به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

سیستم‌های نهفته بی‌درنگ

گزارش تمرین کامپیوتری اول

علی اخگری ۸۱۰۱۹۸۳۴۱

پریا خوش‌تاب ۸۱۰۱۹۸۳۸۷

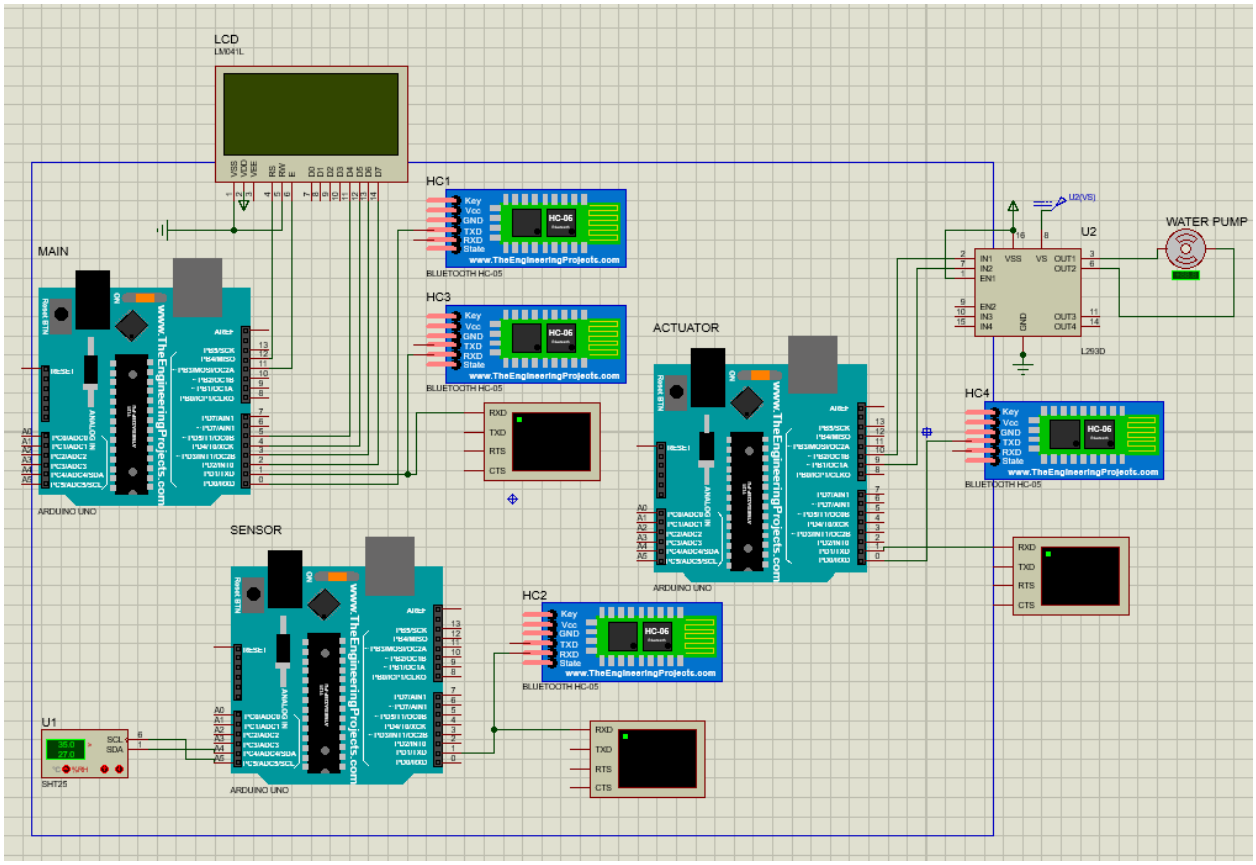
پرنیان فاضل ۸۱۰۱۹۸۵۱۶

مونا محدث مجتهدی ۸۱۰۱۹۸۵۵۷

بهار ۱۴۰۲

فهرست

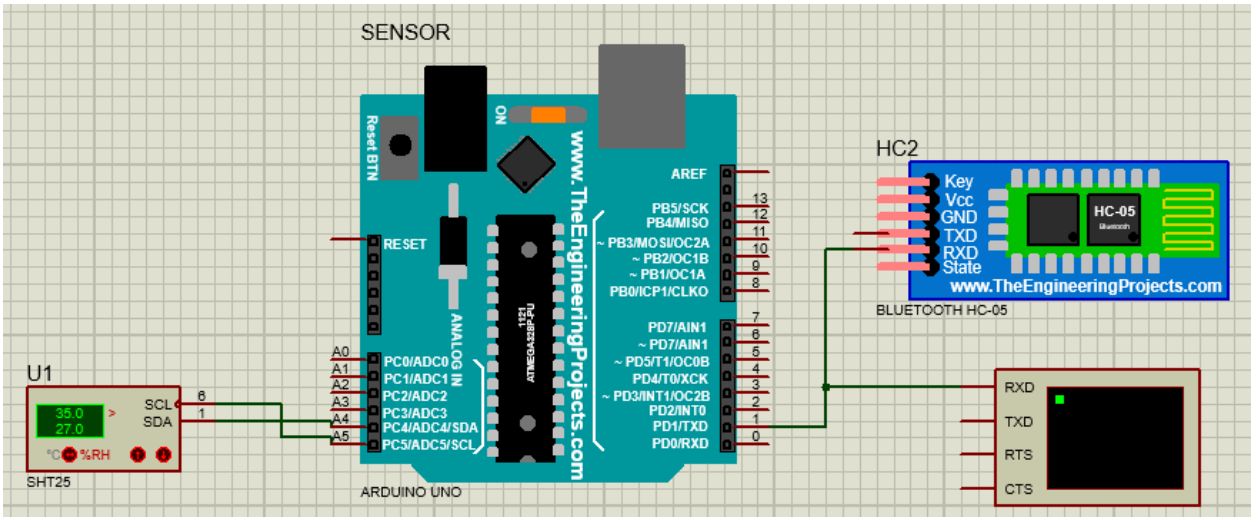
طراحی کلی مدار در محیط Proteus



طراحی فوق، چیدمان اجزای مختلف را در یک سیستم گلدان هوشمند نشان می‌دهد که به منظور خودکار سازی فرایند آبیاری گیاهان بر اساس شرایط محیطی در نظر گرفته شده است. همان‌گونه که مشاهده می‌شود، این طراحی از سه گره حسگر، عملگر و مرکزی و همچنین از یک موتور پمپ آب و درایور آن، یک سنسور اندازه‌گیری دما و رطوبت و ماژول‌های ارتباطی بلوتوث تشکیل شده است. از یک LCD و تعدادی virtual terminal به منظور تست و دیباگ کردن استفاده شده است. در ادامه به توضیح هر گره می‌پردازیم.

گره حسگر

شبیه‌سازی در Proteus



ماژول های مورد نیاز برای این گره یک بورد Arduino Uno،یک بلوتوث HC05،یک سنسور SHT25 است و برای تست و دیباگ کردن از یک virtual terminal استفاده کردیم:

- **بورد Arduino**: برای پردازش اطلاعات و ارسال دستورات مورد نیاز
- **HC-05**: یک بلوتوث برای ارتباط بی‌سیم بین گره سنسور و گره مرکزی
- **SHT25**: سنسور اندازه‌گیری دما و رطوبت

حال عملکرد کلی گره سنسور را شرح می‌دهیم:

1. اطلاعات مربوط به دما و رطوبت به طور پیوسته از سنسور مربوطه گرفته می‌شود. (این سنسور قابلیت تنظیم این دو مقدار را هم دارد.)
2. داده‌های دریافت شده از سنسور در بورد مربوطه پردازش می‌شوند و به فرمت مورد نظر تبدیل می‌شوند. رطوبت دریافت شده به فرمت درصد و دمای دریافت شده به سلسیوس تبدیل می‌شوند.
3. اگر مقدار رطوبت استخراج شده به اندازه 5 درصد از آخرین مقدار رطوبت ارسال شده به گره مرکزی متفاوت باشد، مقدار رطوبت و دمای جدید دریافت شده از طریق ارتباط بلوتوثی به گره مرکزی ارسال می‌شود. (با توجه به اینکه از

ارتباط سریالی استفاده می‌شود، امکان دارد گره مرکزی در میانه‌ی ارسال داده، داده را بخواند و داده‌ی معتبری نباشد. به همین دلیل قراردادی را برای فرمت ارسال داده بین گره حسگر و گره مرکزی گذاشتیم. در این قرارداد دما و رطوبت را با کاراکتر / از هم جدا می‌کنیم و انتهای یک پکت را با کاراکتر # مشخص می‌کنیم. در واقع فرمت ارسال داده به صورت humidity/temperature# است.)

شبیه‌سازی در PlatformIO

در ادامه به توضیحات مربوط به پیاده‌سازی کد می‌پردازیم.

در تابع setup یک ارتباط I2C را برای ارتباط با سنسور دما و رطوبت و یک ارتباط UART Serial برای ارتباط بلوتوثی با گره مرکزی ایجاد می‌کنیم. مقدار baud rate برای ارتباط سریالی بلوتوث را 9600 قرار می‌دهیم.

```
void setup(){
   Wire.begin();
   Serial.begin(BLUETOOTH_BAUD_RATE);
}
```

سپس در تابع loop که به طور مداوم اجرا می‌شود داده‌های متناظر با دما و رطوبت را از سنسور می‌خوانیم و بعد شرط فرستادن اطلاعات به گره مرکزی را چک می‌کنیم. با توجه به صورت پروژه، اگر اختلاف آخرین رطوبت ارسال شده به گره مرکزی(lastHumidity) و رطوبت فعلی(currentHumidity)، بیشتر از 5% باشد (یا اولین گزارش دما به گره مرکزی از زمان اجرای برنامه باشد) باید اطلاعات رطوبت و دما به گره مرکزی گزارش شود و مقدار lastHumidity به مقدار currentHumidty آپدیت شود:

```
void loop(){
   static float lastHumidity = INITIAL_HUMIDITY;
   float currentHumidity = getHumidity();
   float currentTemperature = getTemperature();
   if (needToSendSensorData(lastHumidity, currentHumidity)){
       lastHumidity = currentHumidity;
       sendData(currentHumidity, currentTemperature);
   }
}
```

در ادامه نحوه عملکرد توابع getHumidity و getTemperature بررسی می‌کنیم.

در هر دو تابع باید با سنسور SHT25 ارتباط برقرار شود و داده‌ها از آن دریافت شوند. برای این کار از تابع beginTransmission() در کتابخانه Wire.h استفاده می‌کنیم. مقدار ورودی داده شده به این تابع نمایانگر آدرسی است که می‌خواهیم با آن ارتباط برقرار کنیم. طبق اطلاعات مربوط به سنسور SHT25 آدرس I2C مربوطه برابر با 0x40 است که در بالای کد با نام SHT25_I2C_ADDRESS دیفاین شده است. سپس نیاز است تا command codeهایی که می‌خواهیم از آن‌ها اطلاعات را بخوانیم مشخص کنیم که این کد برای رطوبت برابر با 0xF5 و برای دما نیز برابر با 0xF3 است. سپس I2C Transmission را متوقف می‌کنیم.

در مرحله بعد باید با استفاده از تابع requestFrom() آدرس و مقدار بایتی که می‌خواهیم بخوانیم را مشخص کنیم. سپس با استفاده از تابع available() چک می‌کنیم که داده‌ها برای خواندن آماده باشند و با استفاده از تابع read() به خواندن داده‌ها می‌پردازیم.

در انتها نیاز است تا داده‌های خام را به فرمت مد نظرمان (مثلا برای دما سلسیوس) تبدیل کنیم. این تبدیل بر اساس موارد ذکر شده در اطلاعات مربوط به سنسور SHT25 انجام شده است. تصاویر مربوط به توابع getHumidity() و getTemperature() قرار داده شده است:

```
float getHumidity(){
   unsigned int data[2];
   Wire.beginTransmission(SHT25_I2C_ADDRESS);
   Wire.write(CMD_HUM_NOHLD);
   Wire.endTransmission();
   delay(500);
   Wire.requestFrom(SHT25_I2C_ADDRESS, 2);
   if (Wire.available() == 2){
       data[0] = Wire.read();
       data[1] = Wire.read();
       float humidity = (((data[0] * 256.0 + data[1]) * 125.0) / 65536.0) - 6;
       return humidity;
   }
}
```

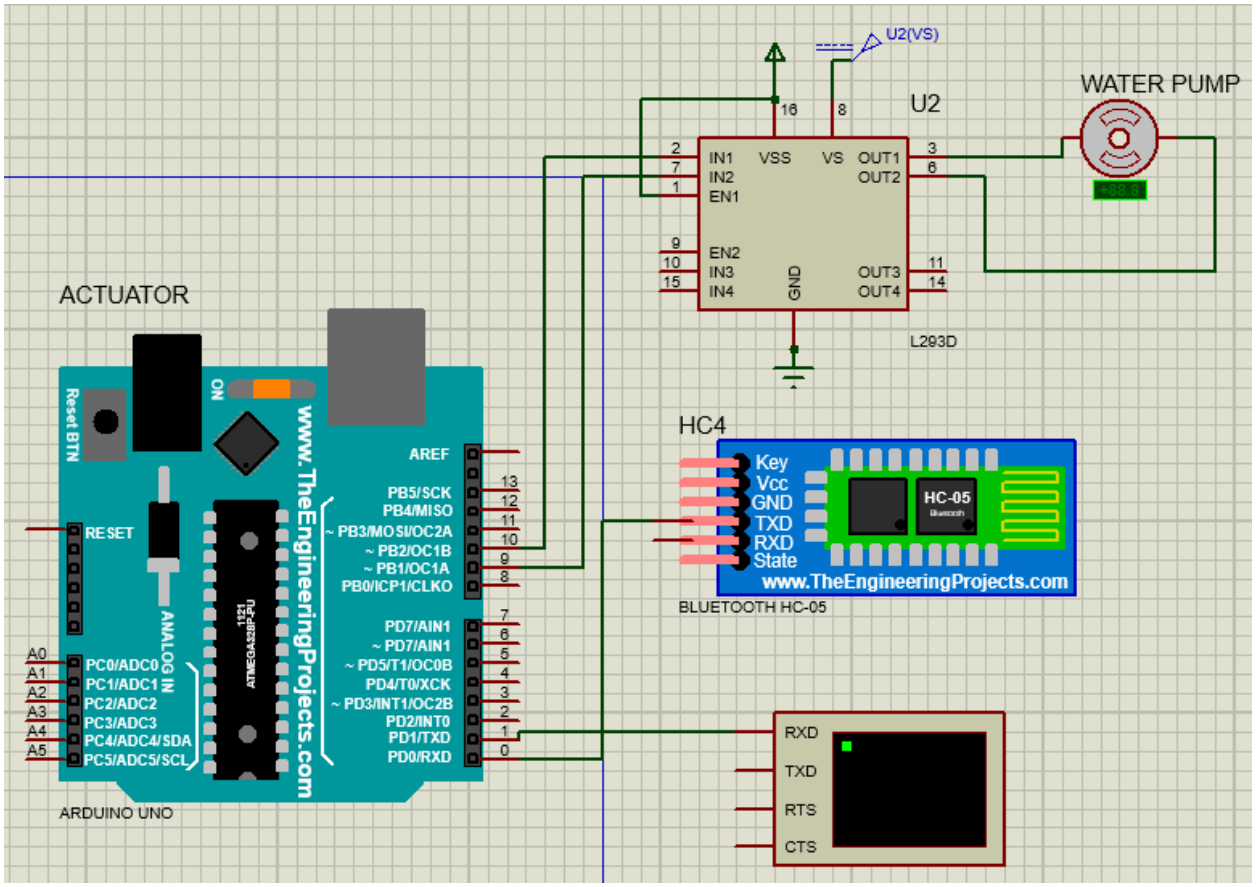
```
float getTemperature(){
   unsigned int data[2];
   Wire.beginTransmission(SHT25_I2C_ADDRESS);
   Wire.write(CMD_TMP_NOHLD);
   Wire.endTransmission();
   delay(500);
   Wire.requestFrom(SHT25_I2C_ADDRESS, 2);
   if (Wire.available() == 2){
       data[0] = Wire.read();
       data[1] = Wire.read();
       float cTemp = (((data[0] * 256.0 + data[1]) * 175.72) / 65536.0) - 46.85;
       return cTemp;
   }
}
```

