Cyber-physical Systems

Computer Assignment #1

Dr Modarresi & Dr Kargahi

Soroosh Mirzasarvari - Mohammad Farrahi - Mahdi Vakili - Parmida Zarghami

Table of Content

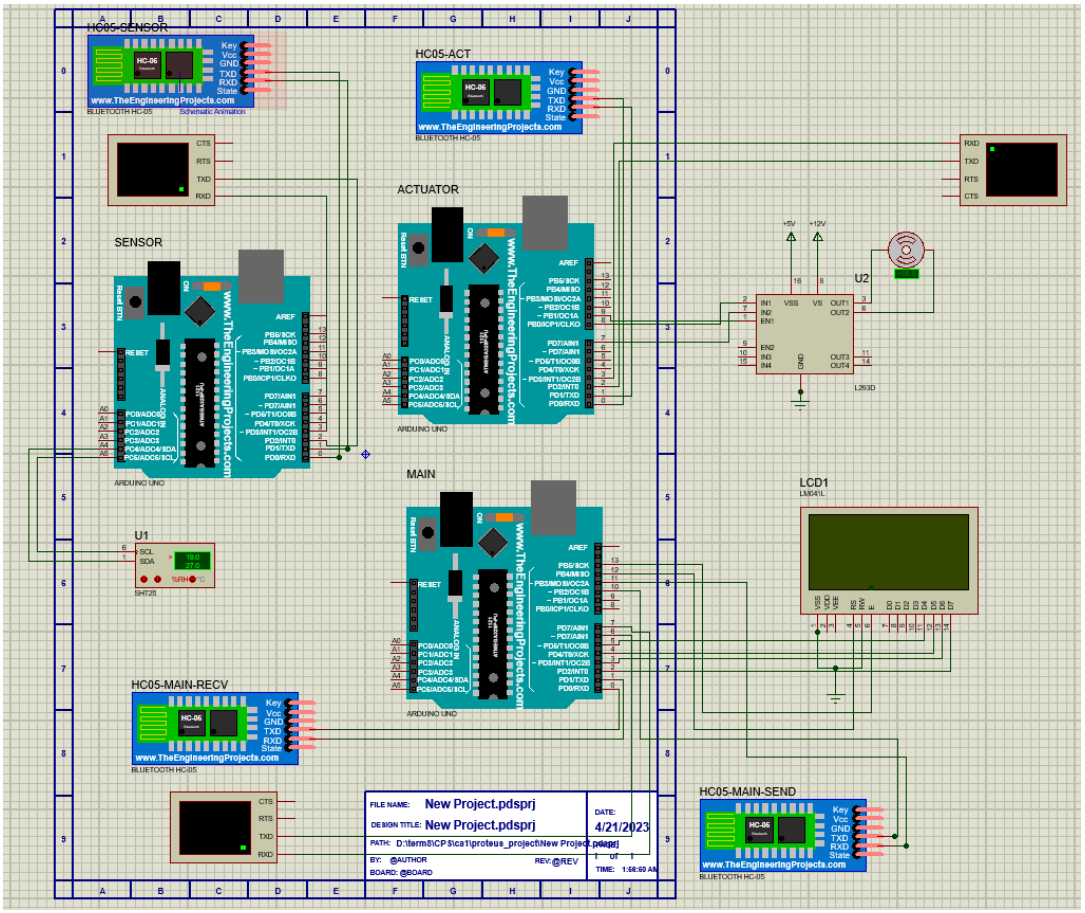
Overview

In this assignment, we build a full system consisting of three boards to measure the humidity and temperature of a flower pot and act on it depending on the said data using a DC motor to water the plant.

The project is done using Arduino boards for ICs. PlatformIO is used as an Integrated Development Environment and Arduino libraries. To simulate the system we use Proteus and Virtual Serial Port software.

The system consists of three boards: sensor, main, and actuator. As the name suggests, the sensor board is responsible for sending humidity and temperature data to the main board. Based on the data, the main board acts as a central unit and decides to open or close a valve connected to a DC motor via the actuator board. Thus, it sends commands to the actuator board to tune the speed of the DC motor. The actuator board receives the command and adjusts the speed of the DC motor.

The Proteus figure of the whole system is as follows:



In the following three sections, each board is explained along with its Proteus figures and codes. After that, the simulation results are displayed and finally, questions are answered.

Sensor Board

Overview

As explained, the sensor board is an Arduino board as well as an HC05-ACT module for Bluetooth connection, an SHT25 sensor for gathering humidity and temperature data, and a Virtual Terminal for debugging purposes.
The board receives and parses the messages from SHT25 and sends them to the main board using the HC05-ACT module via its serial ports (TXD and RXD) using the UART protocol.
To read the data from the SHT25, we use the I2C protocol. First, the board commands the sensor to read relative humidity using the "0xF5" command (Trigger RH measurement) taken from the datasheet. Then the sensor sends the humidity data in two bytes as MSB and LSB that should later be interpreted into actual humidity value using the below formula:

$$RH = -6 + 125 \cdot \frac{S_{RH}}{2^{16}}$$

After that, the board commands the sensor to read the temperature using the "0xF3" command on the I2C bus. The sensor then sends a two-byte number that should be interpreted into actual temperature value using the below formula:

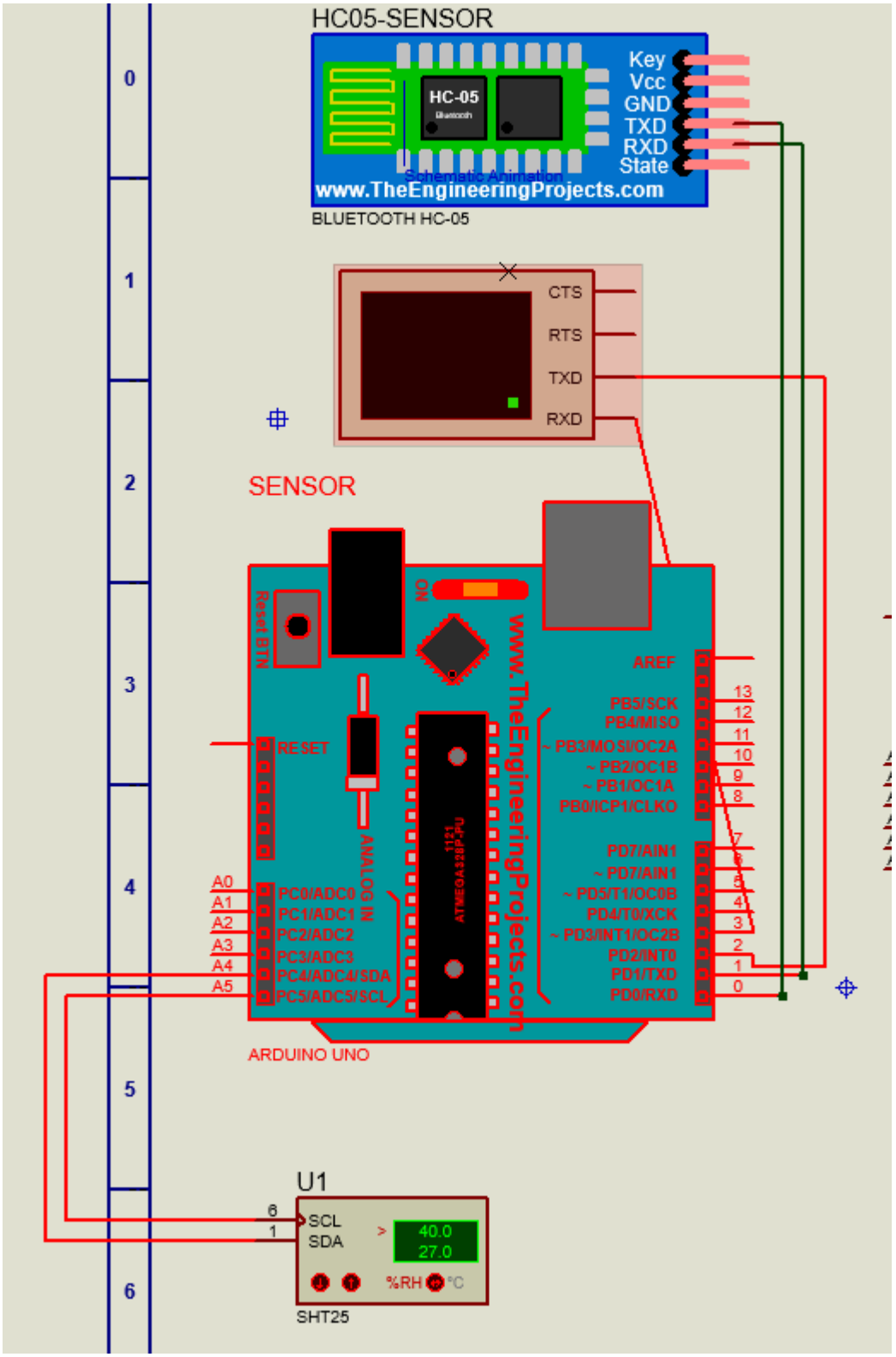
$$T = -46.85 + 175.72 \cdot \frac{S_T}{2^{16}}$$

Humidity and temperature then later are sent to the main board using the UART ports of Arduino and the HC05-ACT module. **We have developed a format for the byte stream of these two values as below:**

"H<humidity_value>T<temprature_value>$"

Which is later parsed by the main board to extract the values.

Proteus Figure



Code

First, we need to set up the modules:

```
8   // SHT25 I2C address is 0x40(64)
9   #define Addr 0x40
10
11  // TX|RX of Virtual Monitor
12  #define VMTX 3
13  #define VMRX 2
14
15  struct SensorData {
16    float humidity;
17    float cTemp;
18  };
19
20  SoftwareSerial VMonitor(VMRX, VMTX); // RX | TX
21
22  void setup() {
23    // Start the I2C communication with the SHT25 sensor
24    Wire.begin();
25    // Initialise serial communication, set baud rate = 9600
26    Serial.begin(9600);
27    VMonitor.begin(9600);
28  }
```

- ➔ “Addr” is the address 64 for the I2C output for the SHT25 sensor.
- ➔ “VMTX and VMRX” are the additional pins instead of the standard UART pins to connect the Virtual Monitor to Arduino.
- ➔ “SensorData is a structure to hold temperature and humidity as floats.
- ➔ “VMonitor” is an object of the “SoftwareSerial” class which is an API library to work with the Virtual Monitor.
- ➔ The “setup” function is the standard setup function of IOPlatform which is the first function called upon the Arduino board. In the setup function, we first initialize the “Wire” object using the “begin” method. “Wire” is an object in the “Wire.h” library which is an API to work with I2C protocol. The “Serial” object is another API class to work with **the default pins of UART** (PD1/TXD and PD0/RXD). The 9600 value assigned in the “begin” method of both “Serial” and “VMonitor” objects is the baud rate.

The “loop” function is as follows:

```
void loop() {
  struct SensorData sensorData = PollFromSHT25();
  if(abs(latestSentHumdity - sensorData.humidity) > 5) {
    sendToMainNode(&sensorData);
    latestSentHumdity = sensorData.humidity;
  }
  delay(1000);
}
```

The “loop” function is the function that is constantly being executed by the Arduino board in which we first call the “PollFromSHT25” function to get the data from the sensor, then based on the condition pointed out in the CA description, we check if the last reported humidity has a difference of at least 5% from the currently recorded humidity. If true, the “sendToMainNode” function is called and the “latestSentHumidity” is updated.

The “PollFromSHT25” function’s body consists of two parts: reading humidity and reading temperature.

