2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #32517

MicroPython in a Wireless Communications Systems Course

Prof. David R. Loker, Pennsylvania State University, Behrend College

David R. Loker received the M.S.E.E. degree from Syracuse University in 1986. In 1984, he joined General Electric (GE) Company, AESD, as a design engineer. In 1988, he joined the faculty at Penn State Erie, The Behrend College. In 2007, he became the Chair of the Electrical and Computer Engineering Technology Program. His research interests include wireless sensor networks, data acquisition systems, and communications systems.

Nathan Wayne Brubaker, Pennsylvania State University, Behrend College
Mr. Daniel Albert Bohbot, Pennsylvania State University, Behrend College

I am from Conneaut, OH and currently a senior Electrical and Computer Engineering Technology student at Penn State Behrend. My future plans are to attend the Renewable Energy and Sustainability Systems masters program through Penn State World Campus. With this degree I hope to work in developing and improving renewable energy systems.

MicroPython in a Wireless Communications Systems Course

Abstract

Part of the requirements for an Electrical Engineering Technology program or Computer Engineering Technology program includes the ability for students to design and implement embedded systems in a variety of courses. Typical courses can include embedded processors, instrumentation and measurement systems, wireless communications systems, networking, and control systems, and they can be lower-division and/or upper-division courses. Due to the free and open source software availability of Python, and its popularity, a small subset of the Python standard library (called MicroPython) is optimized to run on a variety of microcontrollers for embedded applications. Additionally, some of these microcontrollers have wireless capabilities. One such device includes the Digi XBee3 module, which implements the Zigbee, IEEE 802.15.4, and Bluetooth Low Energy (BLE) protocols. Another wireless device is the ESP8266 Wi-Fi module, which implements the IEEE 802.11 family of protocols. These devices are inexpensive and suitable for embedded applications in a wireless communications systems course. The Community Edition of PyCharm is available for free and can be used as the software development environment. The goal of this paper is to introduce a series of labs, utilizing the Digi XBee3 module, that can be used within a variety of courses, including a wireless communications systems course. Possible topics for lab projects include: network connectivity, analog-to-digital conversion, sensor data collection, pulse-width modulation (PWM), digital input/output, Universal Asynchronous Receiver/Transmitter (UART) communication, and inter-integrated circuit (I^2C) protocol. For lab projects, engineering requirements, software code or flowcharts, and schematics are provided.

Introduction

In an Electrical Engineering Technology program or Computer Engineering Technology program, there are many courses that use embedded systems to meet the program requirements. The format for these courses is often to teach C/C++ programming, with embedded hardware and software as a core component to the course. An example is the usage of a Programmable System-on-Chip (PSoC 5LP) device in engineering technology programs for embedded applications [1-2]. This device is programmed in C and utilized in a variety of courses for many laboratory projects. Another device is the BeagleBone Black (BBB), which can be programmed in C++ and used for various measurement and control applications [3]. Due to the free and open source software availability of Python and with the growing interest in graduates having Python experience, some programs are supplementing C/C++ courses with Python [4]. A small subset of the Python standard library (called MicroPython) is optimized to run on a variety of microcontrollers for embedded applications [5]. One such device includes the Digi XBee3 module, which implements the Zigbee, IEEE 802.15.4, and BLE protocols [6].

MicroPython on the XBee3

The MicroPython programming guide has a listing of the modules that can be used with the XBee3 [7]. These modules consist of the following.

The machine module contains specific functions associated with the XBee3 module. The

following classes can be used from the machine module.

- machine.PWM() for enabling pulse-width modulation
  - duty() method for setting the duty cycle
  - deinit() method for turning off PWM on the specific pin
- machine.ADC() for enabling analog to digital conversion
  - read() method for reading the 12-bit ADC value
- machine.I2C() for enabling I^2C
  - scan() method for scanning for slaves
  - writeto_mem() method for writing bytes to a slave address
  - readfrom_mem() method for reading bytes from a slave address
- machine.Pin() for configuring the specific pin
  - value() method for setting or getting the value of the pin
  - mode() method for specifying whether the pin is an input or output
  - pull() method for enabling an internal pull-up or pull-down resistor on an input pin

XBee-specific functions are shown below.

