THE DARIU FOUNDATION

Build Your Own IoT Gateway with Python

Dr. Le Trong Nhan

Contents

CHAPTER 1

Create an Adafruit IO Account

1 Introduction

The new generation of the internet network, which is well-known as the Internet of Things (IoTs), is a revolution in connecting wireless devices. At first, Internet, an achievement in Digital Revolution, authorizes computers to connect and exchange information worldwide. Moreover, not only computers but also many devices today can access the Internet with the development of the Micro Electronic Mechanical System. The most common devices in our daily life that can connect to the network are smartphones, tablets, smart cards, or nodes in Wireless Sensor Networks. According to the science community's estimation, by the year 2025, there will be 75 billion devices that can connect to the Internet with each other. With all of those characteristics, a new generation of the network has been established and became a specialty for the fourth industrial revolution, the Internet all thing network, also called IoT - Internet of Things.

Basing on IoTs, applications are no longer staying at the smart concept but take one step further, which is called **autonomous**. For instance, some supervising and self-adapting projects about indoor services, parking lots, or monitor systems in agriculture and aquaculture. According to Timothy Chou - a famous speaker, his architecture about smart app base on internet of things was divided into a 5-layer model. The figure below will describe this model.

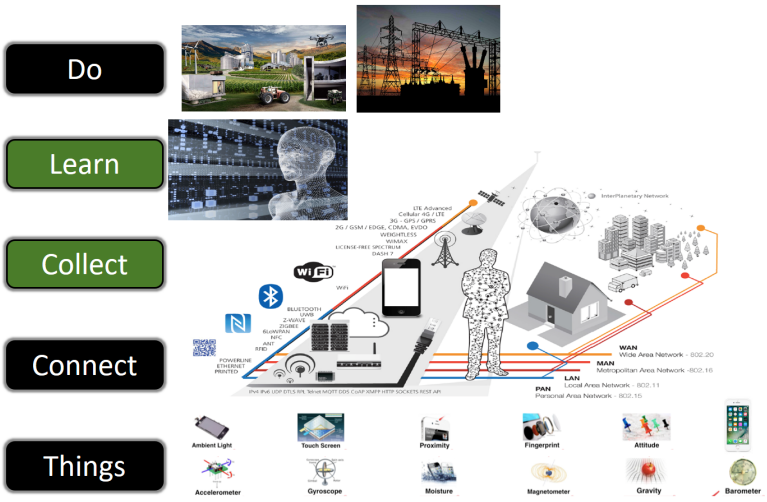

Figure 1.1: 5 layers structure of an Internet of Things application

Main function of each layer in this structure can be generalized as follow:

- **Things:** Devices in supervising app. As we can see, this is a very diversified layer in terms of number and function. A lot of sensors will be used, depending on supervising app. Besides, sensor nodes primarily rely on wireless communication.
- **Connect:** Data collection from sensor nodes. Because existing connection standards are based on utilization, so this layer must support many connection types. Those types include Zigbee and Wifi in smart homes with short-range communication or long-range communication as LoRa or 3G/4G.
- **Collect:** After the data is collected, a central server will receive and store data. Then, a large amount of data will be pushed back to smaller servers, creating a big challenge for them which processing by Big Data technology.

- **Learn:** The role of this layer is to filter out specific info and explicit semantic for different applications. Technologies of Machine Learning and currently Deep Learning are applied here.

- **Do:** The system will create some adaptive rules to fit the environment and suggest decisions based on specific info. With each decision, the performance is evaluated automatically by the system. Then, The system carries out deviation of action from the optimal goal for the next time. In this way, the system automatically accumulates "experience" over a long period, to become more and more intelligent and perfect.

This tutorial will focus on layer **Connect**, with the function namely **Gateway IoTs**. A device will play a role of Gateway. It collects info and send them to the server as well as receive control signal from server at **Collect** layer. We will create a Gateway in Python programming language. The main content of this tutorial include:

- 4-layer structure application base on IoT
- Create an Adafruit IO Server's account
- Create a channel to store data on Adafruit IO

2 The 4-component architecture of the application

Based on the 5-layer model of IoT architecture, we will divide the application into 4 basic components, including sensor node, central Gateway, server and devices for data monitoring and remote control, as shown in the figure below:

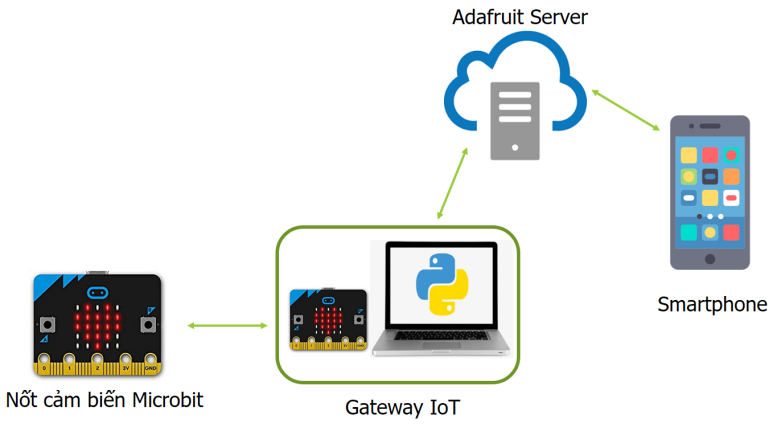

Figure 1.2: The 4-component architecture of IoT application

The key factor in this architecture is the Gateway IoT, which will be built by combining a computer and a Microbit. The reason for this combination is because the function of Gateway will be implemented in Python programming language. Our computer is a powerful and handy device for this requirement. In addition, when implementing the system, the Python program can be easily used on embedded computers, such as Raspberry PI for example. The Microbit circuit was added because of hte need to expand the connection

with many other sensor circuits using the wireless communication technology of the Microbit circuit.

Many Microbit can act as sensor node and send data back to the central Microbit, where it sent data to computer. In the computer, the program written in Python will send this data to the Adafruit IO server. From there, remote monitoring devices such as mobile phones or even another computer can monitor the system's data. In case you want to control a device, the data flow will go backward from the terminal to the sensor node, to execute the control command.

In this first tutorial, we will start creating an account on the Adafruit IO server, before we can use it to store sensor data and circulate control signals.

3 Create an account on Adafruit IO

Step 1: Access the main website at https://io.adafruit.com/. This interface will appear.

Figure 1.3: Main website of Adafruit IO

Choose **Sign In** to login if you already have an account. However, this interactive button will also lead us to creating new account page in the next step.

Step 2: Because we don't have an account yet, so at this step, we choose **Sign Up** to register an account.

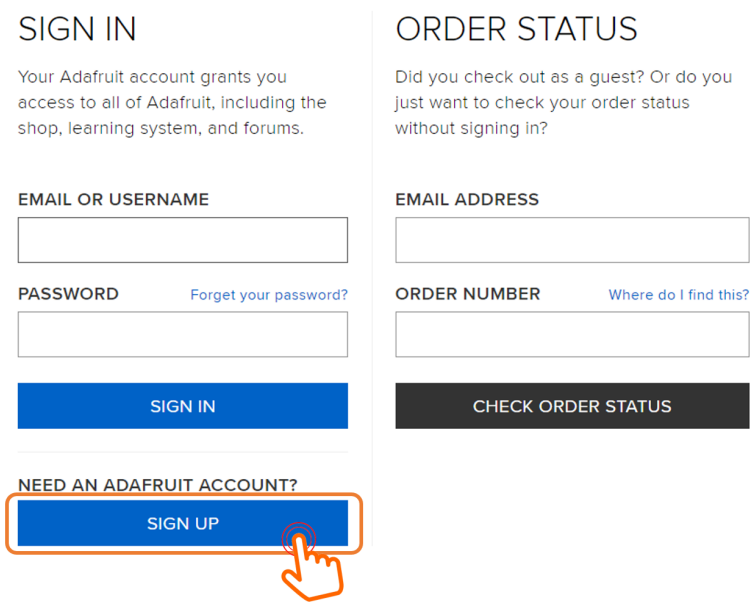

Figure 1.4: Creating an account on Adafruit IO

Step 3: Provide personal information (FIRST NAME and LAST NAME), Email, username and password. Except for **USERNAME, you can not use special characters**. Finally, we choose **CREATE ACCOUNT** to create an account, as the instruction below.

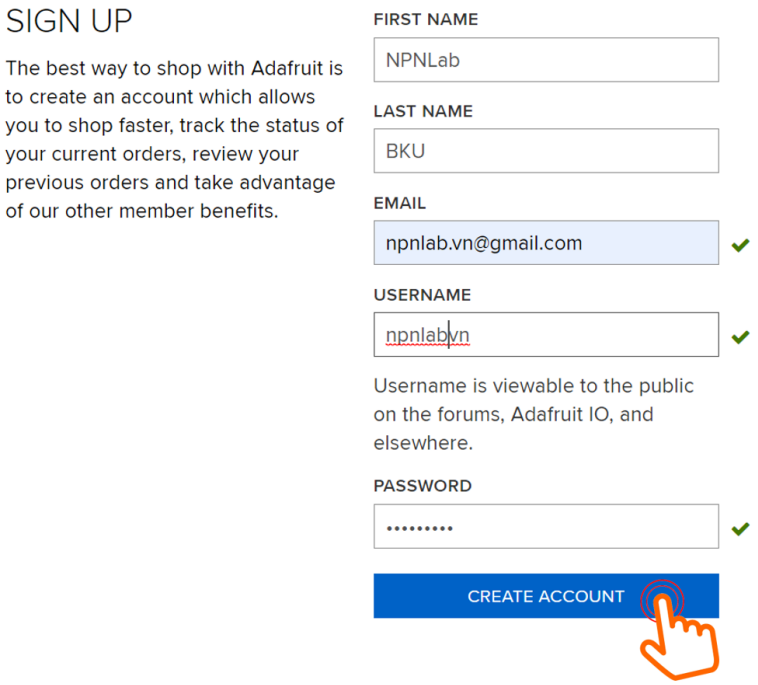

Figure 1.5: Sign up an account interface

Although, after successfully created an account, the system will login automatically, in the next working session, user can login with **Sign In** function.

4 Create a Feed

To be able to store data on server, we must classify it. Normally, We call it a **feed**. Usually, each of the object in the system will have its own data channel. For example, too record the state of a LED, we need a channel which name is **BBC_LED**. After successfully logging in to the system, we begin to create the first data channel, with the instructions below.

Step 1: Open the list of data channels by clicking on Feeds, as showed in the picture below:

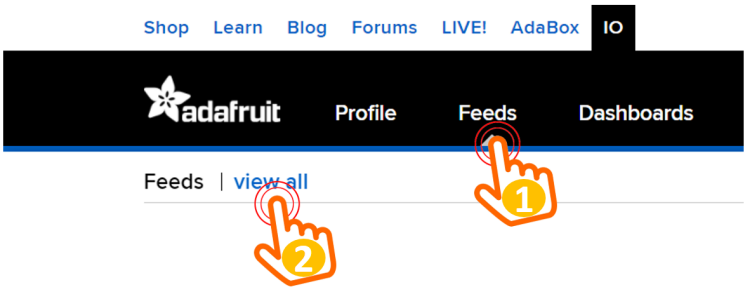

Figure 1.6: Access to data channel

Next, we chose **View all** to list all available data channels in your account, as shown in the following figure:

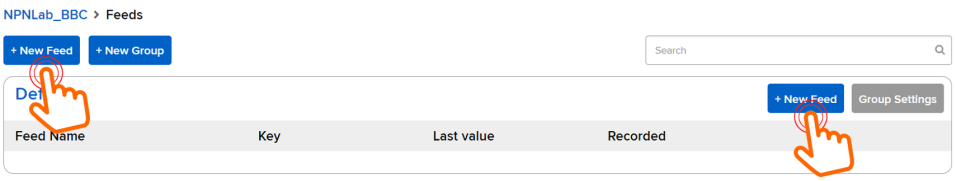

Figure 1.7: List of existing channels

In image above, our account does not have a data channel yet, because it has just been registered for the first time. A data channel will be created by clicking the **New Feed** button.

Step 2: Fill in the information for the data channel, as shown in the figure below.

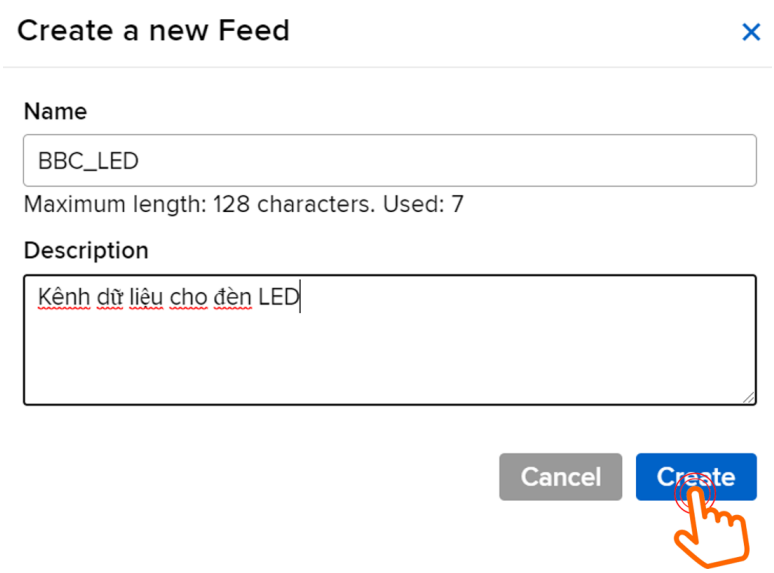

Figure 1.8: Fill in the required information for data channel

The most important is the **Name** of the data channel. Readers should name it with a prefix, to easily distinguish which data channel is for which project. In this tutorial, we illustrate data from a led display on Microbit, and name the channel **BBC_LED**. The **Description** is optional, readers can add additional comment information. Finally click the **Create** button. A new channel will be created and our interface will now look like this:

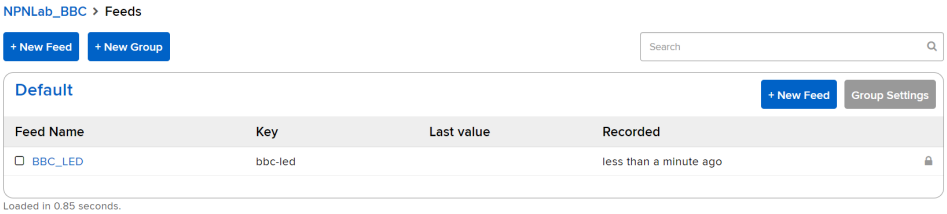

Figure 1.9: Create a new data channel successfully

In case you want to delete an existing data channel, you can chose it and click **Delete Feed**, as the picture below:

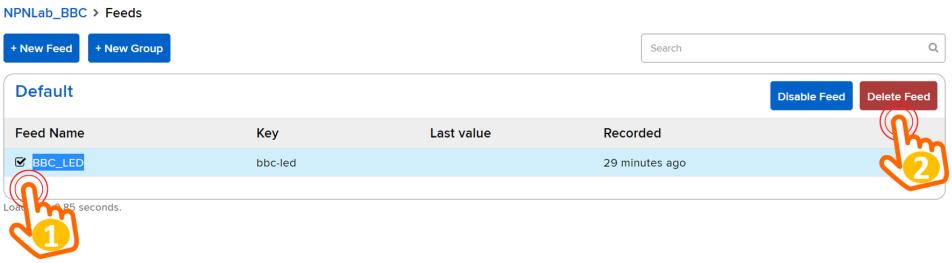

Figure 1.10: Delete an existing data channel

5 Share Feed in public

When a feed is created, it is set to **private** by default, only the logged in account can access and send data to it. To simplify programming from IoT devices to feed, we will configure

it to be **Public**.

To do this, we need to access the feed directly, by clicking directly on the feed name, here **BBC_LED**. There are many ways for readers to find your own channel. According to the current process, readers can directly tap **BBC_LED** at Figure 1.10. In case you just login to your account, just select Feeds, this data channel will appear, as shown in the image below:

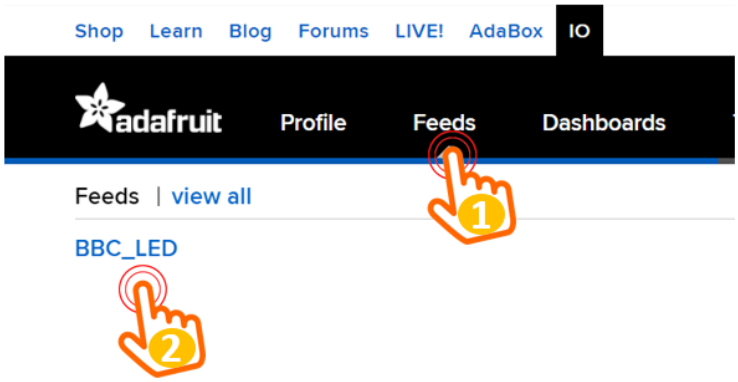

Figure 1.11: Another way to access Feed

The detail infor of Feed will be showed as the picture below::

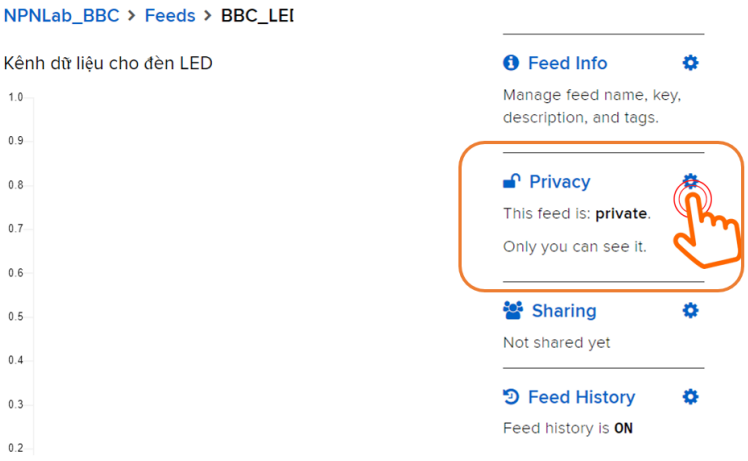

Figure 1.12: Info of a feed

Let's notice the **Privacy** section in the right pane, it's currently in **private** mode. We click on the settings icon, to go to the following interface:

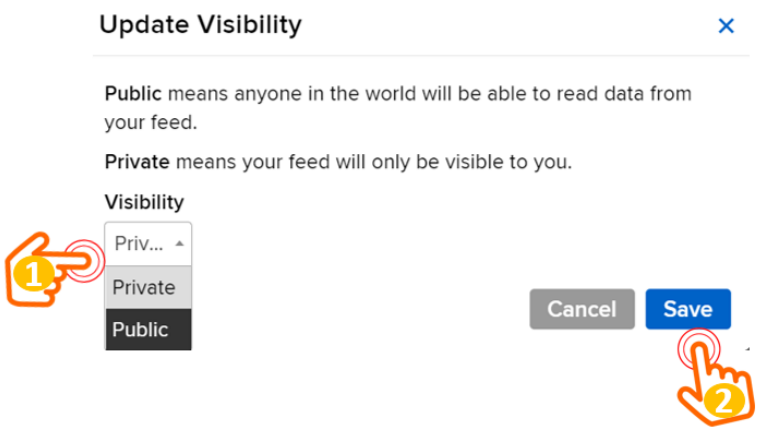

Figure 1.13: Setting share mode for Feed

In the above interface, we select **Public** in the **Visibility** section and finally, click the **Save** button to complete the channel adjustment in shared mode.

Now, the information in the **Privacy** section has changed, with the additional instructions **Anyone can see it at this link**. You can share your data channel with others by sending this link. However, this step is only necessary in checking whether the data channel can communicate instantaneously. Also, it's not really a useful feature in the applications we're going to implement. Usually, you will have the need to mask your channel to protect your system's data. Our purpose when setting the channel to Public is just to simplify future programming.

6 Review questions

1. What server was introduce in the tutorial?
   A. ThingSpeak
   B. Google
   C. Amazon
   D. Adafruit IO

2. What is the name of a channel for storing data on server?
   A. Client
   B. Server
   C. Feed
   D. Channel

3. Timothy Chou proposes an architecture about applications of the internet of things with how many layers?
   A. 2
   B. 3
   C. 4
   D. 5

4. Devices like sensors, pumps, power circuits belong to which layers in the IoT model?
   A. Things
   B. Connect
   C. Collect
   D. Learn

5. Which layer below does Gateway IoT have its place?
   A. Things
   B. Connect
   C. Collect
   D. Learn

6. To simplified future programming, what actions are necessary?
   A. create a data channel with a reminding name
   B. adding prefix for channel's name
   C. Set the channel to Public
   D. All of the above

7. To access the details of a feed, what are the operations required?
   A. Select Feeds, select data channel (feed name)
   B. Select Feeds, select view all, select data channel (feed name)
   C. Both of the above will work
   D. All of the above

Solution

1. D 2. C 3. D 4. A 5. C 6. D 7. D

CHAPTER 2

Designing Dashboard on Adafruit IO

1 Introduction

Dashboard can be understood as an collective table which show all indispensable information of a system. On dashboard, Information can be collected and classified to synthesize in order to display as a graph which contains historical data, present information as well as statistic of minimum, maximum or average values. Beside, Dashboard can be used as a friendly control board, so, users can interact and operate the system remotely. Depending on specific characteristics and background of application, the requirement of dashboard can be different. In many companies, Dashboard show a comprehensive view on capacity of each components, trends, activities, KPIs (Key Performance Indicator - performance evaluation index).

