Communicate with ADXL345 accelerometer sensor via SPI protocol

Siong Moua 11/28/2021 ECE 522

Background:
In this lab we will learn to use the triple axis ADXL345 accelerometer sensor from Analog Devices to measure acceleration. Note that the ADXL345 can be communicated via I^2C or SPI protocol. Though we will be using SPI in this lab. One form of acceleration measurement is in G's. The earth have a gravitational pull of 1 G, which mean it have an acceleration of 9.8 m/s^2. It is highly recommended that the student read the article [1] from Analog Devices in the reference section of this document for an understanding of the SPI protocol. The student is also required to read the datasheet of the ADXL345, as it will help improve their confidence, research, and datasheet reading skills.

*(**Note:** The content of the provided material for this lab is **not achievable** without reading through the ADXL345 datasheet. Now think about how **powerful** the datasheet is.)*

Objective:
Learn to communicate PIC24FV16KM202 with the ADXL345 accelerometer using SPI protocol to extract raw data. Learn to configurate the ADXL345 to interrupt PIC24 on activity and inactivity detection via an external interrupt pin of the PIC24.

Materials:
1. PIC24FV16KM202 MCU
2. ADXL345 breakout board
   Link to purchase:
   https://www.amazon.com/dp/B01DLG4OU6?psc=1&ref=ppx_yo2_dt_b_product_details
3. RN42 Bluetooth module

Procedure
1. Read and understand how SPI protocol works. This is presented in Appendix A. Read as many times as it takes for you to understand. Also please go to the link for the reference [1] in the reference section and read to understand the material thoroughly.
2. Make sure you fully understand how SPI protocol works before continuing.
3. Study the given code provided in Appendix B. Please look at Figure 1 for the logic flow of the provided program.
4. Read the data sheet of the ADXL345
   Note: *You should download the datasheet onto your computer and highlight key information as you read. Remember to save it so that the highlighted information is not lost. This will make you focus more as you read and easier to look for information later.*
   Link to ADXL345 datasheet:
   https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

5. Build the circuit as shown below in Figure 2 below.
6. Create a new project in CCS C and use the provided code in Appendix B.
   Note: The code is working properly. If your system is not showing what is as shown in Figure 3 and 4 below, then something is wrong with your setup and **NOT** the given code.

