Lab 6 Summary

Schuster, Mark

Section 1540

11/05/2017

B) Pre-Lab Answers:

1. 10 MHz, thus we must use a prescaler of 4 to set the input CLK to 8MHz.
2. MSB as stated in the write/read cycle section of the datasheet
3. They are enabled by setting the sampling rate in the CTRL_REG5_A register, whereas the gyroscope is enabled by setting the GYRO_ENABLE pin on port A.
4.
5. Because SPI works by shifting the data out and simultaneously into the data register. Thus to receive data, dummy data must first be present to by shifted out.
6. Because this minimizes time spent inside the interrupt. Prolonged time spent inside an interrupt can limit the amount of time available to other potential interrupts, and slow the program down.

C) Problems encountered:

The graph was not plotting the correct data, and the accelerometer was not taking samples. I asked two TAs but neither were able to help. It wasn't until the .gif was posted that I realized I had to use "Serial Port" control scheme under "External Connection" instead of the USART configuration immediately present upon connection to the device.

D) Future Work/Applications:

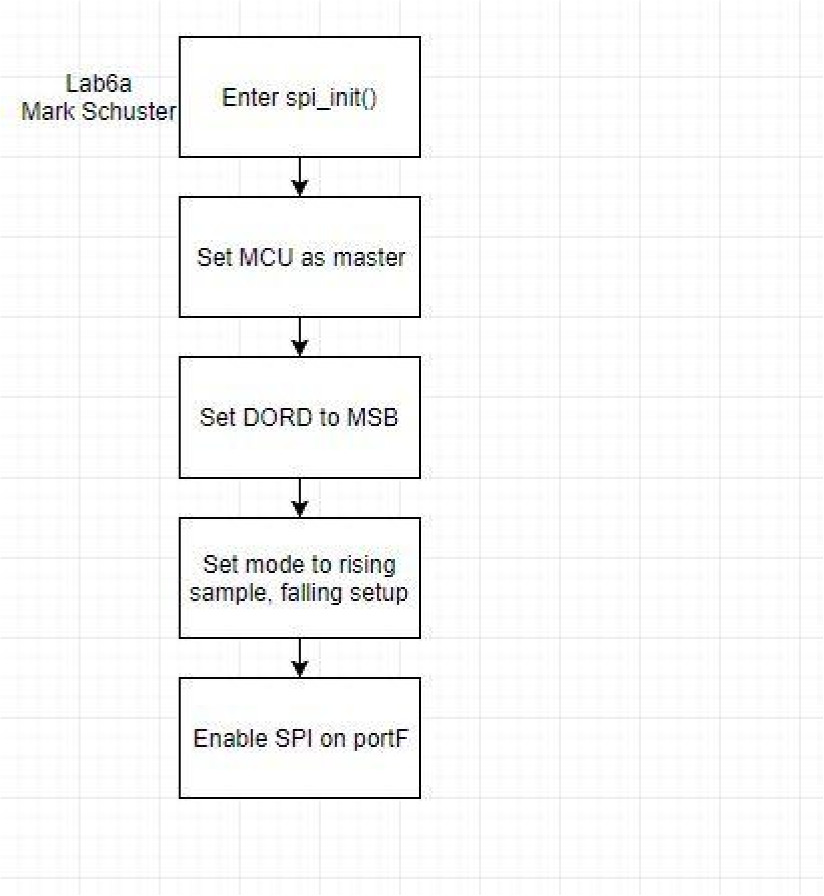
I enjoyed working with SPI and I look forward to using it in high throughput applications.

E) Schematics:

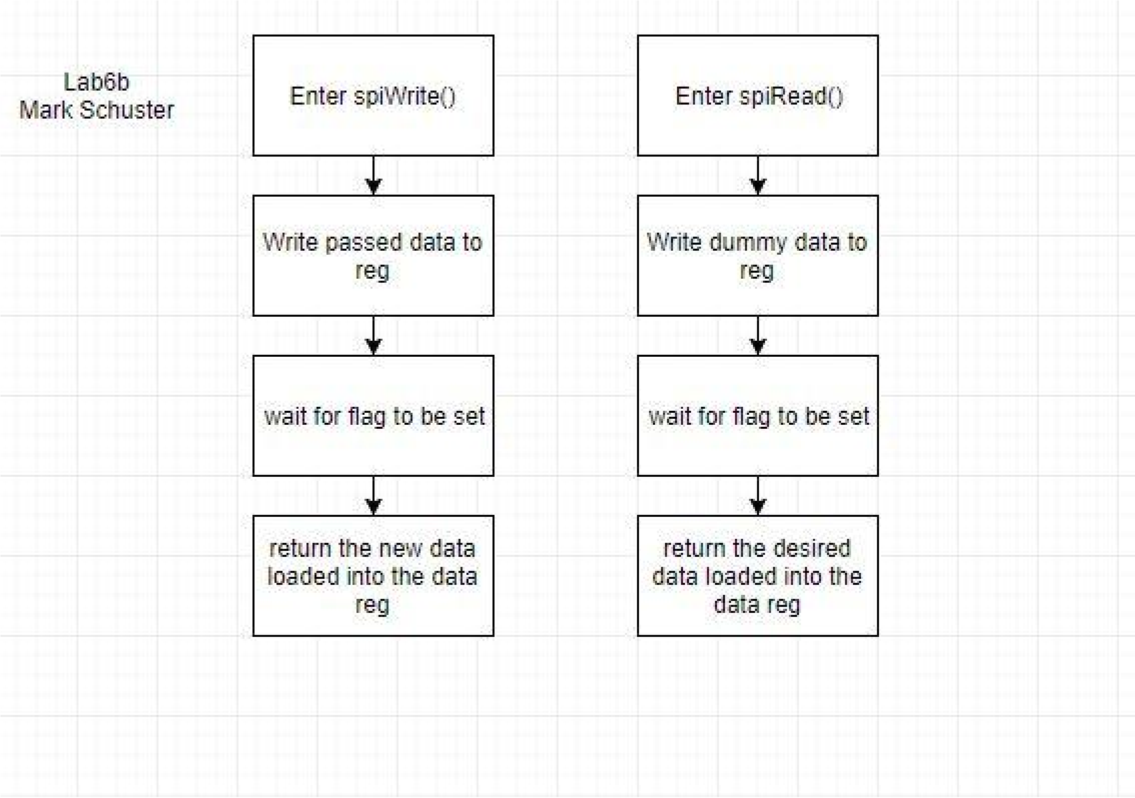
Not applicable for this lab.