With different uses, the Feed data channel which introduced in previous lesson, often used by the manager in order to check raw data of the system. Meanwhile, Dashboard is a eye-catching and user-friendly interface. These two objects are usually called Back End for Feed and Front End for Dashboard. Obviously, these two objects will have closed association: If data is sent to Feed, Dashboard interface will be updated correlatively and vice versa.

In this lesson, readers will be guided to create a simple Dashboard on Adafruit IO. This Dashboard is used to connect to Feed that we have created in the previous lesson and have a button which users can turn on/off the LED on Microbit. Dashboard Interface looks like this:

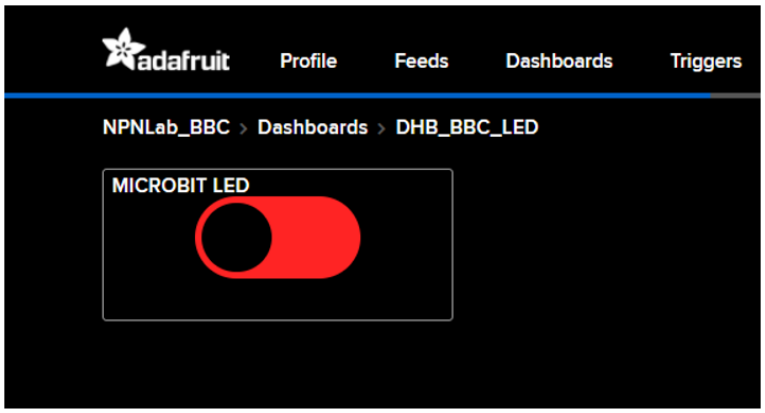

Figure 2.1: *LED control Dashboard interface*

The main purposes of this chapter are as follows:

- Create Dashboard with a button
- Link Dashboard and Feed
- Check interaction between Dashboard and Feed

2 Create new Dashboard

In this section, this book will help you to make a simple interface, with a button on dashboard, used to turn om/off a device, such as LED blink on Microbit. The interface and

operation to create a new Dashboard is similar to creating a new Feed in the previous chapter, showed step by step as below.

Step 1: After signing in Adafruit IO account, you choose **Dashboard**, and click **View all**, as instruction below:

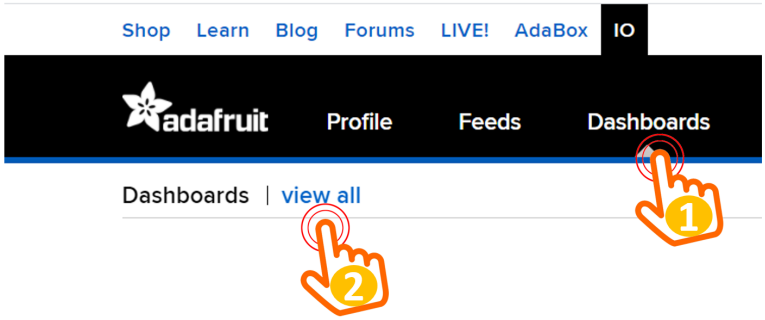

Figure 2.2: Access to account Dashboard

Step 2: Press **+New Dashboard** to create a new project on Adafruit, as shown in the following figure:

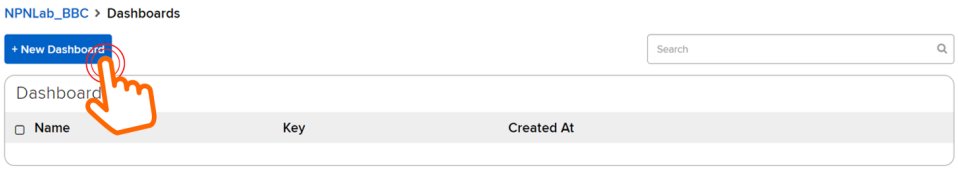

Figure 2.3: Make a new Dashboard

After that, one window will appear to fill in the name and description for Dashboard. Attention! **Name** is required information, meanwhile, Description is optional. When naming the Dashboard, you should also add prefixes for future management, as shown in the image below:

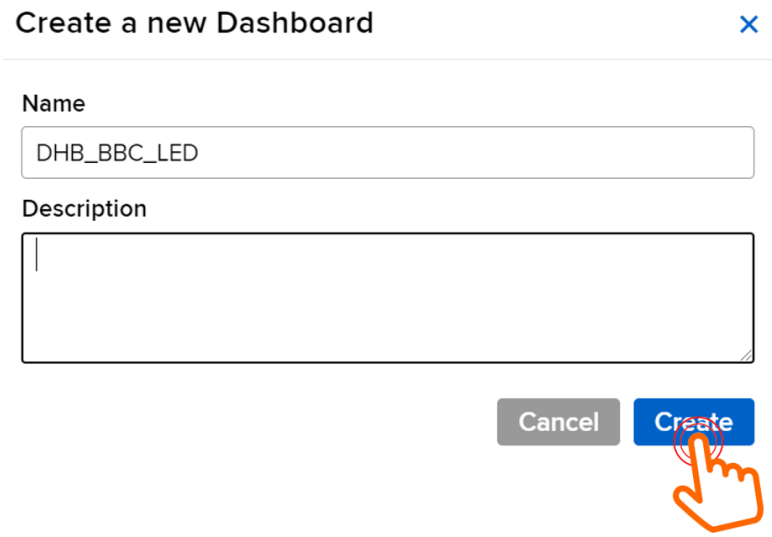

Figure 2.4: Dashboard information completion

Finally, press **Create** button, a new Dashobard will be created in your Adafruit IO homepage, as shown in the following image:

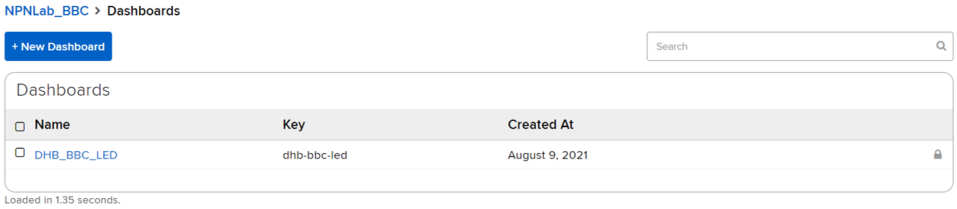

Figure 2.5: A new Dashboard has been created

In case you wan to delete the Dashboard, you need to choose it in the above list, and press **Delete Dashboard** button, as shown in the following image:

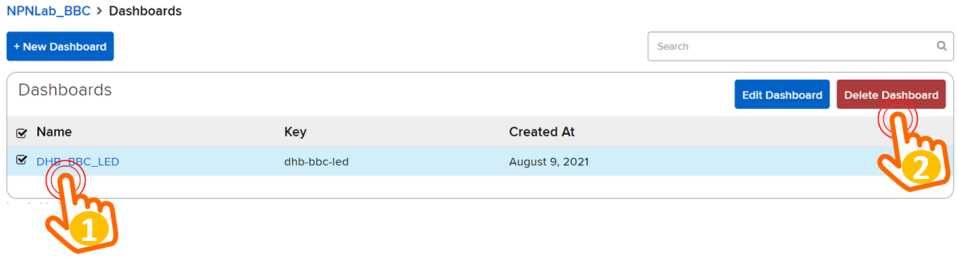

Figure 2.6: Delete a Dashboard from the list

After successfully creating a Dashboard in previous steps, you can access into it(click directly Dashboard button you want to choose), an interface will be initialized and appear like this:

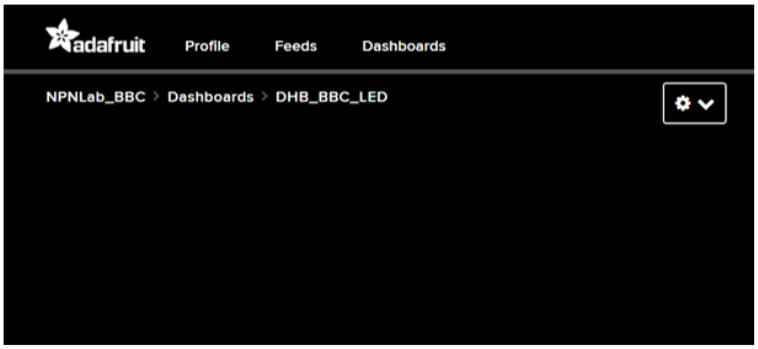

Figure 2.7: Dashboard initialization interface

Before designing objects on Dashboard interface (in this chapter, the object is a button), we need to configure Dashboard to be **Public**, so we can share on many different devices, such as mobile phone or tablet, in order to remote control and monitor system.By clicking on the settings icon on the right side of Dashboard, we click on **Dashboard Privacy** icon, as the following instructions:

Figure 2.8: Customization for Dashboard in Public

3 Design Dashboar interface

A new default Dashboard interface doesn't have any elements, we will put a button on the interface by clicking on the Settings icon, and choose **Create New Block**, as follows:

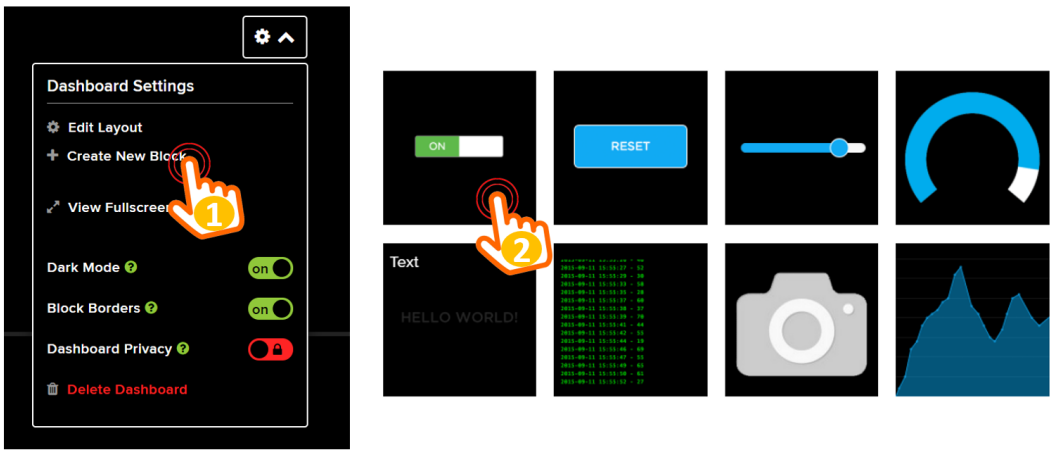

Figure 2.9: Add a new element to Dashboard

The Dashboard support interface objects are plentiful. However, in this section, we choose the first object. This object is called **Toggle Button**, and it is totally appropriate for application which turn on/off a device in the future lesson. After that, this interface will appear.

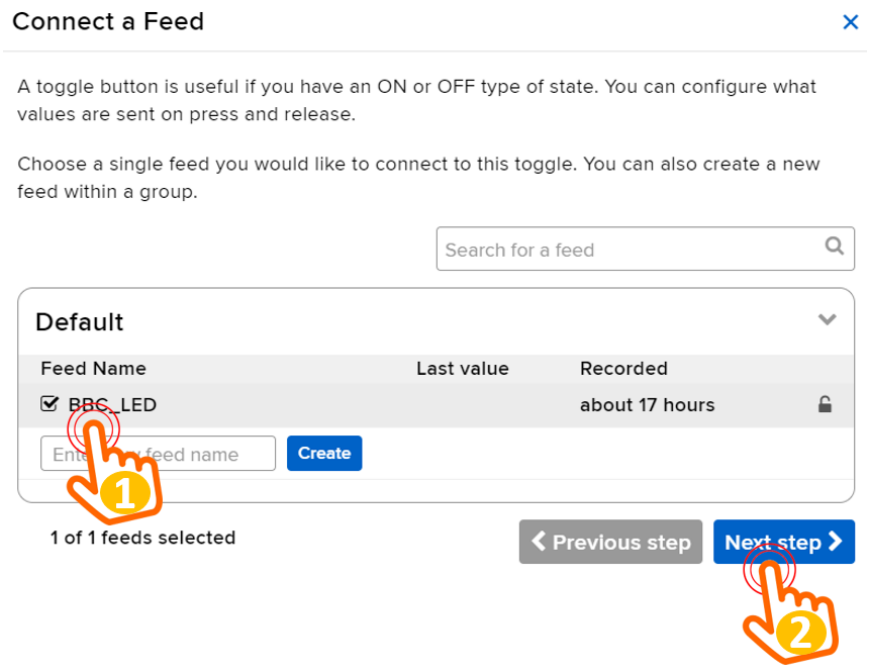

Figure 2.10: Connect to created Feed

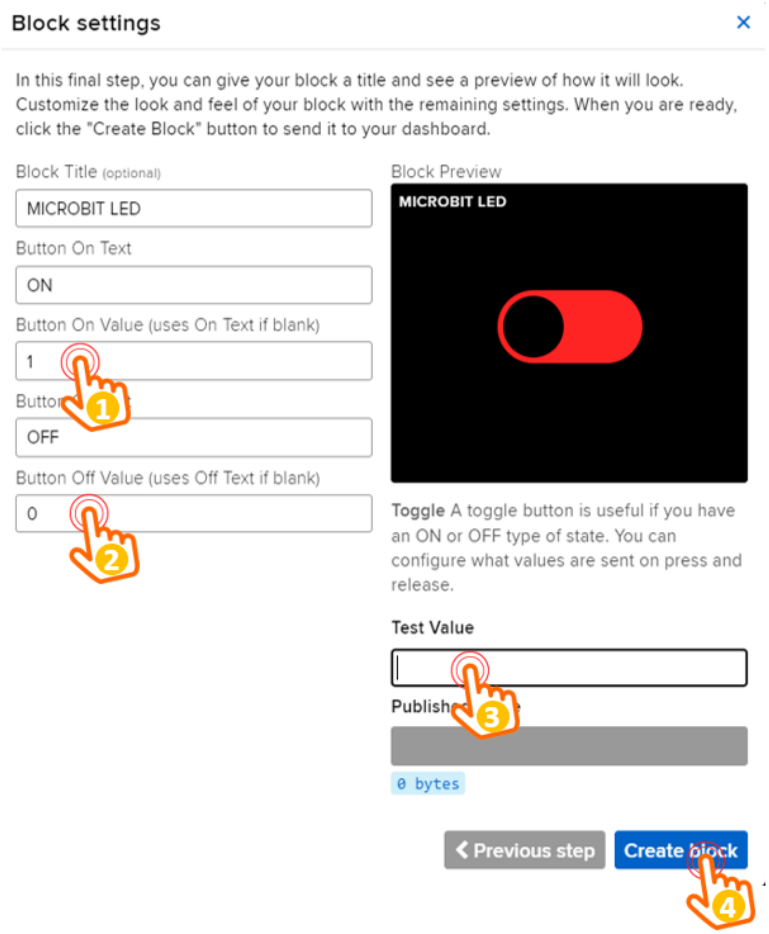

Figure 2.11: All settings functions for Button object

This is the most important step in designing interface for Dashboard. Up to this lesson,

because we only have one Feed (is BBC_LED), it is easy to choose. In case of multiple data feeds, you need to have a right decision. Then we click **Next step** button and continue to customize the settings for the interface below.

The information function of object is summarized as follows:

- **Block Title**: This part is optional, not required. Here we can give the button a name.
- **Button On Text**: In this part, we will have text on button if it on. At default settings, if the state of the button is on, button will show the word "ON".
- **Button On Value**: This path allows us to customize data sent when this button is clicked to be on state. For simplicity, **we will specify the data for it as 1**.
- **Button Off Text**:In this part, we will have text on button if it off. At default settings, if the state of the button is off, button will show the word "OFF"..
- **Button Off Value**: This path allows us to customize data sent when this button is clicked to be off state. For simplicity, **we will specify the data for it as 0**.

Besides, readers can check the change of the interface of the button, every time data is sent to Feed, by typing 0 or 1 on **Test Value** box. This function simulate button in reality. Finally,we click **Create block** icon and new button will appear in your Dashboard, as follows.

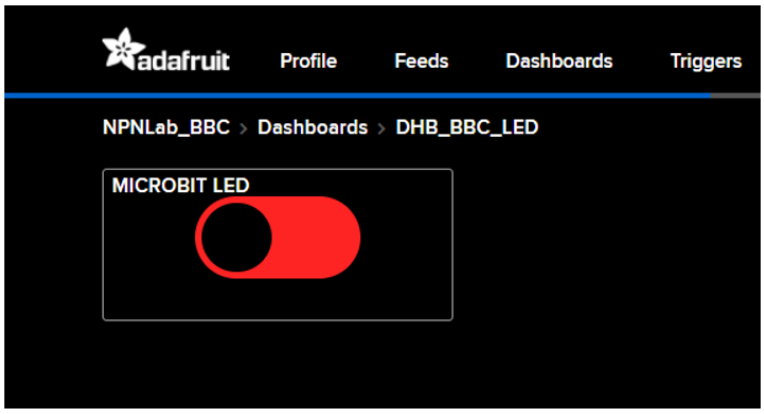

Figure 2.12: A new button is created on Dashboard

4 Edit button

In some cases, we will need to reset the button object, for example, we can make it bigger, change the value of on/off state, or delete this button object. it is able to make a few adjustments by click on the Dashboard settings icon, and select **Edit Layout**, as instructed below:

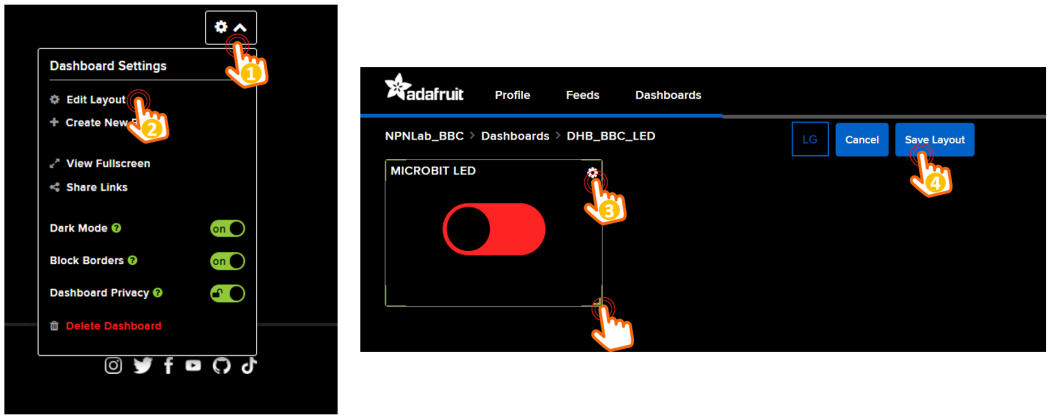

Figure 2.13: Edit button object

Interface objects on Dashboard will have their own settings icons. When we click this icon, and click **Edit Block**, an old interface introduced at Figure 2.11 will appear for us to change object configuration. The delete function will also appear in the list when we click this icon.

In case you want to change the size of the object, you simply drag and drop its 4 corners to change. Finally, press **Save Layout** button.

5 Check the connection between Feed and Dashboard

This is the final configuration check on Adafruit IO, before coding and sending data from Gateway to Feed. The testing process will be in the following order:

Step 1: From Dashboard interface, you should try pressing the button on the interface, remembering the times you press, corresponding to the ON and OFF states of the button.

Step 2: if you reopen the Feed interface,you will see raw data, which is 0 and 1, send to Feed, as shown in the following figure.

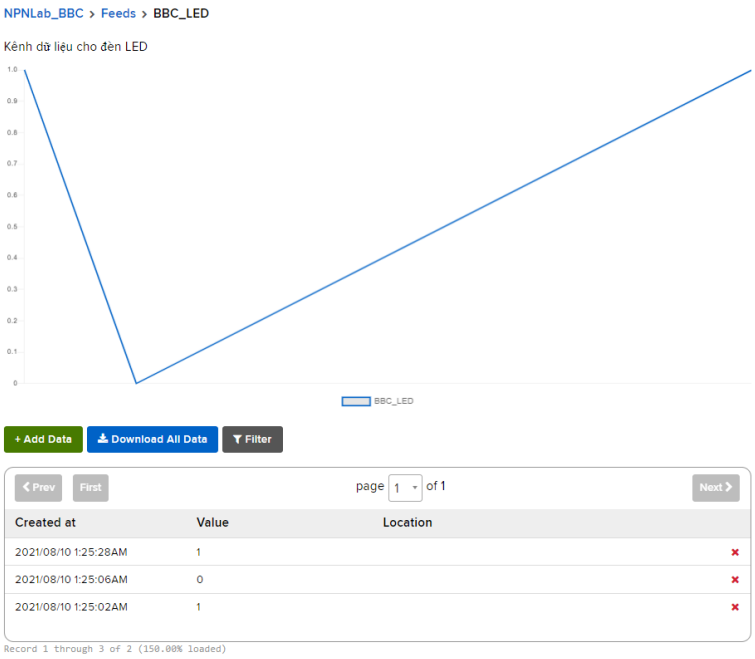

Figure 2.14: Data is sent to feed when it interact with Dashboard

Although a graph is plotted over the data channel, the reader should note its raw data, listed in a table below. Every time we press the button, its state will be saved, along with the time parameter. This raw information, will be sent down to the Gateway in the future..