- xbee.atcmd() for setting or querying an AT command
- xbee.discover() for performing a network discovery
- xbee.receive() for receiving packets
- xbee.transmit() for transmitting a packet to a specific destination

System-specific functions can be used for accessing the primary UART for communication.

- sys.stdin.buffer.read() for reading from the UART
- sys.stdout.buffer.write() for writing to the UART

For the software development environment, the Community Edition of PyCharm is available for free [8]. An XBee plugin is available from PyCharm's plugins marketplace, and installation instructions are shown in the plugin user guide [9]. There are numerous sample programs available, as shown in Figure 1. One such program is the ADC Polling example shown in Figure 2. This program reads an analog value on "AD3" (pin 17) of the XBee3 module, uses the internal ADC to convert to a binary value, and converts to a voltage based on the resolution and full-scale voltage setting of the Analog-to-digital Converter (ADC) [10]. After a one second wait, the program re-loops.

Wireless Communications Systems Course Overview

A wireless communications systems course can be a lower or upper division course in an Electrical Engineering Technology program or Computer Engineering Technology program. The course can provide foundational material in spectrum analysis, filtering, serial communications, and analog and digital modulation and demodulation. Applications can include wireless networking for the development of personal area networks (PAN) and local area networks (LAN). An embedded system can be used as the primary means for remote sensor data collection and transmission. MicroPython can be used for programming the embedded device.

Typical lab projects with an embedded system (i.e., XBee3 module) using MicroPython can

include the following.

- LED Driver (Pin output interface)
- Serial LCD Display (UART interface)
- Temperature Data Collection (ADC interface)
- Accelerometer Interface (I^2C protocol)
- Wireless Communication Link
- Wireless Sensor Network

The goals of this paper are to show how an inexpensive embedded device (i.e., XBee3 module) with MicroPython can be used in a wireless communications systems course to implement several of the above lab projects, and to evaluate its effectiveness.

Figure 1 – Listing of MicroPython Example Programs

```
1.  from machine import ADC
2.  import xbee
3.  import time
4.
5.  # Pin D3 (AD3/DIO3)
6.  ADC_PIN_ID = "D3"
7.
8.  # ADC reference voltage
9.  AV_VALUES = {0: 1.25, 1: 2.5, 2: 3.3, None: 2.5}
10.
11. print(" +--------------------------------------+")
12. print(" | XBee MicroPython ADC Polling Sample |")
13. print(" +--------------------------------------+\n")
14.
15. # Read the module's Analog Digital Reference
16. try:
17.     av = xbee.atcmd("AV")
18. except KeyError:
19.     # Reference is set to 2.5 V on XBee 3 Cellular
20.     av = None
21. reference = AV_VALUES[av]
22. print("Configured Analog Digital Reference: AV:{}, {} V".format(av, reference))
23.
24.
25. # Create an ADC object for pin DIO0/AD0.
26. adc_pin = ADC(ADC_PIN_ID)
27.
28. # Start reading the analog voltage value present at the pin.
29. while True:
30.     value = adc_pin.read()
31.     print("- ADC value:", value)
32.     print("- Analog voltage [V]:", value * reference / 4095)
33.     time.sleep(1)
```

Figure 2 – ADC Polling Example Program

System Block Diagram

The system block diagram for the lab projects presented in this paper is shown in Figure 3. For lab projects requiring a communication link between two XBee3 modules, one module is configured as a coordinator and the 2nd module can be configured as either a router or end device, according to the Zigbee protocol. Alternatively, if a communication link is not needed, only one XBee3 module with a breakout board is used [11]. The FTDI USB to TTL converter is used to power the XBee3 module and convert to TTL logic levels for transmitting and receiving [12]. If there are no connections to the XBee3, an explorer dongle can be used to connect the XBee3 directly to the USB port on the PC [13].

