**DESIGN LAB EXPERIMENT #2**

**MOTION SENSING SYSTEM IMPLEMENTATION USING RASPBERRY PI**

# Purpose

In this lab, students are required to build a motion sensing system using a Raspberry Pi single-board computer, where the sensing data are transmitted to a server through a client-server communication. By using UDP connection between a Raspberry Pi and a server, encrypted data from the Raspberry Pi are sent to the server, where the data are decrypted and displayed in a graphical format.

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

* Raspberry Pi single-board computer
* Adafruit ADXL345 triple-axis accelerometer
* Linux development environment
* I2C (Inter-Integrated Circuit) bus
* SPI (Serial Peripheral Interface) bus
* Client-server internet communication architecture

The purpose of this exercise is to introduce students with different methods of communication between a microcomputer and I/O devices using Raspberry Pi, a simple data encryption-decryption method using a symmetric key, and a client-server UDP connection.

# Component Requirements

* 1 x Raspberry Pi 3 single-board computer
* 1 x Adafruit ADXL345 triple-axis accelerometer
* 1 x Desktop (as a data storage server)

# Background

## Description of Raspberry Pi Single-Board Computer

Raspberry Pi is a series of small single-board computers developed by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. As of 2018, the third generation of Raspberry Pi was released, known as Raspberry Pi 3, will be used. Note that other types of Raspberry Pi can be used to accomplish this experiment. (e.g., Raspberry Pi 4)

A Raspberry Pi 3 is equipped with the following components:

* CPU: Broadcom BCM2837 1.2 GHz 64-bit Quad-core (ARMv8 Cortex-A53)
* GPU: Broadcom VideoCore IV
* RAM: 1GB LPDDR2 (900 MHz)
* Network: 10/100 Ethernet, 2.4 GHz 802.11n Wi-Fi, Bluetooth 4.1
* Storage: microSD card
* GPIO: 40-pin header, populated
* Ports: HDMI, 3.5mm audio jack, 4xUSB 2.0 ports, Camera Serial Interface (CSI), Display Serial Interface (DSI)

All Raspberry Pi 3 in this lab are installed with Raspbian, a variation of Linux operating system tailored for Raspberry Pi. Advantages of using Raspberry Pi in this lab as following. First, Raspberry Pi is a full-fledged microcomputer, like SANPER, Raspberry Pi also equipped with CPU, RAM and the ability to interface a variety of peripheral devices. Second, Raspberry Pi is equipped with modern technologies, such as USB ports, networking via Ethernet, Wi-Fi, Bluetooth, and running an operating system, etc. Moreover, it is easily reconfigured by re-programming, and has extensibility of adding many peripheral devices.

## Description of Adafruit ADXL345 Triple-Axis Accelerometer

The ADXL345 is a digital-output, 3-axis accelerometer whose low power consumption and built-in features make it ideal for use in a wide variety of applications. In this lab, ADXL345 serves as the sensor by providing the acceleration measurements to Raspberry Pi. Due to its small size, the ADXL345 has minimal effect on the performance of the system and of the accelerometer, and thus it is ideal for evaluation of the ADXL345 in an existing system. Note that the ADXL345 must be communicated with digitally via SPI or I2C. With SPI and I2C equipped on both ADXL345 and Raspberry Pi, the two devices can communicate easily without the help of any other devices.

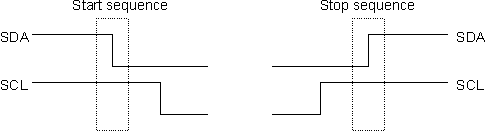
Technical Specification of ADXL345:

* 4 sensitivity levels: ±2g, ±4g, ±8g or ±16g
* occupies I2C 7-bit address 0x53
* VCC takes up to 5V input and regulates it to 3.3V

## I2C (Inter-integrated Circuit) Bus

I2C bus can be used to transfer data between master device and slave devices. I2C uses just two wires, SCL and SDA. SCL is the clock line. It is used to synchronize all data transfers over the I2C bus. SDA is the data line. It is used to specify addresses and transfer data. The SCL & SDA lines are connected to all devices on the I2C bus. Both SCL and SDA lines are "open drain" drivers. Devices on the I2C bus are either masters or slaves. The master is always the device that drives the SCL clock line. The slaves are the devices that respond to the master. A slave cannot initiate a transfer over the I2C bus, only a master device can do that. There can be, and usually are, multiple slaves on one I2C bus. It is possible to have multiple master devices on the same I2C bus, but it is unusual, thus will not be covered here. Both master and slave devices can transfer data over the I2C bus, but the master always controls the transmission.

An I2C transmission is initiated by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change while SCL remains high. The start and stop sequences mark the beginning and end of a transmission, shown as the following:



On an I2C bus, data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is then pulsed high, then low. For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer 8 bits of data. If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back ACK as logic high, then no further data will be sent, and the data transfer will be terminated by master.

All I2C addresses are either 7 bits or 10 bits. The use of 10-bit addresses is rare thus not covered in this experiment. Even though we are transmitting the 7-bit address over the bus, it is required to send 8 bits. The extra bit is used to inform the slave if the master is writing to it or reading from it. If the bit is 0, the master is writing to the slave. If the bit is 1 the master is reading from the slave. The 7-bit address occupies from bit 7 and bit 1 of the byte, and the Read/Write (R/W) bit is in the LSB (Least Significant Bit).

Example of I2C write process:

1. Send a start sequence

2. Send the I2C address of the slave with the R/W bit low (even address)

3. Send the internal register number of slave device you want to write to

4. Send the data byte

5. [Optionally, send any further data bytes]

6. Send the stop sequence

Example of I2C read from register 0x01 on device 0xC0 process:

1. Send a start sequence

2. Send 0xC0 (I2C address of the device with the R/W bit low (even address)

3. Send 0x01 (Internal address of the bearing register)

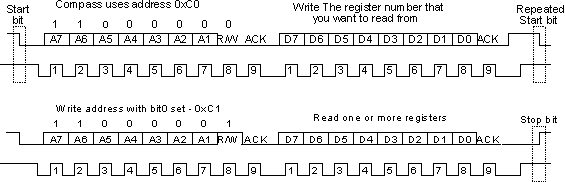
4. Send a start sequence again (repeated start)

5. Send 0xC1 (I2C address of the device with the R/W bit high (odd address)

6. Read data byte from device

7. Send the stop sequence

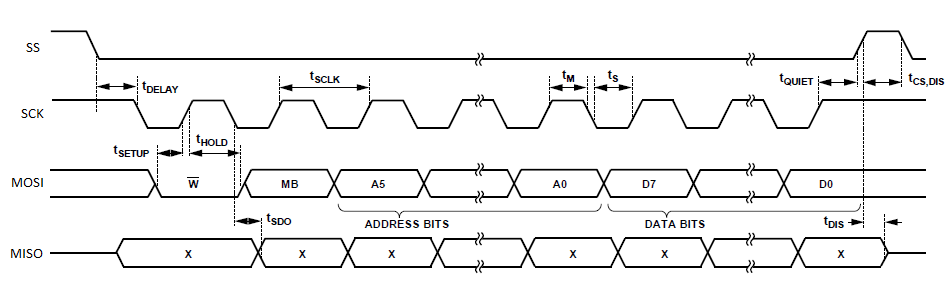
Bit sequence of reading is shown as following:



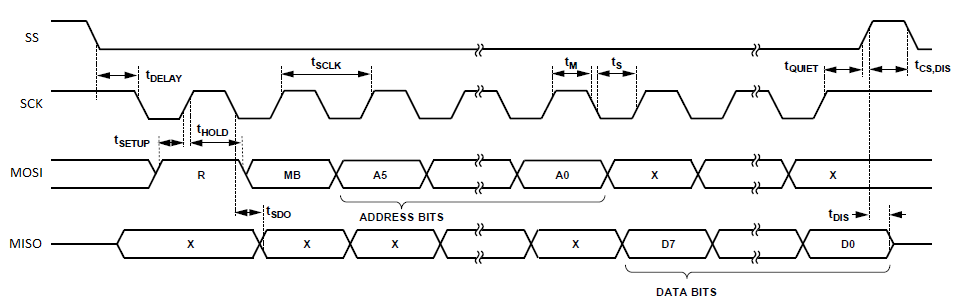
## SPI (Serial Peripheral Interface) Bus

Similar to I2C bus, SPI bus can be used to transfer data between master and slave devices. Unlike I2C, SPI uses 3 wires or 4 wires to connect between master and slave devices. In this lab, only 4-wire mode is used. These 4 wires are SCK, MOSI, MISO and SS. SCK is the clock line. It is used to synchronize all data transmission over the SPI bus. MOSI and MISO are the data lines. MOSI stands for master out slave in. It is used to transfer data (information) from master to slave. MISO stands for master in slave out. It is used to transfer data (information) from slave to master. SS stands for slave select. It is a signal generated by master device, slave device with logic 0 showing on its SS pin will be enabled.

In 4-wire mode, each transmission on SPI will take 2 bytes. When writing, the master device initiates the communication by driving the SS line of the slave to low. In the first byte, the master will clear MSB for writing, then put the internal address of the register to be written on bit 6 to bit 0. Unlike I2C bus, SPI relies on the SS line to select the slave device. Therefore, the address showing on the bus is always the internal address on the selected slave device. In the second byte, data to be written will be sent by master device.

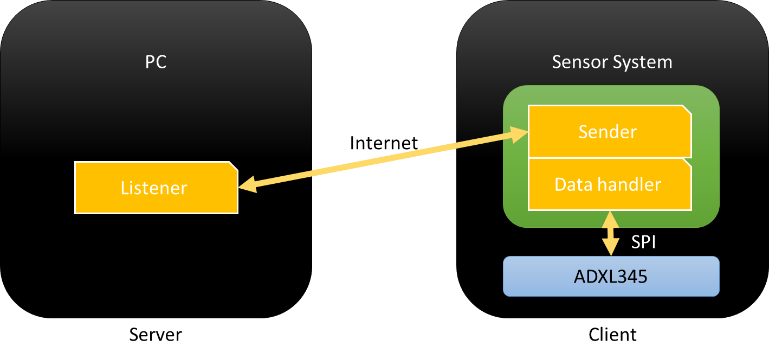


When reading, the master device initiates the transmission by driving the SS line of the slave to low. In the first byte, the master will set MSB for writing, and put the internal address of the register to be read on bit 6 to bit 0. In the second byte, the slave device will respond by sending the data from the register on to master. Shown as the following:



## Server-Client Architecture

Client-server architecture (client/server) is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power. In this lab, a server-client architecture is adopted. The Raspberry Pi as a client to collect information from the environment (accelerometer) while a PC as a server to log data from the client(s). The system architecture is shown as the following.



## Encryption of Electronic Data

The purpose of data encryption is to protect digital data confidentiality as it is stored on computer systems and transmitted using the internet or other computer networks. As mentioned in Server-Client Architecture, data collected by the client will be transmitted to the server over the internet. Data transmitted over the internet is prone to eavesdropping. Therefore, data being transmitted are often encrypted.

In this lab we use AES (Advanced Encryption Standard) algorithm to encrypt data transmitted over the internet on the application layer. To be more specific, once data been collected by the client device, it will be encrypted before transmission. While the server will decrypt received data before using (logging) it. AES is a symmetric-encryption algorithm, therefore, the keys used for encryption and decryption are the same. To avoid brutal-force deciphering, AES algorithms are usually further enforced by IV (Initialization Vector).

In the example code, both AES key and IV are specified. Please review the example code carefully, and understand the processes regarding encryption, decryption, and transmission.

# Statement of Problem

In this experiment, you will implement a real-time 3 axis motion sensing system using Raspberry Pi and ADXL345. The system should be able to retrieve data from a sensor and send them to a server.

# Preliminary Work

1. A specific Raspberry Pi OS Image is recommended to achieve this laboratory experiment. **Download Raspbian Image version 2020-02-14 (**[**Link #1**](https://downloads.raspberrypi.org/raspbian/images/raspbian-2020-02-14/)**,** [**Link #2**](http://ecasp.ece.iit.edu/ECE442/raspbian-2020-02-14/)**)** and install it to your Raspberry Pi. *(Click on the link above to download)* To learn about how to install Raspberry Pi image, refer to Reference [11] which can be found on Page 14.

Alternatively, you can use a newer image and [go to](http://archive.raspberrypi.org/debian/) <https://github.com/WiringPi/WiringPi>, where under Releases you can select the latest one and download the version compatible with the raspberry pi architecture image you installed(armhf for 32 bit and arm64 for 64 bit). Afterwards, you can go to the Downloads folder and run “sudo apt-get install ./wiringpi\*.deb” to install the newest wiringpi library.