Step 3: On the Feed, you press **+ Add Data** button. This interface will appear:

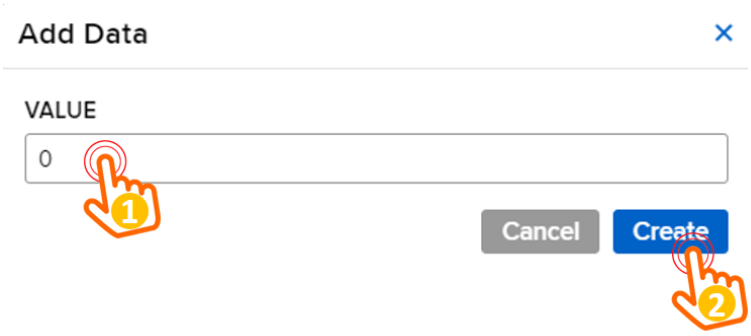

Figure 2.15: Add data to Feed directly

You enter valid values, in this case you can choose 0 or 1, then click the **Create** button. Obviously, A new data line will be added to the Feed. But at the same time, the interface of the Dashboard has also been changed accordingly.

Step 4: Checking system latency. This will be something new that Adafruit IO server have. You should try to share your Dashboard link with others, so that both parties can check remotely. We will see that the data communication between Feed and Dashboard has very low latency.

With Gateway IoT will be implemented in the next post, every time you interact on the Dashboard, the data will be sent to the Feed. Then, the Feed will automatically send down

to the IoT Gateway to execute this command. In the opposite case, when data is sent to the Feed from the IoT Gateway, Dashboad's interface will automatically update accordingly. Thanks to this mechanism, **Adafruit IO sever is suitable for network monitoring and remote control applications**.

6 Review questions

1. Which of the following objects can be used to monitor and control the system?
   A. Gateway IoT
   B. Feed
   C. Dashboard
   D. Adafruit IO

2. Which objects acts like a Back-End?
   A. Gateway IoT
   B. Feed
   C. Dashboard
   D. Adafruit IO

3. Which objects acts like a Front-End?
   A. Gateway IoT
   B. Feed
   C. Dashboard
   D. Adafruit IO

4. When configuring a button on the Dashboard, what is the value for the 2 states of the button?
   A. 0 and 1
   B. 1 and 2
   C. ON and OFF
   D. different from each others

5. The latency of data communication between the Feed and the Dashboard is:
   A. Fast
   B. Slow
   C. Average
   D. indeterminable

6. The latency of data communication between the Feed and the IoT Gateway is:
   A. Fast
   B. Slow
   C. Average
   D. indeterminable

7. The latency of data communication between Dashboard and IoT Gateway is:
   A. Fast
   B. Slow
   C. Average
   D. indeterminable

Solution

1. C 2. B 3. C 4. D 5. A 6. A 7. A

CHAPTER 3

IoT Gateway using Python

1 Introduction

In this chapter, we will make the functionality of an IoT Gateway by using Feed and Dashboard in the previous chapter. The program can not only work on the personal computer (PC) but also on embedded computer to create practical applications on Raspberry PI or Jetson Nano. This is a great advantage of Python because of compatibility with several different operating systems. Furthermore, the performance of Python on embedded systems seems better than normal computers. Source code of Python will be executed better in these computer with Linux operating system.

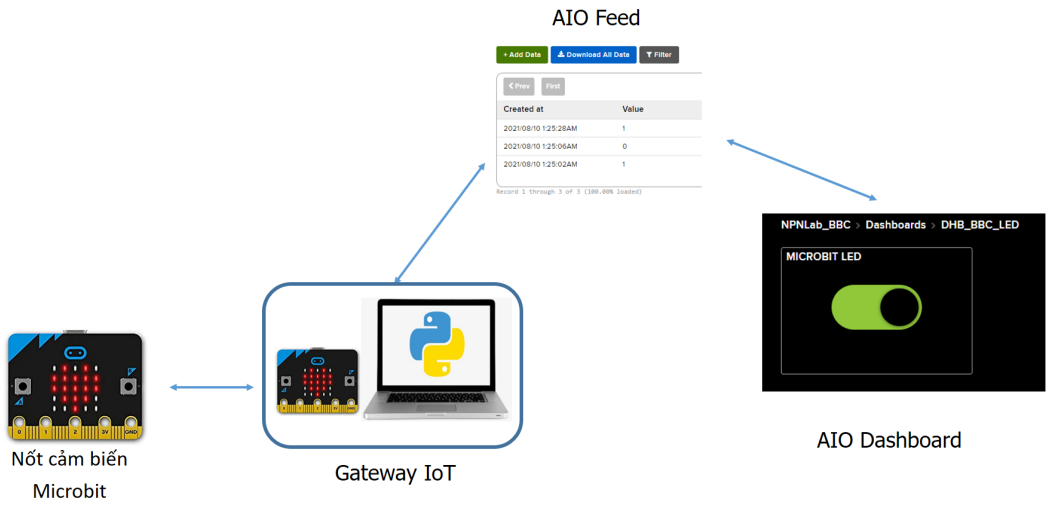

Figure 3.1: 4-level architecture in IoT

The architecture with 4 level in Internet of things is like the diagram above. The Feed on Ádafruit server become intermediary for transmitting data between Gateway IoT and Dashboard. In this chapter, we focus on create a program on computer which received data when user interact with Dashboard. This is new feature that you need for remote-controlled applications. This feature on common server like ThingSpeak wiill have significant delay. In other word, user must take at least 30 second waiting to turn on a light bulb after press its button by using ThingSpeak. However, this will done under 1 seconde witt server Adafruit IO.
The main purposes of this chapter are as follows:

- Setting the library for Adafruit IO server
- Make a Python program on computer
- Check the program with Dashboard

2 Adafruit Information

Gateway need information from adafruit to connect with Feed to transmit data to feed. From task bar on Adafruit IO, after you log in into the systems, this information will be on c **My Key**, like the picture below:

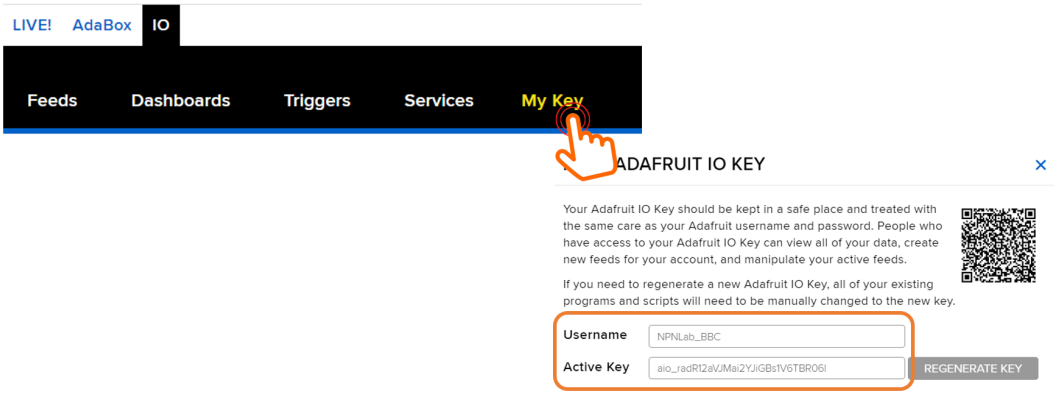

Figure 3.2: Adafruit information of account

The information will be shown only you click on **My Key**, so that we need to save **Username** and **Active Key**.

Next, you need to check the Feed's name of the data accessed by type **Active Key**. This name will be create automatically by system and different with the name for Feed in the previous chapter. You choose **Feeds** on the task bar and Feed info to access this information, for instance: with feed **BBC_LED**, the below display will be shown:

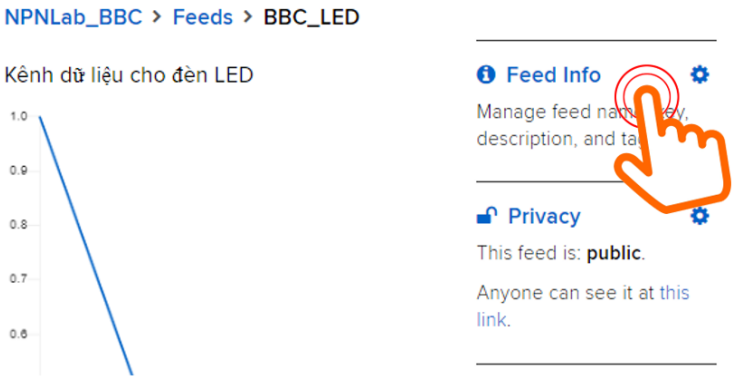

Figure 3.3: Check information of Feed

After click on **Feed Info**, another display will be shown. Here, we have the feed channel for the program, as like an example below:

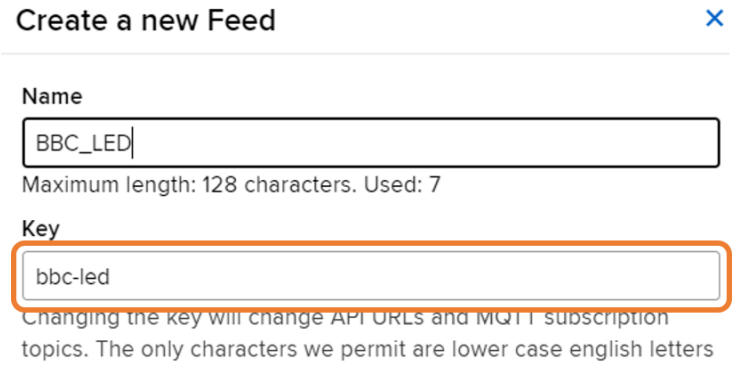

Figure 3.4: Name of accessed Feed

Some information you need to save before create a program is that Feed's name in type key (usually lower case Feed's name), Username and Active Key. These information whill be used when the Python program executed.

3 Project on PyCharm

Now, we use PyCharm IDE to create functionality program for Gateway IoT on PC. PyCharm is one of the most application for Python developer because of its compatibility with many different operating systems Windows, Linux or MacOS. This IDE provide us a virtual environment to easily install extend library for some advanced features. We use python with version 3.8 in this example.

Figure 3.5: PyCharm IDE

Installation instructions for the PyCharm software as well as the Python compiler program can be found in the Basic Python curriculum, shared at the following link:

https://drive.google.com/file/d/1dJLE3CdRJvU2QUOfjqMcX6azHMBYBl_q/view

After starting PyCharm, we choose **File/New Project**, as shown in the picture below:

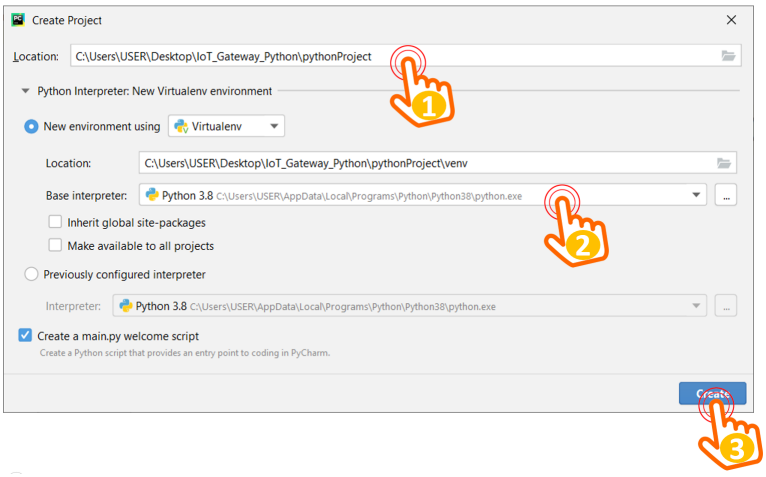

Figure 3.6: Create a project on Pycharm

You should note the path to save the project. Firstly, you should create a new folder to save all projects in this folder. We also do not forget to check the version of the Python interpreter before clicking the **Create** button. In case you have multiple versions of Python installed on your computer, there will be a selection list for you in the section **Base interpreter**.

A message window may appear after you click **Create** then choose to create a new project in the current window of PyCharm so as not to have to open multiple windows at the same time, by clicking **This Window**, as shown below:

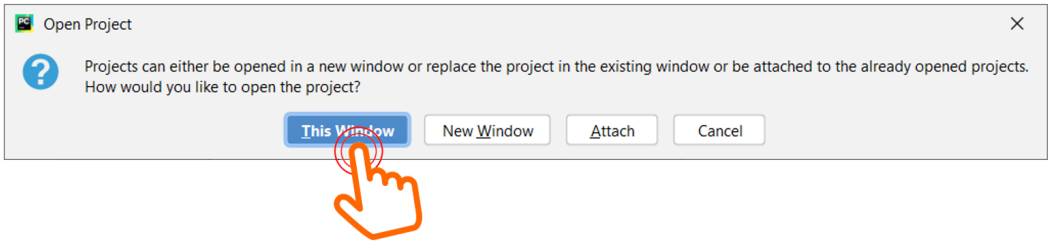

Figure 3.7: Create a project at the current window of PyCharm

Once the project is created, you can delete the default program content in **main.py**, to start implementing the program for the IoT Gateway.

4 Install library

One of the indispensable first steps for advanced features with the Python language is to install the library. The extension library we need to integrate is **adafruit-io**. There are 2 ways for us to install this library: Online installation using the **pip install** tool and installing from our self-developed GIT source code. The first method will be more familiar with practician. However, for those who are just starting out or method 1 is not successful, you can try method 2. Git server, which we maintain regularly and provide the most suitable library for you to read. . The details for the 2 methods of installing the library are presented as follows:

4.1 Install by pip install

By using this method, we will need to open the **Terminal** window on PyCharm, and type **pip install adafruit-io**, then press Enter, as shown below:

Figure 3.8: Install library Adafruit IO

PyCharm will download the library to our computer and install it. After successful installation, the following display will appear:

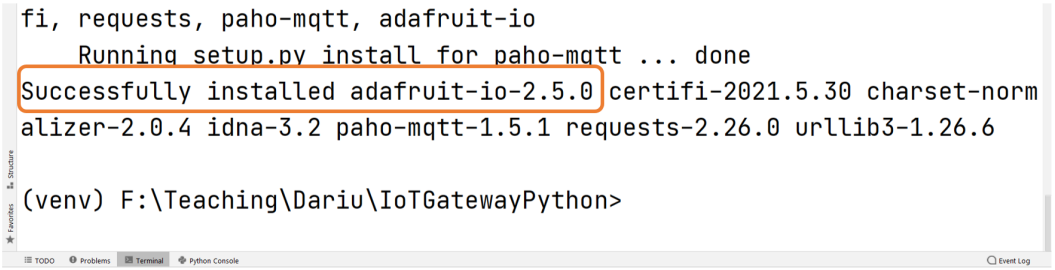

Figure 3.9: Install successfully

Up to this step, we are ready to implement the program. If the installation of the library fails, you can try the second method, in the next section.

4.2 Using GIT

or this second method, you need to download a ZIP file from the GIT server we built. First, visit the following homepage:

https://github.com/npnlab-vn/python_libs

Then the interface as shown below will appear, with a lot of libraries built by us for each course.

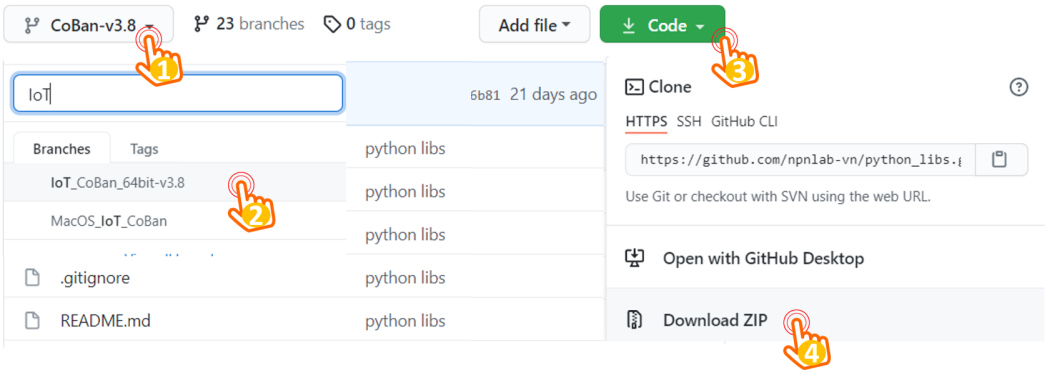

Figure 3.10: Library Python on GIT

Step 1: You need to select the correct folder before downloading the library to your computer, by clicking the option at position 1. Then, navigate to the **IoT_CoBan_64bit-v3.8** folder. For machines with different configurations, or other Python versions, you can choose the appropriate library for their computer.
Step 2: Once we have selected the correct folder, we can download the library, by clicking the **Code** button and selecting the ZIP file download mode in **Download ZIP**. You need to choose the path to save the downloaded file.
Step 3: Extract the downloaded ZIP fil by right-clicking and selecting **Extract Here**. With the newly extracted folder, continue to press right to copy (or press the hotkey Ctrl+C), as shown below:

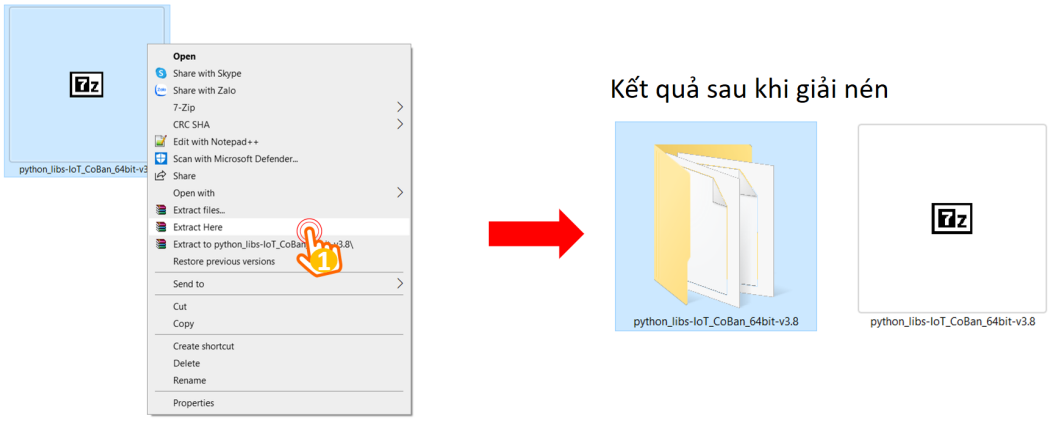

Figure 3.11: Extract library by Extract Here command

Step 4: Copy the extracted folder, and paste it into the project folder created in the previous section. In this directory, you will see the file **main.py**, like the following result:

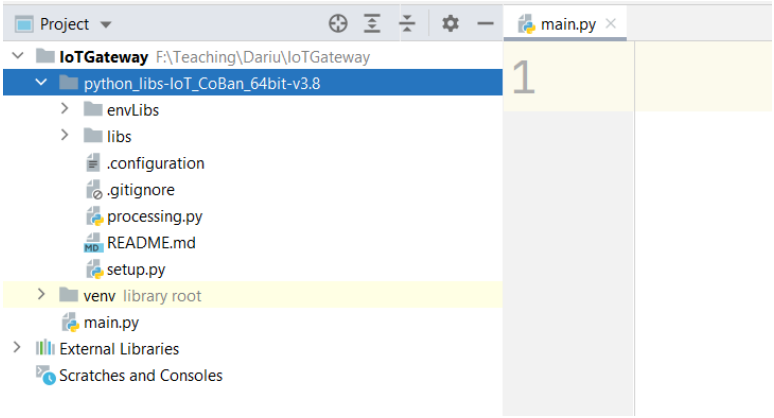

Figure 3.12: The library has been added to the project

After this step, when we return to the PyCharm programming environment, we will see the libraries and attachments appear in the project window.
Step 5: Right click on the file **setup.py** and select **Run**, to install the library, as shown in the image below:

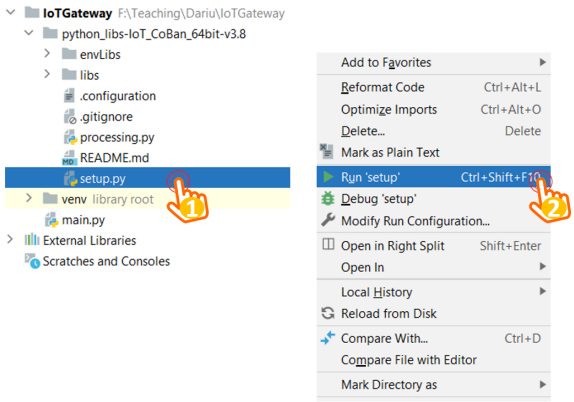

Figure 3.13: Install by running file setup.py

After the installation is complete, the following message will appear:

```
Installing collected packages: urllib3, idna, charset
Successfully installed adafruit-io-2.5.0 certifi-202
==================================
Successfully installed packages!!!

Process finished with exit code 0
```

Figure 3.14: Install successfully

In case the installation fails, you need to check their Python version. When creating the project, this information will be located in the **Base Interpreter** section to reload the library accordingly.

5 Implement program

Now, we move to work in the file **main.py** to implement the program's functionality. The actual steps in this file are detailed as below.

5.1 Import librayry and initialize

In this step, the most important thing is to double check that the installation of the library is really complete or not. Besides, you also need to fill in their data Feed information, as well as the Username and Key of the account. The first command lines will look like this:

```
import sys
from Adafruit_IO import MQTTClient

AIO_FEED_ID = "bbc-led"
AIO_USERNAME = "NPNLab_BBC"
AIO_KEY = "aio_radR12aVJMai2YJiGBs1V6TBR06l"
```

Code 3.1: Import library and init

You can execute the program in the file **main.py**, by **right-clicking on this file and selecting Run**. From the second, you can select the Run button on the PyCharm toolbar.