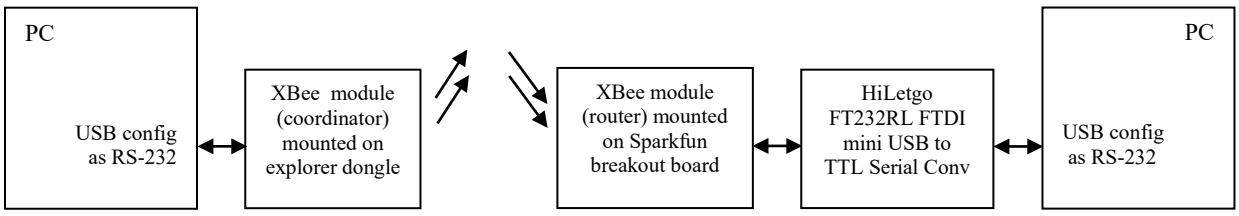

Figure 3 – System Block Diagram

Lab Projects

Possible topics for lab projects include: network connectivity, analog-to-digital conversion, sensor data collection, pulse-width modulation (PWM), digital input/output, serial communication (UART), and inter-integrated circuit (I^2C) protocol. The goal for this paper is to discuss the following labs.

1. Introduction to Python using PyCharm – The objectives are to introduce the PyCharm IDE and to review Python programming. Two projects are created: A simple "Hello World" project, and then another project that calculates the average value of 10 simulated temperatures (each temperature is simulated by scaling a random number to represent temperature between 30°C and 50°C).
2. Temperature Data Collection with XBee3 module – The objectives are to connect a temperature sensor (LM35) to an analog input pin on the XBee3, use the internal ADC to convert into a binary value, use software to convert into temperature in degrees C, and display the temperature on an LCD display using the UART interface.
3. Accelerometer Interface to XBee3 module – The objectives are to connect an accelerometer device (ADXL345) to the XBee3 using the I^2C pins, read the x, y, and z-axis values, convert these values to gravity, filter the data to remove noise, and transmit the values to the PC using the UART interface.
4. Remote Sensor Network with XBee3 modules – The objectives are to configure two XBee3 modules (one as a coordinator and the other as a router) to set up a communication link, transmit temperature information from the router to the coordinator, and display the information on a PC using the UART interface.

Introduction to Python using PyCharm

The engineering requirements for this lab project are listed below.

- Develop two Python projects using PyCharm.
    - Project 1: **Hello World** project that prints Hello World to the output console.
    - Project 2: **Temp Average** project that calculates the average value of 10 temperature readings. Each temperature reading is a floating-point number generated from a random number that is scaled to represent temperature in °C between 30 and 50. The program delays for one second after each number is generated. The program prints each average value to the output console.

Online resources are available for help with learning PyCharm [14]. Additionally, a free open educational resource (OER) text is available online for students that do not have a background in Python [15]. Figure 4 shows the Python code for the **Temp Average** project (filename is **random_num.py**).

```
1. print("This program determines the average value of a set of floating point
   random numbers.")
2. #
3. import random
4. import time
5. #
6. rand_sum=0.0
7. loop_num=10
8. for x in range(loop_num):
9.     # Create a floating point random number representing temp in deg C between
   30 and 50
10.    rand_num = 20.0*random.random()+30.0
11.    print("random number is ", "%.4f" % rand_num)
12.    rand_sum=rand_sum+rand_num
13.    time.sleep(1)  # Delay for 1 sec
14. avg_num=rand_sum/loop_num
15. print("average number is ", "%.3f" % avg_num)
```

Figure 4 – **Temp Average** Python Project Code

Temperature Data Collection

The engineering requirements for this lab project are listed below.