1. Review course materials on I2C and SPI, understand how Raspberry Pi and ADXL345 communicate, and how such process is implemented in software.
2. Use information provided in *Appendix D* and *Reference 8*, draw a schematic of wiring between Raspberry Pi and Adafruit ADXL345 using I2C. Include your schematic in your report, as well as a photo of the physical wiring.
3. Use information provided in *Appendix D* and *Reference 8*, draw a schematic of wiring between Raspberry Pi and Adafruit ADXL345 using SPI. Include your schematic in your report, as well as a photo of the physical wiring.
4. Preview instruction on Raspberry Pi in Procedure A.1, understand how to compile a project on Raspberry Pi.
5. Implement the program as required in Procedure A.2. Write the program with proper comments, explain how you implemented such program in your report.
6. Implement the program as required in Procedure B.2. Write the program with proper comments, explain how you implemented such program in your report.
7. Familiar yourself with the client-server program architecture as explained in procedure B, implement the program that sending data to server. Explain how you implemented such program in your report.
8. **Assuming that you do not have an external monitor to connect to the Raspberry Pi, please follow tutorials provided on the Blackboard (Headless RPi with Access to Router or Headless RPi with no Access to Router)to use Raspberry Pi without a monitor nor a keyboard, and VNC environment remotely. We recommend you install *Raspbian Buster with desktop and recommended software* and update all software to the most recent version before implementing the lab.**
9. **Make sure you enabled both I2C and SPI interfaces on your Raspberry Pi.**
10. **If you do not have Python installed on your PC, please follow “How to install and use Anaconda on your Windows” tutorial in Appendix E.**

# Procedure

## Implementation of Real-Time Motion Sensing Device using I2C

In this part of the lab, you are required to implement a program on Raspberry Pi that retrieve and log real-time readings from ADXL345 using I2C bus.

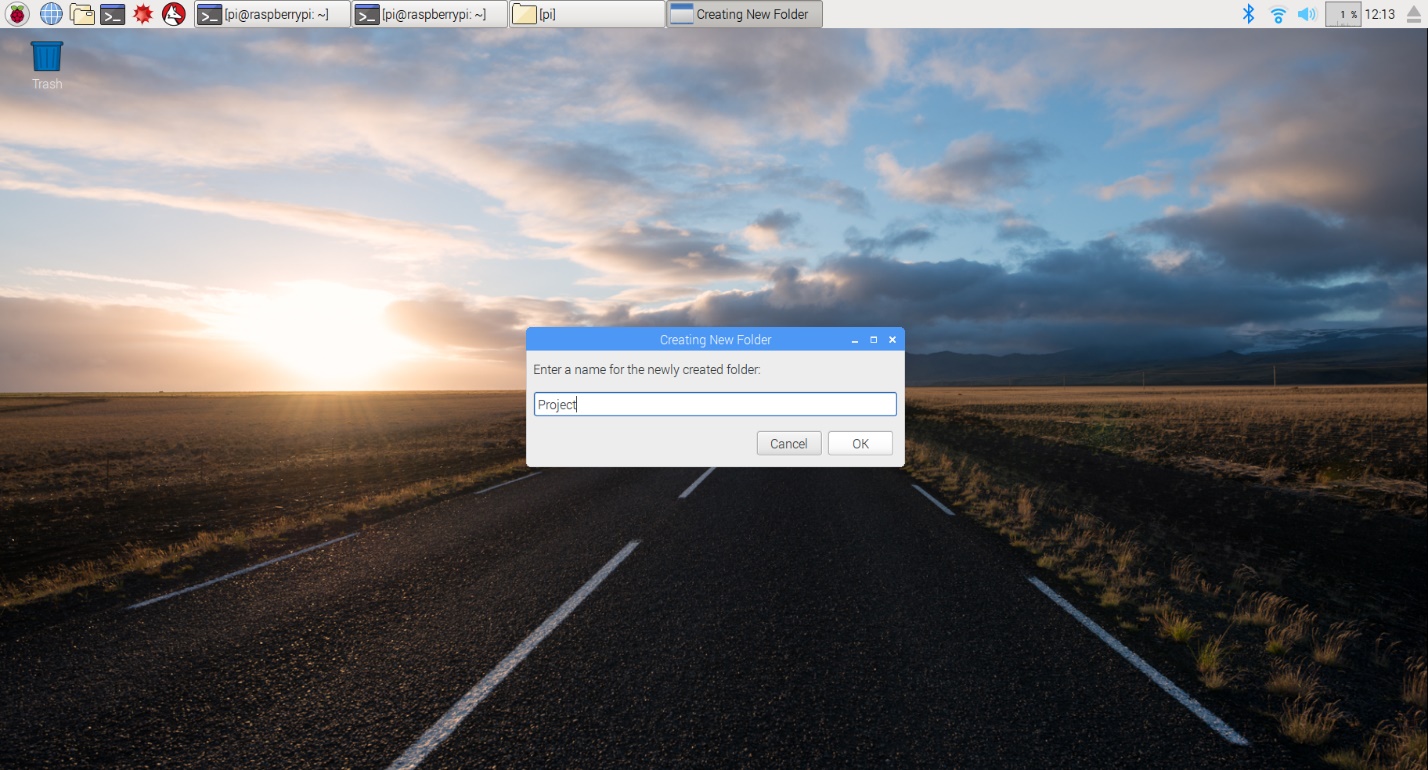
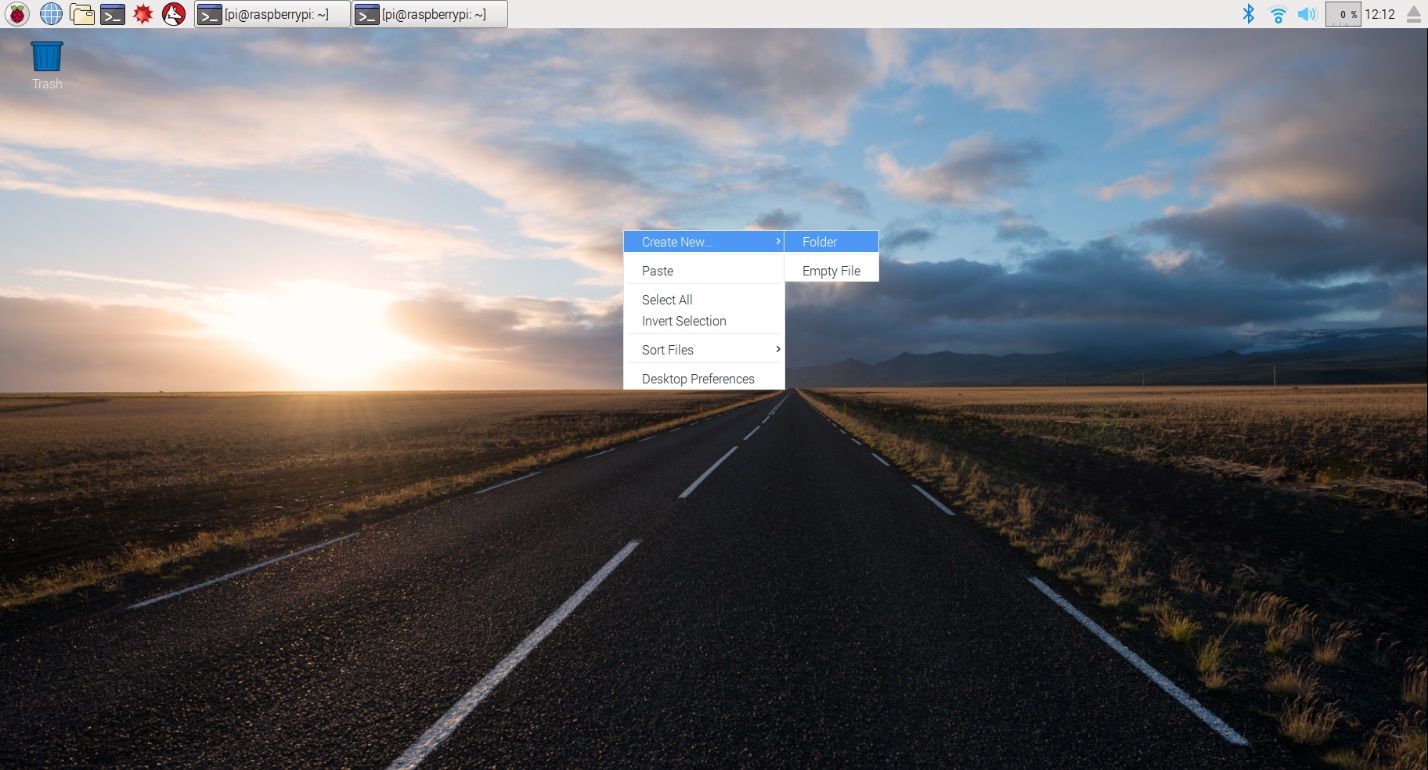
1. Remove the cover of Raspberry Pi, wire the Raspberry Pi and ADXL345 properly. So that these two devices can communicate using I2C bus.
2. Compile Example Code Provided in Appendix A on Raspberry Pi. Follow the instructions in this section, compile the example code on Raspberry Pi (Replace main.cpp with code from Appendix A). Record the result if necessary.