5.2 Functional implementation

The connection between IoT Gateway and Adafruit Server is based on a special protocol, called MQTT (Message Queuing Telemetry Transport). This is a publish/subscribe-based communication protocol, dedicated to Internet of Things (IoT) devices. The following four implementation functions serve to operate the MQTT protocol at Gateway IoT. The implementation of these functions is presented as follows:

```
def  connected(client):
    print("Ket noi thanh cong...")
    client.subscribe(AIO_FEED_ID)

def  subscribe(client , userdata , mid , granted_qos):
```

```
    print("Subcribe thanh cong...")

def  disconnected(client):
    print("Ngat ket noi...")
    sys.exit (1)

def  message(client , feed_id , payload):
    print("Nhan du lieu: " + payload)
```

Code 3.2: Functional Functions

When connecting successfully with the server, the Gateway will subscribe to a data channel to receive data from it. When there is data from any source sent to the Feed, this data will be automatically sent to the IoT Gateway, and the message function will automatically run without processing much.

5.3 Configuration for Gateway

The final step in the Gateway implementation, is to create an MQTT Client object, so that it can be linked to the functions created above. Additional commands to the program are presented as follows:

```
client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()

while True:
    pass
```

Code 3.3: Configuration for the MQTT Client object

At the end of the program, we need **an endless loop** so that the program does not terminate. Thus, it listens to the information sent back from the Adafruit server and automatically calls the **message** function for us. Since in this endless loop, we don't need to perform any function yet, so the pass command is added to correct the syntax in the Python language.

The program in this article is shared at the following link:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab3/IoT_Lab.py

5.4 Demo

Now, we will execute the file **main.py** to test its functionality. Once launched, the Gateway will connect to the Adafruit IO server and then subscribe to the Feed **BBC_LED** channel. The following information will be printed to the screen if these two tasks are successful:

```
F:\Teaching\Dariu\IoTGatewayPython\venv\Scripts\python.exe F:/Teaching/Dariu/IoTGatewayPython/main.py
Connected to Adafruit IO!
Ket noi thanh cong...
Subcribe thanh cong...
```

Figure 3.15: Gateway connect and subscribe successfully

Now, you can reopen their Dashboard and interact on the push button to illustrate sending the On/Off command. Almost immediately, this command will be passed down to the IoT Gateway, as shown in the image below:

```
F:\Teaching\Dariu\IoTGatewayPython\venv\Scripts\python.exe F:/Teaching/Dariu/IoTGatewayPython/main.py
Connected to Adafruit IO!
Ket noi thanh cong...
Subcribe thanh cong...
Nhan du lieu: 1
Nhan du lieu: 0
```

Figure 3.16: Gateway receive data from Dashboard

Obviously, all of this data is saved on the Data Feed, which you can check easily.

6 Review questions

1. What information is needed for Gateway programming?
   A. Feed's name in type Key
   B. Username
   C. Active Key
   D. All information above

2. Gateway IoT connects to which of the following objects?
   A. Server Adafruit IO
   B. Data Feed
   C. Dashboard
   D. All answers are correct

3. Gateway IoT registers with which of the following objects?
   A. Server Adafruit IO
   B. Data Feed
   C. Dashboard
   D. All answers are correct

4. When configuring a button on the Dashboard, to which object is the value sent?
   A. Gateway IoT
   B. Data Feed
   C. Dashboard
   D. All objects above

5. What is the connection protocol between IoT Gateway and Feed?
   A. HTTP
   B. HTTPS
   C. MQTT
   D. All answers are correct

6. What is the central mechanism of the connection between the IoT Gateway and the Feed?
   A. request/response
   B. ask/wait
   C. publish/subscribe
   D. Unable to determine

7. What is Gateway IoT workflow?
   A. Connect - Subcribe - Receive data
   B. Connect - Subcribe - Sent data
   C. Connect - Subcribe - Sent or Receive data
   D. Unable to determine

Solution

1. D 2. A 3. B 4. B 5. C 6. A 6. C 7. C

CHAPTER 4

Publish Data to Adafruit IO

1 Introduction

In this tutorial, the opposite direction of communication with the data sent from the IoT Gateway to the Data Feed will be presented. This need will be needed when readers want to build an application that periodically monitors the system's data remotely. For example, we can monitor the temperature of the incubator every 30 seconds. This data will be transformed to a graph on the Dashboard. For instance, when the temperature reaches the desired threshold, the reader can turn off the heater to save electricity. Readers may think that a smart system can turn itself off when the desired temperature is reached. However, with real systems, the need for manual control is always necessary for possible exceptions.

In this lesson, We will design an interface of the Dashboard to display temperature. Of course, readers can completely send other information, depending on the application they want to deploy. The interface of the Dashboard will be as shown below:

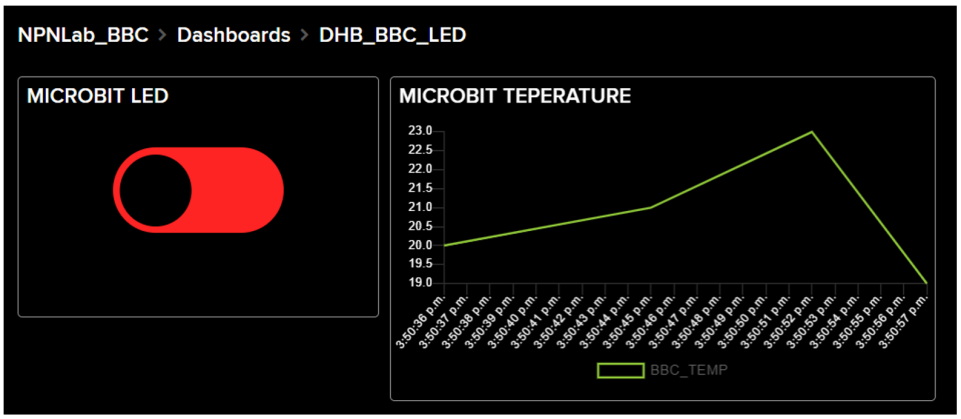

Figure 4.1: *Dashboard graph interface*

With MQTT protocol, every time the IoT Gateway sends data, you will see this data immediately update to the graph. This is very different from traditional sever such as ThingSpeak. Normally, ThingSpeak has high latency, about more than 5 seconds for data to be updated on graph. However, since we are using a free account from Adafruit IO, there is also a limit on the number of times we can send data to the server. Currently, we can only send no more than a packet per second. So, for monitoring applications, which does not necessarily send data continuously, you can send data with 30 seconds or even 60 seconds. We will reserve network resources for push buttons to control the device below the IoT Gateway.
The main purposes of this chapter are as follows:

- Create another feed
- Create graph interface on Dashboard
- Program and send data from Gateway to Feed

2 Create new Feed

Before creating a graphical interface on the Dashboard, you need to create a data channel (Feed) for it. The process of creating an data channel is completely similar to the instructions in the previous . In this lesson, you will create another Feed, called **BBC_TEMP**. You need to go back to **Feeds** section, click **View All** and finally select **New Feed**.

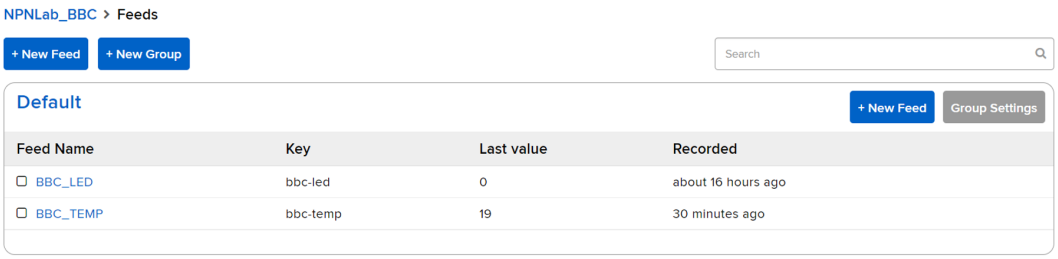

Figure 4.2: Create new Feed in your account

After creating new Feed, the result will be shown in the image above. However, you need to access directly to Feed in order to change the privacy mode to **Public**. This operation was described in the previous post and will not be detailed here.

3 Add graph to Dashboard

After creating a new data channel for the project, readers can start to rebuild the Dashboard, with the added graph object. From the main menu, you select **Dashboard** in order to access the details of the current Dashboard (DHB_BBC_LED), as follows:

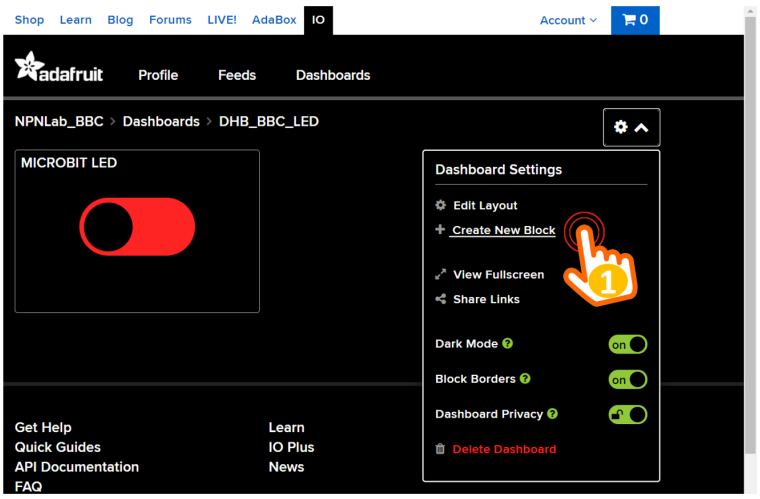

Figure 4.3: Interface design for Dashboard

From the settings icon in the right corner, Readers add new interface objects by clicking **+ Create New Block**. With the interface objects supported on Adafruit IO, we will select the graph object, as shown below:

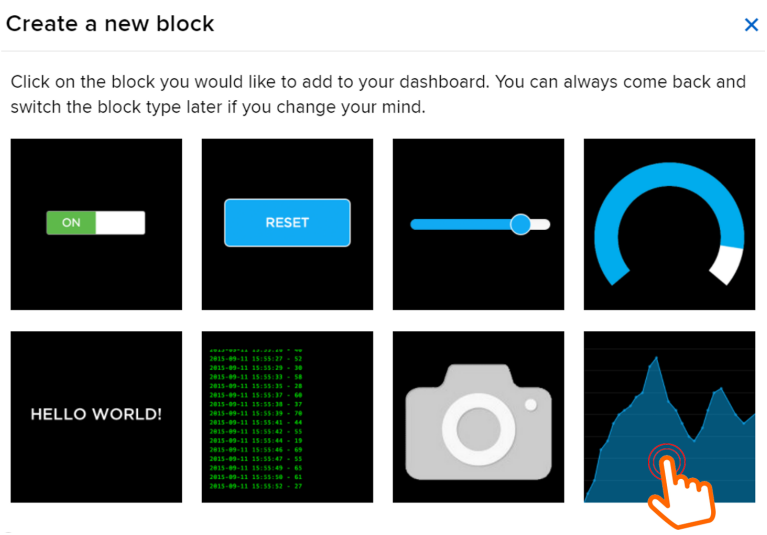

Figure 4.4: choose interface graph

After that, an interface will appear, so that we can link the graph to the Data Feed. Here, we will select Feed **BBC_TEMPERATURE**, as shown in the figure below:

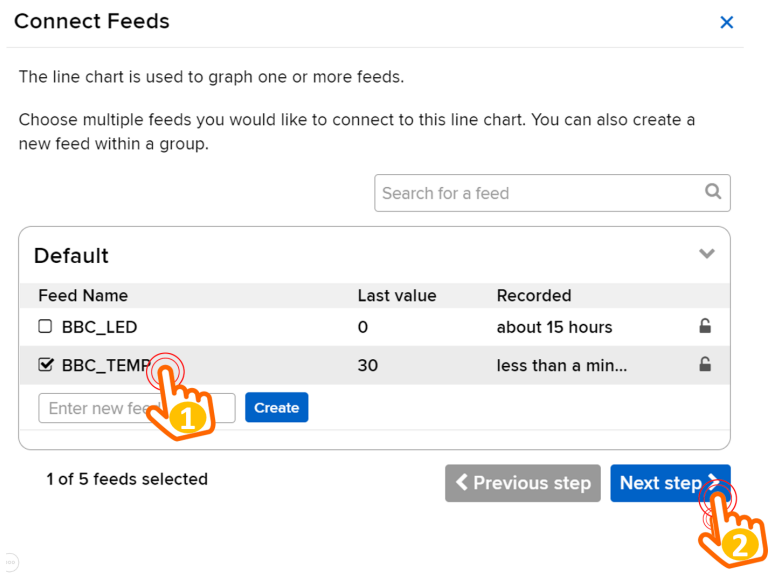

Figure 4.5: Select the Data Feed associated with the graph

Press **Next step** button to configure the display of the graph. The interface is shown below:

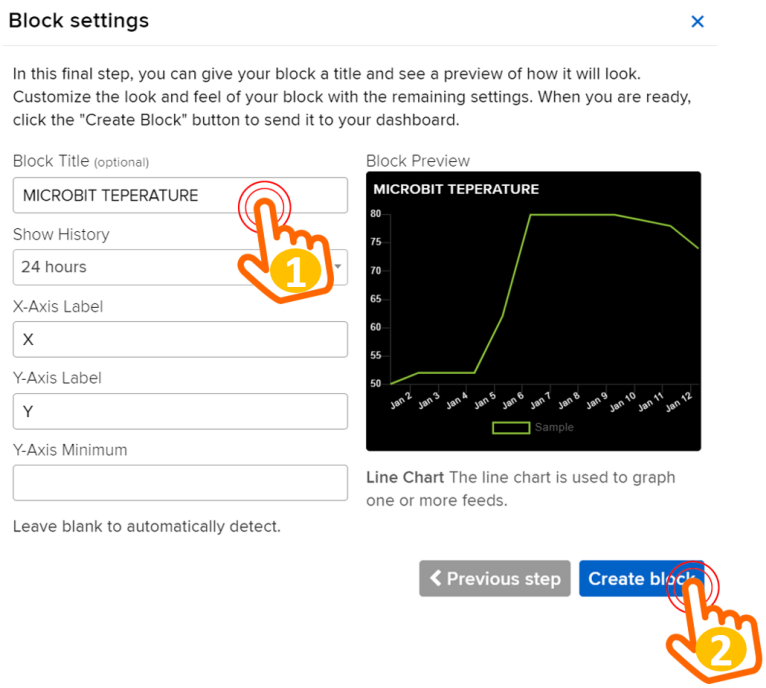

Figure 4.6: Configuration displayed on the graph

There are many information that you can change. However, the configuration information is in default mode without changing anythings. Here, we simply add information to **Block Title**, and press **Create block** button at the end of this interface settings. As a result, a graph will be added to the Dashboard as follows:

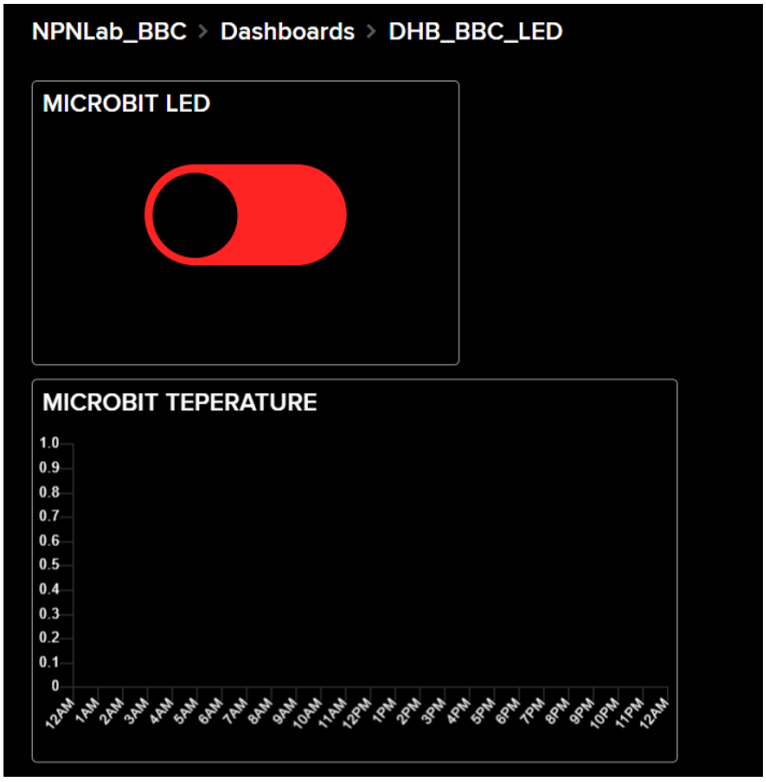

Figure 4.7: Configuration displayed on the graph

By default, the newly added graph will be arranged vertically. In case you want to rearrange it in another location, or even change the size of the graph, the operation is similar to the way you do it with button: click settings icon, then select **Edit Layout** icon, then drag and drop the object on the Dashboard to change. Finally, we press **Save Layout** button to finish the operation.

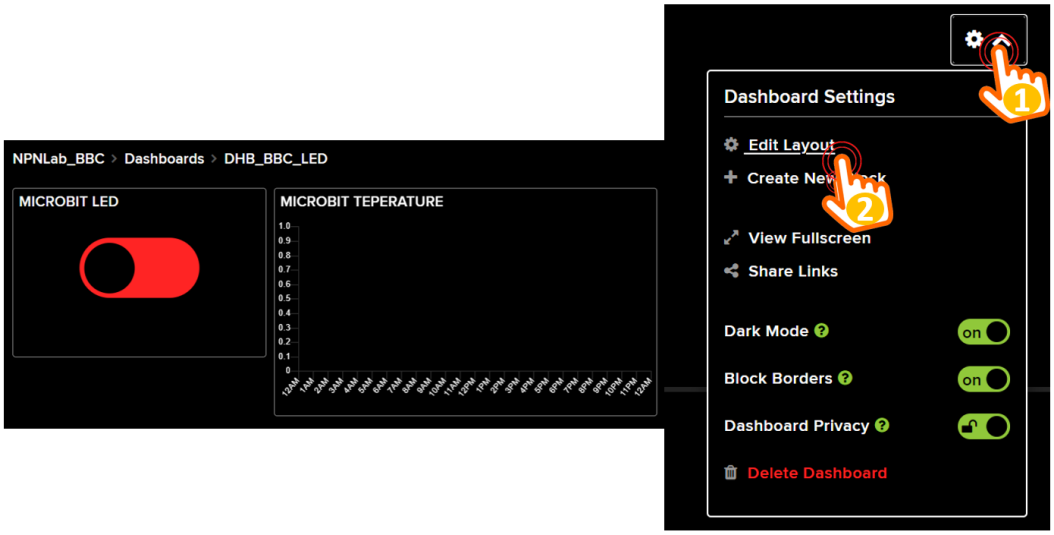

Figure 4.8: Change the layout of the interface on the Dashboard

4 Check the interaction between Feed and Dashboard

This is the required step, before starting to program the Gateway. We need to make sure that the interaction between Feed and Dashboard is stable. This work can be done by adding data manually to Feed **BBC_TEMP**, as introduced below:

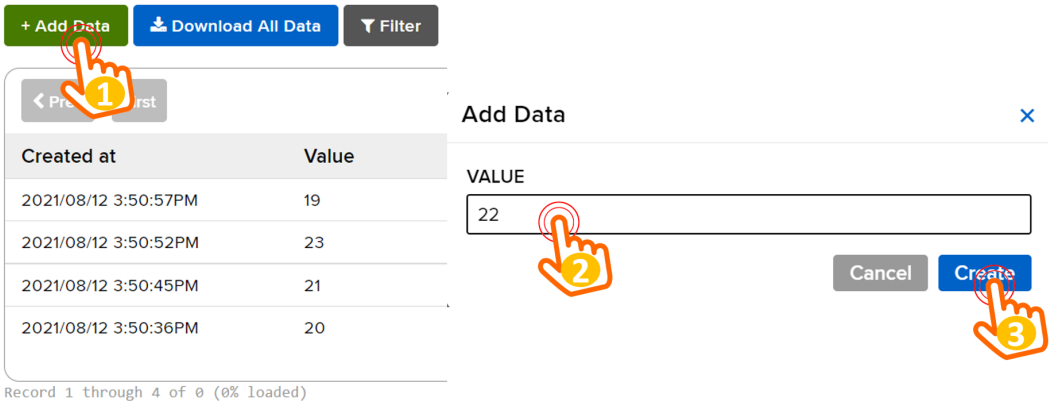

Figure 4.9: Add data manually to Feed

Once some test data has been added, readers can reopen the Dashboard. If the installation on the Dashboard is successful, a graph will be drawn according to the values added manually on the Feed, as follows:

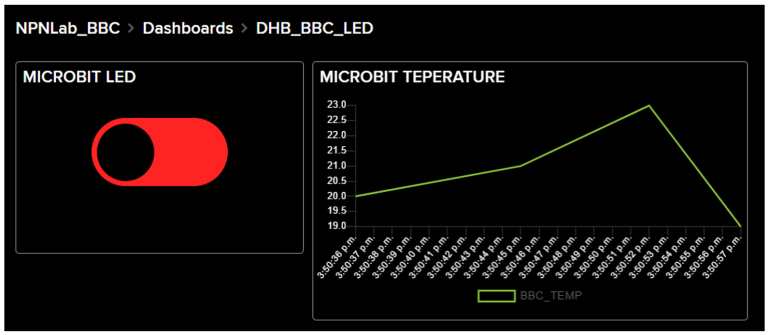

Figure 4.10: Graph interface on Dashboard

5 Implement Gateway

After completing all tasks on Adafruit sever, which include Feed and completed Dashboard, We can start programming for Gateway to send data periodically to sever. As shown in the introduction, we will send a random value (simulated for temperature) every 30 seconds. You need to review the information of the Feed to use it for programming. In this case, this feed is called **"bbc-temp"**.

When we send data to server, We don't have to access to Feed channels. The commands in AdafruitIO library allow us to send data(publish) by specifying the name of the Feed directly.

The first step of programming, we need two available libraries in the system to set time and get random numbers. You can add two commands at the begin of the program:

```
import random
import time
```

Code 4.1: Add libraries to system

Next, we will change the program in the infinite loop at the end of the program, like this:

```
while True:
    value = random.randint(0, 100)
    print("Cap nhat:", value)
    client.publish("bbc-temp", value)
    time.sleep(30)
```

Code 4.2: Add libraries to system

At the moment, you can check on Feed or Dashboard. Every 30 seconds, a random data will be sent. Meanwhile, every time pressing the button on Dashboard, data is immediately sent to Gateway. The complete program is shared at the following link:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab4/IoT_Lab.py

The source code of the program is shown below:

```
import random
import time
import  sys
from  Adafruit_IO import  MQTTClient

AIO_FEED_ID = "bbc-led"
AIO_USERNAME = "NPNLab_BBC"
AIO_KEY = "aio_radR12aVJMai2YJiGBs1V6TBR06l"

def  connected(client):
    print("Ket noi thanh cong...")
    client.subscribe(AIO_FEED_ID)

def  subscribe(client , userdata , mid , granted_qos):
    print("Subcribe thanh cong...")

def  disconnected(client):
    print("Ngat ket noi...")
    sys.exit (1)

def  message(client , feed_id , payload):
    print("Nhan du lieu: " + payload)

client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()

while True:
    value = random.randint(0, 100)
    print("Cap nhat:", value)
    client.publish("bbc-temp", value)
    time.sleep(30)
```

Code 4.3: Gateway's two-way communication program with Server

6 Review questions

1. What is the mechanism for sending data from the Gateway to the server called?
   A. connect
   B. subscribe
   C. publish
   D. All of them are correct

2. Which of the following statements is false?
   A. To receive data from the server, it is necessary to subscribe to Feed.
   B. To send data to the server, it is necessary to subscribe to Feed
   C. Default Dashboard subscribed to Feed
   D. Dashboard is private.

3. In what structure is the periodic data sending in this article performed?
   A. while True
   B. connected()
   C. Subscribe()
   D. All of them are correct

4. Which of the following commands has the effect of waiting 5 seconds in Python?
   A. time.sleep(5)
   B. time.sleep(5000)
   C. delay(5000)
   D. pause(5000)

5. Which of the following commands is correct to send data to a Feed on Adafruit IO?
   A. client.publish("bbc-temp", 30)
   B. client.publish("BBC-TEMP", 30)
   C. client.publish("BBC-TEMP", "30")
   D. All of them are correct

6. To randomly generate an integer, what library is used?
   A. import random
   B. import randint
   C. import rand
   D. All of them are wrong

7. What is the required library for the wait effect?
   A. import time
   B. import delay
   C. import pause
   D. All of them are wrong

Solution

1. C 2. B 3. A 4. A 5. A 6. A 7. A

CHAPTER 5

Intergrate the Microbit Platform

1 Introduction

Up to this chapter, we have successfully established a 2-way communication between the gateway running on computer and the Data Feed of the server. In this part of the tutorial, Gateway will be connected with Microbit board by USB port. the Microbit connect with the computer is called **central Micriobit** to distinguish it from Microbit which acts as sensor node. This method has some advantages as follow:

- The Gateway are running on a computer. In the future, this process will be brought to a embeded computer. The general feature of every computer is common but not specialized. Connecting with a specialized system must be through USB connection.
- Connecting with a dedicated system will provides almost limitless scalability for computer system. In this occation, we want the capability of wireless communition of Microbit for Gateway IoT on computer.
- With wireless connection of Microbit,every sensor nodes using Microbit can easily send data wirelessly to Gateway.

In this lecture, we focus on the direction of communication from Gateway to microbit which directly connects to Gateway. In this case, when a button toggle on Dashboard, Microbit will display data 1 and 0 accordingly.

Although Microbit board connects with the computer via the USB connection, this method has another name, called virtual COM serial communication. This is the most common protocol for every microcontrollers (like Microbit) and embedded computer. The serial connection also fully supports many programming languages, from drag and drop to Python..

The main purposes of this chapter are as follows:

- Programming to receive data on Microbit circuit
- Install library for Serial communication
- Implement the program to send commands from Gateway for Microbit

For the convenience of readers, small Python programs are introduced in each section. However, at the end of this tutorial, we will cover the entire program. In the process of reading and understanding the document, readers can refer to the program at the end so that they can find the appropriate location to integrate the functional program.

2 Microbit's program

Obviously, when the Gateway will send a command, Microbit needs to have a program on it to receive the command. We will implement this program on the Microbit borad, before integrating it with the Gateway. This program will be executed on MakeCode, and the Microbit circuit will simply display 0 or 1 on its screen.

However, to make programming on MakeCode simpler, we will specify a command sent from GatewayIoT that will end with a special character #. This is a simple communication technique, based on a special character as a statement terminator. The program on MakeCode will look like this:

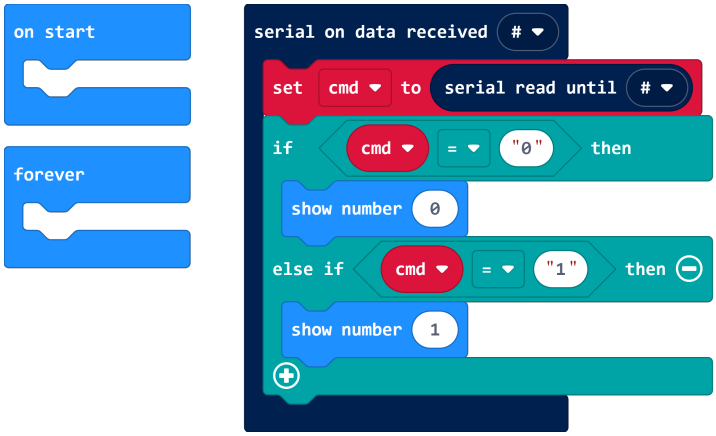

Figure 5.1: Program on central Microbit

For the time being, for the central Microbit board, we do not need any instructions in the **on start** block or the **forever** block. However, readers should keep it for further development in the future.

An important note, the data received from the serial communication is **string data**. Therefore, when you want to process an instruction, the string comparison operator needs to be used for precision (there is a character "" for string data). This program is shared at the following link:

https://makecode.microbit.org/_5Rj8TeeM1RFb

After download programm to Microbit, we can start program the Gateway IoT, with the detail steps are presented in the next section.

3 Implement Gateway

3.1 Add program library

As with all premium features in Python, adding libraries should be the first step to do. If you choose to install manually, the command from the Terminal window will be **pip install pyserial**, like this:

```
(venv) F:\Teaching\Dariu\IoTGatewayPython>pip install pyserial
Collecting pyserial
  Using cached pyserial-3.5-py2.py3-none-any.whl (90 kB)
Installing collected packages: pyserial
Successfully installed pyserial-3.5

(venv) F:\Teaching\Dariu\IoTGatewayPython>
```

Figure 5.2: Install Pyserial library

After successfully installing the library, readers can add the library import statement below at the beginning of the program.

```
1 import serial.tools.list_ports
```

Code 5.1: Add Pyserial library

3.2 Add library for serial communication

When connecting the Microbit circuit to the computer, it will be recognized as a virtual COM port. The most important thing is that we must determine the name of this COM port. As shown in the following figure, in Device Manager, the Microbit board is being connected to the COM9 port.

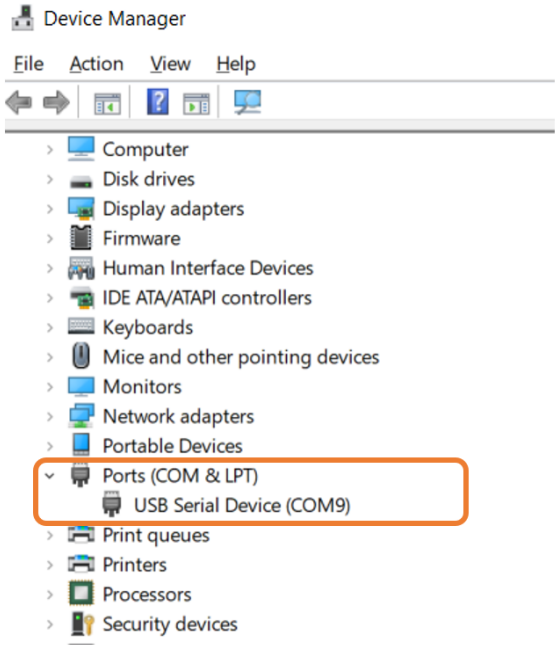

Figure 5.3: Name of COM port connect to Microbit

However, this COM port can completely be changed every time we restart the computer, or when we change the USB port. Therefore, we provide you with a function to automatically find this COM port, as instructed below :

```
def getPort():
    ports = serial.tools.list_ports.comports()
    N = len(ports)
    commPort = "None"
    for i in range(0, N):
        port = ports[i]
        strPort = str(port)
        if "USB Serial Device" in strPort:
            splitPort = strPort.split(" ")
            commPort = (splitPort[0])
    return commPort
```

Code 5.2: Find COM port to connect to Microbit

Besides, this program can work correctly when only one Microbit connect with the system. Thank to USB driver of this board, the program can automatically recognize it. In case the function do not work, readers need to check in Device Manager to see if the device's name has the keyword **"USB Serial Device"** or not.

The code for opening the Serial port to the Microbit board, using the function presented above will be as follows:

```
ser = serial.Serial( port=getPort(), baudrate=115200)
```

Code 5.3: Open connection with Microbit

In the above code, we specify the COM port name by calling the auxiliary function **getPort()** and the communication speed, configured to the default Microbit speed of 115200 bits/s

Once the connection can be successfully opened, sending data to the Microbit is very simple. Readers can refer to the program in the next section: When receiving data from the server, this data will be sent to the Microbit.

4 Intergrate with Microbit

The program for the system is now quite long now, presented below. Readers, please pay attention to the important part. In line 24: When receiving information from the Data Feed, the payload variable is packed to send to the Microbit. A special character **#** is added to the payload before being sent to the Microbit circuit. The program is shared at the following link:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab5/IoT_Lab.py

The source code of the program is presented below:

```
import serial.tools.list_ports
import random
import time
import  sys
from  Adafruit_IO import  MQTTClient

AIO_FEED_ID = "bbc-led"
```

```
AIO_USERNAME = "NPNLab_BBC"
AIO_KEY = "aio_radR12aVJMai2YJiGBs1V6TBR06l"

def  connected(client):
    print("Ket noi thanh cong...")
    client.subscribe(AIO_FEED_ID)

def  subscribe(client , userdata , mid , granted_qos):
    print("Subcribe thanh cong...")

def  disconnected(client):
    print("Ngat ket noi...")
    sys.exit (1)

def  message(client , feed_id , payload):
    print("Nhan du lieu: " + payload)
    ser.write((str(payload) + "#").encode())

def getPort():
    ports = serial.tools.list_ports.comports()
    N = len(ports)
    commPort = "None"
    for i in range(0, N):
        port = ports[i]
        strPort = str(port)
        if "USB Serial Device" in strPort:
            splitPort = strPort.split(" ")
            commPort = (splitPort[0])
    return commPort

ser = serial.Serial( port=getPort(), baudrate=115200)

client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()

while True:
    value = random.randint(0, 100)
    print("Cap nhat:", value)
    client.publish("bbc-temp", value)
    time.sleep(30)
```

Code 5.4: Open connection to Microbit

5 Review questions

1. What is the connection between Gateway and Microbit?
   A. Serial
   B. SPI
   C. I2C
   D. All of the above

2. What library is install to connect with Microbit?
   A. serial
   B. uart
   C. pyserial
   D. pip

3. How fast is the default speed of serial communication of Microbit?
   A. 4800
   B. 9600
   C. 115200
   D. None of the above

4. What character can be use as the end character for the command sending from Gateway to central Microbit?
   A. #
   B. :
   C. $
   D. All of the above

5. What is the command block to receive data on Microbit?
   A. on radio received number
   B. on radio received string
   C. serial on data received #
   D. None of the above

6. On Microbit, What is the type of the receive data from serial?
   A. Integer
   B. Float
   C. String
   D. Undefined

7. The request to send data from Gateway to central Microbit is implemented at what function?
   A. connected
   B. subscribe
   C. message
   D. getPort

Solution

1. A 2. C 3. C 4. D 5. C 6. C 7. C

CHAPTER 6

Transfer Data from Microbit to Gateway

1 Introduction

In this chapter, the second communication dimension of the Microbit circuit to the IoT Gateway will be realized. This function will have a much greater complexity than the tutorial function in the previous chapter. The problem here is that when a string of information is sent from Microbit to the Gateway, it will be cut off, but not received continuously. Our gateway will not know how many characters it received each time nor how many times it received. Therefore, the information sent from the Microbit circuit must also have a specified starting and ending character, to notify that the Gateway has received enough data and begins to process the information.

The limitation presented above is a general limitation of systems using operating systems, such as computers in general and embedded computers in particular. These systems, which have advantages such as stable network connection, support many new services on the Internet. However, some tasks, such as receiving data from the serial port, are the world of Microcontrollers like Microbit circuits.

Even so, we also have classical models to deal with for this. By encapsulating data in a predefined format, as shown in the figure below:

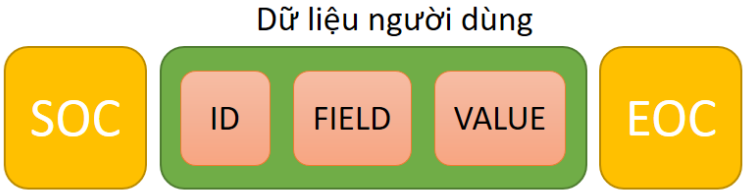

Figure 6.1: Data structure for communication with the computer

The structure above is a classic form in computer communication (Computer Communications). To apply in this course, we will specify the starting character (SOC = Start Of Character) as **"!"** and the ending character (EOC = End Of Character) as **"#"**. The middle information field, which we will define as **ID:FIELD:VALUE**, where ID is the identifier of a node, FIELD is the definition information, and VALUE is the value to send. These fields are separated by **":"** - the technical term called Escape Character.
The main purposes of this chapter are as follows:

- Implement program to send temperature data on Microbit
- Data Encapsulation on Microbit
- Decrypt data on Gateway
- Receive and decrypt data on Gateway
- Send data to Adafruit server

2 Program for Microbit

Before starting to implement the function of receiving data on the Gateway, we need to implement the function of sending data from the Microbit circuit. Here, we give every 30 seconds, the temperature information will be sent up. The specified data format for communication is **"!1:TEMP:xx#"**. In which, **1** is the ID information of the Gateway node, **TEMP** is the keyword that we define for the temperature information. Finally, is the temperature read from the sensor of the Microbit circuit. The packet ends with the character **#**.

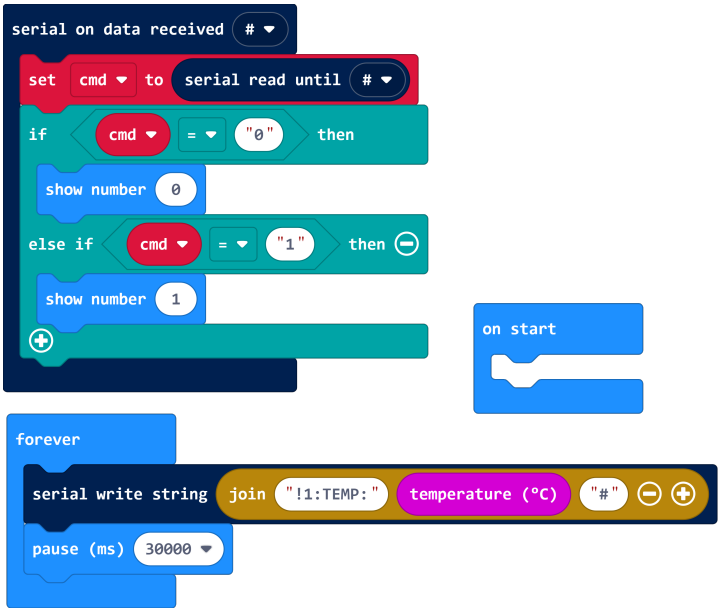

Figure 6.2: Program to send data to Microbit

Periodic sending is implemented in the **forever** block. The statement **serial write string** is taken from the group **Advance/Serial**. The above program is shared at the following link:

https://makecode.microbit.org/_AXYRzvURMJVY

3 Data decomposition function

As mentioned above, the data we receive will have to be parsed, based on the special character **:**. Other characters such as the start character **!** and the end character **#** will be removed. Depending on the keyword of the data, the information will be sent to the corresponding feed. This function will be implemented as follows:

```
def processData(data):
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    print(splitData)
    if splitData[1] == "TEMP":
        client.publish("bbc-temp", splitData[2])
```

Code 6.1: Split the data and send it to the feed

In a more detailed case, you can construct additional conditions of the **if** statement, such as checking that the number of elements in the array **splitData** must be equal to 3, for example. When the test conditions are valid, the data is sent to the feed using the **publish** statement. **You should note about the first parameter in the publish command, it is the FEED_ID of the data channel**.

4 Serial data reading function

This is the most important function to receive data from Microbit circuit. Because of the special nature of the computer, which does not know how many times the data will be sent, and how many characters each time, the processing of this function is relatively complicated. However, the main idea is to wait for the start character (**"!"**) and the end character (**"#"**) to be received before we begin processing, by calling the function **processData** above. The statements used in this function are mainly tasks related to string processing.

```
1  mess = ""
2  def readSerial():
3      bytesToRead = ser.inWaiting()
4      if (bytesToRead > 0):
5          global mess
6          mess = mess + ser.read(bytesToRead).decode("UTF-8")
7          while ("#" in mess) and ("!" in mess):
8              start = mess.find("!")
9              end = mess.find("#")
10             processData(mess[start:end + 1])
11             if (end == len(mess)):
12                 mess = ""
13             else:
14                 mess = mess[end+1:]
```

Code 6.2: Receive data from Microbit

In this function, we need a global variable, named **mess** to accumulate the data received from Microbit. You should notice how to specify a variable to be globally accessible with the command at line 5, **global mess**.

5 Integration into Gateway

Up to this point, we have been able to integrate the function into the Gateway program, with an endless loop that will periodically read data from the Microbit circuit. When the data is full, it will automatically be processed and sent to the Adafruit server. The timeout is being set to 1 second, using the command **time.sleep(1)**. Gateway's full program is shared at the following link:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab6/IoT_Lab.py

The source code for the program up to this step is presented as follows:

```
1  import serial.tools.list_ports
2  import random
```

```
import time
import  sys
from  Adafruit_IO import  MQTTClient

AIO_FEED_ID = "bbc-led"
AIO_USERNAME = "NPNLab_BBC"
AIO_KEY = "aio_radR12aVJMai2YJiGBs1V6TBR06l"

def  connected(client):
    print("Ket noi thanh cong...")
    client.subscribe(AIO_FEED_ID)

def  subscribe(client , userdata , mid , granted_qos):
    print("Subcribe thanh cong...")

def  disconnected(client):
    print("Ngat ket noi...")
    sys.exit (1)

def  message(client , feed_id , payload):
    print("Nhan du lieu: " + payload)
    ser.write((str(payload) + "#").encode())

client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()

def getPort():
    ports = serial.tools.list_ports.comports()
    N = len(ports)
    commPort = "None"
    for i in range(0, N):
        port = ports[i]
        strPort = str(port)
        if "USB Serial Device" in strPort:
            splitPort = strPort.split(" ")
            commPort = (splitPort[0])
    return commPort

ser = serial.Serial( port=getPort(), baudrate=115200)

mess = ""
def processData(data):
    data = data.replace("!", "")
    data = data.replace("#", "")
```

```
52      splitData = data.split(":")
53      print(splitData)
54      if splitData[1] == "TEMP":
55          client.publish("bbc-temp", splitData[2])
56
57  mess = ""
58  def readSerial():
59      bytesToRead = ser.inWaiting()
60      if (bytesToRead > 0):
61          global mess
62          mess = mess + ser.read(bytesToRead).decode("UTF-8")
63          while ("#" in mess) and ("!" in mess):
64              start = mess.find("!")
65              end = mess.find("#")
66              processData(mess[start:end + 1])
67              if (end == len(mess)):
68                  mess = ""
69              else:
70                  mess = mess[end+1:]
71
72  while True:
73      readSerial()
74      time.sleep(1)
```

Code 6.3: Integrate the program into the IoT Gateway

6 Review questions

1. To receive data from serial port, what is the limitation at Gateway?
   A. Don't know how many times the data will be received, how many characters each time
   B. Data is received only once
   C. Loss of data when reading from Serial
   D. All answers above are correct

2. What is the processing technique for receiving Serial data at the Gateway?
   A. Declare a global variable to accumulate data
   B. Process information based on the beginning and ending characters of the information
   C. Cut information based on special characters
   D. All of the above techniques

3. With the format **"!ID:KEY:VALUE#"**, which of the following character strings is valid?
   A. !1TEMP30#
   B. !1:TEMP:30#
   C. !2:HUMI:60#
   D. There are 2 valid strings

4. What effect does increasing the wait time in the while True loop have on the processing of receiving data from Serial?
   A. Lose data
   B. Increases the delay when sending sensor data to the server
   C. Doesn't matter
   D. It's all wrong

5. To distinguish from which node of the sensor network the data is sent, which of the following information should be considered?
   A. ID
   B. KEY
   C. VALUE
   D. All the above information

6. To make sense of data (temperature or humidity), which of the following should be considered?
   A. ID
   B. KEY
   C. VALUE
   D. All the above information

7. To access the sensor's value, which of the following information will be used?
   A. ID
   B. KEY
   C. VALUE
   D. All the above information

Solution

1. A 2. D 3. D 4. B 5. A 6. B 7. C

CHAPTER 7

Toggle Button on Dashboard

1 Introduction

Until now, we have built a system that has enough features at Gateway with the combination of the Python program running on computer and the connection of Microbit. Our system can receive the signal from a button when users have action on Dashboard and the data will be sent to the center Microbit which is connecting to the Python program running on our computer.