- LM35 used as the temperature sensor
- Internal ADC component used for reading sensor voltage
- Full-scale of 1.25V with 12 bits of resolution
- Output temperature in degrees C
- Temperature shown on an LCD serial display using the UART interface

The schematic is shown in Figure 5. The LM35 provides an analog output with a temperature coefficient of 10mV/°C. It is connected to an ADC input on the XBee3 module. The HiLetgo module is an FTDI USB to TTL converter used to program and power the XBee3 module. The serial LCD is used to connect directly to the UART interface of the XBee3 module.

The code is shown in Figure 6. After the LCD is initialized, the ADC is used to read the voltage from the LM35. The ADC has a resolution of 12 bits, and it defaults to a full-scale voltage of 1.25V. The voltage is converted to temperature in °C, and then the temperature is filtered to remove noise.

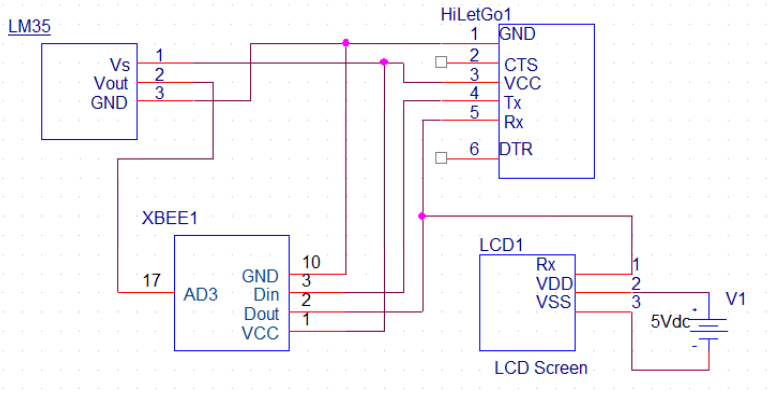

Figure 5 – Temperature Data Collection

```
1.  # imports ADC class from the machine library
2.  from machine import ADC
3.  # imports stdout class from the sys library
4.  from sys import stdout
5.  import time
6.
7.  # List of LCD commands as an array of bytes.
8.  # returns the cursor to line 1 column 1
9.  home = bytearray([0xFE, 0x46])
10. # clears the entire display and places the cursor at line 1 column 1
11. clear = bytearray([0xFE, 0x51])
12. # sets the display contrast where the contrast setting can be between 1 and 50
13. contrast = bytearray([0xFE, 0x52, 50])
14.
15. # Renames each class used in code.
16. # create an ADC object for pin D3 (AD3/DIO3)
17. adc = ADC('D3')
18. # stdout assigned to variable "lcd"
19. lcd = stdout
20.
21. # Initialize LCD screen.
22. # sets LCD to max contrast
23. lcd.write(contrast)
24. # clears entire LCD screen
25. lcd.write(clear)
26.
27. # initialize variable for average
28. avg = 0
29. while True:
30.     lcd.write(home)
31.      raw_adc = adc.read()
32.     # scales 12 bit A to D data to 0 - 1250 mV
33.     mv_adc = (raw_adc * 1250.0) / 4095.0
34.     # each mv represents 1 degree C
35.     temp_c = mv_adc / 10.0
36.     # equation used for averaging temp data
37.     avg = 0.8 * avg + 0.2 * temp_c
38.     # average temp rounded to 1 decimal and displayed to LCD as a string
39.     lcd.write('Temp in C: ' + str(round(avg, 1)))
40.     time.sleep(1)
```

Figure 6 – Code for Temperature Data Collection

Accelerometer Interface

The engineering requirements for this lab project are listed below.

- Analog Devices ADXL345 as the accelerometer
- I^2C interface to the ADXL345
- Full-scale of +/- 16 g with 13 bits of resolution
- 3-axis outputs (x, y, and z) in G's
- Output values transmitted to the PC using the UART interface

The schematic is shown in Figure 7. The FTDI USB to TTL HiLetgo module is used to program and power the XBee3 module. The ADXL345 accelerometer is connected directly to the XBee3 module using the I^2C interface [16]. Output from the ADXL345 is displayed within the PyCharm IDE.