در تابع sendData() اطلاعات را با استفاده از پروتکل UART و به صورت بی‌سیم ارسال می‌کنیم. ما در رشته ارسالی، رطوبت و دما را با '/' که با نام BLUETOOTH_DATA_SEPERATOR تعریف شده و انتهای خط را با '#' که با نام BLUETOOTH_DATA_DELIMITER تعریف شده، مشخص کردیم تا به درستی خوانده شود:

```
void sendData(float humidity, float temperature)
{
   String message = String(humidity) + BLUETOOTH_DATA_SEPERATOR +
                    String(temperature) + BLUETOOTH_DATA_DELIMITER;
   Serial.println(message);
}
```

گره عملگر

شبیه‌سازی در Proteus



ماژول‌های مورد نیاز برای گره عملگر یک بورد Arduino Uno، یک HC-05، یک L293D و MOTOR-DC می‌باشد.

برای تست و دیباگ کردن از یک virtual terminal استفاده کردیم:

- **بورد Arduino**: برای پردازش اطلاعات و ارسال دستورات مورد نیاز
- **HC-05**: یک بلوتوث برای ارتباط بی‌سیم بین گره عملگر و گره مرکزی
- **MOTOR-DC**: یک موتور DC برای کنترل کردن شیر آب
- **L293D**: یک درایور موتور برای کنترل کردن سرعت و جهت موتور DC

حال عملکرد کلی گره عملگر را شرح می‌دهیم:

1. duty cycle پالس ارسالی به موتور از طریق ارتباط بلوتوثی از گره مرکزی دریافت می‌شود.

2. داده‌ی دریافت شده پردازش می‌شود و duty cycle از آن استخراج می‌شود. (با توجه به اینکه از ارتباط سریالی استفاده می‌شود، امکان دارد گره در میانه‌ی ارسال داده، داده را بخواند و داده‌ی معتبری نباشد. به همین دلیل قراردادی را برای فرمت ارسال داده بین گره مرکزی و گره عملگر گذاشتیم و انتهای یک پکت را با کاراکتر # مشخص می‌کنیم، این فرمت به صورت #duty_cycle می‌باشد).
3. بر اساس duty cycle دریافت شده، سیگنال PWM از فرمول زیر محاسبه می‌شود:

$$PWM = \frac{duty\ cycle}{100} * 256$$

4. با استفاده از PWM محاسبه شده و تنظیم سرعت موتور، قطع و وصل شدن و شدت جریان آب کنترل می‌شود. (اگر پین IN1 موتور HIGH و پین IN2 موتور LOW باشد، چرخش موتور ساعتگرد و در غیر این صورت پادساعتگرد می‌شود)

شبیه‌سازی در PlatformIO

در ادامه به توضیحات مربوط به پیاده سازی کد می پردازیم.

ابتدا متغیرها و توابع استفاده شده را تعریف کردیم. در تابع setup یک ارتباط UART Serial برای ارتباط بلوتوثی با گره مرکزی ایجاد می‌کنیم. مقدار baud rate برای ارتباط سریالی بلوتوث را 9600 قرار می‌دهیم. همچنین در این قسمت دو پین 9 و 10 را برای اتصال بین بورد آردوینو و درایور موتور DC از نوع OUTPUT قرار می‌دهیم.

```
void setup(){
 pinMode(MOTOR_PIN1, OUTPUT);
 pinMode(MOTOR_PIN2, OUTPUT);
 Serial.begin(BLUETOOTH_BAUD_RATE);
}
```

در تابع loop که به طور مداوم اجرا می‌شود، ابتدا چک می‌کنیم اگر گره مرکزی داده‌ای فرستاده باشد، آن را از بلوتوث به صورت string دریافت کرده و در بافری ذخیره می‌کنیم. با توجه به اینکه انتهای داده را با دلیمیتر # مشخص کرده بودیم، اگر کاراکتر # در رشته دریافت شده موجود باشد، یعنی داده به طور کامل دریافت شده است. پس از دریافت کامل داده، به کمک تابع parseDutyCycleData که از قبل نوشتیم مقدار duty cycle را به صورت float استخراج می‌کنیم و با

فراخوانی تابع getWateringRate نرخ آبیاری (PWM) را بر اساس duty cycle استخراج شده، محاسبه می‌کنیم. سپس مقدار pwm به دست آمده را روی پین دوم درایور موتور که 10 تعیین کرده بودیم، قرار می‌دهیم و همچنین روی پین اول درایور موتور LOW قرار می‌دهیم تا شدت جریان آب کنترل شود و موتور با توجه pwm دریافت شده در جهت ساعتگرد بچرخد. در نهایت بافر را خالی می‌کنیم.

```
void loop(){
 float pwm, dutyCycle;
 String recievedData = readBluetoothData();
 if (recievedData != ""){
   buffer += recievedData;
   if(buffer.indexOf(BLUETOOTH_DATA_DELIMITER) != -1){
     dutyCycle = parseDutyCycleData(buffer);
     pwm = getWateringRate(dutyCycle);
     Serial.println(pwm);
     digitalWrite(MOTOR_PIN1, LOW);
     analogWrite(MOTOR_PIN2, pwm);
     clearBuffer();
   }
 }
}
```

در ادامه به توضیح توابع کمکی که در این بخش استفاده کردیم، می‌پردازیم:

تابع getWateringRate مقدار PWM را که حاصل تقسیم duty cycle ضرب در بیشینه مقدار سرعت موتور (256) می باشد را حساب می‌کند و آن را برمی‌گرداند.

```
float getWateringRate(float dutyCycle){
 float pwm = (dutyCycle / 100) * MAX_MOTOR_SPEED;
 return pwm;
}
```

تابع parseDutyCycleData از رشته‌ای که در آن duty cycle ذخیره شده، مقدارش را به صورت string استخراج می‌کند و سپس با استفاده از تابع atof آن را به float تبدیل می‌کند و برمی‌گرداند.

```
float parseDutyCycleData(String dutyCycleData){
 dutyCycleData = dutyCycleData.substring(0, dutyCycleData.length() - 1);
```

```
 float dutyCycle = atof(&dutyCycleData[0]);
 return dutyCycle;
}
```

تابع readBluetoothData داده را از بلوتوث از طریق Serial.readString() به صورت string دریافت می‌کند و آن را برمی‌گرداند.

