

4ID3 - IoT Devices and Networks

Lab 5

Communicating Sensor Data over a ZigBee Network

Adam Sokacz, Ishwar Singh, and Salman Bawa

Sponsored by *Future Skills Center, Canada* and *McMaster W Booth SEPT*

Group Number:

Name, Student Number:

Name, Student Number:

Name, Student Number:

Submission Date:



Objective

In this lab, sensor data will be collected on a node in the ZigBee network. This data will be passed on to the controller node while the devices are in API mode. Once at the controller, it will be communicated to the PC using wired serial communication, and aggregated into a database.

Contents

Objective	2
Feedback	3
Additional Resources	3
Pre-Lab Questions	4
Post-Lab Questions	4
Exercise A Results:	6
Exercise B Results:.....	6
Exercise C Results:.....	6
Project Setup.....	7
Installing XCTU	8
Connecting and Updating the XBee3 Modules	10
Configuring Boards using XCTU.....	14
Transparent Mode	16
API Mode.....	20
GPIO Forwarding	27
Connections	27
Python script setup	29
Exercise A	35
Exercise B	35
(Optional) Exercise C.....	36
Syncing with GitHub.....	38

Feedback

Q1 - What would you rate the difficulty of this lab?

(1 = easy, 5 = difficult)

1

2

3

4

5

Comments about the difficulty of the lab:

Q2 - Did you have enough time to complete the lab within the designated lab time?

YES

NO

Q3 - How easy were the lab instructions to understand?

(1 = easy, 5 = unclear)

1

2

3

4

5

List any unclear steps:

Q4 - Could you see yourself using the skills learned in this lab to tackle future engineering challenges?

(1 = no, 5 = yes)

1

2

3

4

5

Additional Resources

Lab GitHub Repo (<https://github.com/sokacza/4ID3>)

MySQL Basics (<https://youtu.be/Cz3WcZLRaWc>)

NodeRED Fundamentals Tutorial (<https://youtu.be/3AR432bguOY>)

Pre-Lab Questions

Q1 - Name **6** advantages of ZigBee over other **access technologies**.

(Suggested: List)

Q2 – ZigBee devices operate as a **mesh network**. Draw the mesh network topology using **4 devices**.

(Suggested: Sketch)

Q3 - ZigBee builds on the IEEE standard **802.15.4** physical layer. What layer of the **ISO model** does ZigBee operate in?

(Suggested: Sentence)

Q4 – Since ZigBee networks ping data across many devices, what **security features** are built into ZigBee to prevent data from being exposed with malicious intent?

(Suggested: List)

Q4 – In documentation, you may hear the terms **XBee** and **ZigBee** being used **interchangeably**. What is the **difference** between these two terms?

(Suggested: A few sentences)

Post-Lab Questions

Q1 - Draw a **diagram** to identify **each component** of the IoT network produced in this lab and describe the **information being exchanged** between the components.

(Suggested: Sketch)

Q2 – There are **3** types of XBee devices: **Coordinators**, **Routers**, and **End Devices**. In a table, compare and contrast the role of each category of device in the ZigBee network.

<https://www.digi.com/resources/documentation/Digidocs/90001942-13/>

(Suggested: Table)

Q3 – **AT commands** are typically used when communicating with low-level **embedded** devices such as **communication modules**, especially for **device configuration**. Using page **40** in the **product manual**, describe how the AT commands for this product are **structured**. Relate AT commands with what happens when you press **write setting** in the **XCTU** XBee software?

<https://www.digi.com/resources/documentation/Digidocs/90001942-13/>

(Suggested: Paragraph)

Q4 – There are **two operating modes** for these XBee3 units: **API** and **Transparent**. Using the manual, research each of these modes and compare and contrast **3** points of each in a **table**.

<https://www.digi.com/resources/documentation/Digidocs/90001942-13/>

(Suggested: Table)

Lab 5 - Communicating Sensor Data over a ZigBee Network

Q5 – The END_DEVICE publishes the message **GPIO4 ON** to the network address **00 00 00 00 00 00 00**. The ROUTER device is just within range of the END_DEVICE, and the COORDINATOR is just within range of the ROUTER. The END_DEVICE is out of range of the COORDINATOR. Which device(s) receives the message? Explain why?

(Suggested: Paragraph)

Q6 - You are tasked with making a desktop application that has a user log in. The username and password are stored in a MySQL database. Investigate the topic of password hashing:

<https://youtu.be/cczlpiiu42M>

Write and execute a **Python script** that hashes the following password:

Password: 9055259140

<https://www.geeksforgeeks.org/how-to-hash-passwords-in-python/>

Paste a screenshot of your terminal output:

Write a MySQL query to insert the hashed password into the following database:

Accounts	
Email	Password
test@example.com	<Your password hash>

Q7 - What is the difference between **hashing** and **encryption**? Write and execute a **python script** to **encrypt** and **decrypt** the following string:

message = “ { \“GroupA\”: { \“DeviceA\”: { \“Temperature\”: 23.9 } } } ”

<https://www.geeksforgeeks.org/how-to-encrypt-and-decrypt-strings-in-python/>

Paste a screenshot of your terminal output:

Q8 - Write a brief LinkedIn post about **5 key learning takeaways** from this lab.

(Suggested: Short paragraph)

Exercise A Results:

Exercise B Results:

Exercise C Results:

Project Setup

Issue the following commands from the root directory of your existing local repository for this class:

```
git pull origin main
```

<Create your folders and README.md file>

```
git add .
```

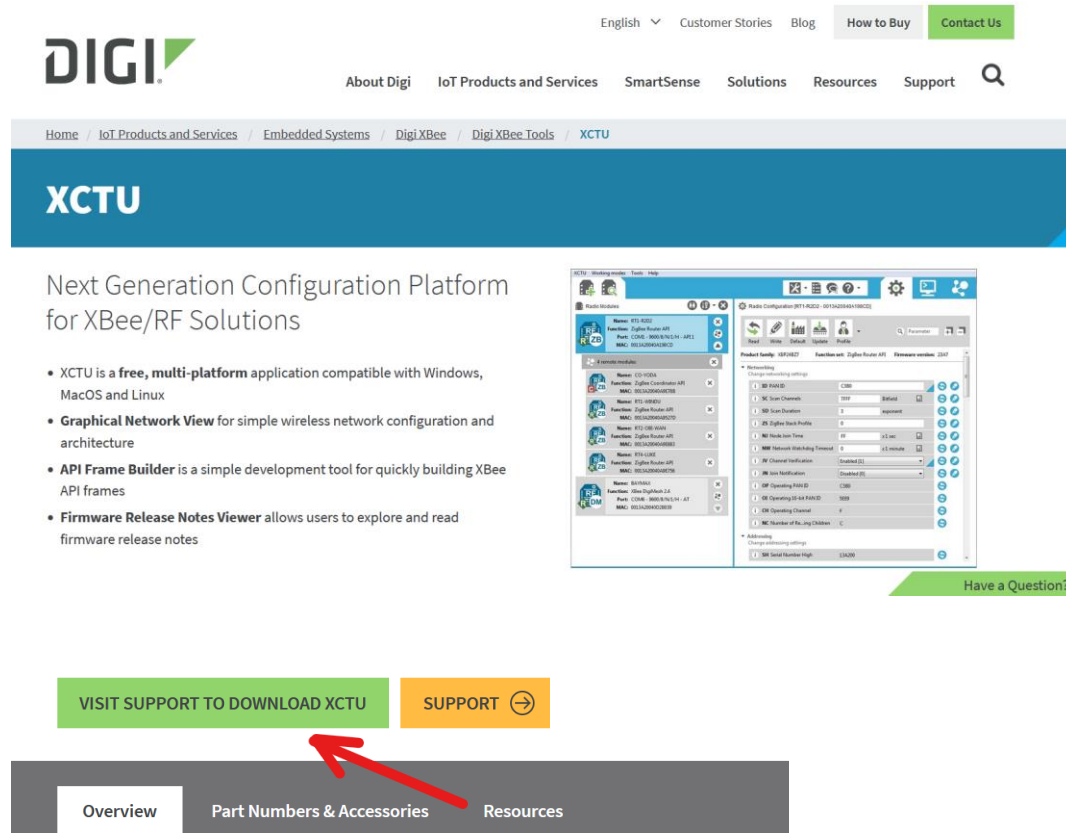
```
git commit -m "lab 6"
```

```
git push origin main
```

Installing XCTU

Navigate to the following URL and install the XCTU XBee3 configuration utility:

<https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu#productsupport-utilities>



English Customer Stories Blog How to Buy Contact Us

About Digi IoT Products and Services SmartSense Solutions Resources Support

Home / IoT Products and Services / Embedded Systems / Digi XBee / Digi XBee Tools / XCTU

XCTU

Next Generation Configuration Platform for XBee/RF Solutions

- XCTU is a **free, multi-platform** application compatible with Windows, MacOS and Linux
- **Graphical Network View** for simple wireless network configuration and architecture
- **API Frame Builder** is a simple development tool for quickly building XBee API frames
- **Firmware Release Notes Viewer** allows users to explore and read firmware release notes

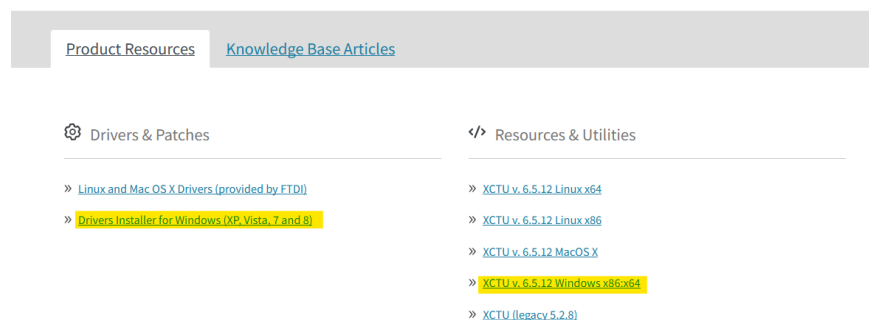
Have a Question?

VISIT SUPPORT TO DOWNLOAD XCTU SUPPORT

Overview Part Numbers & Accessories Resources

What is XCTU?

Install both the utility and drivers for your PC. Select **Add Drivers to Windows** when prompted.