In this chapter, we will expand our Dashboard by adding a new button. It is important to note that in order to receive the data from Adafruit server, our Gateway need to subscribe to the feed, which is similar to the thing that we subscribe to some Youtube channels to get notification when they have new videos. In previous chapters, we have subscribed to the feed that is related to turning on and off the led. In this lecture, we will create a new feed to do something new such as turning on and off a water pump.

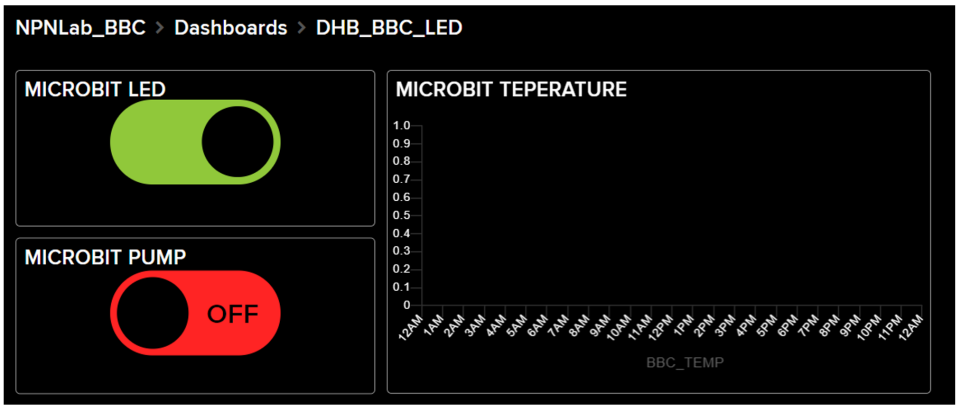

Figure 7.1: Adding a new button on Dashboard

We also re-implement the center Microbit in order that it can have the correct execution: Whenever the center Microbit receive a command from the Gateway, it will use the Radio feature to forward the command to the sensor nodes. In the other hand, when this center Microbit receive some data from sensor nodes (through Radio), it also send this data the Gateway to process. Using Microbit in the previous chapters is just for checking the communication feature of the Gateway.

The main purposes of this chapter are as follows:

- Completing the program for the center Microbit.
- Creating a new feed and adding a button on Dashboard.
- Modifying the program of the Gateway.

2 Program for the center Microbit

With two features defined in the introduction section, the center Microbit becomes completely passive. It does not send its sensor's data to the Gateway periodically anymore.

Instead, it will wait for the data from surrounding sensor nodes. At the same time, whenever there is command from Gateway, the center Microbit simply forward the command to the sensors nodes to execute. In some simple applications, you can use one Microbit in two roles center and sensors. However, in this tutorial, we will follow the general approach and use the **center Microbit as a bridge between sensor nodes and the Gateway**.

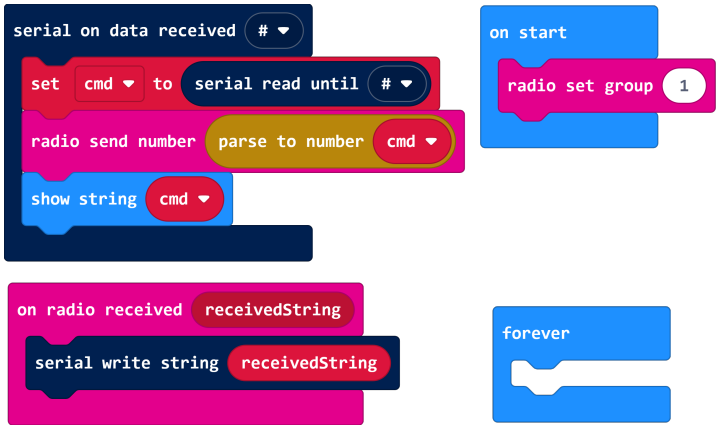

Figure 7.2: Program for the center Microbit

Program in the **forever** block will not exist anymore. Reading and packaging the data from the sensors will be left for the sensor nodes and will be presented in the next chapter. Meanwhile, when receiving a command from the Gateway, the center Microbit will change the data into numeric type before sending it to the sensor nodes with the **radio send number** statement. The group of Microbit circuits in one system will be defined in block **on start** with the **radio set group** statement.

When we use the center Microbit as a wireless communication bridge, we form a specialized wireless network which can meet the requirements of many applications. With this approach, if we want to change to other wireless communications, for instance Lora or ZigBee, it is feasible. Indeed, we just need to connect Lora device to all Microbit circuits of the system. The program for the center Microbit will be shared in teh link below:

https://makecode.microbit.org/_7dFKhAecaf4A

3 Create a new feed

This is the first step, before you create a new button on Dashboard. The steps to create a new feed is the same as we introduce in the previous chapter, so it is not presented in details again. Besides, you need to note that you must change the access of the feed to **Public** for the new feed. Here, we create a new feed the name **BBC_PUMP** using for storing the data to control the water pump.

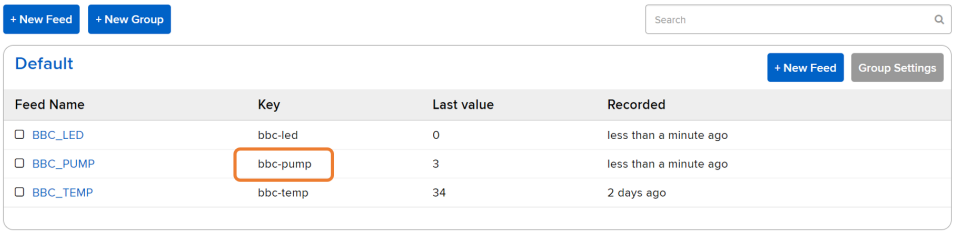

Figure 7.3: Adding Feed for water pump

You need to pay attention the key of the new feed to use it for programming, in our case here is **bbc-pump**.

4 Create a new button on Dashboard

In order to create a new button on Dashboard, you do the same steps in previous chapter: Click on the setting icon, choose **Create New Block** and then select the **button** icon. However, this time you will link your new button with the new feed which you just have created before. The result will look like the below figure:

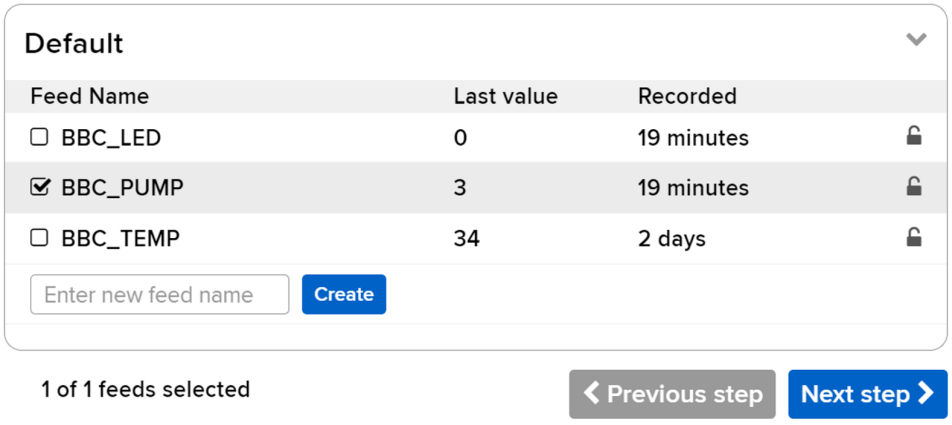

Figure 7.4: Adding a new button on Dashboard

Finally, you configure two different value for the button. With two values 0 and 1 for the button of the LED, we will choose 2 and 3 for this button. This will make the programming in the future much easier.

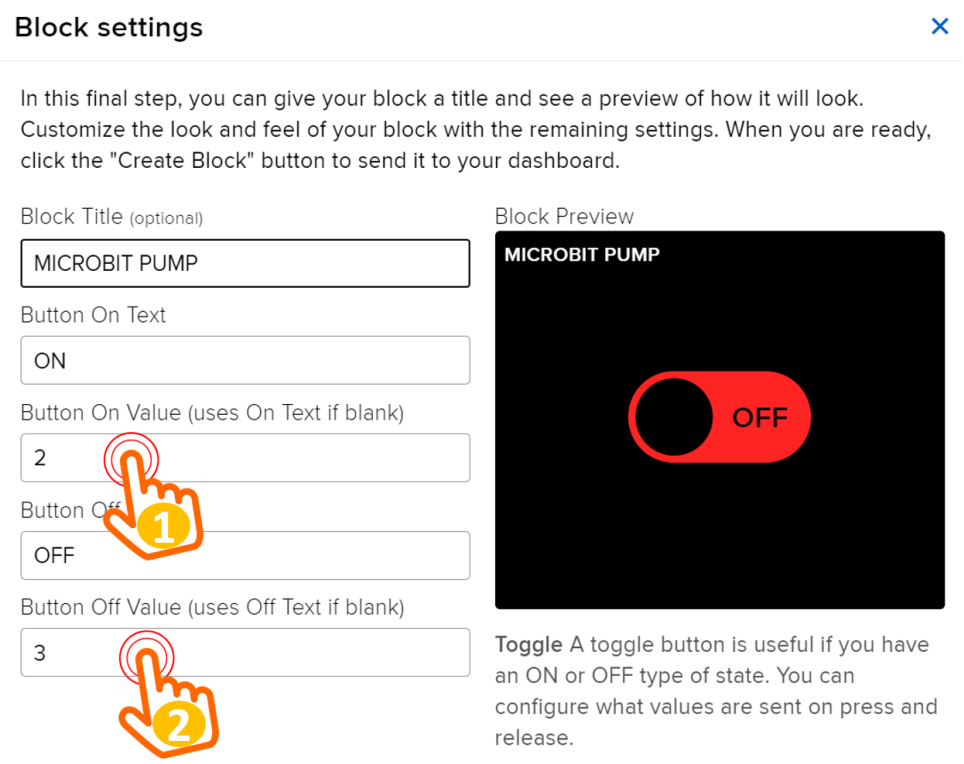

Figure 7.5: Adding a new button on Dashboard

After creating a new button, you need to check again the communication between Dashboard and Feeds. **If the communication between Dashboard and Feeds fails, you can try to Refresh the Dashboard one or two times.** After creating a new component on the Dashboard, it can behave incorrectly with its functionality in the first time.

5 Improve the program of the Gateway

As we have presented in the introduction section, in order to receive data from a button (or a control component on Dashboard), our Gateway have to subscribe to its feed. Now, we have two feeds **"bbc-led"** and **"bcc-pump"**.

To generalize the way we approach and program with many buttons (corresponding with many feeds) in the future, we provide a general code frame with some important notices:

Define an array of feeds that we need to subscribe: AIO_FEED_ID will be modified into an array to store Feeds that we need to subscribe, like the figure below:

```
AIO\_FEED\_ID = ["bbc-led", "bbc-pump"]
```

Code 7.1: Define AIO_FEED_ID as an array

In case that there are more feeds, you just need to list them in this **AIO_FEED_ID** variable.

Modify the connected function: Instead of just subscribing one feed like we did before, one for loop will be used to subscribe all feeds listed in **AIO_FEED_ID** array.

```
AIO_FEED_ID = ["bbc-led", "bbc-pump"]
def  connected(client):
```

```
    print("Ket noi thanh cong...")
    for feed in AIO_FEED_ID:
        client.subscribe(feed)
```

Code 7.2: Define AIO_FEED_ID as an array

When the program is executed, the result will be like this:

```
F:\Teaching\Dariu\IoTGatewayPython\venv\Scripts\python
Connected to Adafruit IO!
Ket noi thanh cong...
Subcribe thanh cong...
Subcribe thanh cong...
```

Figure 7.6: Adding a new button on Dashboard

Depending on how many feeds that you subscribe, there will be as many lines of notifications printed to the screen. Now, when we press two buttons on Dashboard respectively, we will receive 4 different values. The result will look like the figure below:

```
Nhan du lieu: 1
Nhan du lieu: 0
Nhan du lieu: 2
Nhan du lieu: 3
```

Figure 7.7: The result of receiving data at the Gateway

The program of the Gate will be shared at the link below:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab7/IoT_Lab.py

In order to avoid unexpected error, conditional statements are added in case that the Gateway cannot connect to the center Microbit circuit. The source code of the whole program is attached below:

```
import serial.tools.list_ports
import random
import time
import  sys
from  Adafruit_IO import  MQTTClient

AIO_FEED_ID = ["bbc-led", "bbc-pump"]


AIO_USERNAME = "NPNLab_BBC"
AIO_KEY = "aio_radR12aVJMai2YJiGBs1V6TBR06l"

def  connected(client):
```

```
    print("Ket noi thanh cong...")
    for feed in AIO_FEED_ID:
        client.subscribe(feed)

def  subscribe(client , userdata , mid , granted_qos):
    print("Subcribe thanh cong...")

def  disconnected(client):
    print("Ngat ket noi...")
    sys.exit (1)

def  message(client , feed_id , payload):
    print("Nhan du lieu: " + payload)
    if isMicrobitConnected:
        ser.write((str(payload) + "#").encode())

client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()

def getPort():
    ports = serial.tools.list_ports.comports()
    N = len(ports)
    commPort = "None"
    for i in range(0, N):
        port = ports[i]
        strPort = str(port)
        if "USB Serial Device" in strPort:
            splitPort = strPort.split(" ")
            commPort = (splitPort[0])
    return commPort

isMicrobitConnected = False
if getPort() != "None":
    ser = serial.Serial( port=getPort(), baudrate=115200)
    isMicrobitConnected = True


def processData(data):
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    print(splitData)
    if splitData[1] == "TEMP":
        client.publish("bbc-temp", splitData[2])
```

```
63
64 mess = ""
65 def readSerial():
66     bytesToRead = ser.inWaiting()
67     if (bytesToRead > 0):
68         global mess
69         mess = mess + ser.read(bytesToRead).decode("UTF-8")
70         while ("#" in mess) and ("!" in mess):
71             start = mess.find("!")
72             end = mess.find("#")
73             processData(mess[start:end + 1])
74             if (end == len(mess)):
75                 mess = ""
76             else:
77                 mess = mess[end+1:]
78
79 while True:
80     if isMicrobitConnected:
81         readSerial()
82
83     time.sleep(1)
```

Code 7.3: Sample code to handle multiple buttons

6 Review questions

1. In order to receive data from 2 buttons, the Gateway needs to?
   A. Subscribe to 1 feeds of 2 buttons
   B. Subscribe to 2 feeds of 2 buttons
   C. Send data to those 2 buttons
   D. Do not need to do anything, server will automatically send

2. To create two separate buttons, we need to?
   A. Create one shared feed.
   B. Create two separate feeds.
   C. Create 4 different values for buttons.
   D. All answers are correct.

3. To distinguish which button is pressed, we use which fields at the Gateway?
   A. payload
   B. feed_id
   C. Combine two above information
   D. All answers are incorrect,

4. In which cases, we can distinguish data from buttons just using the payload field?
   A. When these buttons return 4 different values
   B. When they have 2 different feeds
   C. When they are on the same Dashboard
   D. All answers are correct

5. Which statement is correct for the center Microbit circuit?
   A. Receive command from Gateway and send wireless data to sensor nodes
   B. Receive wireless data from sensor nodes and send to Gateway
   C. Both above features
   D. Only 1 between 2 above features

6. Which statement is used to convert string to number in Makecode?
   A. parse to number
   B. string to number
   C. parse to string
   D. number to string

7. Which statement is used in **on start** block to send data by Radio statements?
   A. radio send number
   B. radio send string
   C. radio set group
   D. All answers are correct

CHAPTER 8

Peripherals Control on Sensor Nodes

1 Introduction

In this chapter, we implement the last part of an Internet of Things application, the node sensors. We will utilize the Microbit with a variety of built-in features such as temperature and light sensor as well as wireless communication in order to actualize the role of a node sensor.

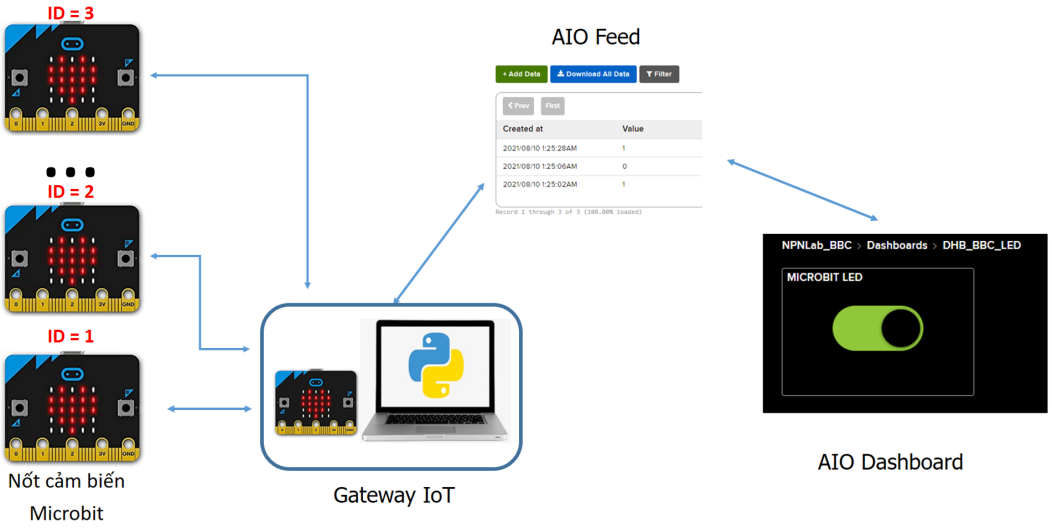

Figure 8.1: A system with multiple node sensors

Working with node sensors means that we are dealing with the hardest part of the entire system. Although sensor node actualization technology is not new, it's problems comes from having many, even overwhelmingly many sensor nodes. They will take care of gathering 1 kind of information individually or as a group. Furthermore, some nodes can be responsible for controlling either 1 or some sort of devices. Therefore, the world of sensors is no doubt diverse.

A crucial thing for sensor chips is to identify the data that it will be forwarding to the Gateway so as to categorize the source of the data and what information is contained in that data (temperature or humidity for example). The problem has been introduced in the previous chapter will be further clarified on this chapter where multiple sensor nodes can be seen interacting with the Gateway.

In this chapter, we will recreate a sensor node receiving orders from a central Gateway. In particular, 4 simple commands will be sent to 1 sensor node to turn ON/OFF a light bulb and a pump. The main requirements of this chapter are as follows:

- Connects electrical devices: electrical relay, pump and LED.
- Receives wireless data from a node sensor.
- Control peripherals at a sensor node.

2 Connect devices to the node sensor

Firstly, for the pump to work we need to use an external electrical source (PIN or adapter for example) since it requires a high amount of voltage. That is why computers' USB port

through a Microbit won't be suffice for a working pump. The relay module is used for controlling the pump since it acts as an electrical switch with 2 lines COM and NO (Common and Normal Open). Normally the relay should be opened with the disconnected COM and NO connection points.

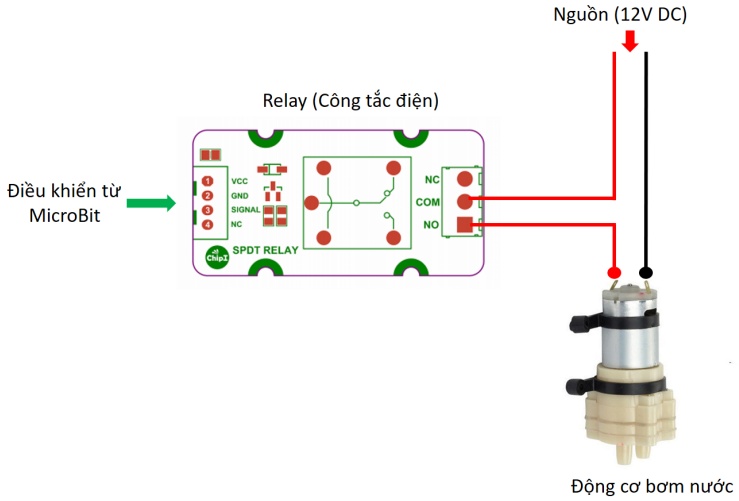

Figure 8.2: Relay and Pump connection

Therefore, in the above connection, we have left out the power cords (anodes) from the battery while the cathodes of the battery is connected to a pole of the device (the pump's engine unidirectional). It wouldn't affect the system functionalities at all if we were to connect the pump with the cathodes left opened. **If you don't have access to a separate battery, you can utilize the 2 power ports on the Chipi Base Shield**. These 2 ports are able to provide voltage to the pump with the external adapter's voltage.

