Lab 5
Pengzhao Zhu
Section: 112D

B) Prelab Questions

1. What are some examples of useful macro functions, in the context of this lab?
   An example of a useful macro function is the use of #define (function-like macro) to take in arguments. This is then preprocessed so when I have to use that definition again later in the lab, I don't have to keep typing in. For example, I can do "#define hello(X) for(int i = 0; i < X; i++)". This code is good for when I have to use it for strings in Part A. Other similar macro functions can also be used to simplify coding process (while loop is another one).

2. Why do we use UART to communicate with the Atmel Studio Data Visualizer extension program?
   We have to get the data from the accelerometer to the Data Visualizer. We need to use the UART system (UART is actually physically connected to the computer through the USB) to get the information to Atmel Studio Data Visualizer.

3. What is the highest speed of communication that the IMU can handle?

   10 MHZ

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

   In general, a program should spend as little time as possible in the interrupt service routine as we want the program to be interrupted as little as possible. So it is better to just modify global flag variables inside of ISRs instead of doing everything inside of them.

5. Why is the "-D" tag used within the Data Stream Protocol defined in data_stream_protocol.txt?
   The "-D" determines the type of data used for that particular stream going towards the data stream visualizer. -D means signed short.

6. What is the most positive value that can be received from the accelerometer (in decimal)? What about the most negative?
   -D is a signed short. So -32,768 (negative) to 32,767 (positive).

C) Problems Encountered

The first problem I encountered in this lab was the use of C programming. It took me a while to get used to C syntax and understand how to use functions in C. I had to do a lot of research online to understand the different data types and pointers in C.

One other major problem I encountered in this lab was configuring the accelerometer. It was very difficult to find all the control signals between the port connections and the accelerometer. I had to go very deep in the schematics and the LSM specifications to complete this lab.

D) Future Application

By completing this lab, I gained more knowledge of the SPI and UART system. Using SPI and USART (asynchronous mode), I can communicate with other communication devices with microcontrollers. This skill is very important when it comes to future projects where I need to work with multiple devices at once. For example, I might have to use microcontrollers to control robots in senior design or I might have to use SPI at my future engineering job.

E) Schematics

N/A

F) Pseudocode/Flowcharts

Part A1 Pseudocode

Call function to initialize 32 Mhz clock
Call function to initialize USART
a=”U’

```
while(1) {
OUT_CHAR(a);
}
```

*USART Initialization code
*32 MHZ initialization code
*OUT_CHAR function to output data to putty
*OUT_STRING function to output strings to putty
* IN-CHAR function to take in information from the keypad

Part A2 Pseudocode

Call function to initialize 32 Mhz clock
Call function to initialize USART
Set red LED as output
Turn off red LED
CHECK:;

Read data from IN_CHAR
OUT_CHAR character

```
   if ((character != 'R') && (character !='r')) {
                              goto CHECK;
                    }
```

Read data from IN_CHAR
OUT_CHAR character

```
if ((character != 'E') && (character != 'e')){
                    goto CHECK;
}
```

Read data from IN_CHAR
OUT_CHAR character

```
if ((character != 'D') && (character != 'd')){
                        goto CHECK;
}

PORTD_OUTTGL=0x10;
goto CHECK;
```

*USART Initialization code
*32 MHZ initialization code
*OUT_CHAR function to output data to putty
*OUT_STRING function to output strings to putty
* IN-CHAR function to take in information from the keypad

Part B Pseudocode

```
int main(void){

            //nothing in the main. This program just contain functions that we need later

while(1) {

}
};
```

* Function to initialize SPI
            - Set input, output signals, SPIF_CTRL

*Function to write using SPI
*Function to read using SPI
*Function to initialize 32 MHZ clock

Part C Pseudocode

```
int main(void){
CLK_32MHZ();
SPI();   //call function to initialize SPI
while(1){
SPI_WRITE(0x53);
}
return 0;
```

*Function to initialize SPI
*Function to write using SPI

*Function to read using SPI
*Function to initialize 32MHZ clock

Part D Pseudocode

```
int main(void){
CLK_32MHZ();
SPI();   //call function to initialize SPI

uint8_t hello;
hello=ACCEL_READ(WHO_AM_I_A);

while(1);

return 0;
}
```

*Function to initialize SPI
*Function to write using SPI
*Function to read using SPI
*Function to initialize 32MHZ clock

*void ACCEL_WRITE(uint8_t addr, uint8_t data)-Function to write to the accelerometer. First byte will determine write/read and the address. Each byte will contain the data that we want to write to the accelerometer
*uint8_t ACCEL_READ(uint8_t addr)- Function to read to the accelerometer. Within the function, we will first write a garbage byte to accelerometer. Then we will read from it

Part E Pseudocode

```
int main(void){
CLK_32MHZ();
SPI();   //call function to initialize SPI

uint8_t hello;      //most are carry over code from Part D
hello=ACCEL_READ(WHO_AM_I_A);

while(1);

return 0;
}
```

*Function to initialize SPI
*Function to write using SPI
*Function to read using SPI
*Function to initialize 32MHZ clock