F) Pseudocode/Flowcharts:

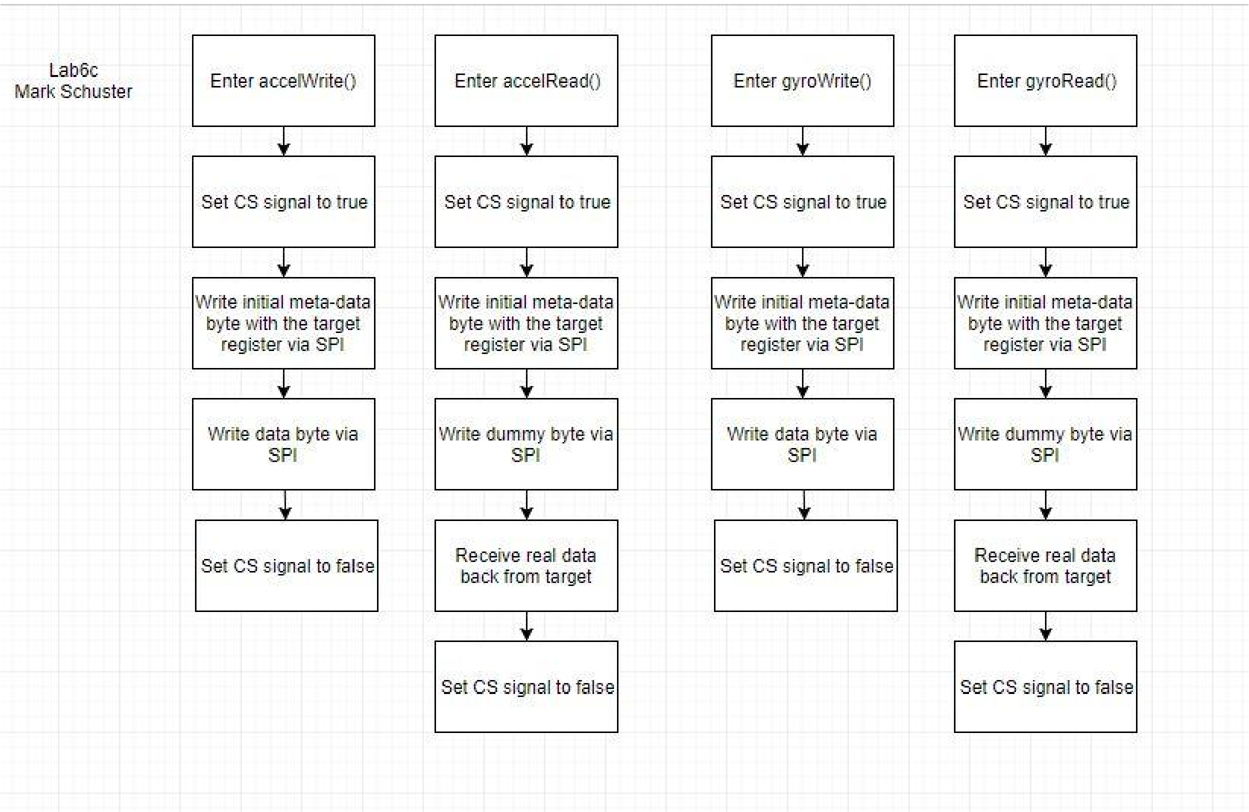
Lab6a flow diagram:



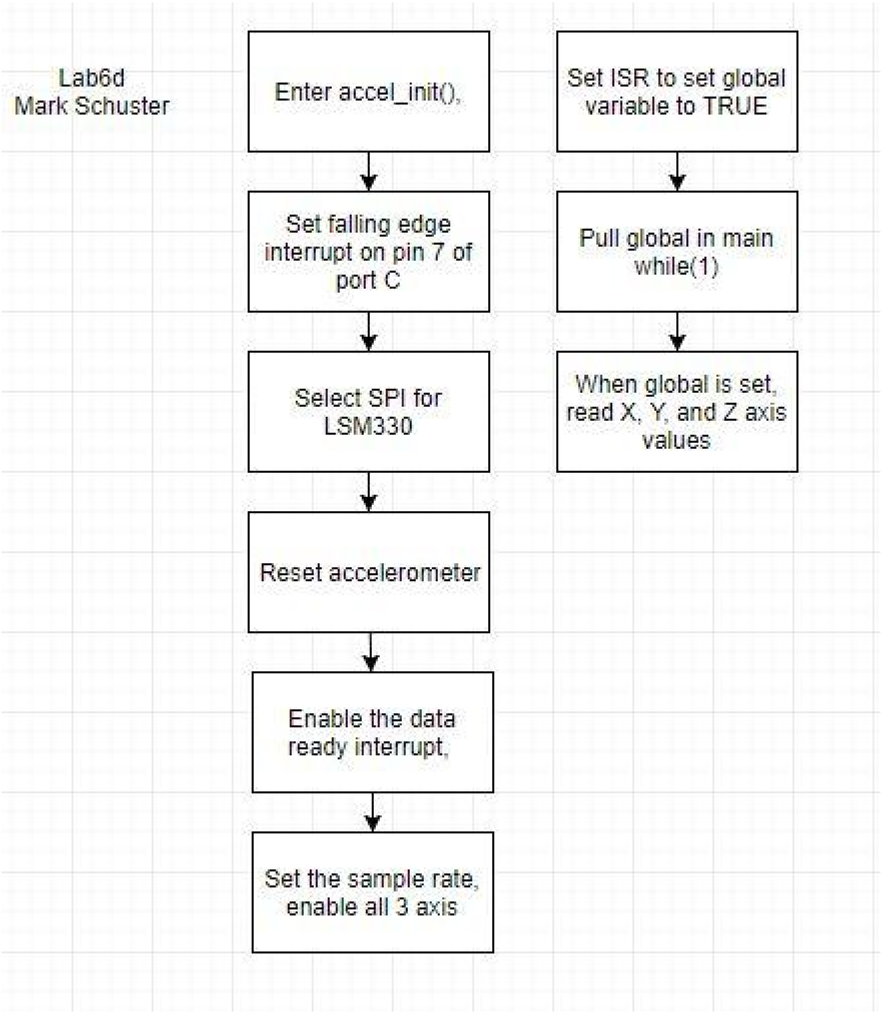
Lab6b flow diagram:



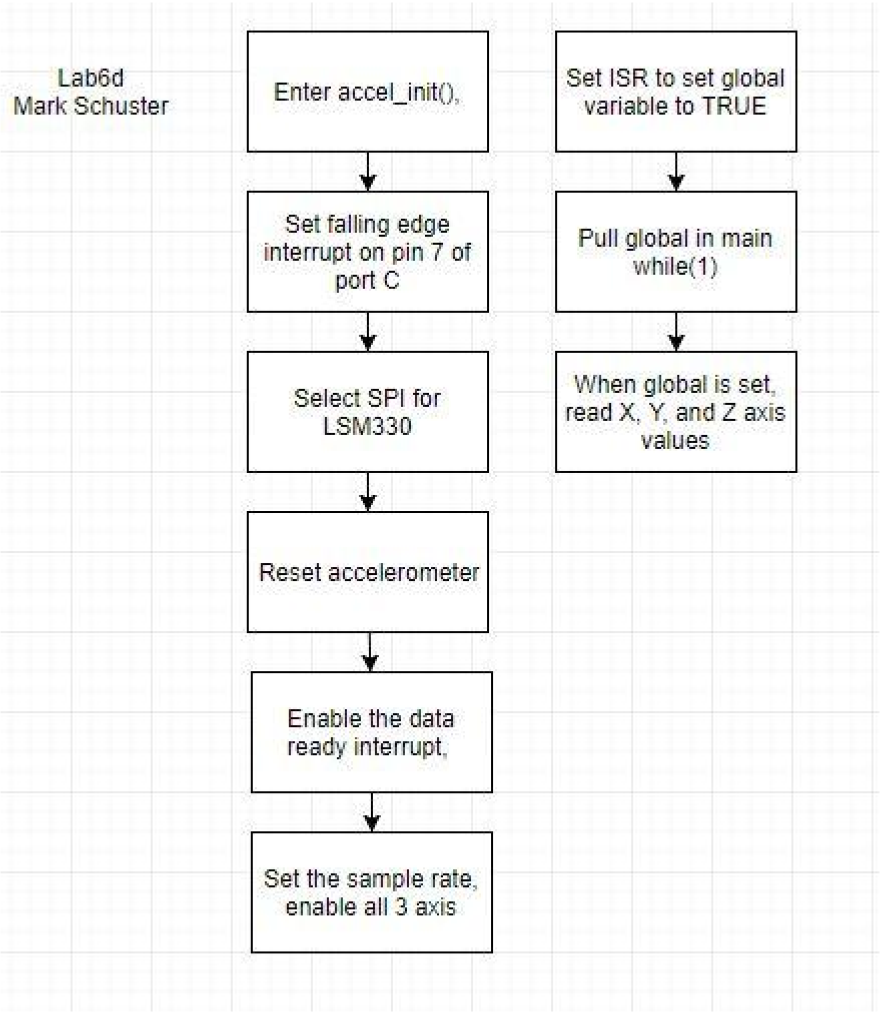
Lab6c flow diagram:



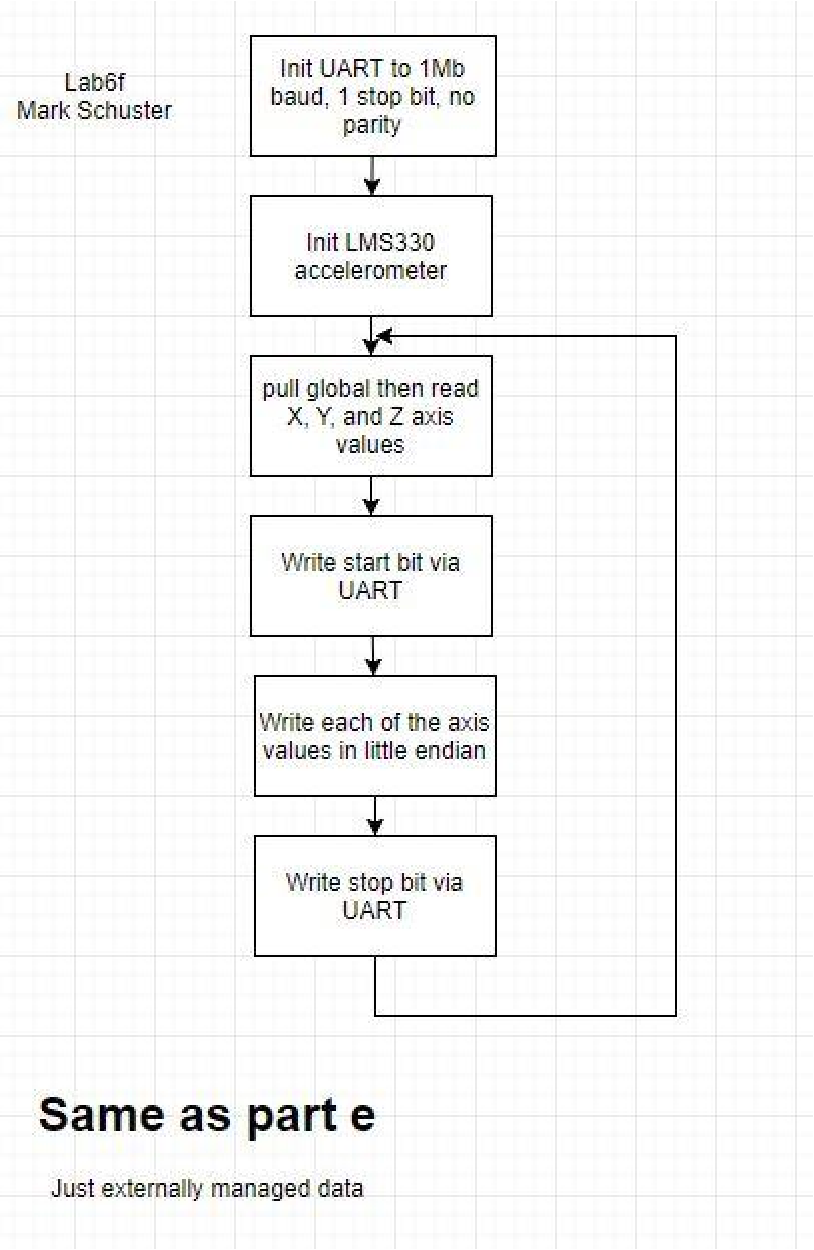
Lab6d flow diagram:


Lab6d
Mark Schuster
Enter accel_init(),
Set falling edge interrupt on pin 7 of port C
Select SPI for LSM330
Reset accelerometer
Enable the data ready interrupt,
Set the sample rate, enable all 3 axis
Set ISR to set global variable to TRUE
Pull global in main while(1)
When global is set, read X, Y, and Z axis values

Lab6e flow diagram:


Lab6d
Mark Schuster
Enter accel_init(),
Set falling edge interrupt on pin 7 of port C
Select SPI for LSM330
Reset accelerometer
Enable the data ready interrupt,
Set the sample rate, enable all 3 axis
Set ISR to set global variable to TRUE
Pull global in main while(1)
When global is set, read X, Y, and Z axis values

Lab6f flow diagram:

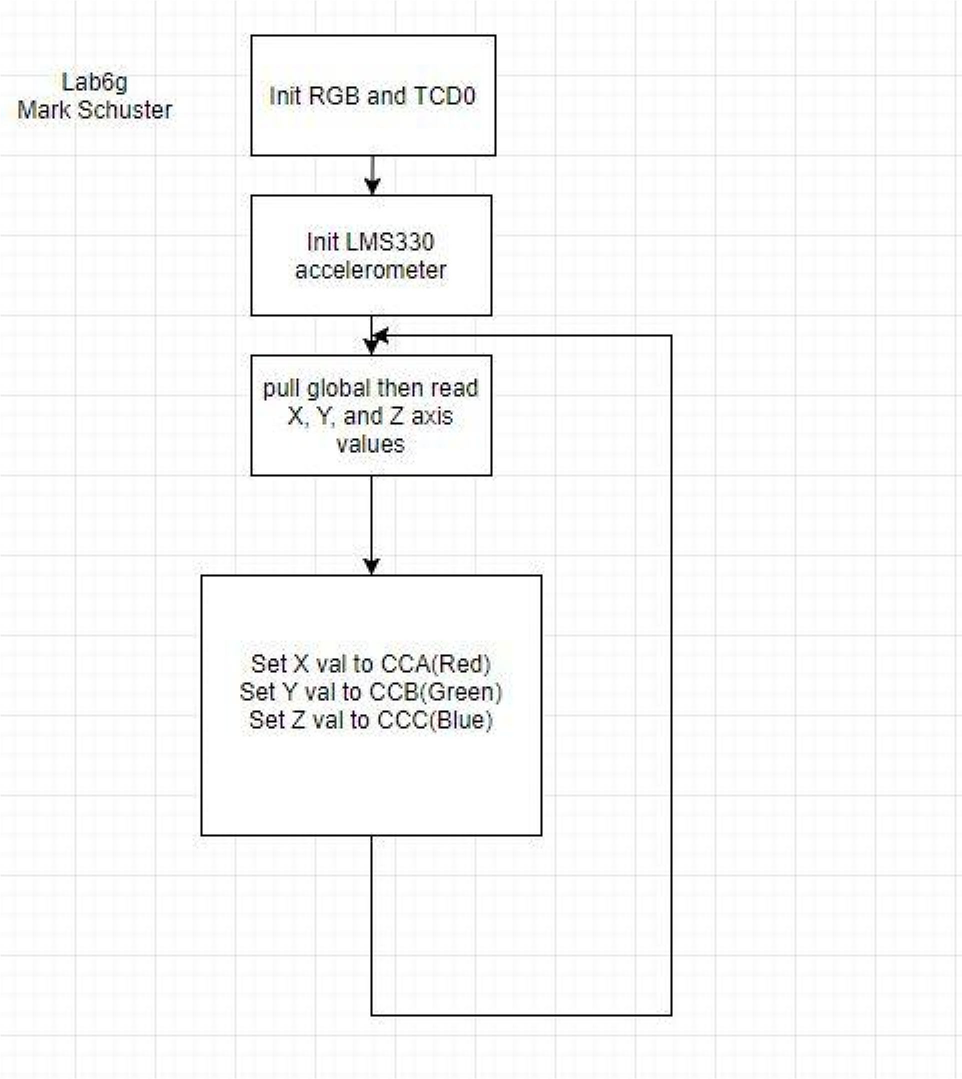

Init UART to 1Mb baud, 1 stop bit, no parity
Init LMS330 accelerometer
pull global then read X, Y, and Z axis values
Write start bit via UART
Write each of the axis values in little endian
Write stop bit via UART

Lab6f
Mark Schuster

Same as part e

Just externally managed data

Lab6g flow diagram:



G) Program Code:

Lab6 part a spi_init():

```
/**********************FUNCTIONS***************************************/
 // Subroutine Name: spi_init
 // Configure the SPI peripheral
 // Inputs: None
 // Outputs: None
 // Affected: None
 void spi_init(void){

        PORTF_DIRSET = 0b10111100;
        PORTF_DIRCLR = 0b01000011;
        // Master, MSB DORD, LE:RS, TE:FS, SYS_CLK/4 = 8MHz
        SPIF_CTRL = SPI_ENABLE_bm | SPI_MASTER_bm | SPI_MODE_0_gc |
SPI_PRESCALER_DIV4_gc;

        return;
 }
```

Lab6 part b spiWrite():

```
/**********************FUNCTIONS***************************************/
 // Subroutine Name: spiWrite
 // Write a byte of data using SPI
 // Inputs: data - data to be sent
 // Outputs: None
 // Affected: None
 uint8_t spiWrite(uint8_t data){
        SPIF_DATA = data;
        while(!(SPIF_STATUS & SPI_IF_bm));
        return SPIF_DATA;
 }
```

Lab6 part b spiRead():

```
/**********************FUNCTIONS***************************************/
 // Subroutine Name: spiRead
 // Reads the return value of spiWrite.
 // Inputs: None
 // Outputs: Return value of spiWrite.
 // Affected: None
 uint8_t spiRead(void){
        return  spiWrite(0xFF);
 }
```

Lab6b.c:

```c
// Lab 6 part B
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:  Send 0x53 over spi forever.

#include <avr/io.h>
#include "spi.h"
#define CLK_PRESCALER       CLK_PSADIV_1_gc

void INIT_CLK(void);
void INIT_ADC(void);



int main(void)
{
       // Init CLK, pin dir, and ADC.
       INIT_CLK();
       spi_init();

       while (1){
              spiWrite(0x53);

       }

}

/**********************FUNCTIONS***************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
       // Enable 32Mhz CLK.
       OSC_CTRL = OSC_RC32MEN_bm;

       // Wait for 32Mhz flag to be set.
       while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

       // Write to restriction register to allow writing
       // to the CLK CTRL, then sel the 32MHz CLK.
       CPU_CCP = CCP_IOREG_gc;
       CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

       // Write to restriction register to allow writing
       // to the CLK PSCTRL, then set the prescaler.
       CPU_CCP = CCP_IOREG_gc;
       CLK_PSCTRL = CLK_PRESCALER;
}
```

Lab6 part c accelWrite():

```
/**********************FUNCTIONS***************************************/
// Subroutine Name: accelWrite
// Write a byte to the accelerometer.
// Inputs: Byte to be written.
// Outputs: None
// Affected: None
void accelWrite(uint8_t targetReg, uint8_t data){
       uint8_t spiVal = 0x00 | targetReg;
       PORTF_OUTCLR = RBB_SSA;
       spiWrite(spiVal);
       spiWrite(data);
       PORTF_OUTSET = RBB_SSA;
}
```

Lab6 part c accelRead():

```
/**********************FUNCTIONS***************************************/
// Subroutine Name: accelRead
// Read a byte from the accelerometer.
// Inputs: None
// Outputs: Byte read from accelerometer.
// Affected: None
uint8_t accelRead(uint8_t targetReg){
       uint8_t spiVal = 0x80 | targetReg;
       uint8_t result;
       PORTF_OUTCLR = RBB_SSA;
       spiWrite(spiVal);
       result = spiRead();
       PORTF_OUTSET = RBB_SSA;
       return result;
}
```