Product Resources Knowledge Base Articles

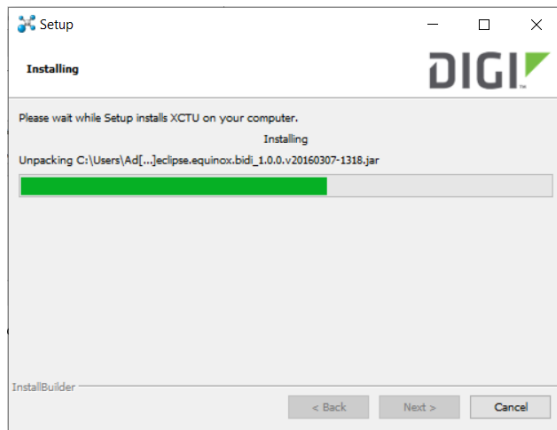
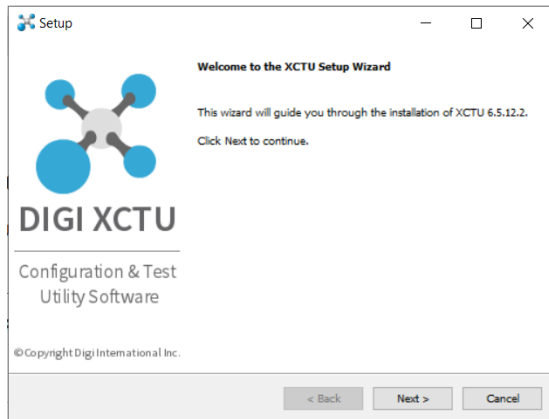
Drivers & Patches

- » [Linux and Mac OS X Drivers \(provided by FTDI\)](#)
- » [Drivers Installer for Windows \(XP, Vista, 7 and 8\)](#)

Resources & Utilities

- » [XCTU v. 6.5.12 Linux x64](#)
- » [XCTU v. 6.5.12 Linux x86](#)
- » [XCTU v. 6.5.12 MacOS X](#)
- » [XCTU v. 6.5.12 Windows x86/x64](#)
- » [XCTU \(legacy 5.2.8\)](#)

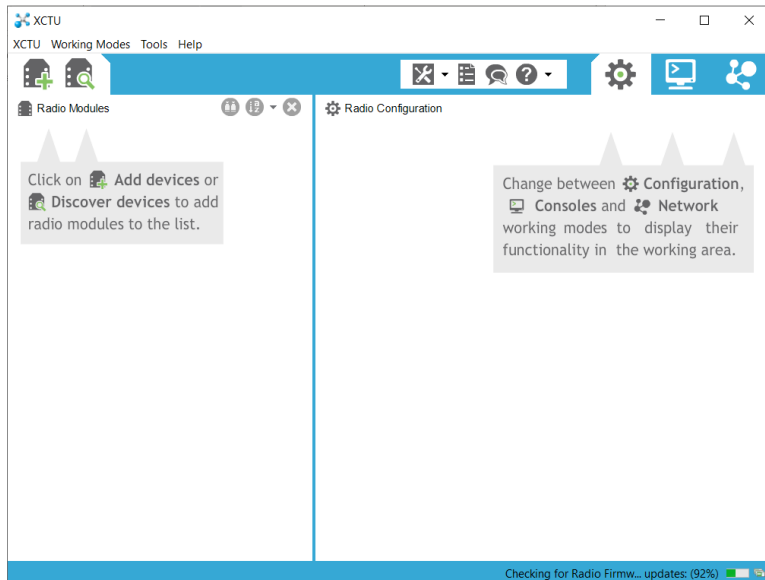
Lab 5 - Communicating Sensor Data over a ZigBee Network



Once installed, **restart** your PC.

Connecting and Updating the XBee3 Modules

Open the XCTU XBee3 configuration utility.

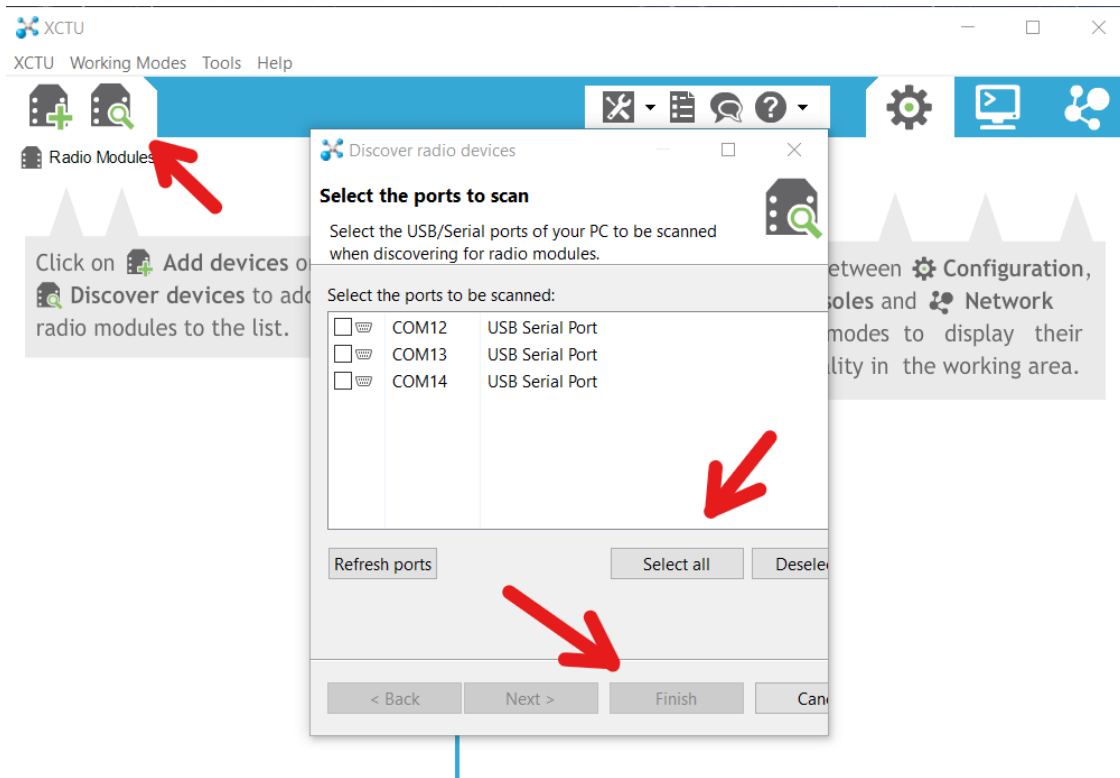


Plug in your XBee devices

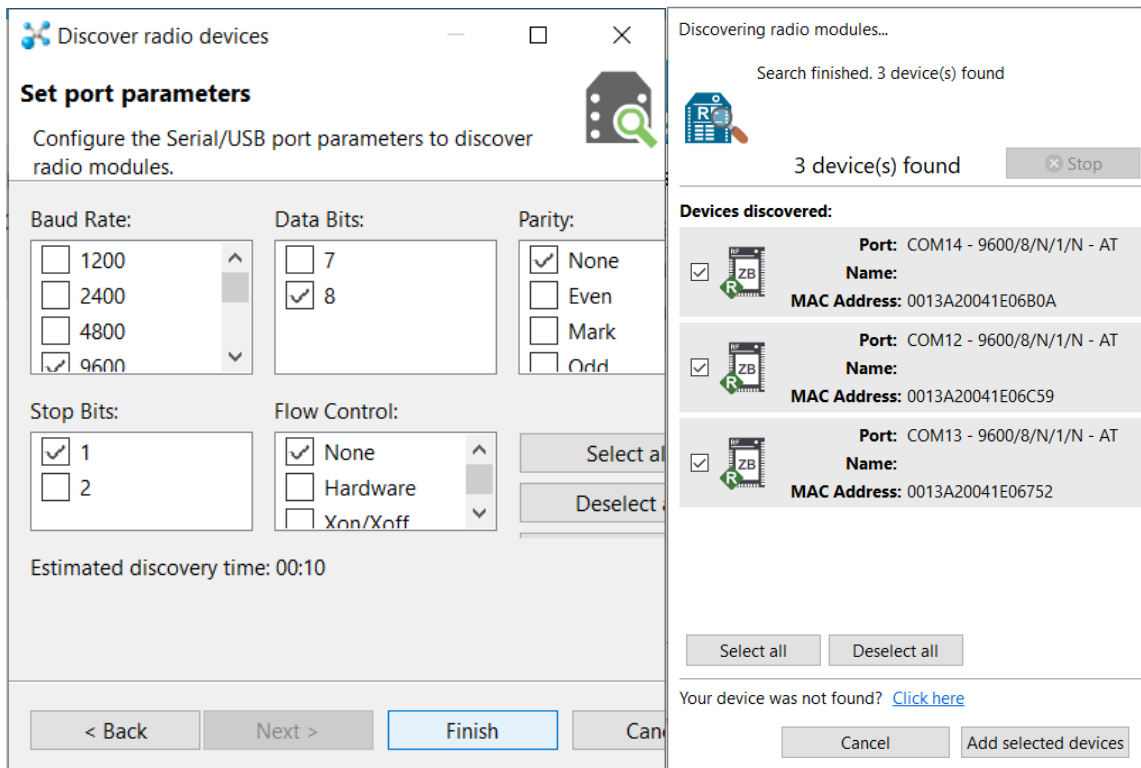


Connect the devices using the **Device Search** button on the **top toolbar**.

Lab 5 - Communicating Sensor Data over a ZigBee Network

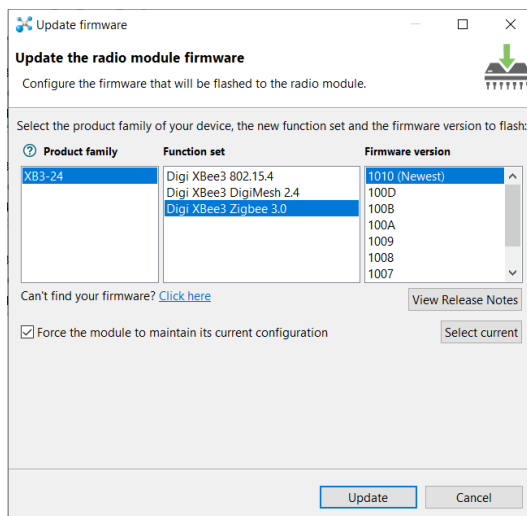
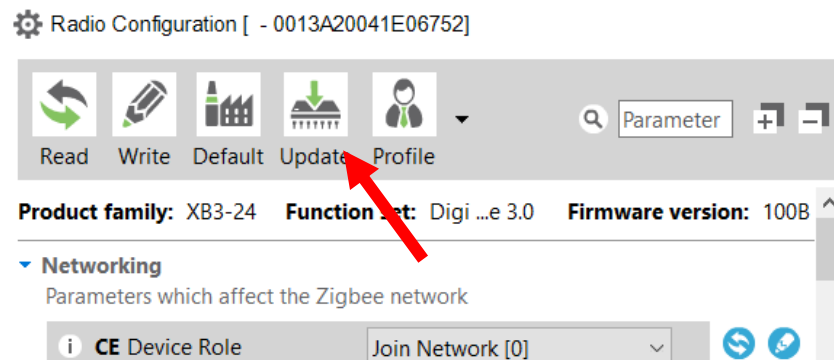
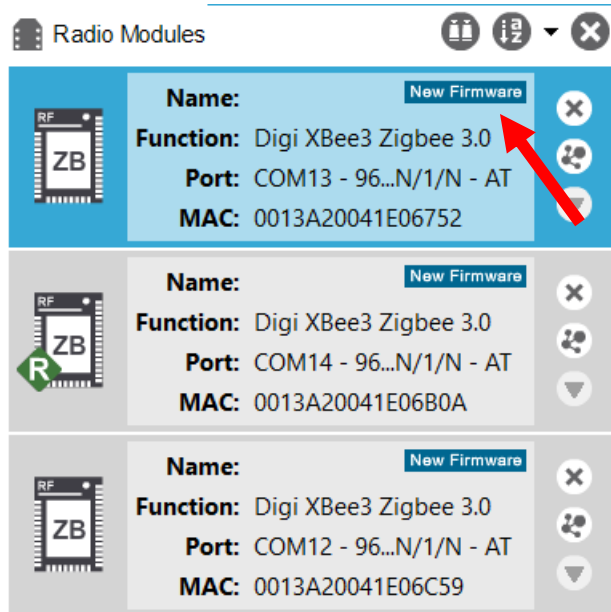


Use **default** parameters when prompted and add each of your **COM** devices.

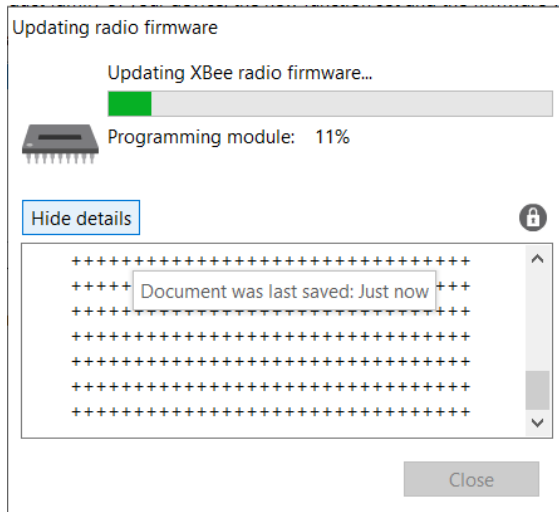
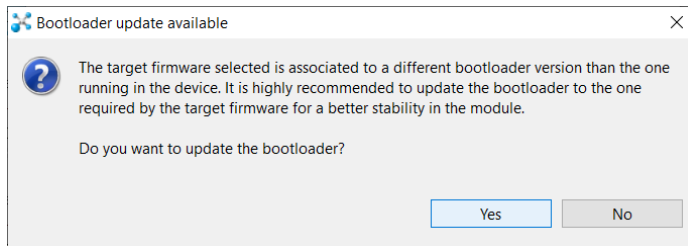


Lab 5 - Communicating Sensor Data over a ZigBee Network

You may see a **new firmware** prompt above each device. If so, update the firmware of each development board to the latest version.

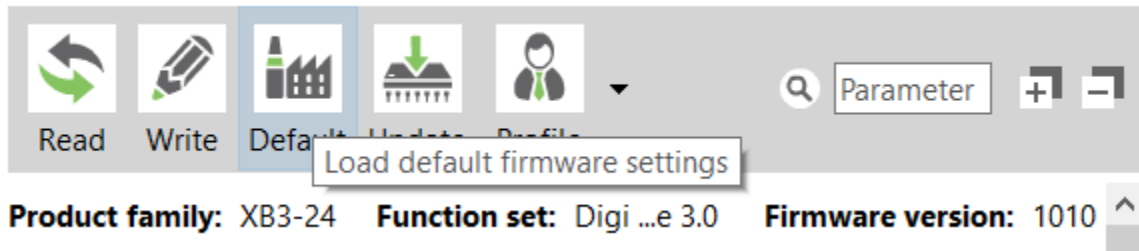


Lab 5 - Communicating Sensor Data over a ZigBee Network



Configuring Boards using XCTU

Use the **Load Default Firmware** to reset the configuration of each device.



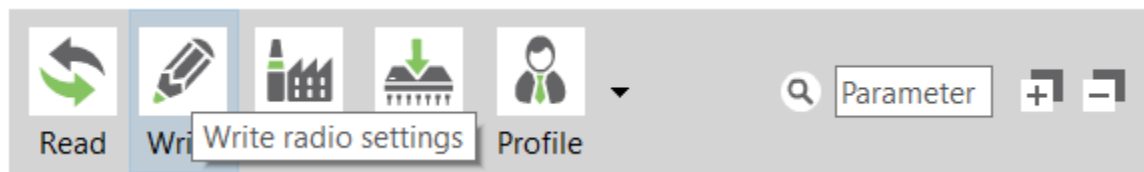
Following the table below, configure each device. (-- symbol means leave default)

USE A UNIQUE ID VARIABLE THAT IS THE SAME FOR EACH NODE IN YOUR NETWORK.

Device Setup				
Parameter	XBee A (Coordinator)	XBee B (Router)	XBee C (End Device)	Effect
ID	123	123	123	Network ID
JV	—	Enabled [1]	Enabled [1]	Join or become coordinator
CE	—	—	—	Force as coordinator
DH	—	0	0	Destination address (high)
DL	—	0	0	Destination address (low)
NI	COORD	ROUTER	END_DEVICE	Node name
SP	1F4	1F4	1F4	Sleep time = 1F4 (hexadecimal) = 500 (decimal) x 10 ms = 5 seconds.
SM	—	—	Cyclic sleep [4]	Sleep setting (only end devices sleep)
SO	—	—	2	Keep awake

There are 2 ways to write configurations:

1. Write all configurations



2. Write individual configuration

▼ **Networking**

Parameters which affect the Zigbee network

i	CE Device Role	Join Network [0]	↺	💡
i	ID Extended PAN ID	123	↺	💡

Once all of these configurations are changed and written to each XBee module, we will attempt to transmit data using **transparent mode** from the **router** to the **coordinator**.

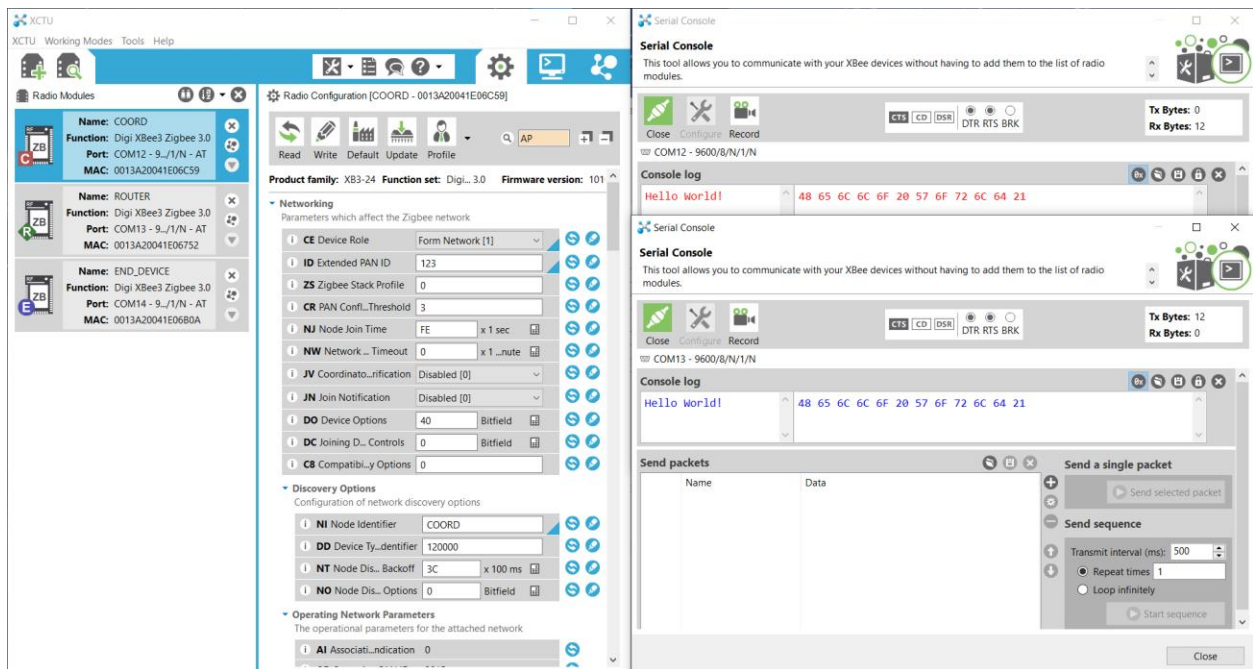
Transparent Mode

In one **console** window, open the **COORD** device.

In another **console** window, open the **ROUTER** device.

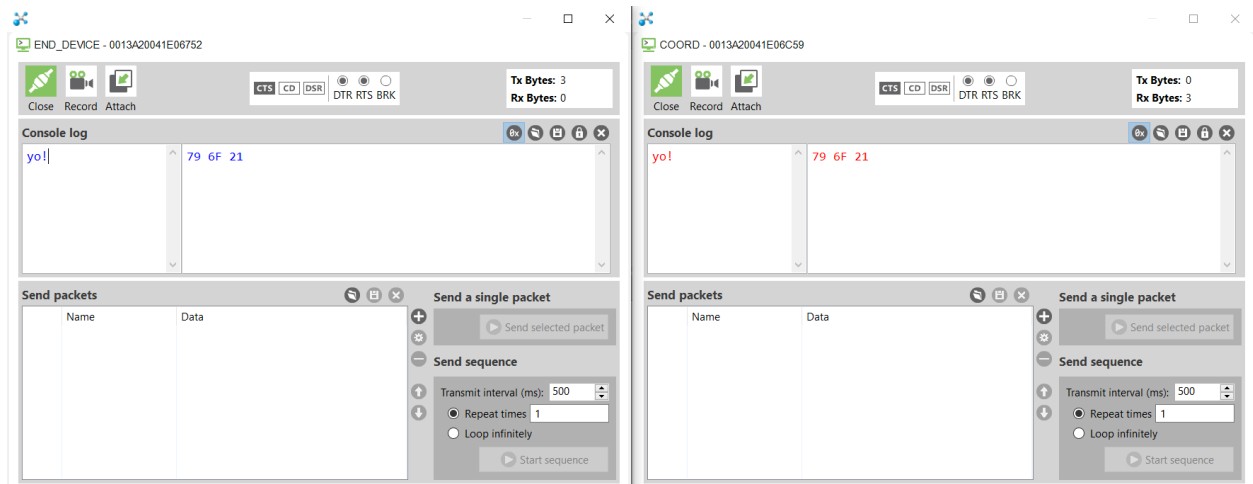
Type **Hello World!** into the **ROUTER** console window.

The data should print in the **COORD** window.

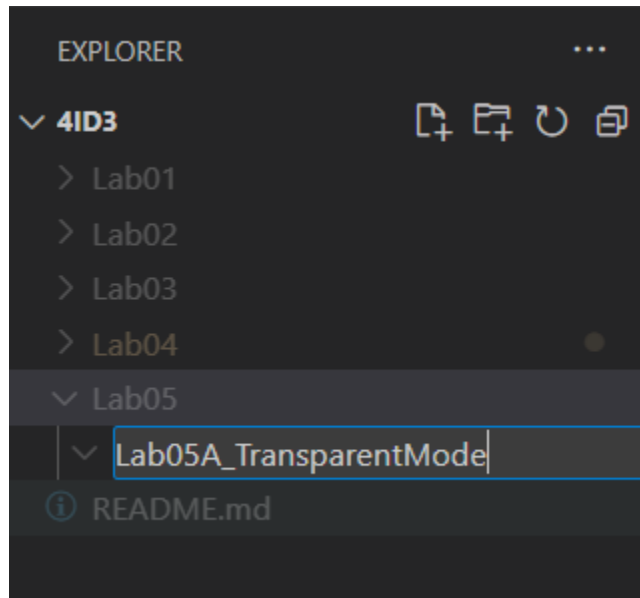


Next, type **Yo!** into the **END_DEVICE** console window.

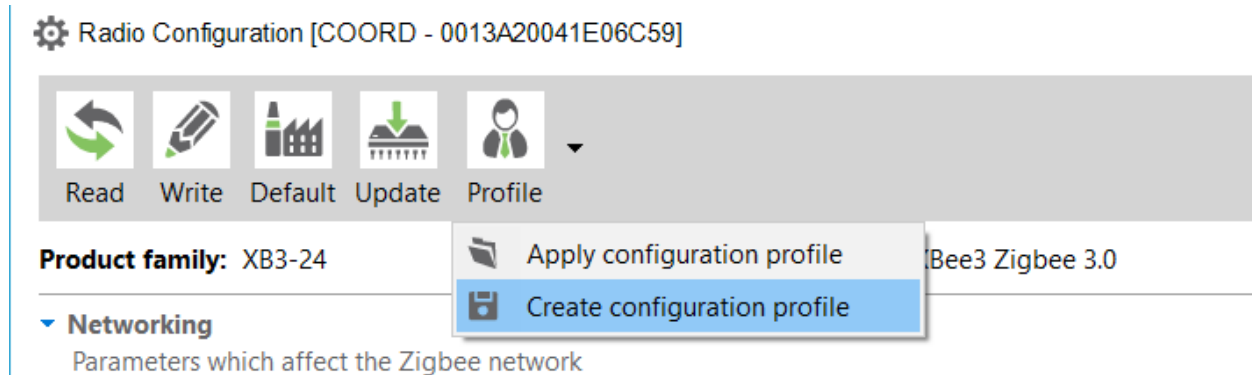
The data should print in the **COORD** window.



Create a new folder in your lab folder called **Lab05A_TransparentMode**.



In XCTU, for each device, select **Profile > Create Profile**.



Lab 5 - Communicating Sensor Data over a ZigBee Network

Create a profile

Set the profile configuration

Provide a description and configure the profile contents.

Profile configuration:

- ☒ Flash radio firmware
 - ☒ Flash if firmware is different
 - ☐ Flash always
- ☒ Reset module to factory defaults before applying settings
- ☐ Format existing file system
- ☐ Flash a new file system
- ☐ Use custom scripts for XBee Multi Programmer

Profile description:

< Back Next > **Create profile** Cancel

Press **Next** three times, then select **Create profile**. **Rename** this profile to match the device type, then **save** it to your lab folder.

Create a profile

Configure the XBee firmware settings

Choose and configure the parameters of the XBee firmware to be set in the profile.

et: Digi XBee3 Zigbee 3.0 **Firmware version:** 1010

Default

Networking

Parameters which affect the network:

- ☒ **CE** Device Role
- ☒ **ID** Extended ID
- ☐ **ZS** Zigbee Sleep
- ☐ **CR** PAN Coordinator
- ☐ **NJ** Node Join
- ☐ **NW** Network
- ☐ **JV** Coordinator
- ☒ **JN** Join Notification

Select the destination profile file.

4ID3 > Lab05 > Lab05A_TransparentMode

File name: **COORD.xpro**

Save as type: *.xpro

Save Cancel

< Back Next > **Create profile** Cancel

COORD

120000

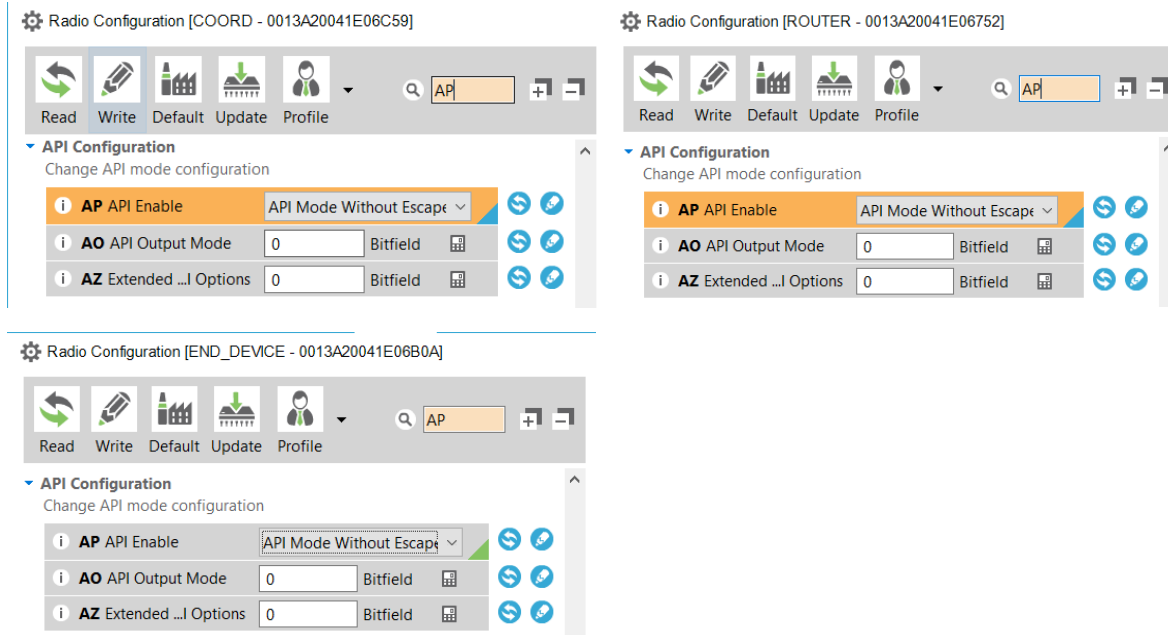
Repeat for each device.

API Mode

Navigate to the XCTU configure and modify the **AP** parameter in each device's configuration.

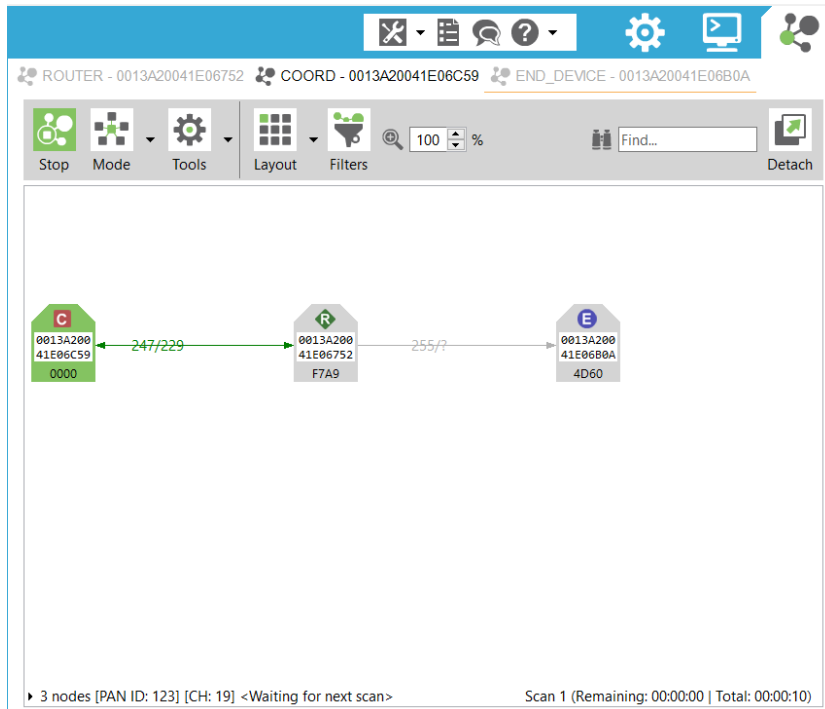
Modify **AP 0 (transparent)** -> **AP 1 (API mode without escape)**.

Press **Write**.

