MINI PROJECT-1

TOPIC: Design a thermometer using DHT11 & Arduino UNO to display the temperature in °F.

1.1 INTRODUCTION

Temperature measurement is a fundamental requirement in various fields such as meteorology, environmental monitoring, and many industrial applications. In this mini-project, we design a digital thermometer using the DHT11 temperature and humidity sensor in conjunction with the Arduino UNO microcontroller. The primary goal is to measure ambient temperature accurately and display the temperature readings in degrees Fahrenheit on the serial monitor of the Arduino IDE.

The DHT11 sensor is chosen for its simplicity, ease of use, and cost-effectiveness. It provides digital output directly, which eliminates the need for complex analog-to-digital conversion circuitry. The Arduino UNO, a widely used microcontroller board based on the ATmega328P, serves as the brain of this project, processing the sensor data and managing serial communication.

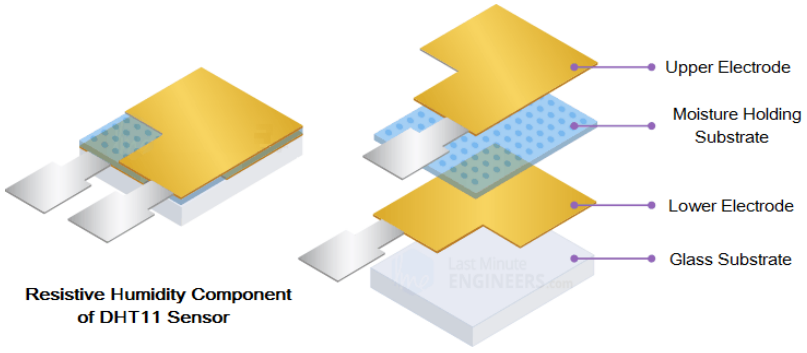
This project aims to provide hands-on experience with sensor integration, data processing, and serial communication using Arduino.

The DHT-11 sensor

DHT-11 is a basic digital temperature and humidity sensor. The sensor comes pre-calibrated and requires no external circuit for measuring the temperature or humidity.

For sensing humidity, DHT-11 has a resistive component that has two electrodes with a moisture-holding substrate between them. When the substrate absorbs water vapours, it releases ions that increase the conductivity between the electrodes.

When there's higher relative humidity, the resistance between the electrodes is reduced, and when there's a lower relative humidity, the resistance between electrodes is increased. This change in resistance is proportional to the relative humidity.

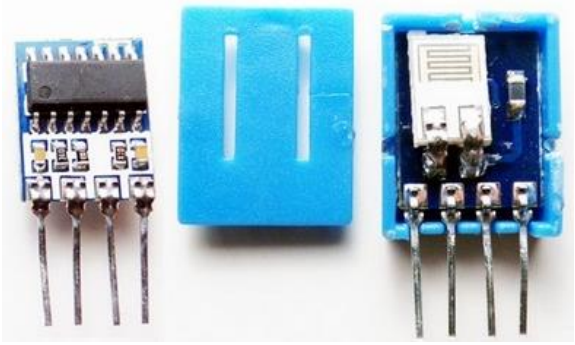


For sensing temperature, the DHT-11 has an NTC Thermistor, which is a thermal resistor. The thermistor contained in the DHT11 has a negative temperature coefficient, so its resistance decreases with increases in temperature.

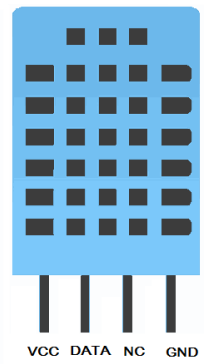
NTC Thermistor

The sensor contains a 14-pin, 8-bit microcontroller IC that:

- Senses analog signals from the humidity-resistive component and the thermistor
- Converts analog voltages to digital values, according to the stored calibration coefficients
- Outputs a digital signal carrying values of humidity, temperature, and a checksum byte



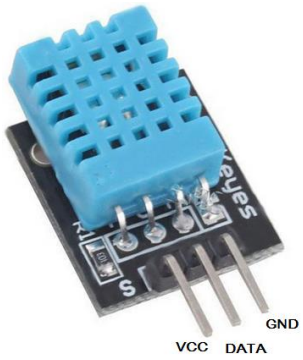
The DHT-11 sensor reads te relative humidity value in a percentage range from 20 to 90% RH. It senses temperature in degrees centigrade, ranging from 0˚ to 50˚C.



DHT-11 Sensor Pinout

The DHT11 sensor module

A DHT11 sensor module is also available. This module has a built-in supporting circuitry, meaning the sensor can be interfaced without additional circuits. The module has a 10 KΩ pull-up resistor between the VCC and DATA pin, and a decoupling capacitor on the power supply for filtering noise. The module has only three pins: VCC, DATA, and Ground.



DHT11 Sensor Module

To support the circuit, the sensor module must ensure proper communication with a controller. If the sensor is used, an external pull-up resistor can be employed. The GPIO of microcontroller boards has built-in, pull-up and pull-down resistors. This ensures the DATA pin of the DHT11 sensor can be directly interfaced with a microcontroller pin that's configured to use an internal pull-up.

1.2 LITERATURE SURVEY

Temperature sensors are crucial components in various applications, from industrial processes to consumer electronics. They provide accurate and reliable measurements necessary for monitoring and controlling environments. Common temperature sensors include LM35, DS18B20, and DHT11, each with unique characteristics and use cases.

1. **LM35 Temperature Sensor:**

   The LM35 is a popular analog temperature sensor that offers a linear voltage output directly proportional to the temperature in Celsius. It is known for its high accuracy (±0.5°C) and simplicity. However, being an analog sensor, it requires an analog-to-digital converter (ADC) to interface with digital microcontrollers like the Arduino. This can introduce complexity and the need for calibration to ensure accurate readings [1]. LM35 sensors are commonly used in scenarios where accurate temperature measurements are crucial, such as laboratory environments and precision climate control systems [1].

2. **DS18B20 Temperature Sensor:**

   The DS18B20 is a digital temperature sensor that uses the 1-Wire communication protocol, allowing multiple sensors to be connected to a single data line. It provides temperature data in a digital format, which simplifies interfacing with microcontrollers. The DS18B20 offers higher accuracy (±0.5°C over a range of -10°C to +85°C) and a wider temperature range than the DHT11. Its digital nature eliminates the need for an ADC, making it easier to use in digital systems. This sensor is particularly suitable for industrial applications where high accuracy is necessary and multiple sensors need to be monitored simultaneously [2].

3. **DHT11 Temperature and Humidity Sensor:**

   The DHT11 is a low-cost digital sensor that provides both temperature and humidity data. It is widely used in hobbyist and educational projects due to its affordability and ease of use. The DHT11 communicates via a single-wire protocol, which simplifies connections to microcontrollers. However, it has a limited accuracy of ±2°C and a narrower operating temperature range (0°C to 50°C) compared to sensors like the DS18B20. Despite these limitations, the DHT11 is suitable for basic applications where high precision is not critical[3].

Comparison and Relevance to the Project:

While the LM35 and DS18B20 are excellent choices for applications requiring high accuracy and a broad temperature range, the DHT11 was chosen for this project due to its simplicity and dual-functionality (temperature and humidity measurement). The DHT11's digital output simplifies interfacing with the Arduino UNO, as it does not require additional circuitry for analog-to-digital conversion. Moreover, the sensor's ease of use and availability make it a practical choice for educational purposes and introductory projects in embedded systems.

Challenges and Limitations:

Despite the widespread use of temperature sensors, several challenges must be addressed:

- **Accuracy:** Ensuring precise readings requires careful calibration and consideration of sensor placement.
- **Response Time:** The speed at which a sensor responds to temperature changes is crucial for real-time applications.
- **Environmental Factors:** Humidity, airflow, and other environmental conditions can affect sensor performance.

The DHT11's digital output simplifies interfacing with the Arduino UNO, as it does not require additional circuitry for analog-to-digital conversion. Moreover, the sensor's ease of use and availability make it a practical choice for educational purposes and introductory projects in embedded systems.

The DHT11 sensor, with its balance of cost, ease of use, and functionality, is an appropriate choice for this mini project. It provides a practical introduction to digital sensor interfacing and temperature measurement, laying the groundwork for more advanced projects. By using the Arduino UNO, students can focus on learning core concepts without the added complexity of analog signal processing.