Finally, for the 4 ports controlling the Relay, we connect them to the control ports **VCC-GND-P0-P1** on the expansion board on the ChiPi Base Shield. The LED will be connected to the adjacent ports which are **VCC-GND-P1-P2**. The entire system's connections should be as follows:

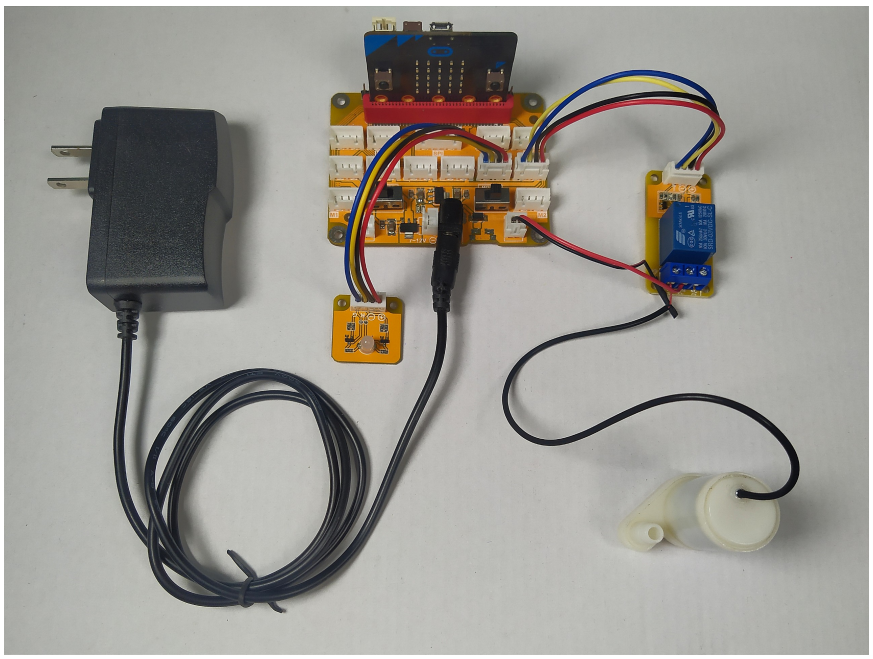

Figure 8.3: A connected electrical system

As such, in order to control the motor, we will program on the port P0 and the LED at port P1 and P2.

3 Implement a sensor node

Firstly, to receive data from Radio, the sensor node needs to be in the same group as the Gateway, in this case is 1. Next, the receiving process is implemented in the **on radio received receivedNumber** block. The data receiving block should be compatible with the way to send data on the previous chapter. Our program's structure should be as follows:

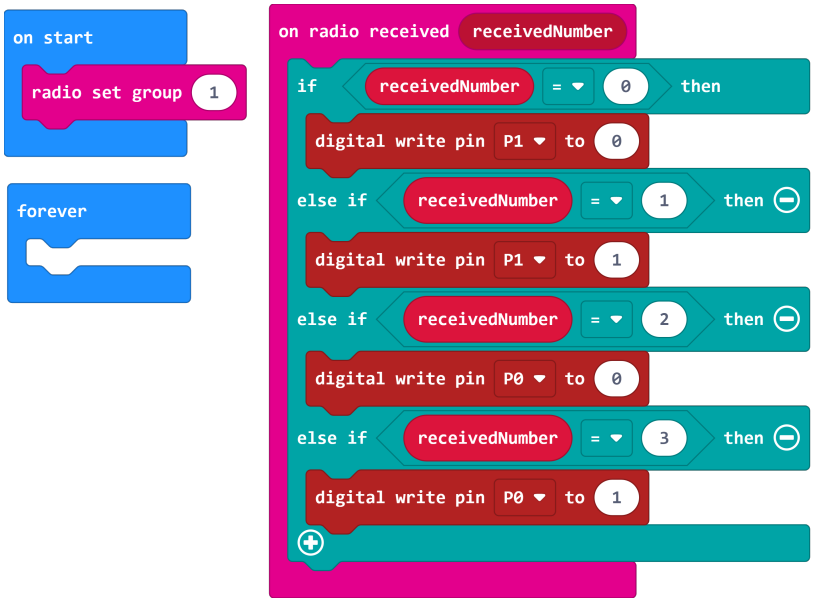

Figure 8.4: LED and pump control block

Programming the ON/OFF effect of a device on Microbit is extremely simple. We just need

to use 1 command which is **digital write pin** with 2 options 0 and 1 (corresponding to OFF and ON). You need to identify the correct port that the desired device is connected to on the Microbit.

If you have 2 Microbit where the first is connected to the LED and the second is connected to the Relay + pump, we only need to reduce the amount of conditions for the **if** statement. An important node is that you are not allowed to use the **else** structure when implementing receiving and executing features from the Gateway. It will cause unnecessary confusions when the system has multiple nodes. You should just implement the correct condition statements.

The above example program can be found in this link:

https://makecode.microbit.org/_0zoE2PhgUigm

In this lesson, we will temporarily not actualizing the functionality in the **forever** loop. In the later lessons, the process of sending the collected data will be implemented in the **forever** loop.

4 Review questions

1. What is the device that can turn ON or OFF a pump called?
   A. Electrical switch
   B. Relay
   C. All are correct

2. Normally when a pump is not working, which 2 ports of the Relay are connected?
   A. COM and NO
   B. COM and NC
   C. NO and NC
   D. All are correct

3. To turn ON or OFF a pump, which of these commands are most suitable?
   A. digital write pin
   B. set pull pin
   C. The combination of the 2 above answers
   D. All are wrong

4. Which of these commands are correct to receive orders from the Gateway Microbit?
   A. seriel on data received
   B. on radio received receivedNumber
   C. on radio recevied receivedString
   D. All are possible

5. To control the status of a pump, which of these commands are suitable?
   A. 1 if else statement
   B. 2 if statements
   C. 1 if else if statement
   D. There are 2 correct answers

6. Which of these sources are relevant for powering the engine of a pump?
   A. 3.3V source from the Microbit
   B. 5V source from the USB port of a computer
   C. A 5V source from an external adapter
   D. All of them can be used

7. How many pins of the Microbit can be used to control the ChiPi LED?
   A. 1
   B. 2
   C. 3
   D. 4

Solution

1. B 2. B 3. C 4. A 5. C 6. A 7. C 1. C 2. A 3. A 3. C 4. B 5. D 6. C 7. B

CHAPTER 9

Integrate Monitoring Sensors

1 Introduction

One of the most powerful applications in the Internet of Things platform is the utility of environmental monitoring. Thanks to the wireless connections, many sensory nodes can simultaneously participate in monitoring the environment's data and broadcasting them back to the central node, which in turn will upload them onto the server, elevating the living standards, boosting the applications both in terms of research and implementation towards services in the future's smart cities.
In this sections, the aforementioned sensory nodes will be used to send the environmental data to the gateway via Radio communication. Therefore, not until now do the sensors act as their vital roles as **inputs** to the system. As a result, generally speaking, the choice of sensors for a specific application should be considered very carefully.

Figure 9.1: *Temperature and Humidity sensor DHT11*

In this section, we will discuss about the integrated temperature and humidity sensor - DHT11 as shown in the above figure. This sensor is able to simultaneously provide 2 kinds of data from the environment, hence the word "*integrated*". DHT11 is a simple sensor for beginners, however its advanced versions like DHT22 or AM2305 are more generally used in practice. Nonetheless, these sensors are highly compatible in terms of software, allowing us to switch for better devices with accuracy and durability at ease.
The main purposes of this chapter are as follows:

- Connection with DHT11 sensor.
- Programming for data retrieval from DHT11.
- Improving the Gateway.
- Upgraded versions of DHT11.

2 Connect to DHT11 sensor.

In order to connect with DHT11, we only need 3 pins, including VCC (ranging from 3V to 5V), GND (0V) and signal pin for programming. It is obvious that, with a wide voltage range, DHT11 is highly compatible with many general programmable hardware devices, particularly the Microbit board. There are various online documents regarding the project using Microbit and DHT11 sensor with connection instruction, as illustrated below:

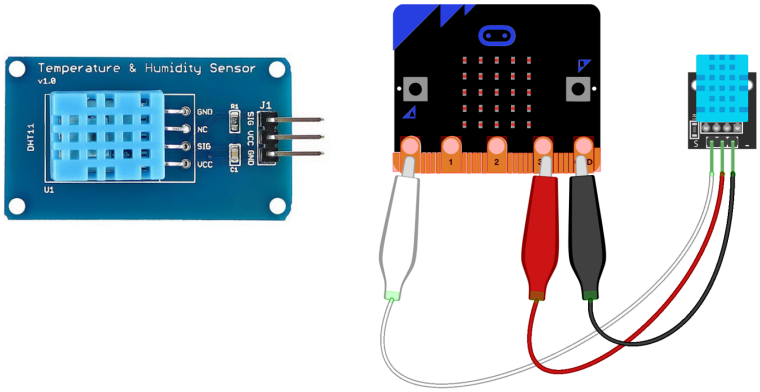

Figure 9.2: Connection with DHT11 with 3 pins

In this instruction, we will use a standardized DHT11 for connection compatibility with the ChiPi Base Shield extension board. We simply connect a 4-pin wire with the extension board. On connection, you should notice which pin of the Microbit board is connected with the signal pin of DHT11. This information will be used in the next programming phase. Following the previous sections, we will connect the sensor to pin **P3** as in the below figure:

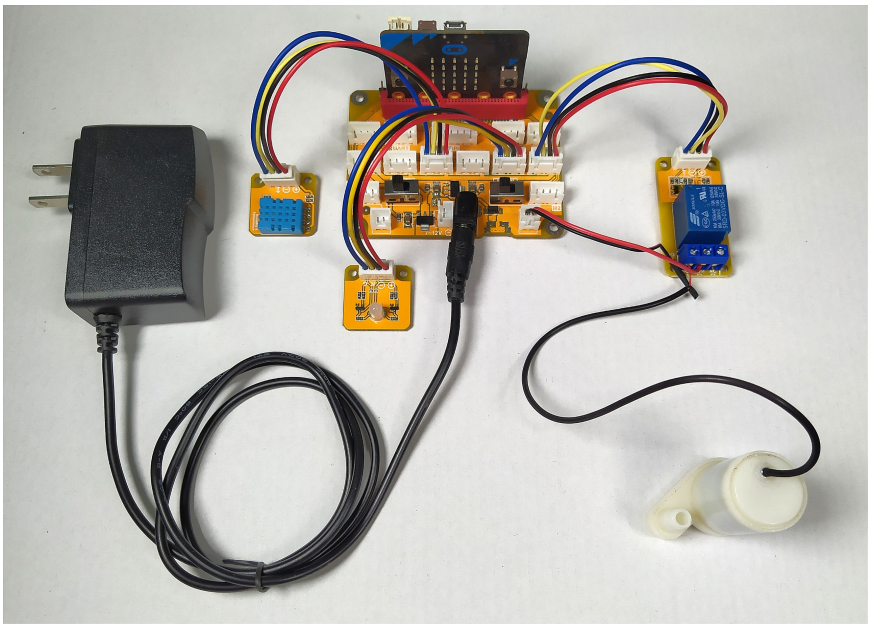

Figure 9.3: Adding DHT11 to the sensor collection

3 DHT11 working mechanism

DHT11 is a simple integrated sensor and widely used in applications with micro-controllers in general and Microbit board in particular. In order to read the data from DHT11, the Microbit board must send a **query** signal. On receiving the signal, the sensor will start calculating and then return the data to the Microbit board. For this sole reason, if the Microbit board "forgets" to send the query signal, the received data may be outdated and,

therefore, invalid.

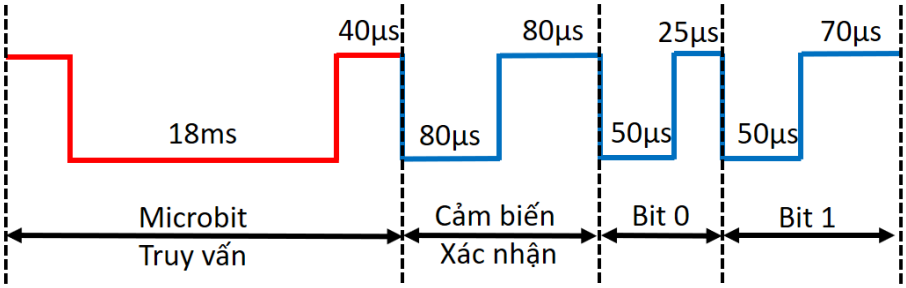

Figure 9.4: Communication principle for DHT11

Despite being one of the simplest integrated sensor, it is rather complicated when in comes to data retrieval from DHT11. In order to initiate the communication protocol, a query signal from Microbit board to the sensor is a low signal, which lasts 18ms. Afterwards, Microbit board will raise the signal to high level, waiting for response from DHT11. After about 40μs, the sensor will acknowledge by sending back an 80-μs low signal, followed by an 80-μs high signal. Finally, data of 40 bits will be sent back to the Microbit, with bit 0 being comprised of a 50-μs low signal followed by a 25-μs high singal, while bit 1 being comprised of a 50-μs low signal followed by a 70-μs high signal. Readers should pay attention to the unit of time, illustrated in figure 9.4.
After the decoding of the 40-bit data, Microbit board will continue processing to extract the necessary information for the application, with the first 16 bits representing the air humidity, followed by 16 bits for the temperature and ending with an 8-bit error detection signal (40 = 16 + 16 + 8).
Due to the complication characteristic, mainly from the processing of the signal, most application written for Microbit will use supporting programming libraries. Thanks to these libraries, the programming will be a lot more simple and convenient for the readers to focus more on building the application itself, rather than the deep interaction to the system regarding tasks for signal processing.

4 Program with DHT11

In order to support for this section's programming phase, and also for other sections in this textbook, **IoTBitKit** library will be added. The sequence to add this library is as follow: from the main programming screen, click **Advanced**, then **Extensions**, as illustrated below: In the popup screen, we will type in the following link:

https://github.com/npnlab-vn/IoTBitKit

Finally, press the search button. The result will be displayed as below:
Many groups of commands for IoTBitKit library will be included into the system. For this instruction, all the necessary commands for DHT11 programming belong to the command group NPNBitKit, as shown below:

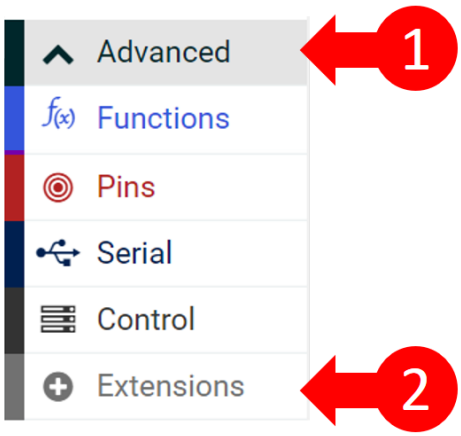

Figure 9.5: Adding libraries in Makecode

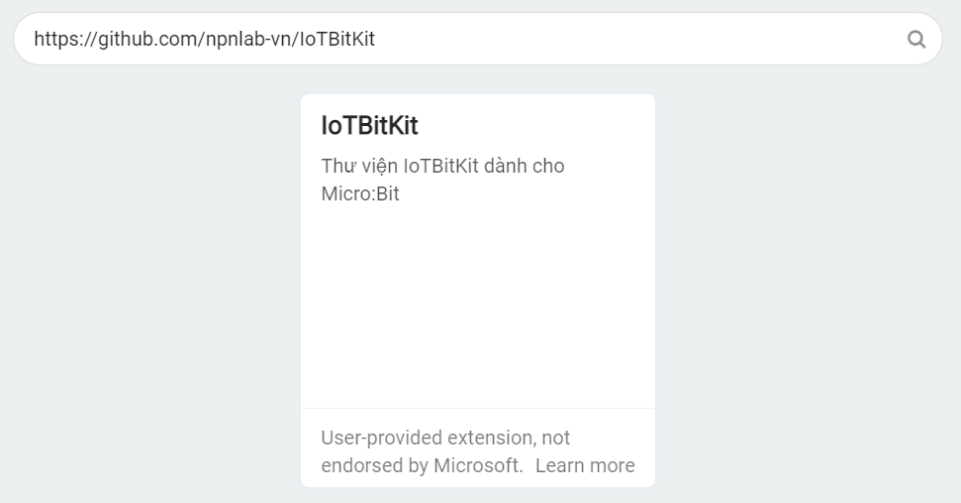

Figure 9.6: Adding IoTBitKit library

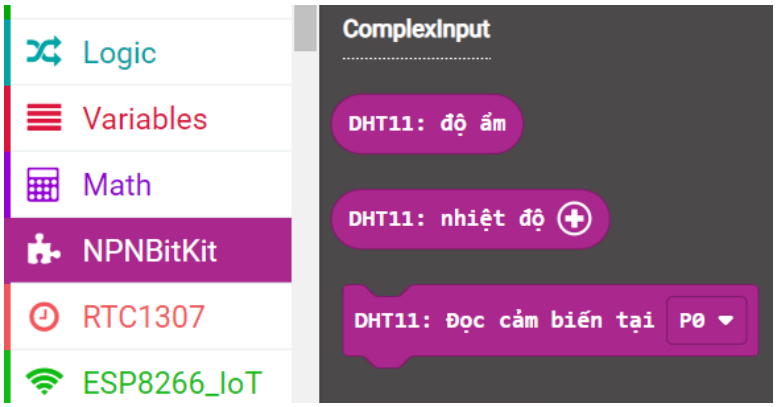

Figure 9.7: Commands related to DHT11 in NPNBitKit

Note that in order to retrieve updated values from the sensor, we must query it with command **DHT1: read sensor information at P3**. In the case that the sensor is connected to another pin, readers must change the connection pin accordingly. After the query, data from the sensor can be accessed via **DHT11: temperature** and **DHT11: Moisture**.

Reusing the developed program in previous sections, we will add a feature to read the data from the sensor every 30 seconds in the **forever** loop. This is the suggested program:

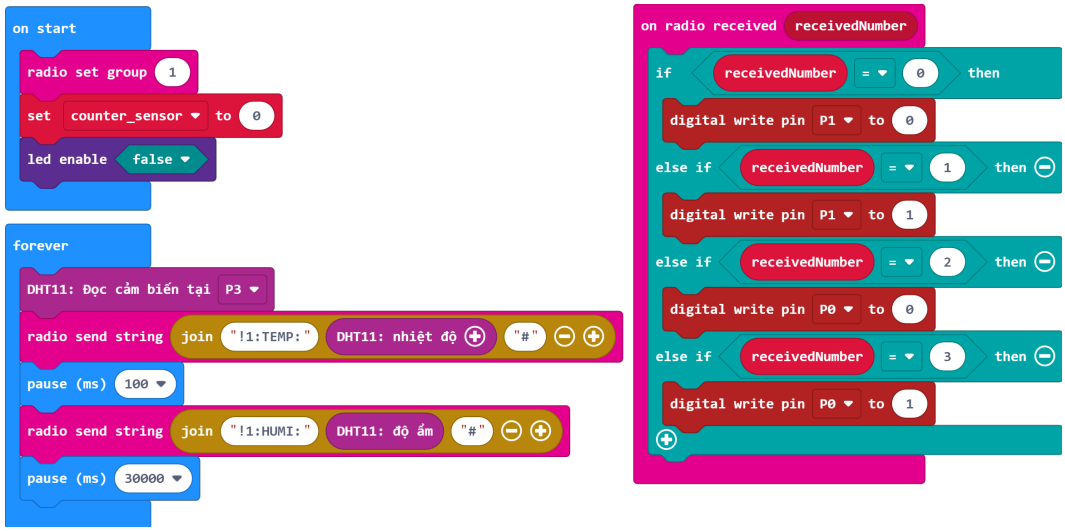

Figure 9.8: The program to read data from DHT11

The sensory data will be sent to the central node in the predefined format. Two pieces of information are sent a small time apart, reducing the probability of data collision. Furthermore, in order to simplify the programming, we add a 30-second wait command at the end of the **forever** loop. Readers may use their initiative by getting rid of this command, using only the wait command **pause(100)**. For the second implementation, the system will work a lot better.
Readers should notice that due to the connection to pin P3, we must add a command **led enable false** in the **on start** block. With this command, all 25 LEDs of the Microbit will be disabled in return for the usage of extra pins, such as P3. The program is shared in this link:

https://makecode.microbit.org/_Y9ufwoAFAiKp

5 Improve Gateway

With the said 2-piece information, readers may add another feed and plot the humidity data. Improvements for the Gateway program can be made by adding code to the **processData** function, as shown below:

```
def processData(data):
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    print(splitData)
    try:
```

```
        if splitData[1] == "TEMP":
            client.publish("bbc-temp", splitData[2])
        elif splitData[1] == "HUMI":
            client.publish("bbc-humi", splitData[2])
    except:
        pass
```

Code 9.1: Improved function for the data processing

When various data are sent to the central node, errors may occur in the reception of data. Therefore, the command **try except** is added in order to skip invalid data. This is very useful for practical applications, where we cannot anticipate all the errors that may occur when the system is functioning. The Gateway program is shared via this link:

https://github.com/npnlab-vn/code-manager/blob/IoT_Lab9/IoT_Lab.py

6 Upgraded versions of DHT11

One of the main advantages of DHT11 is its replacement ability, allowing it to be replaced by more advanced sensors with higher accuracy that meets the industrial standards, for example in fruit dehydrators or egg incubators, etc.
In this last section, two more advanced versions of DHT11 will be presented. These sensors have the same communication protocol. We can simply swap the hardware without the need of re-programming.

6.1 DHT22 sensor

This sensor resembles DHT11 in terms of appearance, but it is usually white for differentiation. Its price is commonly double to that of DHT11. Therefore, its accuracy is undeniably higher than DTH11's, with only 2% error for humidity data, while it's 5% for DTH11. On the other hand, temperature readings from DHT22 only yield at most 0.5°C, compared with the error of 2°C of DHT11's.

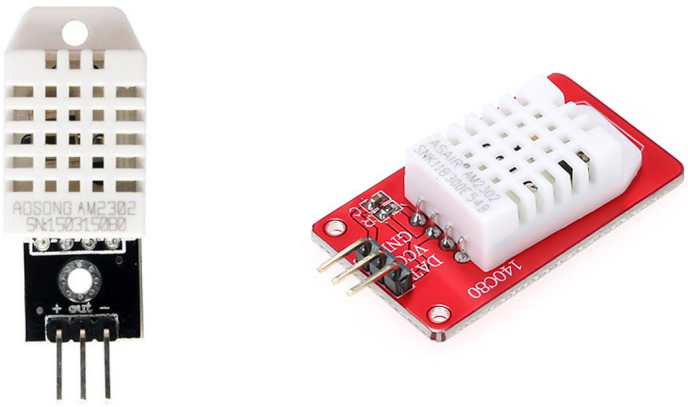

Figure 9.9: DHT22 sensor on the market

As a result, for applications where temperature measurement is critical, for example an egg incubator or a large-scale dehydrator, the usage of DHT22 will yield a much better result than DHT11. With complete pin compatibility, the replacement of DHT11 with DHT22 is extremely easy and no modification on the program or software is required.

6.2 AM2305 sensor

This temperature - humidity sensor comes in a firmer package and higher durability than DHT22. Normally, devices used for industrial purposes require high durability and high temperature resistance. The image of this product is presented below.

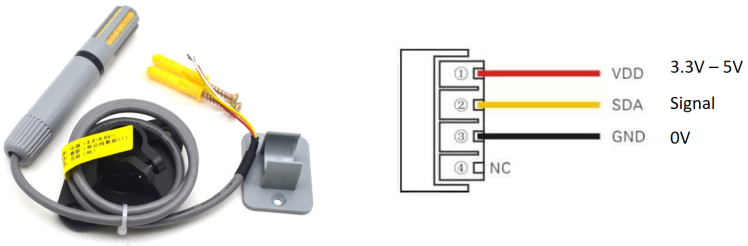

Figure 9.10: AM2305 sensor with 3 connection pins

When using this product, readers must be aware of its number of pins. The upgraded version of AM2305 is AM2315, which has 4 pins in stead of only 3 like AM2305. With its VCC pin being compatible with wide voltage range (from 3.3V to 5V), AM2305 is suitable with Microbit board. However, in large applications, 5V power source should be used for the sensors in order to guarantee the stability. The accuracy for AM2305 is significantly better than that of DHT11, with errors being only 0.1°C for temperature and 1% for humidity.

7 Review questions

1. Which of the following statement is correct about DHT11?
   A. Integrated sensor.
   B. Able to measure both the temperature and the humidity.
   C. Is an input device.
   D. All of the above.

2. Before accessing values from DHT11, what should be done first?
   A. Pulling the signal pin to high level.
   B. Pulling the signal pin to low level.
   C. Querying the sensor.
   D. All of the above.

3. What is the name of the command group used for DHT11 and LCD programming that was introduced in this section?
   A. DHT11.
   B. LCD.
   C. NPNBitKit.
   D. All of the above.

4. How many pins are needed to connect DHT11 with the Microbit extension board?
   A. 1.
   B. 2.
   C. 3.
   D. 4.

5. What is the connection voltage that is supported by DHT11?
   A. 3.3V.
   B. 4.1V.
   C. 5V.
   D. All of the above.

6. Which of the following sensors are compatible with DHT11's connection pins?
   A. DHT22.
   B. AM2305.
   C. AM2315.
   D. A and B.

7. Which of the following sensors has the highest accuracy?
   A. DHT11.
   B. DHT22.
   C. AM2305.
   D. Not determinable.

Solution

1. D 2. C 3. C 4. C 5. D 6. D 7. C

CHAPTER 10

Overview on Analog Sensors

1 Introduction

In this chapter, we will introduce about Analog sensor, its function is the same as the integrated we have covered in the previous chapter. However, the sensor's output is much simpler, which is called equivalent output, you can directly read the sensor's data without any external library. Reading equivalent output from this sensor is nothing but reading ADC output (Analog to Digital Converter). When working with Microbit, the signal is converted from analog to digital ranges from 0 to 1023.//

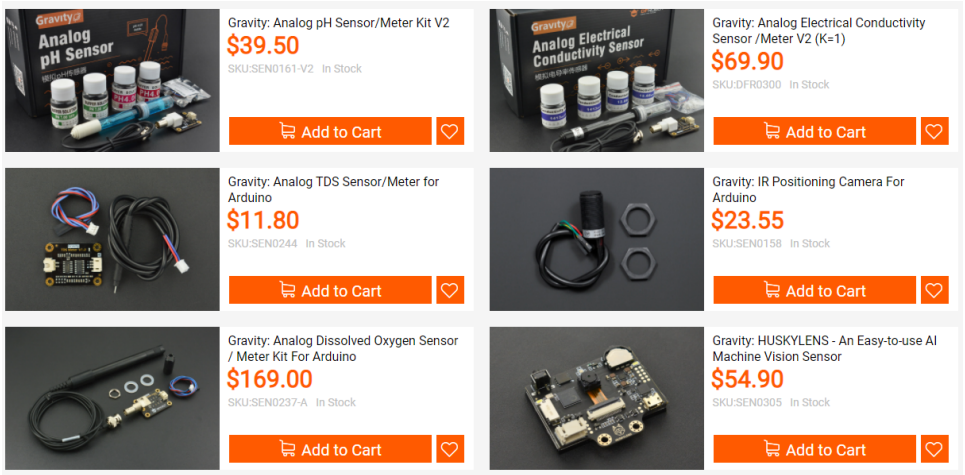

Figure 10.1: ADC sensors in DFRobot

Due to the fact that operating this sensor is simple, projects which specifically related to educational purple generally support the ADC output. Even the sensor with complex components still supports the output with ADC standards, e.g. Analog Gravity sensor in DFRobot, supports up to a hundred sensors with ADC output. One of the most important of this kind of sensor is to supply a suitable voltage, which is should neither too high nor too low as voltage is a critical factor to keep the sensor working normally
In this tutorial, we will talk about the how ADC sensor works and using the gas sensor for illustration. We will also introduce the Chipi sensors which is a simple sensor system with the standard connection designed for educational purposes, targets secondary students **implementation will remain for the readers to design with your creativeness**. This chapter aims to:

- The principle of ADC sensors.
- Setup the gas sensor.
- Write a program to read the sensor.
- Further knowing of ADC sensors in Chipi system.

2 Principle of Analog sensors

The connection between the Analog sensor with the peripheral is apparently operated due to the prearrangement in the circuit. Additionally, writing the program to read the light sensors is much more convenient. These kinds of the sensor depend on how tracking condition changes, it will change correspondingly. For instance, a sensor to measure

how much temperature an electronic component emits will depend on how much temperature affects the sensor's resistor. This component is so-called a **thermistor**. Therefore, a light sensor utilizes a **photoresistor**, this resistor will change corresponding to the intensity of the environment light. The core of sensors is illustrated below:

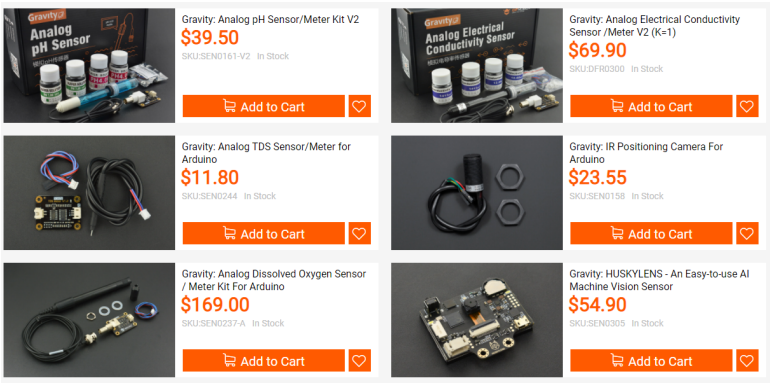

Figure 10.2: The core of ADC sensors

Firstly, take a look at the resistor bridge on the left side, which includes resistor R1 and resistor R2, wired from the source to the ground. The resistor R1 represents the sensor and its changes depend on the environment while the R2 is a constant. Apply the Ohm's law, the voltage at SIG1 will be calculated as:

$$SIG1 = \frac{VCC * R2}{R1 + R2}$$

Apparently, when R1 changes, SIG1 will change too. Now recall the soil moisture sensor in the previous chapter, when the soil dries the more resistant R1 is. As the R1 is under the denominator, the less voltage at SIG1 will be. In reverse, when soil is humid, less R1 will result in the high voltage at SIG1. This behavior is reversible when we change the position of R1 and R2 in this circuit.

Most of the devices in the ChiPi system are designed by following this rule. Therefore, the output of a sensor essentially includes 3 pins: VCC, GND, and SIG. However, in the ChiPi system, there is an amplifier OPAM integrated on the right side to keep the signal at SIG2 more stable. This circuit is called a feedback amplifier as the output is feedback wired to the OPAM pin. This wiring is to ensure the voltage at SIG2 and SIG1 is equal, however, with the resistant capability of the OPAM, there is no current directly apply to Microbit. Therefore, Microbit is more durable and does not overheat during operation. When there is no OPAM, there is a small current from SIG1 apply to Microbin pins, hence the more voltage SIG1 is (when the sensor hails or signal interferences) will, unfortunately, damaged the Microbit

About the sensor which outputs a voltage, the programming in Microbit easily gets with the function **analog read**. The return of this function is a number that ranges from 0 to 1023 (or ADC 10 bit). The result has no unit name. Therefore, to convert it to unit **%** of the soil sensor, or **lux** of the intensity, even the **ppm** of the CO2 concentration, we need to construct another conversion function. This construction will cost too much as we need a stable product to map the voltage value to the desired value. In the book, we make it as simple as possible so that we just differentiate between when it is over a defined threshold or not, we do not deeply investigate the conversion function.

3 Introduction to gas detection sensor

In this section, the gas sensor is named MQ2, it's in the MQx family, a simple gas detection technology. This sensor works as followed by the electrochemical rule. Therefore, it will need a current with high power to make it possible to separate the desired gas detection and return the correct value. For this reason, MQ2 needs an external power of at least 5V-1A, which may exert a little heat when operating.

In the sensor market, there are gas sensors that work on the electrochemical rule, here is the list of common sensors:

- MQ3: Alcohol, Ethanol, smoking
- MQ4: Methane gass
- MQ7: Carbon Monoxide CO
- MQ8: hydro gas
- MQ9: CO gas and another dangered gas

Hence, in this tutorial, we can extend to another application by replacing the MQ2 sensor to another sensor in MQx family. However, most importantly, the sensor should be supplied with a voltage of 5V, the reader must follow this rule when work with Microbit and its based shield. We will use ChiPi Based Shield V2 in this section with the support of 2 ports of 5V. To supply voltage for these ports, we need a power supply called an adapter. Then, MQ2 sensor will be attached to the based shield via a 4-pin wire, as shown in the figure below:

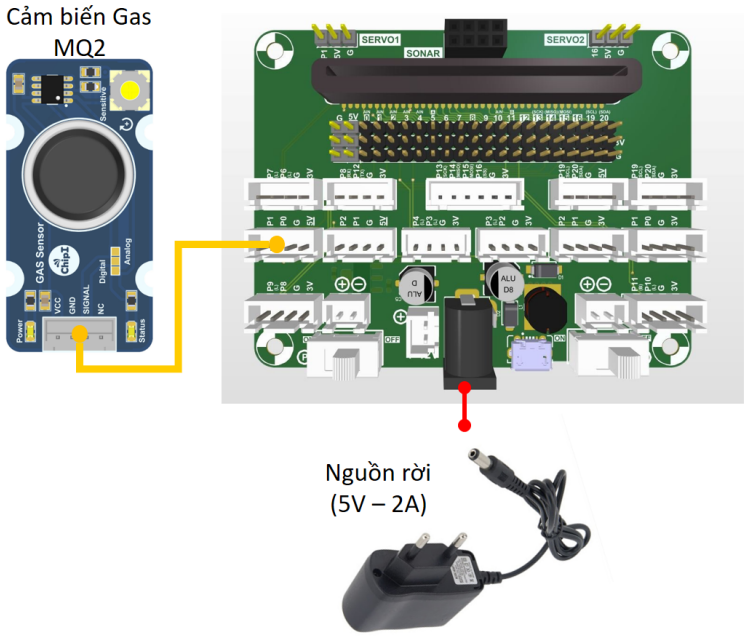

Figure 10.3: Gas MQ2 attached to the Chipi Based Sheld V2

Take a look at this arrangement, the gas sensor is attached to pin P0 of Microbit. To find this pin position, readers should take care of the yellow wire which is connected to the SIGNAL pin of the sensor and the ChiPi Base Shield V2.

4 Read gas detection sensor

It is really simple to get raw data from the ADC sensor, using **analog read pin** statement in **Pins**. Here is the sample code:

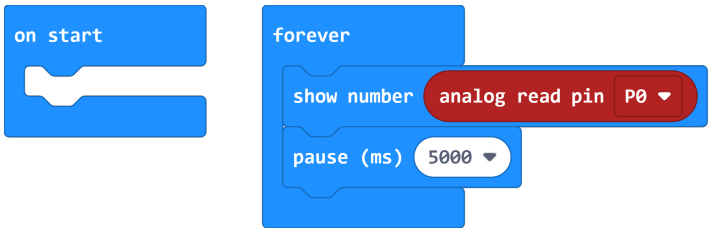

Figure 10.4: Read raw data from ADC sensor

With these simple applications, the above program is run to record the value from the sensor. Readers need to choose your threshold for your comparison to intentionally give a warning when it reaches over your threshold. Take an example with the gas sensor, when we carefully chose a threshold to warn the user if the gas is above an allowed level. You can further send the data to the ThingSpeak server and activate the warning email as we had covered in the previous chapters.

The output value from the ADC ranges from 0 to 1023 when working with the Microbit. On the other hand, an ADC converter on Microbit represents 10 bits. Therefore, with the voltage input from the sensor, we choose it to range from 0V to 3.3V to result in a linear mapping in the 10-bit domain value, from 0 to 1023 ($2^{10} - 1$). In case we need to calculate

the actual value from the raw data, we construct a conversion function for each sensor. For instance, **the value of the gas sensor, calculated in ppm, is twice as the voltage calculated in miliVolt**. With this definition, we need to convert the raw data to mV by using the block **map** in the category **Math**. Here is an example program for this conversion:

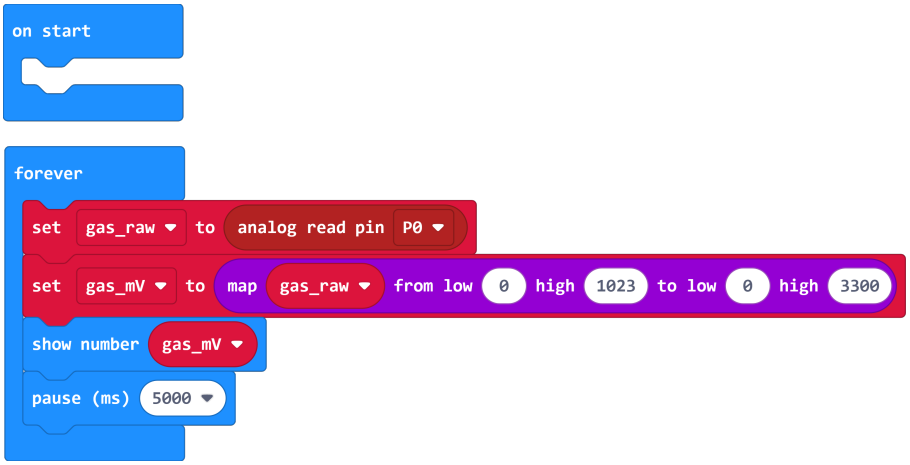

Figure 10.5: Convert the raw data to mV

In the example program, two variables are declared to temporarily store the calculating value, includes **gas_raw** to read the raw data and **gas_mV** to store the result value when converted to mV. Because the value range of **gas_raw** is from 0 to 1023, the corresponding value range of voltage is from 0 to 3300 mV (3.3V), hence the parameter of the block is scaled as in the example program.

When the raw data is converted to the voltage, readers should use one more conversion function to finalize the desired unit. This second conversion depends on which sensor is used, so we don't mention it in detail. When applying this second conversion, readers will use the calculation function from Math category, we also don't mention it in detail for this section.

5 Analog ChiPi sensors

For the best convenient purpose, specifically with the readers learning about sensors from the beginning, a pretty much understanding of the electronic principle is required, but the sensors from the ChiPi system are designed to be at most clear and easy to operate. All the ports and connectors from the ChiPi cannot be plug in the other direction, so we just need to connect at the right port and get the right value, readers should not worry about the wrong side when plugging. Here is an example about 2 other sensors, with ADC output, to sense and light intensity and obstacle detection:

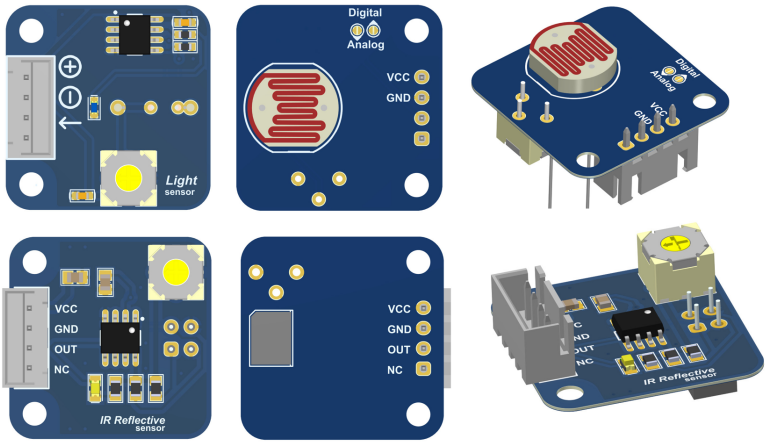

Figure 10.6: Some ADS sensors in standard Chipi

For the infrared obstacle sensor, when an obstacle is nearby, the voltage increases (due to the reflecting light from the sensor). In reverse, when an obstacle is placed further or there is no obstacle, the voltage decreases. Moreover, the soil sensor or the sound sensor also works based on this rule. There is some precaution when connecting with the Microbit:

- Analog sensors are supported at pins: P0, P1, P2, P3, P4, and P10. Therefore, we just connect at most 6 analog sensors in a single Microbit. If we want to connect more analog sensors, we can use another Microbit and then using the Radio to communicate between them
- When connecting via P3, P4, and P10, block **on start** will need to use the statement **led enable false** to initialize the ADC control of these pins. As the result, we cannot use the LED display of the Microbit
- We should choose the threshold carefully. When you cannot use the 25 LEDs in Microbit (as statement **led enable false**), you can send the data to your computer or using Data Streamer add-on of Excel.

When you work with a worthier sensor, such as a water quality sensor or air quality sensor in the sensor market with an ADC output, you can integrate it into the Microbit. Therefore, readers should take care of supply a suitable voltage prior to the specification of your sensor.

With the implementation of sensors and sending the data to the center node, we will leave it for the readers for further investigation. Readers should define a keyword when sending the data of the sensor to the center node and processed it in a **processData** function of the Gateway. Furthermore, not only one node but you can use multiple nodes and send the data concurrently to the center node. In this case, readers should define an identifier called **ID** for each sensor node so that the center node can differentiate between all sensor nodes

6 Review questions

1. Which type of signal output of the CO2 sensor?
   A. digital
   B. analog
   C. uart
   D. All of the above

2. The voltage output of analog sensor works as the changes of
   A. Voltage bridge with 2 resistors
   B. Potentiometer
   C. Constant resistor
   D. All of the above

3. Which components is used when we design a seperated analog sensor?
   A. Resistor
   B. Potentiometer
   C. OPAM
   D. All of the above

4. In Microbit, which is the maximum number of analog pins supported?
   A. 1
   B. 2
   C. 6
   D. All of the above

5. Which is the conversion function supported in the Makecode to convert the raw data to desired voltage?
   A. Math
   B. map
   C. linear
   D. None of the above

6. When connect the sensor to pin P3, which statement is used in block on start?
   A. led enable true
   B. led enable false
   C. led turn on
   D. led turn off

7. When we turn off the LEDs control in Microbit, which statement will have no effect?
   A. show number
   B. show icon
   C. show string
   D. All of the above

Solution

1. B 2. A 3. C 4. B 5. B 6. B 7. D