Experiment No. 08:

Motion Sensing System Implementation using Raspberry Pi

By: Theo Guidroz

Instructor: Dr. Jafar Saniie

TA: Xinrui Yu

ECE 441

Due Date: 05-01-2020

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature : [illegible]

I. Introduction

A. Purpose

The purpose of this lab is to familiarize students with Raspberry Pi microcomputer. Students will interact with the device and it I/O pins. They will also learn about the I^2C and SPI protocols which allow communication between the microcomputer and other devices.

B. Background

I^2C and SPI are the two protocols used to communicate data between an IC device and a microprocessor. For the I^2C protocol to work, a Serial Data Line, and a Serial Clock Line are needed. In an I^2C protocol, a master device initiates the data transfer. When the transfer is initiated, the Serial Data Line is asserted low while the Serial Clock Line is asserted high. During data transfer, the master device sends data bits, an R/W bit and an ACK bit to ensure the other devices received the data. The memory transferred from the master device is stored in internal registers of other devices. On the other hand, the SPI protocol features 4 wires: Slave Select, MOSI, MISO, and CLK. For reference, a slave device is a device with which the master communicates with.

The Raspberry Pi is a full-fledged computer system, featuring a System On Chip. Similar to the SANPER unit, a Raspberry Pi includes a CPU, RAM and ability to interface a variety of peripheral devices. In addition to those, Raspberry Pi is equipped with modern technologies, such as USB ports, networking via Ethernet, Wi-Fi, Bluetooth, and running an operating system (Debian OS).

II. Lab Procedure and Equipment List

A. Procedure

- Finish prelab tasks (Draw schematic, write code, comment your code)
- Use information provided in Appendix D and Reference 8, draw a schematic of wiring between Raspberry Pi 3 and Adafruit ADXL345 using I2C.
- Use information provided in Appendix D and Reference 8, draw a schematic of wiring between Raspberry Pi 3 and Adafruit ADXL345 using SPI.
- Implement the program as required in procedure A.2.
- Implement the program as required in procedure B.2.
- Finish discussion questions/tasks

