International Institute of Information Technology

Introduction to IoT, Spring 2020
Lab 2

Overview: In this lab session, you will be interfacing a DHT11 temperature-humidity sensor with the nodemcu and write the outputs to the serial monitor. Once that is done, you will be working to communicate the temperature-humidity values from the nodemcu with the sensor to the arduino UNO using I2C interface.

Part 1) DHT11 Temperature & Humidity sensor on NodeMCU using Arduino IDE

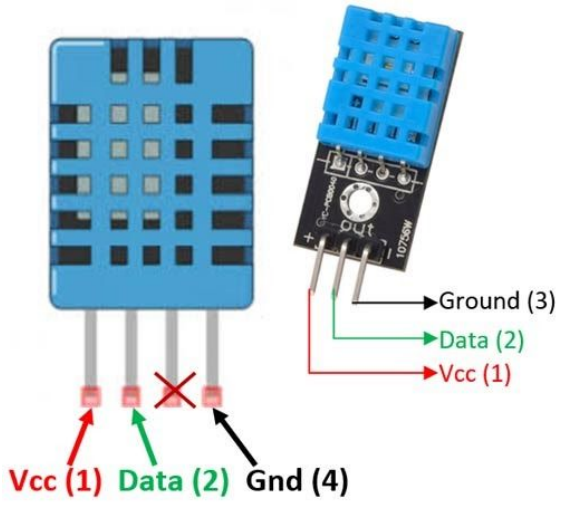
a) Introduction and understanding the DHT11 sensor:

i) The DHT11 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data.

ii) The DHT11 is chosen because it is lab calibrated, accurate and stable and its signal output is digital.

iii) Most important of all, it is relatively inexpensive for the given performance.

iv) Below is the pinout diagram of the DHT11 sensor



v) DHT11 Specifications:

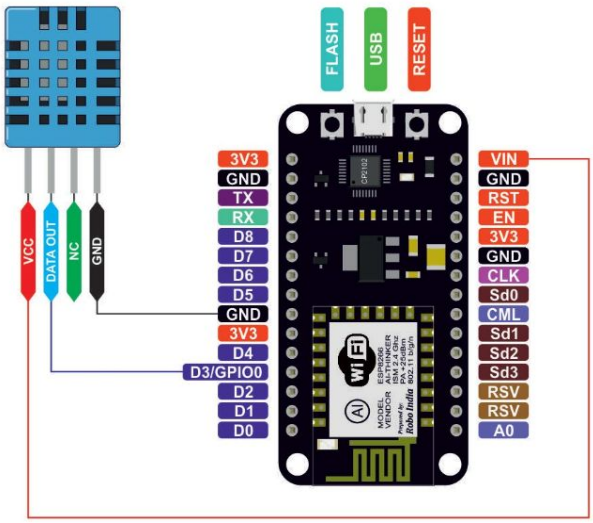
1) Operating Voltage: 3.5V to 5.5V
2) Operating current: 0.3mA (measuring) 60uA (standby)
3) Output: Serial data
4) Temperature Range: 0°C to 50°C
5) Humidity Range: 20% to 90%
6) Resolution: Temperature and Humidity both are 16-bit
7) Accuracy: ±1°C and ±1%

vi) Please refer to the data sheet for more information about the DHT11 sensor. Link to the data sheet here.

b) Components required for this exercise:

i) DHT11 sensor
ii) nodemcu
iii) Breadboard
iv) Few jumper wires

c) Circuit Diagram:



d) **Library Files:** Following two libraries will be required to run this code. Download the zip file, extract the same and copy this to your Arduino library folder.
   i) **Library 1**: You may download library file from here.
   ii) **Library 2**: You may download library file from here.