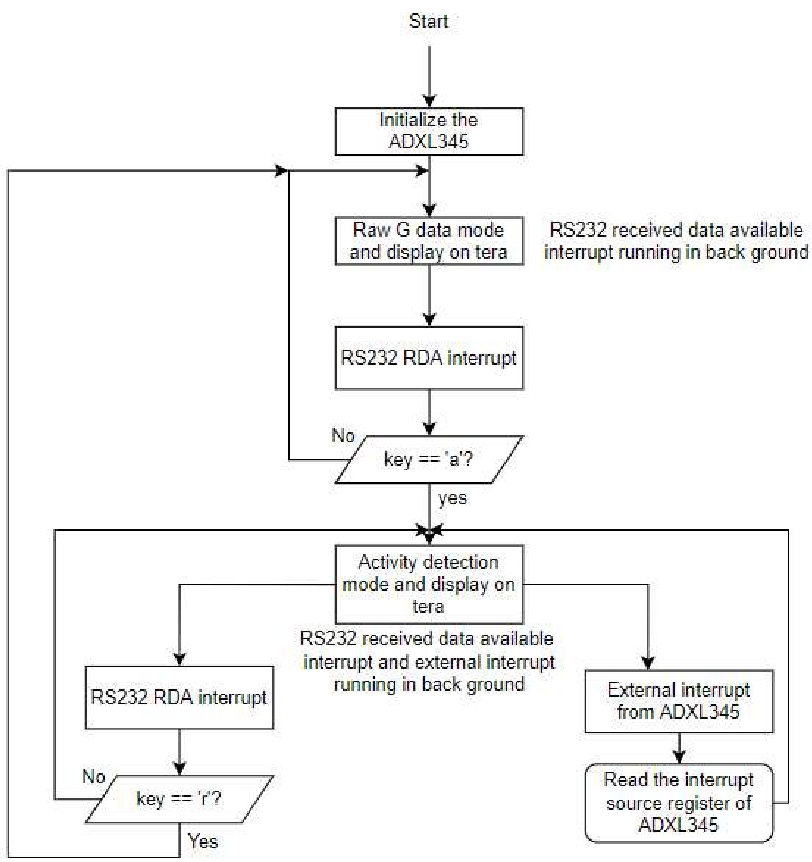

Figure 1: Program logic flow diagram

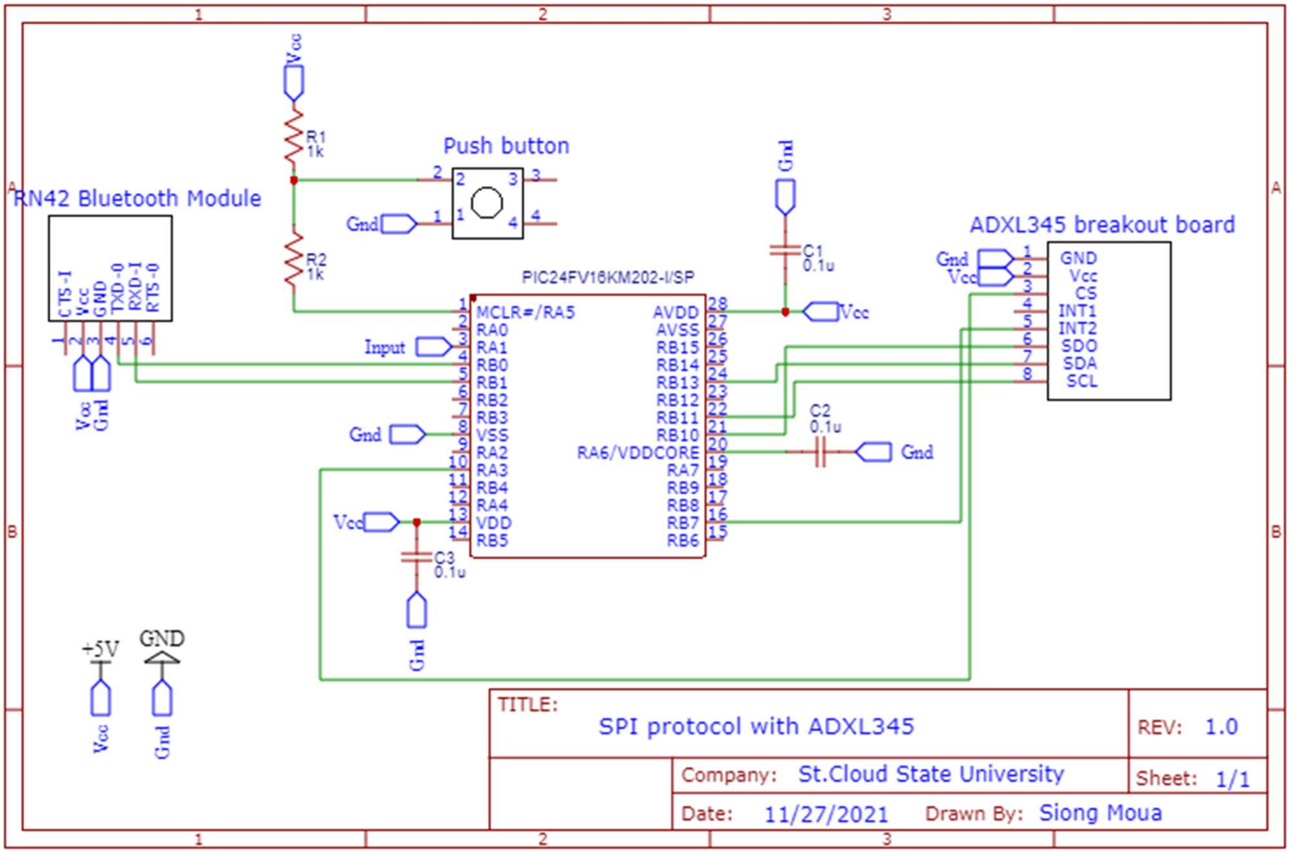

Figure 2: Schematic of the bread board layout.