The code is shown in Figure 8. In Figure 8(a), there are several internal registers for the ADXL345 that set the output bit rate, bandwidth, resolution, and full-scale range. The values are written to the memory addresses using the i2c.writeto_mem() function. Figure 8(b) shows the function for determining the gravity value. Lastly, in Figure 8(c), all three axis (x, y, and z) gravity values are determined. Each is filtered to remove noise.

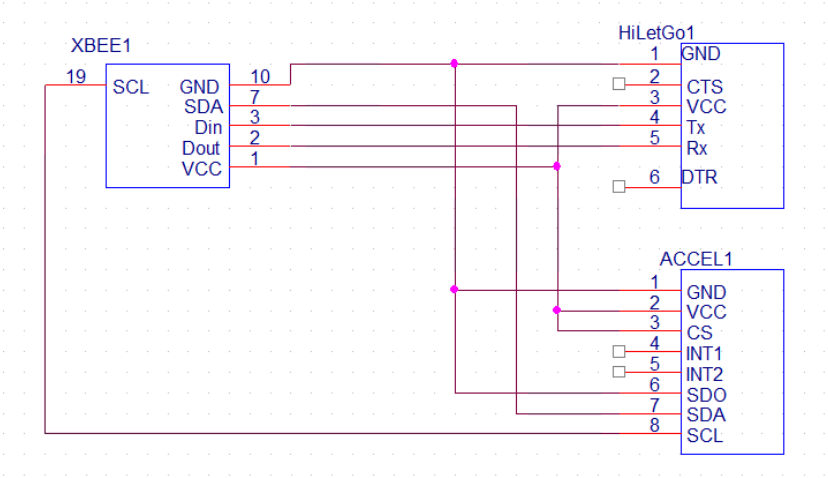

Figure 7 – Accelerometer Interface Schematic

```
from machine import I2C
import time

# This is the Slave address for I2C protocol
ADXL345_ADDR = 0x53
# Accelerometer registers for the X,Y, and Z data
X0_REG = 0x32
X1_REG = 0x33
Y0_REG = 0x34
Y1_REG = 0x35
Z0_REG = 0x36
Z1_REG = 0x37

# Accelerometer configuration registers locations and values
REG_2C_BW = 0x2C
BW_100HZ = 0x0B

REG_2D_PWR = 0x2D
PWR_MEASURE = 0x08

REG_31_CTL = 0x31
CTL_16G = 0x0B

REG_38_FIFO = 0x38
FIFO_BYPASS = 0x00

# Enable I2C protocol for serial communication and set the overall frequency
i2c = I2C(1, freq=100000)

# Configure the accelerometer
i2c.writeto_mem(ADXL345_ADDR, REG_2C_BW, bytearray([BW_100HZ]))
i2c.writeto_mem(ADXL345_ADDR, REG_2D_PWR, bytearray([PWR_MEASURE]))
i2c.writeto_mem(ADXL345_ADDR, REG_31_CTL, bytearray([CTL_16G]))
i2c.writeto_mem(ADXL345_ADDR, REG_38_FIFO, bytearray([FIFO_BYPASS]))

```

Figure 8(a) –Configuration Code for Accelerometer

```
# This function reads the data from the accelerometer then formats it. Once formatted it detects if the value is negative or not
def get_data(slave, data0, data1):
    data0val = i2c.readfrom_mem(slave, data0, 1)
    data1val = i2c.readfrom_mem(slave, data1, 1)
    Actualdata = (int.from_bytes(data1val, "big") << 8) | int.from_bytes(data0val, "big")
    if Actualdata > 32767:
        Actualdata -= 65536
    return Actualdata

# This function converts the value read from the accelerometer into G values
def convert_G(Actualdata):
    Gdata = (Actualdata / 4095) * 16
    return Gdata
```