e) **Pseudocode**:
   i) Include the library of the DHT11 temperature and humidity sensor using #include “DHT.h”
   ii) Define the DHTtype and digital pin, in this case type is DHT11 and digital pin is GPIO 0/D3 using #define DHTTYPE and #define dht_dpin.
   iii) Instantiate the DHT dht(dht_dpin, DHTTYPE)
   iv) In the setup block, use dht.begin(), Serial.begin(<baud rate>) to begin the serial communication. Use delay(<delay in milliseconds>) to give an appropriate delay.
   v) In the loop block, use float variables to read the temperature and humidity values from the sensor using dht.readHumidity(), dht.readTemperature(); Serial.print()/Serial.println() to print the values on the serial monitor and give an appropriate delay using delay(<delay in milliseconds>) according to the frequency of sensor.
   vi) You can refer to the example codes in the library you already downloaded for the syntaxes as well as some additional points.

f) **Expected output**:
   i) The output values should be the temperature and humidity values measured by the DHT11 sensor at each instant it is sensing corresponding to your delay.
   ii) Compare your values with the room temperature and verify if the output temperature value is not too far from it, else your sensor is not working.
   iii) Try playing around with the sensor a little like blowing on to it etc. which should increase the temperature and verify.

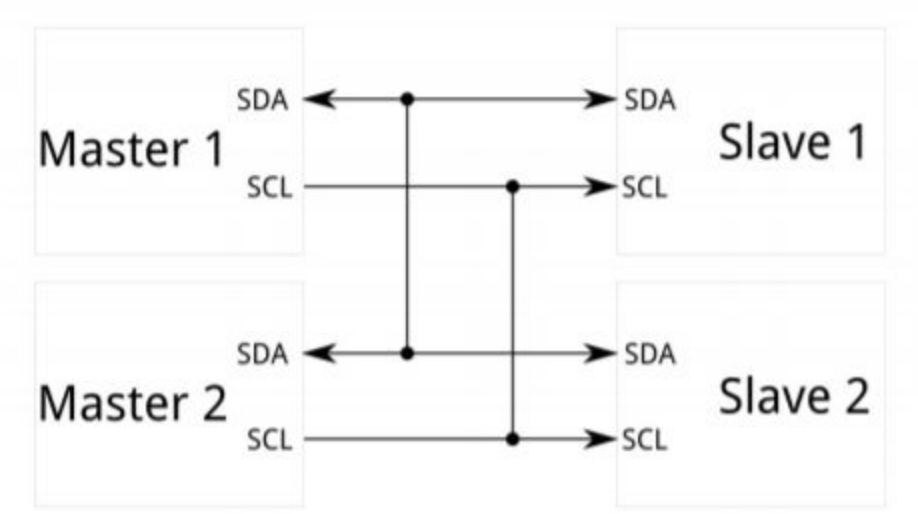
Note: The above tutorial is available on the net. Please refrain from copying the code directly, if anyone is seen copying from the net blindly, he/she will be penalized.

Part 2) Serial Communication between the arduino and nodemcu

a) **Introduction:**

i) We will be using the I2C communication protocol for establishing a serial communication between the arduino and the nodemcu, so it is important to learn about the I2C communication protocol first.

ii) I2C- The Inter-integrated Circuit is a synchronous, two-wire protocol which allows multiple slave and master devices on the network.



iii) It is only intended for short range communication and only requires two signal wires to exchange information. SDA: Serial Data, SCL: Serial Clock.

iv) Communication via I2C is more complex than UART or SPI. The signalling must adhere to a certain protocol for the devices on the bus to recognize it as valid I2C communication.

v) **Procedure of I2C communication protocol:** First up, the messages are broken up into two types of frames: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Note that each slave on the network has a unique 7-bit address. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high.

1) **Start Condition:** To initiate the address frame, the master leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start.

