I. Introduction

This project involves designing a device to measure and display relative humidity. The system uses a DHT11 humidity sensor connected to a microcontroller unit (MCU), which processes the sensor data and displays the readings on an LCD 4-bit display. This device is useful in various applications, including home automation, weather stations, and industrial humidity monitoring.

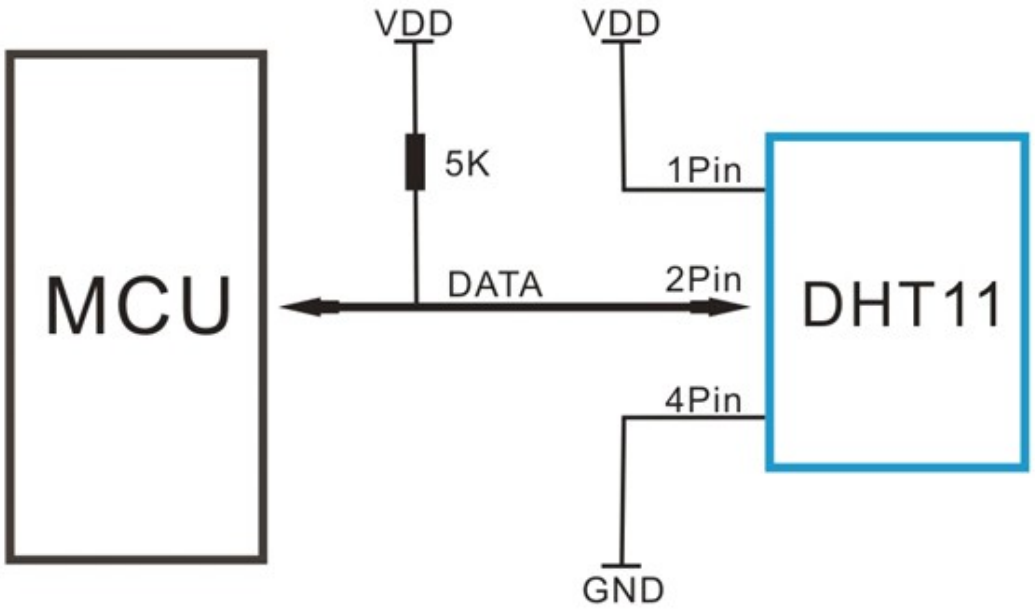


Figure 1 Typical Application

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

Source: DHT11 Humidity & Temperature Sensor Datasheet

The DHT11 sensor uses a single wire to transmit data. This wire is bidirectional and serves both to send commands from the microcontroller to the sensor and to transmit data from the sensor back to the microcontroller.

A pullup resistor is required between the data wire and VCC because the DHT11 uses a bidirectional communication system on a single wire. When neither end is communicating both ends of the link will be in "high impedance" mode - i.e., input mode. In that case the signal will be "floating" and needs the pullup to keep it in a known state.

II. Basic Theory

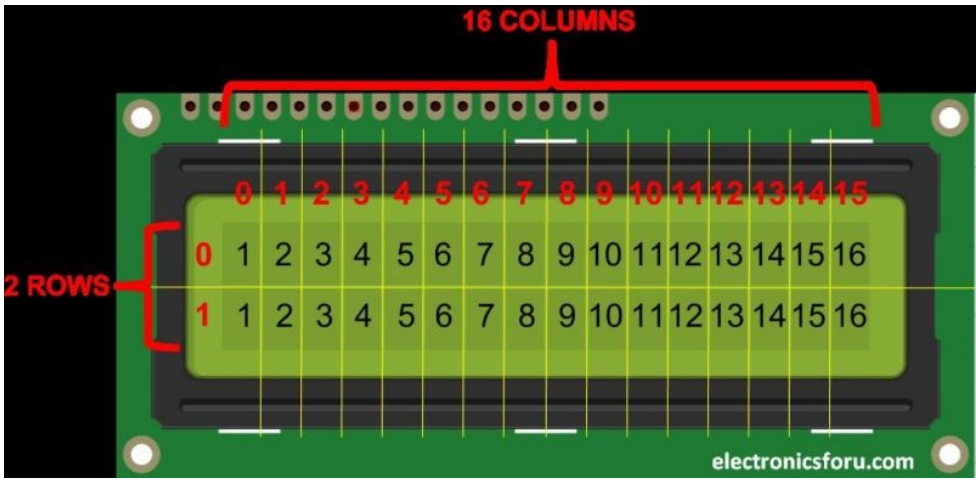
The DHT11 sensor measures relative humidity by using a capacitive humidity sensor and a thermistor. The sensor provides a digital output that can be read by the MCU. We included a checksum formula to ensure the data integrity.

```
for (index = 0, dataChecksum = 0; index < 4; index++)          // Loop to calculate checksum
    dataChecksum += dataSignal[index];
```