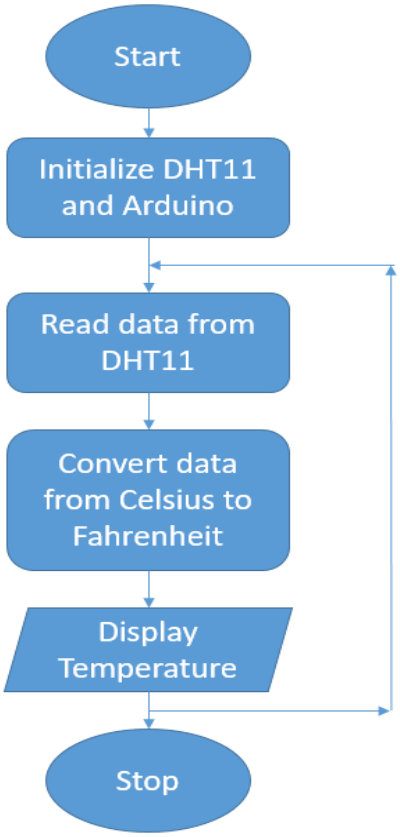
1.3 BLOCK DIAGRAM

1.4 FLOWCHART/ALGORITHM

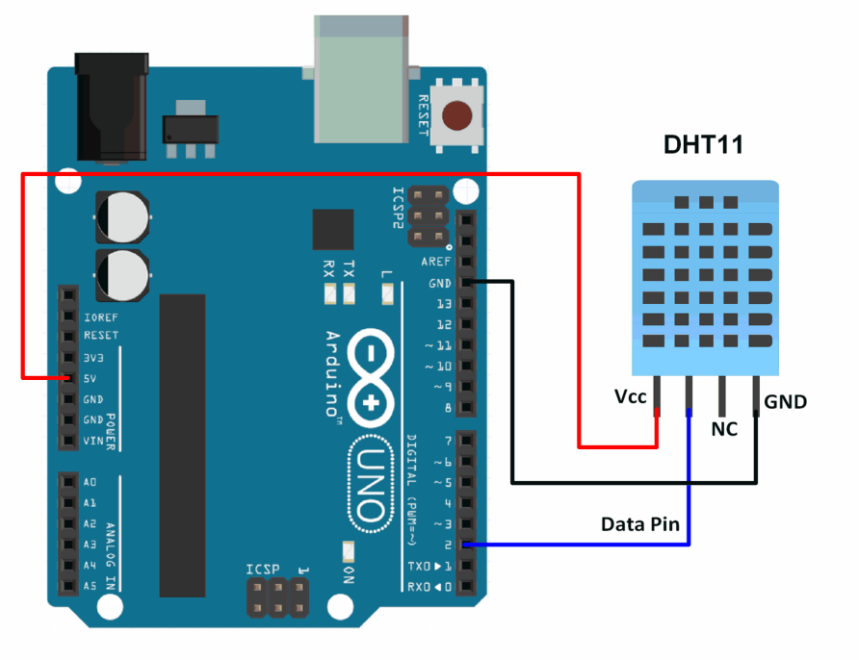
Algorithm:

1. Set up serial communication for displaying data on the serial monitor.
2. Set the DHT11 sensor data pin as input.
3. Send the start signal to the DHT11 sensor.
4. Read the raw data bits from the sensor.
5. Extract temperature data from the raw bits.
6. Convert the temperature from Celsius to Fahrenheit.
7. Display the temperature in Fahrenheit on the serial monitor.
8. Repeat the process at regular intervals.

Flowchart:



1.5 CIRCUIT DIAGRAM



1.6 COMPONENT LIST

- Arduino UNO
- DHT11 Temperature and Humidity Sensor
- Breadboard
- Jumper wires
- USB cable (for Arduino power and programming)

1.7 DETAILED DESCRIPTION

The DHT11 sensor is connected to the Arduino UNO, which reads the temperature data in Celsius. The Arduino then converts this data to Fahrenheit using the formula $T(°F)=T(°C)\times1.8+32$. The converted temperature is displayed on the serial monitor of the Arduino IDE. This process is done without using any predefined libraries for the DHT11 sensor. Instead, the communication protocol and data parsing are implemented manually in the Arduino code.

Interfacing DHT-11 with Arduino

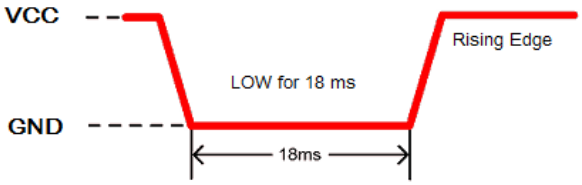
The DHT-11 sensor can be directly interfaced with Arduino. It can be provided 5V DC and ground from Arduino. The sensor's data pin can also be directly connected to any of Arduino's digital I/O pins. But the digital I/O pin to which it's interfaced must be configured to use an internal pull-up. Otherwise, an external pull-up resistor of 10K must be connected between VCC and DATA pin of DHT-11. Alternatively, the DHT-11 sensor module can be used.