2) **Address Frame:** The address frame is always first in any new communication sequence. For a 7-bit address, the address is clocked out with MSB first, followed by a R/W bit indicating whether this is a read (1) or a write (0) operation. The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits (address + r/w) of the frame are sent, the slave with the same address is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message.
3) **Data frame:** After the address frame has been sent, data transmission begins. The master will simply continue to generate clock pulses at regular intervals and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation.
4) **Stop condition:** Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a low to high transition on SDA after a low to high transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should not change when SCL is high, to avoid false stop conditions.

vi) **The I2C Wire library in Arduino:** The A4 pin of Arduino performs an alternate function of I2C SDA and A5 pin, I2C SCL. Similarly in the nodemcu D1 performs alternate function of I2C SCL and D2 pin, I2C SDA The I2C protocol is supported natively by the Arduino with header. The following table lists some common functions of the Wire library -

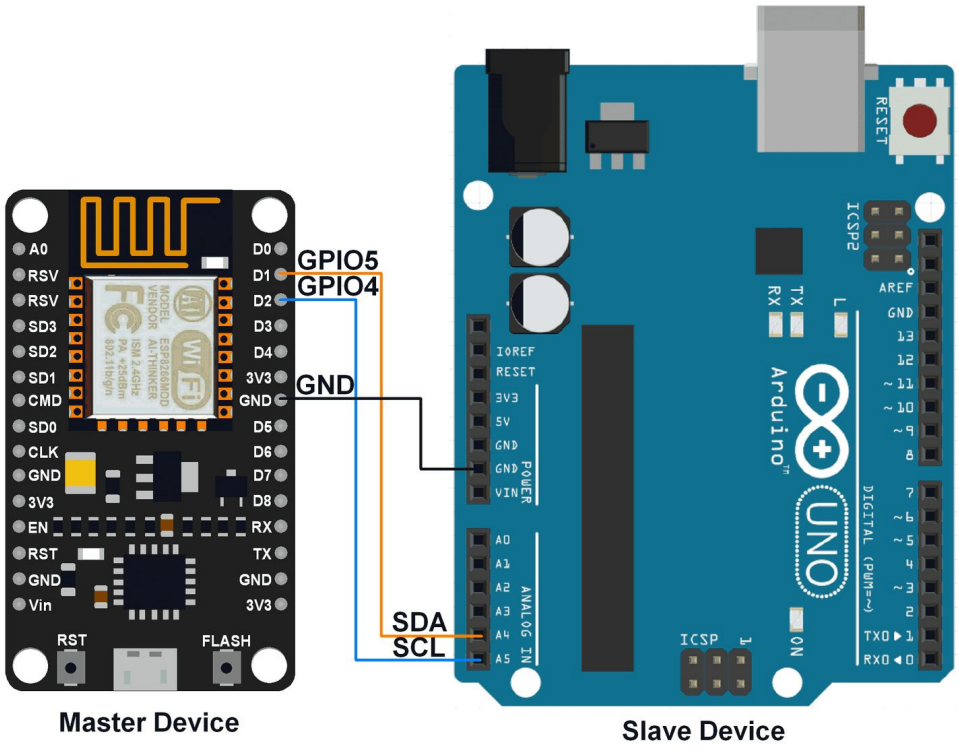
Function	Description
Wire.begin(address)	Joins the I2C bus as a master if address is not given. If an address is specified as a parameter, the Arduino joins the bus as a slave with that address. This function is called in the setup.
Wire.beginTransmission(address)	This is for Arduino configured as I2C master: initiate transmission with the slave that has the given address.

Wire.write(data)	Writes data over I2C. Data can be a byte, integer or a floating point. For master configured in Write mode, this function is called after beginTransmission. For slave, this is called in onRequest function.
Wire.endTransmission()	Ends the current transmission.
Wire.onReceive(func_name)	Registers a function to be called when a slave device receives a transmission from a master.
Wire.read()	Used by slave to read bytes from buffer after master's transmission. Used by master to read bytes from buffer after calling requestFrom.
Wire.requestFrom(address, bytes)	Request a specified number of bytes from the slave with the given address. This changes the Write configuration of master to Read.
Wire.onRequest(func_name)	Register a function to be called when a master requests data from this slave device.
Wire.available()	Returns the number of bytes available for retrieval with read(). This should be called on a master device after a call to requestFrom() or on a slave inside the onReceive() handler.

