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. 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.
5. 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:

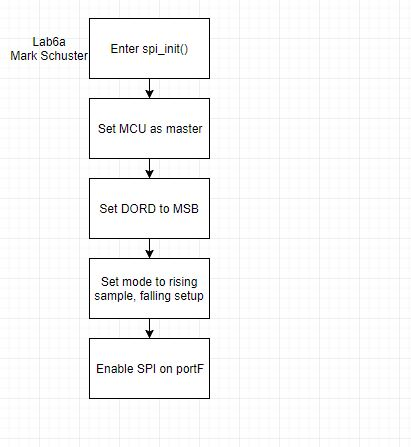
I enjoyed working with SPI and I look forward to using it in high throughput applications.

**E)** Schematics:

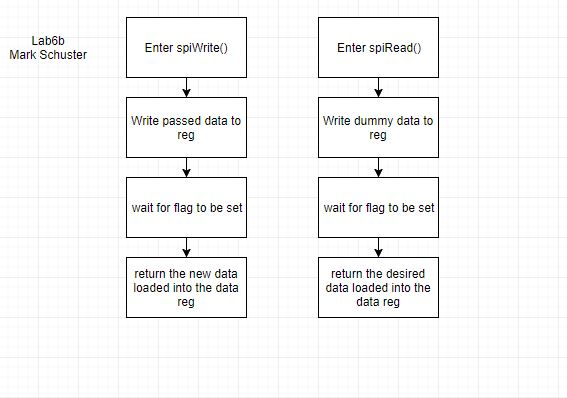
Not applicable for this lab.

**F)** Pseudocode/Flowcharts:

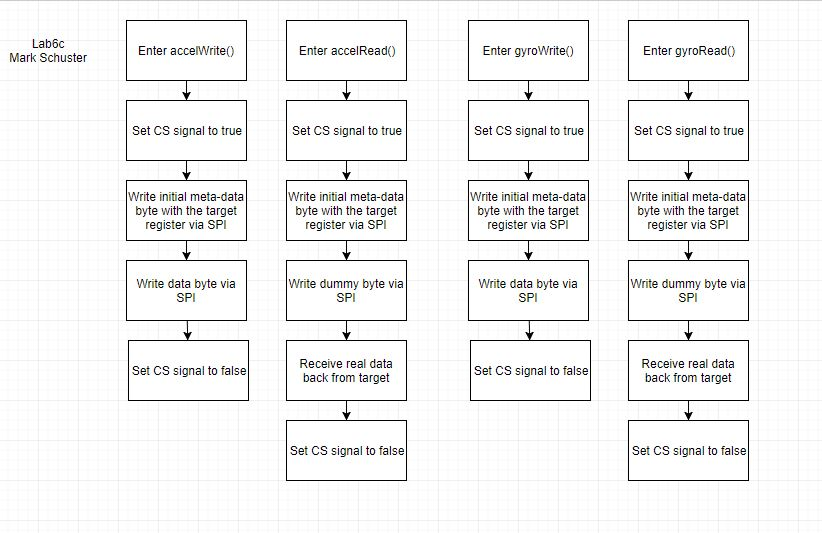
Lab6a flow diagram:



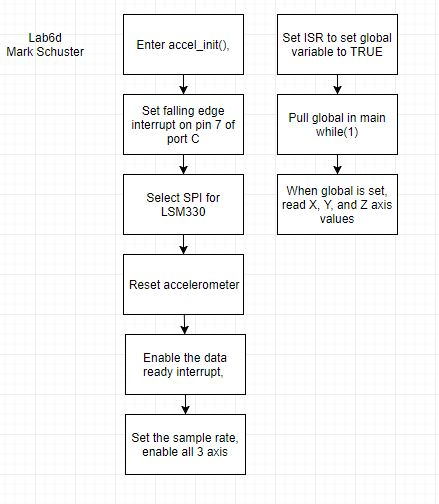
Lab6b flow diagram:



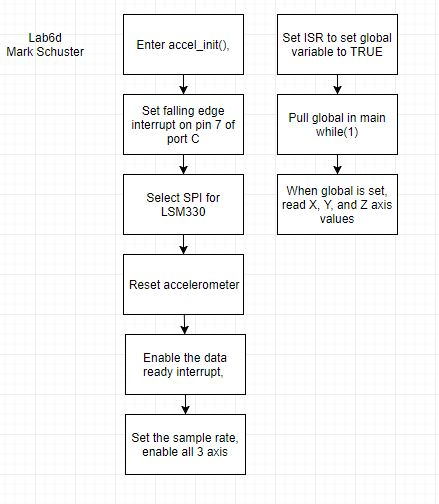
Lab6c flow diagram:



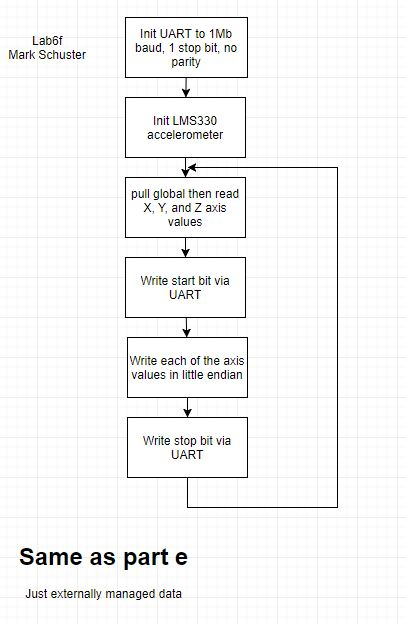
Lab6d flow diagram:

****

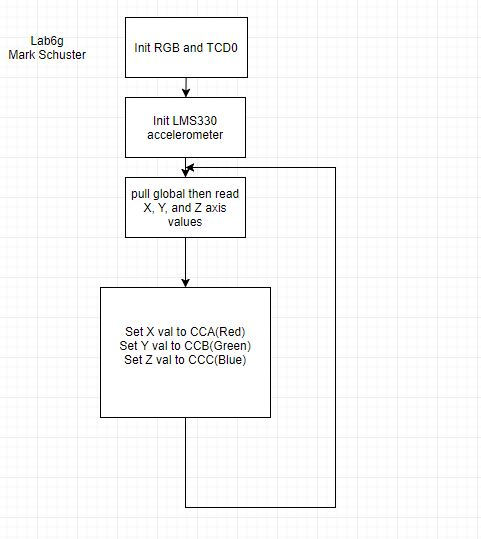
Lab6e flow diagram:



Lab6f flow diagram:



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:**

// 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():**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*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():**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*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.

//

// 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){

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:**

// 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:**

// 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:**

// 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\_ \*/

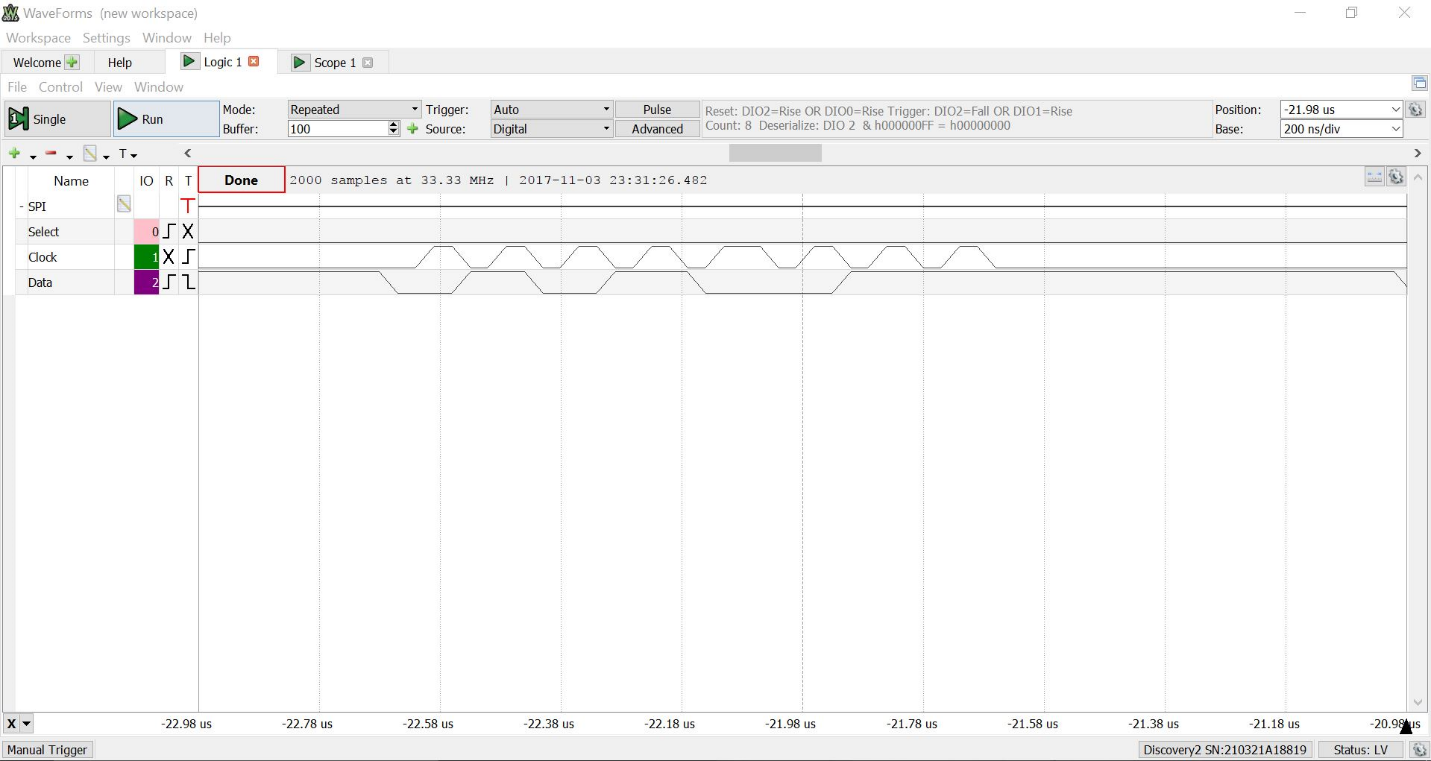
**H)** Appendix:

Files:

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

Screenshots:

Spi frame:



Accelerometer graph:

