PART A

#include <avr/io.h>

void CLK\_32MHZ(void);

void USART(void);

void OUT\_CHAR(*uint8\_t* data);

*uint8\_t* IN\_CHAR(void);

void OUT\_STRING(volatile *uint8\_t*\* data); //pointing the point at the first address. we have to pass in the address

//without the dereferencing mark

#define BSELHIGH (((4)\*((32000000/(16\*57600))-1))>>8) //bscale of -2

#define BSEL ((4)\*((32000000/(16\*57600))-1)) //bscale of -2

volatile *uint8\_t* name[]="Pengzhao Zhu";

int main(void)

{

CLK\_32MHZ();

USART();

// OUT\_STRING(name);

*uint8\_t* a='U'; //pointer to point to address of "U"

while(1) {

OUT\_CHAR(a); //retrieve the data from that address

}

}

void USART(void)

{

PORTD\_DIRSET=0x08; //set transmitter as output

PORTD\_DIRCLR=0X04; //set receiver as input

USARTD0\_CTRLB=0x18; //enable receiver and transmitter

USARTD0\_CTRLC= 0X33; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

USARTD0\_BAUDCTRLA= (*uint8\_t*) BSEL; //load lowest 8 bits of BSEL

USARTD0\_BAUDCTRLB= (((*uint8\_t*) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits of BSEL. bitwise OR them

PORTD\_OUTSET= 0x08; //set transmit pin idle

}

void CLK\_32MHZ(void)

{

volatile *uint8\_t* \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((\*p) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void OUT\_CHAR(volatile *uint8\_t* data) {

volatile *uint8\_t* \*p=&USARTD0\_STATUS; //load the status flag data

while( ((\*p) & 0x20) != 0x20); //keep looping if DREIF flag is not set

USARTD0\_DATA= (*uint8\_t*) data;

}

void OUT\_STRING(volatile *uint8\_t*\* data) { //pointing the pointer at that address

for (int i=0; data[i]!=0x00; i++) { //go through the whole string except the null terminator

OUT\_CHAR((*uint8\_t*) data[i]); //output the value

}

/\*

while(\*data != 0) //dereferencing

{

OUT\_CHAR((uint8\_t)\*data); //output the value

data++;

} \*/

}

*uint8\_t* IN\_CHAR(void) {

volatile *uint8\_t* \*p=&USARTD0\_STATUS; //load the status flag data

while( ((\*p) & 0x80) != 0x80); //keep looping if DREIF flag is not set

return USARTD0\_DATA;

}

Part A part B

#include <avr/io.h>

#include "LSM.h"

void CLK\_32MHZ(void);

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

int main(void){

while(1) {

}

};

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011111; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=11

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

}

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //enable slave(accelerometer) device by setting it low

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

Part C

#include <avr/io.h>

#include "LSM.h"

void CLK\_32MHZ(void);

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

while(1){

SPI\_WRITE(0x53);

}

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011111; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=11

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

}

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //enable slave(accelerometer) device by setting it low

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

Part D

#include <avr/io.h>

#include "LSM.h"

void CLK\_32MHZ(void);

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data);

*uint8\_t* ACCEL\_READ(*uint8\_t* addr);

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

*uint8\_t* hello;

hello=ACCEL\_READ(WHO\_AM\_I\_A);

while(1);

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011111; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=11

PORTA\_DIRSET=0x10; //set PROTOCOL\_SEL as output

PORTA\_OUTCLR=0x10; //clear PROTOCOL\_SEL to configure it as SPI. I2C is when i set it.

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

}

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it low

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr= addr & 0b00111111 ; //RW is always 0 (write) and MS is always 0

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 0). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

SPIF\_DATA=data; //write the actual data

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

}

*uint8\_t* ACCEL\_READ(*uint8\_t* addr){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr=addr | 0b10000000; //bitwise OR so RW (bit 7) is always 1 (Read). Gotta be careful of the MS signal

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 1). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

*uint8\_t* hi=SPI\_READ();

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

return hi; //data read from the ACCEL register

}

PART D REPEAT

#include <avr/io.h>

#include "LSM.h"

void CLK\_32MHZ(void);

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data);

*uint8\_t* ACCEL\_READ(*uint8\_t* addr);

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

*uint8\_t* hello;

hello=ACCEL\_READ(WHO\_AM\_I\_A);

while(1);

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011111; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=11

PORTA\_DIRSET=0x10; //set PROTOCOL\_SEL as output

PORTA\_OUTCLR=0x10; //clear PROTOCOL\_SEL to configure it as SPI. I2C is when i set it.

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

}

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it low

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr= addr & 0b00111111 ; //RW is always 0 (write) and MS is always 0

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 0). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

SPIF\_DATA=data; //write the actual data

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

}

*uint8\_t* ACCEL\_READ(*uint8\_t* addr){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr=addr | 0b10000000; //bitwise OR so RW (bit 7) is always 1 (Read). Gotta be careful of the MS signal

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 1). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

*uint8\_t* hi=SPI\_READ();

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

return hi; //data read from the ACCEL register

}

PARTF

#include <avr/io.h>

#include <avr/interrupt.h>

#include "LSM.h"

void CLK\_32MHZ(void);

//SPI

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data);

*int8\_t* ACCEL\_READ(*uint8\_t* addr);

void ACCEL\_INIT(void);

//USART

void USARTD0\_init(void);

void OUT\_CHAR(*int8\_t* data);

*uint8\_t* IN\_CHAR(void);

void OUT\_STRING(*uint8\_t*\* data); //pointing the point at the first address. we have to pass in the address

//without the dereferencing mark

#define BSELHIGH (((4)\*((32000000/(16\*1000000))-1))>>8) //bscale of -2

#define BSEL ((4)\*((32000000/(16\*1000000))-1)) //bscale of -2

volatile *uint8\_t* intbit;

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

ACCEL\_INIT(); //call function to initialize accelerometer

USARTD0\_init(); //call function to initialize USART system

*int8\_t* XL;

*int8\_t* XH;

*int8\_t* YL;

*int8\_t* YH;

*int8\_t* ZL;

*int8\_t* ZH;

while(1) {

while(intbit != 1); //keep checking if the interrupt is set

XL= ACCEL\_READ (OUT\_X\_L\_A); //read measurements from accelerometer

XH= ACCEL\_READ (OUT\_X\_H\_A);

YL= ACCEL\_READ(OUT\_Y\_L\_A);

YH= ACCEL\_READ(OUT\_Y\_H\_A);

ZL= ACCEL\_READ(OUT\_Z\_L\_A);

ZH= ACCEL\_READ(OUT\_Z\_H\_A);

OUT\_CHAR(0x03); //start byte

OUT\_CHAR(XL);

OUT\_CHAR(XH);

OUT\_CHAR(YL);

OUT\_CHAR(YH);

OUT\_CHAR(ZL);

OUT\_CHAR(ZH);

OUT\_CHAR(0xFC); //end byte. inverse of start byte. One's complement

intbit=0; //set the bit to zero. until the ISR to change the intbit to 1 to output data to data stream

}

return 0;

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011111; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=11

PORTA\_DIRSET=0x10; //set PROTOCOL\_SEL as output

PORTA\_OUTCLR=0x10; //clear PROTOCOL\_SEL to configure it as SPI. I2C is when i set it.

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

}

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it low

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr= addr & 0b00111111 ; //RW is always 0 (write) and MS is always 0

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 0). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

SPIF\_DATA=data; //write the actual data

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

}

*int8\_t* ACCEL\_READ(*uint8\_t* addr){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr=addr | 0b10000000; //bitwise OR so RW (bit 7) is always 1 (Read). Gotta be careful of the MS signal

SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 1). second bit=MS=0

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

*uint8\_t* hi=SPI\_READ();

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

return hi; //data read from the ACCEL register

}

void ACCEL\_INIT(void){

PORTC\_INTCTRL=0x01; //enable low level interrupt

PORTC\_INT0MASK=0x80; //set pin 7 on C as source for interrupt

PORTC\_DIRCLR=0x80; //set pin 7 as input

PORTC\_PIN7CTRL=0x01; //rising edge trigger

PMIC\_CTRL=0x01;// enable low level interrupt in the PMIC

sei(); //enable global interrupt flag

ACCEL\_WRITE(CTRL\_REG2\_A, 0x01); //resetting the LSM system

ACCEL\_WRITE(CTRL\_REG2\_A,0b11101000); //data routed to to INT\_A, interrupt signal active high, edge triggered, INT1\_A signal enable

ACCEL\_WRITE(CTRL\_REG5\_A,0b10010111); //fastest output rate, BDU continous update, X Y Z enabled

} //also enabled PORT C pin 7 interrupt in the XMEGA

void USARTD0\_init(void)

{

PORTD\_DIRSET=0x08; //set transmitter as output

PORTD\_DIRCLR=0X04; //set receiver as input

USARTD0\_CTRLB=0x18; //enable receiver and transmitter

USARTD0\_CTRLC= 0X33; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

USARTD0\_BAUDCTRLA= (*uint8\_t*) BSEL; //load lowest 8 bits of BSEL

USARTD0\_BAUDCTRLB= (((*uint8\_t*) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits of BSEL. bitwise OR them

PORTD\_OUTSET= 0x08; //set transit pin idle

}

void OUT\_CHAR(*int8\_t* data) { //changed it to 8 bit sign for accelerometer

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( ((USARTD0\_STATUS) & 0x20) != 0x20); //keep looping if DREIF flag is not set

USARTD0\_DATA= data;

}

void OUT\_STRING(*uint8\_t*\* data) { //pointing the pointer at that address

for (int i=0; data[i]!=0x00; i++) { //go through the whole string except the null terminator

OUT\_CHAR((*uint8\_t*) data[i]); //output the value

}

/\*

while(\*data != 0) //dereferencing

{

OUT\_CHAR((uint8\_t)\*data); //output the value

data++;

} \*/

}

*uint8\_t* IN\_CHAR(void) {

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( (USARTD0\_STATUS & 0x80) != 0x80); //keep looping if DREIF flag is not set

return USARTD0\_DATA;

}

ISR(PORTC\_INT0\_vect) {

*uint8\_t* status=CPU\_SREG; //push status register

PORTC\_INTFLAGS=0x01 ; //clear the interrupt flag

intbit=1; //change intbit to 1 so we can read and transmit measured data from the accelerometer

CPU\_SREG= status; //pop the status register

}

PART F BACKUP AGAIN

#include <avr/io.h>

#include <avr/interrupt.h>

#include "LSM.h"

void CLK\_32MHZ(void);

//SPI

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data);

*uint8\_t* ACCEL\_READ(*uint8\_t* addr);

void ACCEL\_INIT(void);

//USART

void USARTD0\_init(void);

void OUT\_CHAR(*int8\_t* data);

*uint8\_t* IN\_CHAR(void);

void OUT\_STRING(*uint8\_t*\* data); //pointing the point at the first address. we have to pass in the address

//without the dereferencing mark

//void transmit(uint8\_t XL, uint8\_t XH, uint8\_t YL, uint8\_t YH, uint8\_t ZL, uint8\_t ZH);

#define BSELHIGH (((4)\*((32000000/(16\*1000000))-1))>>8) //bscale of -2

#define BSEL ((4)\*((32000000/(16\*1000000))-1)) //bscale of -2

volatile *uint8\_t* intbit;

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

ACCEL\_INIT(); //call function to initialize accelerometer

USARTD0\_init(); //call function to initialize USART system

*uint8\_t* XL;

*uint8\_t* XH;

*uint8\_t* YL;

*uint8\_t* YH;

*uint8\_t* ZL;

*uint8\_t* ZH;

while(1) {

while(intbit != 1); //keep checking if the interrupt is set

XL= ACCEL\_READ (OUT\_X\_L\_A); //read measurements from accelerometer

XH= ACCEL\_READ (OUT\_X\_H\_A);

YL= ACCEL\_READ(OUT\_Y\_L\_A);

YH= ACCEL\_READ(OUT\_Y\_H\_A);

ZL= ACCEL\_READ(OUT\_Z\_L\_A);

ZH= ACCEL\_READ(OUT\_Z\_H\_A);

//transmit(XL,XH,YL,YH,ZL,ZH);

OUT\_CHAR(0x03); //start byte

OUT\_CHAR(XL);

OUT\_CHAR(XH);

OUT\_CHAR(YL);

OUT\_CHAR(YH);

OUT\_CHAR(ZL);

OUT\_CHAR(ZH);

OUT\_CHAR(0xFC); //end byte. inverse of start byte. One's complement

intbit=0; //set the bit to zero. until the ISR to change the intbit to 1 to output data to data stream

}

return 0;

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why the fuck do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011100; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=00

PORTA\_DIRSET=0x10; //set PROTOCOL\_SEL as output

PORTA\_OUTCLR=0x10; //clear PROTOCOL\_SEL to configure it as SPI. I2C is when i set it.

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

} //GOOD

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

//PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. gotta take it out for ACCEL WRITE

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

//PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it low. gotta take it out for ACCEL WRITE

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr= addr & 0b00111111 ; //RW is always 0 (write) and MS is always 0

SPI\_WRITE(addr);

SPI\_WRITE(data);

//SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 0). second bit=MS=0

//while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

//SPIF\_DATA=data; //write the actual data

//while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

}

*uint8\_t* ACCEL\_READ(*uint8\_t* addr){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr=addr | 0b10000000; //bitwise OR so RW (bit 7) is always 1 (Read). Gotta be careful of the MS signal

SPI\_WRITE(addr);

//SPIF\_DATA=addr; //writing the address byte. MSB bit is RW, Write=0, read=1 (need to be 1). second bit=MS=0

//while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

*uint8\_t* hi=SPI\_READ();

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

return hi; //data read from the ACCEL register

}

void ACCEL\_INIT(void){

PORTC\_INTCTRL=0x01; //enable low level interrupt

PORTC\_INT0MASK=0x80; //set pin 7 on C as source for interrupt

PORTC\_DIRCLR=0x80; //set pin 7 as input

PORTC\_PIN7CTRL=0x01; //rising edge trigger

PMIC\_CTRL=0x01;// enable low level interrupt in the PMIC

sei(); //enable global interrupt flag

ACCEL\_WRITE(CTRL\_REG2\_A, 0x01); //resetting the LSM system

ACCEL\_WRITE(CTRL\_REG2\_A,0b11001000); //data routed to to INT\_A, interrupt signal active high, edge triggered, INT1\_A signal enable 0b11101000

ACCEL\_WRITE(CTRL\_REG5\_A,0b10010111); //fastest output rate, BDU continous update, X Y Z enabled

} //also enabled PORT C pin 7 interrupt in the XMEGA

void USARTD0\_init(void)

{

PORTD\_DIRSET=0x08; //set transmitter as output

PORTD\_DIRCLR=0X04; //set receiver as input

USARTD0\_CTRLB=0x18; //enable receiver and transmitter

USARTD0\_CTRLC= 0x03; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

USARTD0\_BAUDCTRLA= (*uint8\_t*) BSEL; //load lowest 8 bits of BSEL

USARTD0\_BAUDCTRLB= (((*uint8\_t*) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits of BSEL. bitwise OR them

PORTD\_OUTSET= 0x08; //set transit pin idle

}

void OUT\_CHAR(*int8\_t* data) { //changed it to 8 bit sign for accelerometer

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( ((USARTD0\_STATUS) & 0x20) != 0x20); //keep looping if DREIF flag is not set

USARTD0\_DATA= data;

}

void OUT\_STRING(*uint8\_t*\* data) { //pointing the pointer at that address

for (int i=0; data[i]!=0x00; i++) { //go through the whole string except the null terminator

OUT\_CHAR((*uint8\_t*) data[i]); //output the value

}

/\*

while(\*data != 0) //dereferencing

{

OUT\_CHAR((uint8\_t)\*data); //output the value

data++;

} \*/

}

*uint8\_t* IN\_CHAR(void) {

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( (USARTD0\_STATUS & 0x80) != 0x80); //keep looping if DREIF flag is not set

return USARTD0\_DATA;

}

ISR(PORTC\_INT0\_vect) {

*uint8\_t* status=CPU\_SREG; //push status register

PORTC\_INTFLAGS=0x01 ; //clear the interrupt flag

intbit=1; //change intbit to 1 so we can read and transmit measured data from the accelerometer

CPU\_SREG= status; //pop the status register

}

/\*

void transmit(uint8\_t XL, uint8\_t XH, uint8\_t YL, uint8\_t YH, uint8\_t ZL, uint8\_t ZH) {

OUT\_CHAR(0x03); //start byte

OUT\_CHAR(XL);

OUT\_CHAR(XH);

OUT\_CHAR(YL);

OUT\_CHAR(YH);

OUT\_CHAR(ZL);

OUT\_CHAR(ZH);

OUT\_CHAR(0xFC); //end byte. inverse of start byte. One's complement

}

\*/

Final Code Almost

#include <avr/io.h>

#include <avr/interrupt.h>

#include "LSM.h"

void CLK\_32MHZ(void);

//SPI

void SPI(void); //SPI Initialization function

*uint8\_t* SPI\_WRITE(*uint8\_t* data); //SPI write function. returns data written to the SPIF Data register

*uint8\_t* SPI\_READ(void) ; //read function to read from slave by writing junk data. return

//the two functions will be used separately?

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data);

*uint8\_t* ACCEL\_READ(*uint8\_t* addr);

void ACCEL\_INIT(void);

//USART

void USARTD0\_init(void);

void OUT\_CHAR(*uint8\_t* data);

*uint8\_t* IN\_CHAR(void);

void OUT\_STRING(*uint8\_t*\* data); //pointing the point at the first address. we have to pass in the address

//without the dereferencing mark

#define BSELHIGH (((4)\*((32000000/(16\*1000000))-1))>>8) //bscale of -2

#define BSEL ((4)\*((32000000/(16\*1000000))-1)) //bscale of -2

volatile *uint8\_t* intbit;

int main(void){

CLK\_32MHZ();

SPI(); //call function to initialize SPI

ACCEL\_INIT(); //call function to initialize accelerometer

USARTD0\_init(); //call function to initialize USART system

*uint8\_t* XL;

*uint8\_t* XH;

*uint8\_t* YL;

*uint8\_t* YH;

*uint8\_t* ZL;

*uint8\_t* ZH;

while(1) {

while(intbit != 1); //keep checking if the interrupt is set

XL= ACCEL\_READ (OUT\_X\_L\_A); //read measurements from accelerometer

XH= ACCEL\_READ (OUT\_X\_H\_A);

YL= ACCEL\_READ(OUT\_Y\_L\_A);

YH= ACCEL\_READ(OUT\_Y\_H\_A);

ZL= ACCEL\_READ(OUT\_Z\_L\_A);

ZH= ACCEL\_READ(OUT\_Z\_H\_A);

OUT\_CHAR(0x03); //start byte

OUT\_CHAR(XL);

OUT\_CHAR(XH);

OUT\_CHAR(YL);

OUT\_CHAR(YH);

OUT\_CHAR(ZL);

OUT\_CHAR(ZH);

OUT\_CHAR(0xFC); //end byte. inverse of start byte. One's complement

intbit=0; //set the bit to zero. until the ISR to change the intbit to 1 to output data to data stream

}

return 0;

}

void SPI(void){

PORTF\_DIRCLR= 0b01000000; //set MISO as input

PORTF\_DIRSET=0b10111100; //set as output. the 1011 is SCK (SPI) enable, MOSI (SPI), and SSG (gyroscope)

//why do I have to set the gyroscope as output?????

//the 1100 is low true SSA signal of accelerometer and Sensor\_sel of the mux (to accelerometer)

SPIF\_CTRL=0b01011100; // enable SPI (bit 6), MSB first(bit 5), master mode(bit 4), (falling setup, rising sample)=11, 32MHZ/64=00. changed

PORTA\_DIRSET=0x10; //set PROTOCOL\_SEL as output

PORTA\_OUTCLR=0x10; //clear PROTOCOL\_SEL to configure it as SPI. I2C is when i set it.

PORTF\_OUTSET=0b00011000; //set SSA and SSG high so it doesn't start. I will initialize in the write routine

} //GOOD

*uint8\_t* SPI\_WRITE(*uint8\_t* data){ //returns data written to the SPIF Data register

//PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. gotta take it out for ACCEL WRITE

SPIF\_DATA=data; //write a byte of data to DATA register

while((SPIF\_STATUS & 0x80) != 0x80); //keep looping until interrupt flag is set. Also act as step one (reading STATUS REGISTER)

//OF clearing the interrupt flag.

//PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it low. gotta take it out for ACCEL WRITE

return SPIF\_DATA;

}

*uint8\_t* SPI\_READ(void) { //read function to read from slave by writing junk data

return (SPI\_WRITE(0xFF));

}

void CLK\_32MHZ(void)

{

//volatile uint8\_t \*p=&OSC\_STATUS; //inner volatile saying pointer p could change.

//outer volative saying data in p could change

//reference to OSC\_STATUS

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void ACCEL\_WRITE(*uint8\_t* addr, *uint8\_t* data){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr= addr & 0b00111111 ; //RW is always 0 (write) and MS is always 0

SPI\_WRITE(addr);

SPI\_WRITE(data);

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

}

*uint8\_t* ACCEL\_READ(*uint8\_t* addr){

PORTF\_OUTCLR=0x08; //enable slave(accelerometer) device by setting it low. SPI have no automatic control of the SS line

PORTF\_OUTSET=0x04; //enable sensor\_sel, make it high. sensor\_sel = accelerometer

addr=addr | 0b10000000; //bitwise OR so RW (bit 7) is always 1 (Read). Gotta be careful of the MS signal

SPI\_WRITE(addr);

*uint8\_t* hi=SPI\_READ();

PORTF\_OUTSET=0x08; //disable slave(accelerometer) device by setting it high. SPI have no automatic control of the SS line

return hi; //data read from the ACCEL register

}

void ACCEL\_INIT(void){

PORTC\_INTCTRL=0x01; //enable low level interrupt

PORTC\_INT0MASK=0x80; //set pin 7 on C as source for interrupt

PORTC\_DIRCLR=0x80; //set pin 7 as input

PORTC\_PIN7CTRL=0x01; //rising edge trigger

PMIC\_CTRL=0x01;// enable low level interrupt in the PMIC

sei(); //enable global interrupt flag

ACCEL\_WRITE(CTRL\_REG4\_A, 0x01); //resetting the LSM system

ACCEL\_WRITE(CTRL\_REG4\_A,0b11001000); //data routed to to INT\_A, interrupt signal active high, edge triggered, INT1\_A signal enable 0b11101000

ACCEL\_WRITE(CTRL\_REG5\_A,0b10010111); //fastest output rate, BDU continous update, X Y Z enabled

} //also enabled PORT C pin 7 interrupt in the XMEGA

void USARTD0\_init(void)

{

PORTD\_DIRSET=0x08; //set transmitter as output

PORTD\_DIRCLR=0X04; //set receiver as input

USARTD0\_CTRLB=0x18; //enable receiver and transmitter

USARTD0\_CTRLC= 0x03; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

USARTD0\_BAUDCTRLA= (*uint8\_t*) BSEL; //load lowest 8 bits of BSEL

USARTD0\_BAUDCTRLB= (((*uint8\_t*) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits of BSEL. bitwise OR them

PORTD\_OUTSET= 0x08; //set transit pin idle

}

void OUT\_CHAR(*uint8\_t* data) { //changed it to 8 bit sign for accelerometer

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( ((USARTD0\_STATUS) & 0x20) != 0x20); //keep looping if DREIF flag is not set

USARTD0\_DATA= data;

}

void OUT\_STRING(*uint8\_t*\* data) { //pointing the pointer at that address

for (int i=0; data[i]!=0x00; i++) { //go through the whole string except the null terminator

OUT\_CHAR((*uint8\_t*) data[i]); //output the value

}

/\*

while(\*data != 0) //dereferencing

{

OUT\_CHAR((uint8\_t)\*data); //output the value

data++;

} \*/

}

*uint8\_t* IN\_CHAR(void) {

//volatile uint8\_t \*p=&USARTD0\_STATUS; //load the status flag data

while( (USARTD0\_STATUS & 0x80) != 0x80); //keep looping if DREIF flag is not set

return USARTD0\_DATA;

}

ISR(PORTC\_INT0\_vect) {

*uint8\_t* status=CPU\_SREG; //push status register

PORTC\_INTFLAGS=0x01 ; //clear the interrupt flag

intbit=1; //change intbit to 1 so we can read and transmit measured data from the accelerometer

CPU\_SREG= status; //pop the status register

}