III. Results and Analysis

A. Discussion

Schematics

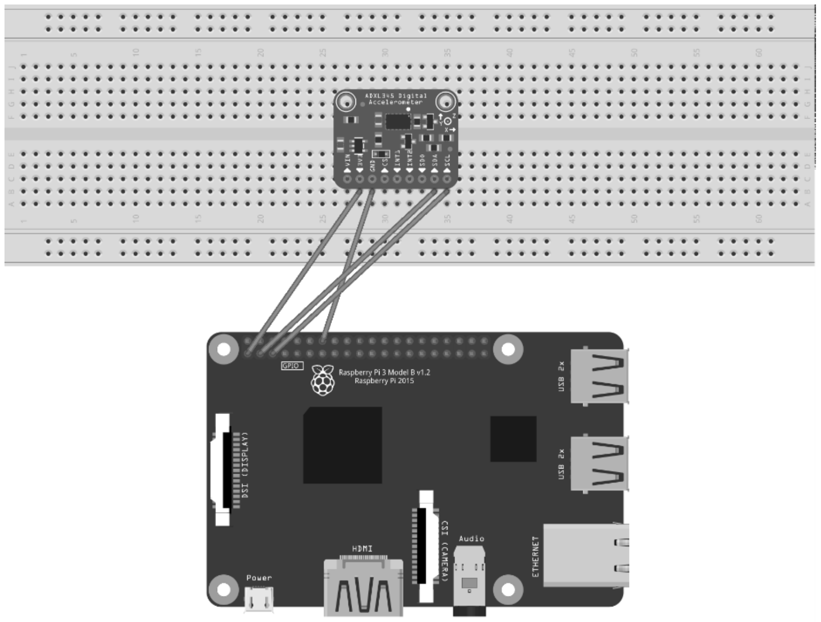


Figure 1: I2C protocol schematic

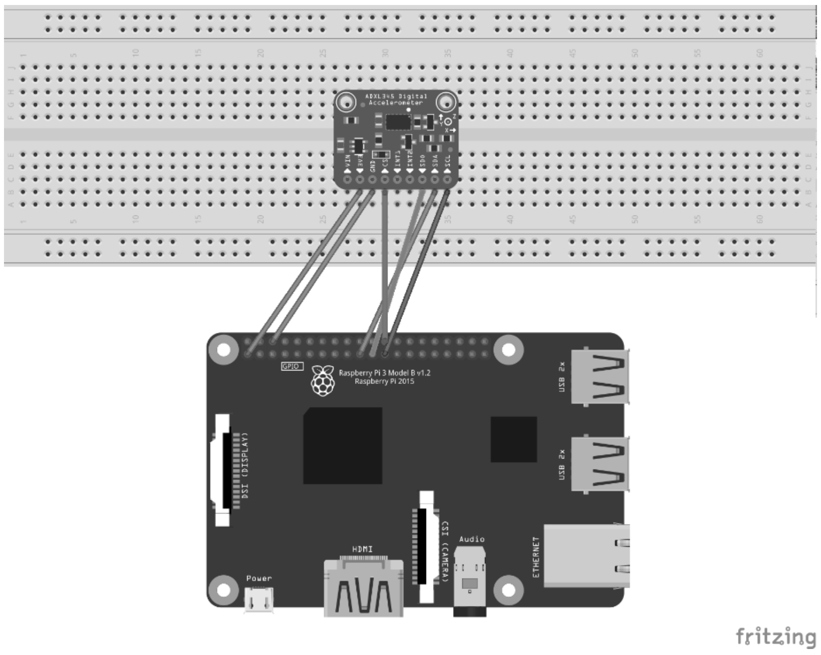


Figure 2: SPI protocol schematic

Programs

```
#include <stdio.h>
#include <signal.h>
#include <sys/time.h>

#define I2C_FILENAME "/dev/i2c-1"

void INThandler(int sig);

int main(int argc, char **argv){
    int i2c_fd = open(I2C_FILENAME, 0_RDWR);
    if(i2c_fd < 0){
        printf("Unable to open i2c control file, err=%d\n", i2c_fd);
        exit(1);
    }

    printf("Opened i2c control file, id=%d\n", i2c_fd);
    ADXL345 myAcc(i2c_fd);
    int ret = myAcc.init();
    if (ret) {
        printf("failed init ADXL345, ret=%d\n", ret);
        exit(1);
    }

    usleep(100*1000);

    signal(SIGINT, INThandler);
    short ax,ay,az;

    FILE *fp;
    fp = fopen("./output.txt","w+");

    char TimeString(128);
    timeval curTime;

    while(1){

        gettimeofday(&curTime, NULL);
        strftime(TimeString, 80, "%Y-%m-%d %H:%M:%S",
localtime(&curTime.tv_sec);
        printf(TimeString);
        printf(":   ");

        myAcc.readXYZ(ax,ay,az);

        printf("AX: %hi \t Ay: %hi \t Az: %hi \n", ax,ay,az);
        printf("-----------------------\n");
        fprintf(fp,TimeString);
        fprintf(fp,":   ");
        fprintf(fp,"AX: %hi \t Ay: %hi \t Az: %hi \n", ax,ay,az);
        fprintf(fp,"-----------------------\n");
        if (getchat() == 'q') break;
    }

    fclose(fp);
    return 0;
}

void INThandler(int sig){
    signal(sig, SIG_IGN);
    exit(0);
}
```

Figure 3: Appendix A

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <termios.h>
#include <fcntl.h>

#include <wiringPiSPI.h>

#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#include "encryption.hpp"

// specify the size of buffer and port for transmit
#define BUFFERSIZE 64
#define PORTNUMBER 50123

// specify key and TV
const char KEY[] = "3874460957140850";
const char IV[] = "9331626268227018";

// specify the address for server
const char hostname[] = "0.0.0.0";

// specify the channel used for SPI
const int spiChannel(0);
void initialSPI()

{

    wiringPiSPISetupMode(spiChannel, 1000000, 3);

    usleep(10);

    unsigned char buf[2];