To compile this code in Raspberry Pi, there are a few steps needs to take:

1. Connect the mouse, keyboard via USB. You can use the touch screen on mounted on the device or connect to a monitor through HDMI cable. Once powered on, the video output device cannot be changed, thus you must choose your output before powering on.
2. Turn on the Raspberry Pi, by plug in the micro USB cable into Raspberry Pi. Wait for the system to boot up.

Once the system is boot up, you can create a working directory on desktop and put all source code including header files in it. To create a folder, simply right click on the desktop, then select *“Create New >> Folder”* then type folder name of your desire (in this instruction let’s use *“Project”* as name for working directory), as shown in following screenshots.

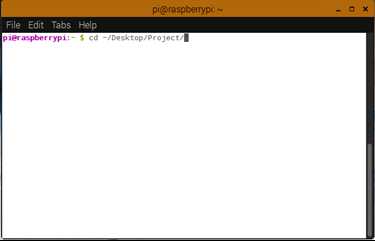
\*\*If you are using a terminal environment, create a folder using *mkdir* command. (e.g., *mkdir Project*) Other approach to this is to connect to your Raspberry Pi via SFTP/FTP program. By using this tool, you can edit your program on your PC then upload the program file to the Raspberry Pi using SFTP/FTP program. For more details, please follow the guideline specified in *Reference 10*.



1. Now you are ready to compile the code and generate an executable file. To do so, you will have to open a *“terminal”* by pressing combination of *ctrl + alt + t*. Then navigate to you working directory, in this case *“Project”* on desktop, by typing in following command in the terminal:

cd ~/Desktop/Project/

Your terminal should look like this:



Then hit *enter* to enter the directory.

1. After entering the working directory, you can compile the source code using *“g++”*. To do so simple typing the command in the terminal:

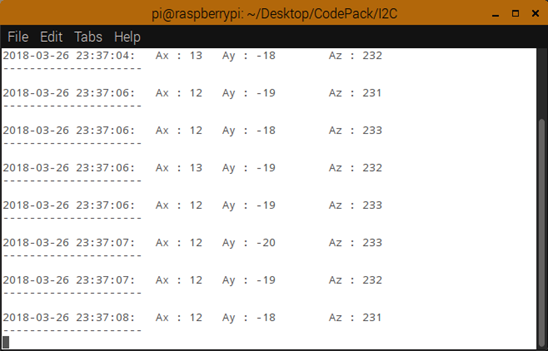
g++ -g main.cpp ADXL345.cpp -o exe.out

This command will use *“g++”* as the compiler, compile *“main.cpp”*. *“-o exe.out”* specifying that the compiler should generate an executable output named as *“exe.out”*. *“-g”* means enable debug symbols, without such flag, the program will not be able to be debugged. You can use GDB to debug your program, please refer *Reference 6*.

1. Now execute the file by typing:

./exe.out

Your output should look like this:



1. Finally, check the output on the screen as well as in the *“output.txt”*. A copy of your output should be stored in the text file. You can terminate the process forcefully by pressing *ctrl+c* at any time, however, this method should only be used when your program stops responding.
2. In this step, please modify the example code and make it store a timestamp in the **output file** as well. To be more specific, your program should store the sensor reading into an output text file, moreover, for each entry, you need to store the time as well. Take screenshots and include them in your report.

## Implementation of Real-Time Motion Sensing Device using SPI and Client Server Communication

To build a server-client application, two major parts need to be implemented. One is a data sender on the client. The other part is a listener running on the server. In this part of the lab, you are required to implement a data sender on Raspberry Pi that retrieves and log real-time data from ADXL345 sensor, then send them to a server (PC for this lab experiment). *For this part of the lab, your Raspberry Pi and your server* ***must be in the******same network*** *if you are not using a global IP address.*

1. Regarding implementing the data sender, you are required to change the wiring on Raspberry Pi so it can communicate with ADXL345 using SPI instead of I2C. An example program of client sender been provided in Appendix B. This example program meant to be compiled and run on Raspberry Pi. In the example, the program encrypts and sends predefined information to the server after encryption. Run the example code and check the predefined data shown on the server.
2. Remove the cover of Raspberry Pi, and re-wire the Raspberry Pi and ADXL345. So that these two devices can communicate using SPI bus. (Use your design from pre-lab assignment)
3. Compile example code provided in Appendix B on Raspberry Pi (review procedure A.2.3-A.2.6 if necessary). This example allows the client (Raspberry Pi) to retrieve data from ADXL345, then encrypt the data and send them to the server.
4. In this step, please modify the example code so that **it creates an output file on client which stores accelerometer reading and timestamp**. For each entry in the output file, you need to store readings from the accelerometer and the time of retrieval. Record your results. (Without the server listener ready, you may only see your data logged locally.)
5. Please make any necessary change(s) to the example program provided in Appendix B so the client can collect encrypt and send data collected from the sensor in real-time (review instruction in Procedure A.2.1-A2.4 if necessary). Please use the following command to compile your code:

g++ -g client.cpp encryption.cpp -o exe.out -lwiringPi -lcrypto

while “-lwiringPi” and “-lcrypto” are the flags for the linker. In this case, linker will link the project with wiringPi and crypto libraries locate in system environment paths. wiringPi is included in “wiringPi” package, crypto is included in “libssl-dev” package. Both packages should be included in the latest Raspbian. In case your Raspberry Pi is missing these pakcages,, you need to install both (ideally using apt-get) to compile the output file.

