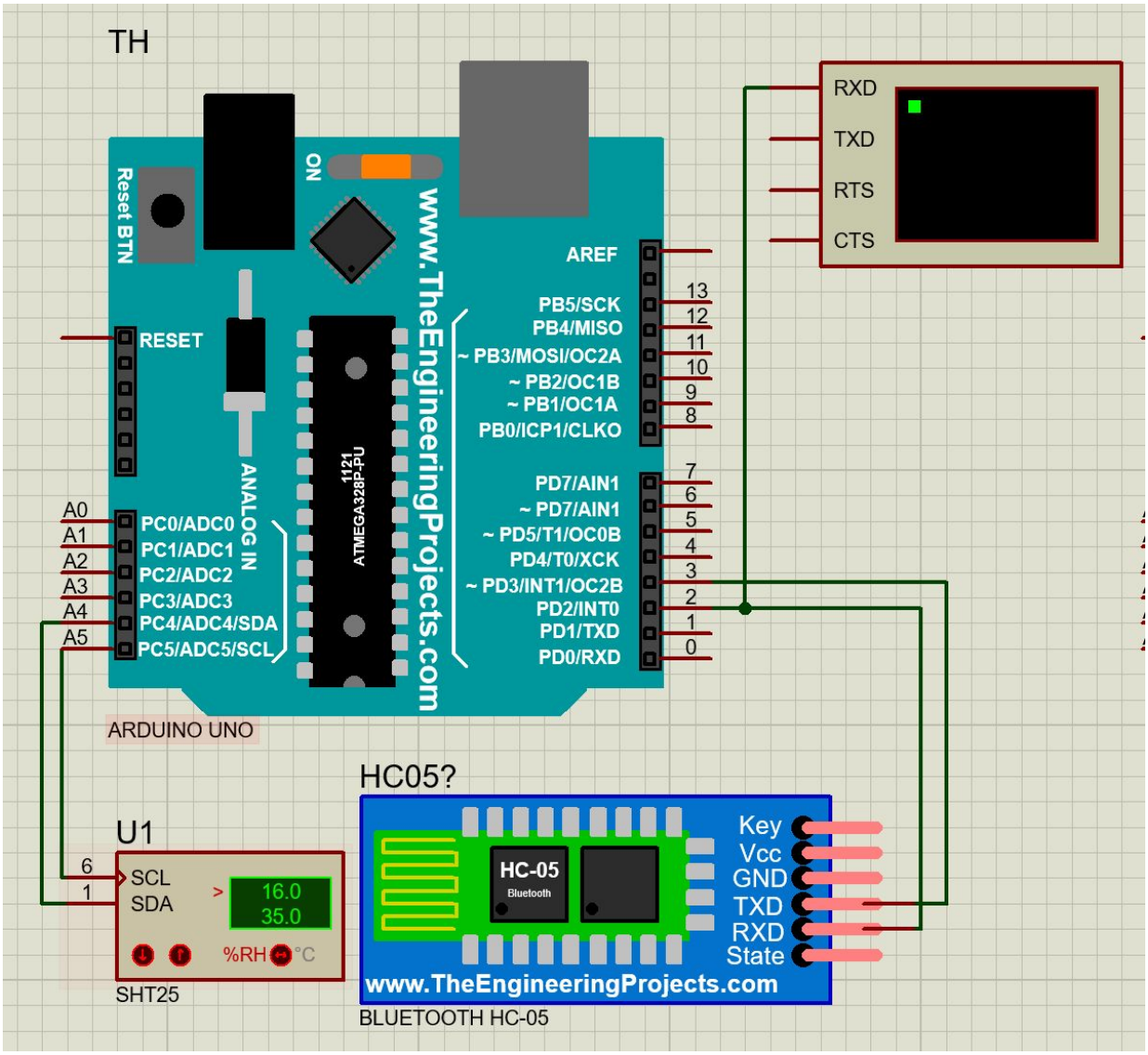
گزارش کار تمرین دوم

در این تمرین، به پیاده‌سازی یک مدار به منظور کنترل میزان رطوبت و آب رسانی به یک گلدان می‌پردازیم و با توجه به میزان رطوبت و دمای محیط به کنترل یک موتور DC که نقش شیر آب را ایفا می‌کند می‌پردازیم. این مدار از دو نود اصلی تشکیل شده است که به هر کدام در زیر می‌پردازیم:

برد TH

هدف اصلی این برد این است که اطلاعات مورد نظر را از سنسورهای دما و رطوبت دریافت می‌کند و از طریق بلوتوث آن را به برد اصلی (همان برد Main) ارسال می‌دارد. تصویر مرتبط به این بخش مدار را در زیر می‌بینیم:



این مدار از سه ماژول اصلی SHT25، HC05 و آردوینو UNO تشکیل شده است و علاوه بر آن‌ها یک ترمینال مجازی نیز برای نمایش اطلاعاتی که از برد به ماژول ارسال می‌شود و دیباگ استفاده می‌شود. هر یک از بخش‌های مورد نظر را در زیر توضیح می‌دهیم.

ماژول SHT25

این ماژول، یک ماژول با دقت بالای اندازه‌گیری دما و رطوبت است که برای انتقال داده از سیگنال‌ها را به فرمت I2C ارسال می‌کند. در این ماژول با استفاده از فرمت I2C که با دو خط SDA و SCL که به ترتیب مرتبط به ارسال دیتا و کلاک می‌باشد ارتباط برقرار می‌کند، دیتا را به ماژول اصلی ارسال می‌دارد.

ماژول HC-05

این ماژول برای ارسال داده از طریق بلوتوث استفاده می‌شود و با دریافت داده از ماژول اصلی و همچنین با اتصال از طریق بلوتوث به ماژول مشابه برد اصلی دیتا را از برد TH به برد اصلی ارسال می‌کند.برای انجام این کار دیتا از طریق پورت شماره ۲ برد به پورت RXD ماژول ارسال می‌شود و همچنین پورت TXD ماژول نیز به پورت شماره ۳ برد متصل شده است که در صورت لزوم ارسال داده از ماژول به برد اصلی از آن بتوان استفاده نمود. همچنین مورد دیگری نیز که در این بخش دیده می‌شود اتصال ماژول ترمینال به پورت شماره ۲ است که به منظور نمایش داده استفاده می‌شود.

برد TH

برد اصلی در این بخش است که وظیفه‌ی دریافت، تبدیل و در نهایت ارسال داده‌های دریافت شده از سنسور را دارد. از نظر اتصالات، این ماژول به SHT25 از طریق پورت‌های A4 و A5 متصل است و از طرفی هم با استفاده از پورت‌های ۲ و ۳ به ماژول HC-05 وصل شده است. برای بررسی منطق برنامه به بررسی کد می‌پردازیم.

تعاریف و مقادیر اولیه

ابتدا برخی از مواردی را که برای ادامه‌ی کار بدان‌ها نیاز داریم، تعریف می‌کنیم. جدای از کتابخانه‌ی Arduion.h که برای توابع اصلی loop و setup و تعیین پین‌ها و ... نیاز است، برای اتصال به سنسور دما و رطوبت باید از کتابخانه‌ی Wire.h استفاده کنیم و برای اتصال به ماژول بلوتوث نیز از SoftwareSerial.h استفاده می‌کنیم.

```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>


#define SHT25_ADDR 0x40
#define HUMIDITY_CMD 0xF5
#define TEMPERATURE_CMD 0xF3

#define BLUETOOTH_TRANSMITION_START_CHAR '@'
#define BLUETOOTH_TRANSMITION_END_CHAR '#'
#define BLUETOOTH_RX 3
#define BLUETOOTH_TX 2
#define BLUETOOTH_BAUD_RATE 9600


SoftwareSerial bluetooth(BLUETOOTH_RX ,BLUETOOTH_TX);
```

پس از آن مقادیری را که برای ارتباط I2C با SHT25 نیاز است تعریف می‌کنیم. اولین مورد آدرس I2C سنسور است که برابر با مقدار ۶۴ می‌باشد و دو مورد بعدی نیز دستوراتی که برای دریافت دما و رطوبت باید به سنسور ارسال شود نشان می‌دهند.

مقدارهای تعریف شده در قسمت بعد برای کار با بلوتوث استفاده می‌شود. دو کاراکتر @ و # برای نمایش ابتدا و انتهای یک پیام استفاده می‌شود تا مشخص شود که اعداد ارسال شده‌ی یک پیام کدامند و پیام‌ها از هم دیگر قابل تمایز باشد. در نهایت نیز Baud Rate بلوتوث را مشخص می‌کنیم که باید با مقدار مورد نظر در مدار یکسان باشد. در نهایت نیز یک نمونه از کلاس SoftwareSerial در کد ایجاد می‌کنیم که از این طریق بتوانیم با ماژول بلوتوث ارتباط برقرار کنیم. همانگونه هم که دیده می‌شود پورت‌های ۲ و ۳ را نیز به ترتیب به عنوان خروجی و ورودی ماژول اصلی برای ارتباط با بلوتوث تعیین می‌کنیم.

تابع setup

```
19  void setup() {
20    Wire.begin();
21    pinMode(BLUETOOTH_RX,INPUT);
22    pinMode(BLUETOOTH_TX,OUTPUT);
23    bluetooth.begin(BLUETOOTH_BAUD_RATE);
24  }
```

در این تابع ابتدا اتصال بین برد و SHT25 را با استفاده از تابع begin برقرار می‌کنیم و سپس نوع پین‌هایی که در اتصال با HC-05 هستند تعیین می‌کنیم. در نهایت نیز ماژول بلوتوث را راه‌اندازی می‌کنیم.

توابع مرتبط با دریافت داده از SHT-25

این توابع شامل موارد زیر است:

```
26  void sendCMD(int cmd){
27    Wire.beginTransmission(SHT25_ADDR);
28    Wire.write(cmd);
29    Wire.endTransmission();
30  }
31
32  float receiveData(unsigned int data[2]){
33    Wire.requestFrom(SHT25_ADDR,2);
34    if (Wire.available() == 2) {
35      data[0] = Wire.read();
36      data[1] = Wire.read();
37    }
38  }
39
40  float getHumidity(){
41    unsigned int data[2];
42    sendCMD(HUMIDITY_CMD);
43    delay(500);
44    receiveData(data);
45    return (((data[0] * 256.0 + data[1]) * 125.0) / 65536.0) - 6;
46  }
47
48  float getCTemperature(){
49    unsigned int data[2];
50    sendCMD(TEMPERATURE_CMD);
51    delay(500);
52    receiveData(data);
53    return (((data[0] * 256.0 + data[1]) * 175.72) / 65536.0) - 46.85;
54  }
```

نخستین تابع sendCMD نام دارد که در آن دستورات لازم را برای دریافت داده به سنسور ارسال می‌کنیم. برای ارسال این دستور در خلال یک Transmission دستور مورد نظر را که در ورودی تابع دریافت می‌شود برای سنسور ارسال می‌داریم. پس از ارسال دستور به مدت معینی منتظر می‌مانیم و سپس در صورت بازگشت پاسخ مقدار داده را از سنسور دریافت می‌کنیم. این کار در تابع receiveData انجام می‌شود.

در نهایت نیز با توجه به این که در حال خواندن دما و یا رطوبت هستیم با توجه به رابطه‌ی مورد نیاز که در دیتاشیت[1] سنسور آمده است، مقادیر خوانده شده را به مقدار واقعی این موارد تبدیل می‌کنیم. این کار نیز در توابع getCTemprature و همچنین getHumidity انجام می‌دهیم. در این تابع با استفاده از دو تابع نخست به ارسال دستور و دریافت نتیجه، و در نهایت تبدیل مقدار می‌پردازیم.

تابع ارسال داده به بلوتوث

بدنه‌ی این تابع در زیر دیده می‌شود:

```
56  void sendWithBluetooth(float temperature,float humidity){
57      bluetooth.print(BLUETOOTH_TRANSMITION_START_CHAR);
58      bluetooth.println(temperature);
59      bluetooth.print(humidity);
60      bluetooth.print(BLUETOOTH_TRANSMITION_END_CHAR);
61    }
```

همانگونه که در ابتدای این بخش نیز اشاره شد، برای ارسال یک پیام به بلوتوث از دو کاراکتر برای تعیین ابتدا و انتهای پیام استفاده می‌کنیم. برای جدا کردن دو عدد از یک دیگر نیز از یک کاراکتر \n که به صورت ضمنی در تابع println استفاده می‌شود بهره می‌بریم و محتوای پیام‌ها را برای بلوتوث ارسال می‌کنیم.

تابع loop

در نهایت نیز به تابع loop می‌رسیم که به صورت مداوم عملیات زیر را انجام می‌دهد:

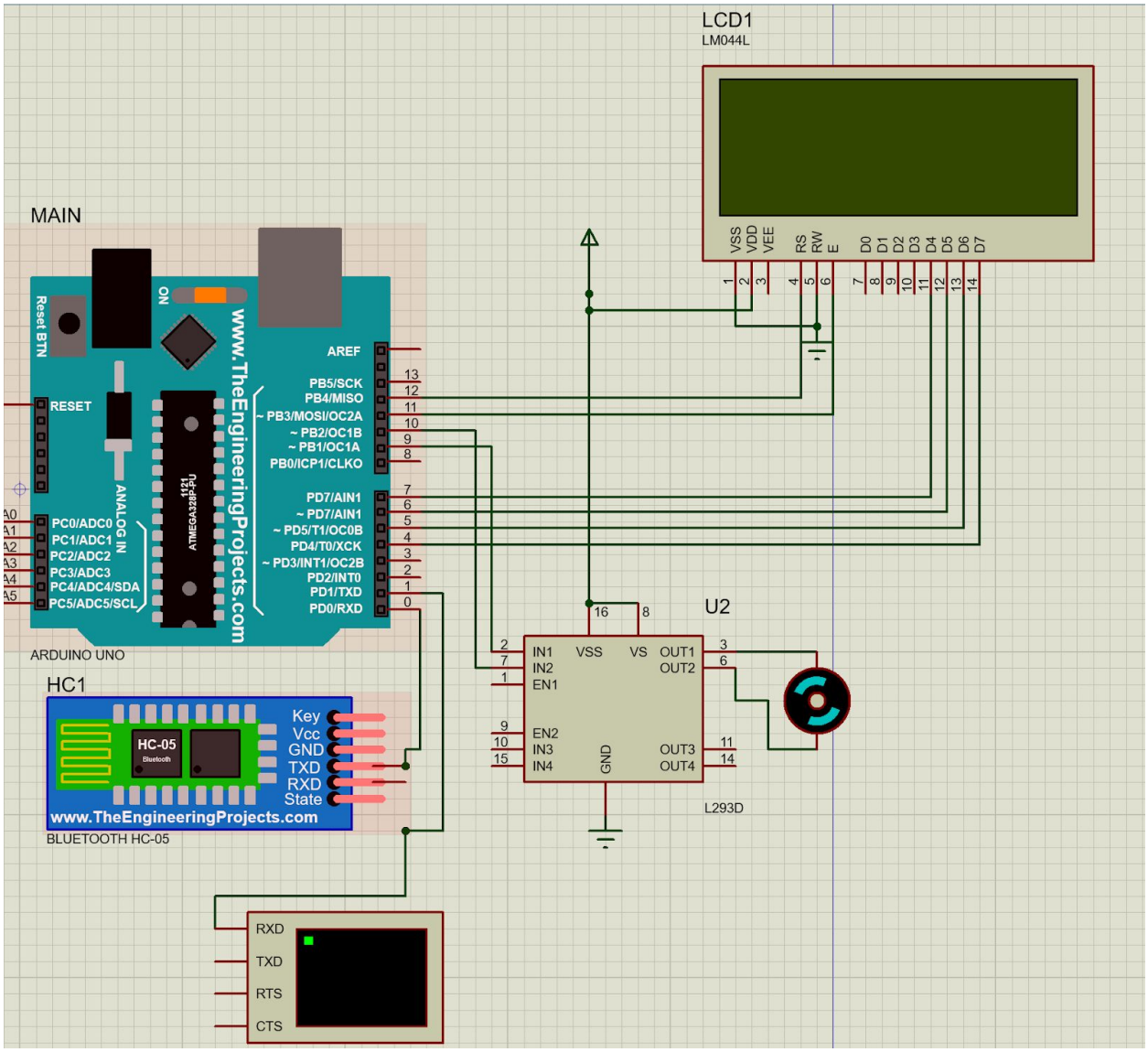
- دریافت مقادیر رطوبت و دما
- ارسال به ماژول بلوتوث
- به مدت ۴ ثانیه منتظر ماندن (دقت شود که در صورت پروژه گفته شده هر ۵ ثانیه یکبار وضعیت به روز شود اما با توجه به این که برای دریافت مقادیر دما و رطوبت از سنسور برای هر کدام به مدت ۵۰۰ میلی‌ثانیه انتظار می‌کشیم، میزان انتظار نهایی ۵ ثانیه می‌شود.)

[1] https://cdn.sos.sk/productdata/bf/f4/e1c6ad4c/sht-25.pdf

```
63 void loop() {
64     float humidity = getHumidity();
65     float temperature = getCTemperature();
66     sendWithBluetooth(temperature, humidity);
67     delay(4000);
68   }
```

برد Main

این برد وظیفه دریافت اطلاعات از برد TH و پردازش آن و کنترل مقدار سرعت چرخش موتور و نمایش داده‌های دریافتی و تصمیمات اخذ شده بر روی صفحه نمایش را دارد. شمای کلی آن در تصویر زیر قابل مشاهده است:



اجزای مدار

در ادامه درباره هر یک از اجزای استفاده شده و شیوه ارتباط آن‌ها با برد آردوینو توضیحاتی ارائه می‌شود.

ماژول بلوتوث HC-05

از این ماژول برای دریافت اطلاعات ارسالی از برد TH استفاده می‌شود. دقت کنید برد Main نیازی به ارسال اطلاعات به برد TH ندارد بنابراین در شکل مدار پورت TX از برد به RX از ماژول HC-05 وصل نشده است و

فقط RX از برد به TX از ماژول HC-05 وصل گردیده است. درباره کد و پروتکل ارسال داده در ادامه توضیح داده خواهد شد.

ماژول L293D و موتور DC

همانند پروژه قبل از این ماژول L293D به عنوان درایور موتور استفاده شده است. برای این منظور پورت vss , vs از این ماژول به منبع تغذیه و همچنین GND نیز به زمین وصل شده است. از طرف دیگر دو ورودی از برد آردوینو برای کنترل سرعت موتور به ماژول L293D وارد شده و از خرجی مناسب نیز به موتور DC وصل شده است. شرح این که چگونه سرعت موتور کنترل می‌شود در ادامه توضیح داده خواهد شد.

ماژول LCD LM044L

از این ماژول برای نمایش داده‌های دریافتی از برد TH و همچنین تصمیماتی که بر اساس شرایط مختلف اتخاذ می‌شود نمایش داده می‌شود. برای این که این ماژول را به برد آردوینو وصل کنیم باید پین‌های مشخصی از LCD را به پین‌های مشخصی از آردوینو که رابط کار با آن روی آردوینو که در این پروژه LiquidCrystal است وصل کنیم. این اتصالات مطابق شکل فوق است و جزئیات مربوط به نمایش اطلاعات بر روی آن در ادامه توضیح داده خواهد شد.

کد مربوط به برد Main

در ادامه اجرای مختلف کد و شیوه هندل کردن ارتباطات بین آن‌ها را شرح می‌دهیم.

متغیرهای وضعیت

src > C++ main.cpp

```cpp
#include <Arduino.h>
#include <LiquidCrystal.h>

#define BLUETOOTH_TRANSMITION_START_CHAR '@'
#define BLUETOOTH_TRANSMITION_END_CHAR '#'
#define BLUETOOTH_BAUD_RATE 9600
#define PWM_MAX 64
#define PWM_MIN 0

String decision;
int pwm_counter = 0;
int pwm_velocity = 64;
int dc_pinA = 9;
int dc_pinB = 10;
float globalHumidity = 0;
float globalTemperature = 0;
int rs = 12, en = 11;
int d4 = 7,d5 = 6,d6 = 5,d7 = 4;
bool stateUpdated = false;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

برای نگهداری وضعیت در برد Main به تعدادی متغیر نیاز داریم که در تصویر فوق قابل مشاهده هستند:

- متغیر decision: این متغیر تصمیم اتخاذ شده توسط قسمت لاجیک کد را در خود ذخیره می‌کند و برای چاپ روی LCD مورد استفاده قرار می‌گیرد.
- متغیر pwm_counter: این متغیر برای کنترل وضعیت PWM که سرعت موتور را کنترل می‌کند استفاده می‌شود و صرفا یک شمارنده است با هر بار اجرای loop مقدار آن یکی زیاد می‌شود.
- متغیر pwm_velocity: این متغیر سرعتی که موتور با آن می‌چرخد را مشخص می‌کند. این متغیر با توجه مقدار ارسالی از برد TH تنظیم می‌گردد.

- متغیرهای dc_pinA و dc_pinB: این دو متغیر همواره ثابت هستند و مشخص کننده پین‌هایی است که به درایور موتور وصل شده است.
- متغیر globalHumidity: این متغیر آخرین مقدار **صحیحی** که به عنوان رطوبت از برد TH دریافت کرده است را در خود نگه داری می‌کند و بسیاری از تصمیمات بر حسب مقدار این متغیر است.
- متغیر globalTemperature: این متغیر آخرین مقدار **صحیحی** که به عنوان دما از برد TH دریافت کرده است را در خود نگه داری می‌کند و بسیاری از تصمیمات بر حسب مقدار این متغیر است.
- متغیرهای en,rs,d4,d5,d6,d7: این متغیرها همواره ثابت هستند و پین‌هایی که به LCD وصل شده اند را مشخص می‌کنند.
- متغیر lcd: این متغیر که از نوع LiquidCrystal است از کتابخانه LiquidCrystal گرفته شده است و برای کنترل کردن LCD در مدار از آن استفاده می‌شود.
- متغیر stateUpdated: این متغیر برای این در نظر گرفته شده است که فقط درصورتی که به طور موفقیت آمیز داده را از بلوتوث دریافت کرد LCD اپدیت شود.

مقدار دهی اولیه

```
23  void setup() {
24    // put your setup code here, to run once:
25    lcd.begin(20,4);
26    pinMode(dc_pinA,OUTPUT);
27    pinMode(dc_pinB,OUTPUT);
28    pinMode(0, INPUT);
29    Serial.begin(9600);
30  }
```

در این تابع که خودکار صدا زده می‌شود برخی از متغیرها مقدار دهی اولیه می‌شوند. ابتدا ابعاد LCD که شامل ۴ ردیف و ۲۰ ستون است مقدار دهی شده و ارتباط با آن آغاز می‌گردد. سپس پین‌های dc_pinA و dc_pinB از نوع

خروجی تعیین می‌شوند. در پایان نیز چون برای ارتباط با بلوتوث از پورت‌های دیفالت خود آردوینو استفاده کردیم، کافی است که ارتباط سریال را با استفاده از Serial.begin بر روی BAUD RATE توافق شده آغاز کنیم.

منطق برنامه

```
void updateVelocity(){
  if (globalHumidity > 50){
    pwm_velocity = 0;
    decision = "0/100DC H>50";
  }
  else if (globalHumidity < 20){
    pwm_velocity = PWM_MAX/4;
    decision = "25/100DC H<20";
  }
  else if (globalHumidity >= 20 && globalHumidity <= 50){
    if (globalTemperature >= 25){
      pwm_velocity = PWM_MAX/10;
      decision = "10/100DC 20<H<50&T>25";
    }
    else {
      decision = "0/100DC 20<H<50&T<25";
      pwm_velocity = 0;
    }
  }
}
```

با توجه به مقدار دما و رطوبت دریافت شده از برد TH باید سرعت گردش موتور تنظیم شود. طبق صورت سوال شرایط زیر برقرار است:

- اگر رطوبت بالاتر از ۵۰ باشد:
  - سرعت گردش موتور باید صفر شود
  - تصمیم اتخاذ شده هم آپدیت شود.
- اگر رطوبت زیر ۲۰ باشد:

- سرعت گردش موتور ۲۵ درصد از سرعت کامل باشد، بنابراین مقدار سرعت را برابر با ۲۵ درصد از مقدار بیشینه PWM قرار می‌دهیم.
- تصمیم اتخاذ شده هم آپدیت شود.

- اگر رطوبت بین ۲۰ تا ۵۰ باشد:
  - اگر دما بیش از ۲۵ باشد:
    - موتور با سرعت ۱۰ درصد از سرعت بیشینه حرکت کند
    - تصمیم اتخاذ شده هم آپدیت شود.
  - در غیر این صورت:
    - موتور متوقف شود
    - تصمیم اتخاذ شده آپدیت شود.

در ادامه مکان فراخوانی این تابع توضیح داده خواهد شد.

به روزرسانی وضعیت

```
69  void updateState(float humidity,float temperature){
70    globalHumidity = humidity;
71    globalTemperature = temperature;
72    updateVelocity();
73    stateUpdated = true;
74  }
```

این تابع وظیفه دارد که مقدار دریافتی توسط بلوتوث را به عنوان مقدار دریافتی صحیح ثبت کند و همچنین با توجه به مقدارها دریافتی با فراخوانی updateVelocity دستورات جدید را اعمال کند. محل فراخوانی این تابع در ادامه توضیح داده خواهد شد.

کنترل کردن سرعت موتور

```
53  void handlePWM(){
54    pwm_counter = (pwm_counter + 1) % (PWM_MAX - PWM_MIN);
55    if (pwm_counter < pwm_velocity){
56      digitalWrite(dc_pinA,HIGH);
57      digitalWrite(dc_pinB,LOW);
58    }
59    else {
60      digitalWrite(dc_pinA,LOW);
61      digitalWrite(dc_pinB,LOW);
62    }
63  }
```

همانگونه که در پروژه قبل دیدیم، PWM می‌تواند باعث شود که سرت موتور را کنترل کنیم. برای پیاده‌سازی PWM به صورت نرم‌افزاری از یک شمارنده استفاده می‌کنیم. تا زمانی که شمارنده مقدار کمتری از pwm_velocity داشته باشد به یکی از پورت‌های درایور موتور سیگنال ۱ و به دیگری صفر می‌دهد. اما پس از عبور مقدار آن از pwm_velocity به هر دو پورت مقدار صفر می‌دهد. به این ترتیب برای مدتی مقدار صفر و پس از آن مقدار یک دریافت می‌شود و این عمل به صورت دوره‌ای تکرار می‌شود. این کد برای هندل کردن این بخش است. محل فراخوانی این تابع در loop است.

دریافت داده‌ها از بلوتوث

```
72  void readBluetooth(){
73    if (Serial.available() >= 13){
74      char start = Serial.read();
75      if (start == BLUETOOTH_TRANSMITION_START_CHAR){
76        float temperature = Serial.parseFloat();
77        Serial.read();
78        float humidity = Serial.parseFloat();
79        char end = Serial.read();
80        if (end == BLUETOOTH_TRANSMITION_END_CHAR){
81          updateState(humidity,temperature);
82        }
83      }
84    }
85  }
```

فرمت پیام‌های ارسالی از برد TH به برد Main به این صورت است که ابتدا یک کارکتر مشخص به عنوان آغاز پیام ارسال شده، سپس عدد مربوط به دما ارسال می‌شود و پس از آن یک کارکتر جدا کننده ارسال شده و پس از آن عدد مربوط به رطوبت و در پایان کارکتر پایان مشخص می‌شود. حال در صورتی یک پیام را می‌پذیریم و فرض می‌کنیم که به درستی دریافت شده است که پیام حاوی کارکتر شروع و پیایان و کارکتر بین بیاشد. دقت کنید برای نمایش یک پیام به ۱۳ کارکتر نیاز است بنابراین فرایند خواندن از بلوتوث در صورتی آغاز می‌شود که حداقل ۱۳ کارکتر در بافر آن باشد بنابراین با تابع available این موضوع را چک می‌کنیم. اگر پیام به درستی دریافت شد تابع updateState فراخوانی می‌شود تا عملیات آپدیت صورت بگیرد.

نمایش اطلاعات بر روی LCD

```
87  void printLCD(){
88    if (stateUpdated) {
89      lcd.clear();
90      lcd.setCursor(0, 0);
91      lcd.println(("T: " + String(globalTemperature)).c_str());
92      lcd.println(("H: " + String(globalHumidity)).c_str());
93      lcd.setCursor(0, 1);
94      lcd.println(decision.c_str());
95      stateUpdated = false;
96    }
97  }
```

این تابع وضیفه دارد در صورتی که از آخرین آپدیت LCD تغییری در وضعیت ایجاد شده بود دوباره LCD را آپدیت نماید. همانگونه که مشاهده می‌کند ابتدا تصویر پاک شده سپس cursor به نقطه ۰ و ۰ روی LDC منتقل شده و دما و رطوبت نوشته می‌شود. سپس cursor به ابتدای خط بعد منتقل شده و تصمیم گرفته شده نمایش داده می‌شود. همچنین متغیر stateUpdated نیز به مقدار false تنظیم می‌شود تا بدانیم که در اجرای بعد loop برای وضعیت فعلی LCD را آپدیت کرده ایم، و فقط در صورتی که وضعیت جدید آمده باشد آن را آپدیت کنیم.

حلقه اصلی برنامه

```
void loop() {
  readBluetooth();
  handlePWM();
  printLCD();
}
```

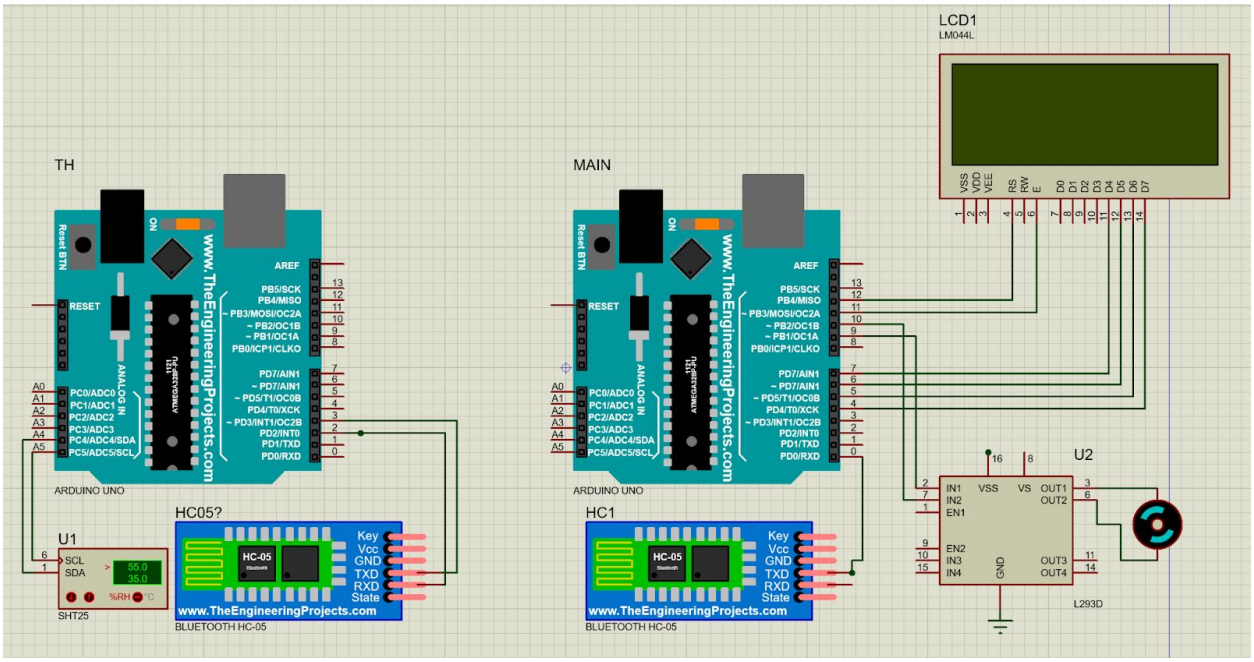
این تابع در حقیقت حلقه اصلی اجرای برنامه است، در هر حلقه ابتدا بلوتوث را برای مقدار جدید کنترل کرده سپس با توجه به مقدار دریافتی و آپدیت شده PWM را تنظیم می‌کند و در پایان نیز عملیات چاپ بر روی LCD را انجام می‌دهد.

سوالات پروژه:

۱: در بلوتوث از امواج رادیویی که به اختصار UHF می‌باشند استفاده شده که رنج آنان بین ۲.۴۰۲ تا ۲.۴۸۰ گیگاهرتز می‌باشد. یکی از راه‌هایی که می‌توان از بروز collision بین داده‌های ارسالی جلوگیری کرد این است که از یکی از مدل‌های FHSS که مخفف frequency hoping spread spectrum می‌باشد استفاده کرد. این مدل AFH نام دارد که مخفف Adaptive frequency hoping می‌باشد. در این روش باند فرکانسی به قطعه‌ها (کانال) های کوچکتر تقسیم‌بندی می‌شود و در اصل به سرعت بین آن کانال‌ها به اصطلاح جهش می‌کند (این سرعت چیزی بالغ بر ۱۶۰۰ مرتبه در یک ثانیه می‌باشد) . به طور مثال زمانی که انرژی رو به پایان است اغلب باند فرکانسی به ۴۰ کانال کوچکتر شکسته می‌شود. علاوه بر این مورد یک کار دیگری که انجام می‌شود (همان بخش Adaptive متد) این است که کانال‌هایی که به هر دلیل یا شلوغ بوده یا نویز زیادی دارند شناسایی شده و از آنان خودکاری می‌شود.

۲: بله از طریق آدرس‌ها. طبق سایت robot-electronics برای آدرس‌های I2C دو حالت وجود دارد. یا ۷ بیتی یا ۱۰ بیتی که ۱۰ بیتی آن کمتر استفاده شده و ۷ بیتی گسترده‌تر است. تمام ماژول‌ها و تراشه‌ها نیز این ۷ بیت را دارا می‌باشند. حال چون ۷ بیت آدرس داریم یعنی حداکثر تا ۱۲۸ دستگاه قابلیت اتصال از طریق درگاه I2C دارند. لازم به ذکر است که علاوه بر این ۷ بیت یک بیت اضافه‌تر هم ارسال شده که برای مشخص کردن نوع عمل (خواندن یا نوشتن) می‌باشد. این ۷ بیت نیز در قسمت فوقانی byte قرار داشته و آن بیت اضافه نیز در کم‌ارزش‌ترین جایگاه بیت می‌باشد.

:۳


TH
MAIN
LCD1
LM044L
ARDUINO UNO
ARDUINO UNO
HC05?
HC1
U1
SHT25
U2
L293D
BLUETOOTH HC-05
BLUETOOTH HC-05
www.TheEngineeringProjects.com
www.TheEngineeringProjects.com
Key
Vcc
GND
TXD
RXD
State
HC-05
RESET
AREF
ON
Reset BTN
ANALOG IN
DIGITAL (PWM~)
PB5/SCK
PB4/MISO
~ PB3/MOSI/OC2A
~ PB2/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~ PD7/AIN1
~ PD5/T1/OC0B
PD4/T0/XCK
~ PD3/INT1/OC2B
PD2/INT0
PD1/TXD
PD0/RXD
PC0/ADC0
PC1/ADC1
PC2/ADC2
PC3/ADC3
PC4/ADC4/SDA
PC5/ADC5/SCL
SCL
SDA
55.0
35.0
%RH
°C
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
IN1
IN2
EN1
VSS
VS
OUT1
OUT2
EN2
IN3
IN4
GND
OUT3
OUT4