Lab6 part c gyroWrite():

```
/**********************FUNCTIONS***************************************/
// Subroutine Name: accelWrite
// Write a byte to the gyroscope.
// Inputs: Byte to be written.
// Outputs: None
// Affected: None
void gyroWrite(uint8_t targetReg, uint8_t data){
       uint8_t spiVal = 0x00 | targetReg;
       PORTF_OUTCLR = RBB_SSG;
       spiWrite(spiVal);
       spiWrite(data);
       PORTF_OUTSET = RBB_SSG;
}
```

Lab6 part c gyroWrite():

```c
/**********************FUNCTIONS***************************************/
// Subroutine Name: gyroRead
// Read a byte from the gyroscope.
// Inputs: None
// Outputs: Byte read from gyroscope.
// Affected: None
uint8_t gyroRead(uint8_t targetReg){
	uint8_t spiVal = 0x80 | targetReg;
	uint8_t result;
	PORTF_OUTCLR = RBB_SSG;
	spiWrite(spiVal);
	result = spiRead();
	PORTF_OUTSET = RBB_SSG;
	return result;
}
```

Lab6 part d accel_init():

```c
/**********************FUNCTIONS***************************************/
// Subroutine Name: accel_init
// Init the accelerometer.
// Inputs: None
// Outputs: None
// Affected: None
void accel_init(void){
	PORTA_DIRSET = RBB_PROTOCOL_SEL;
	PORTA_OUTCLR = RBB_PROTOCOL_SEL;
	PORTF_OUTSET = RBB_SSA | RBB_SSG | RBB_SENSOR_SEL_ACCEl_bp;
	spi_init();
	accelWrite(CTRL_REG4_A, CTRL_REG4_A_STRT);
	PORTC_DIRCLR = RBB_INT1A;
	PORTC_PIN7CTRL = PORT_ISC_FALLING_gc;
	PORTC_INT0MASK = RBB_INT1A;
	PORTC_INTCTRL = PORT_INT0LVL_LO_gc;
	uint8_t reg4AInitData = CTRL_REG4_A_DR_EN | CTRL_REG4_A_IEA | CTRL_REG4_A_INT1_EN;
	accelWrite(CTRL_REG4_A, reg4AInitData);
	uint8_t reg5AInitData = CTRL_REG5_A_ODR3 | CTRL_REG5_A_ODR0 | CTRL_REG5_A_ZEN |
CTRL_REG5_A_YEN | CTRL_REG5_A_XEN;
	accelWrite(CTRL_REG5_A, reg5AInitData);


}
```

Lab6 part e USART out function:

```
/**********************FUNCTIONS***************************************/
// Subroutine Name: STREAM_DATA
// Send axis data over UART.
// Inputs: accelXData, accelYData, accelZData
// Outputs: None
// Affected: None
void STREAM_DATA(uint16_t accelXData, uint16_t accelYData, uint16_t accelZData){
	OUT_CHAR(0x03);
	OUT_CHAR((uint8_t)(accelXData));
	OUT_CHAR((uint8_t)(accelXData >> 8));
	OUT_CHAR((uint8_t)(accelYData));
	OUT_CHAR((uint8_t)(accelYData >> 8));
	OUT_CHAR((uint8_t)(accelZData));
	OUT_CHAR((uint8_t)(accelZData >> 8));
	OUT_CHAR(0xfc);
	
	return;
}
```

Lab6f.c:

```
// Lab 6 part C
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:

#include <avr/io.h>
#include <avr/interrupt.h>
#include "roboticsBackPack.h"
#include "LSM330.h"

#define CLK_PRESCALER       CLK_PSADIV_1_gc

typedef enum{
       FALSE,
       TRUE
}bool;

void INIT_CLK(void);
void INIT_INTS(void);
void INIT_USART(void);
void OUT_CHAR(char);
void OUT_STRING(char*);
void STREAM_DATA(uint16_t,uint16_t,uint16_t);

volatile bool accelDataReady = FALSE;



int main(void)
{
       // Init CLK, pin dir, and ADC.
       volatile uint16_t accelXData;
       volatile uint16_t accelYData;
       volatile uint16_t accelZData;

       INIT_CLK();
       INIT_USART();
       INIT_INTS();
       accel_init();

       while (1){
              if(accelDataReady){
                     accelXData =  (accelRead(OUT_X_H_A) << 8) | accelRead(OUT_X_L_A);
                     accelYData =  (accelRead(OUT_Y_H_A) << 8) | accelRead(OUT_Y_L_A);
                     accelZData =  (accelRead(OUT_Z_H_A) << 8) | accelRead(OUT_Z_L_A);

                     STREAM_DATA(accelXData, accelYData, accelZData);
                     accelDataReady = FALSE;
              }
       }
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
       // Enable 32Mhz CLK.
       OSC_CTRL = OSC_RC32MEN_bm;

       // Wait for 32Mhz flag to be set.
       while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

       // Write to restriction register to allow writing
       // to the CLK CTRL, then sel the 32MHz CLK.
       CPU_CCP = CCP_IOREG_gc;
       CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

       // Write to restriction register to allow writing
       // to the CLK PSCTRL, then set the prescaler.
       CPU_CCP = CCP_IOREG_gc;
       CLK_PSCTRL = CLK_PRESCALER;
}


/**********************FUNCTIONS***************************************/
// Subroutine Name: INIT_USART
// Init the USART regs.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_USART(void){
       // Set the direction of the Tx & Rx pins.
       PORTD_DIRSET = USART_TXEN_bm;
       PORTD_DIRCLR = USART_RXEN_bm;

       // Set the baud rate.
       USARTD0_BAUDCTRLA = 1;
       USARTD0_BAUDCTRLB = 0;

       // Set the data size and the mode.
       USARTD0_CTRLC = USART_CHSIZE_8BIT_gc;
       USARTD0_CTRLB = 0b00011000;
}


/**********************FUNCTIONS***************************************/
// Subroutine Name: OUT_CHAR
// Send a char over USART.
// Inputs: char c = input char
// Outputs: None
// Affected: None
void OUT_CHAR(char c){
       // Wait for the data buffer to be ready for
       // input, load the character, finally wait until
       // transmission is complete.
       while(!(USARTD0_STATUS & USART_DREIF_bm));
       USARTD0_DATA = c;
       while(!(USARTD0_STATUS & USART_TXCIF_bm));

       return;
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: OUT_STRING
// Send a string over USART.
// Inputs: char* s = input string
// Outputs: None
// Affected: None
void OUT_STRING(char* s){
       // Wait for the data buffer to be ready for
       // input, load the character, finally wait until
       // transmission is complete. Loop until the
       // NULL terminator is reached.
       for(uint8_t i=0; s[i]!='\0'; i++){
              while(!(USARTD0_STATUS & USART_DREIF_bm));
              USARTD0_DATA = s[i];
              while(!(USARTD0_STATUS & USART_TXCIF_bm));
       }

       return;
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: INIT_INTS
// Init interrupts.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_INTS(void){
       // Set the PMIC to enable low level interrupts.
       PMIC_CTRL = PMIC_LOLVLEN_bm;

       // Set the interrupt enable bit.
       CPU_SREG |= 0x80;
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: STREAM_DATA
// Send axis data over UART.
// Inputs: accelXData, accelYData, accelZData
// Outputs: None
// Affected: None
void STREAM_DATA(uint16_t accelXData, uint16_t accelYData, uint16_t accelZData){
       OUT_CHAR(0x03);
       OUT_CHAR((uint8_t)(accelXData));
       OUT_CHAR((uint8_t)(accelXData >> 8));
       OUT_CHAR((uint8_t)(accelYData));
       OUT_CHAR((uint8_t)(accelYData >> 8));
       OUT_CHAR((uint8_t)(accelZData));
       OUT_CHAR((uint8_t)(accelZData >> 8));
       OUT_CHAR(0xfc);

       return;
}

/*********************INTERUPT****************************************/
// Sets up an interrupt to be triggered by rising edge on port C's pin 7.
//
```

```c
// Inputs: None
// Outputs: None
// Affected: None
ISR(PORTC_INT0_vect){
	accelDataReady = TRUE;
	return;
}
```

