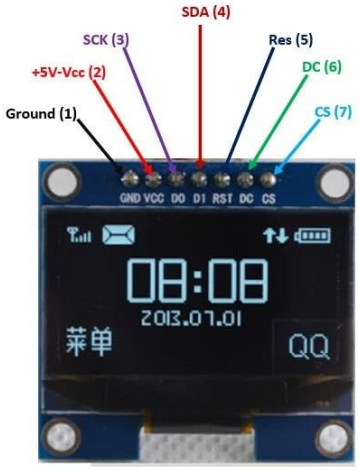


Figure 3.

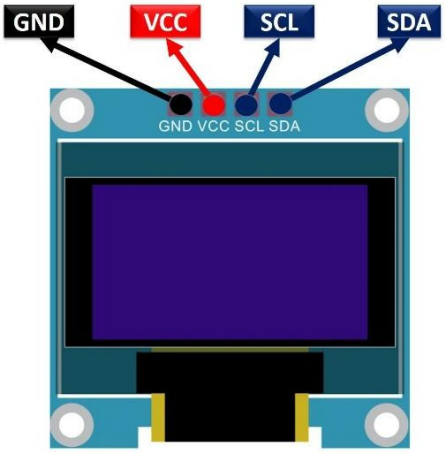


Figure 2.

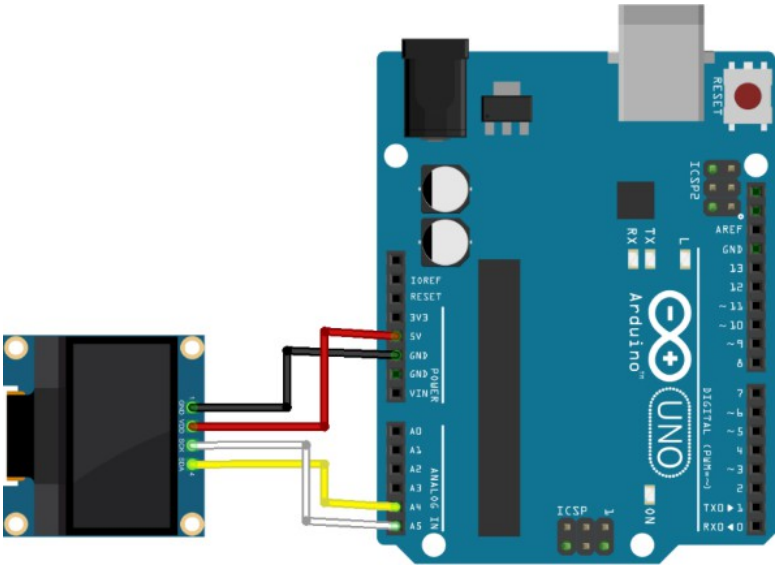
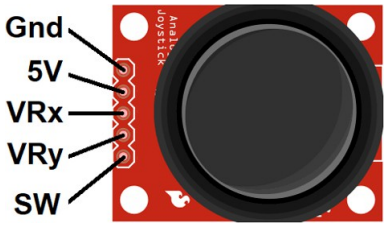
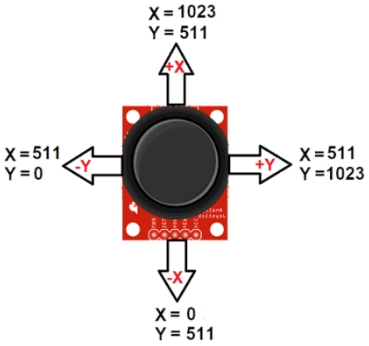


Figure 1.

Joystick module

- In this project I use a **two-axis** analogue joystick module, which allows the user to provide physical input to the system.
- This module consists of two potentiometers and a push button, so it can detect X-Y movement and click.
  The following pictures show the connectors of the module and the values returned in relation to the displacement.



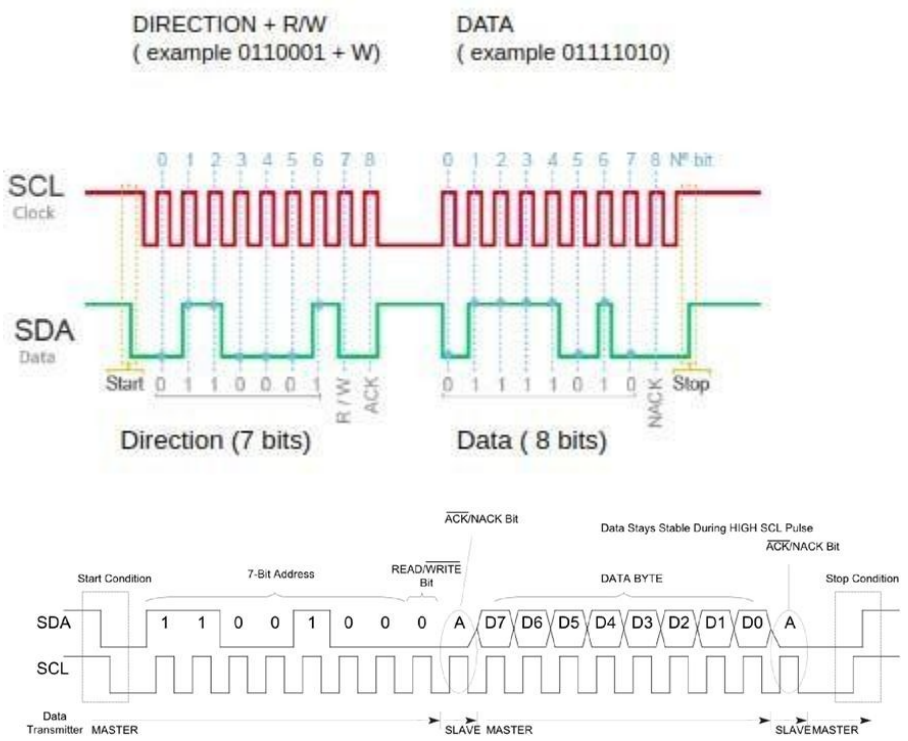


Protocols

In the project, I use different communication protocols to transfer data between devices. These protocols allow sensors, displays and the ESP8266 Wi-Fi module to communicate with the Arduino.

1. I²C (Inter-Integrated Circuit) protocol
    - **I²C** is **a serial communication protocol** mainly used to connect sensors, displays and other peripherals. It is a two-wire system consisting of a **clock line (SCL**) and a **data line (SDA).**
    - I²C is typically used for short-range, relatively low-speed communication between ICs and on-board systems
    - The I²C (Inter-Integrated Circuit) is a multi-master, multi-slave, packet-switched serial bus developed by Philips Semiconductor (now NXP Semiconductors).
    - In this project, the I2C protocol is used to communicate with the display, but as a development option it can be extended with an RTC module to monitor the time in real time.

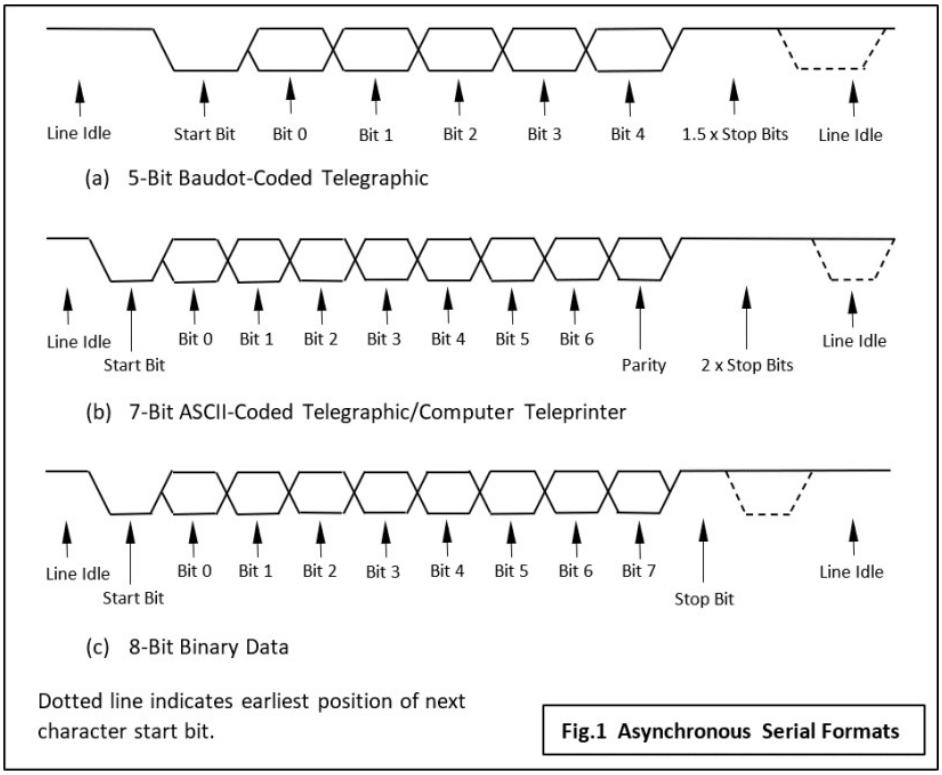
Illustration of the I2C protocol



2. UART (Universal Asynchronous Receiver-Transmitter) - Soros communication
    - A universal asynchronous transceiver (UART for short) is a hardware device that translates between serial and parallel interfaces.