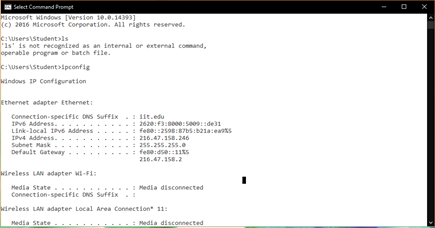
1. Regarding the server listener, it will run on the server while waiting for any incoming connections and response accordingly. Once the connection is established, data from the sensor will be received, decrypted (using [Cryptodome package](https://www.dlitz.net/software/pycryptodome/api/current/) in Python) and displayed on the server, a live plot will be generated (using [Matplotlib package](https://matplotlib.org/contents.html) in Python) on the server as well. The server listener software is provided in *Appendix B*, the server listener software needs no change. By taking following steps, you should be able start the server listener.
2. With the data sender ready, the connection can be established once the listener on the server is able to process received information properly. The listener on server is implemented in Python. Python is a script language; hence it can be executed by invoking a Python interpreter. The advantage of this script language is, the environment will be easier to be configured. In this lab, you will be using a tool that manage Python runtime as well as packages called [Anaconda](https://docs.anaconda.com/anaconda/) tool. This tool includes [Matplotlib package](https://matplotlib.org/contents.html) that we need in this lab, however Cryptodome will need to be installed using “anaconda prompt” to run “conda install -c conda-forge pycryptodomex”. See Appendix E on how to install Anaconda on your PC.
3. To execute the Python script, first copy *“server.py”* to any desired directory on server computer. Then open *“anaconda prompt”* and navigate to the folder contain your Python script. Finally type “python server.py” in Anaconda prompt. Once the server listener is running, you should see a window as following:



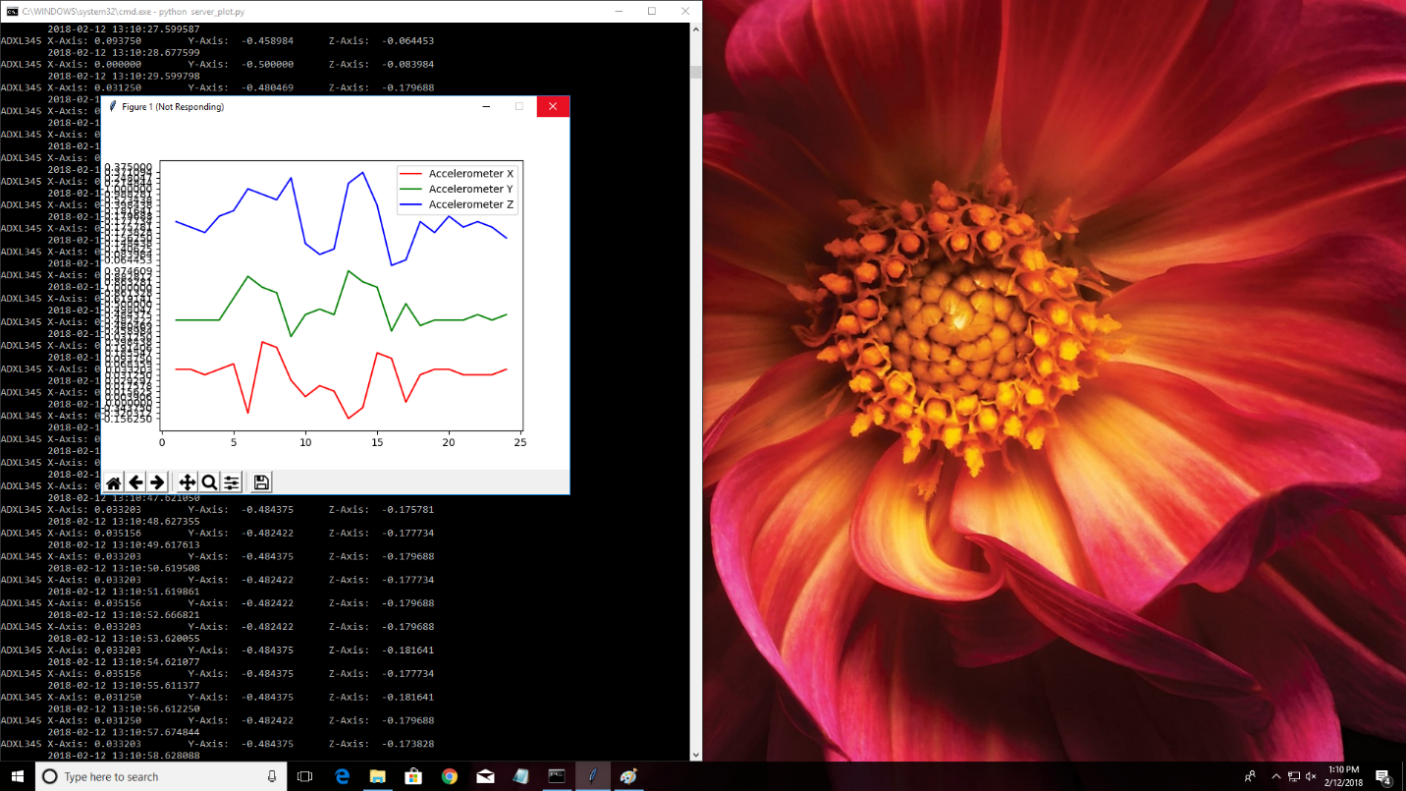
1. As explained earlier, the server listener software has been implemented, therefore, you don’t need to modify the code. However, you need to modify IP address; port number; AES key and AES IV to receive incoming connections properly. To find the IP address of your server, you need to open CMD (command prompt), a counterpart of terminal on windows. You can open it by press *“Windows\_Key+r”*, then type *“cmd”* and press *“run”*. In the CMD window type:

ipconfig

You should get the network information about your computer, shown as following:



1. Find the entry shown as *“IPv4 Address”*, that would be the address of your computer. As to the port for connection, if you are performing this lab on campus, due to the network policy of Illinois Tech, only Port 6000 is allowed for UDP connection. Update your *“server.py”* accordingly. Once your client and server are able to run properly, you should see the server showing following content on your server computer:



1. With both server listener and client sender explained, now you are required implement a program on client that collect motion sensor reading and send them to the server. Take screenshots of your screen and include in your report.
2. Record a short demonstration video of the finalized system, including your physical Raspberry Pi setup, terminal outputs in real-time of the accelerometer on the Raspberry Pi, and the server outputs displaying the received sensor data.