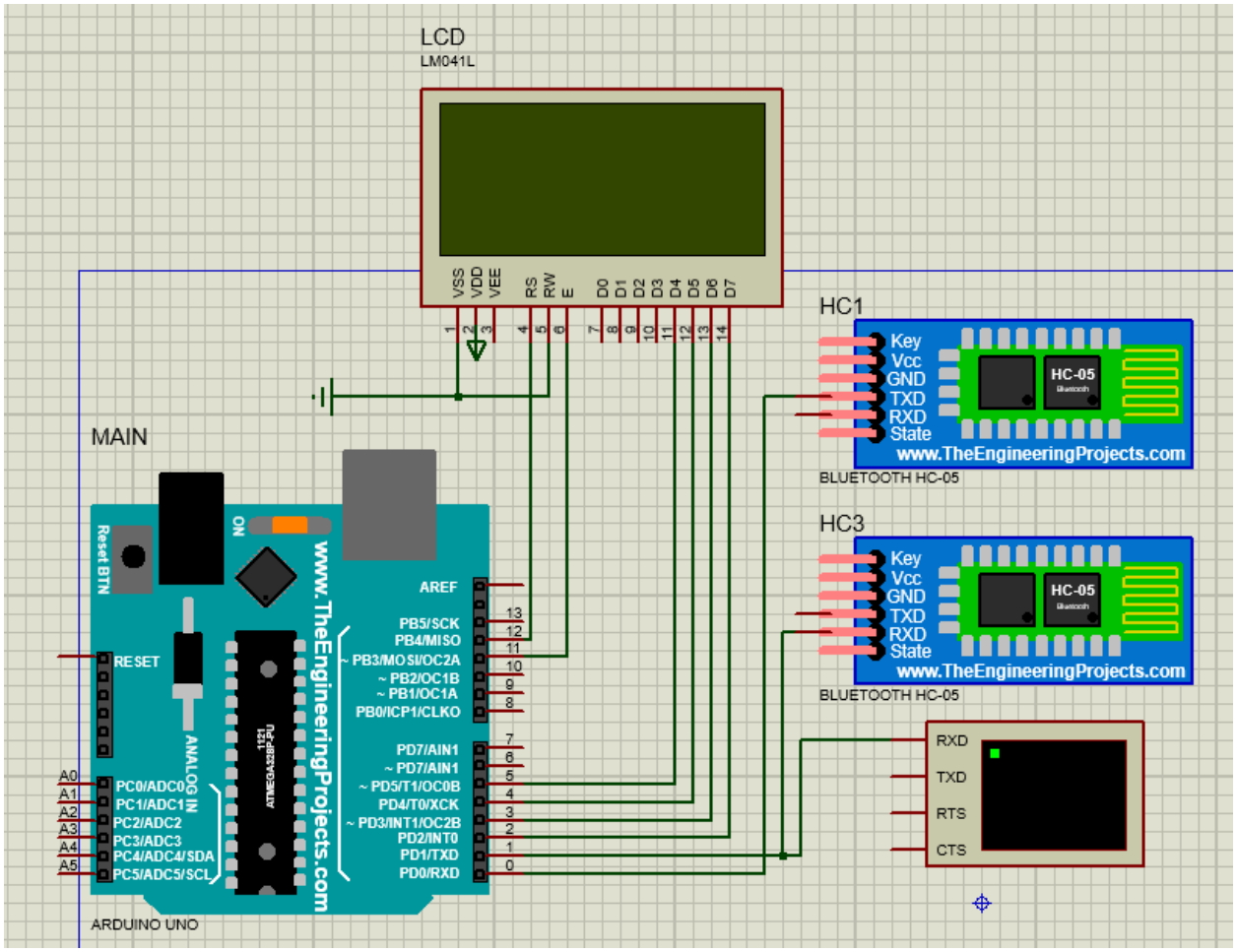
```
String readBluetoothData(){
 String data = Serial.readString();
 return data;
}
```

تابع clearBuffer هم بافر مربوط به اطلاعات که تعریف شده بود را خالی میکند تا سری بعدی داده‌های جدید در آن ذخیره شوند.

```
void clearBuffer(){
 buffer = "";
}
```

گره مرکزی

شبیه‌سازی در Proteus



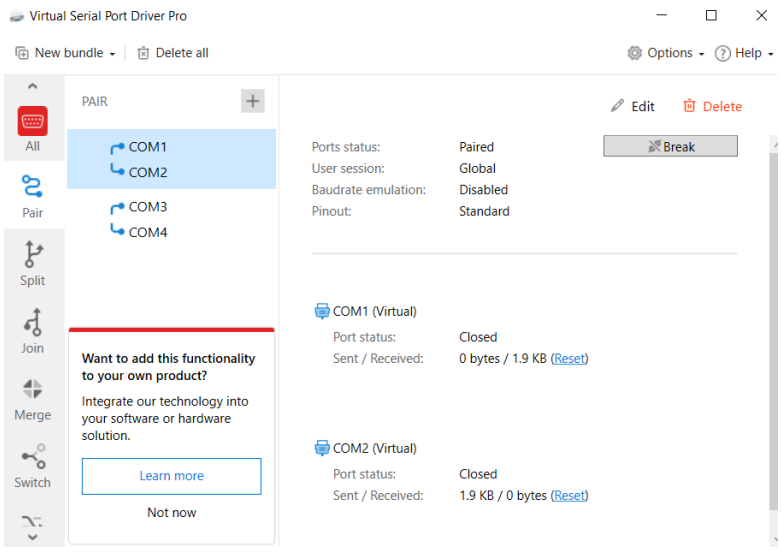
ماژول‌های مورد نیاز برای گره مرکزی یک بورد Arduino Uno، دو HC-05 (بلوتوث) و یک LCD می‌باشد. برای تست و دیباگ کردن از یک virtual terminal استفاده کردیم:

- **بورد Arduino**: برای پردازش اطلاعات و ارسال دستورات مورد نیاز
- **HC-05**: یک HC-05 برای ارتباط بی‌سیم بین گره حسگر و گره مرکزی و یک HC-05 برای ارتباط بین گره مرکزی و گره عملگر
- **LCD**: جهت نمایش اطلاعات دما، رطوبت و نرخ آبیاری

حال عملکرد کلی گره مرکزی را شرح می‌دهیم:

1. اطلاعات مربوط به دما و رطوبت از طریق ارتباط بلوتوثی از گره حسگر دریافت می‌شود.
2. داده‌ی دریافت شده پردازش می‌شود و اندازه دما و رطوبت از آن استخراج می‌شود (با توجه به اینکه از ارتباط سریالی استفاده می‌شود، امکان دارد گره در میانه‌ی ارسال داده، داده را بخواند و داده‌ی معتبری نباشد. به همین دلیل قراردادی را برای فرمت ارسال داده بین گره حسگر و گره مرکزی گذاشتیم و انتهای یک پکت را با کاراکتر '#' مشخص می‌کنیم. فرمت داده‌ی دریافت شده در گره مرکزی به صورت humidity/temperature# است که در بافر ذخیره شده و سپس اندازه دما و رطوبت مطابق فرمت ذکر شده استخراج می‌شوند).
3. بر اساس اندازه دما و رطوبت، برای duty cycle و نرخ آبیاری تصمیم‌گیری می‌شود.
4. اندازه دما، رطوبت و نرخ آبیاری بر روی LCD نمایش داده می‌شود.
5. مقدار duty cycle از طریق ارتباط بلوتوثی به گره عملگر فرستاده می‌شود (همانطور که در مرحله 2 ذکر شد، فرمتی برای ارسال داده بین گره مرکزی و گره عملگر گذاشتیم. این فرمت به صورت duty_cycle# می‌باشد).

* لازم به ذکر است که برای ارتباط بلوتوثی بین HC–05های استفاده شده، از برنامه‌ی Virtual Serial Port Driver استفاده کردیم به این صورت که COM1 و COM2 جهت ارتباط بین گره حسگر و گره مرکزی pair شد و COM3 و COM4 برای ارتباط بین گره مرکزی و گره عملگر pair شد:



شبیه‌سازی در PlatformIO

در ادامه به توضیحات مربوط به پیاده سازی کد می پردازیم.