    //buf[0] = 0x80;

    //wiringPiSPIDataRW(spiChannel, (unsigned char *)&buf, 2);

    //std::cout << std::hex << (short)buf[1] << std::endl;

    //std::cout << "finished testing" << std::endl;



    // configure power

    buf[0] = 0x2D;

    buf[1] = 0x18;
    wiringPiSPIDataRW(spiChannel, (unsigned char *)&buf, 2);
    //std::cout << "finished setting up powerct1" << std::endl;

    // configure data format (Full_res, left justify, +-2g)
    buf[0] = 0x31;
    buf[1] = 0x00;
    wiringPiSPIDataRW(spiChannel, (unsigned char *)&buf, 2);
    //std::cout << "finished setting up data format" << std::endl;
    return;
}

void readRawXYZ(short &X, short &Y, short &Z)
{
    unsigned char txRxData[2];
    unsigned char buf[6];
    // read data
    for (unsigned short i(0); i < 6; i++)
    {
        txRxData[0] = (unsigned char) ((unsigned short)0xB2 + i);
        wiringPiSPIDataRW(spiChannel, (unsigned char *)&txRxData, 2);
        buf[i] = txRxData[1];
    }

    X = (buf[1] << 8) | buf[0];
    Y = (buf[3] << 8) | buf[2];
    Z = (buf[5] << 8) | buf[4];

    return;
}
```

```
void readXYZ(float &X, float &Y, float &Z, const short &scale = 2)
{
    short x_raw, y_raw, z_raw;
    readRawXYZ(x_raw, y_raw, z_raw);
    X = (float) x_raw / 1024 * scale;
    Y = (float) y_raw / 1024 * scale;
    Z = (float) z_raw / 1024 * scale;

    return;
}

// return true when keyboard been pressed, false otherwise
bool kbhit(void)
{
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if (ch != EOF)
    {
        ungetc(ch, stdin);
        return true;
    }

    return false;
}

int main()
{
    initialSPI();
    float x, y, z;

    // verify the socket
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0)
    {
        printf("ERROR opening socket\n");
        exit(1);
    }

    // verify host name/address
    struct hostent *server = gethostbyname(hostname);
    if (server == NULL)
    {
        printf("ERROR, no such host as %s\n", hostname);
        exit(1);
    }

    // Build the server internet address

    struct sockaddr_in serveraddr;

    bzero((char *) &serveraddr, sizeof(serveraddr));

    serveraddr.sin_family = AF_INET;

    bcopy((char *)server->h_addr, (char *)&serveraddr.sin_addr.s_addr,
server->h_length);

    serveraddr.sin_port = htons(PORTNUMBER);


    char bufRaw[BUFFERSIZE], bufCiphered[BUFFERSIZE * 2];


    while(!kbhit())

    {

        readXYZ(x, y, z);


        // format the buffer with output

        // float with 2 and 6 digits before and after decimal point

        snprintf(bufRaw, BUFFERSIZE, "%2.6f, %2.6f, %2.6f,\n", x, y, z);

        printf(bufRaw);


        int length = encrypt((const char *)&KEY, (const char *)&IV,
&bufRaw[0], (char *)&bufCiphered);


        for(uint i = 0; i < length; i++)

            printf("%02x", bufCiphered[i]);

        printf("\n");


        int sendStatus = sendto(sockfd, bufCiphered, length, 0, (struct
sockaddr *)&serveraddr, sizeof(serveraddr));

        if(sendStatus < 0)

        {

            printf("Sent failed with status %d\n", sendStatus);

            exit(1);
        }


        usleep(1000000);
    }


    return 0;
}
```

Figure 4: Appendix B

```
#!/usr/bin/env python

import socket

import sys

import datetime

import matplotlib.pyplot as plot

from matplotlib import animation

from Crypto.Cipher import AES


# server network configurations

SERVER_IP_ADDRESS = "0.0.0.0"