Figure 8(b) – Function Definition Code for Accelerometer

```
# Default values for X,Y and Z need to be zero for the averaging to work properly
X = 0
Y = 0
Z = 0
while True:
    # This block of code gets the byte data of X axis from the accelerometer then converts it and averages the data.
    RawXdata = get_data(ADXL345_ADDR, X0_REG, X1_REG)
    GXdata = convert_G(RawXdata)
    X = .8 * X + .2 * GXdata

    # This block of code gets the byte data of Y axis from the accelerometer then converts it and averages the data.
    RawYdata = get_data(ADXL345_ADDR, Y0_REG, Y1_REG)
    GYdata = convert_G(RawYdata)
    Y = .8 * Y + .2 * GYdata

    # This block of code gets the byte data of Z axis from the accelerometer then converts it and averages the data.
    RawZdata = get_data(ADXL345_ADDR, Z0_REG, Z1_REG)
    GZdata = convert_G(RawZdata)
    Z = .8 * Z + .2 * GZdata

    print("X Data: %2.2f " % X)
    print("Y Data: %2.2f " % Y)
    print("Z Data: %2.2f " % Z)
    print("-+-+-+-+-+-+-+-+-+-+-")
    time.sleep(0.5)
```

Figure 8(c) – Results Code for Accelerometer

Remote Sensor Network

The engineering requirements for this lab project are listed below.

- LM35 used as the temperature sensor
- Configure two XBee3 modules (one as a coordinator and the other as a router)
- Set up a communications link between the router and coordinator
- Transmit temperature information from the router to the coordinator
- Display the temperature on a PC using the UART interface

The schematic is shown in Figure 9. The XBee3 module in the Transmit Circuit is configured to be a router. An LM35 is connected to the ADC input on the XBee3 module. A serial LCD is used to connect directly to the UART interface of the XBee3 module to display temperature in °C. The XBee3 module is also configured to transmit the temperature to a second XBee3 module, configured as a coordinator, in the Receive Circuit. This module is mounted on a USB dongle that is connected to the USB port on the PC.

The code is shown in Figure 10. In Figure 10(a), the Router code is provided. The LCD is initialized, the ADC reads to voltage from the LM35, voltage is converted to temperature in °C, and a message containing this temperature is displayed on the LCD and transmitted to the coordinator. Figure 10(b) provides the code for receiving the message. This message is then displayed on the PC using a terminal program (e.g., Putty).