III. Liquid Crystal Display

We had the option to use a multiplexed 7-segment display to display each requirement. However, a 7-segment display has its drawbacks as it is limited in its ability to display any character or symbol that does not align with the configuration of the display (i.e., diagonal line). It can only display a limited set of predefined characters or numbers. On the other hand, the LCD, Liquid Crystal Display, is more flexible as it is able to display a wider range of characters, making it more suitable for this project.

In this project, we used the 2x16 LCD. This LCD is composed of 2 lines (L1,L2) and can display 16 characters on each line, meaning it can display 32 characters in total.



There are 4-bit and 8-bit modes in LCDs. Using the 8-bit mode requires 8 data pins (D0..D7) and 3 control pins, which is a total of 11 pins. However, the 4-bit mode requires 4 data pins (D4..D7) and 3 control pins, a total of 7 pins. As such, the latter was chosen to be used to save pins.

To access the 4-bit LCD, include the library <LCD4PICPROTO.h>.

The 3 control pins of the LCD are: RS, R/$\overline{W}$, and E.

The Register Select bit, RS, informs the LCD whether the upcoming information is an instruction or data. When set, it signifies data; when cleared, it indicates an instruction. The Read/Write bit, R/$\overline{W}$, conveys to the LCD whether the operation is a read or write action. If set, it's a read operation; if cleared, it's a write operation. The Enable bit, E, serves as a signal to the LCD, indicating when it should read the data lines. When transitioning from low to high, it instructs the LCD to capture and process the data on the data lines.

- RS: Register Select → 0 : LCD Command Mode
  → 1 : LCD Data Mode
- R/$\overline{W}$: Read/Write → 0 : Write Data
  → 1 : Read Data
- E: Enable Control Signal → Rising Edge
  → Falling Edge

To initialize the required pins and settings, we call the InitLCD() function within the setup function. The InitLCD() function initializes the required pins and settings, eliminating the need for manual setup.

IV. RAM vs ROM

Random Access Memory, RAM, can be both read from and written to, and it temporarily stores the data. When power is removed from the system, the data is lost and the system resets. As such it is known to be a volatile part. On the other hand, Read Only Memory, ROM, is considered a non-volatile part as it stores data and keeps it unchanged. Data in ROM cannot be changed or erased.

V. DispRomStr Function

Display Read-Only Memory String, DispRomStr, function is used to display on the LCD predefined texts that remain unchanged throughout the operations. In this project, we want the

LCD to display fixed messages such as: “Temp: ”, “Hum: ”, etc. to inform the user about the steps to follow while using the design.

VI. DispRamStr Function

Display Variable String, DispVarStr, function is used to display on the LCD dynamic content. Information such as humidity and temperature values, which will change depending on the user and the program’s execution, are displayed using this function. As such, it helps display updated and real-time data on the LCD.

VII. Delay

Delay functions are essential for controlling timing in such projects. These functions are based on a set number of clock cycles. ‘TCYx’ refers to clock cycles, and the number that follows it indicates the number of cycles to wait. The delay is based upon 2 factors: the specified number of cycles and the number in parentheses. If we want to create a pause of 50milliseconds, for example, we can use ‘Delay10KTCYx(5)’, where 5 * 10,000 cycles = 50 milliseconds. We can also use ‘Delay1KTCYx(50)’, as they give the same result where 50*1000 cycles = 50 milliseconds.

