Watering System

HALMAI ERIK 30432

1. Introduction

I am making a time-based watering system that can be controlled by connecting to a Wi-Fi network and going to the system's website. On the website, the user can input the time interval between each irrigation and the system will water the flowers, using a water pump, according to the input. The system also detects the amount of water that is around the water pump and when it gets under a certain amount, a led will turn on to prompt the user to refill the water.

2. Design

To connect the system to the internet, I will be using the ESP8266 module. Because the standalone ESP8266 module's hardware IO configuration needs a lot of redundant work, I choose a NodeMCU LoLin V3 development kit, which has an integrated ESP8266 and offers Arduino-like hardware IO. I will set up the site and the input processing using this development kit.

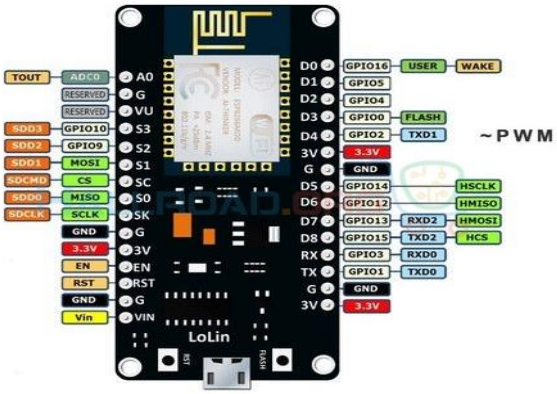

Figure 1 NodeMCu LoLin V3

For the water pump, I will be using a 3-6V DC pump that can pump 80-120L/H and I will place it in a water tank. I will also connect a rubber tube to the pump that will go the plant that I want to water. Because the voltage provided by the NodeMCU isn't enough to power the pump, I connected it to an Arduino Uno and I made the NodeMCU and Uno to communicate using serial communication.

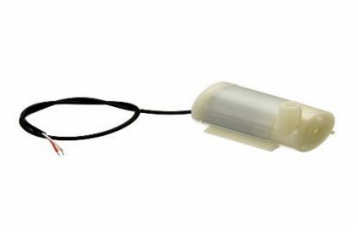

Figure 2 Water pump

Figure 3 Arduino Uno

To detect the water level around the pump, I will be using an analog water-level sensor that I will put inside the water tank. To prompt the user when there isn't enough water, I will also use an LED.

Figure 4 Water level sensor

3. Implementation

The following block diagram shows the connection between each component. The NodeMCU has an analog input signal, in the form of the water level sensor, a LED output signal and it is connected to the Internet using the specified Wi-Fi network, waiting for the user's input on the website. The Arduino Uno has a single DC motor connected to it, the water pump, which will be written high when the timer reaches the value inputted by the user. The NodeMCU will send the input data received on the site to the Uno, using serial communication.

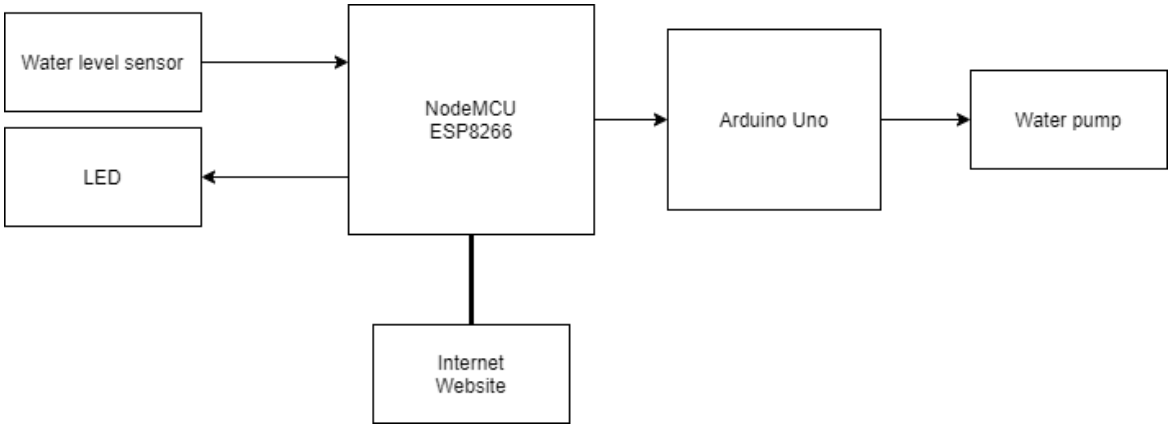

Figure 5 Block diagram

Because I used two microcontrollers, I created two .ino files, one for the Arduino Uno and one for the NodeMCU.

Using the NodeMCU's ESP8266, I made the website using the ESPAsyncWebServer library, which builds an asynchronous web server where the user can submit his/her input by using a text field and a submit button. I first connected the ESP8266 to my local WiFi and I created the website's HTML code and sent it to the client. After that, I handled the /get requests, generated by the submit button and the text

field, and parsed the inputted text. The inputted text is than sent to the Arduino Uno as an array of chars with a '_' character at the end, so that the Arduino knows when to stop reading.

```
// WiFi
AsyncWebServer server(80);
const char* ssid = "Orange-69vy";
const char* password = "wfEZ9myy";
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
  <title>ESP Input Form</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  </head><body>
  <form action="/get">
    input: <input type="text" name="input">
    <input type="submit" value="Submit">
  </form>
  </body></html>)rawliteral";

void notFound(AsyncWebServerRequest *request) {
  request->send(404, "text/plain", "Not found");
}
// Send web page with input fields to client
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html);
});

// Send a GET request to <ESP_IP>/get?input=<inputMessage>
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
  String inputTime;
  // GET input1 value on <ESP_IP>/get?input1=<inputMessage>
  if (request->hasParam("input")) {
    inputTime = request->getParam("input")->value();
  }
  else {
    inputTime = "No message sent";
  }
  //Serial.println(inputTime);
  if (!inputTime.equals("No message sent")) {
    char t[10];
    inputTime.toCharArray(t, 10);
    t[8] = '_';
    t[9] = '\0';
    Serial.println(t);
    s.write(t);
  }
  request->send(200, "text/html", "Irrigation will happen every " + inputTime +
                              "<br><a href=\"/\">Return to Home Page</a>");
});
server.onNotFound(notFound);
server.begin();
```

Figure 6 Webserver code

To connect the Uno and the NodeMCU, I used the SoftwareSerial header, which allows serial communication on any digital pins. I used the D6 and D5 pins of the NodeMCU and the 5 and 6 pins on the Arduino Uno.

On the Uno's side, I read the sent chars, parse them as integers and I use a 1 second timer interrupt to increment the hour, min and sec variables. When the variables reach the sent values, the pump is turned on for five seconds, the variables are zeroed and the interrupt process repeats.

```
if(s.available() > 0) {
  char inChar = (char) s.read();
  if (inChar != '_') {
     buff[pos++] = inChar;
     //Serial.println(inChar);
  }
  else {
    sscanf(buff, "%d:%d:%d", &irrHr, &irrMin, &irrSec);
    Serial.print(irrHr);
    Serial.print(":");
    Serial.print(irrMin);
    Serial.print(":");
    Serial.print(irrSec);
    Serial.println();
    pos = 0;
    secs = 0;
    mins = 0;
    hrs = 0;
  }
}

if (irrHr != -1 && irrMin != -1 && irrSec != -1) {
  if (hrs == irrHr && mins == irrMin && secs == irrSec) {
    Serial.print(hrs);
    Serial.print(":");
    Serial.print(mins);
    Serial.print(":");
    Serial.print(secs);
    Serial.println();
    digitalWrite(pumpPin, LOW);
    delay(5000);

    secs = 0;
    mins = 0;
    hrs = 0;
  }
}
digitalWrite(pumpPin, HIGH);
```

Figure 7 Input processing and pump turn on

```
// Timer
secs = 0;
mins = 0;
hrs = 0;
cli();
TCCR1A = 0;
TCCR1B = 0;
OCR1A = 15624;
TCCR1B |= (1 << WGM12);
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);
TIMSK1 |= (1 << OCIE1A);
sei();
```

Figure 8 Timer setup

```
ISR(TIMER1_COMPA_vect)
{
  secs = secs + 1;

  if (secs == 60) {
    mins++;
    secs = 0;
  }
  if (mins == 60) {
    hrs++;
    mins = 0;
  }
```

Figure 8 Timer interrupt function

4. Results

The project mostly works as expected. In the beginning I wanted to only use a single microcontroller, the NodeMCU, but I think the voltage wasn't enough, so I added the Uno too. In the future I could also replace it with a battery. Besides this, I can also improve the website in the future and add input validation.

I also wanted to create a phone application to control the system, but I couldn't use my phone because it isn't an Android device and I don't know how to code in Objective C or Swift for IOS.

5. Conclusions

This was a very interesting project and I learned a lot by doing it. Because the semester was held online, I couldn't really gain any hardware skills, but this project introduced me to that aspect as well. I find in fun to build something and have it as a physical thing that I can maybe use in my house, after some upgrading.