# Discussion

1. A fully commented code that you implemented in Procedure A.2 (5 pts)
2. Draw a flow chart and explain how the program you implemented in Procedure A.2 works, please elaborate on how Raspberry Pi get the reading from sensors. (5 pts)
3. How does a master device differentiate slave devices using I2C and SPI bus (in terms of protocol hardware implementation)? (10 pts)
4. If value 0x2F were found in register 0x31 on ADXL345, what is the behavior of the sensor regarding the register? (10 pts)
5. You may notice that the movement achieved on the physical accelerometer has a delay between the output of the Raspberry Pi output and the output of the received accelerometer data on the server side. Discuss and explain why you think this delay is there, what has caused this delay, and how it can be improved to deliver real-time data reaction on the server side. (10 pts)
6. What is the maximum sampling rate (samples/second) for the server (PC)? Note each sample comprises acceleration readings in three axes (X, Y, Z). Sampling rate on the server is bottlenecked by following factors: sampling rate of the sensor; I2C bus bandwidth; Wi-Fi bandwidth and encoding of data when being transmitted (sent as string or raw floating data). To achieve the theoretical maximum sampling rate on the server, how the system need to be configured? (e.g., sampling rate of the sensor, data encoding/decoding, I2C bus bandwidth, etc.) (10 pts)

# References

[1] A more general tutorial on C programming on Raspberry Pi

<https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/rpii/cintro.html>

[2] More insight on socket programming

<https://www.geeksforgeeks.org/socket-programming-cc>

[3] Raspberry Pi hardware documents

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

[4] Common Linux commands

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

[5] Adafuit ADXL345 accelerometer program guide

<https://learn.adafruit.com/adxl345-digital-accelerometer/programming>

[6] GDB debugger quick manual

[http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf](http://darkdust.net/files/GDB Cheat Sheet.pdf)