What you should expect to see

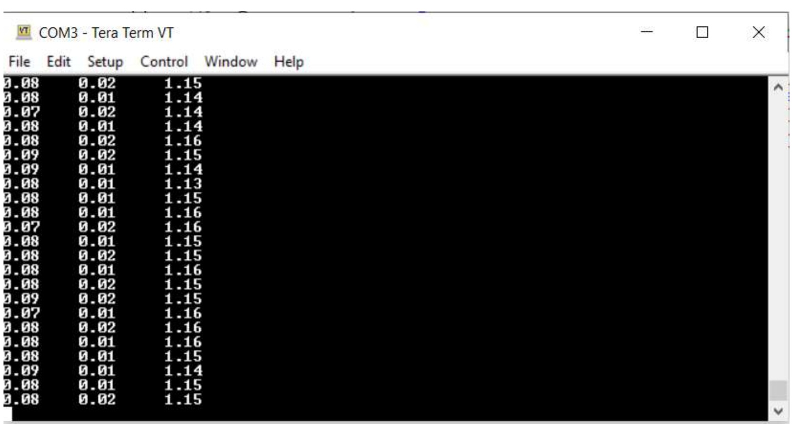

Figure 3: Program in raw G data mode.

What is being displayed in Figure 3 is X, Y, and Z G forces. Notice that the Z axis have some offset error. Ideally Z must be 1 G (assuming that the sensor is sit flat on a well leveled table), but this can be fixed via the offset register of the ADXL345. The student must read the datasheet on how to do so. Otherwise, it can also be fixed in the code directly by subtracting the offset.

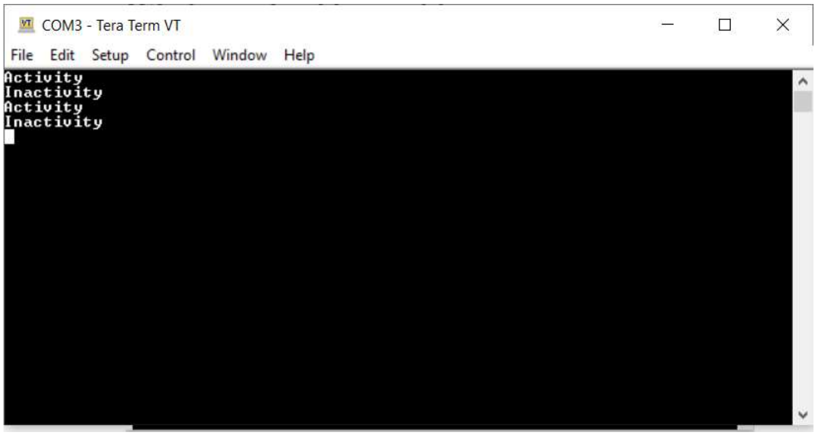

Figure 4: Program in activity detection mode.

Level 1 (70%): Be able to re-create the setup and what is being provided in the given code. You must be able to explain everything within the provided code upon questioning from TA or the Professor.

Level 2 (90%): Now let's really get you on reading the datasheet of the ADXL345 sensor. You must be able to configure ADXL345 to sample at 100 Hz and able to interrupt PIC24 whenever a new data is ready to be read in from ADXL345. These must be collected in PIC24 and sent to a C# GUI application and display accordingly.

Level 3 (100%): Read the datasheet and be able to configure ADXL345 to detect single and double tap, and ADXL345 must also interrupt PIC24 and display which (single or double tap) is being detected on a C# GUI application.

Appendix A

Quick summary and recall of how the SPI protocol works

SPI stands for **s**erial **p**eripheral **i**nterface. SPI is a single master synchronous full duplex serial communication protocol. This mean that it is a clocked, and data can transmit and received at the same time. The SPI protocol allow only 1 master, but multiple slaves. This allow SPI to have higher clock speed as opposed to I^2C protocol.

SPI operate in four different modes depending on the combination of clock polarity (CPOL) and clock phase (CPHA). This is show in Figure 5 below.

CPOL is what determines the polarity of the clock signal during the idle state to a start transmission state. [1]. The idle state is when the clock is not ticking.
CPHA is what determines when (with respect to the clock) data are sampled and shifted out/in [1].