How DHT11 works

The DHT-11 sensor uses a one-wire protocol to communicate the humidity and temperature values. The sensor act as a slave to a host controller.

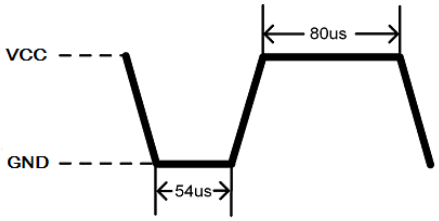
The digital communication between the DHT11 and the host controller (like Arduino) can be breakdown into four steps:

1. **Start signal.** To start communicating with the DHT11 sensor, the host (Arduino) must send a start signal to the DATA pin of the DHT11 sensor. The DATA pin is then pulled HIGH by the default. The start signal is a logical LOW for 18 milliseconds, which is followed by a LOW-to-HIGH transition (rising edge).



Start Signal from Host to DHT11 Sensor

2. **Response.** After receiving a start signal from the host, DHT-11 sends a response signal to indicate that it's ready to transmit the sensor data. The response pulse is a logical LOW for 54 microseconds and then a logical HIGH for 80 microseconds.



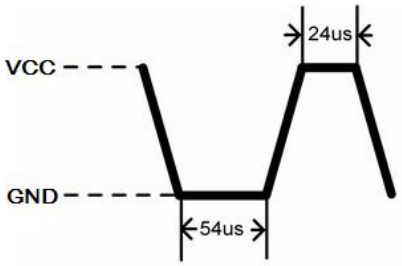
Response Signal from DHT11 to Host

3. **Data.** After sending a response pulse, DHT11 will begin transmitting sensor data containing the values of humidity, temperature, and checksum byte. The data packet consists of 40 bits, with five 8-bit segments or bytes.

   The first two bytes contain the value of relative humidity, whereby the first byte is an integral part of the humidity value and the second byte is a decimal part of the humidity value. The next two bytes contain the value of temperature, whereby the third byte is an integral part of the temperature value and the fourth is a decimal part of the temperature value.

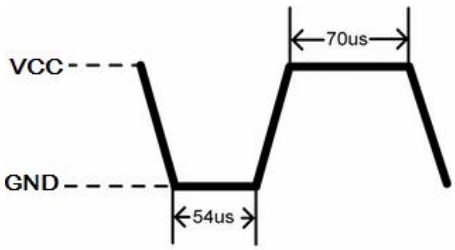
   The last byte is a checksum byte, which must be equal to the binary sum of the first four bytes. If the checksum byte is not equal to the binary sum of the humidity and temperature values, there's an error in the values.

The bits are transmitted as timing signals, where the pulse width of the digital signal determines whether it's bit 1 or bit 0. Bit 0 starts with a logical LOW signal (Ground) for 54 microseconds, followed by a logical HIGH signal (VCC) for 24 microseconds.



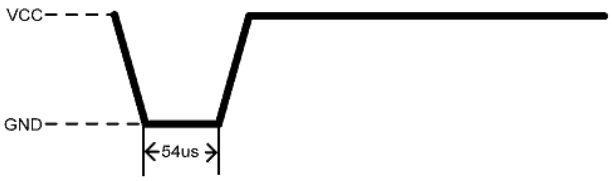
DHT11 Bit 0

Bit 1 starts with a logical LOW signal (ground) for 54 microseconds, followed by a logical HIGH signal (VCC) for 70 microseconds.



DHT11 Bit 1

4. **End of the frame.** After transmitting the 40-bit data packet, the sensor sends a logical LOW for 54 microseconds and then will pull HIGH on the data pin. After this, the sensor goes into a low-power consumption sleep mode. The data from the sensor can be sampled at a rate of 1 Hz or once every second.



DHT11 End of Data Packet

Reading sensor data from DHT11

To read sensor data from the DHT11 sensor, Arduino must first send it the start signal. To do so, the pin that DHT11's DATA pin is interfaced with must be set to a digital output. A digital pulse of 18 milliseconds must be passed to the DATA pin, followed by a rising edge. Immediately afterward, the Arduino pin must be set to a digital input with an internal pull-up.

Now, read the response signal from the DHT11 at the Arduino pin. If a falling edge is detected within 90 microseconds, this means that DHT11 has successfully sent a response pulse.

The data received from the DHT11 sensor can be sampled by polling the logical level of the digital pulse while tracking the pulse width. It's possible to track the pulse width by measuring the time elapsed from a particular instant of time, while polling for a logical HIGH or LOW.