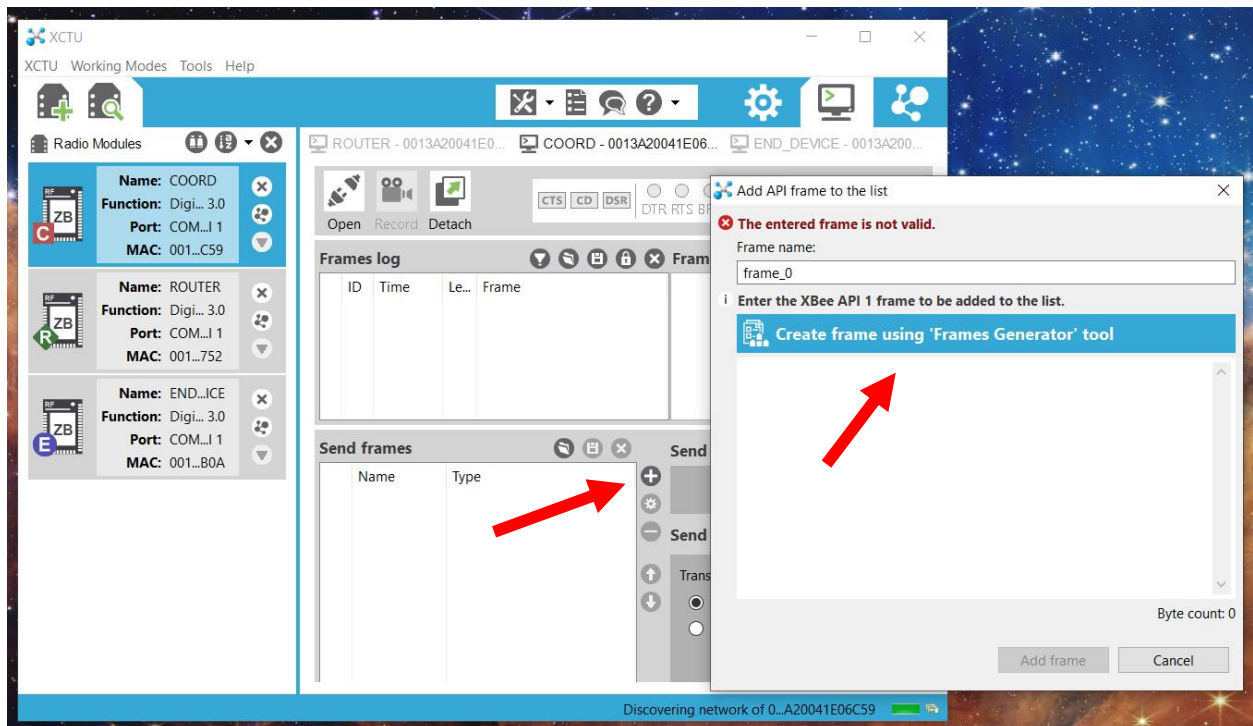


To verify that the network has been established correctly, navigate to **COORD > Networks tab** (top right).

Lab 5 - Communicating Sensor Data over a ZigBee Network



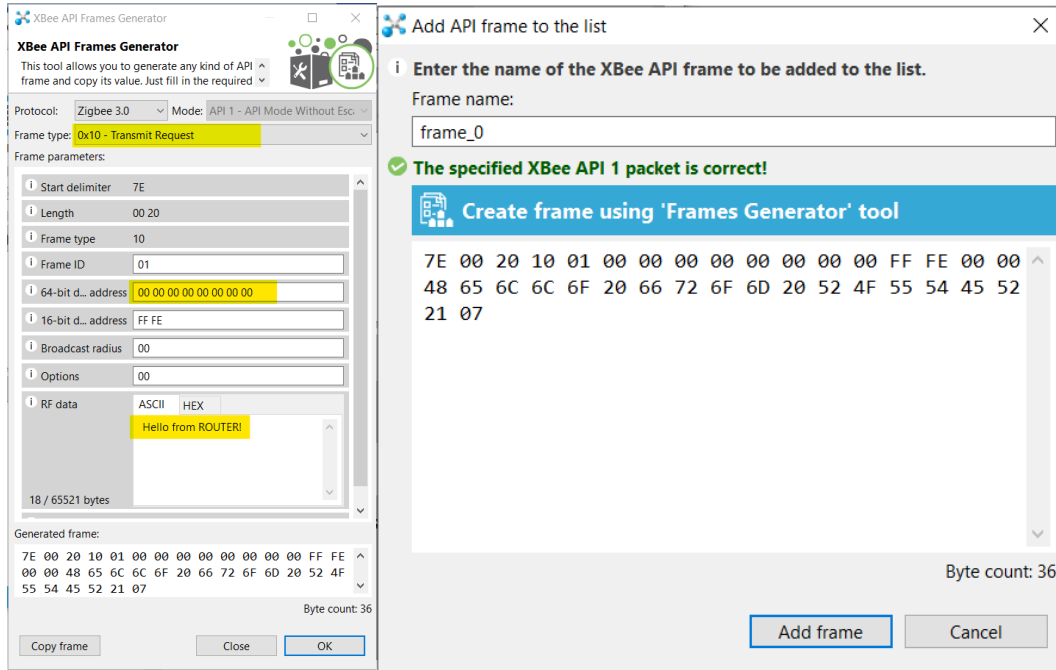
In the **ROUTER** device, create a new packet.



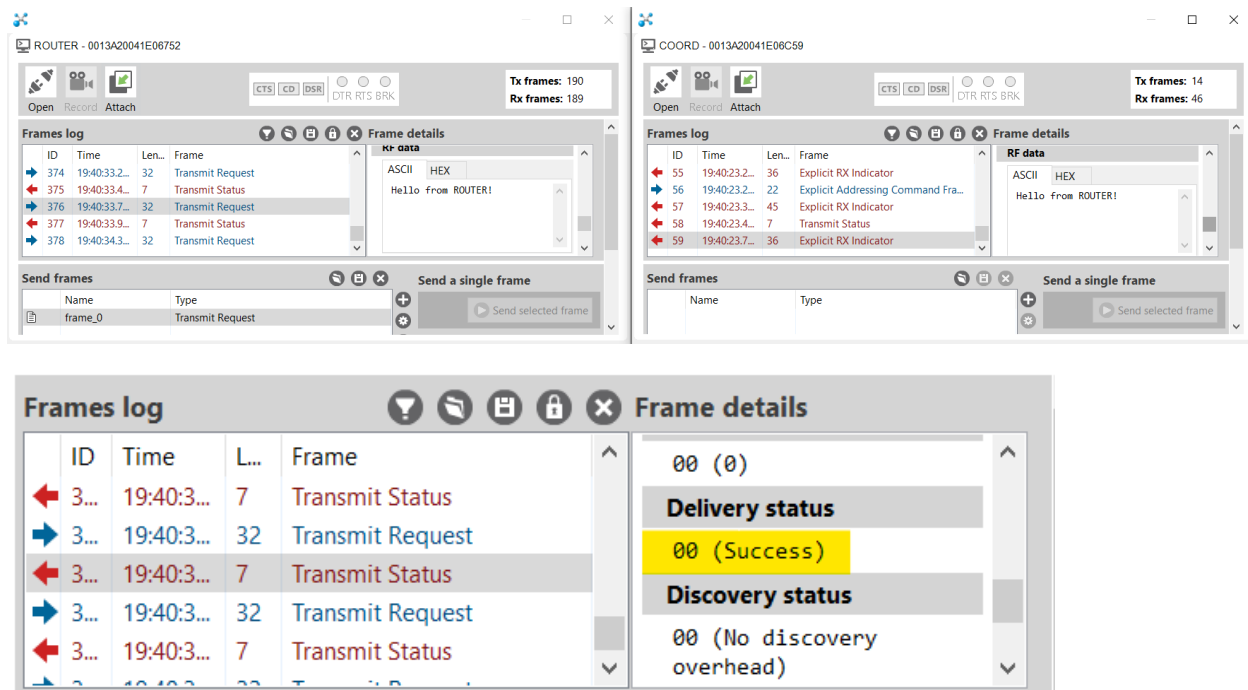
Lab 5 - Communicating Sensor Data over a ZigBee Network

Select **0x10 – Transmit Request** as the frame type. In the **address** field, ensure that it is 0x00. This is the automatic address for the coordinator device on the network. Lastly, type some ASCII characters in the **RF data** field.

Press **OK** to add the new frame.

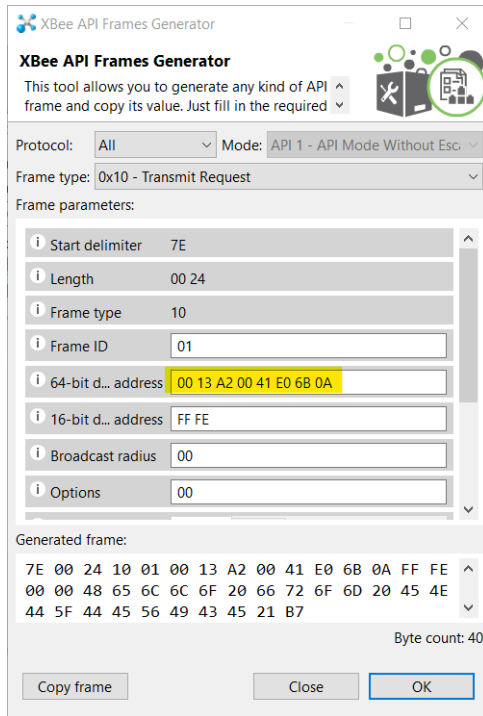


Open both the **ROUTER** and the **COORD** devices in separate console windows. Transmit the packed from the **ROUTER** to the **COORD**. Ensure that it arrives correctly, and an **OK** status is received on the **ROUTER**.



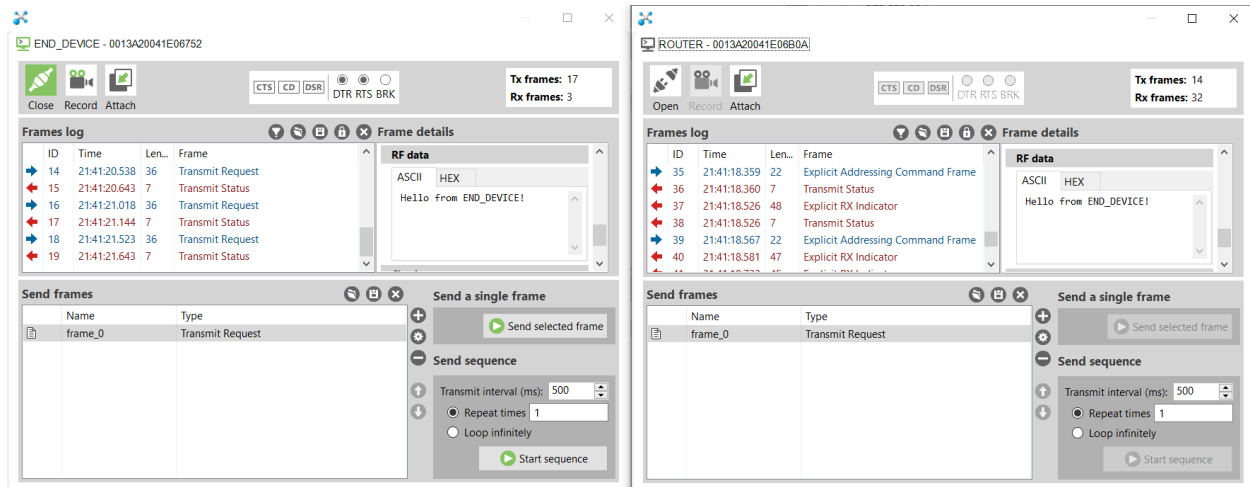
Lab 5 - Communicating Sensor Data over a ZigBee Network

Next, open both the **END_DEVICE** and **ROUTER**. Create a packet for the END_DEVICE to transmit and set its destination address to the ROUTERS MAC address.