در تابع setup یک ارتباط UART Serial برای ارتباط بلوتوثی با گره حسگر و عملگر ایجاد می‌کنیم. مقدار baud rate برای ارتباط سریالی بلوتوث را 9600 قرار می‌دهیم. همچنین یک LCD با 20 ستون و 4 سطر ایجاد می‌کنیم.

```
void setup(){
 Serial.begin(BLUETOOTH_BAUD_RATE);
 lcd.begin(20, 4);
}
```

در ادامه، تابع loop را مشاهده می‌کنیم که به طور مداوم پس از تابع setup اجرا می‌شود. در این تابع، داده‌های دریافتی بلوتوث خوانده می‌شوند و در متغیر recievedData ذخیره می‌شود. اگر داده‌های دریافتی یک رشته خالی نباشد، به متغیر buffer اضافه می‌شود. اگر داده‌های دریافتی حاوی جدا کننده داده بلوتوث باشد("/")، تابع splitSensorData فراخوانی می‌شود تا داده‌های حسگر را تقسیم کند و به متغیرهای humidity و temperature اختصاص دهد. سپس، تابع getDutyCycle برای محاسبه duty cycle بر اساس مقدار رطوبت و دما، و تابع showOnLcd برای نمایش رطوبت، دما و نرخ آبیاری بر روی صفحه LCD فراخوانی می‌شوند. در نهایت، تابع sendData برای ارسال مقدار duty cycle به گره عملگر از طریق بلوتوث، و تابع clearBuffer برای پاک کردن متغیر buffer فراخوانی می‌شود. این فرآیند تا زمانی که برنامه متوقف شود در یک حلقه به کار خود ادامه می‌دهد.

```
void loop(){
 String recievedData = readBluetoothData();
 if (recievedData != ""){
   buffer += recievedData;
   if(buffer.indexOf(BLUETOOTH_DATA_SEPERATOR) != -1){
     splitSensorData(buffer);
     float dutyCycle = getDutyCycle();
     showOnLcd(humidity, temperature, dutyCycle);
     sendData(dutyCycle);
     clearBuffer();
   }
 }
}
```

در ادامه به توضیح توابع کمکی که در این بخش استفاده کردیم، می‌پردازیم:

۱) showOnLcd: سه پارامتر temperature، humidity و dutyCycle را به عنوان ورودی می‌گیرد و مقدار دما، رطوبت و نرخ آبیاری را روی LCD نمایش می‌دهد.

```
void showOnLcd(float humidity, float temperature, float dutyCycle){
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("H: " + String(humidity) + " %");
 lcd.setCursor(0, 1);
 lcd.print("T: " + String(temperature) + " C");
 lcd.setCursor(-4, 2);
 lcd.print("R: " + String(getWateringRate(dutyCycle)) + " CC/min");
}
```

۲) getWateringRate: این تابع dutyCycle را به عنوان ورودی می‌گیرد و سپس طبق صورت پروژه، براساس مقدار dutyCycle، نرخ آبیاری را با واحد سی‌سی بر دقیقه برمیگرداند.

```
float getWateringRate(float dutyCycle){
 if (dutyCycle == 10)
   return 10;
 else if (dutyCycle == 20)
   return 15;
 else if (dutyCycle == 25)
   return 20;
 else
   return 0;
}
```

۳) getDutyCycle: این تابع dutyCycle موتور را بر اساس رطوبت و دما تعیین می‌کند. اگر رطوبت بالاتر از 30٪ باشد، dutyCycle روی 0 تنظیم می‌شود. اگر رطوبت بین 20 تا 30 درصد و دما زیر 25 درجه سانتیگراد باشد، dutyCycle روی 0 درصد تنظیم می‌شود. در غیر این صورت، اگر رطوبت بین 10٪ تا 20٪ باشد، dutyCycle روی 20٪ تنظیم می‌شود. در نهایت، همچنین اگر رطوبت زیر 10٪ باشد، چرخه کاری روی 25٪ تنظیم می‌شود.

```
float getDutyCycle(){
 if (humidity > 30)
   return 0;
 else if (humidity > 20){
   if (temperature < 25)
     return 0;
   else
     return 10;
 }
 else if (humidity > 10)
   return 20;
 else
   return 25;
}
```

۴) splitSensorData: این تابع رشته‌ای از داده‌های رطوبت و دما را دریافت می‌کند و مقادیر رطوبت و دما را با استفاده از "/" جدا می‌کند. ابتدا با استفاده از متد indexOf، اندیس کاراکتر "/" را در رشته ورودی پیدا می‌کند، سپس دو رشته جدید که مقادیر رطوبت و دما می‌باشند را استخراج می‌کند. در نهایت، این زیررشته‌ها را با استفاده از تابع atof به اعداد float تبدیل می‌کند و آنها را در متغیرهای سراسری رطوبت و دما برای استفاده بعدی ذخیره می‌کند.

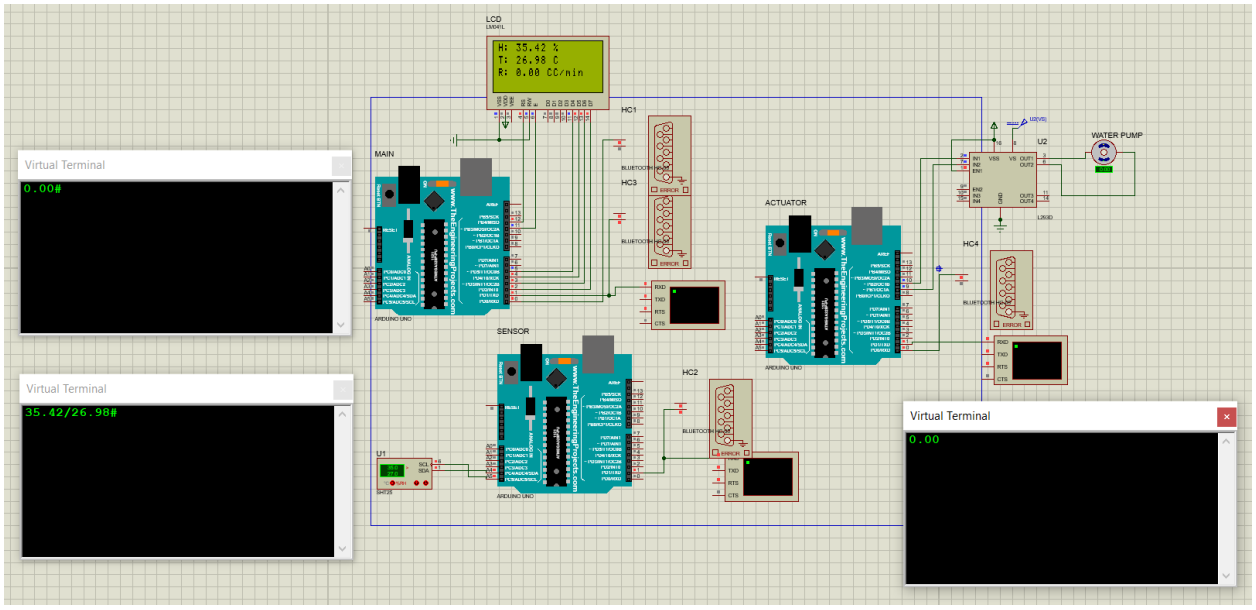
```
void splitSensorData(String humidityTemperatureData){
 int seperatorIdx = humidityTemperatureData.indexOf(BLUETOOTH_DATA_SEPERATOR);
 String humidityStr = humidityTemperatureData.substring(0, seperatorIdx);
 String temperatureStr = humidityTemperatureData.substring(seperatorIdx + 1,
                                        humidityTemperatureData.length() - 1);
 humidity = atof(&humidityStr[0]);
 temperature = atof(&temperatureStr[0]);
}
```