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Figure 5: SPI 4 mode of operation [1].

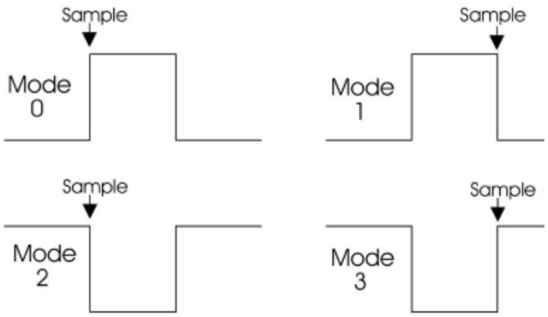

Figure 6: Visual representation of the four different modes [2].

The master device is the one that generates the clock signal. The master must also select the corresponding slave device by controlling its SS (slave select) pin. This SS pin is usually active low, this mean that a 0V will mean **select** and a 5V will mean **not select**.
It will take 8 clock cycles for a full transfer to be complete. During each clock ticks, a data bit is being shifted out/in from the master, while the slave device will shift in/out the data bit.

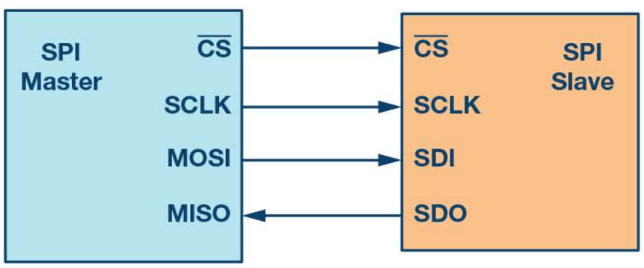

Figure 7: General data control direction of a 4-wire SPI configuration [1].

The figure above shows the general 4-wire pin connection for SPI protocol between a master and slave. Notice the direction of the arrow for each pin, this represent the data direction.

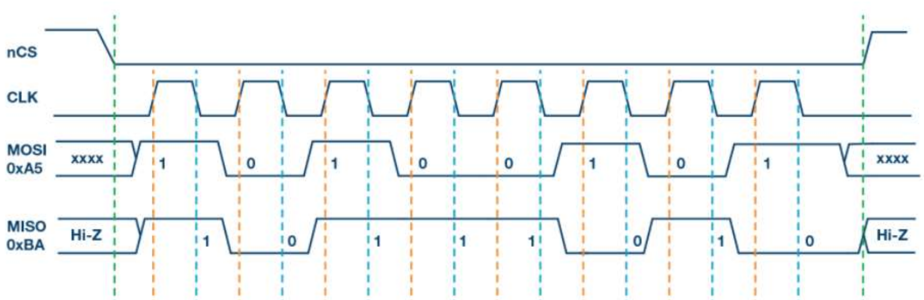

Figure 8: Waveform for SPI Mode 0. [1]
This shows mode 0 of the SPI protocol. Mode 0 implies CPOL = 0 and CPHA = 0.
Notice that the clock phase (CPHA) is 0, which implies that the idle state of the clock is low. In the figure above, the **orange** line is when the input is being sampled, and the **blue** line is when data is being shifted out/in. Since CPHA = 0 this mean that data is being shifted on the falling edge of the clock and mean that data is sampled at the rising edge. For more example and information please refer the reference source [1].

Note: To communicate with the ADXL345, we must use SPI in mode 3. One might ask how did I know? Please check out the data sheet page 16 (bottom left).

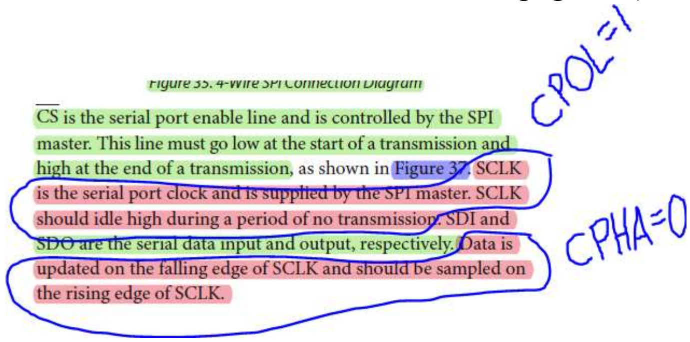

Figure 35. 4-Wire SPI Connection Diagram

CPOL=1

$\overline{CS}$ is the serial port enable line and is controlled by the SPI master. This line must go low at the start of a transmission and high at the end of a transmission, as shown in Figure 37. SCLK is the serial port clock and is supplied by the SPI master. SCLK should idle high during a period of no transmission. SDI and SDO are the serial data input and output, respectively. Data is updated on the falling edge of SCLK and should be sampled on the rising edge of SCLK.

CPHA=0

Figure 9: This is why we must use SPI in mode 3. This is extracted directly from the datasheet of page 16 at the bottom left.

Recall from Figure 5 above, mode 3 is when CPOL = 1 and CPHA = 0. Therefore, SPI must operate in mode 3 to communicate with the ADXL345 sensor properly.

Appendix B
CCS C code

```
/*
This code works well for SPI communication with the ADXL345 accelerometer from Analog Devices.
It reads the three axis (X, Y, and Z) of the ADXL345 in burst mode. This will help improve speed so
that an address isn't needed for registers. This program will also allow the detection of activity and inactivity.

..........................................................................
How to use this program

1. Build the circuit as provided
2. Download the code to PIC24FV16KM202
3. Open a tera terminal session
4. Gently moves your breadboard (where ADXL345 is configured)
5. Observe the activity/inactivity display on tera terminal
6. Type in 'r', and program will go into raw data mode
7. Observe the measured G force on all three axis displaying continuously
8. Type in 'a', and observe the program enter activity/inactivity detection mode again
..........................................................................

For more information regarding the ADXL345 please read the datasheet. The datasheet is very easy
to read and understand. It is highly recommended that one must slowly read and take notes and or even download the
datasheet and highlight key informations as I did.

*/
#include <24FV16KM202.h>
#device ICSP=3
#use delay(internal=32000000)

#FUSES NOWDT                //No Watch Dog Timer
#FUSES CKSFSM               //Clock Switching is enabled, fail Safe clock monitor is enabled
#FUSES NOBROWNOUT           //No brownout reset
#FUSES BORV_LOW             //Brown-out Reset set to lowest voltage


#USE RS232(UART2, BAUD = 115200, PARITY = N, BITS = 8, STOP = 1, TIMEOUT = 500))
```

```
#use spi(MASTER, SPI1, BAUD=1000000, MODE=3, BITS=8, stream = my_ADXL345)


#define SPI_XFER_my_ADXL345(x)   spi_xfer(my_ADXL345, x)

#define my_9250_CS PIN_A3    //Chip select pin

//*************************
#define READ 0b10000000
#define WRITE 0b00000000

#define single_RW 0b00000000
#define mult_RW 0b01000000
//************************

//------------------------Register names

#define RE_DEVID 0x00

#define RE_DATA_FORMAT 0x31
#define RE_POWER_CTL 0x2D
#define RE_BW_RATE 0x2C
#define RE_FIFO_CTL 0x38
//------------------------

//@@@@@@@@@@@@@@@@@@@@@@@@@ Active and Inactive related register
names

#define RE_THRESH_ACT 0x24
#define RE_THRESH_INACT 0x25
#define RE_TIME_INACT 0x26
#define RE_ACT_INACT_CTL 0x27
#define RE_INT_ENABLE 0x2E
#define RE_INT_MAP 0x2F
#define RE_INT_SOURCE 0x30   //Read only

//@@@@@@@@@@@@@@@@@@@@@@@@@



//------------------------Axis Register names
#define RE_DATAZ0 0x36
```

```
#define RE_DATAZ1 0x37

#define RE_DATAY0 0x34
#define RE_DATAY1 0x35

#define RE_DATAX0 0x32
#define RE_DATAX1 0x33
//------------------------


signed int16 data_x;
float data_x_f = 0;

signed int16 data_y;
float data_y_f = 0;

signed int16 data_z;
float data_z_f = 0;

char key = 0;


byte read_interrupt_re = 0;


int1 raw_data_flag = 0;

void put_ADXL345_StandBy_Mode()              //This function will put ADXL345 to be in
standby mode
{
  //------------------Put device in stand-by mode
  output_low(my_9250_CS);

  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_POWER_CTL);
  SPI_XFER_my_ADXL345(0b00000000);                    //Reference datasheet p.26

  output_high(my_9250_CS);
  //------------------
}

void put_ADXL345_Measurement_Mode()        //This function will put ADXL345 to be in
measurement mode
{
```

```
  //------------------Put device in stand-by mode
  output_low(my_9250_CS);

  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_POWER_CTL);
  SPI_XFER_my_ADXL345(0b00001000);                  //Reference datasheet p.26

  output_high(my_9250_CS);
  //------------------
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#INT_EXT0 high
void isr_ext()
{
  disable_interrupts(INT_EXT0);

  delay_ms(100);

  //-------------------------
  output_low(my_9250_CS);

  SPI_XFER_my_ADXL345(READ | single_RW | RE_INT_SOURCE);
  read_interrupt_re = SPI_XFER_my_ADXL345(0);        //read the interrupt source register

  output_high(my_9250_CS);
  //------------------------

  read_interrupt_re = read_interrupt_re & 0b00011000;  //extract the interrupt bit we need only
//Reference page 27


  if(read_interrupt_re == 0b00001000)      //If detect inactivity
  {
    printf("Inactivity\n\r");

    put_ADXL345_StandBy_Mode();

    output_low(my_9250_CS);
    SPI_XFER_my_ADXL345(WRITE | single_RW | RE_INT_ENABLE);
    SPI_XFER_my_ADXL345(0b00010000);      //allow only the activity to trigger interrupt
next time
    output_high(my_9250_CS);
```

```
    put_ADXL345_Measurement_Mode();
  }
  else if(read_interrupt_re == 0b00010000)  //If detect activity
  {
    printf("Activity\n\r");

    put_ADXL345_StandBy_Mode();

    output_low(my_9250_CS);
    SPI_XFER_my_ADXL345(WRITE | single_RW | RE_INT_ENABLE);
    SPI_XFER_my_ADXL345(0b00001000);      //allow only the inactivity to trigger interrupt
next time
    output_high(my_9250_CS);

    put_ADXL345_Measurement_Mode();
  }

  clear_interrupt(INT_EXT0);
  enable_interrupts(INT_EXT0);

  enable_interrupts(INTR_GLOBAL);
}

//=============================================UART serial interrupt (RS232
receive data available interrupt)

#INT_RDA2
void  rda2_isr(void)
{
  key = getc();

  if(key == 'r')
  {
    raw_data_flag = 1; //raw data mode

    disable_interrupts(INT_EXT0);
  }
  else if(key == 'a')
  {
    raw_data_flag = 0; //activity detection mode

    enable_interrupts(INT_EXT0);
```

```
   }
}
//==============================================


void main()
{

  delay_ms(100);       //Provide some delay for ADXL345 to startup


  //------------------Put device in stand-by mode before start to configure

  put_ADXL345_StandBy_Mode();

  //------------------

  //------------------Disable all interrupt of ADXL345
  output_low(my_9250_CS);
  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_INT_ENABLE);
  SPI_XFER_my_ADXL345(0b00000000);                              //Reference datasheet p.27
  output_high(my_9250_CS);
  //------------------

  //------------------------Read the interrupt to clear the flag in ADXL345
  output_low(my_9250_CS);

  SPI_XFER_my_ADXL345(READ | single_RW | RE_INT_SOURCE);
  SPI_XFER_my_ADXL345(0);

  output_high(my_9250_CS);
  //------------------------


  //-----------------------------------------Set range to 8-G and full-resolution (4mg/LSB), Note: For
10-bit resolution it is (15.6mg/LSB)
  output_low(my_9250_CS);

  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_DATA_FORMAT);
  SPI_XFER_my_ADXL345(0b00001010);                        //Reference datasheet page 27-28

  output_high(my_9250_CS);
```

```
   //-----------------------------------------

   //-----------------------------------------Set device to sample at 800 Hz
   output_low(my_9250_CS);

   SPI_XFER_my_ADXL345(WRITE | single_RW | RE_BW_RATE);
   SPI_XFER_my_ADXL345(0b00001101);                      //Reference datasheet page 26 and
Table 7 (page 15)

   output_high(my_9250_CS);

   //-----------------------------------------

   //$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ Setup the active and
inactive feature of the ADXL345

   output_low(my_9250_CS);
   SPI_XFER_my_ADXL345(WRITE | single_RW | RE_THRESH_ACT);
   SPI_XFER_my_ADXL345(0b00000010);       //2 -> 2*62.5mg = .125g for THRESH_ACT
register            //Reference page 25
   output_high(my_9250_CS);

   output_low(my_9250_CS);
   SPI_XFER_my_ADXL345(WRITE | single_RW | RE_THRESH_INACT);
   SPI_XFER_my_ADXL345(0b00000010);       //setting RE_THRESH_INACT to have same
threshold as for RE_THRESH_ACT      //Reference page 25
   output_high(my_9250_CS);

   output_low(my_9250_CS);
   SPI_XFER_my_ADXL345(WRITE | single_RW | RE_TIME_INACT);
   SPI_XFER_my_ADXL345(0b00000101);       //set the time of inactive to be 5 second before
ADXL345 detect inactive event   //Reference page 25
   output_high(my_9250_CS);

   output_low(my_9250_CS);
   SPI_XFER_my_ADXL345(WRITE | single_RW | RE_ACT_INACT_CTL);
   SPI_XFER_my_ADXL345(0b11111111);    //Do AC couple and ALLOW ALL AXIS to
participate in activity/inactivity event detection   //Reference page 25
   output_high(my_9250_CS);

   //-----------------------------------------Continue setting active and inactive

   output_low(my_9250_CS);
```

```
  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_INT_MAP);
  SPI_XFER_my_ADXL345(0b00011000);      //Make activity and inactivity trigger on interrupt
2 pin      //Reference page 27
  output_high(my_9250_CS);

  output_low(my_9250_CS);
  SPI_XFER_my_ADXL345(WRITE | single_RW | RE_INT_ENABLE);
  SPI_XFER_my_ADXL345(0b00011000);      //allow only the activity and inactivity to trigger
interrupt   //Reference page 27
  output_high(my_9250_CS);

  //--------------------------------------------Set Enable Measurement mode

  put_ADXL345_Measurement_Mode();

  //$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ Done setting the active &
inactive detection



  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Enable external interrupt

  enable_interrupts(INT_RDA2);

  ext_int_edge(L_TO_H);
  clear_interrupt(INT_EXT0);
  enable_interrupts(INT_EXT0);

  enable_interrupts(INTR_GLOBAL);

  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  //--------------------------------------------
  while(true)
  {
    if(raw_data_flag == 1)
    {

      //----------------------------------------------Read the x-axis

      output_low(my_9250_CS);
```

```
        SPI_XFER_my_ADXL345(READ | mult_RW | RE_DATAX0);  //tell which address to
start reading

        data_x = SPI_XFER_my_ADXL345(0);                //give slave 8 clock to send in the LSB

        data_x = data_x | (SPI_XFER_my_ADXL345(0) << 8);  //give slave another 8 block to
send in the MSB

        //-----------------------------------------Read the y-axis

        data_y = SPI_XFER_my_ADXL345(0);

        data_y = data_y | (SPI_XFER_my_ADXL345(0) << 8);

        //-----------------------------------------Read the z-axis

        data_z = SPI_XFER_my_ADXL345(0);

        data_z = data_z | (SPI_XFER_my_ADXL345(0) << 8);

        output_high(my_9250_CS);

        //-----------------------------------------

        data_x_f = (float)data_x * .004;                //Convert the raw data to G forces
//Reference page 5
        data_y_f = (float)data_y * .004;
        data_z_f = (float)data_z * .004;

        printf("\n\r%f    %f     %f",data_x_f, data_y_f, data_z_f);

        delay_ms(100);
      }
   }

}
```

Reference

[1] https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

[2] EMBEDDED C PROGRAMMING Techniques and Applications of C and PIC MCUS, by Mark Siegesmund (Note that this is the book used in this class ECE 422/522)