Lab6g.c:

```
// Lab 6 part G
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:

#include <avr/io.h>
#include <avr/interrupt.h>
#include "roboticsBackPack.h"
#include "LSM330.h"

#define CLK_PRESCALER       CLK_PSADIV_1_gc

typedef enum{
       FALSE,
       TRUE
}bool;

void INIT_CLK(void);
void INIT_RGB(void);
void INIT_INTS(void);
void setRGB(uint16_t, uint16_t, uint16_t);


volatile bool accelDataReady = FALSE;



int main(void)
{
       // Init CLK, pin dir, and ADC.
       volatile uint16_t accelXData;
       volatile uint16_t accelYData;
       volatile uint16_t accelZData;

       INIT_CLK();
       INIT_RGB();
       INIT_INTS();
       accel_init();

       while (1){
              if(accelDataReady){
                     uint8_t accelXLData = accelRead(OUT_X_L_A);
                     uint8_t accelXHData = accelRead(OUT_X_H_A);
                     accelXData = (accelXHData << 8) | accelXLData;
                     uint8_t accelYLData = accelRead(OUT_Y_L_A);
                     uint8_t accelYHData = accelRead(OUT_Y_H_A);
                     accelYData = (accelYHData << 8) | accelYLData;
                     uint8_t accelZLData = accelRead(OUT_Z_L_A);
                     uint8_t accelZHData = accelRead(OUT_Z_H_A);
                     accelZData = (accelZHData << 8) | accelZLData;
                     accelXData =  (accelXData < 0) ? (0-accelXData) : accelXData;
                     accelYData =  (accelYData < 0) ? (0-accelYData) : accelYData;
                     accelZData =  (accelZData < 0) ? (0-accelZData) : accelZData;

                     setRGB(accelXData, accelYData, accelZData);
```

```
            accelDataReady = FALSE;
        }
    }
}

/*********************FUNCTIONS***************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
    // Enable 32Mhz CLK.
    OSC_CTRL = OSC_RC32MEN_bm;

    // Wait for 32Mhz flag to be set.
    while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

    // Write to restriction register to allow writing
    // to the CLK CTRL, then sel the 32MHz CLK.
    CPU_CCP = CCP_IOREG_gc;
    CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

    // Write to restriction register to allow writing
    // to the CLK PSCTRL, then set the prescaler.
    CPU_CCP = CCP_IOREG_gc;
    CLK_PSCTRL = CLK_PRESCALER;
}


/*********************FUNCTIONS***************************************/
// Subroutine Name: INIT_RGB
// Init the RGB LED.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_RGB(void){
    PORTD_DIRSET  = 0b01110000;
    PORTD_REMAP   = 0b00000111;

    // Value to set the prescaler of the TC to be 1 times the sys CLK.
    TCD0_CTRLA    = 0b00000001;

    // Sets the PWM mode of the TC to single slope.
    TCD0_CTRLB           = 0b01110011;
    TCD0_PER             = 0xFFFF;
    PORTD_OUTSET  = 0x00;

}


/*********************FUNCTIONS***************************************/
// Subroutine Name: setRGB
// Sets the value of the RGB LED.
// Inputs: None
// Outputs: None
// Affected: None
void setRGB(uint16_t redVal, uint16_t greenVal, uint16_t blueVal){
```

```c
       TCD0_CCA = ~redVal;
       TCD0_CCB = ~greenVal;
       TCD0_CCC = ~blueVal;
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: INIT_INTS
// Init interrupts.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_INTS(void){
       // Set the PMIC to enable low level interrupts.
       PMIC_CTRL = PMIC_LOLVLEN_bm;

       // Set the interrupt enable bit.
       CPU_SREG |= 0x80;
}


/*********************INTERUPT****************************************/
// Sets up an interrupt to be triggered by rising edge on port C's pin 7.
//
// Inputs: None
// Outputs: None
// Affected: None
ISR(PORTC_INT0_vect){
       accelDataReady = TRUE;
       return;
}
```

Spi.c:

```c
// spi.c
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:       Definitions of SPI functions.


 #include "spi.h"

 /**********************FUNCTIONS***************************************/
 // Subroutine Name: spi_init
 // Configure the SPI peripheral
 // Inputs: None
 // Outputs: None
 // Affected: None
 void spi_init(void){

        PORTF_DIRSET = 0b10111100;
        PORTF_DIRCLR = 0b01000011;
        // Master, MSB DORD, LE:RS, TE:FS, SYS_CLK/4 = 8MHz
        SPIF_CTRL = SPI_ENABLE_bm | SPI_MASTER_bm | SPI_MODE_0_gc |
SPI_PRESCALER_DIV4_gc;

        return;
 }

 /**********************FUNCTIONS***************************************/
 // Subroutine Name: spiWrite
 // Write a byte of data using SPI
 // Inputs: data - data to be sent
 // Outputs: None
 // Affected: None
 uint8_t spiWrite(uint8_t data){
        SPIF_DATA = data;
        while(!(SPIF_STATUS & SPI_IF_bm));
        return SPIF_DATA;
 }

 /**********************FUNCTIONS***************************************/
 // Subroutine Name: spiRead
 // Reads the return value of spiWrite.
 // Inputs: None
 // Outputs: Return value of spiWrite.
 // Affected: None
 uint8_t spiRead(void){
        return  spiWrite(0xFF);
 }
```