The XBee API Frames Generator window is shown. It has a title bar with the text 'XBee API Frames Generator'. Below the title bar is a header area with a logo and a description: 'XBee API Frames Generator. This tool allows you to generate any kind of API frame and copy its value. Just fill in the required'. Below this is a section for 'Protocol' (set to 'All') and 'Mode' (set to 'API 1 - API Mode Without Esc'). The 'Frame type' is set to '0x10 - Transmit Request'. The 'Frame parameters' section contains several fields: 'Start delimiter' (7E), 'Length' (00 24), 'Frame type' (10), 'Frame ID' (01), '64-bit d... address' (00 13 A2 00 41 E0 6B 0A), '16-bit d... address' (FF FE), 'Broadcast radius' (00), and 'Options' (00). Below these fields is a 'Generated frame' section showing the hex values: 7E 00 24 10 01 00 13 A2 00 41 E0 6B 0A FF FE 00 00 48 65 6C 6C 6F 20 66 72 6F 6D 20 45 4E 44 5F 44 45 56 49 43 45 21 B7. The 'Byte count' is 40. At the bottom are buttons for 'Copy frame', 'Close', and 'OK'.

Send the packet.



Two windows from the XBee API Frames Generator are shown side-by-side. The left window is titled 'END_DEVICE - 0013A20041E06752'. It shows a 'Frames log' with a table of frames. The 'Send frames' section shows a table with one frame: 'frame_0' of type 'Transmit Request'. The 'Send a single frame' section has a 'Send selected frame' button. The 'Send sequence' section has a 'Transmit interval (ms): 500' and 'Repeat times 1' selected. The right window is titled 'ROUTER - 0013A20041E06B0A'. It shows a 'Frames log' with a table of frames. The 'Send frames' section shows a table with one frame: 'frame_0' of type 'Transmit Request'. The 'Send a single frame' section has a 'Send selected frame' button. The 'Send sequence' section has a 'Transmit interval (ms): 500' and 'Repeat times 1' selected.

You will see that the data has been sent to the ROUTER.

Now, modify that packet to contain the destination address 00 00 00 00 00 00 00 00.

Lab 5 - Communicating Sensor Data over a ZigBee Network

XBee API Frames Generator

This tool allows you to generate any kind of API frame and copy its value. Just fill in the required

Protocol: All Mode: API 1 - API Mode Without Esc.

Frame type: 0x10 - Transmit Request

Frame parameters:

Start delimiter	7E
Length	00 24
Frame type	10
Frame ID	01
64-bit d... address	00 00 00 00 00 00 00 00
16-bit d... address	FF FE
Broadcast radius	00
Options	00
RF data	ASCII HEX Hello from END_DEVICE!

Generated frame:

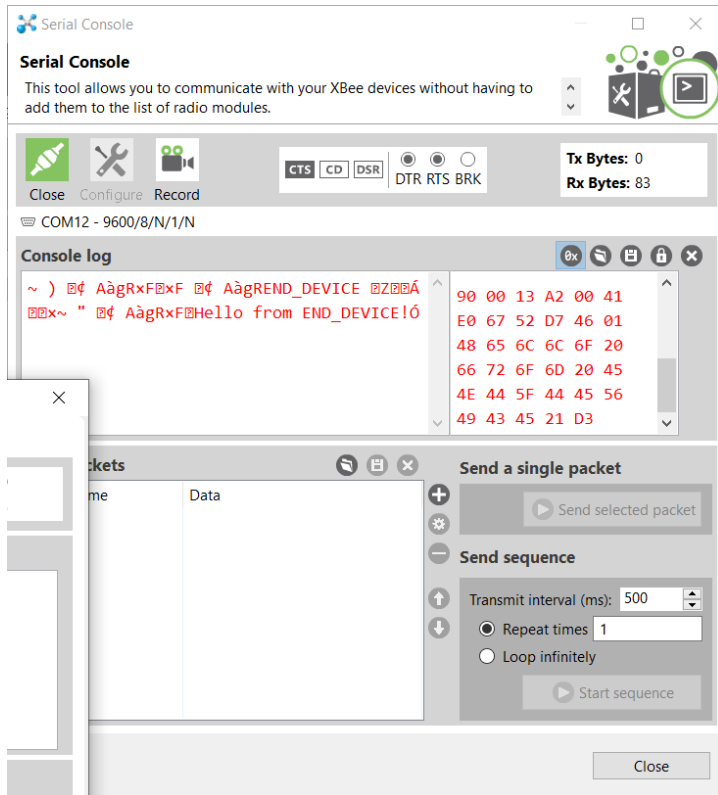
```
7E 00 24 10 01 00 00 00 00 00 00 00 00 00 FF FE
00 00 48 65 6C 6C 6F 20 66 72 6F 6D 20 45 4E
44 5F 44 45 56 49 43 45 21 02
```

Byte count: 40

Copy frame Close OK

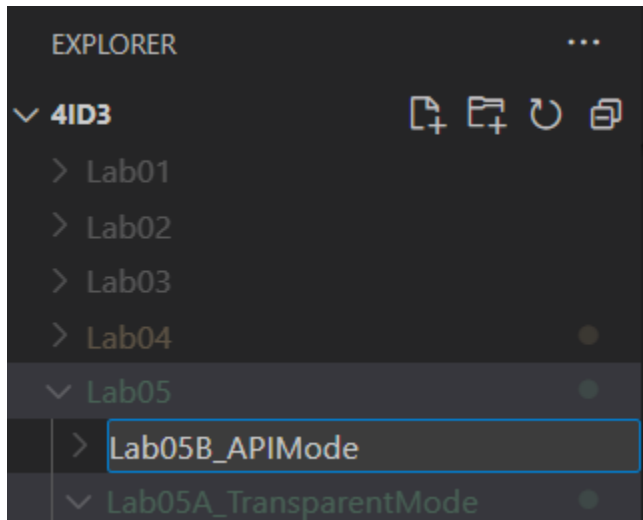
Save and transmit this message. Observe that **on END_DEVICE wake**, the data is transmitted to the **COORD** device.

Lab 5 - Communicating Sensor Data over a ZigBee Network

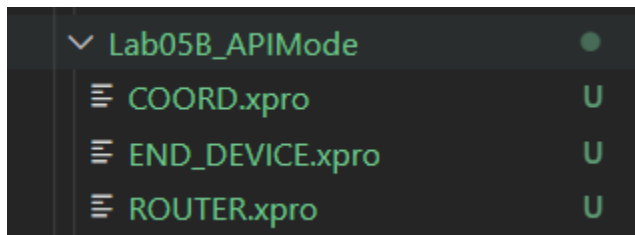
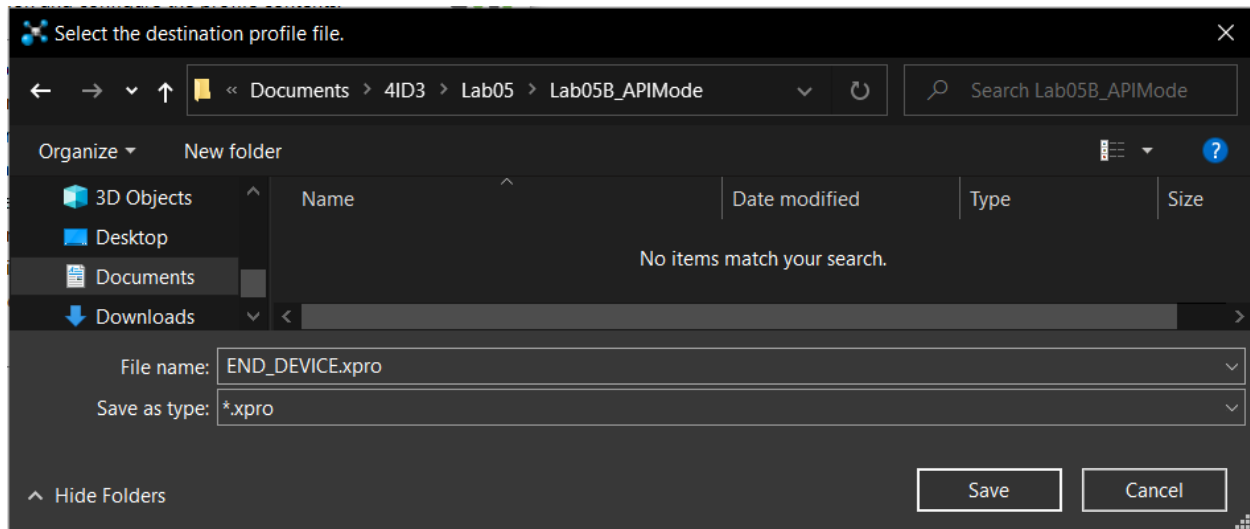


Create a new folder in your lab called **Lab05B_APIMode**.

Generate and export your profiles to this folder.



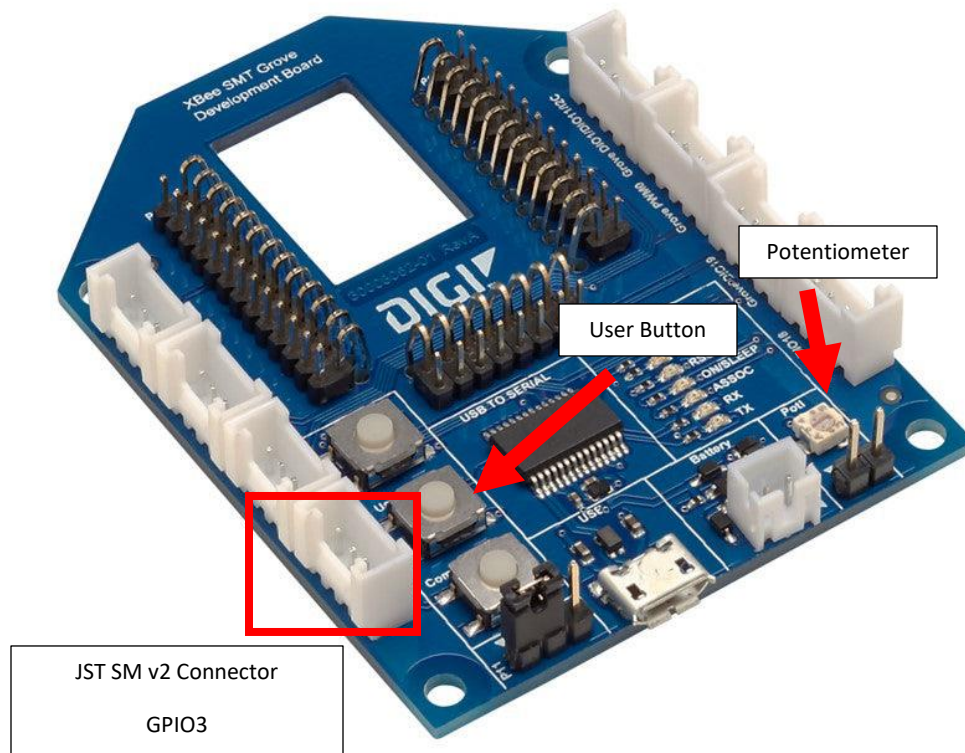
Lab 5 - Communicating Sensor Data over a ZigBee Network