In this project, the use of delay functions in terms of managing time is opted to due to the port configuration.

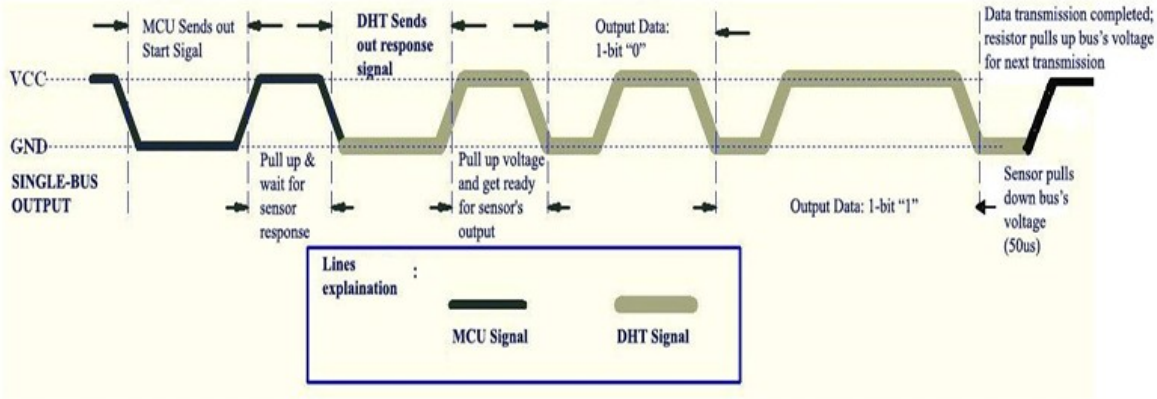


Figure 2 Overall Communication Process

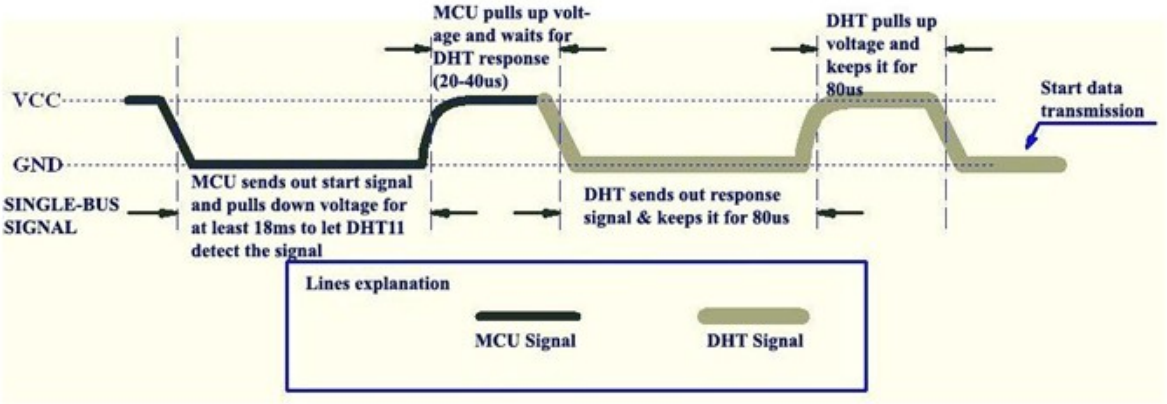
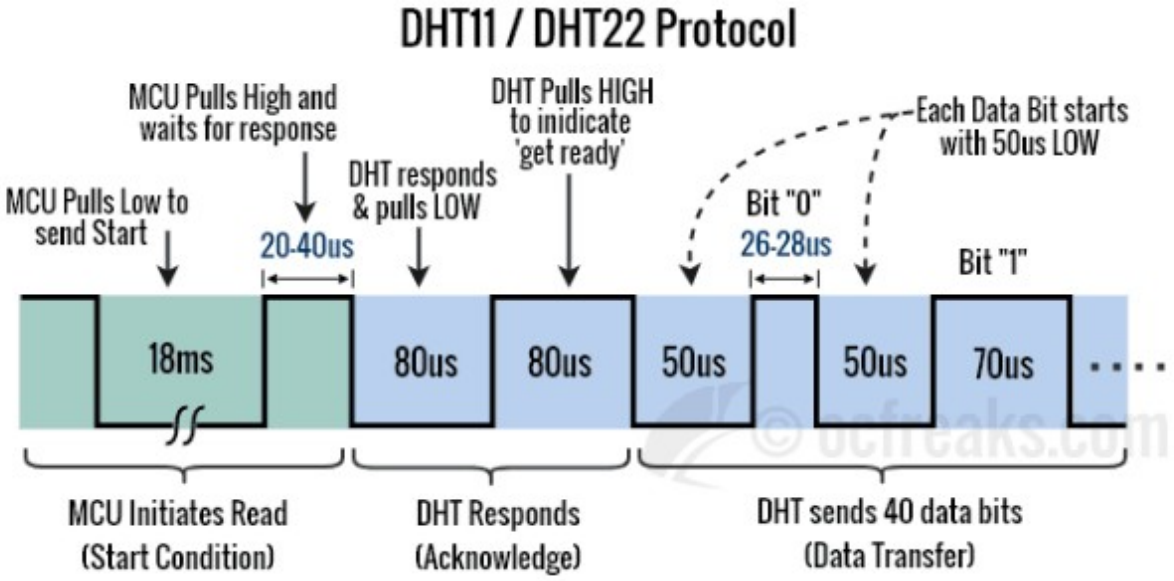