PORT = 50123


time = [0]*50

for i in range(0,50):

    time[i] = i


ax_points = [float(0)]*50

ay_points = [float(0)]*50

az_points = [float(0)]*50

print("starting UDP Server Setup")

sys.stdout.flush()


sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind( (SERVER_IP_ADDRESS, PORT) )


print("waiting for data to receive")

sys.stdout.flush()
```

```
KEY = b'3874460957140850'

iv = b'9331626268227018'


fig = plot.figure()

ax = plot.axes(xlim=(0, 50), ylim=(-2, 2))

lineX, lineY, lineZ, = ax.plot([], [], [], [], [], [], lw=2)


def init():

    lineX.set_data([], [])

    lineY.set_data([], [])

    lineZ.set_data([], [])

    return lineX, lineY, lineZ,


def updateData(i):

    decryptionisuite = AES.new(KEY, AES.MODE_CBC, IV=iv)

    data, addr = sock.recvfrom(64)

    print (''.join('{:02x}'.format(x) for x in data))

    plain_text = decryption_suite.decrypt(data)

    data = plainitext.deccde('utf-8')

    ax,ay,az,dump = data.split(",")


    print("ADXL345 X-Axis: " + ax + "\tY-Axis: " +  ay +  "\tZ-Axis: "

            + az + "\t" + datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S.%f"))


    sys.stdout.flush();


    del ax_points[0]

    del ay_points[0]
    del az_points[0]

    ax_points.append(float(ax))

    ay_points.append(float(ay))

    az_points.append(float(az))

    lineX.set_data(time,ax_points)

    lineY.set_data(time,ay_points)

    lineZ.set_data(time,az_points)

    return lineX, lineY, lineZ,


anim = animation.FuncAnimation(fig, updateData, init_func = init,

        frames = 200, interval=20, blit=True)

plot.show()
```

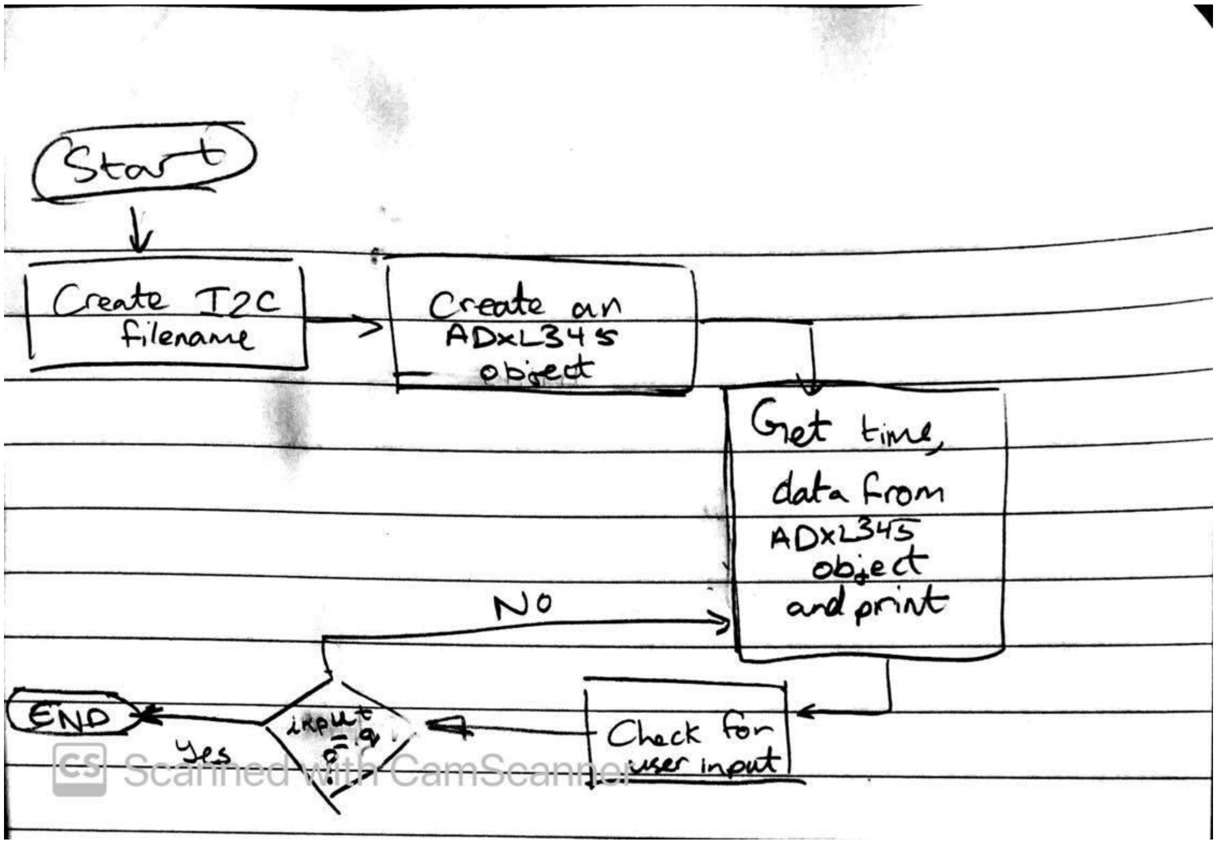
Figure 5: Appendix C

Questions

1. **A fully commented code that you implemented in Procedure A.2.**

   See figure 1.

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.**



The program first starts by creating an I2C filename. The value is returned and stored in int i2c_fd and is positive if successful. The file is used to store the data from the ADXL345. The program checks for the value of i2c_fd and recognizes an error that happened if the value returned is less than 0. Next, the program declares an instance of the ADXL345 class and assigns it to int "ret". If the device was not able to be selected, the value of "ret" is -1, otherwise, it is 0. 3 variables are declared and stored by the Raspberry Pi from the ADXL345. The next step is to open a file called "output.txt" to store the values of the chip.
A character array is declared to store the time reading. The program runs an infinite loop, reading from the slave device and the infinite loop is broken if the user enters the character "q".

3. **How does a master device differentiate slave devices using I2C and SPI bus (in terms of protocol hardware implementation)?**

   A master device uses separate pins to differentiate slave devices. For I^2C, the Raspberry PI uses the GPIO pins 0-3 whereas, for the SPI bus, it uses the GPIO pins 7-11 and 16-21 to differentiate between slave devices.

4. **If value 0x0A were found in register 0x31 on ADXL345, what is the behavior of the sensor regarding the register?**

   D7-D15 would all be 0x0 meaning that the SELF_TEST bit is 0, which disables the self-test force. It sets the device to 4-wire mode and the INT_INVERT bit interrupts are logic high. Since D3 would have a value of 1, the device would be in full resolution mode and the output resolution would increase. The justify bit is also, meaning right-justified mode with sign extension. The range bit would be set to 10 meaning the g range would be -/+ 8.

5. **What's the difference between terminal on Raspberry Pi and TUTOR on SANPER? What are the advantages and disadvantages each of them has?**

   The TUTOR's commands serve to interact solely with the 68000 microcomputers. Its main advantage is that it is mostly used for educational purposes since the user can easily interact with the hardware and see information such as the value of registers. However, it is limited to this purpose and not much can be done aside from this purpose. On the other hand, the Raspberry Pi terminal is full-fledged and has an extensive amount of commands that can be used for different purposes. It looks very similar to the command prompt on windows since it can perform tasks such as creating and modifying text files and change directors. The main advantage of the Raspberry Pi's terminal is that it is very versatile, and a lot can be done using it. The user can even compile and run programs without leaving the terminal. It can also be used to interact with the numerous I/O ports available on the microprocessor such as the USB and the HDMI port. However, unlike the SANPER, this terminal cannot be used to study the hardware of the microprocessor. The Raspberry Pi uses an ARM processor, and the user cannot directly see how the registers change when commands are run in the terminal. Changing the values of registers and setting exceptions cannot be done easily.

IV. Conclusions

During this lab, students became familiar with the hardware of Raspberry Pi, and the SPI and I^2C bus protocol. Students also learned how to implement programs on this microprocessor.

References

[1] Experiment 7 Lab Manual