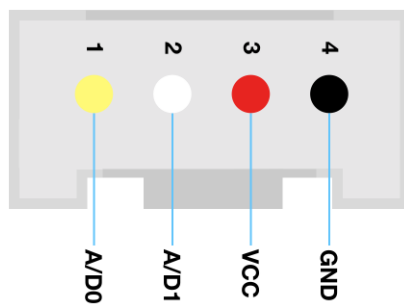
GPIO Forwarding

Connections

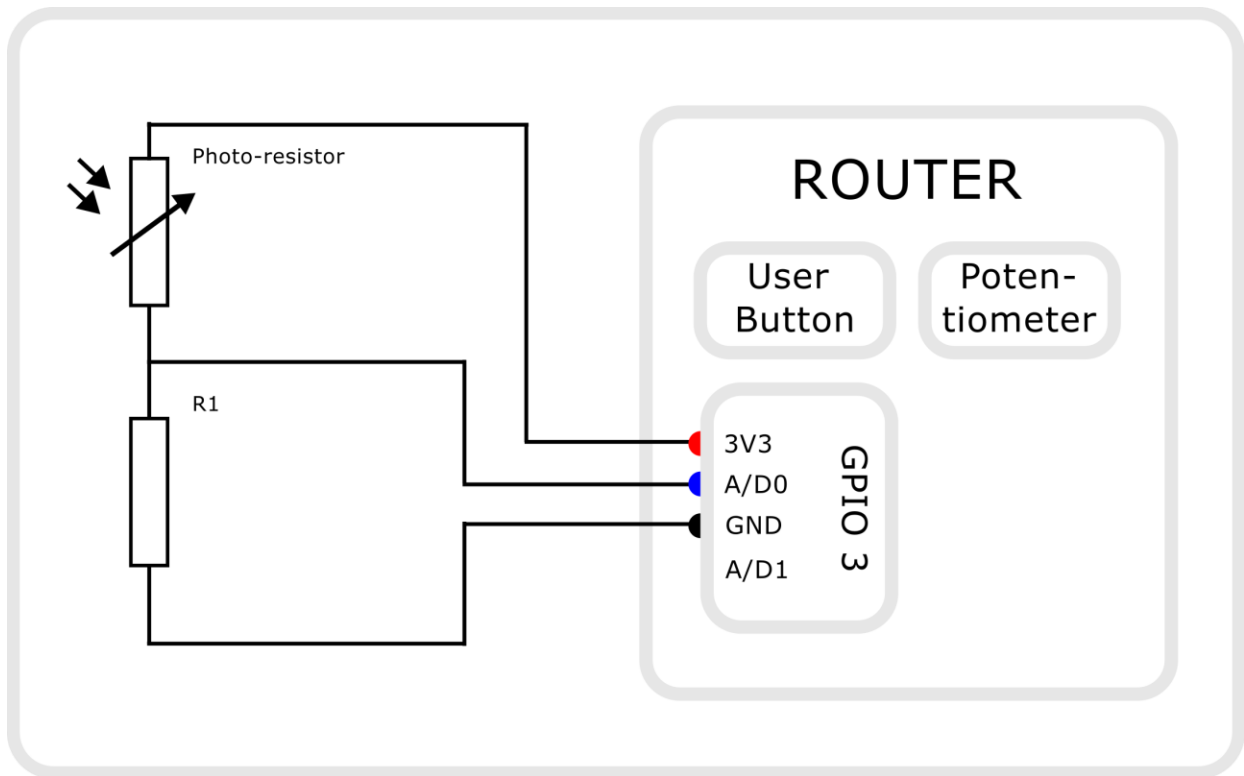
Make all connections on the **ROUTER** development board.



JST SM v2 (Grove) connector pinout:

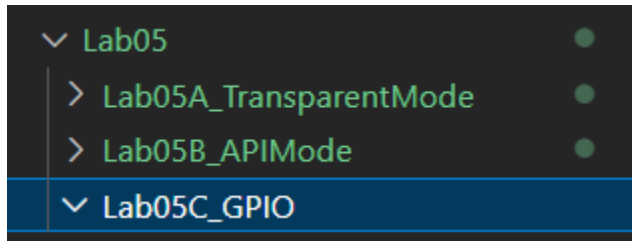


Wiring Diagram:

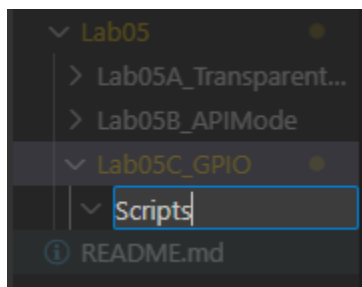


Python script setup

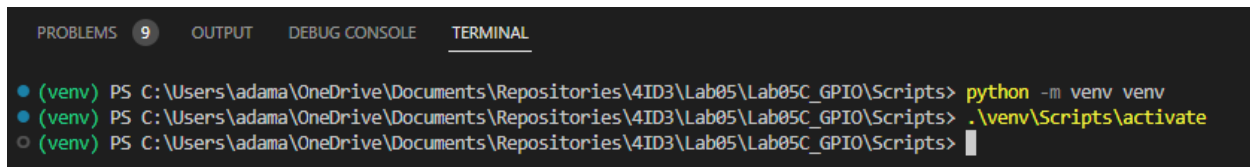
Create a new folder in your project folder called **Lab05C_GPIO**.



Inside of **Lab05C_GPIO**, create a new folder called **Scripts**.



Inside of **Scripts**, create a new virtual environment and source that virtual environment.

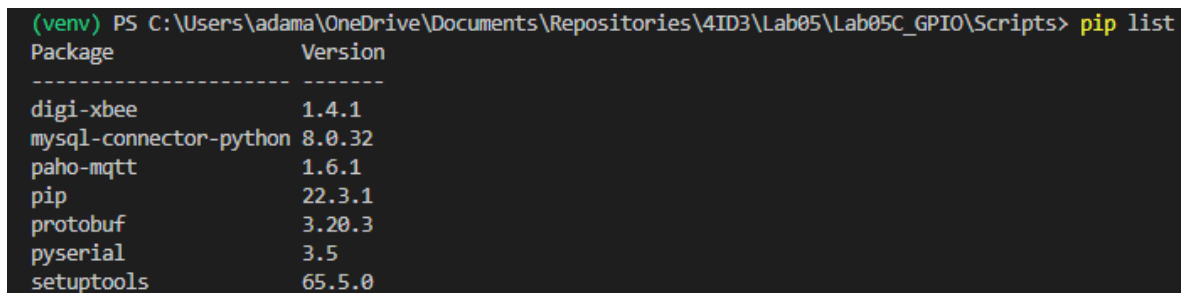


First, lets start off by installing the required libraries:

paho-mqtt

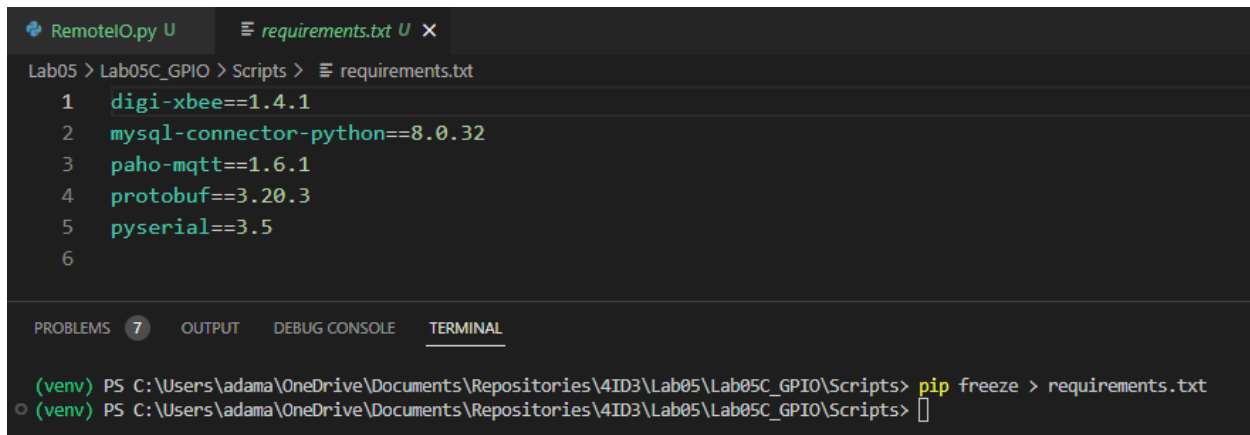
digi-xbee

mysql-connector-python



Freeze the libraries into a requirements.txt file:

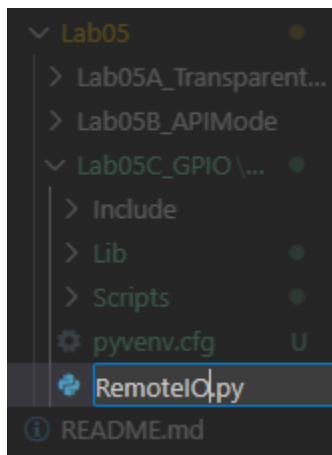
```
pip freeze > requirements.txt
```



```
Lab05 > Lab05C_GPIO > Scripts > requirements.txt
1  digi-xbee==1.4.1
2  mysql-connector-python==8.0.32
3  paho-mqtt==1.6.1
4  protobuf==3.20.3
5  pyserial==3.5
6

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL
(venv) PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab05\Lab05C_GPIO\Scripts> pip freeze > requirements.txt
(venv) PS C:\Users\adama\OneDrive\Documents\Repositories\4ID3\Lab05\Lab05C_GPIO\Scripts>
```

Create a new Python script named **RemoteDIO.py**.