Figure 3 MCU Sends out Start Signal & DHT Responses

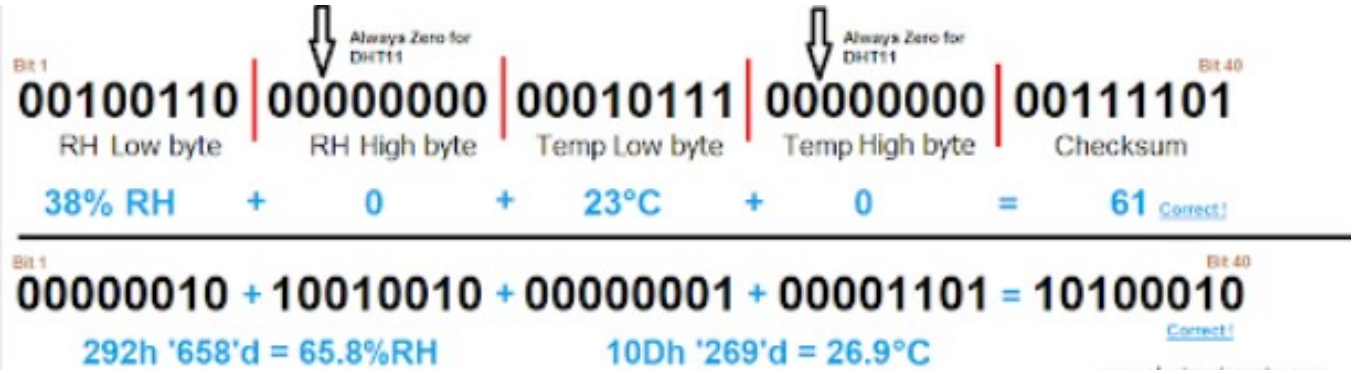
Source: DHT11 Humidity & Temperature Sensor Datasheet (DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf (mouser.com))

The microcontroller acts as the bus master and hence is responsible for initiating communication (i.e. Read). DHT11 Humidity and Temperature sensor always remain as slave and responds with data when MCU asks for it. The protocol used for communication is simple and can be summarized as follows:



Source: Basics of Interfacing DHT11/DHT22 Humidity and Temperature Sensor with MCU (ocfreaks.com)

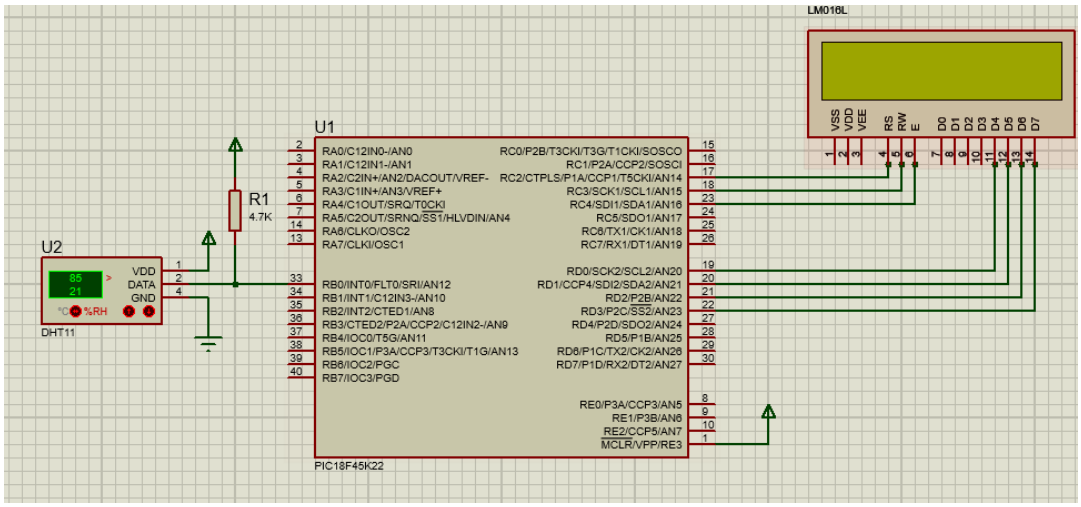
As an example for further illustration, we managed to find a picture that summarizes all the calculation in a simple example.



Source: [Electronics Gate]

VIII. Schematics in PROTEUS

The hardware setup includes connecting the DHT11 sensor to the MCU, specifically using the data pin of the DHT11 to an input pin on the MCU. The LCD display is connected to the MCU using a 4-bit parallel interface. Below is the schematic diagram created in PROTEUS:



Source: ~/Proteus/MicroProject.pdsprj

IX. Software Implementation

When writing the Main Class code, we covered four main ideas.

1) **Initialize the LCD and configure the MCU ports.**
2) **Send a start signal to the DHT11 and wait for the sensor's response.**
3) **Read the data from the sensor and validate the checksum.**
4) **Format the temperature and humidity data and display it on the LCD.**

Code Breakdown:

```
#include <p18cxxx.h>                                    // Include the PIC18 MCU library
#include <LCD4PICPROTO.h>                               // Include the LCD library for PIC microcontrollers

#define timeMultiplier    2
#define isSignalValid     FLAGS.B0

// Function initialization
void initialize(void);
void activateSignal(void);
void validateSignal(void);
char retrieveData(void);


// Strings to display temperature and humidity
char temperatureStr[]   = "Temp: 00.00  ";
char humidityStr[]      = "Hum:  00.00  ";

// Variables to store data
unsigned char index, dataSignal[5], dataChecksum;

void main(void) {
   initialize();                                        // Initialize the system

   while(1) {                                           // Infinite loop
      activateSignal();                                 // Activate the signal
      validateSignal();                                 // Validate the signal

      if (isSignalValid) {                              // If the signal is valid
         for (index = 0; index < 5; index++)            // Loop to retrieve data
            dataSignal[index] = retrieveData();

         for (index = 0, dataChecksum = 0; index < 4; index++)   // Loop to calculate checksum
            dataChecksum += dataSignal[index];

         if (dataChecksum == dataSignal[4]) {           // If checksum is correct

            // Update temperature string
            temperatureStr[6] = dataSignal[2] / 10 + '0';
            temperatureStr[7] = dataSignal[2] % 10 + '0';
            temperatureStr[9] = dataSignal[3] / 10 + '0';
            temperatureStr[10] = dataSignal[3] / 10 + '0';
            temperatureStr[12] = 'C';

            // Update humidity string
            humidityStr[6] = dataSignal[0] / 10 + '0';
            humidityStr[7] = dataSignal[0] % 10 + '0';
            humidityStr[9] = dataSignal[1] / 10 + '0';
            humidityStr[10] = dataSignal[1] % 10 + '0';
            humidityStr[12] = '%';
```

```
            // Display temperature and humidity
            DispRamStr(Ln1Ch0, temperatureStr);
            DispRamStr(Ln2Ch0, humidityStr);

          } else {                                                  // If checksum is incorrect

            // Display error message
            DispRomStr(Ln1Ch0, (ROM *) "Checksum        ");
            DispRomStr(Ln2Ch0, (ROM *) "Mismatch        ");
          }
        } else {                                                    // If the signal is not valid

          // Display error message
          DispRomStr(Ln1Ch0, (ROM *) "DHT11 SENSOR    ");
          DispRomStr(Ln2Ch0, (ROM *) "DID NOT RESPOND ");
        }

        Delay10KTCYx(60 * timeMultiplier);                          // Delay for 60 * 10K = 600ms
    }
}

// Function to initialize the system
void initialize(void) {
    InitLCD();                                                      // Initialize the LCD
    ANSELBbits.ANSB0 = 0;                                           // Set ANSB0 bit to digital
    TRISBbits.TRISB0 = 1;                                           // Set TRISB0 bit to input
}

// Function to activate the signal
void activateSignal(void) {
    PORTBbits.RB0 = 0;                                              // Set RB0 bit to '0'
    TRISBbits.TRISB0 = 0;                                           // Set TRISB0 bit to output
    Delay1KTCYx(18 * timeMultiplier);                               // Delay for 18ms
    PORTBbits.RB0 = 1;                                              // Set RB0 bit to '1'
    Delay10TCYx(3 * timeMultiplier);                                // Delay for 30ms
    TRISBbits.TRISB0 = 1;                                           // Set TRISA0 bit to input
}

// Function to validate the signal
void validateSignal(void) {
    Delay10TCYx(4 * timeMultiplier);                                // Delay for 40us

    if (PORTBbits.RB0 == 0) {                                       // If RB0 bit is '0'
        Delay10TCYx(8 * timeMultiplier);                            // Delay for 80us
        isSignalValid = PORTBbits.RB0;                              // Update the signal valid flag
        Delay10TCYx(4 * timeMultiplier);                            // Delay for 40us
    }
}

// Function to retrieve data
char retrieveData(void) {
    char dataMessage, index;                                        // Variables to store data and index

    for (index = 0; index < 8; index++) {                           // Loop to retrieve data
        while (!PORTBbits.RB0);                                     // Wait until RB0 bit is '1'
        Delay10TCYx(3 * timeMultiplier);                            // Delay for 30us

        if (PORTBbits.RB0 == 0) {                                   // If RB0 bit is '0'
            dataMessage &= ~(1 << (7 - index));                     // Clear the bit in dataMessage

        } else {                                                    // If RB0 bit is '1'
            dataMessage |= (1 << (7 - index));                      // Set the bit in dataMessage
            while (PORTBbits.RB0);                                  // Wait until RB0 bit is '0'
        }
    }

    return dataMessage;
}
```

X. Conclusion

The project successfully achieved the goal of measuring and displaying relative humidity using a DHT11 sensor and an LCD. The data is accurately read from the sensor and displayed on the LCD screen. Future improvements could include adding temperature compensation for more accurate humidity readings and implementing data logging features to track humidity over time.

XI. References

- *Electronics Gate*
- *DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf (mouser.com))*
- *Basics of Interfacing DHT11/DHT22 Humidity and Temperature Sensor with MCU (ocfreaks.com)*