IRR_MEGA2560

1.1 SYSTEM ARCHITECTURE

1. **Initialization**: ESP8266 initializes serial communication, SPIFFS, WiFi, MQTT connection, and NTP time synchronization.
2. **Event Handling**: ESP8266 processes event data from Mega 2560 to determine actions based on temperature, VPD, and irrigation events.
3. **Manual Relay Status**: Updates and manages manual relay statuses based on incoming JSON data from Mega 2560.
4. **Alert Management**: Processes and handles alert modes and levels, triggering actions based on alert data.
5. **Data Transmission to Cloud**: ESP8266 publishes sensor data and status updates to a cloud server via MQTT.
6. **MQTT Communication**: Manages MQTT connections, handles publishing and subscribing to topics, and processes incoming messages.
7. **Timing and Scheduling**: Executes periodic tasks, such as sending data, updating statuses, and managing time-based functions.

1.2 LIBRARIES:

1. **Arduino.h:** Core Arduino functions.
2. **MemoryFree.h:** Monitors available memory.
3. **ArduinoJson.h:** Handles JSON data for communication or configuration.
4. **RunningAverage.h:** Calculates running averages, likely for sensor data smoothing.
5. **Wire.h:** Facilitates I2C communication.
6. **RTClib.h, Time.h, TimeLib.h:** Manages real-time clock (RTC) functionalities and timekeeping.
7. **EEPROMex.h:** Extended EEPROM functionalities for storing persistent settings.

1.3 GLOBAL VARIABLES:

1. **Timing Variables:** Track current time (hours, minutes, seconds) and scheduled irrigation times.
2. **Relay States:** Manage the states (ON/OFF) of various relays controlling pumps and valves.
3. **Sensor Data:** Variables to store water level measurements from ultrasonic sensors and calculate water volume.
4. **EEPROM Addresses:** Predefined memory addresses to store and retrieve irrigation schedules and pump settings.
5. **Pump and Valve Control:** Boolean flags and pin assignments for controlling different pumps and solenoid valves in the irrigation system.

1.4 FUNCTION DECLARATIONS:

- **Utility Functions:** Handle password management (`pwdname()`, `pwd()`), mesh networking (`meshpass()`), time updates (`timeMin()`, `timeHr()`, etc.), data transmission (`sendData(String)`), and scheduling (`scheduleFun()`).

1.4.1 Class `Switcher`:

The `Switcher` class is designed to control a relay by toggling its state between on and off at specified intervals. It uses the `millis()` function to keep track of time without blocking the main program execution, allowing other tasks to run concurrently.

```
class Switcher

{

   // class member variables

   byte relayPin; // number of pin for relay

   long OnTime;

   long OffTime;

   int relayState; // set relay state (active HIGH)

   unsigned long previousMillis; // set time since last update

 public:

   Switcher(byte pin)

   {

     relayPin = pin;

   }



   void begin(long on, long off)

   {

     pinMode(relayPin, OUTPUT);

     OnTime = on;

     OffTime = off;

     relayState = HIGH;

     previousMillis = 0;
```

```
  status1 = true;
}
void Update()
{

  unsigned long currentMillis = millis();
  if ((currentMillis < OffTime) && (status1 == true))
  {
    Serial.println("ok");
    currentMillis =  OffTime;
    status1 = false;
  }

  if ((relayState == HIGH) && (currentMillis - previousMillis >= OffTime))
  {
    unsigned long currentMillis = millis();
    relayState = LOW; // Turn it off
    Serial.println("relay ON");
    previousMillis = currentMillis; // Remember the time
    digitalWrite(relayPin, relayState); //update the relay
  }
  else if ((relayState == LOW) && (currentMillis - previousMillis >= OnTime))
  {
    relayState = HIGH ; // turn it on
```

```
        previousMillis = currentMillis;
        Serial.println("relay OFF");
        digitalWrite(relayPin, relayState);
      }
    }
};
```

1.4.2 othr():

- Extracts and converts the hour value for another operation from a specific string format.

```
int othr() {

  int val = dfd.indexOf("g") + 1;//
  dfd.remove(0, val);
  otHr = dfd.toInt();
  return (otHr);
}
```

1.4.3 otmin():

- Extracts and converts the minute value for another operation from a specific string format.

```
int otmin() {
  char *nt = strtok(buff, "g");//off time hr val
  str = nt;
  int val = str.indexOf("f") + 1;
  str.remove(0, val);
  otMin = str.toInt();
```

```
  return (otMin);
}
```

1.4.4 ofthr():

- Extracts and converts the hour value for the off time from a specific string format.

```
int ofthr() {
  char *fr = strtok(buff, "f");
  str = fr;
  int val = str.indexOf("e") + 1;
  str.remove(0, val);
  oftHr = str.toInt();
  return (oftHr);
}
```

1.4.5 oftmin():

- Extracts and converts the minute value for the off time from a specific string format.

```
int oftmin() {
  char *ui = strtok(buff, "e");
  str = ui;
  str.remove(0, 5);
  oftMin = str.toInt();
  return (oftMin);
}
```

1.4.6 `Z1PAsavetime()`

Purpose: Saves the start time (hour and minute) of a specific event to EEPROM memory.

```
void Z1PAsavetime() {
```

```
  Z1PASThr = othr();  // Get the hour value
  Z1PASTmin = otmin(); // Get the minute value
  EEPROM.update(EepromZ1PASThr, Z1PASThr); // Store hour in EEPROM
  EEPROM.update(EepromZ1PASTmin, Z1PASTmin); // Store minute in EEPROM
}
```

1.4.7 `Z1PAsavedue()`

Purpose: Saves the due time (hour and minute) of a specific event to EEPROM memory.

```
void Z1PAsavedue() {
  Z1PASPhr = ofthr();  // Get the due hour value
  Z1PASPmin = oftmin(); // Get the due minute value
  EEPROM.update(EepromZ1PASPhr, Z1PASPhr); // Store hour in EEPROM
  EEPROM.update(EepromZ1PASPmin, Z1PASPmin); // Store minute in EEPROM
}
```

1.4.8 `EepromReadZ1AP()`

Purpose: Reads the saved start and due times from EEPROM memory.

```
void EepromReadZ1AP() {
  Z1PASThr = EEPROM.read(EepromZ1PASThr); // Read start hour from EEPROM
  Z1PASTmin = EEPROM.read(EepromZ1PASTmin); // Read start minute from EEPROM
  Z1PASPhr = EEPROM.read(EepromZ1PASPhr); // Read due hour from EEPROM
  Z1PASPmin = EEPROM.read(EepromZ1PASPmin); // Read due minute from EEPROM
}
```

1.4.9 `Z1PAsavetimeq()`

Purpose: Saves the start time for a different event schedule to EEPROM memory.

```
void Z1PAsavetimeq() {
```

```
  Z1PASThrq = othr();  // Get the hour value
  Z1PASTminq = otmin(); // Get the minute value
  EEPROM.update(Eepromsch8hr, Z1PASThrq); // Store hour in EEPROM
  EEPROM.update(Eepromsch8min, Z1PASTminq); // Store minute in EEPROM
}
```

1.4.10 `Z1PAsavedueq()`

Purpose: Saves the due time for a different event schedule to EEPROM memory.

```
void Z1PAsavedueq() {
  Z1PASPhrq = ofthr();  // Get the due hour value
  Z1PASPminq = oftmin(); // Get the due minute value
  EEPROM.update(Eepromsch8hr1, Z1PASPhrq); // Store hour in EEPROM
  EEPROM.update(Eepromsch8min1, Z1PASPminq); // Store minute in EEPROM
}
```

1.4.11 `EepromReadZ1APq()`

Purpose: Reads the saved start and due times for a different event schedule from EEPROM memory.

```
void EepromReadZ1APq() {
  Z1PASThrq = EEPROM.read(Eepromsch8hr); // Read start hour from EEPROM
  Z1PASTminq = EEPROM.read(Eepromsch8min); // Read start minute from EEPROM
  Z1PASPhrq = EEPROM.read(Eepromsch8hr1); // Read due hour from EEPROM
  Z1PASPminq = EEPROM.read(Eepromsch8min1); // Read due minute from EEPROM
}
```

1.4.12 `Z1PAsavetimeqq()`

Purpose: Saves the start time for yet another event schedule to EEPROM memory and prints the values to the Serial Monitor.

```
void Z1PAsavetimeqq() {
```

```
  Z1PASThrqq = othr();  // Get the hour value
  Z1PASTminqq = otmin(); // Get the minute value


  Serial.println("hr");
  Serial.println(Z1PASThrqq); // Print hour to Serial Monitor


  Serial.println("min");
  Serial.println(Z1PASTminqq); // Print minute to Serial Monitor


  EEPROM.update(Eepromsch9hr, Z1PASThrqq); // Store hour in EEPROM
  EEPROM.update(Eepromsch9min, Z1PASTminqq); // Store minute in EEPROM
}
```

1.4.13 `Z1PAsaveduеqq()`

Purpose: Saves the due time for yet another event schedule to EEPROM memory.

```
void Z1PAsaveduеqq() {
  Z1PASPhrqq = ofthr();  // Get the due hour value
  Z1PASPminqq = oftmin(); // Get the due minute value


  EEPROM.update(Eepromsch9hr1, Z1PASPhrqq); // Store hour in EEPROM
  EEPROM.update(Eepromsch9min1, Z1PASPminqq); // Store minute in EEPROM
}
```

1.4.14 `EepromReadZ1APqq()`

Purpose: Reads the saved start and due times for yet another event schedule from EEPROM memory.

```
void EepromReadZ1APqq() {
```

```
  Z1PASThrqq = EEPROM.read(Eepromsch9hr1); // Read start hour from EEPROM

  Z1PASTminqq = EEPROM.read(Eepromsch9min); // Read start minute from EEPROM

  Z1PASPhrqq = EEPROM.read(Eepromsch9hr1); // Read due hour from EEPROM

  Z1PASPminqq = EEPROM.read(Eepromsch9min1); // Read due minute from EEPROM

}
```

1.4.15 `SerialCom()`

The `SerialCom` function handles serial communication between an Arduino (or compatible microcontroller) and a connected display (like a Nextion display). It processes incoming commands from the display, updates various settings or schedules, and saves them to EEPROM. It also handles WiFi credentials, mesh network settings, and RTC (Real-Time Clock) data.

1.4.16 `sendData()`

The `sendData` function is responsible for controlling various pumps, lights, and valves based on the command received (dfd). The function interprets the command string and activates or deactivates the corresponding device. It also handles time-based operations for certain devices by calculating the total on and off durations.

1.5 OVERALL WORKFLOW

1. **Initialization:**
   - Set up RTC, initialize sensors, relays, and load saved settings from EEPROM.
2. **Time Management:**
   - Continuously track current time using RTC to trigger irrigation based on the schedule.
3. **Sensor Monitoring:**
   - Regularly read water levels using ultrasonic sensors to ensure adequate water supply and prevent overflows.
4. **Irrigation Control:**
   - Activate/deactivate pumps and valves based on the schedule and sensor data to manage water distribution across different zones.
5. **Data Handling:**
   - Use JSON for communication, possibly for interfacing with a display or remote monitoring system.
   - Store and retrieve schedules and settings from EEPROM to maintain configurations.
6. **User Interaction:**
   - Functions like `pwdname()` and `pwd()` suggest password-protected settings or controls, enhancing security.