Spi.h:

```c
// spi.h
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:       Declarations of SPI functions.
#include <avr/io.h>

#ifndef SPI_H_
#define SPI_H_

void spi_init(void);
uint8_t spiWrite(uint8_t data);
uint8_t spiRead(void);


#endif /* SPI_H_ */
```

roboticsBackpack.c:

```c
// roboticsBackpack.c
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:       Definitions of robotics backpack functions.

#include <avr/io.h>
#include "LSM330.h"
#include "spi.h"
#include "roboticsBackpack.h"

/**********************FUNCTIONS***************************************/
// Subroutine Name: accel_init
// Init the accelerometer.
// Inputs: None
// Outputs: None
// Affected: None
void accel_init(void){
       PORTA_DIRSET = RBB_PROTOCOL_SEL;
       PORTA_OUTCLR = RBB_PROTOCOL_SEL;
       PORTF_OUTSET = RBB_SSA | RBB_SSG | RBB_SENSOR_SEL_ACCEl_bp;
       spi_init();
       accelWrite(CTRL_REG4_A, CTRL_REG4_A_STRT);
       PORTC_DIRCLR = RBB_INT1A;
       PORTC_PIN7CTRL = PORT_ISC_FALLING_gc;
       PORTC_INT0MASK = RBB_INT1A;
       PORTC_INTCTRL = PORT_INT0LVL_LO_gc;
       uint8_t reg4AInitData = CTRL_REG4_A_DR_EN | CTRL_REG4_A_IEA | CTRL_REG4_A_INT1_EN;
       accelWrite(CTRL_REG4_A, reg4AInitData);
       uint8_t reg5AInitData = CTRL_REG5_A_ODR3 | CTRL_REG5_A_ODR0 | CTRL_REG5_A_ZEN |
CTRL_REG5_A_YEN | CTRL_REG5_A_XEN;
       accelWrite(CTRL_REG5_A, reg5AInitData);


}

/**********************FUNCTIONS***************************************/
// Subroutine Name: accelRead
// Read a byte from the accelerometer.
// Inputs: None
// Outputs: Byte read from accelerometer.
// Affected: None
uint8_t accelRead(uint8_t targetReg){
       uint8_t spiVal = 0x80 | targetReg;
       uint8_t result;
       PORTF_OUTCLR = RBB_SSA;
       spiWrite(spiVal);
       result = spiRead();
       PORTF_OUTSET = RBB_SSA;
       return result;
}

/**********************FUNCTIONS***************************************/
// Subroutine Name: accelWrite
// Write a byte to the accelerometer.
// Inputs: Byte to be written.
// Outputs: None
// Affected: None
void accelWrite(uint8_t targetReg, uint8_t data){
	uint8_t spiVal = 0x00 | targetReg;
	PORTF_OUTCLR = RBB_SSA;
	spiWrite(spiVal);
	spiWrite(data);
	PORTF_OUTSET = RBB_SSA;
}

/**********************FUNCTIONS****************************************/
// Subroutine Name: gyroRead
// Read a byte from the gyroscope.
// Inputs: None
// Outputs: Byte read from gyroscope.
// Affected: None
uint8_t gyroRead(uint8_t targetReg){
	uint8_t spiVal = 0x80 | targetReg;
	uint8_t result;
	PORTF_OUTCLR = RBB_SSG;
	spiWrite(spiVal);
	result = spiRead();
	PORTF_OUTSET = RBB_SSG;
	return result;
}

/**********************FUNCTIONS****************************************/
// Subroutine Name: accelWrite
// Write a byte to the gyroscope.
// Inputs: Byte to be written.
// Outputs: None
// Affected: None
void gyroWrite(uint8_t targetReg, uint8_t data){
	uint8_t spiVal = 0x00 | targetReg;
	PORTF_OUTCLR = RBB_SSG;
	spiWrite(spiVal);
	spiWrite(data);
	PORTF_OUTSET = RBB_SSG;
}
```

roboticsBackpack.h:

```
// roboticsBackpack.h
// Name:          Mark L. Schuster
// Section #:     1540
// TA Name:       Christopher Crary
// Description:        Declarations of robotics backpack functions.
#include <avr/io.h>

#ifndef ROBOTICSBACKPACK_H_
#define ROBOTICSBACKPACK_H_

//     PORTF CONTROL SIGNALS
#define RBB_SDA                                              (0x01<<0)
#define RBB_SCL                                              (0x01<<1)
#define RBB_SENSOR_SEL                                (0x01<<2)
#define RBB_SSA                                              (0x01<<3)
#define RBB_SSG                                              (0x01<<4)
#define RBB_MOSI                                      (0x01<<5)
#define RBB_MISO                                      (0x01<<6)
#define RBB_SCK                                              (0x01<<7)

//     PORTC CONTROL SIGNALS
#define RBB_PWMA                                      (0x01<<0)
#define RBB_PWMB                                      (0x01<<1)
#define RBB_AIN2                                      (0x01<<2)
#define RBB_AIN1                                      (0x01<<3)
#define RBB_BIN2                                      (0x01<<4)
#define RBB_BIN1                                      (0x01<<5)
#define RBB_INT2A                                     (0x01<<6)
#define RBB_INT1A                                     (0x01<<7)

//     PORTA CONTROL SIGNALS
#define RBB_STBY                                      (0x01<<0)
#define RBB_INT2G                                     (0x01<<1)
#define RBB_INT1G                                     (0x01<<2)
#define RBB_GYRO_ENABLE                               (0x01<<3)
#define RBB_PROTOCOL_SEL                       (0x01<<4)
#define RBB_PA5                                              (0x01<<5)
#define RBB_PA6                                              (0x01<<6)
#define RBB_PA7                                              (0x01<<7)



#define RBB_SENSOR_SEL_GYRO_bm                         0
#define RBB_SENSOR_SEL_GYRO_bp                         (0x01<<2)
#define RBB_SENSOR_SEL_ACCEl_bm                        1
#define RBB_SENSOR_SEL_ACCEl_bp                        (0x01<<2)

#define RBB_PROTOCOL_SEL_SPI_bm                        0
#define RBB_PROTOCOL_SEL_SPI_bp                        (0x01<<4)
#define RBB_PROTOCOL_SEL_I2C_bm                        1
#define RBB_PROTOCOL_SEL_I2C_bp                        (0x01<<4)


// FUNCTIONS
void accel_init(void);
uint8_t accelRead(uint8_t);
```

```
void accelWrite(uint8_t, uint8_t);
uint8_t gyroRead(uint8_t);
void gyroWrite(uint8_t, uint8_t);



#endif /* ROBOTICSBACKPACK_H_ */
```