The part where we extract the humidity is as follows:

```
32  struct SensorData PollFromSHT25() {
33    struct SensorData result;
34    unsigned int data[2];
35
36    // Start I2C transmission
37    Wire.beginTransmission(Addr);
38    // Send humidity measurement command, NO HOLD master
39    Wire.write(0×F5);
40    // Stop I2C transmission
41    Wire.endTransmission();
42
43    delay(500);
44
45    // Request 2 bytes of data
46    Wire.requestFrom(Addr, 2);
47
48    // Read 2 bytes of data in the following format:
49    // [humidity msb, humidity lsb]
50    if(Wire.available() == 2) {
51      data[0] = Wire.read();
52      data[1] = Wire.read();
53
54      // Convert the data
55      float humidity = (((data[0] * 256.0 + data[1]) * 125.0) / 65536.0) - 6;
56      result.humidity = humidity;
57      // Output data to Serial virtualMonitor
58      VMonitor.print("Humidity :");
59      VMonitor.print(humidity);
60      VMonitor.println(" %RH");
61    }
```

➔ First, the “beginTransmission” method is used for the board to tell SHT25 that we want to transmit data to the I2C bus. Then, the “0xf5” command is sent telling the SHT25 to report relative humidity. After a 500ms delay, we request from the 0x40 (64) pin of I2C for 2 bytes of data.

➔ If the two-byte data is available, the humidity percentage is calculated using the said formula and put into the data structure.

➔ The result is also displayed in the VMonitor using the “print” method.

The part where we extract the temperature is as follows:

```
63    // Start I2C transmission
64    Wire.beginTransmission(Addr);
65    // Send temperature measurement command, NO HOLD master
66    Wire.write(0xF3);
67    // Stop I2C transmission
68    Wire.endTransmission();
69    delay(500);
70
71
72    // Request 2 bytes of data
73    Wire.requestFrom(Addr, 2);
74
75    // Read 2 bytes of data in the following format:
76    // [temp msb, temp lsb]
77    if(Wire.available() == 2) {
78      data[0] = Wire.read();
79      data[1] = Wire.read();
80
81      // Convert the data
82      float cTemp = (((data[0] * 256.0 + data[1]) * 175.72) / 65536.0) - 46.85;
83      result.cTemp = cTemp;
84
85      // Output data to Serial virtualMonitor
86      VMonitor.print("Temperature in Celsius :");
87      VMonitor.print(cTemp);
88      VMonitor.println(" C");
89      VMonitor.println("——");
90    }
91
92    return result;
93  }
```

➔ First, the “beginTransmission” method is used for the board to tell SHT25 that we want to transmit data to the I2C bus. Then, the “0xf3” command is sent telling the SHT25 to report the temperature. After a 500ms delay, we request from the 0x40 (64) pin of I2C for 2 bytes of data.

➔ If the two-byte data is available, the temperature is calculated using the said formula and put into the data structure.

➔ The result is also displayed in the VMonitor using the “print” method.

The “sendToMainNode” function’s body is as follows:

```
95  void sendToMainNode(struct SensorData* sensorData) {
96    // Data will be sent in the following format:
97    "H<humidity>T<temperature>$"
98
99    Serial.print("H");
100   Serial.print(sensorData→humidity);
101   Serial.print("T");
102   Serial.print(sensorData→cTemp);
103   Serial.print("$");
104 }
105
```

➔ The “sensorData” struct is sent using the “Serial” object and its “print” method to the HC05-ACT using the format explained in the overview section.

Main Board

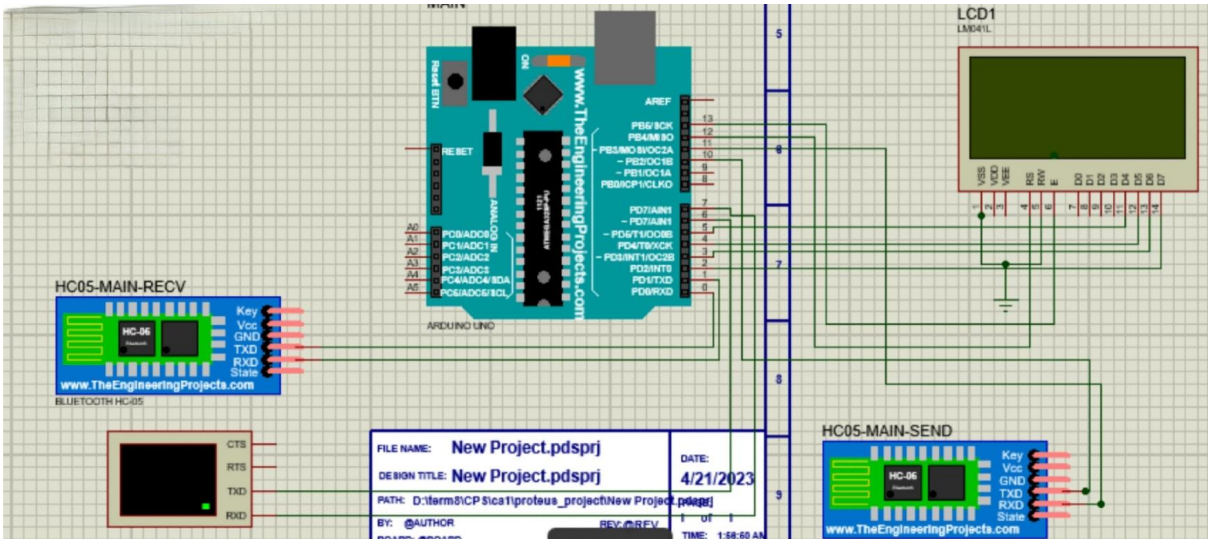
Overview

The main board consists of an Arduino board as well as two HC05-ACT modules for Bluetooth connection, an LM041L LCD module for displaying humidity and temperature data, and a Virtual Terminal for debugging purposes.
The board receives and parses data from the sensor board and processes them to find the duty cycle in which the DC motor has to rotate. The main board has to command the actuator board on how fast the DC motor has to rotate.
To receive the data from the sensor board, we use an HC05-ACT Bluetooth module using the UART protocol. This data is essentially a byte stream in the format explained in the previous section.
After acquiring the humidity and temperature, the duty cycle is calculated based on the conditions explained in the project description. The data is also displayed in the LM041LC LCD module.
The calculated duty cycle is then sent to the actuator board using another HC05-ACT module and UART protocol. **Since the default UART pins (PD1/TXD and PD0/RXD) are occupied with the receiver Bluetooth module, the sender Bluetooth module is connected to pins 10 and 11 for TX and RX respectively.** The duty cycle is transferred as a byte stream in the following format:

“DC:<duty_cycle_value>”

For working with UART protocol, “SoftwareSerial.h” and for working with LCD, the “LiquidCrystal.h” library is used.

Proteus Figure



Code

To start, we have to initialize the system:

```
#define LCD_RS 12
#define LCD_E 13
#define LCD_D4 5
#define LCD_D5 4
#define LCD_D6 3
#define LCD_D7 2

#define VMTX 7
#define VMRX 6

#define SBTX 11
#define SBRX 10

#define Null -10000.0

struct SensorData {
  float humidity;
  float cTemp;
};

SoftwareSerial VMonitor(VMRX, VMTX); // RX | TX
SoftwareSerial senderBluetooth(SBRX,SBTX);
// Set up the LCD
LiquidCrystal lcd(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7); // RS, E, D4, D5, D6, D7

void setup() {
  // Start the serial communication(attached to reciever bluetooth module)
  Serial.begin(9600);
    // Start the serial communication to sender bluetooth module
  senderBluetooth.begin(9600);
  // Start the Virtual Monitor communication
  VMonitor.begin(9600);
  // Set up the LCD
  lcd.begin(20, 4);
}
```

- ➔ Lines 7 to 12, define pins for connection between Arduino board and LCD.
- ➔ Pins 6 and 7 are dedicated to Virtual Monitor.
- ➔ Pins 10 and 11 are dedicated to the Bluetooth module that sends data to the actuator board.
- ➔ A similar “SensorData” struct is used to contain the humidity and temperature data.
- ➔ "VMmonitor” object is instantiated.
- ➔ “SenderBluetooth” object is instantiated.
- ➔ “lcd” object from the “LiquidCrystal” class is instantiated.
- ➔ “setup” function is called in which “Serial”, “senderBluetooth”, “VMonitor”, and “lcd” objects are initialized. The 9600 value is the baud rate and the (20,4) value used in the “begin” method of “lcd” is the LCD size.

After setup, the “loop” function is constantly called of which the body is as follows:

```
99
100 void loop() {
101   SensorData recievedData = recieveFromSensor();
102   // check if data recieved correctly
103   if (recievedData.humidity ≠ Null) {
104     // get the duty cycle based on the specified conditions
105     int irrigationDutyCycle = getIrrigationDutyCycle(recievedData);
106     // Display the temperature and humidity values on the LCD
107     printOnLCD(recievedData, irrigationDutyCycle);
108     // Send the irrigation rate command to the Actuator Node via Bluetooth
109     senderBluetooth.print("DC:");
110     senderBluetooth.print(irrigationDutyCycle);
111   }
112 }
```

- ➔ The “recieveFromSensor” function is called to get the data from the sensor board.
- ➔ The received data is checked to be valid.
- ➔ The duty cycle is calculated in the “getIrrigationDutyCycle” function.
- ➔ The temperature, humidity, and calculated duty cycle are displayed on the LCD using the “printOnLCD” function.
- ➔ The duty cycle is transferred to the actuator board using the mentioned format.

The “recieveFromSensor” function’s body is as follows:

```
SensorData recieveFromSensor()
{
  SensorData recievedData = {Null, Null};
  // Check if data is available from the Sensor Node via Bluetooth
  if (Serial.available())
  {
    // Read the received data
    String data = Serial.readStringUntil('\n');
    // Parse the temperature and humidity values from the received data
    float humidity = data.substring(data.indexOf("H") + 1, data.indexOf("T")).toFloat();
    float temperature = data.substring(data.indexOf("T") + 1).toFloat();
    // Print the received data to the Serial Monitor
    VMonitor.print("Received data: ");
    VMonitor.println(data);
    recievedData.cTemp = temperature;
    recievedData.humidity = humidity;
  }
  return recievedData;
}
```

- ➔ A condition is checked on whether there is any data available in the **Default UART pins**. As explained before, the “Serial” object is itself a “SfotwareSerial.h” object that works with the default pins of the UART protocol. In this case, the receiver’s Bluetooth module is connected to these pins.
- ➔ One line is read from the byte stream.
- ➔ The line is parsed into the humidity and temperature.
- ➔ Data is printed on the VMonitor for debugging purposes.
- ➔ The data is returned in the form of a “SensorData” struct.

The “getIrrigationDutyCycle” function’s body is as follows:

```
int getIrrigationDutyCycle(SensorData sensorData)
{
  // Calculate the irrigation rate based on the specified conditions
  int irrigationDutyCycle = 0;
  if (sensorData.humidity ⩾ 20 && sensorData.humidity ⩽ 30)
  {
    if (sensorData.cTemp > 25)
    {
      // Irrigation rate of 10 cc/min with 10% duty cycle
      irrigationDutyCycle = 10;
    }
  }
  else if (sensorData.humidity ⩾ 10 && sensorData.humidity < 20)
  {
    // Irrigation rate of 15 cc/min with 20% duty cycle
    irrigationDutyCycle = 20;
  }
  else if (sensorData.humidity < 10)
  {
    // Irrigation rate of 20 cc/min with 25% duty cycle
    irrigationDutyCycle = 25;
  }

  VMonitor.print("IRR-DC:");
  VMonitor.print(irrigationDutyCycle);
  VMonitor.println("——");
  return irrigationDutyCycle;
}
```

➔ The “sensorData” is used to calculate the “irrigationDutyCycle” based on the rules explained in the project description.

➔ It is printed on the VMonitor for debugging purposes and returned.

The “printOnLCD” function’s body is as follows:

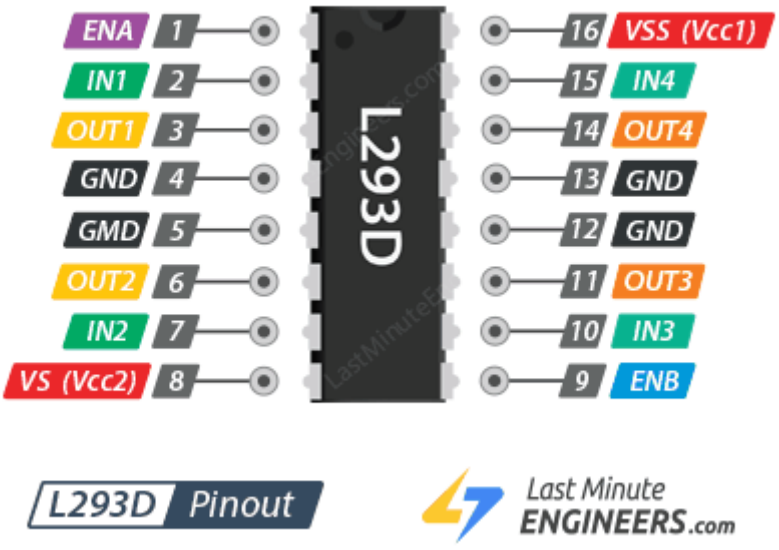
```
void printOnLCD(SensorData sensorData, int irrigationDutyCycle)
{
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(sensorData.cTemp);
  lcd.print(" C");
  lcd.setCursor(0, 1);
  lcd.print("Humid: ");
  lcd.print(sensorData.humidity);
  lcd.print(" %");
  lcd.setCursor(0, 2);
  lcd.print("IRR-DC:");
  lcd.print(irrigationDutyCycle);
  lcd.print(" %");
}
```

➔ We essentially use the “LiquidCrystal.h” library to print data on the LM041L LCD.

Actuator Board

Overview

The actuator board consists of an Arduino board, as well as an HC05-ACT module for Bluetooth connection, a DC motor, an L293D driver, used to work with the DC motor, and a Virtual Terminal for debugging purposes.
The board receives and parses the duty cycle from the main board and acts on it by tuning the speed of the DC motor using the L293D driver.
The command coming from the main board is first received from the HC05-ACT and then parsed based on the format introduced in the previous section.
The duty cycle - which is a percentage from 0 to 100 - is then converted into a number between 0 to 255 that essentially represents the PWM (Pulse Width Modulation) of the motor. The datasheet for L293D and guide on how to use it is adopted from [this website](#).
The L293D module is a driver able to control two DC motors. A figure featuring all pins of this device is as below:



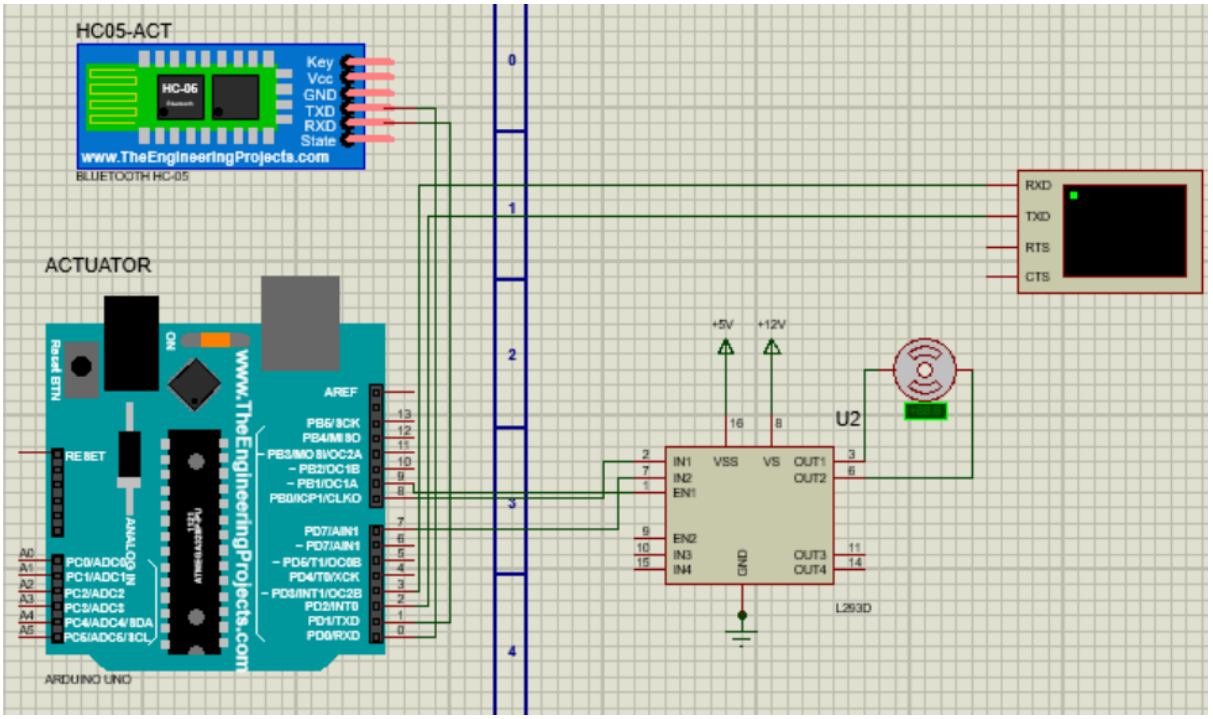
VSS and VS pins are the High power supply for the device. VSS is used as logical 1 for the internal logic circuit that is connected to a 5V supply and VS is the High power supply for the motor that in this project is connected to a 12V supply.
OUT1 and OUT2 are two control 1 motor and OUT3 and OUT4 are to control another.

For each motor, two “IN” pins control the direction in which the motor spins and are set based on the below table:

IN1	IN2	Spinning Direction
Low(0)	Low(0)	Motor OFF
High(1)	Low(0)	Forward
Low(0)	High(1)	Backward
High(1)	High(1)	Motor OFF

In this project, IN1 and IN2 are set to High(1) and Low(0) which lead to the motor going forward (clockwise).
The speed of the motor is controlled by EN pins with the PWM technique which essentially means that in a unit of time (namely a clock), on average, what proportion of the unit, is the EN pin is on high. This is achieved by writing an analogue pulse on the EN pin concerning the duty cycle. Meaning, if the duty cycle is 100%, 255 is written on EN and if the duty cycle is 0%, 0 is written on EN.

Proteus Figure



Code

To set things up the below code is executed:

```
5  #define VMTX 3
6  #define VMRX 2
7  #define MOTOR_EN_PIN 9
8  #define MOTOR_IN1_PIN 8
9  #define MOTOR_IN2_PIN 7
10
11 SoftwareSerial VMonitor(VMRX, VMTX); // RX, TX pins for Vritual Monitor
12
13 void setup() {
14   // setup DC motor pins
15   pinMode(MOTOR_EN_PIN, OUTPUT);
16   pinMode(MOTOR_IN1_PIN, OUTPUT);
17   pinMode(MOTOR_IN2_PIN, OUTPUT);
18   digitalWrite(MOTOR_IN1_PIN, HIGH);
19   digitalWrite(MOTOR_IN2_PIN, LOW);
20
21   VMonitor.begin(9600); // Start serial communication with Vurtyak Monitor module
22   Serial.begin(9600); // Start serial communication with Bluetooth module
23 }
```

- ➔ Lines 5 to 9 define pins for Virtual Monitor and L293D pins.
- ➔ The “VMonitor” object is instantiated.
- ➔ The “setup” function is called in which the “pinMode” function is called. This function sets the mode of the pin to be of type “OUTPUT”
- ➔ To set the direction of the motor, the “digitalWrite” function is called on the IN1 and IN2 pins of the motor setting the direction to HIGH and LOW respectively which results in forward rotation as explained in the previous section.
- ➔ The “VMonitor” and “Serial” objects are initiated with a 9600 baud rate.

The “loop” function is then called which is as follows:

```
42 void loop() {
43   int dutyCycle = recieveFromMain();
44   if (dutyCycle ≥ 0) {
45     // Control the motor based on the received speed
46     int pwmValue = map(dutyCycle, 0, 100, 0, 255);
47     analogWrite(MOTOR_EN_PIN, pwmValue);
48   }
49 }
50
```

- ➔ The “recieveFromMain” function is called to receive the duty cycle from the main board via the Bluetooth module.
- ➔ Duty cycle is then checked to be a positive value.

➔ PWM is calculated by mapping the duty cycle -which is from 0 to 100- to a range of 0 to 255.

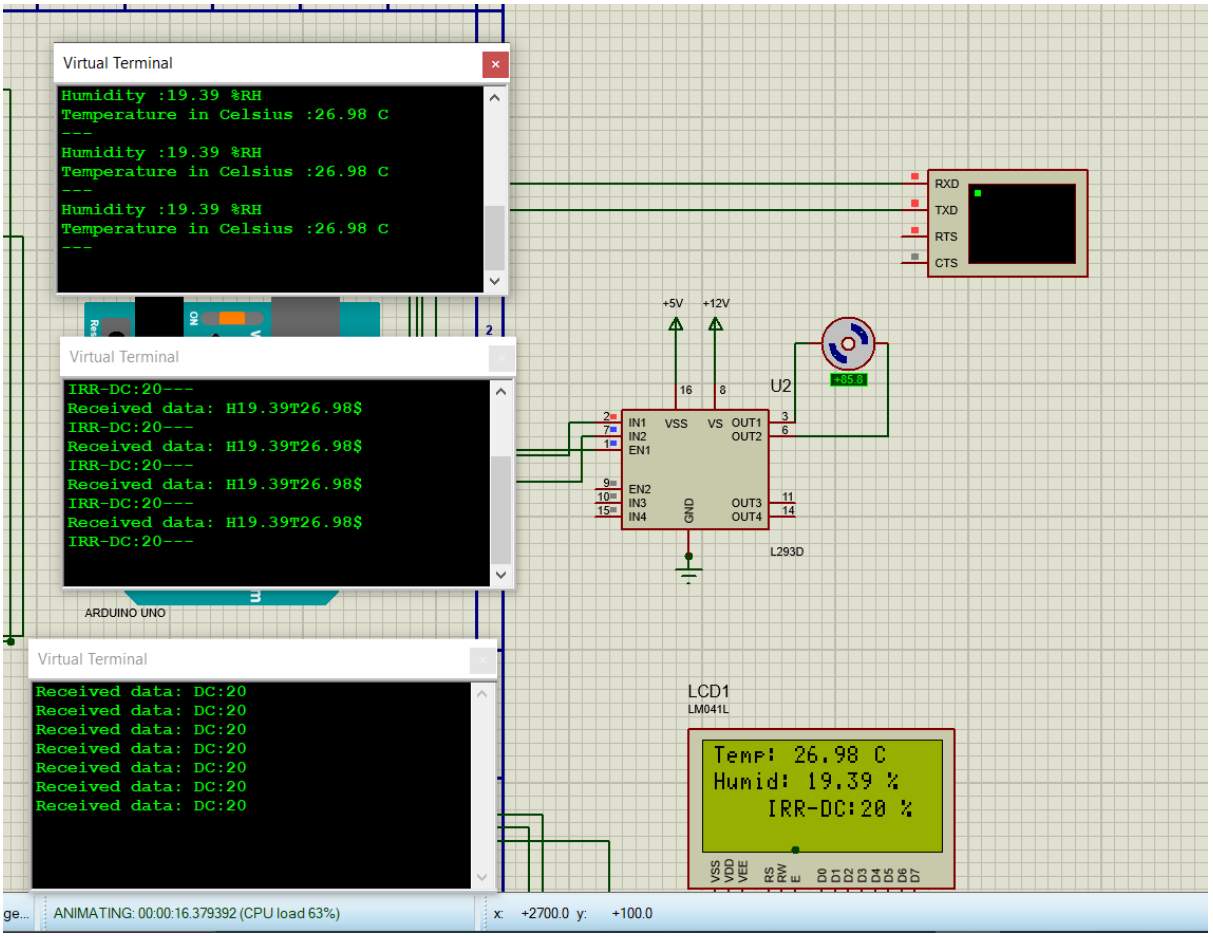
➔ The calculated PWM is then written on the EN pin of the motor by calling the “analogWrite” function.

The “recieveFromMain” function’s body is similar to the previous two boards:

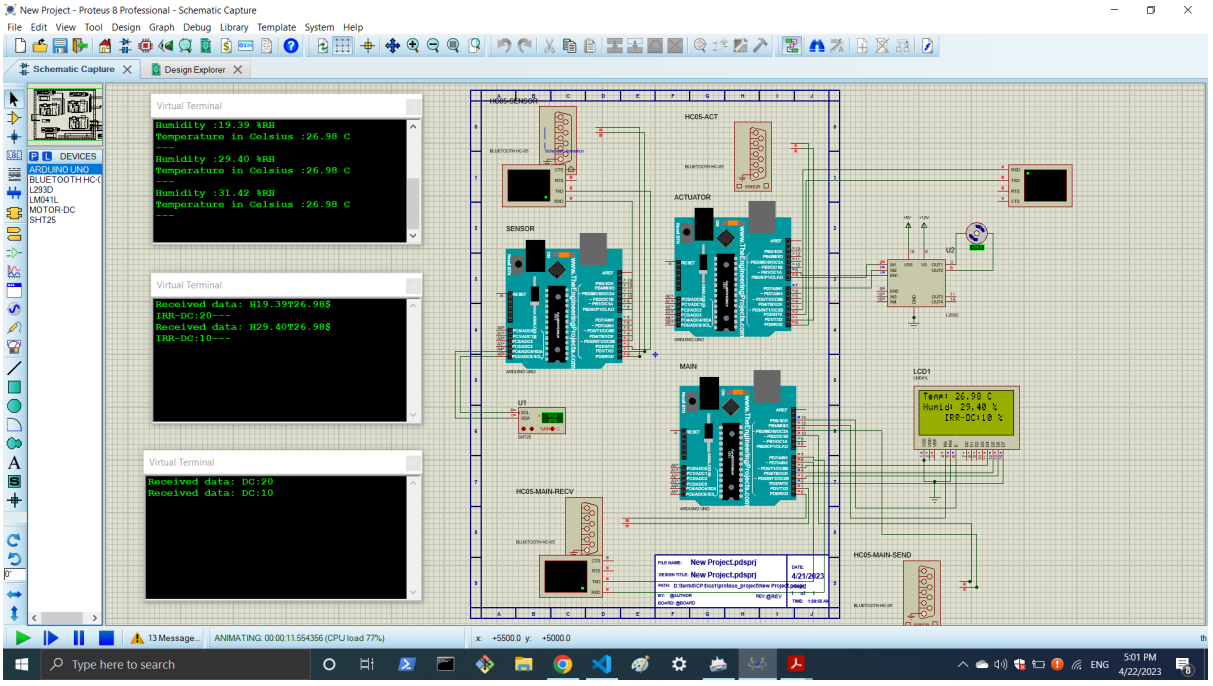
```
25  int recieveFromMain() {
26      int dutyCycle = -1;
27      // Check if there is any data available on the Bluetooth module
28      if (Serial.available()) {
29        // Read the received data
30        String data = Serial.readStringUntil('\n');
31        // Parse the duty cycle value from the received data
32        dutyCycle = data.substring(data.indexOf("DC:") + 3).toInt();
33        // Print the rec(const char [16])"Received data: "
34        VMonitor.print("Received data: ");
35        VMonitor.println(data);
36      }
37      return dutyCycle;
38  }
```

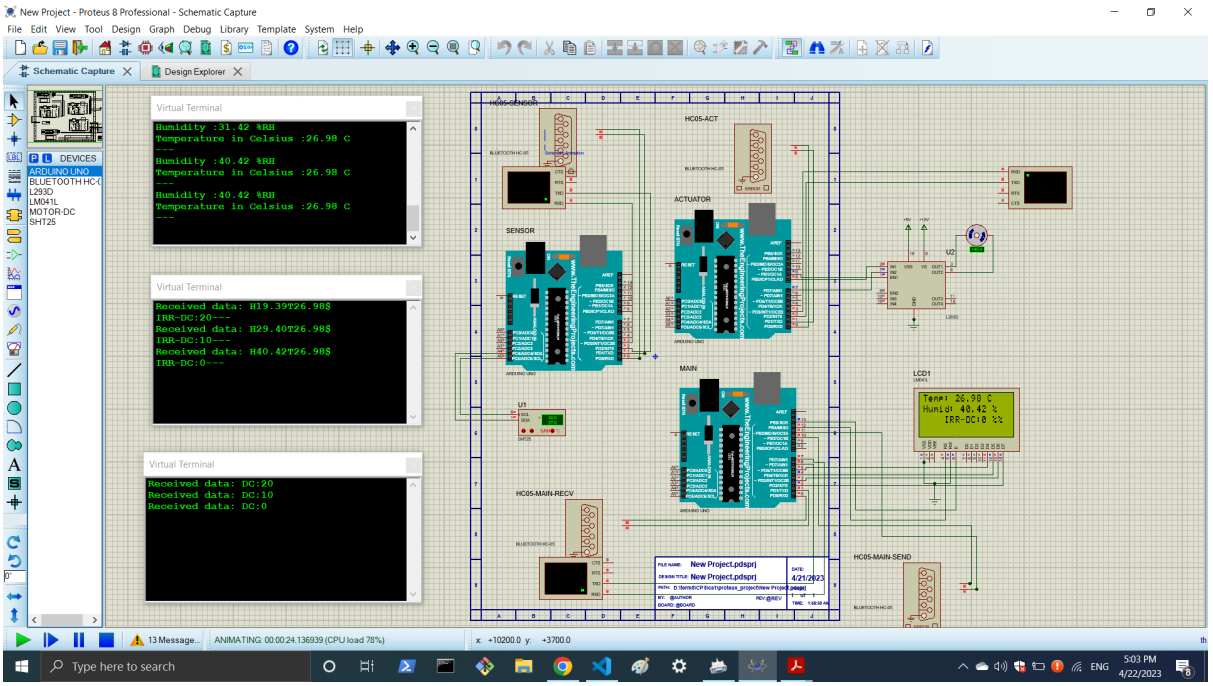
Simulation Results

Below, figures featuring the simulation results are present:

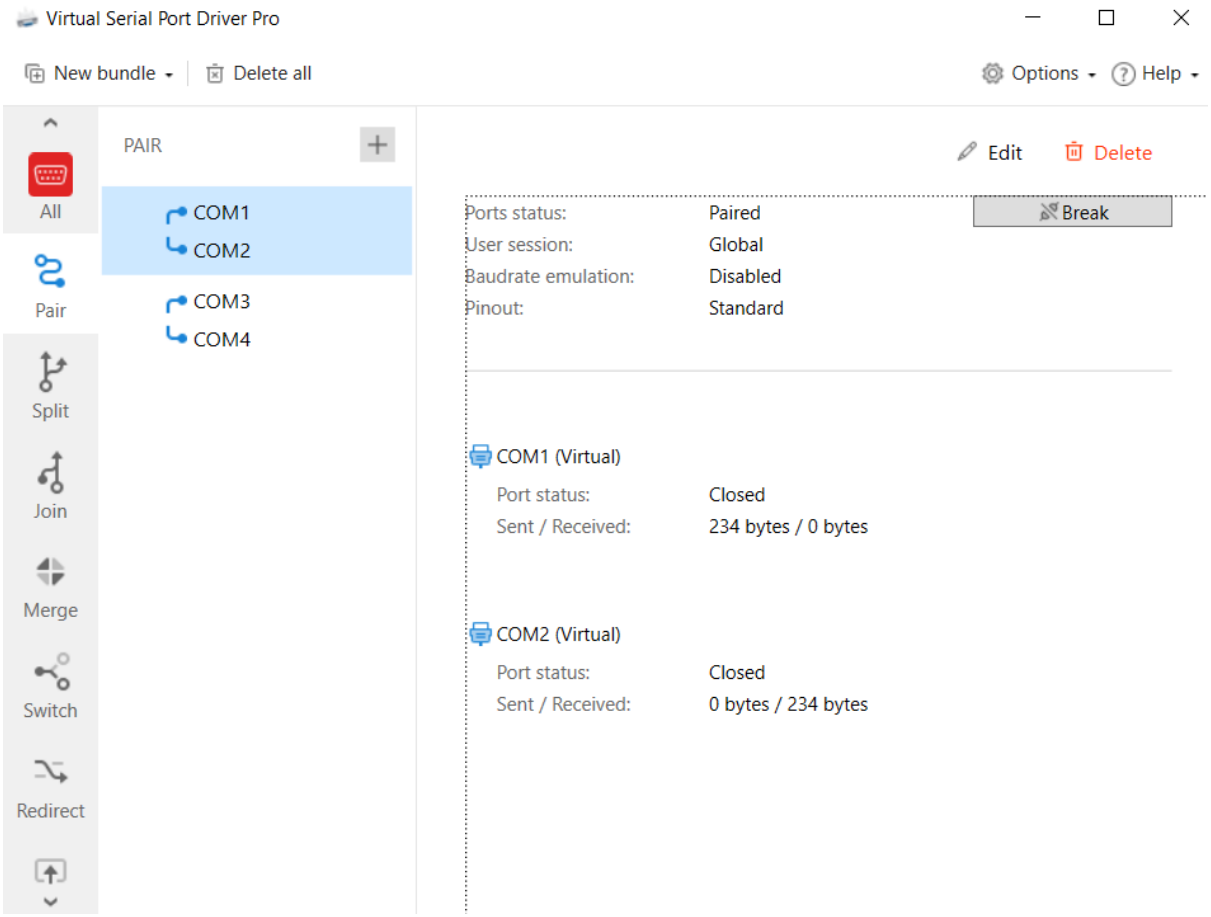


When we change the relative humidity to more than 5%, the system starts to react to it and the duty cycle is first 20%, then 10%, and eventually 0%. Two consecutive snapshots of the system are as follows:





As explained in the first Overview section, to establish Bluetooth connection in the Proteus software, another software called Virtual Serial Port Driver is used. Below is a figure featuring the connections established between COM1 and COM2, which is the connection from the sensor board to the main board, and COM3 and COM4 which is the connection between the main board and the actuator board:



Questions

1. According to the datasheet for HC05-ACT, the frequency is equal to 2.45GHz. To avoid the conflict between the three nodes, since every connection has to be established with a master and a slave, in the main board, two separate modules are used.

2. The I2C protocol has a handshake-like mechanism to avoid the intervention of unwanted nodes. This is achieved by the master sending the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low. Then in the next frame, the master sends each slave the 7 or 10-bit address of the slave it wants to communicate with, along with the read/write bit and thus only the slave whose address matches the requested address sends/receives the data.

3.
    a. Stepper motor: It is a type of electric motor that moves in discrete steps. It is controlled by a series of electronic pulses that cause the motor to rotate step by step. Stepper motors are typically used in applications where precise positioning is required, such as robotics, 3D printing, and CNC machines. They have high torque at low speeds and can maintain a position without using feedback.

    b. DC motor: A DC motor uses direct current (DC) electricity to power its magnetic field, which in turn causes the motor's rotor to rotate. DC motors are widely used in applications such as fans, drills, and conveyor systems. They are less precise than stepper motors and require external feedback to achieve accurate positioning.

    c. Servo motor: A servo motor is a type of DC motor that incorporates a feedback mechanism to precisely control the motor's position. Servo motors are used in applications that require precise and accurate positioning, such as robotic arms, cameras, and aerospace systems. They have high torque and can maintain a position without using external feedback.

Stepper motors are best suited for precise positioning, DC motors are well-suited for low-cost and simple applications, and servo motors are best suited for applications requiring high precision and accuracy.