[7] Python 3 tutorial

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

[8] ADXL345 datasheet

<http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf>

[9] How to use Raspberry Pi without a monitor nor a keyboard and VNC environment

<https://desertbot.io/blog/headless-raspberry-pi-4-remote-desktop-vnc-setup>

[10] How to set up SSH and SFTP on a Raspberry Pi

<https://www.codedonut.com/raspberry-pi/set-ssh-sftp-raspberry-pi/>

[11] Raspberry Pi Documentation – Getting Started

<https://www.raspberrypi.com/documentation/computers/getting-started.html>

[12] ECE 442 Lab 2 Tutorial

<https://tinyurl.com/ECE442-Lab2>

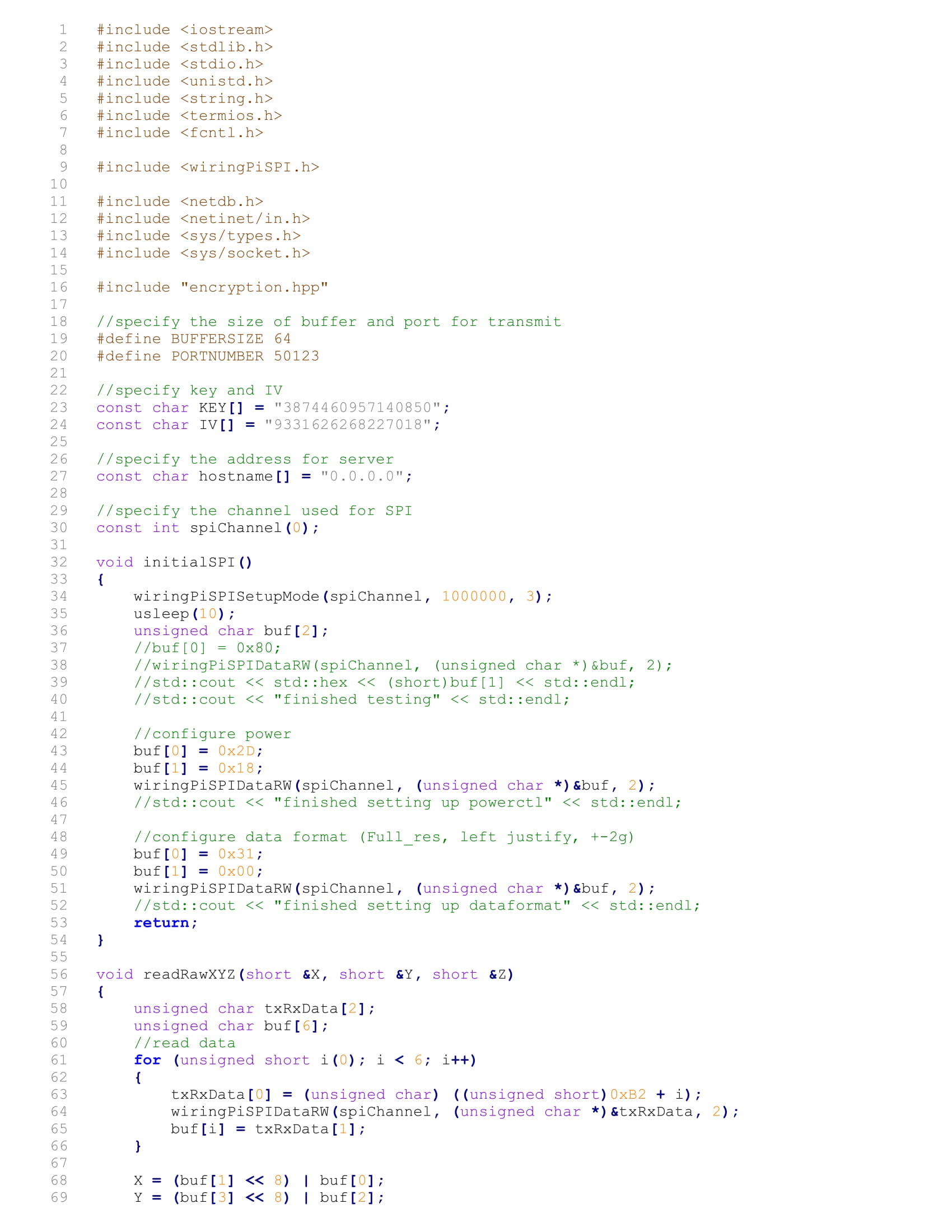
**Appendix A**

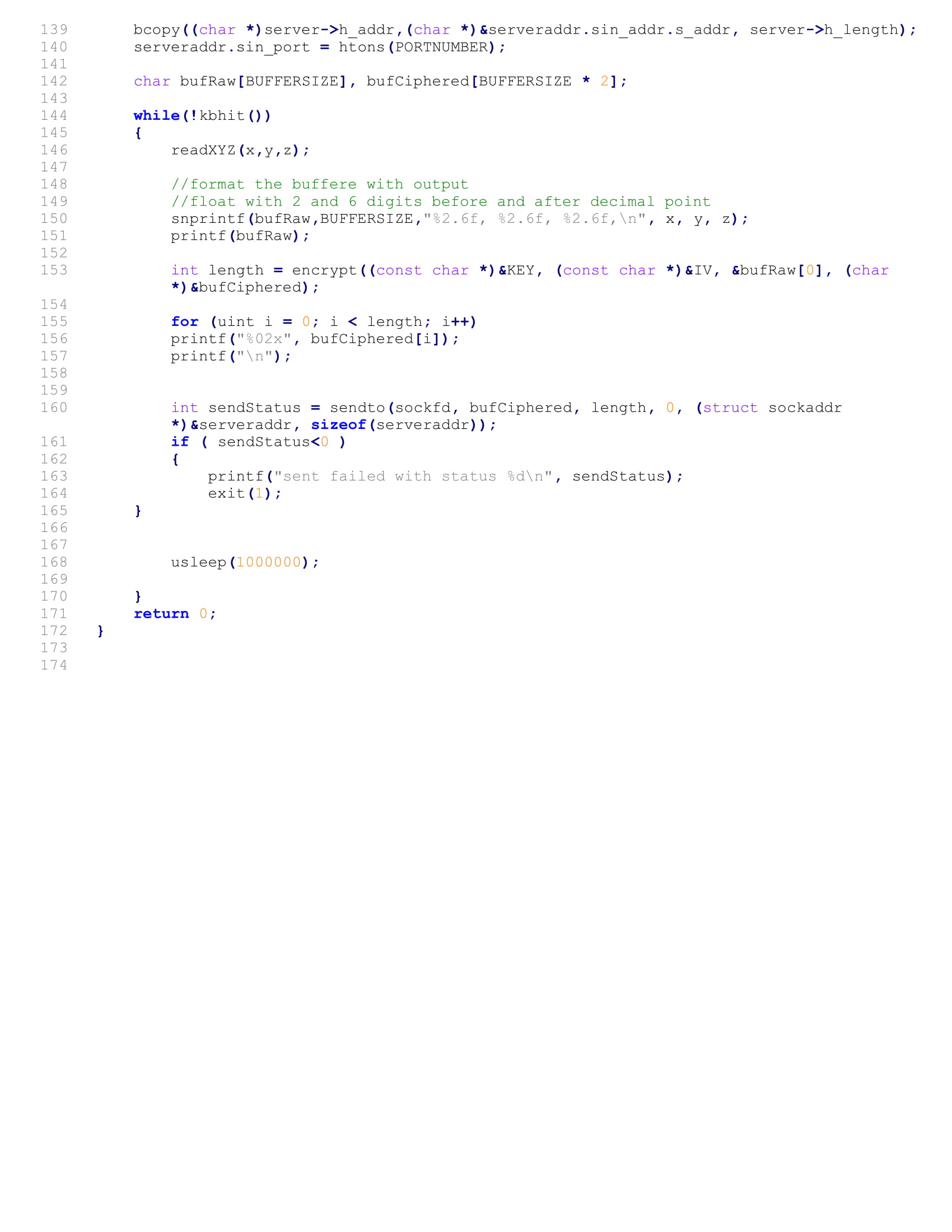
Example for sensor data retrieval and logging using I2C



**Appendix B**

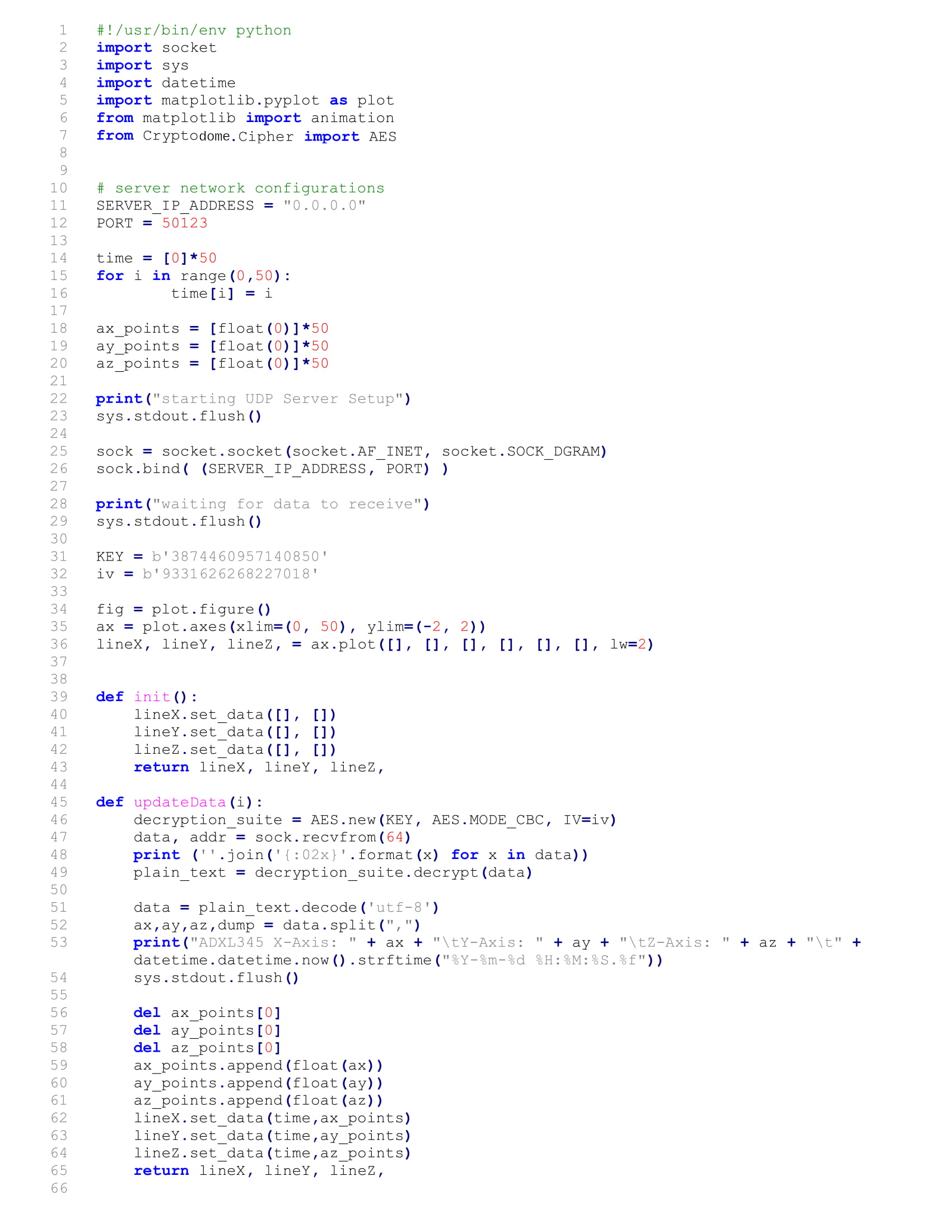
Example of the sensor data retrieval using SPI protocol and client module to send data to server