There are millis() and micros() functions available that track the time elapsed since Arduino boots. The millis() function provides the time from the boot in milliseconds and the micros() function provides the time from the boot in microseconds.

Arduino Serial Monitor

Arduino IDE has an integrated serial monitor that can be used to receive and send serial data via a desktop's USB ports. The serial monitor can be opened by navigating to Tools->Serial Monitor.

In this project, we will send sensor data that's read from the DHT11 sensor to a desktop computer via Arduino UNO. In this case, the DHT11 will be interfaced with Arduino, and Arduino will be connected to the desktop computer via a USB cable. The Arduino is programmed so that the sensor data is observed on Arduino IDE's Serial Monitor.

How the project works

The data pin of DHT11 is connected to Arduino UNO's pin 2. Arduino is programmed to send a start signal to the DHT11 sensor, and then to read the response pulse and serial data from the sensor.

For detecting the response signal and reading serial data from DHT11, the input digit signal at the Arduino pin is polled and the pulse width for the TON and TOFF is determined. This is done by comparing the time elapsed for a logical HIGH input after the rising edge of the signal and the time elapsed for a logical LOW input after the falling edge of the signal at the Arduino pin.

By measuring the pulse width of the TON signal of the response signal, it determines if the start pulse has been successfully applied to DHT11's data pin. If the pulse width of the TON after the rising edge in response signal does not exceed 90 microseconds, this means the start signal has been successfully applied to DHT11's DATA pin. Otherwise, there will be a TIMEOUT_ERROR when applying the start signal to DHT11's data pin.

After receiving the response pulse from DHT11, the serial data from it is read at the Arduino pin. Bits 0 and 1 are detected by measuring the pulse width after the rising edge of the signal. If the pulse width exceeds 30 microseconds, it's bit 1. Otherwise, it's bit 0.

The bits are stored in a 16-bit variable and transferred to other 16-bit variables for the humidity and temperature values, and to an 8-bit variable for checksum byte.

The read bits — with the humidity and temperature value, checksum byte, and error codes — are sent to the serial port in this format.

Arduino Code:

```
void setup()
{
  Serial.begin(9600);
}

void dec2bin(int n)
{
  int c, k;

  for (c = 15; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      Serial.print("1");
    else
      Serial.print("0");
  }
}

void dec2bin8(int n)
{
  int c, k;

  for (c = 7; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      Serial.print("1");
    else
      Serial.print("0");
  }
}

void wait_for_dht11()
{
  delay(2000);
}

void start_signal(uint8_t dht11_pin)
{
  pinMode(dht11_pin, OUTPUT);
  digitalWrite(dht11_pin, LOW);
  delay(18);
  digitalWrite(dht11_pin, HIGH);
```

```
  pinMode(dht11_pin, INPUT);
  digitalWrite(dht11_pin, HIGH);
}

void read_dht11(uint8_t dht11_pin)
{
  uint16_t rawHumidity = 0;
  uint16_t rawTemperature = 0;
  uint8_t checkSum = 0;
  uint16_t data = 0;

  uint8_t humi;
  uint8_t humd;
  uint8_t tempi;
  uint8_t tempd;

  unsigned long startTime;

  for (int8_t i = -3; i < 80; i++)
  {
    byte live;
    startTime = micros();

    do
    {
      live = (unsigned long)(micros() - startTime);
      if (live > 90)
      {
        Serial.println("ERROR_TIMEOUT");
        return;
      }
    } while (digitalRead(dht11_pin) == (i & 1) ? HIGH : LOW);

    if (i >= 0 && (i & 1))
    {
      data <<= 1;

      // TON of bit 0 is maximum 30 usecs and of bit 1 is at least 68 usecs.
      if (live > 30)
      {
        data |= 1; // we got a one
      }
    }

    switch (i)
    {
    case 31:
      rawHumidity = data;
```

```
      break;
    case 63:
      rawTemperature = data;
    case 79:
      checkSum = data;
      data = 0;
      break;
    }
  }

  Serial.println("Temperature Raw Data: ");
  dec2bin(rawTemperature);
  Serial.print("\t");
  tempi = rawTemperature >> 8;
  dec2bin8(tempi);
  Serial.print("\t");
  rawTemperature = rawTemperature << 8;
  tempd = rawTemperature >> 8;
  dec2bin8(tempd);
  Serial.println("");
  Serial.print("Temperature Degree Celcius: ");
  Serial.print(tempi);
  Serial.print(".");
  Serial.print(tempd);
  Serial.print("C");
  Serial.println("");
  float x = tempi + 0.1 * tempd;
  float temp_f = x * (9.0 / 5.0) + 32.0;
  Serial.print("Temperature Degree Farenheit: ");
  Serial.print(temp_f);

  Serial.println("");
  Serial.println("");
  Serial.println("");
}

void loop()
{
  for (unsigned int x = 0; x < 1000; x++)
  {
    wait_for_dht11();
    start_signal(2);
    read_dht11(2);
  }
  Serial.end();
}
```

Programming guide:

The Arduino sketch begins with the setup() function, where the baud rate for the serial communication is set to 9600 bps.

Then, the following user-defined functions are written:

dec2bin(int n) – converts a 16-bit integer to its binary representation. It loops through a 16-bit integer value bit-by-bit right, shifting one bit each time and masking it with 1 (using & operator) to determine if that particular bit in the integer value is 1 or 0. This function returns nothing, but serially prints the binary representation of the argument passed to it. It's used to convert the read humidity and temperature values to their binary representations.

dec2bin8(int n) – converts an 8-bit integer to its binary representation. It loops through the 8-bit integer value bit-by-bit right, shifting one bit each time and masking it with 1 (using & operator) to determine if that particular bit in the integer value is 1 or 0. The function returns nothing, but serially prints the binary representation of the argument passed to it. It's used to convert the read checksum value, the integral and decimal parts of humidity value, and the integral and decimal parts of temperature value to their binary representations.

wait_for_dht11() – provides a delay of two seconds. This time is required to get DHT11 ready for data transmission after its host (Arduino) boots. If this delay is not provided, the DHT11 sensor will not respond to the host controller and, in initial attempts to read the sensor data from DHT11, there will be a timeout error.

```
void wait_for_dht11()
{
        delay(2000);
}
```

start_signal(uint8_t dht11_pin) – generates the start signal for the DHT11 sensor. This function takes the pin where DHT11's DATA pin is interfaced as an argument. The pin is first set as digital output using the pinMode() function. It's cleared (LOW) for 18 milliseconds using the digitalWrite() and the delay() functions. Then, it's set (HIGH) to provide the rising edge of the digital signal. Immediately afterward, the pin is set as digital input and it's pulled HIGH.

```
void start_signal(uint8_t dht11_pin)
{
        pinMode(dht11_pin, OUTPUT);
        digitalWrite(dht11_pin, LOW);
```

```
        delay(18);
        digitalWrite(dht11_pin, HIGH);
        pinMode(dht11_pin, INPUT);
        digitalWrite(dht11_pin, HIGH);
}
```

read_dht11(uint8_t dht11_pin) – used to read sensor data from DHT11. It takes the pin where DHT11's DATA pin is connected as an argument. First, the variables are defined to store the:

- Raw humidity value (containing both integral and decimal parts)
- Raw temperature value (containing both integral and decimal parts)
- Checksum byte
- Bit stream from the DHT11 sensor (variable 'data')
- Integral part of the humidity value
- Decimal part of the humidity value
- Integral part of the temperature value
- Decimal part of the temperature value
- Variable to store instant of time at the rising or falling edge of the signal

```
void read_dht11(uint8_t dht11_pin)
{
        uint16_t rawHumidity = 0;
        uint16_t rawTemperature = 0;
        uint8_t checkSum = 0;
        uint16_t data = 0;
        uint8_t humi;
        uint8_t humd;
        uint8_t tempi;
        uint8_t tempd;
        unsigned long startTime;
```

A for-loop is run to detect DHT11's response signal and bit stream. A variable 'live' is declared to store the TON time and the current instant of time is stored in the variable 'startTime.'

```
for ( int8_t i = -3 ; i < 80; i++ ) {
   byte live;
   startTime = micros();
```

In a do-while loop, the response signal will be detected. It's determined if the digital signal at Arduino pin is HIGH as a condition for the loop. While the signal remains HIGH, the elapsed time from the last measured time instant is polled.

If it's greater than 90 microseconds, the start signal was not properly applied to the DHT11 DATA pin. Otherwise, if there is a falling edge causing the while loop to exit before 90 microseconds, the response signal has been successfully received from DHT11.

Now, we are ready to detect the bit stream from DHT11.

```
do {
    live = (unsigned long)(micros() – startTime);
    if ( live > 90 ) {
      Serial.println("ERROR_TIMEOUT");
      return;
    }
  }while ( digitalRead(dht11_pin) == (i & 1) ? HIGH : LOW );
```

The for-loop has already run four times in an attempt to detect the response signal. The Arduino pin is read LOW after exiting the previous do-while loop due to a falling edge. Now we can start reading the bit stream. When the loop counter is greater or equal to 0 and is even (like 0, 2, 4, 6, 8, etc.) when it is masked with 1 (& operation), the following if the condition will be false. So, do nothing when there is the TOFF of the digital pulse.

```
if ( i >= 0 && (i & 1) ) {
```

When the loop counter is odd, the if condition will be true. Check if the time elapsed (for the TON of the digital pulse) is greater than 30 microseconds. If so, left shift the variable storing the bit stream and append 1. Otherwise, simply left shift the variable storing bit 0.

```
data <<= 1;
    // TON of bit 0 is maximum 30 usecs and of bit 1 is at least 68 usecs.
    if ( live > 30 ) {
      data |= 1; // we got a one
    }
  }
```

When the loop counter has run 31 times since detecting the bit stream, which means 16 bits have been read, store that bit stream to the raw humidity value. When the loop counter has run 63 times since detecting bit stream, so the next 16 bits have been read, store that bit stream to the raw temperature value.

When the loop counter has run 79 times since detecting a bit stream, and 8 bits have been read, store that bit stream to checksum byte.

```
switch ( i ) {
    case 31:
      rawHumidity = data;
      break;
    case 63:
      rawTemperature = data;
    case 79:
      checkSum = data;
      data = 0;
      break;
  }
}
```

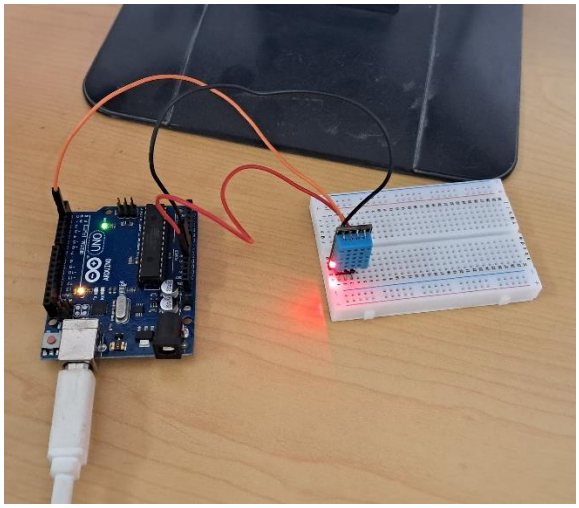
Now that we have the humidity value, temperature value, and checksum byte, transfer them to the serial port in the format stated above. In this project as we are required to make a thermometer, we will only print Temperature Data Only

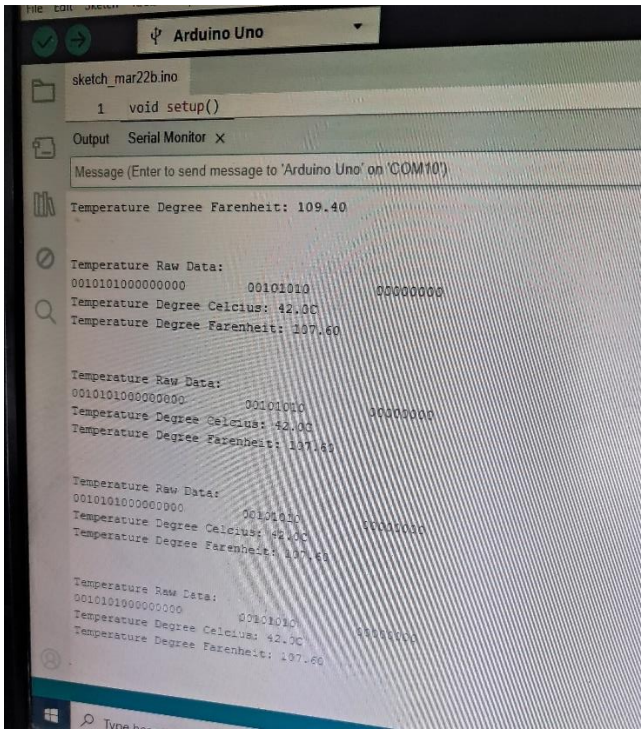
```
Serial.println("Temperature Raw Data: ");
dec2bin(rawTemperature);
Serial.print("\t");
tempi = rawTemperature >> 8;
dec2bin8(tempi);
Serial.print("\t");
rawTemperature = rawTemperature << 8;
tempd = rawTemperature >> 8;
dec2bin8(tempd);
Serial.println("");
Serial.print("Temperature Degree Celcius: ");
Serial.print(tempi);
Serial.print(".");
Serial.print(tempd);
Serial.print("C");
Serial.println("");
```

```
  float x = tempi + 0.1 * tempd;
  float temp_f = x * (9.0 / 5.0) + 32.0;
  Serial.print("Temperature Degree Farenheit: ");
  Serial.print(temp_f);
  Serial.println("");
  Serial.println("");
  Serial.println("");
}
```

In the loop() function, the wait_for_dht11(), start_signal() and read_dht11() functions are executed 1000 times in the same sequence. So, the sensor data is read five times from the DHT11 sensor. The serial communication is then closed using the Serial.end() method.