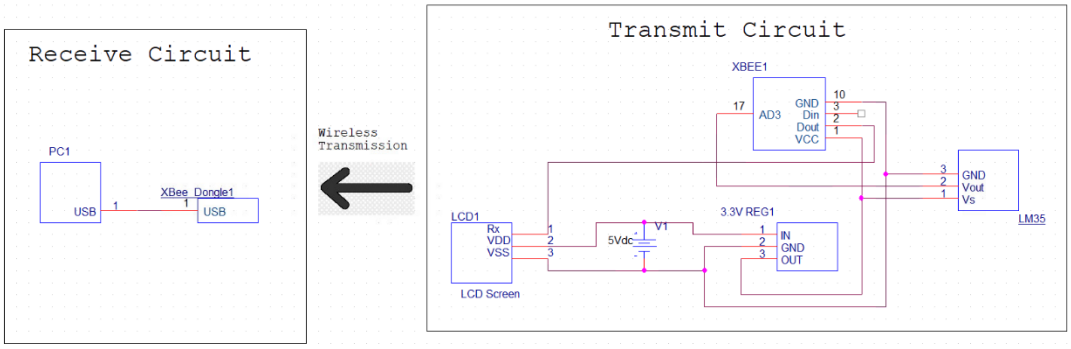

Figure 9 – Remote Sensor Network Schematic

```
1.  from machine import ADC  # imports ADC class from the machine library
2.  from sys import stdout  # imports stdout class from the sys library
3.  import time
4.  import xbee
5.  # TODO: replace with the 64-bit MAC address of your target device.
6.  COORD_ADDR = b'\x00\x13\xA2\x00\x41\xA8\x02\xC6'
7.  # List of LCD commands as an array of bytes.
8.  home = bytearray([0xFE, 0x46])  # returns the cursor to line 1 column 1
9.  line2 = bytearray([0xFE, 0x45, 0x40])  # places the cursor to line 2 column 1
10. clear = bytearray([0xFE, 0x51])  # clears the entire display
11. contrast = bytearray([0xFE, 0x52, 50])  # sets the display contrast
12. # Renames each class used in code.
13. adc = ADC('D3')  # create an ADC object for pin D3 (AD3/DIO3)
14. lcd = stdout  # stdout assigned to variable "lcd"
15. # Initialize LCD screen.
16. lcd.write(contrast)  # sets LCD to max contrast
17. lcd.write(clear)  # clears entire LCD screen
18. # Function reads LM35 output and converts to degrees C.
19. def read_temp():
20.     raw_adc = adc.read()
21.     mv_adc = (raw_adc * 1250.0) / 4095.0  # scales 12 bit A to D data to 0 -
    1250 mV
22.     temp_c = mv_adc / 10.0  # each mv represents 1 degree C
23.     return temp_c
24. avg = 0  # initialize variable for average
25. while True:
26.     lcd.write(clear)
27.     time.sleep_ms(1)
28.     avg = 0.8 * avg + 0.2 * read_temp()  # equation used for averaging temp
    data in degrees C
29.     MESSAGE = 'Temp in C: ' + str(round(avg, 1))  # string containing average
    temp rounded to 1 decimal
30.     lcd.write(MESSAGE)  # displays string message to LCD
31.     lcd.write(line2)
32.     lcd.write('Sending...')
33.     time.sleep(1)
34.     lcd.write(line2)
35.     try:  # transmits data to coordinator XBee
36.         xbee.transmit(COORD_ADDR, MESSAGE)
37.         lcd.write('Data delivered')
38.         time.sleep(1)
39.     except Exception as e:  # displays error code if failure to send data
40.         lcd.write('Transmit failure: %s' % str(e))
41.         time.sleep(2)
```

Figure 10(a) – Router Code for Remote Sensor Network

```
1. import xbee
2. print("Waiting for data...\n")
3. while True:
4.     received_msg = xbee.receive()  # check if the XBee has any message
5.     if received_msg:
6.         payload = received_msg['payload']  # get the payload from received msg
7.         print("Data from ROUTER -> %s" % payload.decode())
```

Figure 10(b) – Coordinator Code for Remote Sensor Network

Student Assessment

Table 1 provides results from a student questionnaire regarding the lab projects. The questionnaire was completed at the end of the completion for all of the projects, and two students completed the questionnaire. Thus, more data will need to be collected for statistically relevant results. Students found the overall level of difficulty to be moderate, and the approximate times for each lab project ranged from 8 to 12 hours (excluding the first lab). The most difficult lab was the accelerometer lab, where the students had to learn about the I^2C interface, how to configure the accelerometer, and how to convert raw accelerometer data into G's. Students found that the Digi MicroPython Programming Guide provided useful information regarding MicroPython syntax and functions for programming the XBee3 module.

Table 1. Student Questionnaire Results.

Questions	Responses
Approximate average time in hours to complete each lab project	• Besides the "Introduction to Python using PyCharm" Lab, approximate times ranged from 8 to 12 hours
Overall level of difficulty (e.g., easy, moderate, difficult, extremely hard)	• Moderate • Accelerometer Interface was the most difficult
Areas of difficulty	• MicroPython syntax • Accelerometer configuration • Implementation of I^2C
Prerequisite information regarding MicroPython to successfully complete the lab projects	• MicroPython syntax • Referencing the Digi MicroPython Programming Guide (for the XBee) provides support
Suggestions for improvement to help ensure student success	• Overview of hardware devices (and I^2C and UART) • Network configuration for the XBee modules • Converting raw accelerometer data into G's

Instructor Assessment

The instructor provided engineering requirements for each lab project to the students. Students provided a written report for each project, and they demonstrated their results to the instructor. These students had no background in MicroPython. However, they had embedded programming experience in C (with the PIC and PSoC 5LP) utilizing the devices listed for these projects. Using their experience in programming, they learned MicroPython readily with online resources that

included the programming guide for the XBee3 module. In general, students successfully completed each lab project within 2 weeks. Due to the ease of programming in MicroPython, it took less time for students to complete the projects as compared to utilizing embedded C. Depending upon the background and academic level of the students, it may be helpful to discuss MicroPython syntax and the hardware overview for each device.