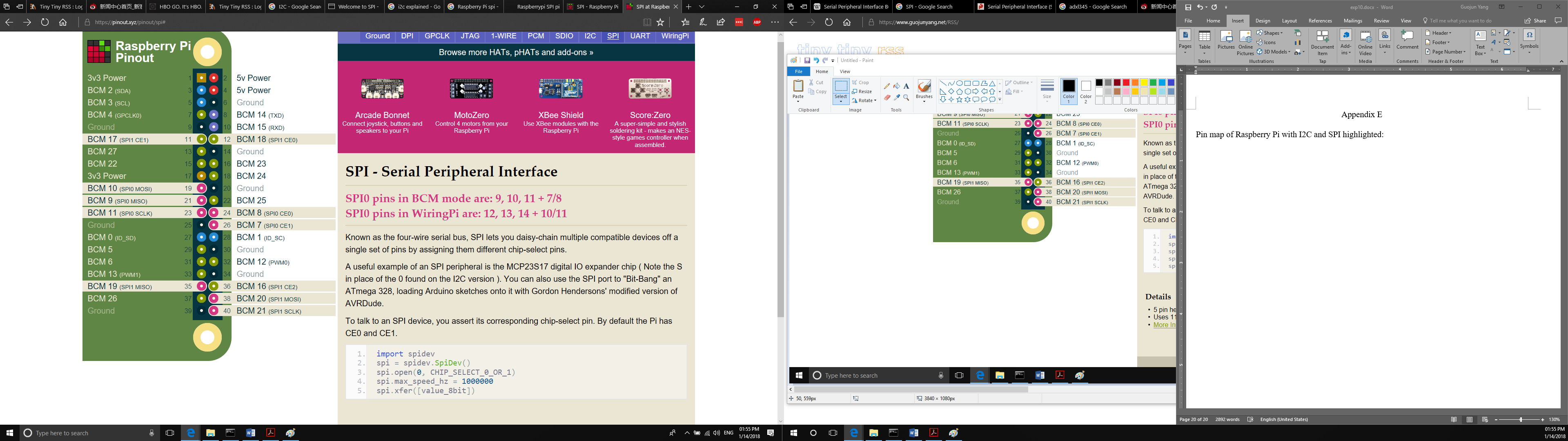
**Appendix C**

Example of the listener (Server)



**Appendix D**

Pin map of Raspberry Pi with I2C and SPI highlighted:

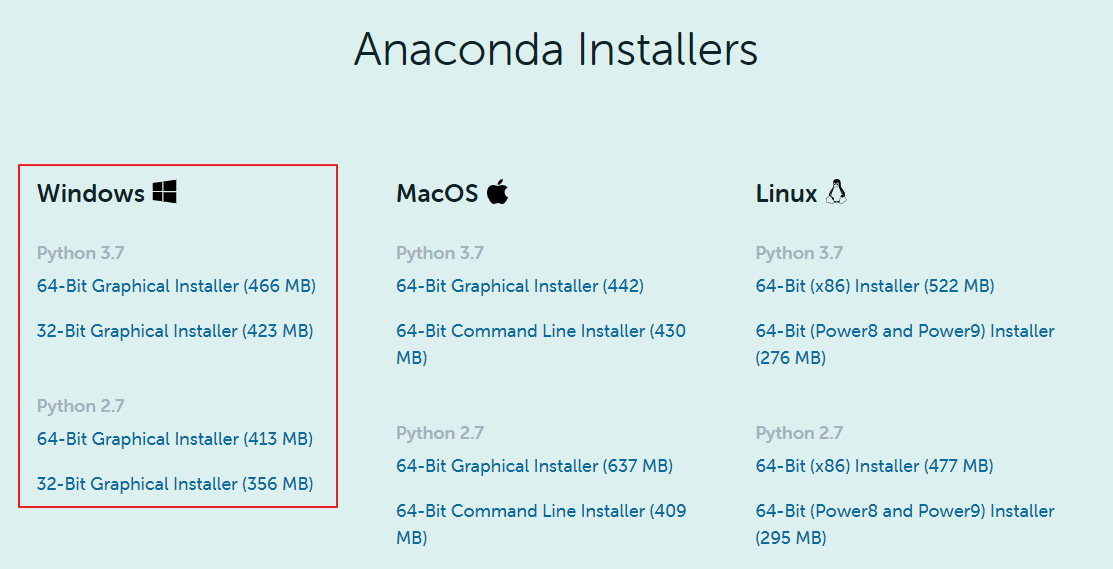


**Appendix E**

How to install and use Anaconda on your Windows

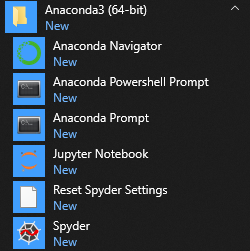
In order to perform this lab, you need to install Python programming environment set up on your PC. This lab utilizes [Anaconda](https://www.anaconda.com/) to perform Python programming which includes Matplotlib and Crypto packages for visualizing your data on a graph, and for data encryption/decryption. Anaconda is an open-source distribution of Python (including R) programming language, designed for scientific computing with simplified package environment.

1. Go to [https://www.anaconda.com/distribution/#download-section](https://www.anaconda.com/distribution/" \l "download-section) and select your current operating system. Download Python 3.x version.



2. Once finished downloading, install Anaconda to your PC. Installation may take awhile.

3. Once it has finished installation, you will notice list of Anaconda programs listed on your Start Menu like below:

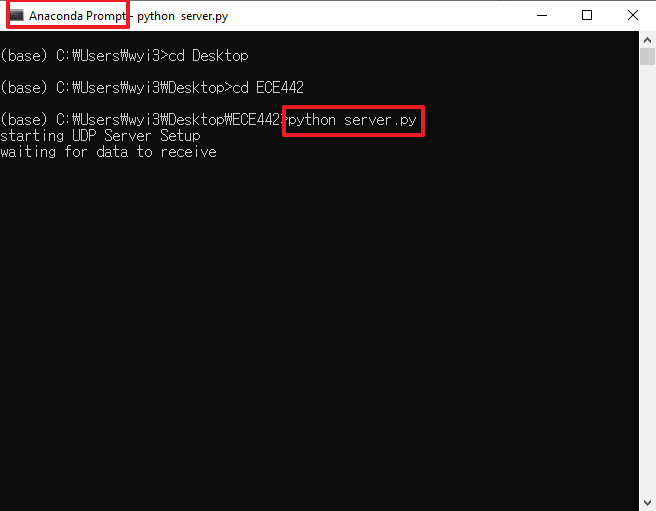


4. Launch “Anaconda Prompt” from the Start Menu, and you’ll see a window like below. Note that your window should be “Anaconda Prompt”:

5. run “conda install -c conda-forge pycryptodome” to install the Cryptodome library.

6. Navigate to the location where your “server.py” is located. Use change directory (cd) command to move (e.g. cd c:\user\wyi3\Desktop\ECE442\)

7. Execute your “server.py” by typing “python server.py”.



8. One window will pop-up to draw incoming accelerometer data, and will display similar to below screenshot:

