**DESIGN LAB EXPERIMENT #3**

**TEMPERATURE SENSING SYSTEM USING BLUETOOTH**

# Purpose

This lab introduces the fundamental approach to utilize two different embedded system platforms to exchange data using a wireless communication protocol, based on an analog external sensor interfacing through on-board Analog-to-Digital (ADC) converter or based on a digital external sensor interfacing through digital input/output pins.

In this lab, students are required to build an ambient temperature sensing system with an external sensor interfacing with an Arduino UNO board. Students are also required to set up a central node for wireless sensor data collection with a Raspberry Pi single-board computer, which will communicate with the Arduino UNO board using Bluetooth protocol to display incoming temperature sensor data. By using Internet connection on the Raspberry Pi, either Wi-Fi or ethernet connection, received temperature sensor data are sent to a ThingSpeak private channel to store and display data through a web browser.

The purpose of this experiment is to introduce the following items:

* Arduino UNO and Raspberry Pi single-board computer
* LM35 Ambient Temperature Sensor or DHT11 Temperature and Humidity Sensor
* Linux development environment
* Interfacing Arduino UNO with Bluetooth transceiver and on-board Raspberry Pi Bluetooth transceiver
* Programming in Arduino and Python in Raspberry Pi for Bluetooth communication and HTTP protocol over the Internet
* Using open-source IoT application and API for data storing and logging

# Component Requirements

* 1 x Raspberry Pi 3 single-board computer
* 1 x Arduino UNO board and USB cable
* 1 x HC-06 Bluetooth transceiver
* 1 x LM35 Ambient Temperature Sensor or DHT11 Temperature and Humidity Sensor
* 1 x 5K or 10K Ohms Pull Up Resistor (for DHT11 Sensor)
* 1 x Breadboard

# Background

## Sensor Interfacing with Arduino

Analog sensors generate a continuous output signal in different voltages, where the output is proportional to the captured phenomena in the physical world that is quantified in different levels of voltages. In order to process this captured analog data, it is required to translate analog signal into digital signal through an Analog-to-Digital Conversion (ADC). On the Arduino UNO board, there are 6 Analog Input Pins available for users to interface with analog output devices. Built-in ADC allows analog inputs into a digital value received from any Analog Input Pins from #A0 to #A5, located as shown in Figure 1. The ADC of the Arduino UNO board has the resolution of 10-bit, which will return a digital value from 0 to 1023, and its reference voltage is 5V. In programming, user can read-in the converted analog input value by using a function called *“analogRead(pin)”*, where the return value of this function will be always between 0 to 1023.

Compared to the analog sensors, digital sensors generate continuous outputs where the phenomena in the physical world are captured in different levels of voltages and converted to digital signals. It is unnecessary to convert the output value of the external sensor on the Arduino UNO board, rather it is necessary to connect to one of the Digital I/O Pins. Similar to the Take-home Lab #1, you are required to use *“digitalRead(pin)”* function to read-in the digital input value. As shown in Figure 1, Digital I/O Pins from #2 to #13 can be used to interface with digital sensors, whereas Pin #0 and #1 should be used for serial communications if necessary with other serial device.

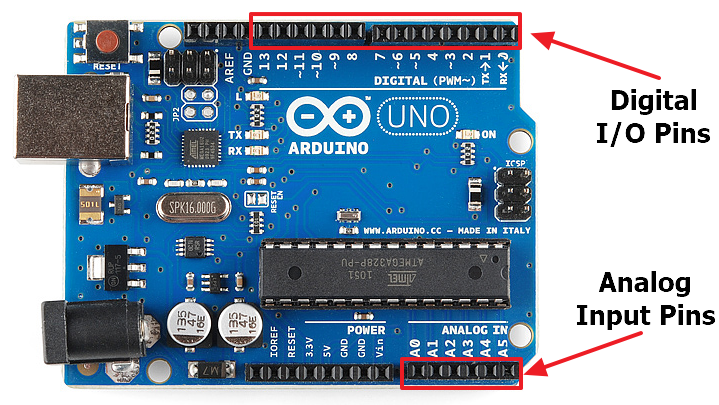


Figure 1. Arduino UNO board’s Analog and Digital I/O Pin Locations

In this lab experiment, depending on the Arduino kit that you have purchased, you will be required to connect an analog (LM35) or a digital (DHT11) temperature sensor to the Arduino UNO board where it will be transmitted to the Raspberry Pi over Bluetooth connections. LM35 Precision Centigrade Temperature Sensor is a small and precise ambient temperature sensing device with an output voltage linearly proportional to the Centigrade temperature. As per its datasheet, it features direct calibrated temperature value in Celsius, linear + 10-mV/ °C scale factor, ensures 0.5 °C accuracy and rate for full -55 °C to 150 °C range. It is suitable for remote applications, especially for IoT system development, and low-cost (less than $1.00) due to wafer-level trimming. This device operates from 4V to 30V, and draws less than 60 μA. Figure 2 (left) is the pin layout of the LM35 temperature sensor. DHT11 is a widely used digital sensor to monitor ambient temperature and humidity. As per its datasheet, this device features strictly calibrated for temperature and humidity measurement in Centigrade and in percentages (RH), respectively, ensuring ±2°C accuracy of temperature and ±5%RH accuracy of humidity. It is suitable for measurement ranges from 0-50°C in temperature and 20-90%RH in humidity, being cost-effective similar to LM35, and operates typically in 5V, and draws 0.5mA in measuring mode, 0.2mA on an average and 100μA in standby mode. Figure 2 (right) is the pin layout of the DHT11 temperature and humidity sensor.

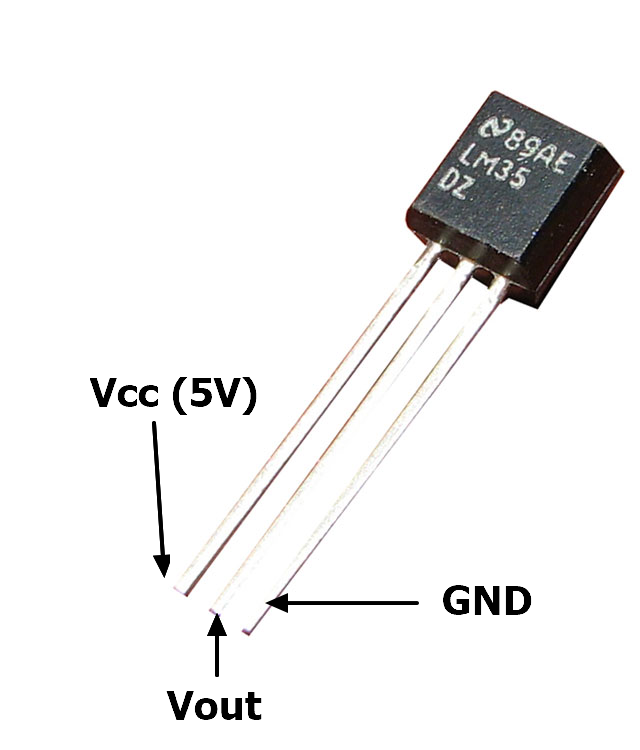
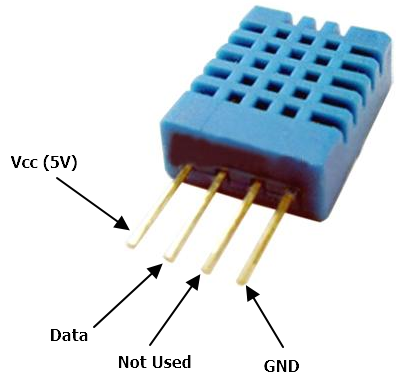
 

Figure 2. LM35 Temperature Sensor (left), DHT11 Temperature and Humidity Sensor (right)

## Bluetooth for Arduino and Raspberry Pi

Bluetooth can be one of the options for low power consuming wireless technology for sensor data collection. Its design goals are for replacing unnecessary cables, low cost and low power consumption, small sized and ability to handle ad-hoc network configuration. Bluetooth is based on the IEEE 802.15.1 standard, but it is now under Bluetooth SIG (Special Interest Group) for its protocol standardization. Bluetooth operates in the frequency band between 2.4GHz to 2.5GHz. This frequency is globally unlicensed and reserved for industrial, scientific and medical applications: it is called ISM (Industrial-Scientific-Medical) band. For example, Wi-Fi also uses this frequency for data transmission. ZigBee is another example of wireless protocol which shares this frequency, and not to mention common wireless keyboards and mouse use this frequency. Bluetooth uses a special radio technique called Frequency-Hopping Spread Spectrum (FHSS) to divide data into packets to be sent. It uses up to 79 channels(bands) for data transmission to avoid signal interference in the operating frequency. In other words, Bluetooth will automatically make 1,600hops every second throughout 79 channels, which are 1 MHz each, by spreading the traffic over the entire ISM band.

Bluetooth can be adapted to many different areas, including cargo identifiers in freight containers, hands-free headsets for voice and music, wireless keyboards and mice, game controllers and more. It is mostly known as the short-range wireless communication protocol that is adequate for establishing PAN (Personal Area Network) that is cost effective and small-sized embedded systems. Bluetooth does not have only one specification. Bluetooth performance differs on their classes and versions. The class of the Bluetooth is categorized based on their wireless performance range in different classes. Bluetooth Class 1 achieves the best range as in terms of the connection range, whereas Bluetooth Class 4 delivers the best power consumption as shown in Table 1.

Table 1. Bluetooth Classification

|  |  |  |  |
| --- | --- | --- | --- |
|  | Maximum permitted power | | Max. Range (m) |
|  | mW | dBm |
| Class 1 | 100 | 20 | 100 |
| Class 1.5 | 10 | 10 | 20 |
| Class 2 | 2.5 | 4 | 10 |
| Class 3 | 1 | 0 | 1 |
| Class 4 | 0.5 | -3 | 0.5 |

There are 5 different versions of Bluetooth available in the market and throughout the smartphone history, from Bluetooth 1.0 to Bluetooth 5.0. Most commonly used Bluetooth communication is either Classic Bluetooth connection or the Bluetooth Low Energy, also known as BLE. Classic Bluetooth connection types are mainly used for serial communication, telephony control, music control and adopted protocols, where it is capable of transmitting up to 2 Mbps. Compared to the classic Bluetooth connection, BLE, which was introduced with Bluetooth version 4, comes with a technology that consumes less power than the classic Bluetooth technology. However, the data rate is about half of the classic Bluetooth technology, which is 1 Mbps at maximum, yet many manufacturers who utilizes this technology limits the bandwidth to 20 – 100 Kbps for even more power reduction. This data rate is still adequate for most of the IoT applications. Table 2 is a brief comparison of the two Bluetooth specifications.

Table 2. Classic Bluetooth and Bluetooth Low Energy Specification Comparison

|  |  |  |
| --- | --- | --- |
|  | Classic Bluetooth | Bluetooth Low Energy |
| Data Payload Throughput | 2 Mbps | ≒ 100 Kbps |
| Robustness | Strong | Strong |
| Range | < 1000 m | < 250 m |
| Low Latency | Strong | Strong |
| Connection Set-up Speed | Weak | Strong |
| Power Consumption | Good | Very Strong |
| Cost | Good | Strong |

Arduino UNO board is not equipped with any Internet connectivity nor any low-power consuming wireless transceivers that could deliver mobility to any sensor data retrieved on the board. In order to add wireless connectivity to Arduino UNO board, it is required to connect a separate/external module to provide wireless/wired communication capability. For IoT system development, it is possible to add either Wi-Fi, a Bluetooth or a ZigBee module to the Arduino board. In this lab experiment, we will be attaching an external Bluetooth transceiver to the Arduino UNO board to communicate with the Raspberry Pi’s on-board Bluetooth transceiver. Also, this experiment will explore the usage of Classic Bluetooth technology via Bluetooth Serial Port Profile (SPP) and utilizing Radio Frequency Communication (RFCOMM) which is a simple set of transport protocols that emulates RS-232 serial ports. RFCOMM is one of the widespread supported protocol and available on almost all operating systems and Bluetooth devices. In addition, any application that uses a serial communication can be ported to the RFCOMM to transmit data over Bluetooth.

For Arduino UNO board, we will be using a Bluetooth module called HC-06. This module is similar to HC-05 module, which has the full capability as a Bluetooth transceiver, but HC-06 can only be a slave device, which requires a master Bluetooth node to communicate (Raspberry Pi Bluetooth transceiver will act as a master node). HC-06 comes with 4 I/O pins including Vcc, GND, TxD and RxD. HC-06 accepts Vcc ranges from 3.6V to 6V, and TxD/RxD voltage levels from 0 (LOW) to 3.3V (HIGH). For Arduino UNO boards, which operates in 5V, a voltage greater than 3V present at the I/O pin will report HIGH, otherwise LOW. This matches with the specification of HC-06, and is able to communicate over digital I/O pins on the Arduino UNO board. Figure 3 is the layout of the HC-06 module.

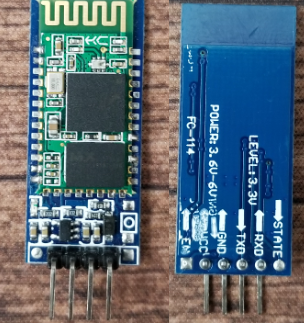


Figure 3. HC-06 Bluetooth Module

HC-06 Bluetooth module can be configured using AT commands over serial connection via Arduino UNO board. AT commands stand for Attention command, which is thoroughly used to change default settings of the Bluetooth module, such as changing the device name, changing PIN used for Bluetooth pairing and baud rate for serial communication. While HC-05 is considered as a “fully” functioning Bluetooth device, slave-limited HC-06 contains limited number of commands. Below Table 3 is the list of AT commands that can be used for configuring HC-06 Bluetooth module.

Table 3. AT Command List for HC-06

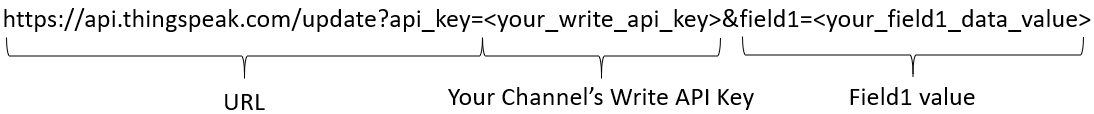
|  |  |  |
| --- | --- | --- |
| **Command** | **Reply** | **Comment** |
| AT | OK | Testing Communication Establishment |
| AT+VERSION | OKlinvorV1.8 | Firmware version |
| AT+NAMElab3BTmodule | OKsetname | Sets the name of the Bluetooth module to “lab3BTmodule” |
| AT+PIN1234 | OKsetPIN | Sets the PIN to 1234 |
| AT+BAUD1 | OK1200 | Sets the baud rate to 1200 |
| AT+BAUD2 | OK2400 | Sets the baud rate to 2400 |
| AT+BAUD3 | OK4800 | Sets the baud rate to 4800 |
| AT+BAUD4 | OK9600 | Sets the baud rate to 9600 *(default)* |
| AT+BAUD5 | OK19200 | Sets the baud rate to 19200 |
| AT+BAUD6 | OK38400 | Sets the baud rate to 38400 |
| AT+BAUD7 | OK57600 | Sets the baud rate to 57600 |
| AT+BAUD8 | OK115200 | Sets the baud rate to 115200 |
| AT+BAUD9 | OK230400 | Sets the baud rate to 230400 |
| AT+BAUDA | OK460800 | Sets the baud rate to 460800 |
| AT+BAUDB | OK921600 | Sets the baud rate to 921600 |
| AT+BAUDC | OK1382400 | Sets the baud rate to 1382400 |

As the HC-06 is a slave device, the connection must be initiated by the master device. Prior to moving forward with any data transmission using HC-06, Bluetooth pairing must be taken place. Pairing HC-06 Bluetooth module with the Raspberry Pi’s on-board Bluetooth transceiver will be described in the later section. Raspberry Pi 3B and later versions are all equipped with integrated Bluetooth 4 compliant module. This Bluetooth module is backward compatible, and it can communicate with HC-06 Bluetooth module using RFCOMM. For our lab experiment, it is recommended to utilize Raspberry Pi 3B+ over Raspberry Pi 3B due to a known issue where Raspberry Pi 3B may not allow using both Bluetooth and Wi-Fi at the same time.

## ThingSpeak IoT Analytics Platform

As per ThingSpeak’s developer’s definition, “ThingSpeak is an open source “Internet of Things” application and API to store and retrieve data from things using HTTP over the Internet or via a Local Area Network. With ThingSpeak, you can create sensor logging applications, location tracking applications, and a social network of things with status updates. In addition to storing and retrieving numeric and alphanumeric data, the ThingSpeak API allows for numeric data processing such as time scaling, averaging, median, summing, and rounding. Each ThingSpeak Channel supports data entries of up to 8 data fields, latitude, longitude, elevation, and status. The channel feeds support JSON, XML, and CSV formats for integration into applications. The ThingSpeak application also features time zone management, read/write API key management and JavaScript-based charts from Highslide Software/Torstein Hønsi.” ThingSpeak serves to provide users to aggregate, visualize and analyze real-time data streams collected in the cloud. It offers sending alerts using web services (e.g., Twitter), ability to write and execute MATLAB code, and enables engineers to prototype and build IoT systems without setting up servers and web applications from scratch. *(See References 8 and 9)*

Uploading to the ThingSpeak is achieved by using either HTTP GET or POST method via a specific URL with a predefined private/public channel ID, unique key to access the channel, and corresponding fields for incoming data. An example of using HTTP GET method to the RESTful API for ThingSpeak is as follows:



where URL is the base address for any case when updating/uploading data to ThingSpeak, write API key is randomly generated from your ThingSpeak channel website, and field1 is where you’ll send your data to. Field can be generated up to 8 in ThingSpeak. Note that, under free license version of ThingSpeak, it is only allowed to update and store new data in every 15 seconds, and is limited to 4 channels.

# Statement of Problem

In this lab experiment, you will implement temperature sensing system using LM35 temperature sensor wired to the Arduino UNO board, transmitting temperature sensor data to the Raspberry Pi via Bluetooth, concurrently uploading received data to your ThingSpeak’s channel, and visualize sensor data online. You are required to perform hardware implementation of LM35 temperature sensor and Bluetooth HC-06 module to the Arduino UNO board, software implementation of programming Arduino UNO board, Raspberry Pi using Python, and configuring your ThingSpeak channel.

# Procedure

## A. Hardware Set Up on Arduino UNO board

In this part of the lab experiment, you are required to connect one LM35 or DHT11 temperature sensor and one Bluetooth HC-06 module to the Arduino UNO board using a breadboard. Check datasheets of both devices for correct I/O pin maps.

1. If using LM35 temperature sensor, connect the output to one of the Arduino UNO board’s analog input pins. Power on LM35 temperature sensor via 5V output pin from the Arduino UNO board by connecting Vcc to 5V pin and GND to GND pin on the Arduino UNO board. If you are using DHT11 temperature sensor, connect the output to one of the Arduino UNO board’s digital I/O pins. Power on DHT11 sensor via 5V output in from the Arduino NO board by connecting Vcc to 5V pin and GND to GND pin on the Arduino UNO board. Identify which analog input pin you have chosen and explain in your report.
2. Connect Bluetooth HC-06 module’s Vcc and GND to 5V and GND of the Arduino UNO board, respectively. Connect Bluetooth HC-06 module’s RxD and TxD to your choice of digital I/O pins of the Arduino UNO board. **Do not connect them to Pin #0 and #1**. Identify which digital I/O pins you have chosen and explain in your report. Two digital I/O pins will be RxD and TxD from Arduino UNO board’s perspective, and they need to be cross-connected, meaning HC-06 module’s RxD connected to Arduino UNO board’s TxD and HC-06 module’s TxD connected to Arduino UNO board’s RxD. Define which pins correspond to RxD and TxD.
3. Take pictures of your hardware wire connections from LM35 or DHT11 temperature sensor and Bluetooth HC-06 to the Arduino UNO board, and include them in your report.

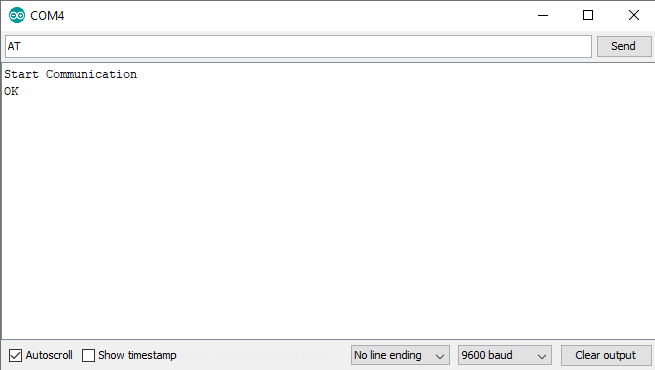
## B. Bluetooth Set Up using Arduino UNO board

In this part of the lab experiment, you are required to assure all Bluetooth settings are correctly set up before moving to the next step. By using AT commands and example code given from Arduino IDE, you are required to modify your Bluetooth HC-06 settings including its name, baud rate and PIN number.

1. Launch Arduino IDE and open an example code called “SoftwareSerialExample”. You can find this under *File >> Examples >> SoftwareSerial >> SoftwareSerialExample*.
2. Modify the example code like the following. Enter the appropriate values for *TxD of BT module* and *RxD of BT module* in the 2nd line of code. These values should match with the defined pins from Procedure A.2.



1. Connect Arduino UNO board to the USB cable and power on. If your Bluetooth HC-06 is properly connected, red LED should blink.
2. Load the modified program to the Arduino UNO board, and launch Serial Monitor within Arduino IDE. Make sure the baud rate of the Serial Monitor matches with the setting in your code. Serial Monitor can be found under *Tools >> Serial Monitor,* or shortcut key *ctrl+shift+m,* or by clickingicon on the top right corner of Arduino IDE.
3. Type in “AT” in the field next to “Send” button in Serial Monitor. If your connection is set up correctly, your output should be the same as below:



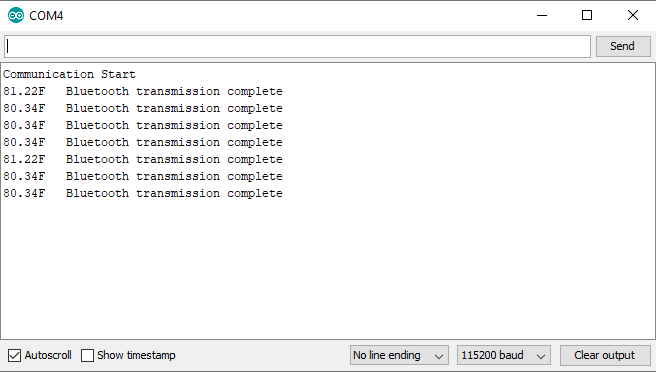
1. If your output does not give you “OK” on the Serial Monitor, it means your Bluetooth HC-06 module’s RxD and TxD are not connected correctly. It is suggested to try switching the two and repeat above Step 5.
2. By using AT commands listed in Table 3, change your Bluetooth HC-06 to your choice of name. (e.g., 442lab3BT) If you are in a lab environment working with different groups, make sure your Bluetooth HC-06 name is unique so that you don’t confuse yourself to connect to others’ device.
3. By using AT commands listed in Table 3, change the Bluetooth pairing PIN number to your choice of 4-digit number (e.g., 9876)
4. Finally, by using AT commands listed in Table, change the baud rate of your Bluetooth HC-06 device to 115200. Take a screenshot of your Serial Monitor and include in your report.
5. You may notice that any AT command that you type to the Serial Monitor does not work after changing the baud rate of your Bluetooth HC-06 device. This is because your Bluetooth module is now communicating with Arduino UNO board at the baud rate of 115200. To verify if your setting is applied to the Bluetooth device, modify your Arduino code from Step 2. You should be able to see the same output as the figure in Step 5, except the baud rate now should be 115200.

## C. Temperature Sensing using Arduino UNO board

In this part of the lab experiment, you are required to retrieve and display current ambient temperature data on the Serial Monitor. You should design your Arduino program should fetch the temperature data every 1 second. Your program is based on the code from Procedure B. Follow Part A if you are using LM35 analog temperature sensor. Follow Part B if you are using DHT11 digital temperature sensor.

*Part A. LM35 Analog Temperature Sensor Data Retrieval*

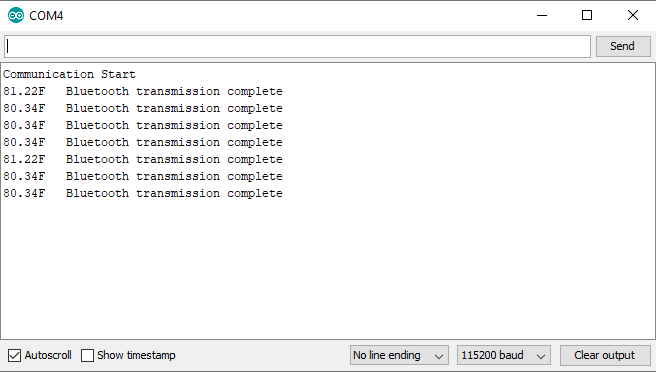
1. Remove all codes under *void loop()*.
2. Write a code under *void loop()* that would read in data from your choice of analog input pin from the Arduino UNO board. Remember that reading in analog input pin value is always between 0 and 1023 (in integer). Save this integer value to an integer variable, name it *raw\_sensor\_value*.
3. Use the following conversion equation to convert retrieved integer value and save to a float variable in Centigrade:
4. Do another temperature conversion from Centigrade to Fahrenheit by using the following equation, and save also to a float variable:
5. Output converted temperature value to the Serial Monitor, and to the Bluetooth HC-06 module by using *println(val)* function for Serial Monitor and *write(val)* function for *SoftwareSerial mySerial*, respectively. Lastly, add a 1 second delay at the end of the *void loop()* function (Use *delay(millisecond)* function). Your output should be similar to the following screenshot:



1. Take the screenshot of your Serial Monitor Output. Your Bluetooth HC-06 module should be still blinking at this stage, because we have not set up receiving part of the device, Raspberry Pi. You may leave your Arduino UNO board running while you perform the rest of the lab procedures.

*Part B. DHT11 Digital Temperature Sensor Data Retrieval (Reference 11 for library details)*

1. Place *“dht.h”* and *“dht.cpp”* files in the same folder as your code. Include the header file *“dht.h”* in your code. This library contains a DHT object called *“dht”*. Create a DHT object after including the header file (e.g., *“dht DHT;”*)
2. Remove all codes under *void loop()* and *void setup()*.
3. By utilizing the created DHT object, you can read the data from the sensor by using *read11(pin)* function. After reading is complete, since DHT11 contains temperature and humidity sensors, both readings will be stored at object variable *temperature* and *humidity*. Write a code under *void loop()* that would read in the data and store the object variable *temperature* value to a float local variable.
4. Stored temperature value will be in Centigrade. Use the following conversion equation to convert the temperature value from Centigrade to Fahrenheit.
5. Output converted temperature value to the Serial Monitor, and to the Bluetooth HC-06 module by using *println(val)* function for Serial Monitor and *write(val)* function for *SoftwareSerial mySerial*, respectively. Lastly, add a 1 second delay at the end of the *void loop()* function (Use *delay(millisecond)* function). Your output should be similar to the following screenshot:

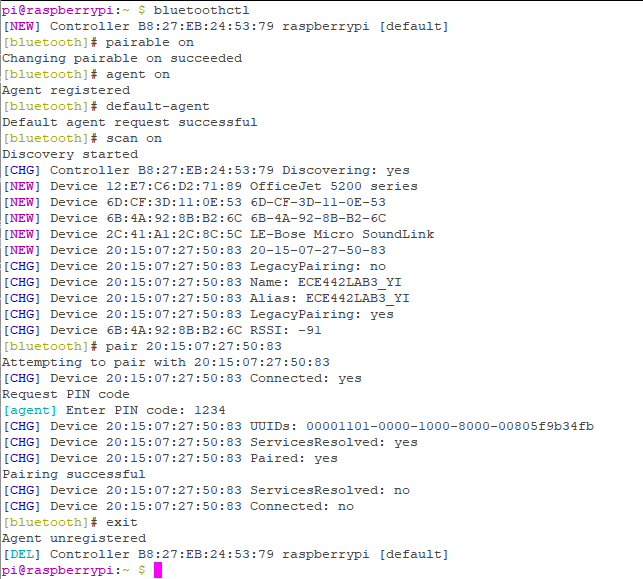


1. Take the screenshot of your Serial Monitor Output. Your Bluetooth HC-06 module should be still blinking at this stage, because we have not set up receiving part of the device, Raspberry Pi. You may leave your Arduino UNO board running while you perform the rest of the lab procedures.

## D. Bluetooth and Python Library Setting on Raspberry Pi

In this part of the lab experiment, you are required to initialize the Bluetooth setup between the Raspberry Pi and the Bluetooth HC-06 on the Arduino UNO board. You are required to, first, identify the MAC address of the Bluetooth HC-06 module that you are using, second, pairing the Bluetooth HC-06 module with the Raspberry Pi’s on-board Bluetooth module prior to any type of communication, lastly, installation of any required libraries for Python programming.

1. Make sure your Raspberry Pi is connected to the Internet. You may use either on-board Wi-Fi or Ethernet connection.
2. Open up a terminal by pressing shortcut key *ctrl+alt+t* on your Raspberry Pi.
3. Type *bluetoothctl* on the terminal. By default, NOOBS will contain a program called **bluetoothctl**. This program allows you to configure Bluetooth connections on your Raspberry Pi. Now type the following commands one by one:
   1. pairable on
   2. agent on
   3. default-agent
   4. scan on
4. Once you have typed *“scan on”*, Raspberry Pi’s Bluetooth device will scan for available Bluetooth devices around. Wait until you see your Bluetooth HC-06 module with your choice of name appear on the terminal.
5. If you have identified your HC-06 module, copy the MAC address of the HC-06 module, and type *“pair MAC ADDRESS”* and enter PIN number of your choice from Procedure B. If successful, exit **bluetoothctl** by typing *“exit”*. Include your own terminal output in your report. Your terminal output should be similar to the following screenshot:



1. For your Python program to interface with the Bluetooth and ThingSpeak, there are few libraries as a prerequisite. Type following commands to install both libraries to your Raspberry Pi:
   1. Bluetooth for Python

*sudo apt update*

*sudo apt install bluetooth libbluetooth-dev*

*sudo python3 -m pip install pybluez*

* 1. ThingSpeak for Python

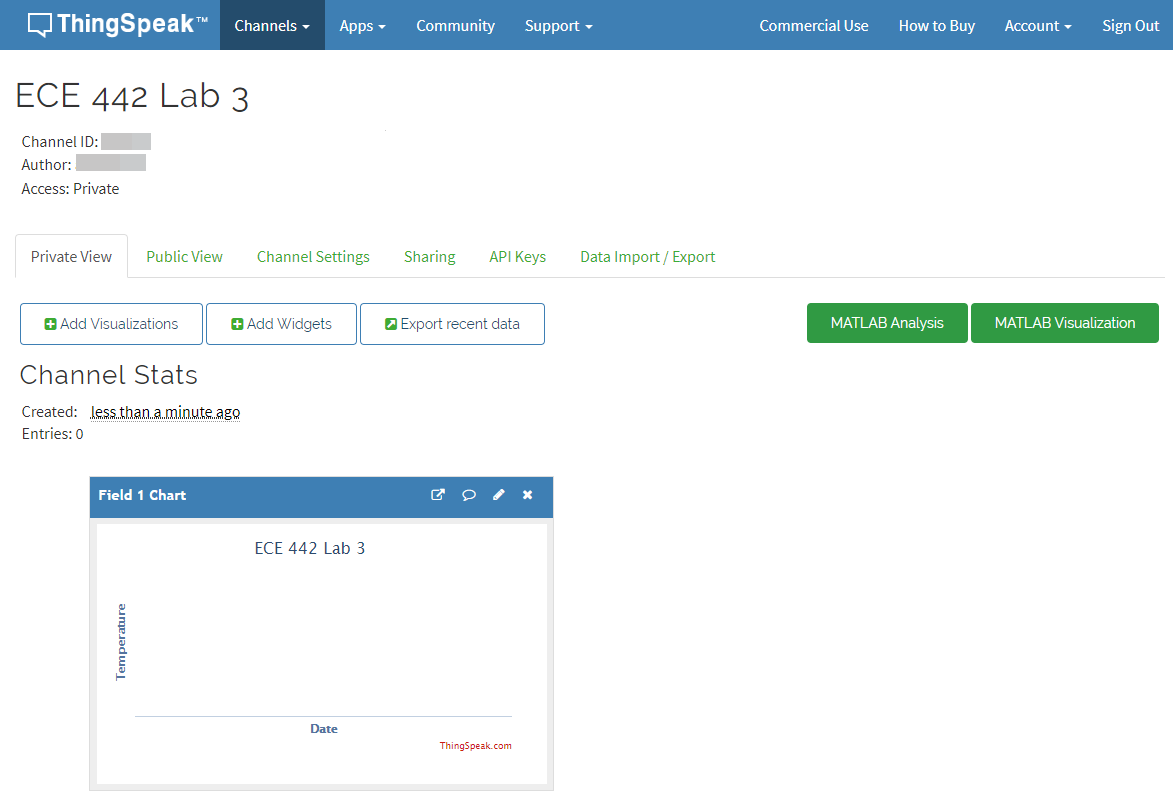
*sudo python3 -m pip install thingspeak*

1. Take screenshots of the Raspberry Pi output from Step 4 and Step 5. Include these in your report.

## E. Connecting Arduino UNO board, Raspberry Pi and ThingSpeak

In this part of the lab experiment, you are required to perform a complete IoT Temperature Sensing System where the temperature data is collected on the Arduino UNO board, transmitted to the Raspberry Pi simultaneously over the Bluetooth, and concurrently synchronizing with your private channel of ThingSpeak to display and store the sensor data. On the Raspberry Pi, you are required to implement a Python program that will consist of reading incoming data over the Bluetooth, and sending updates to your ThingSpeak channel.

1. First, you’ll need to set up your own ThingSpeak account and channel. Go to <http://thingspeak.com>
2. Once login to ThingSpeak, create a new channel by clicking on “New Channel”
3. Name your channel to *“ECE 442 Lab 3”*. For Field 1, make sure you click the checkbox and name it *“Temperature”*. You may leave other fields empty. Click on *“Save Channel”* to save your setting.
4. Your screen now should look like below screenshot. You can identify your channel’s ID under *“ECE 442 Lab 3”*.



1. Obtain your channel’s Write API Key under *“API Keys”* tab.
2. Type in the given Python code from Appendix A on your Raspberry Pi, and save this file as *“lab3.py”*. You need to modify the code corresponding to your system, including the Bluetooth HC-06 MAC address, your ThingSpeak Channel ID and Write API Key. Make sure that when your indentation is consistent throughout your Python code.
3. While your Arduino UNO board is running your code from Procedure C, execute your lab3.py code by executing below command:

*sudo python3 lab3.py*

If all of your settings were correctly setup, red LED on your Bluetooth HC-06 should now be always on. If it blinks, it is a sign that your Raspberry Pi’s Bluetooth and your HC-06 are not connecting each other.

1. Take screenshots of your Raspberry Pi output and your ThingSpeak’s Private View. Your ThingSpeak screenshot should be similar to the figure shown in Step 4 but with your temperature sensor data plotted under *‘Field 1 Chart’*. You must show that your Raspberry Pi is receiving the same data from the Arduino UNO board, and show corresponding data have been updated to the ThingSpeak. Include the screenshots in your report. Note that free version of ThingSpeak will only accept new data every 15 seconds. Thus, not all sensor data will be transferred to ThingSpeak.
2. You may stop your program by pressing *ctrl+c* to break the loop.
3. Provide a short demonstration video of your entire system with explanation, and upload to the Blackboard along with your report.

# Discussion (Include your answers in your report)

1. Fully commented codes that you implemented for Arduino UNO board and Raspberry Pi
2. Draw a flow chart and explain the overall system architecture. Elaborate on how the sensor data transmitted throughout the lab procedure.
3. Explain the conversion equation shown in Procedure C.Part A.3. You may need to refer to the datasheet of LM35 temperature sensor and ADC conversion.
4. Explain in detail how Raspberry Pi uploads/updates to ThingSpeak server.
5. In this lab experiment, you implemented a classic Bluetooth connection in Python on your Raspberry Pi. Research and explain how it would be different if you were asked to interface with a Bluetooth Low Energy (BLE) device using Python. Describe your answer in terms of Python libraries, specific functions, flow of your Python code.
6. Instead of using Bluetooth for communication, you can also use other types of low power consuming wireless devices such as ZigBee protocol. For embedded systems, you can utilize XBee modules for ZigBee communication (*See Reference 10*). If you were asked to replace Bluetooth modules with XBee modules, how would you layout your hardware configuration on your Arduino and your Raspberry Pi? Provide a schematic with specific pin connection information for both Arduino UNO board and Raspberry Pi. Provide detailed explanation of your connections for both boards.

# References

Some references you may find useful:

[1] Bluetooth Socket Programming

<https://people.csail.mit.edu/albert/bluez-intro/c212.html>

[2] Raspberry Pi hardware documents

<https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>

[3] Arduino UNO hardware information

<https://store.arduino.cc/usa/arduino-uno-rev3>

[4] HC-06 Bluetooth to Serial module datasheet

<https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf>

[5] Common Linux commands

<https://www.raspberrypi.org/documentation/linux/usage/commands.md>

[6] Python 3 tutorial

<https://docs.python.org/3/tutorial/>

[7] LM35 Precision Centigrade Temperature Sensor datasheet

<http://www.ti.com/lit/ds/symlink/lm35.pdf>

[8] ThingSpeak

<https://www.mathworks.com/products/thingspeak.html>

[9] ThingSpeak Github README

<https://github.com/iobridge/thingspeak/blob/master/README.textile>

[10] XBee/XBee-PRO S2C Zigbee

<https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf>

[11] DHT11 Library

<https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib>

[12] DHT11 Datasheet

<https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf>

**Appendix A**

- Raspberry Pi Python Program (lab3.py) –

import bluetooth # needed for bluetooth communication

import thingspeak # needed for thingspeak

bluetooth\_addr = *"YOUR HC-06 MAC ADDRESS GOES HERE"* # The address from the HC-06 sensor

bluetooth\_port = 1 # Channel 1 for RFCOMM

bluetoothSocket = bluetooth.BluetoothSocket (bluetooth.RFCOMM)

bluetoothSocket.connect((bluetooth\_addr,bluetooth\_port))

#thingspeak information

channel\_id = *YOUR\_CHANNEL\_ID* # channel ID from your Thingspeak channel

key = *YOUR\_CHANNEL'S\_WRITE\_API\_KEY* # obtain from Thingspeak

url = 'https://api.thinkspeak.com/update' # default URL to update Thingspeak

ts = thingspeak.Channel(channel\_id, key, url)

while 1:

try:

received\_data = bluetoothSocket.recv(1024)

temperature = int.from\_bytes(received\_data,byteorder='big')

print("Current Temperature: %d" % temperature)

thingspeak\_field1 = {"field1": temperature}

ts.update(thingspeak\_field1) # update thingspeak

except KeyboardInterrupt:

print("keyboard interrupt detected")

break

bluetoothSocket.close()