Summary

The Digi XBee3 module is an inexpensive device suitable for embedded applications in a variety of courses, including a wireless communications systems course. MicroPython is optimized to run on a variety of microcontrollers, including the XBee3 module. The Community Edition of PyCharm is available for free and can be used as the software development environment. There are example programs provided within PyCharm, and these can be used as the basis for many lab projects.

The projects presented in this paper are only a subset of the various projects that can be implemented using the XBee3 module. Student responses to a questionnaire indicate that these projects were a great learning experience. Students successfully demonstrated various embedded programming applications in MicroPython with the XBee3 module for a wireless communications systems course. With the ease of programming using MicroPython, this provided an effective supplement to embedded C/C++ programming applications. Overall, the inexpensive XBee3 module and MicroPython can be used for many embedded applications in a variety of courses.

References

[1] S. Strom and D. Loker, "Programmable System-On-Chip (PSoC) Usage in an Engineering Technology Program," *Annual Meeting, American Society for Engineering Education,* 2016.
[2] D. Loker and S. Strom, "Programmable System-On-Chip (PSoC) Usage in Embedded Programming Courses," *Annual Meeting, American Society for Engineering Education,* 2020.
[3] S. Strom and D. Loker, "BeagleBone Black for Embedded Measurement and Control Applications," *Annual Meeting, American Society for Engineering Education,* 2018.
[4] T. Gaddis, *Starting out with Python*, Pearson Education, 2018.
[5] MicroPython.org. [Online]. Available: http://www.mircropython.org
[6] Digi.com. [Online]. Available: https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-rf-modules/xbee3-zigbee-3
[7] Digi.com. [Online]. Available: https://www.digi.com/resources/documentation/digidocs/90002219/default.htm
[8] JetBrains.com. [Online]. Available: https://www.jetbrains.com/pycharm/
[9] Digi.com. [Online]. Available: https://www.digi.com/resources/documentation/digidocs/PDFs/90002445.pdf
[10] Digi.com. [Online]. Available: https://www.digi.com/resources/documentation/digidocs/pdfs/90001543.pdf
[11] SparkFun.com. [Online]. Available: https://www.sparkfun.com/products/8276
[12] Amazon.com. [Online]. Available: https://www.amazon.com/HiLetgo-FT232RL-Converter-Adapter-Breakout/dp/B00IJXZQ7C/ref=pd_sim_147_2?_encoding=UTF8&pd_rd_i=B00IJXZQ7C&pd_rd_r=fc598101-d082-11e8-9fea-e722222b4194&pd_rd_w=cVaLw&pd_rd_wg=vbqpc&pf_rd_i=desktop-dp-sims&pf_rd_m=ATVPDKIKX0DER&pf_rd_p=18bb0b78-4200-49b9-ac91-f141d61a1780&pf_rd_r=9ZDZSQCD32VR6NW4JAM6&pf_rd_s=desktop-dp-sims&pf_rd_t=40701&psc=1&refRID=9ZDZSQCD32VR6NW4JAM6
[13] SparkFun.com. [Online]. Available: https://www.sparkfun.com/products/11697
[14] JetBrains.com. [Online]. Available: https://www.jetbrains.com/pycharm/learn/
[15] mvcc.edu. [Online]. Available: https://www2.mvcc.edu/users/faculty/jfiore/CP/labs/LaboratoryManualForComputerProgramming.pdf
[16] Analog.com. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf