REQUIREMENTS NOT MET

N/A. All Requirements are met in this lab.

PROBLEMS ENCOUNTERED

Some problems encountered with part 1 and 4 of the lab includes the pre-lab question portion where it had to be researched heavily from the manuals and lecture resources to be completed. The problems encountered with part 2 was with figuring out the configuration needed to be done for the SPI read and write functions. For part 3, the major issue was figuring out how to setup the LSM module so that the serial output is readable at WHO_I_AM_A. This required the assistance of PI in explaining the necessary steps to be done. For part 4, it was completed with relatively little issues, but more problems arose in part 5 as I could not get the graph of the measurement from the backpack to work, requiring me to redesign the necessary code for it to function property.

FUTURE WORK/APPLICATIONS

This lab was a good introduction into the implementation and function of SPI and a review of USART in the C language. This lab is to be the expansion of more complex assembly programs, able to give the users' another way to interact with the microprocessor, enabling the running of reactions to human responses by the main program. With the C language users are no longer restricted to the strict assembly-based coding where the user can now implement a higher-level programming that can simplify user's task. Like the subroutine, the way I program is now changed to be inclusive of SPI for more capability of my programs. Having the ability to use SPI allows for me to expand my hardware access to devices. If given more time, the code of the lab could have been more organized and have a much neater layout to further reduce the likelihood of mistakes and further enhance the understanding of the program. With more time a more compacted or efficient communication can be implemented. Additionally, I could have used better instructions to make the code run more efficiently or learn to write more complex programs.

PRE-LAB EXERCISES

Part 1: introduction to SPI and the Lsm330

1. Which device(s) should be given the role of master and which device(s) should be given the role of slave?
   The role of master is given to the device that initiates and controls all data transactions, that being the ATxmega128A1U. The role of slave is given to the Robotic backpack as it updates the DATA register.

2. How will the slave device(s) be enabled? If a slave select is utilized (rather than just have the device[s] be permanently enabled), which pin(s) will be used?
   To enable a slave device, the control register has to be set at bit 6. Slave device is utilized by slave select signal low for the desired slave.

3. What is the order of data transmission? Is the MSB or LSB transmitted first?
   Data are always shifted from master to slave on the master output, slave input line and from slave to master on the master input, slave output line. The data for the LSM330 is transmitted with MSB first.

4. In regard to the relevant clock signal, should data be latched on a rising edge or on a falling edge?
   The data should be latched on a falling edge because in slave mode the module with continue to sleep as long as the slave select is driven high.

5. What is the maximum serial clock frequency that can be utilized by the relevant devices?
   The maximum serial clock frequency that can be utilized by the relevant devices is 10MHz.

Part 5: Plotting Real-Time Accelerometer Data

ii. What are some examples of useful macro functions, in the context of this lab?

Some examples of useful macro functions, in the context of this lab includes #define CTRL_REG5_A_XEN and CTRL_REG5_A_YEN and CTRL_REG5_A_ZEN which allowed me to know what the respect address represent in the context of its utilization rather than having some floating value that is only known through research.

iii. What is the highest speed SPI clock that the IMU can handle?

The highest speed SPI clock that the IMU can handle is 10MHz.

iv. Why is it a better idea to modify global flag variables inside of ISRs instead of doing everything inside of them?

It a better idea to modify global flag variables inside of ISRs instead of doing everything inside of them because ISR halts the processing of other functions so for the ISR to run the entire process inside it would take quite of bit time, stalling the other functions and interfere with the timing of the function whereas modifying an global flag allows the processor to record the result and get back to it on its own functional time.

v. To output two unsigned 32-bit values (0x12345678 [CH1] and 0x9A34F21104 [CH2]) to SerialPlot, list all the bytes in the order you would send them via UART.

CH1: 0x5678, 0x1234

CH2: 0x1104, 0x34F2, 0x009A

vi. What is the most positive value that can be received from the accelerometer (in decimal)? What about the most negative?

Most positive value that can be received from the accelerometer (in decimal) for a signed 24 bit value is 8388607 while the most negative value that can be received from the accelerometer (in decimal) for a signed 24 bit value is -8388608.

PSEUDOCODE/FLOWCHARTS

SECTION X (1, 2, etc.)

Part 2: Communicating with the Lsm330

Johnny L
Lab 6
EEL3744

Lab 6 Part 2 Flowchart

Lab6-2 Main: Initialize SPIF
Loop Continuously
SPIF Write 0x53

SPI.c: void spif_init(void)
set I/O and initial output
Configure SPIF.CTRL, (8MHZ)

uint8_t spif_read(void)
Toggle select
Load Data to SPI, 0x87
Loop till transmission is over
Toggle select
return DATA

void spif_write(uint8_t data)
Toggle select
Load Data
Loop till transmission is over
Toggle select

Part 3: Receiving with SPI & Communicating with the LSM330

Lab 6 Part 3 Flowchart

Lab 6 3
Main: Initialize SPIF
Load a variable with the accel-read(register)
return0;

lsm330.c: accel_read
Enable accelerometer
spif_write(reg_addr | BIT7)
variable = spif_read();
Disable accelerometer
return(variable);

accel_write
Enable accelerometer
spif_write(reg_addr);
Spif_write(data);
Disable accelerometer
return

Part 4: Configuring the LSM330 Accelerometer

Lab 6 Part 4

accel_init: accel_write - reset
- measure all dimensions

init_interrupt: INT0MASK= Pin7
INTCTRL = Pin1
PIN7CTRL = Pin1
PMICCTRL = Pin1
DIRCLR = Pin7

//Global Variable
volatile int aFlag=0

// ISR Vector
Reset Int Flag
set aFlag=1

Part 5: Plotting Real-Time Accelerometer Data

Lab 6 Part 5

main: Initialize SPI
Initialize USARTD0
Initialize Interrupt
Initialize the Accelerometer
lsm330_data_t
Loop: check (if flag is trigger) {
Read -> Output of register
X, Y, Z.

USARTd0_init: Set Port D direction.
Set baud rate.
Set USARTD0 control C and B.

USARTd0_out_char: Loop till transmission is done.
Output char c.

USARTd0_out_string: Loop till end of string
Output string

PROGRAM CODE

SECTION X (1, 2, etc.)

Part 2: Communicating with the Lsm330

```
/********************************************************************************
;Lab 6 Part 2
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: COMMUNICATING WITH THE LSM330---Testing Code
;********************************************************************************/
#include <avr/io.h>
#include "Lab6_2.h"
#include "spi.h"


int main(void){
	//Initialize SPI
	spif_init();

	//Transmit 0x53 continuously
	while(TRUE){
		spif_write(0x53);
	}
}
```

Part 3: Receiving with SPI & Communicating with the LSM330

```
/********************************************************************************
;Lab 6 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Receiving with SPI & Communicating with the LSM330
;*********************************INCLUDES***************************************/
#include <avr/io.h>
#include "lsm330.h"
#include "lsm330_registers.h"
#include "Lab6_3.h"
//#include "spi.h"

int main(void){
	//Initialize SPI
	spif_init();

	//Read WHO_AM_I_A register
	uint8_t check = accel_read(WHO_AM_I_A);
	return 0;
}
```

Part 4: Configuring the LSM330 Accelerometer

Code added to the ISM330.c

```
//Initialize the Accelerometer
void accel_init(void){
	//Reset CTRL_REG4_A, enable interrupt, active high interrupt
	//Int1 connects to pc7, need to config external interrupt on uController
```

```
	accel_write(CTRL_REG4_A, CTRL_REG4_A_STRT | CTRL_REG4_A_INT1_EN | CTRL_REG4_A_IEA);
	//Enable accelerometer to measure all 3 dimensions simultaneously and config measurements rate at 
1600Hz
	accel_write(CTRL_REG5_A, CTRL_REG5_A_XEN | CTRL_REG5_A_YEN | CTRL_REG5_A_ZEN | CTRL_REG5_A_ODR3 | 
CTRL_REG5_A_ODR0);
}
```

Part 5: Plotting Real-Time Accelerometer Data

```
/*****************************************************************************
;Lab 6 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Plotting Real-Time Accelerometer Data
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#include "lsm330.h"
#include "lsm330_registers.h"

//ISR global flag
volatile int accel_flag = 0;

int main(void){
	//Initialize SPI
	spif_init();
	//Initialize USART
	usartd0_init();
	//Initialize External Interrupt and the Accelerometer
	accel_init();

	//Turn on low level interrupts
	PMIC.CTRL=0x07;
	//Turn on global interrupts
	sei();

	// This type is a union that will allow you to access the data read from the LSM330 in a much 
easier way.
	lsm330_data_t lsm_data;

	//Loop always
	while(1){
		//Check on global flag
		if(accel_flag){
			//Read -> Output XL
			lsm_data.byte.accel_x_low = accel_read(OUT_X_L_A);
			usartd0_out_char((char) lsm_data.byte.accel_x_low);
			//Read -> Output XH
			lsm_data.byte.accel_x_high = accel_read(OUT_X_H_A);
			usartd0_out_char((char) lsm_data.byte.accel_x_high);
			//Read -> Output YL
			lsm_data.byte.accel_y_low = accel_read(OUT_Y_L_A);
			usartd0_out_char((char)lsm_data.byte. accel_y_low);
			//Read -> Output YL
			lsm_data.byte.accel_y_high = accel_read(OUT_Y_H_A);
			usartd0_out_char((char)lsm_data.byte. accel_y_high);
			//Read -> Output ZL
			lsm_data.byte.accel_z_low = accel_read(OUT_Z_L_A);
			usartd0_out_char((char) lsm_data.byte.accel_z_low);
```

```
                    //Read -> Output ZH
                    lsm_data.byte.accel_z_high = accel_read(OUT_Z_H_A);
                    usartd0_out_char((char) lsm_data.byte.accel_z_high);

                    //Reset Flag
                    accel_flag = 0;
            }
    }
    return 0;
}
```

APPENDIX

Part 2: Communicating with the LSM330

Support code: SPI.C

```
/*
 * spi.c
 *
 *  Last updated: 10/21/2019 3:17 PM
 *  Author: Dr. Schwartz
*/

#include <avr/io.h>
#include "spi.h"

/*******************************************************************************
;Lab 6 Part 2
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: COMMUNICATING WITH THE LSM330---SPI
;*********************************INCLUDES**************************************/
extern const uint8_t PORTF_DIRSET_CONFIG;
extern const uint8_t PORTF_DIRCLR_CONFIG;
extern const uint8_t PORTF_OUTSET_CONFIG;
//*****************************END OF INCLUDES*********************************
/* initializes the SPI module of Port F to communicate with the LSM330 */
void spif_init(void){
  /* configure pin direction of SPI signals */
  PORTF.DIRSET  = PORTF_DIRSET_CONFIG;    //Output
  PORTF.DIRCLR  = PORTF_DIRCLR_CONFIG;    //Input
  PORTF.OUTSET  = PORTF_OUTSET_CONFIG;    //Set initial output as high.

	/* 8 MHz SPI frequency since 10MHz is the maximum allowed by the LSM330 */
	SPIF.CTRL	=	SPI_PRESCALER_DIV4_gc	|	SPI_MASTER_bm |	SPI_MODE_2_gc |
	SPI_ENABLE_bm;
}

/* writes a single byte of data to the SPIF data register */
void spif_write(uint8_t data){
	//Enable Slave port f pin 4
	PORTF.OUTTGL = 0x10;

	SPIF.DATA = data;
	while((SPIF.STATUS & 0x80) != 0x80);	/* wait for transfer to be complete */

	//Turn off slave select
	PORTF.OUTTGL = 0x10;
}

/* attempts to read a byte of data from device connected to SPIF */
uint8_t spif_read(void){
  PORTF.OUTTGL = 0x10;

  SPIF.DATA = 0x37;                    /* write garbage to cause transaction */
  while((SPIF.STATUS & 0x80) != 0x80);    /* wait for transfer to be complete */

  PORTF.OUTTGL = 0x10;
```

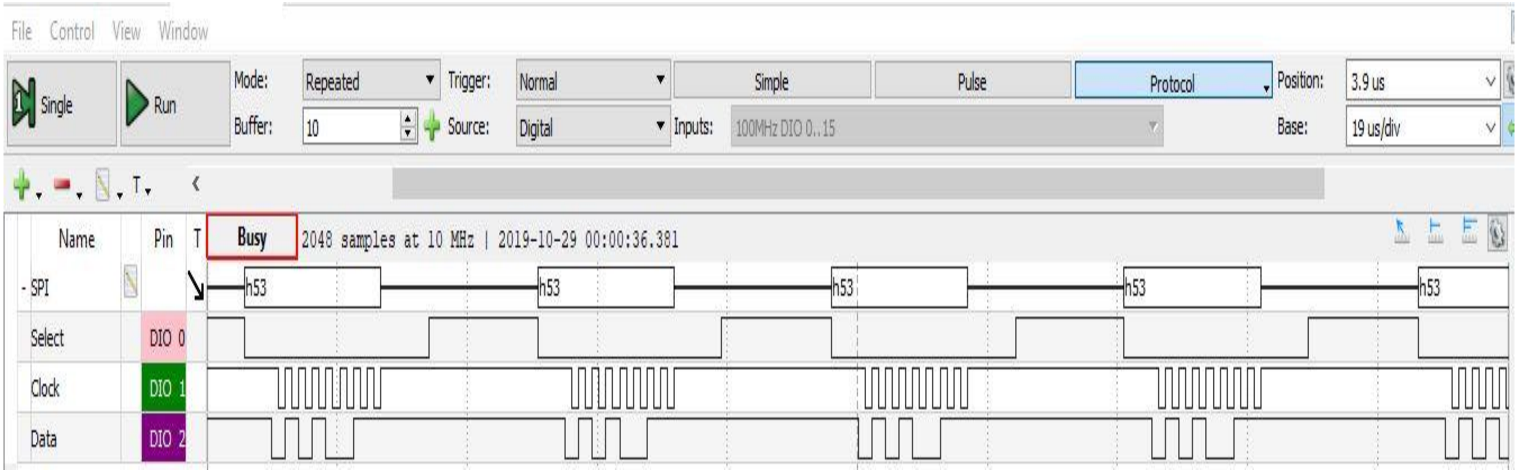
```
  return SPIF.DATA;
}
```

Support Code: SPI.h

```
#ifndef SPI_H_
#define SPI_H_
/*
 * spi.h
 *
 *  Last updated: 10/21/2019 3:17 PM
 *  Author: Dr. Schwartz
 */

#include <avr/io.h>

#define SCK                 PIN7_bm
#define MISO          PIN6_bm
#define MOSI          PIN5_bm

/* initializes the SPI module of Port F to communicate with the LSM330 */
void spif_init(void);

/* writes a single byte of data to the SPIF data register */
void spif_write(uint8_t data);

/* attempts to read a byte of data from device connected to SPIF */
uint8_t spif_read(void);
#endif /* SPI_H_ */
```

Support Code: Lab6_2.h

```
/*******************************************************************************
;Lab 6 Part 2
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: COMMUNICATING WITH THE LSM330
;*******************************************************************************/
#ifndef LAB6_2_H_
#define LAB6_2_H_

//Defines
#define TRUE 1

//****************** SPI.h ***********************************
//PORT F
const uint8_t PORTF_DIRSET_CONFIG = 0xB0; //0b10110000;
const uint8_t PORTF_DIRCLR_CONFIG = 0x40; //0b01000000;
const uint8_t PORTF_OUTSET_CONFIG = 0x10; //0b00010000;

#endif /* LAB6_2_H_ */
```



Screenshot 1: DAD measurement of SPI

Part 3: Receiving with SPI & Communicating with the LSM330

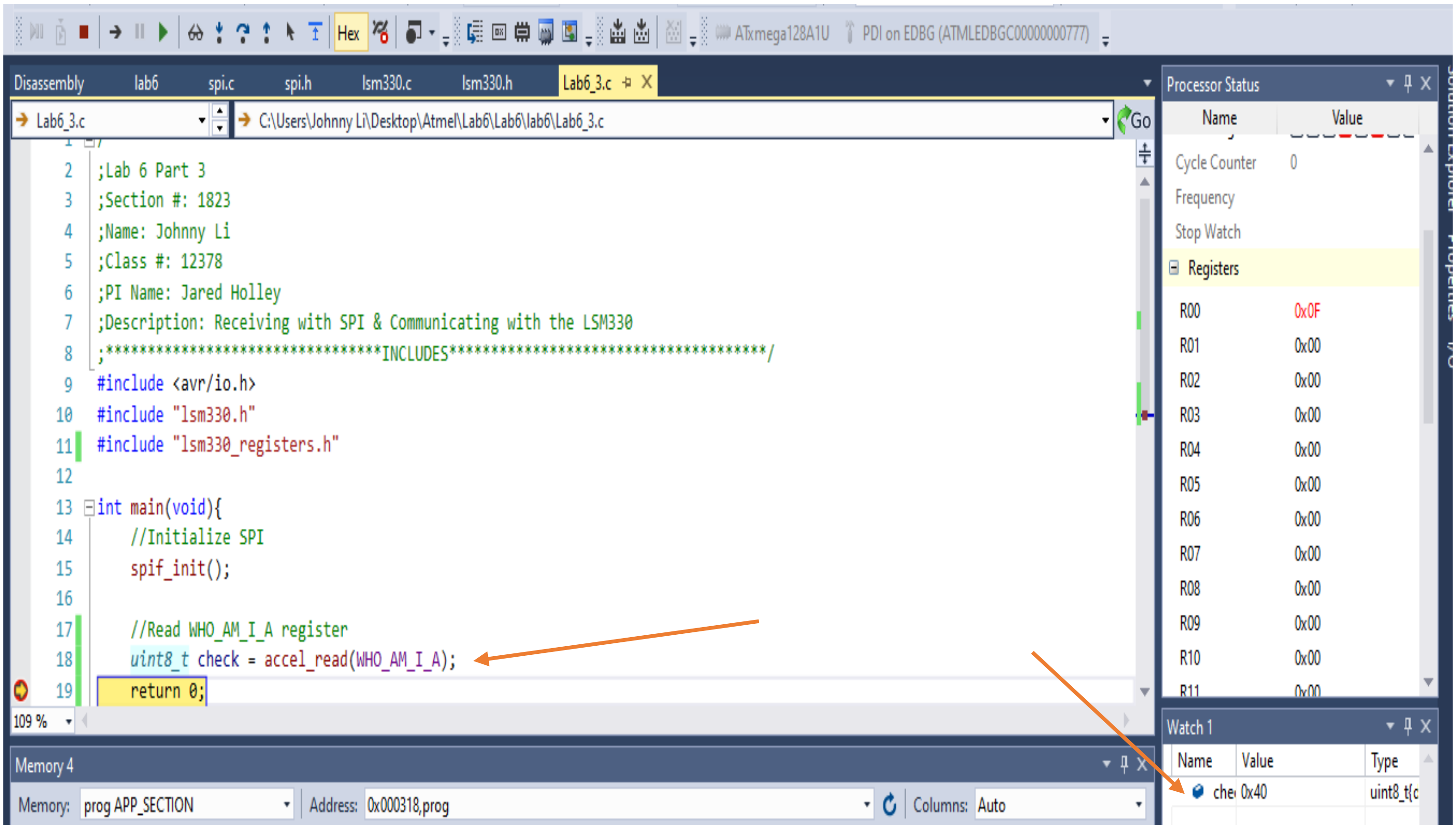
Support Code: LSM330.h

```
/*
 * lsm330.h
 *
 *  Author: Wes Piard
 */

#ifndef LSM330_H_
#define LSM330_H_

/* ------------------------------------------------------------------------ */

/* used to differentiate the accelerometer and gyroscope within the LSM330 */
typedef enum {LSM330_ACCEL, LSM330_GYRO} lsm330_module_t;

/* can be used to contain the separated bytes of data as they are read from
   the LSM330 */
typedef struct lsm330_data_raw
{
  uint8_t accel_x_low, accel_x_high;
  uint8_t accel_y_low, accel_y_high;
  uint8_t accel_z_low, accel_z_high;

  uint8_t gyro_x_low, gyro_x_high;
  uint8_t gyro_y_low, gyro_y_high;
  uint8_t gyro_z_low, gyro_z_high;
}lsm330_data_raw_t;

/* contains the full concatenated signed 16-bit words of data */
typedef struct lsm330_data_full
{
  int16_t accel_x, accel_y, accel_z;
  int16_t gyro_x, gyro_y, gyro_z;
}lsm330_data_full_t;

/* provides the ability to choose how to access the LSM330 data */
typedef union lsm330_data
{
```

```
  lsm330_data_full_t  word;
  lsm330_data_raw_t   byte;
}lsm330_data_t;

/* ------------------------------------------------------------------------ */

/* your lsm330 function prototypes here */

//Writes a single byte of data (data) to a specific accelerometer register (reg_addr) within the LSM330.
void accel_write(uint8_t reg_addr, uint8_t data);

//Returns a single byte of data that is read from a specific accelerometer register (reg_addr) within
the LSM330.
uint8_t accel_read(uint8_t reg_addr);

#endif /* LSM330_H_ */
```

Support Code: LSM330.c

```
/*****************************************************************************
;Lab 6 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Receiving with SPI & Communicating with the LSM330
;*********************************INCLUDES*************************************/
#include <avr/io.h>
#include "lsm330.h"
#include "lsm330_registers.h"
#include "spi.h"

#define READ_BIT PIN7_bm

/* your lsm330 function definitions here */
//Returns a single byte of data that is read from a specific accelerometer register (reg_addr) within
the LSM330.s
uint8_t accel_read(uint8_t reg_addr){
	//Select accel
	PORTF.OUTCLR = SSA;  //SSA enable

	//Address OR with READ CYCLE enable
	spif_write( (reg_addr | READ_BIT) );
	uint8_t value = spif_read();

	PORTF.OUTSET = SSA;  //SSA disable

	return(value);
}

//Writes a single byte of data (data) to a specific accelerometer register (reg_addr) within the LSM330.
void accel_write(uint8_t reg_addr, uint8_t data){
	//Select accel
	PORTF.OUTCLR = SSA;  //SSA enable

	//Write reg address then data
	spif_write(reg_addr);
	spif_write(data);

	PORTF.OUTSET = SSA;  //SSA disable

	return;
}
```



Screenshot 2: Read WHO_AM_I_A register

Part 5: Plotting Real-Time Accelerometer Data

Support Code: Lab6_5.h

```
/******************************************************************************
;Lab 6 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Plotting Real-Time Accelerometer Data
;******************************************************************************/
#ifndef LAB6_5_H_
#define LAB6_5_H_

//Defines
#define TRUE 1

#define BIT7 0x80
#define BIT0 0x01
#define READ PIN7_bm
//****************** LSM.h ***********************************
//PORT C
const uint8_t PORTC_INTCTRL_CONFIG = 0x03;//0b00000011;
const uint8_t PORTC_INT0MASK_CONFIG = 0x80;//0b10000000;
const uint8_t PORTC_PIN7CTRL_CONFIG = 0x02;//0b00000010
const uint8_t PMIC_CTRL_CONFIG = 0x01;//0b00000001
const uint8_t PORTC_DIRCLR_CONFIG = 0x80; //0b10000000;

#endif /* LAB6_5_H_ */
```

Support Code: Remaining part of Lab6 part 5 main.

```
ISR(PORTC_INT0_vect){
	//Push Status Registers
	uint8_t status = CPU_SREG;

	//Set interrupt flags
	PORTC.INTFLAGS = 0x01;

	//Set global variable
	accel_flag = 1;

	//Pop Status Reg
	CPU_SREG = status;

	return;
}
```

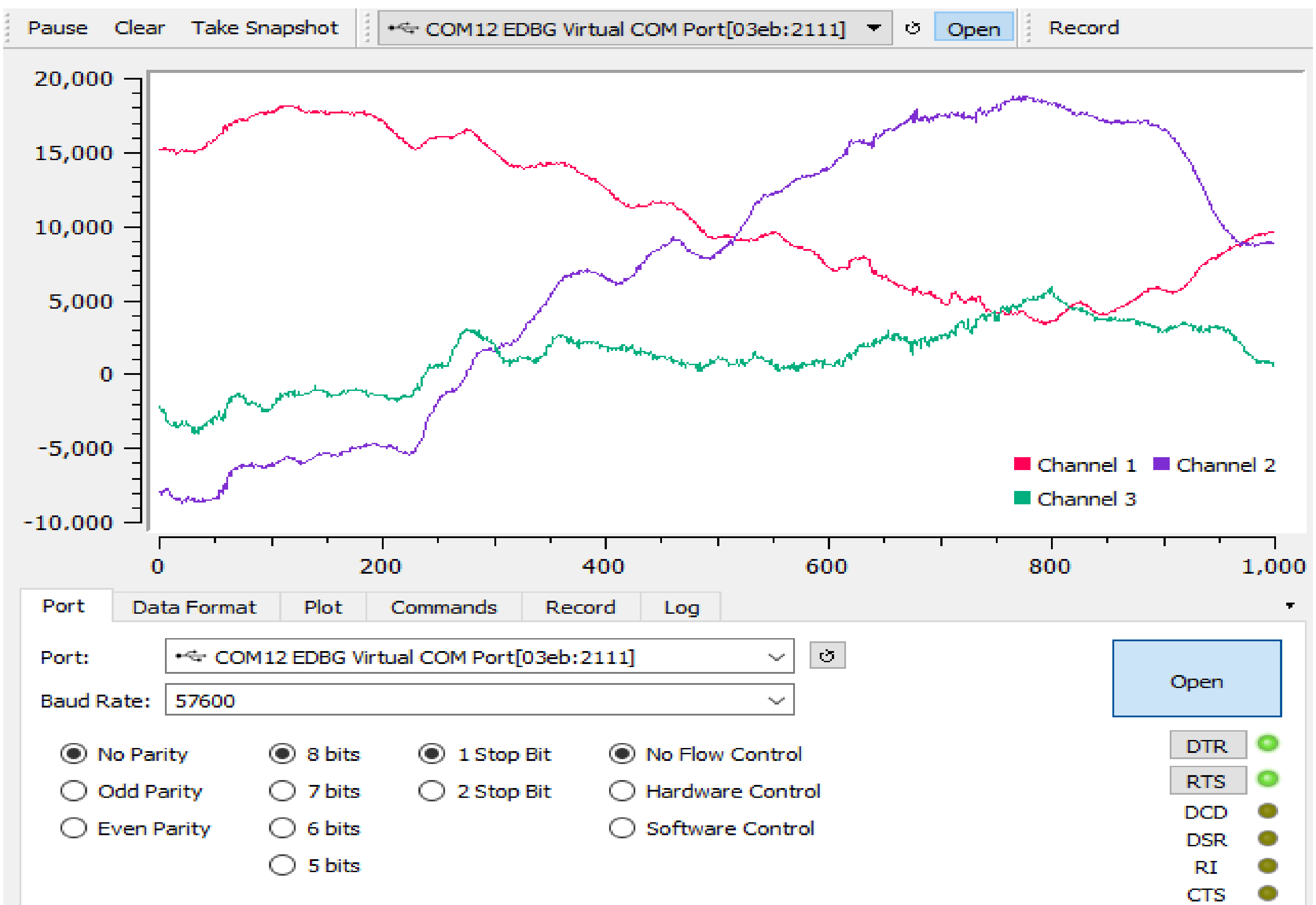
Support Code: USART.h

```
/*******************************************************************************
;Lab 6 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Plotting Real-Time Accelerometer Data
*/
#ifndef USART_H_
#define USART_H_

//USART Initialization
void usartd0_init(void);

//Output character
void usartd0_out_char(char output);

//Output string
void usartd0_out_string(char *str);

#endif /* USART_H_ */
```

Support Code: USART.c

```
/*******************************************************************************
;Lab 6 Part 5
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: Plotting Real-Time Accelerometer Data
*/

#include <avr/io.h>

//USART Initialization
void usartd0_init(void){
	//Configure TxD and RxD pins
	PORTD.OUTSET = PIN3_bm;
	PORTD.DIRSET = PIN3_bm;
	PORTD.DIRCLR = PIN2_bm;

	//Baud rate: At 2 MHz, 150 BSEL, -7 BSCALE corresponds to 57600 bps */
```

```
	USARTD0.BAUDCTRLA = (uint8_t)150;
	USARTD0.BAUDCTRLB = (uint8_t)( (-7 << 4) | (150 >> 8) );

	//8 data bits, no parity, and one stop bit.
	USARTD0.CTRLC = USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc &
~USART_SBMODE_bm;

	//Enable Receiver and/or Transmitter
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;
}

//Output character
void usartd0_out_char(char output){
	//Wait till transmission is done
	while(!(USARTD0.STATUS & USART_DREIF_bm));
	USARTD0.DATA = output;		//output c
}

//Output string
void usartd0_out_string(char *str){
	//Loop char pointer to get string
	while(*str){
		usartd0_out_char(*(str++)); //Output string
	}
}
```



Screenshot 3: Plot accelerometer data for all three coordinate planes (X, Y, and Z).