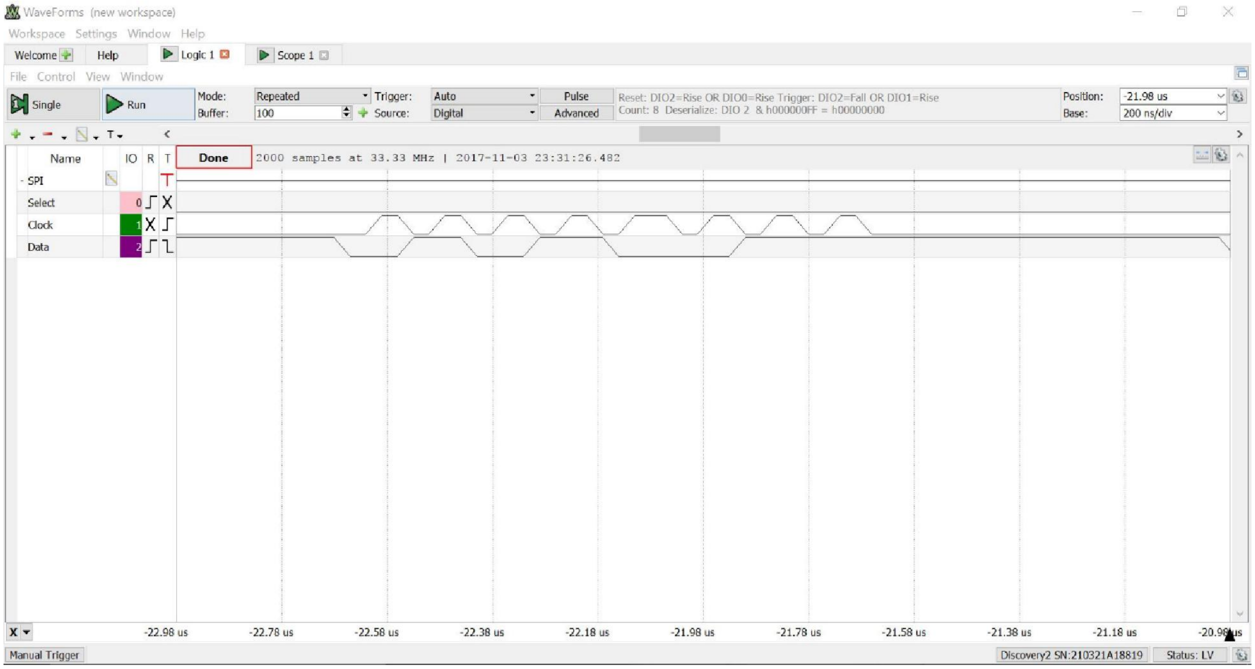
H) Appendix:

Files:

- Lab6.pdf
- Lab6b.c
- Lab6f.c
- Lab6g.c
- spi.c
- spi.h
- roboticsBackpack.c
- roboticsBackpack.h

Screenshots:

Spi frame:



Mode: Repeated
Buffer: 100
Trigger: Auto
Source: Digital

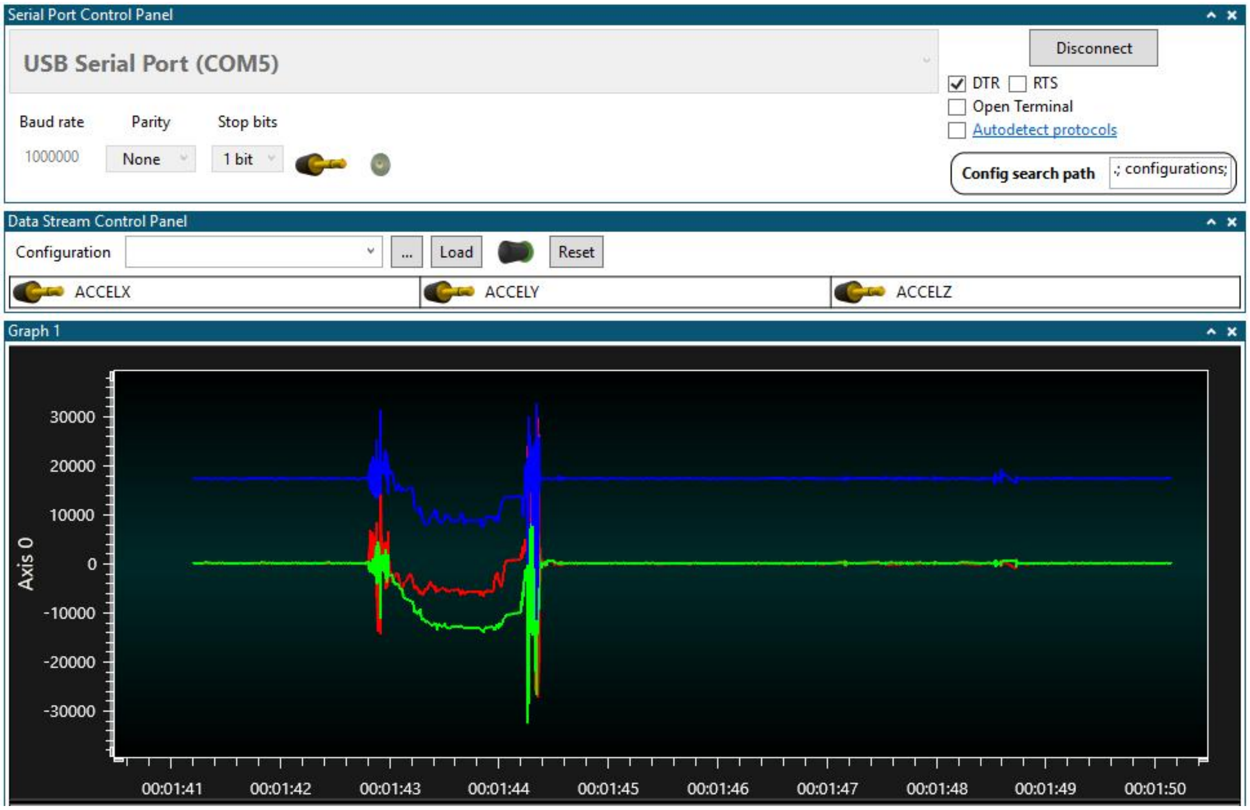
Reset: DIO2=Rise OR DIO0=Rise Trigger: DIO2=Fall OR DIO1=Rise
Count: 8 Deserialize: DIO 2 & h000000FF = h00000000

Position: -21.98 us
Base: 200 ns/div

Done 2000 samples at 33.33 MHz | 2017-11-03 23:31:26.482

Discovery2 SN:210321A18819 Status: LV

Accelerometer graph:



Serial Port Control Panel

USB Serial Port (COM5)

Baud rate 1000000

Parity None

Stop bits 1 bit

☑ DTR ☐ RTS

☐ Open Terminal

☐ Autodetect protocols

Config search path ;configurations;

Data Stream Control Panel

Configuration