- The UART protocol is one of the most commonly used communication methods between microcontrollers and external modules. It is based on two wire systems: TX-Transmitter - Transmitter, RX-Receiver - Receiver
- In my project the arduino and the esp8266 are connected in series

The UART protocol diagram



3. AT protocol
    - The protocol allows a microcontroller or computer to issue instructions to a device, such as an **ESP8266 Wi-Fi module, via serial (UART) communication**.
    - **AT-Commands** (or **Hayes AT-Commands**) is a standard **modem controller command set** originally developed by **Hayes Microcomputer Products** in the 1980s. The abbreviation **"AT"** is derived from the word "attention", since each command begins with the prefix "AT", indicating to the modem that a new command is about to follow.
    - Examples of using at commands with esp:
        - AT+CWJAP="SSID", "PASSWORD" - Connect to Wi-Fi network (ESP8266).
        - Retrieve AT+CIFSR-Ip cim.
        - AT+CIPSTART="TCP","192.168.1.50",8080-szerverhez való joining.

Used programming environment

To implement the project I used **the Arduino programming language**, which is based on C and C++, but has a simplified syntax.

I relied on **Arduino's native functions**, such as:

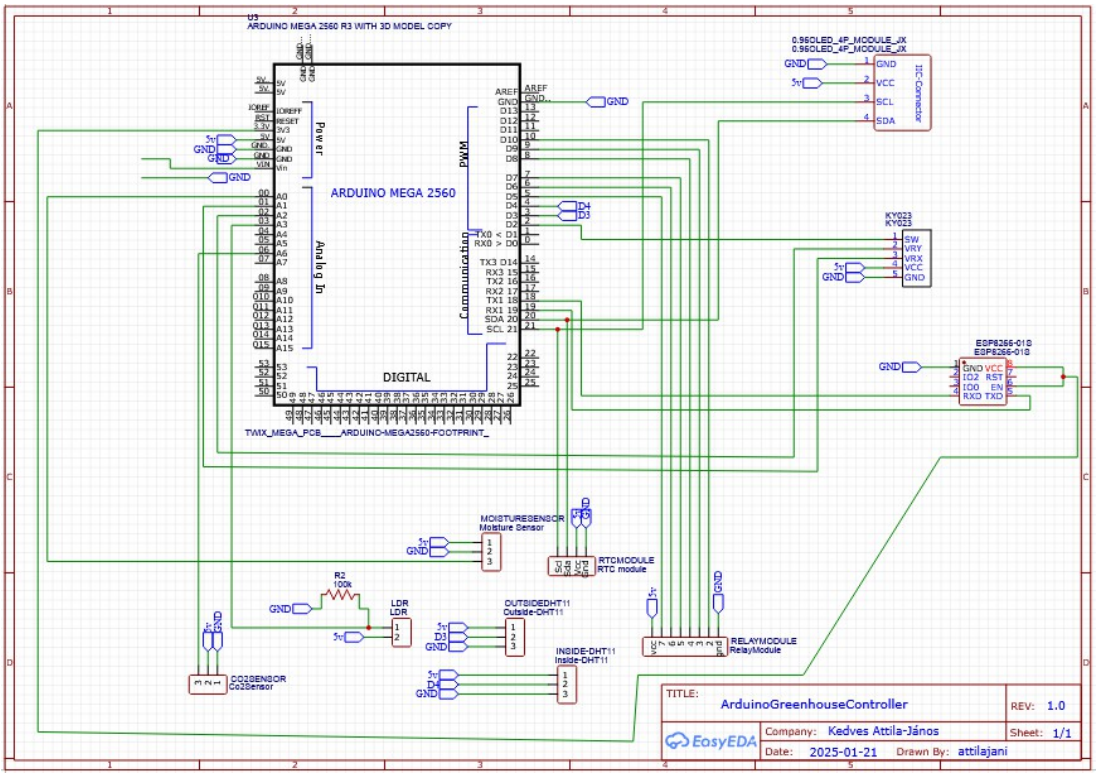
1. **pinMode(pin, mode)**
   Set a given digital leg as input (INPUT), output (OUTPUT), or input with an internal pull-up resistor (INPUT_PULLUP). Specific example:
   - pinMode(13, OUTPUT); // set D13 as output (e.g. for LED control)
   - pinMode(2, INPUT); // set D2 foot as input (e.g. for button)
   - pinMode(3, INPUT_PULLUP); // D3 input, internal pull-up resistor
2. digitalWrite(pin, value)
   Sends a signal **to** a **digital output** (HIGH or LOW). A concrete example:
   - digitalWrite(13, HIGH); // Turns on the LED connected to D13
   - digitalWrite(13, LOW); // Turns off the LED connected to D13
3. digitalRead(pin)
   Reads the status (HIGH or LOW) of a digital input.
4. **analogWrite(pin, value)**
   It generates a **PWM (Pulse Width) signal** at the specified output. The value can be between **0 and 255.** Only works on PWM-enabled legs.
   Concrete example: analogWrite(9, 128); // PWM signal on D9 with 50% fill factor
5. analogRead(pin)
   Reads the value of an **analog input** between **0 and 1023.**
   Concrete example: int Temperature = analogRead(A0); // Read the voltage of leg A0 (0-1023)
6. delay(ms)
   Stops the program running for a specified **millisecond (ms)**. Concrete example: delay(1000); // 1 second delay
7. millis()
   Returns the time **in milliseconds** since the program was started.
   Concrete example: unsigned long inditasotaElteltIdo = millis();
8. map(value, fromLow, fromHigh, toLow, toHigh)
   Scales a number one range to another. Concrete example: int sensor = analogRead(A0);
   int scaled to= map(sensor, 0, 1023, 0, 255);

System design and wiring diagrams

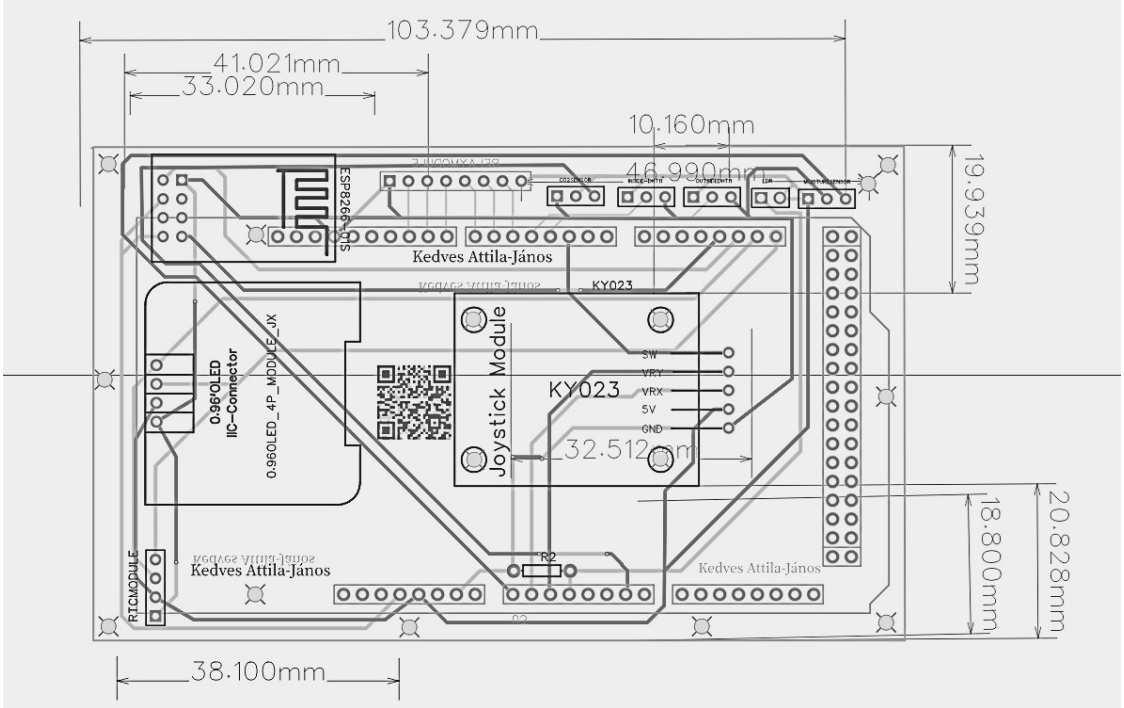
Overview

The purpose of the system is to monitor and control the environmental parameters of the greenhouse.The arduino receives the data from the sensors and controls the relle module based on the parameters. The arduino communicates with the esp8266 wifi module to send the data. The arduino controls the oled display via I2C communication. The joystick module gives user input control through two analog inputs and a digital one through a button.

Wiring Diagram



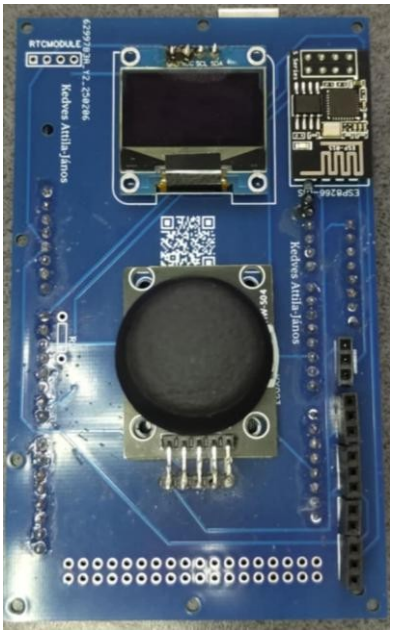
Circuit

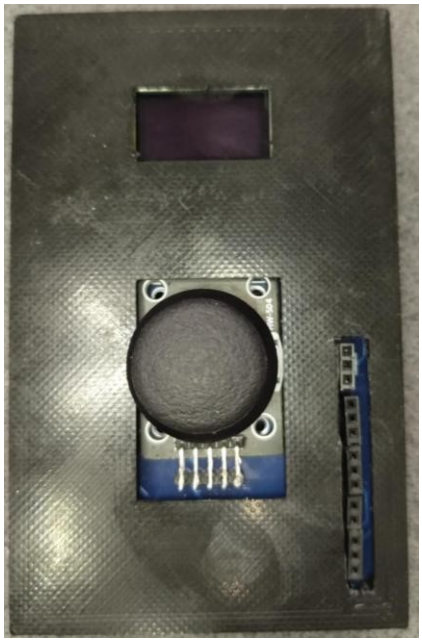


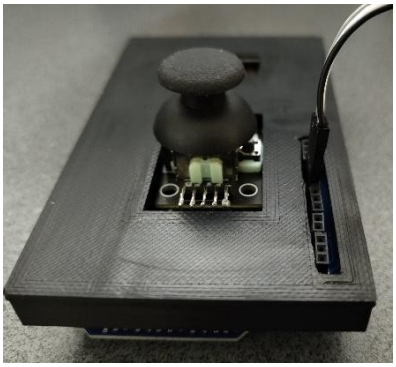
Binding table

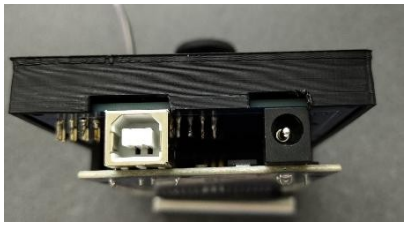
Arduino Mega 2560 Pin	Connected Device	Type of connection	Comment
3V	ESP8266	Food supply	Voltage
5V	Multiple modules (e.g. OLED, DHT)	Food supply	Voltage
GND	More modules (all)	Earthing	Common GND
D2	Push button	Digital input	Gomb
D3	DHT11	Digital input	Temperature/humidity
D4	DHT11	Digital input	Temperature/humidity
D5-D10	Relay module	Digital output	Relay control
Serial1 RX	ESP8266 TX	Soros communications	ESP TX - Mega RX
Serial1 TX	ESP8266 RX	Soros communications	ESP RX - Mega TX
A1	Joystick module X-axis	Analogue input	X Movement
A2	Joystick module Y-axis	Analogue input	Y movement
A3	Light sensor	Analogue input	Photosensitive resistance
A6	C02 sensor	Analogue input	Carbon dioxide level
SDA (A4)	OLED display, I2C module	I2C communication	Dateline
SCL (A5)	OLED display, I2C module	I2C communication	Watchline

The finished controller









Source code

```
1. //Add used libraries
```

```
2.  #include <SPI.h>
3.  #include <Wire.h>
4.  #include <Adafruit_GFX.h>
5.  #include <Adafruit_SSD1306.h>
6.  #include <dht.h>
7.  #include <SoftwareSerial.h>
8.  #include <Vector.h>
9.  #include <avr/wdt.h>
10.
```

```
11. //Specify the connected spikes of the devices and the constant values and declare the objects to be used
```

```
12.  //Temp and humidity
13.  #define dht1 3
14.  #define dht2 4
15.  // Declaration of the two DHT sensor objects
16.  dht DHT1;
17.  dht DHT2;
18.  //Moisture sensor
19.  #define MOISTURESENSOR A0
20.  #define MINMOISTURE 500
21.  #define MAXMOISTURE 180
22.  //Output relays
23.  #define FANPIN 7
24.  #define IRRIGATIONPIN 8
25.  #define SPRAYPIN 9
26.  #define HEATERPIN 10
27.  #define LIGHTPIN 6
28.  // Declaration of oled display with i2c communication
29.  #define SCREEN_WIDTH 128
30.  #define SCREEN_HEIGHT 64
31.  #define OLED_RESET        -1
32.  #define SSD1306_I2C_ADDRESS 0x3C
33.  Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
34.
35.  //joystick
36.  #define JOYSTICK_X A1
37.  #define JOYSTICK_Y A2
38.  #define JOYSTICK_BUTTON 2
39.  #define JOYSTICK_MAXTRESHOLD 900
40.  #define JOYSTICK_MINTRESHOLD 100
41.
42.  #define MAX_MENU_COUNT 7 // the menu items are indexed from 0, so this is 1 to less
43.  #define MIN_CHANGE_SPEED 0
44.  #define MAX_CHANGE_SPEED 100
45.  #define SERIALPRINTINTERVAL 5000
46.  #define DEBUG 0
47.  #define NETWORKTYPE 0 //0-STA, 1 AP
48.  #define SERIALNUMBER 123456
49.  #define WIFINAME "DIGI_57c120"
50.  #define PASSWORD "195da83e"
51.  #define AITHINKER "AI-THINKER_394EDC"
```

```
52. // Declaring input and output variables:
```

```
53.     // Input variables (sensor data):
54.       byte insideTemperature, outsideTemperature, insideHumidity, outsideHumidity, soilMoistureLevel, lightIntensity;
55.     // Output variables (controlled devices):
56.       enum DeviceState
57.       {
58.         OFF,     // 0
59.         ON,      // 1
60.         AUTO     // 2
```

```
    };
    DeviceState fanState, irrigationState, sprayState, heaterState, lightState;

//Declare internal variables:
  //wifi
  String ssid= String(WIFINAME);          // Name of the wifi AI-THINKER_394EDC
  String password= String(PASSWORD);      // Password for the wifi network
  String ipAddress= "";
  int joystickAxisX, joystickAxisY; //A axis-related assignments to the joystick
  bool joystickButtonState;
  byte currentMenuNumber;
  byte currentSubmenuNumber;
  byte maxMenuCount;
  bool isSubMenu;
  long long lastTimeMenuRefresh;
  // Button's time elapsed test
  const byte menuChangeInterval= 500; // menu/submenu valtozas
  long long lastTimeChange= 0;// Check the time elapsed between Left/Right position
  long long lastTimeSerial= 0;
  const char deviceStateStrings[3][4]= {"OFF", "ON", "AUTO"}; // for the quickest way to select the quickest way to select the device
  Declare //Min and max target variables:
  byte maxTemp, minTemp, maxHumidity, minHumidity, maxMoisture, minMoisture, minLightLevel;
  const char* languages[3]= {"Eng", "Hun", "Ro"};
  byte languageSet;
  bool enableSerial= DEBUG;
  bool networkType= NETWORKTYPE;
  byte changeSpeed= 20;
  byte connectionId;
//Save main menu values in flash memory
  const char menu1[] PROGMEM= "Device Owerview";
  const char menu2[] PROGMEM= "Sensor Overview";
  const char menu3[] PROGMEM= "Temperature Control";
  const char menu4[] PROGMEM= "Humidity Control";
  const char menu5[] PROGMEM= "Irrigation Control";
  const char menu6[] PROGMEM= "Light Control";
  const char menu7[] PROGMEM= "System Settings";
  const char menu8[] PROGMEM= "Network Settings";
  const char* const messages[] = {menu1,menu2,menu3,menu4,menu5,menu6,menu7,menu8};
```

```
  //Guidelines
```

```
void readSensor()//Read values measured by sensors
{
  int insideSensor= DHT1.read11(dht1);
  insideTemperature= DHT1.temperature;
  insideHumidity= DHT1.humidity;
  int outsideSensor= DHT2.read11(dht2);
  outsideTemperature= DHT2.temperature;
  outsideHumidity= DHT2.humidity;
  soilMoistureLevel= map(analogRead(MOISTURESENSOR), MINMOISTURE, MAXMOISTURE, 0, 100);
}

void restartArduino() // restart the arduino
{
  wdt_enable(WDTO_15MS); // Enable watchdog for 15ms
  while (1);                    // Wait for restart/Freeze the program
}
void espSetup() // initialize the esp
{
  sendData("AT+RST\r\n",2000,enableSerial); // reset module
  if(networkType)
  {
    ssid= String(AITHINKER);
    password= "-";
    sendData("AT+CWMODE=2\r\n",1000,enableSerial); // configure as access point
```

```
124.    }
125.    else
126.    {
127.       ssid= String(WIFINAME);
128.       password= String(PASSWORD);
129.    delay(1000); // a little extra waiting
130.    //configure as station mode, connect to wifi
131.    sendData("AT+CWMODE=1\r\n",    1000,    enableSerial);
132.    // Connect to wifi (ssid+password)
133.    String connectCommand= "AT+CWJAP=\""+ ssid+ "\",\""+ password+ "\"\r\n";
134.    sendData(connectCommand, 8000, enableSerial);
135.    }
136.    sendData("AT+CIPMUX=1\r\n",1000,enableSerial); // configure for multiple
connections
137.    sendData("AT+CIPSERVER=1,80\r\n",1000,enableSerial); // turn on server on port 80
138. }
139. void getIp() // Get the ip address
140. {
141.    delay(1000);
142.    // Get IP address
143.    ipAddress= sendData("AT+CIFSR\r\n", 1000, enableSerial);
144.    byte first= ipAddress.indexOf('"');
145.    byte second= ipAddress.indexOf('"', first+ 1);
146.    if (first != -1 && second != -1)
147.    {
148.    ipAddress= ipAddress.substring(first+ 1, second);
149.    } else
150.    {
151.       Serial.println("ip error");
152.    }
153. }
154. String booleanStateOnOff(bool state) // return bool as string for ejection
155. {
156.    if(state)
157.       return "On";
158.    else
159.       return "Off";
160. }
161. String sendData(String command, const int timeout, boolean debug) //Commands
send to the esp and receive its reply
162. {
163.       String response= "";
164.       Serial1.print(command);
165.       long int time= millis();
166.       while( (time+ timeout)> millis())
167.       {
168.          while(Serial1.available())
169.             {
170.                char c= Serial1.read(); // read the next character.
171.                response+= c;
172.             }
173.       }
174.       if(debug)
175.             {
176.             Serial.print(response); //displays the esp response messages in arduino
Serial monitor
177.             }
178.       return response;
179. }
180. void espsend(String date) //Set up and execute the espsend command
181. {
182.    String cipSend= " AT+CIPSEND=";
183.    cipSend+= connectionId;
184.    cipSend+= ",";
185.    cipSend+= date.length();
186.    cipSend+= "\r\n";
187.    sendData(cipSend,1000,enableSerial);
188.    sendData(date,1000,enableSerial);
```

```
189. }
190. void showMessageOnScreen(String message) // Show text in the middle of the display
191. {
192.    display.clearDisplay();
193.    display.setTextSize(1);             // Set text size
194.    display.setTextColor(WHITE);
195.    display.setCursor(0,    10);      // Set text position
196.    display.println(message);
197.    display.display();
198. }
199. void transmitDateOnEsp(String data)// Handle communication with esp
200. {
201.    if(Serial1.available())
202.    {
203.      if(Serial1.find("+IPD,")) //watch for request arrival
204.      {
205.       showMessageOnScreen("Network    communication\nin    progress...");
206.       delay(300);
207.       connectionId= Serial1.read()-48;
208.        String webpage= data;
209.        webpage+= connectionId;
210.        espsend(webpage);
211.       }
212.       String closeCommand= "AT+CIPCLOSE="; ////////////////close the socket
connection////esp command
213.       closeCommand+= connectionId; // append connection id
214.       closeCommand+= "\r\n";
215.       sendData(closeCommand,3000,enableSerial);
216.    }
217. }
218. String constructDateString() //A Construct a string sent to the network based on
the controller data
219. {
220.    String dataString= "InTemp="+ String(insideTemperature)+ "&OutTemp="+
String(outsideTemperature)+ "&InHum="+ String(insideHumidity)+ "&OutHum="+
String(outsideHumidity)+ "&Moi="+ String(soilMoistureLevel)+ "&Light="+
String(lightIntensity)+ "&maxTemp="+ String(maxTemp)+ "&minTemp="+ String(minTemp)+
"&maxHum="+ String(maxHumidity)+ "&minHum="+ String(minHumidity)+ "&maxMoi="+
String(maxMoisture) ;
221.    dataString+= "&minMoisture="+ String(minMoisture)+ "&minLightLevel="+
String(minLightLevel)+"&fanState="+ String(fanState)+ "&irrigationState="+
String(irrigationState)+ "&sprayState="+ String(sprayState)+ "&heaterState="+
String(heaterState)   +    "&lightState="   +    String(lightState);
222.    String html= "<!DOCTYPE html><html><html><head><meta http-equiv=\\\"refresh\\\\\"
content=\\\\"5\\\\"><title>Greenhouse"+ String(SERIALNUMBER)+ "</title></head><body>"+
dataString+ "</body></html>";
223.
224.    return html;
225. }
226. void outRelays() // Turn on and off the relay modules based on their
stateminLightLevel
227. {
228.    if(heaterState== 2)
229.    {
230.      if(insideTemperature< minTemp)
231.        digitalWrite(HEATERPIN, HIGH);
232.      if(insideTemperature >(minTemp+ maxTemp)/2)
233.        digitalWrite(HEATERPIN, LOW);
234.    }
235.      if (heaterState== 1)
236.        digitalWrite(HEATERPIN, HIGH);
237.      if (heaterState== 0)
238.        digitalWrite(HEATERPIN, LOW);
239.    if(fanState== 2)
240.    {
241.      if(insideTemperature   >    maxTemp)
242.        digitalWrite(FANPIN, HIGH);
243.      if(insideTemperature< (minTemp+ maxTemp)/2)
244.        digitalWrite(FANPIN, LOW);
245.    }
```

```
246.    if(fanState== 1)
247.        digitalWrite(FANPIN, HIGH);
248.    if(fanState== 0)
249.        digitalWrite(FANPIN, LOW);
250.    if(irrigationState== 2)
251.    {
252.      if(soilMoistureLevel  <   minMoisture)
253.        digitalWrite(IRRIGATIONPIN, HIGH);
254.      if(soilMoistureLevel  >   maxMoisture)
255.        digitalWrite(IRRIGATIONPIN, LOW);
256.    }
257.    if(irrigationState== 1)
258.        digitalWrite(IRRIGATIONPIN, HIGH);
259.    if(irrigationState== 0)
260.        digitalWrite(IRRIGATIONPIN, LOW);
261.    if(lightState== 2)
262.    {
263.      if(lightIntensity  <   minLightLevel)
264.        digitalWrite(LIGHTPIN, HIGH);
265.      if(lightIntensity>= minLightLevel)
266.        digitalWrite(LIGHTPIN,LOW);
267.    }
268.    if(lightState== 1)
269.        digitalWrite(LIGHTPIN, HIGH);
270.    if(lightState== 0)
271.        digitalWrite(LIGHTPIN, LOW);
272.    switch (sprayState)
273.    {
274.      case 0:digitalWrite(SPRAYPIN, LOW);break;
275.      case 1:digitalWrite(SPRAYPIN, HIGH);break;
276.      case 2:
277.      if(insideHumidity< minHumidity)
278.        digitalWrite(SPRAYPIN, HIGH);
279.      if(insideHumidity> (minHumidity+ maxHumidity)/2)
280.        digitalWrite(SPRAYPIN, LOW);
281.        break;
282.    }
283.
284. }
285. short languageChooser (short currentLanguage)// Change the language
chooser default
286. {
287.      if (joystickAxisY> JOYSTICK_MAXTRESHOLD)
288.    {
289.      currentLanguage ++;
290.      if(currentLanguage> 2)
291.        currentLanguage= 0;
292.    }
293.    else
294.    {
295.      if (joystickAxisY< JOYSTICK_MINTRESHOLD)
296.      {
297.        currentLanguage --;
298.        if(currentLanguage< 0)
299.          currentLanguage= 2;
300.
301.      }
302.
303.    }
304.    delay(changeSpeed+ 100);
305.    return currentLanguage;
306. }
307. void changeSubmenuBoolean(bool &booleanToChange)// Change the bool
menu variable, based on joystickAxisY
308. {
309.    if (joystickAxisY> JOYSTICK_MAXTRESHOLD|| joystickAxisY<
JOYSTICK_MINTRESHOLD)
310.    {
311.      booleanToChange= !booleanToChange;
312.    }
```

```
313.     delay(changeSpeed+ 30);
314. }
315.  short changeSubmenuVariable(short numberToChange) // Change the number values in the submenus
316.  {
317.
318.     if (joystickAxisY> JOYSTICK_MAXTRESHOLD)
319.     {
320.        numberToChange++;
321.     }
322.     else
323.     {
324.        if (joystickAxisY< JOYSTICK_MINTRESHOLD)
325.        {
326.           numberToChange--;
327.        }
328.     }
329.     delay(changeSpeed+ 30);
330.     return numberToChange;
331.
332.  }
333.  DeviceState changeDeviceState(DeviceState deviceState)// Change the device state
334.  {
335.     short counter= deviceState;
336.
337.     if (joystickAxisY> JOYSTICK_MAXTRESHOLD)
338.     {
339.        counter ++;
340.        if(counter> 2)
341.           counter= 0;
342.     }
343.     else
344.     {
345.        if (joystickAxisY< JOYSTICK_MINTRESHOLD)
346.        {
347.           counter --;
348.           if(counter< 0)
349.              counter= 2;
350.        }
351.        else return static_cast< DeviceState>(counter);
352.     }
353.     delay(changeSpeed+ 100);
354.     return static_cast< DeviceState>(counter);
355. }
356.  void changeMenuNumber(bool isSubMenu, byte maxNumber)// Increase/decrease the value of the menu and submenu
357.  {
358.     if(millis()-lastTimeMenuRefresh> changeSpeed+ 50)//Test the time between changes for correct operation
359.     {
360.           lastTimeMenuRefresh= millis();
361.        if(isSubMenu) // if(isSubMenu) // whether to change the value of the main menu or the submenu
362.        {
363.           if (joystickAxisX> JOYSTICK_MAXTRESHOLD)
364.           {
365.              if(maxNumber> currentSubmenuNumber)
366.                 currentSubmenuNumber++;
367.              else
368.                 currentSubmenuNumber= 0;
369.
370.           }
371.        else
372.        {
373.           if (joystickAxisX< JOYSTICK_MINTRESHOLD)
374.           {
375.              if(currentSubmenuNumber> 0)
376.                 currentSubmenuNumber--;
377.              else
```

```
378.              currentSubmenuNumber= maxNumber;
379.
380.           }
381.        }
382.        }
383.        else
384.        {
385.           if (joystickAxisX> JOYSTICK_MAXTRESHOLD)
386.           {
387.              if(maxNumber> currentMenuNumber)
388.                 currentMenuNumber++;
389.              else
390.                 currentMenuNumber= 0;
391.
392.           }
393.        else
394.        {
395.           if (joystickAxisX< JOYSTICK_MINTRESHOLD)
396.           {
397.              if(currentMenuNumber> 0)
398.                 currentMenuNumber--;
399.              else
400.              currentMenuNumber= maxNumber;
401.
402.           }
403.        }
404.        }
405.     }
406. }
407.  void readJoystickValues()// Read the state of the joystick module
408.  {
409.     joystickAxisX= 1024-analogRead(JOYSTICK_X);
410.     joystickAxisY  = analogRead(JOYSTICK_Y);
411.     joystickButtonState= !digitalRead(JOYSTICK_BUTTON);
412. }
413. void serialMonitorPrint() // Print data serially for reading
414.     {
415.        if(millis()-lastTimeSerial   >    SERIALPRINTINTERVAL)
416.        {
417.           lastTimeSerial= millis();
418.           Serial.println();
419.           Serial.println(millis());
420.           Serial.print(F("Output variables:")); // F() is required because SRAM
is corrupted, so it loads the Flash memory
421.           Serial.print(F(" fanState: "));
422.           Serial.print(fanState);
423.           Serial.print(F(" irrigationState: "));
424.           Serial.print(irrigationState);
425.           Serial.print(F(" sprayState: "));
426.           Serial.print(sprayState);
427.           Serial.print(F(" fanState: "));
428.           Serial.print(fanState);
429.           Serial.print(F(" lightState: "));
430.           Serial.println(lightState);
431.           Serial.println();
432.           Serial.print(F("Sensor values: "));
433.           Serial.print(F(" insideTemperature: "));
434.           Serial.print(insideTemperature);
435.           Serial.print(F(" outsideTemperature: "));
436.           Serial.print(outsideTemperature);
437.           Serial.print(F(" insideHumidity: "));
438.           Serial.print(insideHumidity);
439.           Serial.print(F(" outsideHumidity: "));
440.           Serial.print(outsideHumidity);
441.           Serial.print(F(" soilMoistureLevel: "));
442.           Serial.print(soilMoistureLevel);
443.           Serial.print(F(" lightIntensity: "));
444.           Serial.println(lightIntensity);
445.           Serial.println();
446.           Serial.print(F("Joystick values: "));
```

```
447.          Serial.print(F(" joystickAxisX: "));
448.          Serial.print(joystickAxisX);
449.          Serial.print(F(" joystickAxisY: "));
450.          Serial.print(joystickAxisY);
451.          Serial.print(F(" joystickButtonState: "));
452.          Serial.println(joystickButtonState);
453.          Serial.println();
454.          Serial.print(F("Menu value: "));
455.          Serial.print(currentMenuNumber);
456.          Serial.print(F("Submenu value: "));
457.          Serial.print(currentSubmenuNumber);
458.          Serial.print(F("isSubmenu: "));
459.          Serial.print(isSubMenu);
460.          Serial.println("\n----------------------------------");
461.
462.       }
463.
464.    }
465. void killAllDevices()//Set all devices to OFF, and disable all relays
466. {
467.    //set the device state to off
468.    fanState= OFF;
469.    irrigationState= OFF;
470.    sprayState= OFF;
471.    heaterState= OFF;
472.    lightState= OFF;
473.    //give the command to the rells
474.    digitalWrite(FANPIN, LOW);
475.    digitalWrite(IRRIGATIONPIN, LOW);
476.    digitalWrite(SPRAYPIN, LOW);
477.    digitalWrite(HEATERPIN, LOW);
478.    digitalWrite(LIGHTPIN, LOW);
479. }
480. void displayInitialize()//Display the home screen
481. {
482.    if(!display.begin(SSD1306_SWITCHCAPVCC,    SSD1306_I2C_ADDRESS)) {
483.       Serial.println(F("SSD1306 allocation failed"));
484.       for(;;); // Failed to establish connection, infinite loop
485.    }
486.    display.clearDisplay();// Clear the buffer
487.
488.    display.setTextSize(1);             // Set text size
489.    display.setTextColor(WHITE);
490.    display.setCursor(0,    0);      // Set text position
491.    display.println(F("Dear Attila Janos"));
492.    display.println("Github:");
493.    display.println(F("https://github.com/KedvesAttilaJanos"));
494.    display.display();
495.    delay(5000);
496.    showLoadingScreen();
497. }
498. void loaderSwitch(byte loadNum) // Determine the execution order of setup tasks
according to the loading screen
499. {
500.    switch(loadNum)
501.    {
502.       case 30: espSetup(); Serial.println(F("Esp setuped")); break;
503.       case 50: getIp(); Serial.println(F("Ip get")); break;
504.       case 60: killAllDevices(); Serial.println(F("Devices off")); break;
505.       case 70: zeroAllSensor(); Serial.println(F("Sensors Zero")); break;
506.       case 80: zeroAllInternalVariable(); Serial.println(F("Internal variables
zero")); break;
507.       case 90: readSensor(); Serial.println(F("First read")); break;
508.    }
509. }
510. bool joystickAxisYMoved(String direction) //overload, you can specify as parameter
which direction to look at the joystick movement
511. {
512.    if(direction== "right")
513.       return    joystickAxisY   >    JOYSTICK_MAXTRESHOLD;
```

```
514.    if(direction== "left")
515.       return   joystickAxisY   <   JOYSTICK_MINTRESHOLD;
516. }
517. bool joystickAxisYMoved() // Monitor left/right movement
518. {
519.    return joystickAxisY< JOYSTICK_MINTRESHOLD|| joystickAxisY>
JOYSTICK_MAXTRESHOLD;
520. }
521. void showLoadingScreen()//Show Loading Screen
522. {
523.    display.clearDisplay();
524.    display.setTextSize(1);            // Set text size
525.    display.setTextColor(WHITE);
526.    display.setCursor(10,   10);     // Set text position
527.    display.println(F("Loading..."));
528.    display.display();
529.
530.    // Animation loading bar
531.    for (int i= 0; i<= 100; i+= 2) {
532.       display.fillRect(10, 30, i, 10, WHITE); // Draw a bar
533.       loaderSwitch(i);
534.       display.display();
535.       delay(2); // lane refresh rate
536.    }
537.
538.    // Clear or next screen
539.    delay(500);
540.    display.clearDisplay();
541.    display.display();
542. }
543. void zeroAllSensor()//Reset all sensor values to zero
544. {
545.    insideTemperature= 0;
546.    outsideTemperature= 0;
547.    insideHumidity= 0;
548.    outsideHumidity= 0;
549.    soilMoistureLevel= 0;
550.    lightIntensity= 0;
551. }
552. void zeroAllInternalVariable()// Reset all internal variables
553. {
554.    joystickAxisX= 0;
555.    joystickAxisY= 0;
556.    joystickButtonState= 0;
557.    isSubMenu= false;
558.    currentMenuNumber= 0;
559.    currentSubmenuNumber= -1;
560.    lastTimeMenuRefresh= 0;
561.    maxMenuCount= MAX_MENU_COUNT;
562.    maxTemp= 0;
563.    minTemp= 0;
564.    maxHumidity= 0;
565.    minHumidity= 0;
566.    maxMoisture= 0;
567.    minMoisture= 0;
568.    minLightLevel= 0;
569.    languageSet= 0;
570.    enableSerial= DEBUG;
571. }
572. void printCentered(const String &text, int y)// Print centered addresses
573. {
574.    int textWidth= text.length() * 6; // 6 pixel wide characters
575.    int x= (SCREEN_WIDTH - textWidth) / 2;
576.    display.setCursor(x, y);
577.    display.println(text);
578. }
579. void changeSelectedMenuVariable()// Change the selected values
580. {
581.    if(isSubMenu)
582.    {
```

```
583.        switch (currentMenuNumber)
584.        {
585.          case 0:
586.            if (currentSubmenuNumber== 0) fanState= changeDeviceState(fanState);
587.            if (currentSubmenuNumber== 1) irrigationState=
changeDeviceState(irrigationState);
588.            if (currentSubmenuNumber== 2) sprayState= changeDeviceState(sprayState);
589.            if (currentSubmenuNumber== 3) heaterState=
changeDeviceState(heaterState);
590.            if (currentSubmenuNumber== 4) lightState= changeDeviceState(lightState);
591.          break;
592.        }
593.    }
594. }
595.  void mainMenuSystem()// Display menus, with decisions to display the menu item
with background
596.  {
597.    display.setTextSize(1);
598.    display.clearDisplay();
599.    short devices[5] {fanState, irrigationState, sprayState, heaterState,
lightState};
600.    if(!isSubMenu) //Check if you are in submenu or main menu
601.    {
602.      display.setTextColor(WHITE);
603.      display.setCursor(0,0);
604.      char buffer[20]; // This is where we load from PROGMEM
605.      for (size_t i= 0; i< 8; i++)
606.      {
607.        strcpy_P(buffer, (char*)pgm_read_word(&(messages[i])));
608.        if (currentMenuNumber== i)
609.          display.setTextColor(BLACK,WHITE);
610.        else
611.          display.setTextColor(WHITE);
612.        display.println(buffer);
613.      }
614.        display.display();
615.
616.    }
617.    else
618.    {
619.      switch(currentMenuNumber)
620.      {
621.        case 0:
622.          maxMenuCount= 4;
623.          switch(currentSubmenuNumber)
624.          {
625.          case 0:
626.          printCentered(F("Device Overview"),0);
627.          display.setTextColor(BLACK,WHITE);
628.          display.print(F("Fan State          "));
629.          display.println(deviceStateStrings[fanState]);
630.          display.setTextColor(WHITE);
631.          display.print(F("Irrigation State "));
632.          display.println(deviceStateStrings[irrigationState]);
633.          display.print(F("Spray State        "));
634.          display.println(deviceStateStrings[sprayState]);
635.          display.print(F("Heater State       "));
636.          display.println(deviceStateStrings[heaterState]);
637.          display.print(F("Light State        "));
638.          display.println(deviceStateStrings[lightState]);
639.          display.display();
640.          fanState= changeDeviceState(fanState);
641.          break;
642.          case 1:
643.          printCentered(F("Device Overview"),0);
644.          display.print(F("Fan State          "));
645.          display.println(deviceStateStrings[fanState]);
646.          display.setTextColor(BLACK,WHITE);
647.          display.print(F("Irrigation State "));
648.          display.println(deviceStateStrings[irrigationState]);
```

```
649.            display.setTextColor(WHITE);
650.            display.print(F("Spray State          "));
651.            display.println(deviceStateStrings[sprayState]);
652.            display.print(F("Heater State         "));
653.            display.println(deviceStateStrings[heaterState]);
654.            display.print(F("Light State          "));
655.            display.println(deviceStateStrings[lightState]);
656.            display.display();
657.            irrigationState= changeDeviceState(irrigationState);
658.            break;
659.            case 2:
660.            printCentered(F("Device Overview"),0);
661.            display.print(F("Fan State            "));
662.            display.println(deviceStateStrings[fanState]);
663.            display.print(F("Irrigation State "));
664.            display.println(deviceStateStrings[irrigationState]);
665.            display.setTextColor(BLACK,WHITE);
666.            display.print(F("Spray State          "));
667.            display.println(deviceStateStrings[sprayState]);
668.            display.setTextColor(WHITE);
669.            display.print(F("Heater State         "));
670.            display.println(deviceStateStrings[heaterState]);
671.            display.print(F("Light State          "));
672.            display.println(deviceStateStrings[lightState]);
673.            display.display();
674.            sprayState= changeDeviceState(sprayState);
675.            break;
676.            case 3:
677.            printCentered(F("Device Overview"),0);
678.            display.print(F("Fan State            "));
679.            display.println(deviceStateStrings[fanState]);
680.            display.print(F("Irrigation State "));
681.            display.println(deviceStateStrings[irrigationState]);
682.            display.print(F("Spray State          "));
683.            display.println(deviceStateStrings[sprayState]);
684.            display.setTextColor(BLACK,WHITE);
685.            display.print(F("Heater State         "));
686.            display.println(deviceStateStrings[heaterState]);
687.            display.setTextColor(WHITE);
688.            display.print(F("Light State          "));
689.            display.println(deviceStateStrings[lightState]);
690.            display.display();
691.            heaterState= changeDeviceState(heaterState);
692.            break;
693.            case 4:
694.            printCentered(F("Device Overview"),0);
695.            display.print(F("Fan State            "));
696.            display.println(deviceStateStrings[fanState]);
697.            display.print(F("Irrigation State "));
698.            display.println(deviceStateStrings[irrigationState]);
699.            display.print(F("Spray State          "));
700.            display.println(deviceStateStrings[sprayState]);
701.            display.print(F("Heater State         "));
702.            display.println(deviceStateStrings[heaterState]);
703.            display.setTextColor(BLACK,WHITE);
704.            display.print(F("Light State          "));
705.            display.println(deviceStateStrings[lightState]);
706.            display.setTextColor(WHITE);
707.            display.display();
708.            lightState= changeDeviceState(lightState);
709.            break;
710.
711.            }
712.          break;
713.          case 1:
714.            printCentered(F("Sensor Overview "),0);
715.            display.print(F("Inside Temp.      "));
716.            display.println(insideTemperature);
717.            display.print(F("Outside Temp.     "));
718.            display.println(outsideTemperature);
```

```
719.            display.print(F("Inside Humidity "));
720.            display.println(insideHumidity);
721.            display.print(F("Outside Humidity"));
722.            display.println(outsideHumidity);
723.            display.print(F("Soil   Moisture  "));
724.            display.println(soilMoistureLevel);
725.            display.print(F("Light Level       "));
726.            display.println(lightIntensity);
727.            display.display();
728.          break;
729.          case 2:
730.            maxMenuCount= 3;
731.            switch(currentSubmenuNumber)
732.            {
733.              case 0:
734.                printCentered(F("Temp Controll"),0);
735.                display.setTextColor(BLACK,WHITE);
736.                display.print(F("Max Temp          "));
737.                display.println(maxTemp);
738.                display.setTextColor(WHITE);
739.                display.print(F("Min Temp          "));
740.                display.println(minTemp);
741.                display.print(F("Fan State         "));
742.                display.println(deviceStateStrings[fanState]);
743.                display.print(F("Heater State      "));
744.                display.println(deviceStateStrings[heaterState]);
745.                display.print(F("Inside Temp       "));
746.                display.println(insideTemperature);
747.                display.print(F("Outside Temp      "));
748.                display.println(outsideTemperature);
749.                display.display();
750.                maxTemp   =    changeSubmenuVariable(maxTemp);
751.              break;
752.              case 1:
753.                printCentered(F("Temp Controll"),0);
754.                display.print(F("Max Temp          "));
755.                display.println(maxTemp);
756.                display.setTextColor(BLACK,WHITE);
757.                display.print(F("Min Temp          "));
758.                display.println(minTemp);
759.                display.setTextColor(WHITE);
760.                display.print(F("Fan State         "));
761.                display.println(deviceStateStrings[fanState]);
762.                display.print(F("Heater State      "));
763.                display.println(deviceStateStrings[heaterState]);
764.                display.print(F("Inside Temp       "));
765.                display.println(insideTemperature);
766.                display.print(F("Outside Temp      "));
767.                display.println(outsideTemperature);
768.                display.display();
769.                minTemp   =    changeSubmenuVariable(minTemp);
770.              break;
771.              case 2:
772.                printCentered(F("Temp Controll"),0);
773.                display.print(F("Max Temp          "));
774.                display.println(maxTemp);
775.                display.print(F("Min Temp          "));
776.                display.println(minTemp);
777.                display.setTextColor(BLACK,WHITE);
778.                display.print(F("Fan State         "));
779.                display.println(deviceStateStrings[fanState]);
780.                display.setTextColor(WHITE);
781.                display.print(F("Heater State      "));
782.                display.println(deviceStateStrings[heaterState]);
783.                display.print(F("Inside Temp       "));
784.                display.println(insideTemperature);
785.                display.print(F("Outside Temp      "));
786.                display.println(outsideTemperature);
787.                display.display();
788.                fanState= changeDeviceState(fanState);
```

```
789.              break;
790.              case 3:
791.                printCentered(F("Temp Controll"),0);
792.                display.print(F("Max Temp          "));
793.                display.println(maxTemp);
794.                display.print(F("Min Temp          "));
795.                display.println(minTemp);
796.                display.print(F("Fan State         "));
797.                display.println(deviceStateStrings[fanState]);
798.                display.setTextColor(BLACK,WHITE);
799.                display.print(F("Heater State      "));
800.                display.println(deviceStateStrings[heaterState]);
801.                display.setTextColor(WHITE);
802.                display.print(F("Inside Temp       "));
803.                display.println(insideTemperature);
804.                display.print(F("Outside Temp      "));
805.                display.println(outsideTemperature);
806.                display.display();
807.                heaterState= changeDeviceState(heaterState);
808.              break;
809.            }
810.
811.          break;
812.          case 3:
813.            maxMenuCount= 2;
814.            switch(currentSubmenuNumber)
815.            {
816.              case 0:
817.                printCentered(F("Humidity Control"),0);
818.                display.setTextColor(BLACK,WHITE);
819.                display.print(F("Max Humidity      "));
820.                display.println(maxHumidity);
821.                display.setTextColor(WHITE);
822.                display.print(F("Min Humidity      "));
823.                display.println(minHumidity);
824.                display.print(F("Spray State       "));
825.                display.println(deviceStateStrings[sprayState]);
826.                display.print(F("Inside Humidity   "));
827.                display.println(insideHumidity);
828.                display.print(F("Outside Humidity "));
829.                display.println(outsideHumidity);
830.                display.display();
831.                maxHumidity= changeSubmenuVariable(maxHumidity);
832.              break;
833.              case 1:
834.                printCentered(F("Humidity Control"),0);
835.                display.print(F("Max Humidity      "));
836.                display.println(maxHumidity);
837.                display.setTextColor(BLACK,WHITE);
838.                display.print(F("Min Humidity      "));
839.                display.println(minHumidity);
840.                display.setTextColor(WHITE);
841.                display.print(F("Spray State       "));
842.                display.println(deviceStateStrings[sprayState]);
843.                display.print(F("Inside Humidity   "));
844.                display.println(insideHumidity);
845.                display.print(F("Outside Humidity "));
846.                display.println(outsideHumidity);
847.                display.display();
848.                minHumidity= changeSubmenuVariable(minHumidity);
849.              break;
850.              case 2:
851.                printCentered(F("Humidity Control"),0);
852.                display.print(F("Max Humidity      "));
853.                display.println(maxHumidity);
854.                display.print(F("Min Humidity      "));
855.                display.println(minHumidity);
856.                display.setTextColor(BLACK,WHITE);
857.                display.print(F("Spray State       "));
858.                display.println(deviceStateStrings[sprayState]);
```

```
859.                display.setTextColor(WHITE);
860.                display.print(F("Inside Humidity    "));
861.                display.println(insideHumidity);
862.                display.print(F("Outside Humidity "));
863.                display.println(outsideHumidity);
864.                display.display();
865.                sprayState= changeDeviceState(sprayState);
866.              break;
867.            }
868.          break;
869.          case 4:
870.            maxMenuCount= 2;
871.            switch(currentSubmenuNumber)
872.            {
873.              case 0:
874.                printCentered(F("Irrigation Controll"),0);
875.                display.setTextColor(BLACK,WHITE);
876.                display.print(F("Max Moisture       "));
877.                display.println(maxMoisture);
878.                display.setTextColor(WHITE);
879.                display.print(F("Min Moisture       "));
880.                display.println(minMoisture);
881.                display.print(F("Irrigation State "));
882.                display.println(deviceStateStrings[irrigationState]);
883.                display.print(F("Moisture Level     "));
884.                display.println(soilMoistureLevel);
885.                display.display();
886.                maxMoisture= changeSubmenuVariable(maxMoisture);
887.              break;
888.              case 1:
889.                printCentered(F("Irrigation Controll"),0);
890.                display.print(F("Max Moisture       "));
891.                display.println(maxMoisture);
892.                display.setTextColor(BLACK,WHITE);
893.                display.print(F("Min Moisture       "));
894.                display.println(minMoisture);
895.                display.setTextColor(WHITE);
896.                display.print(F("Irrigation State "));
897.                display.println(deviceStateStrings[irrigationState]);
898.                display.print(F("Moisture Level     "));
899.                display.println(soilMoistureLevel);
900.                display.display();
901.                minMoisture= changeSubmenuVariable(minMoisture);
902.              break;
903.              case 2:
904.                printCentered(F("Irrigation Controll"),0);
905.                display.print(F("Max Moisture       "));
906.                display.println(maxMoisture);
907.                display.print(F("Min Moisture       "));
908.                display.println(minMoisture);
909.                display.setTextColor(BLACK,WHITE);
910.                display.print(F("Irrigation State "));
911.                display.println(deviceStateStrings[irrigationState]);
912.                display.setTextColor(WHITE);
913.                display.print(F("Moisture Level     "));
914.                display.println(soilMoistureLevel);
915.                display.display();
916.                irrigationState= changeDeviceState(irrigationState);
917.              break;
918.            }
919.          break;
920.          case 5:
921.            maxMenuCount= 1;
922.            switch(currentSubmenuNumber)
923.            {
924.              case 0:
925.                printCentered(F("Light Control"),0);
926.                display.setTextColor(BLACK,WHITE);
927.                display.print(F("Min Light Level: "));
928.                display.println(minLightLevel);
```

```
929.                display.setTextColor(WHITE);
930.                display.print(F("Ligth State        "));
931.                display.println(deviceStateStrings[lightState]);
932.                display.print(F("Ligth Intensity    "));
933.                display.println(lightIntensity);
934.                display.display();
935.                minLightLevel= changeSubmenuVariable(minLightLevel);
936.              break;
937.              case 1:
938.                printCentered(F("Light Control"),0);
939.                display.print(F("Min Light Level: "));
940.                display.println(minLightLevel);
941.                display.setTextColor(BLACK,WHITE);
942.                display.print(F("Ligth State        "));
943.                display.println(deviceStateStrings[lightState]);
944.                display.setTextColor(WHITE);
945.                display.print(F("Ligth Intensity    "));
946.                display.println(lightIntensity);
947.                display.display();
948.                lightState= changeDeviceState(lightState);
949.              break;
950.
951.            }
952.          break;
953.          case 6:
954.            maxMenuCount= 5;
955.            switch(currentSubmenuNumber)
956.            {
957.              case 0:
958.                printCentered(F("System Setting"),0);
959.                display.setTextColor(BLACK,WHITE);
960.                display.println(F("Restart Sensors "));
961.                display.setTextColor(WHITE);
962.                display.println(F("Restart Devices "));
963.                display.print(F("Language          "));
964.                display.println(languages[languageSet]);
965.                display.print(F("Enable Serial      "));
966.                display.println(booleanStateOnOff(enableSerial));
967.                display.print(F("Change Slowenes    "));
968.                display.println(changeSpeed);
969.                display.println(F("Restart Arduino   "));
970.                display.print(F("Name: "));
971.                display.print(SERIALNUMBER);
972.                display.display();
973.                if(joystickAxisYMoved())
974.                {
975.                  zeroAllSensor();
976.                  display.clearDisplay();
977.                  printCentered(F("Sensors Restarted"),16);
978.                  display.display();
979.                  delay(500);
980.                }
981.              break;
982.              case 1:
983.                printCentered(F("System Setting"),0);
984.                display.println(F("Restart Sensors "));
985.                display.setTextColor(BLACK,WHITE);
986.                display.println(F("Restart Devices "));
987.                display.setTextColor(WHITE);
988.                display.print(F("Language          "));
989.                display.println(languages[languageSet]);
990.                display.print(F("Enable Serial      "));
991.                display.println(booleanStateOnOff(enableSerial));
992.                display.print(F("Change Slowenes    "));
993.                display.println(changeSpeed);
994.                display.println(F("Restart Arduino   "));
995.                display.print(F("Name: "));
996.                display.print(SERIALNUMBER);
997.                display.display();
998.                if(joystickAxisYMoved("left"))
```

```
 999.                {
1000.                  killAllDevices();
1001.                  display.clearDisplay();
1002.                  printCentered(F("Devices Restarted"),16);
1003.                  display.display();
1004.                  delay(500);
1005.                }
1006.              break;
1007.              case 2:
1008.                printCentered(F("System Setting"),0);
1009.                display.println(F("Restart Sensors "));
1010.                display.println(F("Restart Devices "));
1011.                display.setTextColor(BLACK,WHITE);
1012.                display.print(F("Language          "));
1013.                display.println(languages[languageSet]);
1014.                display.setTextColor(WHITE);
1015.                display.print(F("Enable Serial     "));
1016.                display.println(booleanStateOnOff(enableSerial));
1017.                display.print(F("Change Slowenes   "));
1018.                display.println(changeSpeed);
1019.                display.println(F("Restart Arduino    "));
1020.                display.print(F("Name: "));
1021.                display.print(SERIALNUMBER);
1022.                display.display();
1023.                languageSet= languageChooser(languageSet);
1024.              break;
1025.              case 3:
1026.                printCentered(F("System Setting"),0);
1027.                display.println(F("Restart Sensors "));
1028.                display.println(F("Restart Devices "));
1029.                display.print(F("Language          "));
1030.                display.println(languages[languageSet]);
1031.                display.setTextColor(BLACK,WHITE);
1032.                display.print(F("Enable Serial     "));
1033.                display.println(booleanStateOnOff(enableSerial));
1034.                display.setTextColor(WHITE);
1035.                display.print(F("Change Slowenes    "));
1036.                display.println(changeSpeed);
1037.                display.println(F("Restart Arduino    "));
1038.                display.print(F("Name: "));
1039.                display.print(SERIALNUMBER);
1040.                display.display();
1041.                changeSubmenuBoolean(enableSerial);
1042.              break;
1043.              case 4:
1044.                printCentered(F("System Setting"),0);
1045.                display.println(F("Restart Sensors "));
1046.                display.println(F("Restart Devices "));
1047.                display.print(F("Language          "));
1048.                display.println(languages[languageSet]);
1049.                display.print(F("Enable Serial     "));
1050.                display.println(booleanStateOnOff(enableSerial));
1051.                display.setTextColor(BLACK,WHITE);
1052.                display.print(F("Change Slowenes    "));
1053.                display.println(changeSpeed);
1054.                display.setTextColor(WHITE);
1055.                display.println(F("Restart Arduino    "));
1056.                display.print(F("Name: "));
1057.                display.print(SERIALNUMBER);
1058.                display.display();
1059.                changeSpeed= changeSubmenuVariable(changeSpeed);
1060.                if(changeSpeed<= MIN_CHANGE_SPEED)
1061.                {
1062.                  changeSpeed= MAX_CHANGE_SPEED;
1063.                }
1064.                else if(changeSpeed> MAX_CHANGE_SPEED)
1065.                  {
1066.                  changeSpeed= MIN_CHANGE_SPEED;
1067.                  }
1068.              break;
```

```
1069.             case 5:
1070.               printCentered(F("System Setting"),0);
1071.               display.println(F("Restart Sensors "));
1072.               display.println(F("Restart Devices "));
1073.               display.print(F("Language          "));
1074.               display.println(languages[languageSet]);
1075.               display.print(F("Enable Serial     "));
1076.               display.println(booleanStateOnOff(enableSerial));
1077.               display.print(F("Change Slowenes   "));
1078.               display.println(changeSpeed);
1079.               display.setTextColor(BLACK,WHITE);
1080.               display.println(F("Restart Arduino    "));
1081.               display.setTextColor(WHITE);
1082.               display.print(F("Name: "));
1083.               display.print(SERIALNUMBER);
1084.               display.display();
1085.               if(joystickAxisYMoved())
1086.               {
1087.                 showMessageOnScreen("Arduino restarting");
1088.                 restartArduino();
1089.               }
1090.
1091.             break;
1092.           }
1093.         break;
1094.         case 7:
1095.           maxMenuCount= 1;
1096.           printCentered(F("Network Settings"),0);
1097.           switch (currentSubmenuNumber)
1098.           {
1099.             case 0:
1100.               display.setTextColor(BLACK,WHITE);
1101.               display.println(F("Left for reset"));
1102.               if(joystickAxisY   >    JOYSTICK_MAXTRESHOLD)
1103.               {
1104.                 display.clearDisplay();
1105.                 printCentered(F("Network restarting"),16);
1106.                 display.display();
1107.                 espSetup();
1108.                 getIp();
1109.                 display.clearDisplay();
1110.                 printCentered(F("Network restarted"),16);
1111.                 display.display();
1112.                 Serial.println(F("Esp restarted"));
1113.                 delay(500);
1114.               }
1115.               display.setTextColor(WHITE);
1116.               display.print("Operation mode: ");
1117.               if(networkType)
1118.                 display.println("AP");
1119.               else
1120.                 display.println("STA");
1121.             break;
1122.
1123.             case 1:
1124.               display.setTextColor(WHITE);
1125.               display.println(F("Left for reset"));
1126.               display.setTextColor(BLACK,WHITE);
1127.               display.print("Operation mode: ");
1128.               changeSubmenuBoolean(networkType);
1129.               if(networkType)
1130.                 display.println("AP");
1131.               else
1132.                 display.println("STA");
1133.                 display.setTextColor(WHITE);
1134.             break;
1135.
1136.           }
1137.           display.print(F("Network Name: "));
1138.             display.println(ssid);
```

```
1139.                 display.print(F("Password: "));
1140.                 display.println(password);
1141.                 display.print(F("Ip address: "));
1142.                 display.println(ipAddress);
1143.                 display.display();
1144.           break;
1145.       }
1146.
1147.    }
1148.
1149.
1150. }
1151. void setup() // runs on startup
1152. {
1153.    // Setup resz, runs once at startup:
1154.    Serial.begin(9600); //Serial Communication to the tester
1155.    Serial1.begin(115200);
1156.    //call the functions created, which should be executed on the first run
1157.    displayInitialize();
1158.    //give the, to the function of the spikes
1159.    pinMode(JOYSTICK_BUTTON, INPUT_PULLUP);
1160.    pinMode(FANPIN, OUTPUT);
1161.    pinMode(IRRIGATIONPIN, OUTPUT);
1162.    pinMode(SPRAYPIN, OUTPUT);
1163.    pinMode(HEATERPIN, OUTPUT);
1164.    pinMode(LIGHTPIN, OUTPUT);
1165.
1166. }
1167.
1168. void loop() {
1169.
1170.    // put your main code here, to run repeatedly:
1171.    readJoystickValues();
1172.    if(enableSerial) //sends data serially only if it receives input serially, so no
resources are wasted
1173.       serialMonitorPrint();
1174.    changeMenuNumber(isSubMenu,maxMenuCount);
1175.    mainMenuSystem();
1176.    if(joystickButtonState== true)
1177.    {
1178.       if(millis()-lastTimeChange> menuChangeInterval)
1179.       {
1180.          isSubMenu= !isSubMenu;
1181.          maxMenuCount= MAX_MENU_COUNT;
1182.          currentSubmenuNumber= 0;
1183.          lastTimeChange= millis(); 1184.
             }
1185.
1186.    }
1187.    transmitDateOnEsp(constructDateString());
1188.    readSensor();
1189.    outRelays();
1190.    display.clearDisplay();
1191. }
1192.
```