Inside this blank script, paste in the provided example template.

https://github.com/sokacza/4ID3/blob/main/Lab05/Lab05C_GPIO/Scripts/RemoteIO.py

Change the configuration parameters at the top of the file. Use the **PORT** of the **COORDINATOR** device.

```
9
10 #-----
11 #       CONFIGURATIONS
12 #-----
13
14 #   Database Connection
15 HOST_IP = "localhost"
16 USER = "root"
17 PASSWORD = "9055259140"
18
19 #   MQTT Connection
20 MQTT_IP = 'test.mosquitto.org'
21 MQTT_PORT = 1883
22 GROUP_NAME = "GroupA"
23 DEVICE_ID = "DeviceA"
24
25 #   Device Connection
26 PORT = "COM7"
27 BAUD_RATE = 9600
28 REMOTE_NODE_ID = "ROUTER"
29 IO_SAMPLING_RATE = 5 # 5 seconds.
30
```

The first thing we have is a dictionary that stores what sensors are on which GPIO, what the name is, and what type of GPIO it is.

```
36 #-----
37 #       CONNECTIONS
38 #-----
39
40 #   IO initialization
41 io = dict({
42     "UserButton": {
43         "IO": IOLine.DI04_AD4,
44         "Prev": None,
45         "Curr": None,
46         "Type": "DIO"
47     },
48     "Potentiometer": {
49         "IO": IOLine.DI02_AD2,
50         "Prev": None,
51         "Curr": None,
52         "Type": "AIO"
53     },
54     "Photoresistor": {
55         "IO": IOLine.DI03_AD3,
56         "Prev": None,
57         "Curr": None,
58         "Type": "AIO"
59     }
60 })
61
```

In the class, there is a **parse** method that looks at the raw string returned by the **digixbee** library.

```
80
81     def parse(self, data: str):
82         for d in range(str(data).count(',')):
83             key = list(self._profile.keys())[d]
84             sensor = self._profile[key]
85             s = str(sensor["IO"])
86             start = str(data).index(s)
87             end = str(data).index(']', start)
88             parsed = str(data)[start + len(s) + 2 : end]
89
90             if(sensor["Type"] == "DIO"):
91                 sensor["Prev"] = sensor["Curr"]
92                 sensor["Curr"] = parsed[8:]
93
94             elif(sensor["Type"] == "AIO"):
95                 sensor["Prev"] = sensor["Curr"]
96                 sensor["Curr"] = parsed
97
98         return self._profile
99
100 sman = IOWrapper(io)
```

This breaks up the string depending on how many items there are and updates the current and previous values of the dictionary.

Each time the script runs, the database is **reset** and MQTT is connected to.

```
104     try:
105         print("---\nDatabase reset:")
106         connection = mysql.connector.connect(host=HOST_IP,
107                                             user=USER,
108                                             password=PASSWORD)
109         if connection.is_connected():
110             db_Info = connection.get_server_info()
111             print("    > Connected to MySQL Server version ", db_Info)
112             cursor = connection.cursor()
113             cursor.execute(f"DROP DATABASE `{GROUP_NAME}`")
114             print("    > Database dropped successfully")
115             cursor.close()
116             connection.commit()
117             cursor = connection.cursor()
118             cursor.execute(f"CREATE SCHEMA IF NOT EXISTS `{GROUP_NAME}` DEFAULT CHARACTER SET utf8;")
119             cursor.close()
120             connection.commit()
121             print("    > Created Database")
```



```
150
151 #   Instantiating MQTT and callback functions
152 def on_connect(client, userdata, flags, rc):
153     print("Connected to " +str(rc))
154
155 def on_message(client, userdata, msg):
156     print(msg.topic+" "+str(msg.payload))
157     data = str(msg.payload.decode('utf-8'))
158
159 client = mqtt.Client()
160 client.on_connect = on_connect
161 client.on_message = on_message
162 client.connect(MQTT_IP, MQTT_PORT, 60)
163 client.subscribe('Light', 2)
164
```

Lastly, the main method handles all the setup required.

```
174 #-----
175 #       MAIN METHOD
176 #-----
177
178 def main():
179     print(" +-----+")
180     print(" | XBee Python Library Handle IO Samples Sample |")
181     print(" +-----+\n")
182
183     device = XBeeDevice(PORT, BAUD_RATE)
184
185     try:
186         device.open()
187
188         # Obtain the remote XBee device from the XBee network.
189         xbee_network = device.get_network()
190         remote_device = xbee_network.discover_device(REMOTE_NODE_ID)
191         if remote_device is None:
192             print("Could not find the remote device")
193             exit(1)
194
195         # Set the local device as destination address of the remote.
196         remote_device.set_dest_address(device.get_64bit_addr())
197         # Enable periodic sampling every IO_SAMPLING_RATE seconds in the remote device.
198         remote_device.set_io_sampling_rate(IO_SAMPLING_RATE)
199
200
```

Lab 5 - Communicating Sensor Data over a ZigBee Network

For each sensor reading parsed out of the string, we insert a database entry and publish to an mqtt topic.

```
217     for x in data.keys():
218         print(f'{GROUP_NAME}/{DEVICE_ID}/{x}', str(data[x]["Curr"]).encode('utf-8'))
219         try:
220             client.publish(f'{GROUP_NAME}/{DEVICE_ID}/{x}', str(data[x]["Curr"]).encode('utf-8'))
221         except:
222             print("Failed to send to MQTT")
223
224         try:
225             connection = mysql.connector.connect(host=HOST_IP,
226                                                  user=USER,
227                                                  password=PASSWORD,
228                                                  database=GROUP_NAME)
229             cursor = connection.cursor()
230             ts = now.strftime("%d/%m/%Y %H:%M:%S")
231             query = f"INSERT INTO `{GROUP_NAME}`.`{DEVICE_ID}` (`Time`, `Sensor`, `Value`) VALUES ('{str(ts)}', '{str(x)}', '{str(data[x]['Curr'])}');"
232             print(query)
233             cursor.execute(query)
234             cursor.close()
235             connection.commit()
236             connection.close()
237         except:
238             print("Failed to send to MySQL")
239
```

Finally, run the Python script:

```
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL
+-----+
| XBee Python Library Handle IO Samples Sample |
+-----+

New sample received from 0013A20041E06B0A - {[IOLine.DIO4_AD4: IOValue.HIGH], [IOLine.DIO2_AD2: 830], [IOLine.DIO3_AD3: 14]}
{'UserButton': {'IO': <IOLine.DIO4_AD4: ('DIO4/AD4', 4, 'D4')>, 'Prev': None, 'Curr': 'HIGH', 'Type': 'DIO'}, 'Potentiometer': {'IO': <IOLine.DIO2_AD2: ('DIO2/AD2', 2, 'D2')>,
}, 'Photoresistor': {'IO': <IOLine.DIO3_AD3: ('DIO3/AD3', 3, 'D3')>, 'Prev': None, 'Curr': '14', 'Type': 'AIO'}}
GroupA/DeviceA/UserButton b'HIGH'
INSERT INTO `GroupA`.`DeviceA` (`Time`, `Sensor`, `Value`) VALUES ('03/03/2023 23:46:35', 'UserButton', 'HIGH');
GroupA/DeviceA/Potentiometer b'830'
INSERT INTO `GroupA`.`DeviceA` (`Time`, `Sensor`, `Value`) VALUES ('03/03/2023 23:46:35', 'Potentiometer', '830');
GroupA/DeviceA/Photoresistor b'14'
INSERT INTO `GroupA`.`DeviceA` (`Time`, `Sensor`, `Value`) VALUES ('03/03/2023 23:46:35', 'Photoresistor', '14');
```

Click your button change your potentiometer, and cover your photoresistor as data is being collected.

And check your database:

```
mysql> USE `GroupA`;
Database changed
mysql> SELECT * FROM `GroupA`.`DeviceA`;
+----+-----+-----+-----+
| id | Time                | Sensor      | Value |
+----+-----+-----+-----+
| 1  | 03/03/2023 23:53:45 | UserButton  | HIGH  |
| 2  | 03/03/2023 23:53:45 | Potentiometer | 830   |
| 3  | 03/03/2023 23:53:45 | Photoresistor | 1023  |
| 4  | 03/03/2023 23:53:50 | UserButton  | HIGH  |
| 5  | 03/03/2023 23:53:50 | Potentiometer | 830   |
| 6  | 03/03/2023 23:53:50 | Photoresistor | 1023  |
| 7  | 03/03/2023 23:53:55 | UserButton  | HIGH  |
| 8  | 03/03/2023 23:53:55 | Potentiometer | 830   |
| 9  | 03/03/2023 23:53:55 | Photoresistor | 22    |
| 10 | 03/03/2023 23:53:57 | UserButton  | LOW   |
| 11 | 03/03/2023 23:53:57 | Potentiometer | 830   |
| 12 | 03/03/2023 23:53:57 | Photoresistor | 22    |
| 13 | 03/03/2023 23:53:58 | UserButton  | HIGH  |
| 14 | 03/03/2023 23:53:58 | Potentiometer | 830   |
| 15 | 03/03/2023 23:53:58 | Photoresistor | 22    |
| 16 | 03/03/2023 23:53:58 | UserButton  | LOW   |
| 17 | 03/03/2023 23:53:58 | Potentiometer | 830   |
| 18 | 03/03/2023 23:53:58 | Photoresistor | 22    |
| 19 | 03/03/2023 23:53:58 | UserButton  | HIGH  |
| 20 | 03/03/2023 23:53:58 | Potentiometer | 830   |
| 21 | 03/03/2023 23:53:58 | Photoresistor | 22    |
| 22 | 03/03/2023 23:54:00 | UserButton  | HIGH  |
| 23 | 03/03/2023 23:54:00 | Potentiometer | 830   |
| 24 | 03/03/2023 23:54:00 | Photoresistor | 19    |
| 25 | 03/03/2023 23:54:02 | UserButton  | LOW   |
| 26 | 03/03/2023 23:54:02 | Potentiometer | 830   |
| 27 | 03/03/2023 23:54:02 | Photoresistor | 19    |
| 28 | 03/03/2023 23:54:02 | UserButton  | HIGH  |
```

Exercise A

Use NodeRED to create a dashboard and visualize the data. Submit a screenshot with your report.

Exercise B

Can be done at home. Either modify the original code or create a separate Python script to export the data to an CSV file. Graph each sensor as its own graph. Submit those 3 screenshots with your report.

Hint: Look at the previous lab

(Optional) Exercise C

Can be done at home. Create a .Net WinForms desktop application to query the database and display the data. A template has been provided in the lab GitHub. Submit a screenshot with your report.

Syncing with GitHub

Use the following commands to resync your Git repo with your GitHub remote repo.

```
git add .
```

```
git commit -m "Lab 04 completed"
```

```
git push origin main
```

```
git pull origin main
```

END