1.8 RESULT AND ANALYSIS





- **Output:** The thermometer successfully measures and displays the ambient temperature in Fahrenheit on the serial monitor. This output confirms that the system correctly reads the temperature data from the DHT11 sensor and converts it to Fahrenheit. For testing we varied the temperature using different items: ice cubes for lower temperatures, resulting in a lowest reading of 68°F; room temperature, which was approximately 82.40°F; and a soldering iron for higher temperatures, giving a highest reading of 113°F.

- **Accuracy:** We used a thermometer to compare the actual temperature and the temperature output of the DHT11 sensor. The temperature readings are accurate within ±2°F, which is within the DHT11 sensor's specified range (0°F - 122°F). This level of accuracy is adequate for basic temperature monitoring applications, such as home environment monitoring or educational projects.
- **Observations:**
  - Response Time: The response time is satisfactory for real-time temperature monitoring. The DHT11 sensor and Arduino system update the temperature reading every two seconds, which is sufficient for most practical applications where rapid temperature changes are not expected.
  - Reliability: The system consistently produces reliable temperature readings without significant fluctuations, demonstrating stable performance over multiple test cycles.
  - User Interface: Displaying the temperature on the serial monitor of the Arduino IDE provides a simple and effective user interface for observing temperature changes in real-time.

These results indicate that the design and implementation of the thermometer using the DHT11 sensor and Arduino UNO are successful, meeting the project's objectives. The system's performance aligns with the expected specifications and demonstrates the feasibility of using Arduino and DHT11 for basic temperature monitoring tasks.

1.9 CONCLUSION

The project effectively demonstrates the use of the DHT11 sensor and Arduino UNO for creating a simple digital thermometer. The system is capable of providing real-time temperature readings in Fahrenheit with reasonable accuracy. The design and implementation process, which included writing custom code for sensor communication and data parsing, highlights the versatility and capability of the Arduino platform for handling sensor data.

Key Achievements:

- **Accurate Temperature Readings:** The thermometer provides reliable temperature readings within the accuracy range of the DHT11 sensor (±2°F), making it suitable for basic monitoring applications.
- **Real-time Monitoring:** The system updates temperature readings every two seconds, ensuring that users can monitor ambient temperature changes in real-time.

- **Educational Value:** By implementing the DHT11 sensor communication without relying on predefined libraries, the project offers valuable insights into the fundamentals of digital sensor interfacing and data handling in embedded systems.
- **Simplicity and Cost-effectiveness:** The use of readily available components like the DHT11 sensor and Arduino UNO ensures that the project is both affordable and easy to replicate, making it accessible for educational purposes and hobbyist projects.

This project serves as a foundation for more complex sensor-based applications, showcasing how basic components can be utilized effectively to create functional and practical solutions. The skills and knowledge gained from this project are transferable to various other projects in the fields of embedded systems and IoT (Internet of Things). Overall, the successful completion of this project underscores the potential of Arduino-based systems in educational settings and their applicability in real-world scenarios.

1.10 REFERENCES

1. "Measurement of Temperature with Sensor LM35", ResearchGate, [https://www.researchgate.net/publication/354598620_Title_MEASUREMENT_OF_TEMPERATURE_WITH_SENSOR_LM35_Introduction]
2. "Exploring One-wire Temperature sensor DS18B20 with Microcontrollers", ResearchGate, [https://www.researchgate.net/publication/330854061_Exploring_One-wire_Temperature_sensor_DS18B20_with_Microcontrollers]
3. https://www.engineersgarage.com/articles-arduino-dht11-humidity-temperature-sensor-interfacing/#:~:text=To%20read%20sensor%20data%20from,set%20to%20a%20digital%20output
4. https://www.sparkfun.com/products/retired/21240
5. https://www.arduino.cc/en/Guide
6. https://docs.arduino.cc/built-in-examples/basics/DigitalReadSerial/