Bibliography

https://hu.wikipedia.org/wiki/Arduino#Szoftver

https://en.wikipedia.org/wiki/Integrated_development_environment

https://web.archive.org/web/20120227114343/http://tldp.fsf.hu/HOWTO/Program-Library-HOWTO-hu/index.html

https://docs.arduino.cc/hardware/mega-2560/

https://hu.wikipedia.org/wiki/I%C2%B2C

https://web.archive.org/web/20130511150526/http://www.nxp.com/documents/user_manual/UM10204.pdf

https://en.wikipedia.org/wiki/Hayes_AT_command_set

https://web.archive.org/web/20151028101531/http://www.zoomtel.com/documentation/dial_up/100498E.pdf

https://docs.arduino.cc/language-reference/

https://docs.arduino.cc/hardware/mega-2560/

https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf

https://docs.arduino.cc/software/ide/#ide-v1

https://docs.arduino.cc/programming/

https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1741392150506&ref_url=https%253A%252F%252Fwww.google.com%252F

https://docs.arduino.cc/micropython/micropython-course/course/serial/

https://docs.arduino.cc/libraries/ssd1306/

https://github.com/lexus2k/ssd1306

https://docs.arduino.cc/libraries/adafruit-gfx-library/

https://github.com/adafruit/Adafruit-GFX-Library

https://github.com/bonezegei/Bonezegei_DHT11

https://docs.arduino.cc/libraries/dht11/

https://github.com/dhrubasaha08/DHT11

https://blog.embeddedexpert.io/?p=613

https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/

https://components101.com/modules/joystick-module

https://www.researchgate.net/figure/The-pin-diagram-of-DHT11-temperature-sensors_fig2_359068957

https://www.google.com/url?sa=i&url=https%3A%2F%2Flastminuteengineers.com%2Fcapacitive-soil-moisture-sensor-arduino%2F&psig=AOvVaw3d6lzLw9fkb919Blm9CME5&ust=1744631596137000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCIjpmrD51IwDFQAAAAAdAAAAABAE
https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
https://dcc-ex.com/reference/hardware/wifi-boards/esp-01.html#gsc.tab=0
https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.analog.com%2Fen%2Fresources%2Ftechnical-articles%2Fi2c-primer-what-is-i2c-part-1.html&psig=AOvVaw0BWqK9NhBzePunKXn-zUoF&ust=1744908285940000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCPjLr4yA3YwDFQAAAAAdAAAAABA9
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.industrialshields.com%2Fblog%2Farduino-industrial-1%2Fi2c-bus-on-the-arduino-based-plc-for- industrial-automation-192&psig=AOvVaw0BWqK9NhBzePunKXn-zUoF&ust=1744908285940000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCPjLr4yA3YwDFQAAAAAdAAAAABBU
https://hu.wikipedia.org/wiki/Verzi%C3%B3kezel%C3%A9s