ارزیابی عملکرد

برای ارزیابی عملکرد، 5 حالت مختلف ذکر شده در صورت پروژه را برای نرخ آبیاری در نظر می‌گیریم:

حالت $humidity < 30$:

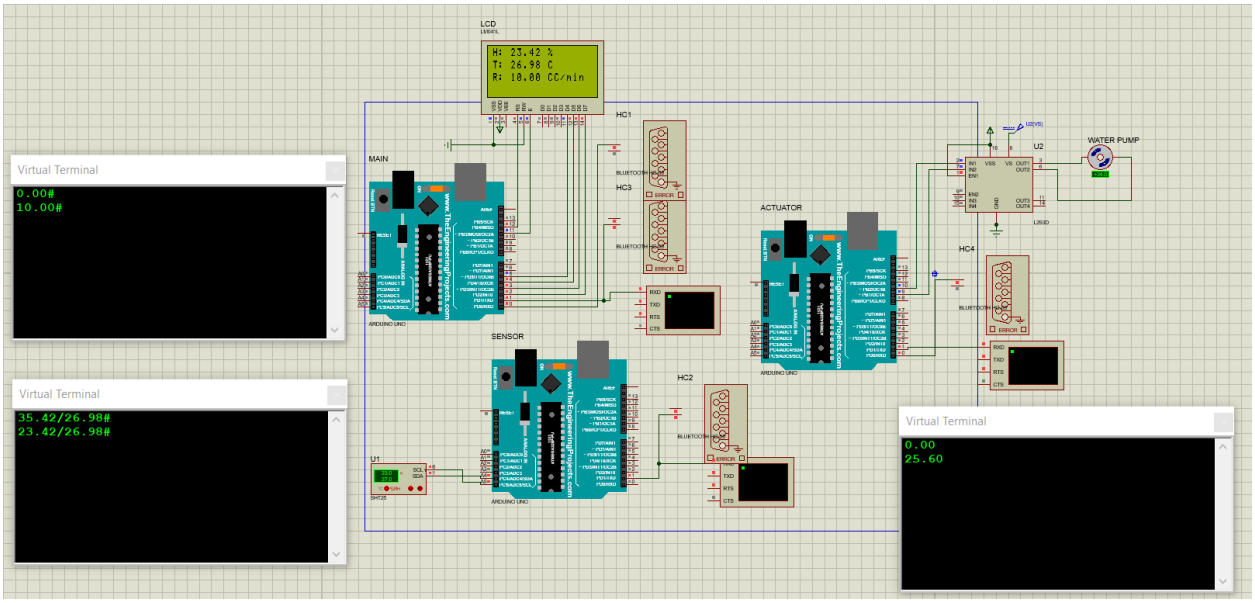
برای این حالت رطوبت 33% و دمای 26 درجه سانتیگراد را در نظر گرفتیم. در این حالت نرخ آبیاری و duty cycle و PWM برابر صفر می‌باشند. همانطور که دیده می‌شود اطلاعات به درستی در LCD نمایش داده شده‌اند:



ترمینال مربوط به گره حسگر داده‌ی فرستاده شده به گره مرکزی را با فرمت مذکور در بخش‌های قبل (اندازه دما و رطوبت) نشان می‌دهد. ترمینال گره مرکزی داده‌ای که برای گره عملگر ارسال کرده را با فرمت مذکور در بخش‌های قبل (اندازه duty cycle) نشان می‌دهد که در اینجا مقدار صفر است. ترمینال مربوط به گره عملگر، PWM را پس از دریافت اندازه duty cycle از گره مرکزی و پردازش آن، نشان می‌دهد که برابر صفر است. در این حالت موتور ثابت است و مطابق حالتی که انتظار می‌رفت، عمل کرد و درست است.

حالت $20 < humidity < 30 \,\&\, 25 < temperature$:

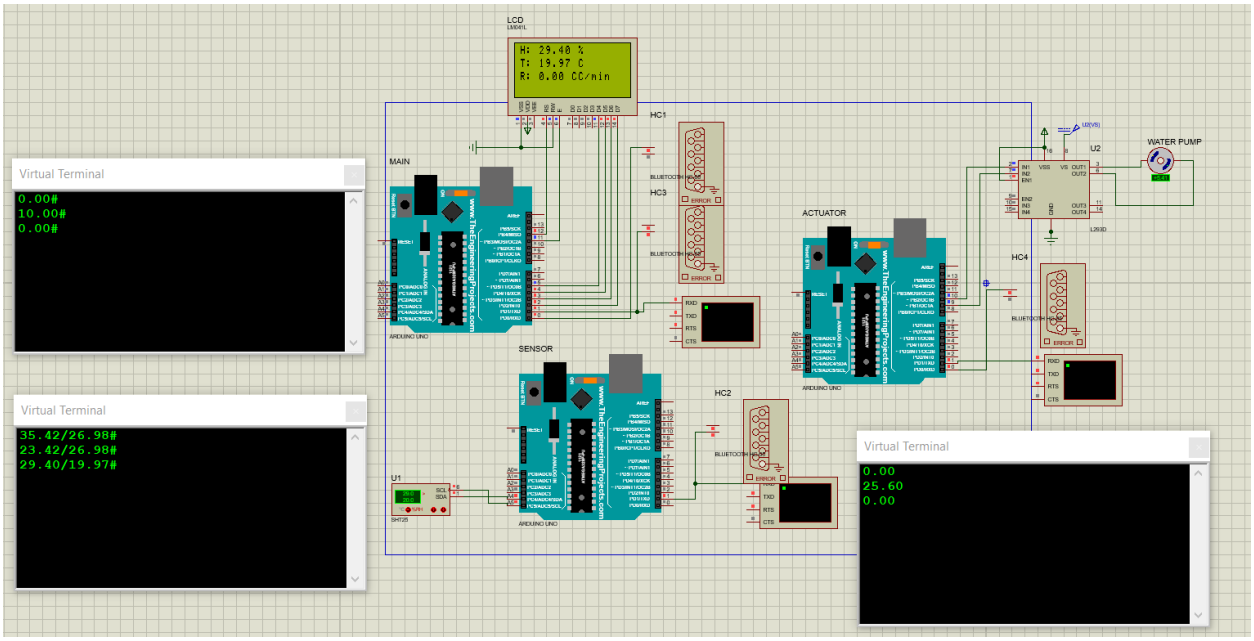
برای این حالت رطوبت 23% و دمای 26 درجه سانتیگراد را در نظر گرفتیم. در این حالت نرخ آبیاری و duty cycle برابر برابر 10 و PWM برابر 25.6 می‌باشند. همانطور که دیده می‌شود اطلاعات به درستی در LCD نمایش داده شده‌اند:



ترمینال مربوط به گره حسگر داده‌ی فرستاده شده به گره مرکزی را با فرمت مذکور در بخش‌های قبل (اندازه دما و رطوبت) نشان می‌دهد. ترمینال گره مرکزی داده‌ای که برای گره عملگر ارسال کرده را با فرمت مذکور در بخش‌های قبل (اندازه duty cycle) نشان می‌دهد که در اینجا مقدار 10 است. ترمینال مربوط به گره عملگر، PWM را پس از دریافت اندازه duty cycle از گره مرکزی و پردازش آن، نشان می‌دهد که برابر صفر است. در این حالت موتور به حرکت درمی‌آید و سرعت می‌گیرد و مطابق حالتی که انتظار می‌رفت عمل کرد و درست است.

حالت $20 < humidity < 30 \,\&\, temperature < 25$:

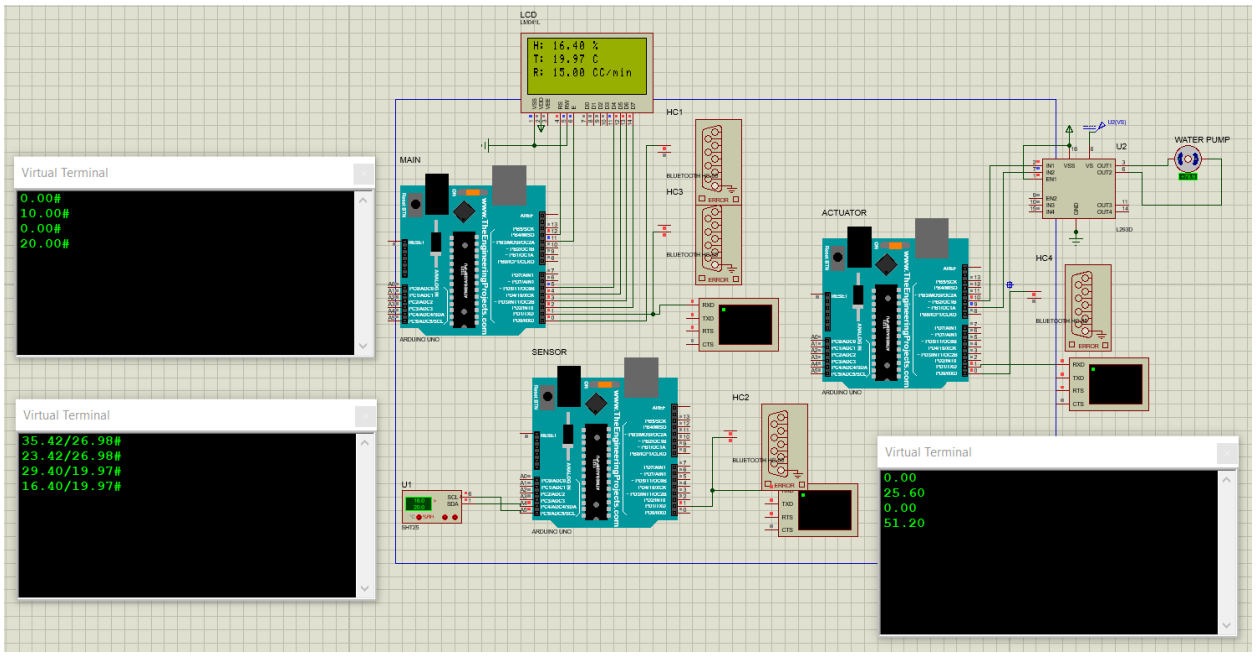
برای این حالت رطوبت 29% و دمای 19 درجه سانتیگراد را در نظر گرفتیم. در این حالت نرخ آبیاری و duty cycle و PWM برابر صفر می‌باشند. همانطور که دیده می‌شود اطلاعات به درستی در LCD نمایش داده شده‌اند:



ترمینال مربوط به گره حسگر داده‌ی فرستاده شده به گره مرکزی را با فرمت مذکور در بخش‌های قبل (اندازه دما و رطوبت) نشان می‌دهد. ترمینال گره مرکزی داده‌ای که برای گره عملگر ارسال کرده را با فرمت مذکور در بخش‌های قبل (اندازه duty cycle) نشان می‌دهد که در اینجا مقدار صفر است. ترمینال مربوط به گره عملگر، PWM را پس از دریافت اندازه duty cycle از گره مرکزی و پردازش آن، نشان می‌دهد که برابر صفر است. در این حالت سرعت موتور کاهش پیدا کرده و پس از مدتی می‌ایستد و مطابق حالتی که انتظار می‌رفت عمل کرد و درست است.

حالت $10 < humidity < 20$:

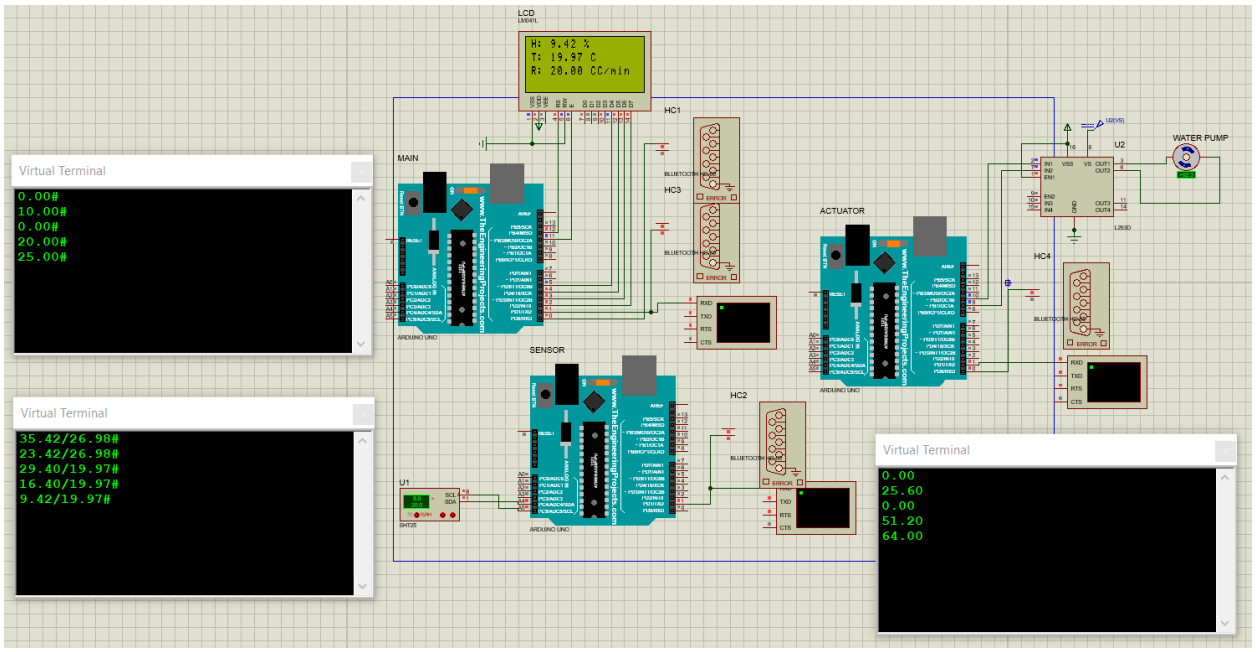
برای این حالت رطوبت 16% و دمای 19 درجه سانتیگراد را در نظر گرفتیم. در این حالت نرخ آبیاری برابر 15CC/min و duty cycle برابر 20 و PWM برابر 51.20 می‌باشند. همانطور که دیده می‌شود اطلاعات به درستی در LCD نمایش داده شده‌اند:



ترمینال مربوط به گره حسگر داده‌ی فرستاده شده به گره مرکزی را با فرمت مذکور در بخش‌های قبل (اندازه دما و رطوبت) نشان می‌دهد. ترمینال گره مرکزی داده‌ای که برای گره عملگر ارسال کرده را با فرمت مذکور در بخش‌های قبل (اندازه duty cycle) نشان می‌دهد که در اینجا مقدار 20 است. ترمینال مربوط به گره عملگر، PWM را پس از دریافت اندازه duty cycle از گره مرکزی و پردازش آن، نشان می‌دهد که برابر 51.20 است. در این حالت سرعت موتور افزایش پیدا کرده و مطابق حالتی که انتظار می‌رفت عمل کرد و درست است.

حالت $humidity < 10$:

برای این حالت رطوبت 9% و دمای 19 درجه سانتیگراد را در نظر گرفتیم. در این حالت نرخ آبیاری برابر 20CC/min و duty cycle برابر 25 و PWM برابر 64 می‌باشند. همانطور که دیده می‌شود اطلاعات به درستی در LCD نمایش داده شده‌اند:



ترمینال مربوط به گره حسگر داده‌ی فرستاده شده به گره مرکزی را با فرمت مذکور در بخش‌های قبل (اندازه دما و رطوبت) نشان می‌دهد. ترمینال گره مرکزی داده‌ای که برای گره عملگر ارسال کرده را با فرمت مذکور در بخش‌های قبل (اندازه duty cycle) نشان می‌دهد که در اینجا مقدار 25 است. ترمینال مربوط به گره عملگر، PWM را پس از دریافت اندازه duty cycle از گره مرکزی و پردازش آن، نشان می‌دهد که برابر 64.00 است. در این حالت سرعت موتور از حالت قبل هم بیشتر می‌شود و مطابق حالتی که انتظار می‌رفت عمل کرد و درست است.

پرسش‌ها

سوال ۱)

در مورد بلوتوث، از چه فرکانسی برای ارتباط بی سیم استفاده می‌شود؟ در این تمرین که دو ارتباط بلوتوثی برای اتصال 3 گره لازم است، چگونه از تداخل داده های ارسالی دستگاه ها جلوگیری می‌شود؟ نیازی به ارائه جزئیات پروتکل ارتباطی بلوتوث نیست. بیان مفاهیم کلی کافی است.

فناوری بلوتوث از امواج رادیویی با فرکانس 2.4 گیگاهرتز برای ارتباطات بی‌سیم استفاده می‌کند. برای جلوگیری از تداخل بین دستگاه های ارسال کننده در یک شبکه بلوتوث، از فناوری frequency hopping استفاده می‌شود. این بدان معنی است که سیگنال فقط برای مدت کوتاهی روی یک فرکانس مشخص باقی می‌ماند و اگر تداخل مانع از برقراری ارتباط در این کانال شود، سیگنال به فرکانس دیگری پرش می‌کند. بنابراین می‌توان با استفاده از فناوری طیف گسترده پرش فرکانس در شبکه های بلوتوث از تداخل بین دستگاه‌های ارسال کننده جلوگیری کرد.

سوال ۲)

در باس I2C، چگونه تضمین می‌شود که داده‌های ارسالی گره‌های روی باس با هم تداخل نمی‌کند؟

برای تضمین اینکه داده‌های ارسالی گره‌های روی باس با هم تداخل نکنند، دو راه‌حل وجود دارد:

۱) راه‌حل نرم‌افزاری: استفاده از یک کتابخانه جایگزین برای مدیریت I2C که از I2C بر روی هر پین I/O پشتیبانی می‌کند.

کتابخانه استاندارد Wire که با محیط توسعه Arduino IDE تعریف شده است، به شما امکان ارتباط با استفاده از اتصالات I2C در حالت master و یا slave را می‌دهد. این کتابخانه از پین‌های مخصوص I2C برای ارتباط با دستگاه‌های I2C استفاده می‌کند، برای مثال در Arduino Uno، پین A4 به عنوان SDA و پین A5 به عنوان SCL استفاده می‌شود.

با تمام ویژگی‌های خوب این کتابخانه، محدودیت‌هایی نیز دارد؛ به این صورت که کتابخانه Wire به شما اجازه نمی‌دهد تا چندین اتصال I2C را داشته باشید، فقط یک نمونه از آن را می‌توانید فراخوانی کنید و نمی‌توانید ترافیک I2C خود را به پین‌های دیگر تغییر دهید.

۲) راه‌حل سخت‌افزاری: استفاده از یک I2C multiplexer

راهکارهای سخت‌افزاری نسبت به راهکارهای نرم‌افزاری گران‌تر هستند، اما می‌توانند چندین مزیت داشته باشند.

- می‌توانید از کتابخانه استاندارد Wire و سایر کتابخانه‌های خود استفاده کنید.
- نیازی به اتصال پین‌های اضافی در Arduino خود، به جز SDA و SCL نخواهید داشت.

دستگاهی که ما به آن نیاز داریم، یک مالتی پلکسر است که در واقع یک دستگاه سوئیچینگ الکترونیکی است. به بیان دقیق‌تر، یک مالتی پلکسر I2C که به چندین باس خارجی I2C متصل می‌شود. زمانی که می‌خواهیم با یک دستگاه slave صحبت کنیم، به باسی که دستگاه slave در آن قرار دارد سوئیچ می‌کنیم و به آن ادرس می‌دهیم.

هر bus البته باید از قوانین مشابه هر I2C bus پیروی کند ، بنابراین Slaveها نمی‌توانند از آدرس‌های استفاده شده توسطSlaveهای دیگر در همان bus استفاده کنند (یعنی در هر bus جدا نباید بین آدرس‌ها تداخل وجود داشته باشد). پس تا زمانی که Slaveهای دارای Slave Address یکسان را در bus های مختلف نگه دارید ، خوب خواهد بود و مشکلی پیش نخواهد آمد.

منبع: https://dronebotworkshop.com/multiple-i2c-bus/

سوال ۳)

در مورد موتورهای الکتریکی، تفاوت و مشخصات سه نوع موتور پرکاربرد Stepper Motor ،DC Motor و Servo Motor را بیان نمایید. تشریح ساختار الکتریکی این موتورها لازم نیست و فقط کاربرد و قابلیت‌های هر موتور و برتری آن بر دیگران را بیان کنید.

:Stepper Motor

این نوع موتورها عمدتا در کاربردهایی استفاده می‌شوند که کنترل دقیق موقعیت چرخشی مورد نیاز است.

این نوع موتورها دارای گشتاور بالایی در سرعت های پایین هستند که آن‌ها را برای کاربردهایی که نیازمند دقت بالا و سرعت پایین است، مناسب می‌کند.

آن‌ها همچنین می‌توانند موقعیت خود را بدون برق حفظ کنند، که آن‌ها را برای برنامه‌هایی که نیاز به قفل کردن در محل دارند، مناسب می‌کند.

در مقایسه با سایر انواع موتور، موتورهای stepper به دلیل مقاومت داخلی بالا، گران‌تر هستند و راندمان پایین‌تری دارند.

:DC Motor

موتورهای DC در طیف گسترده‌ای از کاربردها استفاده می‌شوند، از اسباب بازی‌های کوچک گرفته تا ماشین آلات صنعتی بزرگ.

آن‌ها برای کاربردهایی که نیاز به گشتاور بالا و کنترل سرعت متغیر دارند، مناسب هستند.

موتورهای DC نسبتا ساده و ارزان هستند و آن‌ها را به انتخابی محبوب برای بسیاری از کاربردها تبدیل می‌کند.

همچنین کارآمد هستند و می توانند برای مدت طولانی بدون گرم شدن بیش از حد، کار کنند.

:Servo Motor

موتورهای Servo در کاربردهایی استفاده می‌شوند که در آن‌ها موقعیت‌یابی دقیق مورد نیاز است، مانند رباتیک و ماشین‌های CNC.

این موتورها نسبت گشتاور به اندازه بالایی دارند و برای کاربردهایی که دقت و قدرت بالایی در فضای کوچک مورد نیاز است، مناسب هستند.

آن‌ها می توانند به سرعت به تغییرات در موقعیت و سرعت پاسخ دهند. این موضوع آن‌ها را برای کاربردهای پویا مناسب می‌کند.

در مقایسه با انواع دیگر موتورها، گران‌تر هستند و به سیستم‌های کنترل پیچیده‌تری نیاز دارند.

به طور خلاصه، هر یک از این سه نوع موتور، ویژگی‌ها و کاربردهای منحصر به فردی دارند. موتورهای Stepper برای موقعیت‌یابی دقیق، موتورهای DC برای کنترل سرعت متغیر و موتورهای Servo برای کاربردهای دقیق و دینامیکی ایده آل هستند. انتخاب نوع موتور مناسب برای یک کاربرد معین به نیازهای خاص آن کاربرد مانند دقت، سرعت، قدرت و اندازه مورد نیاز بستگی دارد.