b) Components required:

ArduinoUNO
NodeMCU
DHT11 sensor
Breadboard
Few jumper wires

c) **Circuit Diagram:** Make these new connections apart from the earlier ones in part I. (Make these new connections to the already existing circuit)



d) **Pseudocode:**

i) **For nodemcu (master device)**

1) Include the Wire.h library and DHT library using #include <Wire.h>,#include "DHT.h"
2) Define the DHTtype and digital pin, in this case type is DHT11 and digital pin is GPIO 0/D3 using #define DHTTYPE and #define dht_dpin.
3) Instantiate the DHT dht(dht_dpin, DHTTYPE)
4) In the setup block, use dht.begin(), Serial.begin(<baud rate>) to begin the serial communication. Use Wire.begin(D1, D2) to join I2C bus of nodemcu. Use delay(<delay in milliseconds>) to give an appropriate delay.
5) In the loop block, use float variables to read the temperature and humidity values from the sensor using dht.readHumidity(), dht.readTemperature(); since Wire.write() can only transmit data one byte at a time, convert your float values into strings by using

dtostrf function of the standard C library. To use it, declare a character array for each of the float variable you wish to transmit and then dtostrf(floatvar, StringLengthIncDecimalPoint, numVarsAfterDecimal, charbuf); where floatvar = float variable, StringLengthIncDecimalPoint = This is the length of the string that will be created, numVarsAfterDecimal = The number of digits after the decimal point to print, charbuf = the array to store the results. We want 2 digits after the decimal point.

6) Since 2 arrays can't be transmitted at the same time using Wire.write(), concatenate both the arrays of temperature and humidity into one single array. Make sure to use a separator like "," or " " to separate the data. This should be done in the loop block.
7) Use Wire.beginTransmission(<address>) to begin transmission with device address in the brackets. You can choose any device address in the device address range; Use Wire.write(<concatenated array>) to send the data over the I2C bus; Use Wire.endTransmission() to stop transmitting data for this cycle and finally give an appropriate delay using delay(<delay in milliseconds>) according to the frequency of sensor. This too should be done in the loop block.

ii) For Arduino (slave device)

1) Include the Wire.h library using #include <Wire.h>
2) Declare an empty char array to store the data which arrives from the master.
3) In the setup block, use Wire.begin(<address>) to join i2c bus with the address. Note that this address should be the same address which you used to begin transmission in the masters code; Use Wire.onReceive(receiveEvent) to register receive event; Use Serial.begin(<baud rate>) to start serial for debug.
4) In the loop block, just give a small delay for buffering.
5) Since you are calling the receiveEvent function in the setup block, we need to define that function now. Define the function using the pseudocode void receiveEvent(int howMany) {....}
6) Inside this function run a counter for each bit as it arrives. Update each element of the empty char array you initially declared in a while loop which runs till the statement Wire.available() is true. Use Wire.read() to put every character that arrives in order in the empty char array declared initially.

7) Use Serial.println(<updated char array>) to print the data received.

After this, do the reverse communication, i.e read the values from the arduino and then send them to the nodemcu. One thing to note here is that the nodemcu can't act as a slave in I2C communication, but still it is possible to read the values from the nodemcu and send them to the arduino. To do this, in the master code use Wire.requestFrom(8) to request data from the arduino and then follow the same code which you used for printing the data that arrives in the slave code. In the slave code, use Wire.onRequest(requestEvent) to register the request event. In the request event, use the same code you earlier used for sending the data in the master code.

e) **Expected output:**

Like earlier, the output values should be the temperature and humidity values measured by the DHT11 sensor at each instant it is sensing corresponding to the delay given. It should be displayed on the serial monitor of the slave, in this case the Arduino.

Note: Please be careful while selecting the correct boards and ports for both nodemcu and arduino while dumping the code.