*void ACCEL_WRITE(uint8_t addr, uint8_t data)-Function to write to the accelerometer. First byte will determine write/read and the address. Each byte will contain the data that we want to write to the accelerometer
*uint8_t ACCEL_READ(uint8_t addr)- Function to read to the accelerometer. Within the function, we will first write a garbage byte to accelerometer. Then we will read from it

* void ACCEL_INIT(void){
enable low level interrupt
set pin 7 on C as source for interrupt
set pin 7 as input
rising edge trigger
enable low level interrupt in the PMIC
enable global interrupt flag
resetting the LSM system using ACCEL_WRITE
Write to the accelerometer to set up INT_A, interrupt signal, and trigger edge using ACCEL_WRITE
Write to the accelerometer to determine output rate, BDU, and enable XYZ plot data
}

Part F Pseudocode

*before MAIN. Set up a global volatile variable to determine when to write to data visualizer

```
int main(void){
CLK_32MHZ();
SPI();   //call function to initialize SPI
Call function to initialize USART

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'scomplement


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

return 0;
}
```

*Function to initialize SPI
*Function to write using SPI
*Function to read using SPI
*Function to initialize 32MHZ clock

*void ACCEL_WRITE(uint8_t addr, uint8_t data)-Function to write to the accelerometer. First byte will determine write/read and the address. Each byte will contain the data that we want to write to the accelerometer
*uint8_t ACCEL_READ(uint8_t addr)- Function to read to the accelerometer. Within the function, we will first write a garbage byte to accelerometer. Then we will read from it

* void ACCEL_INIT(void){
enable low level interrupt
set pin 7 on C as source for interrupt
set pin 7 as input
rising edge trigger
enable low level interrupt in the PMIC
enable global interrupt flag
resetting the LSM system using ACCEL_WRITE
Write to the accelerometer to set up INT_A, interrupt signal, and trigger edge using ACCEL_WRITE
Write to the accelerometer to determine output rate, BDU, and enable XYZ plot data
}
*Function to initialize USART
*ISR function to change volatile global variable “intbit” to 1

G) Program Code

Part A1

```c
/* Lab 5 Part A1
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program outputs the ASCII character "U" continously
*/


#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 ( ((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 OUT_CHAR(volatile uint8_t data) {

	//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= (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( (USARTD0_STATUS & 0x80) != 0x80);			//keep looping if DREIF flag is not set
	
	return USARTD0_DATA;
	
}
```

Part A2

```
/* Lab 5 Part A2
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program toggles on/off the red LED of the RBG package upon receiving a string of characters
*/



#include <avr/io.h>
void CLK_32MHZ(void);
void USARTD0_init(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();
	USARTD0_init();
	PORTD_DIRSET=0x10;   //set red led as output
	PORTD_OUTSET=0X10;  //turn off red led
```

```
	CHECK:;
	volatile uint8_t character=IN_CHAR();
	OUT_CHAR(character);


    if ((character != 'R') && (character !='r')) {
			goto CHECK;
		}

	character=IN_CHAR();
	OUT_CHAR(character);


	if ((character != 'E') && (character != 'e')){
		goto CHECK;
	}

	character=IN_CHAR();
	OUT_CHAR(character);

	if ((character != 'D') && (character != 'd')){
		goto CHECK;
	}

		PORTD_OUTTGL=0x10;
		goto CHECK;


	}




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 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 OUT_CHAR(volatile uint8_t data) {

	//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= (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( (USARTD0_STATUS & 0x80) != 0x80);		//keep looping if DREIF 
flag is not set
```

```
        return USARTD0_DATA;

}
```

Part B

```
/* Lab 5 Part B
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program contains functions to initialize the necessary SPI system,
transmit information from master device,
                            and read information from a slave device
*/



#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 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

```
/* Lab 5 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program continuously transmit 0x53. Will be used to verify the
functionality of functions written in Part B and
			the transibility of my SPI module.

*/
```

```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);
       }
       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=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

```
/* Lab 5 Part D
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program contains functions to write and read from the accelerometer. 
The program will read from a predefined register 
			    within the external LSM330 IMU

*/


#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);

#include<stdio.h>
#include<stdlib.h>
```

```c
int main(void){

	CLK_32MHZ();
	SPI();   //call function to initialize SPI

	uint8_t hello;
	hello=ACCEL_READ(WHO_AM_I_A);



	while(1);

	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=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 E

```
/* Lab 5 Part E
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program initialize and configure the necessary LSM330 accelerometer
registers needed for Part F (real-time plotting)
                            of the lab.

*/




#include <avr/io.h>
#include "LSM.h"
#include <avr/interrupt.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);
void ACCEL_INIT(void);





int main(void){

	CLK_32MHZ();
	SPI();	//call function to initialize SPI

	uint8_t hello;
	hello=ACCEL_READ(WHO_AM_I_A);



	while(1);

	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=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
}

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
}
```

Part F

```
/* Lab 5 Part F
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program will ultilize all functions written in this lab to plot real time data from the accelerometer

*/



#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,0b11101000); //data routed to to INT_A, interrupt signal active high, pulsed, 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

}
```

H) Appendix

Figure 1: USART Single Bit Transmission.

***17.337 us= .000017337 s. Hz= bits per seconds. Baud rate=bits per seconds**

***1/(.000017227)=57680.1 =57600 Hz required in the doc**

Figure 2: USART Single Frame Transmission

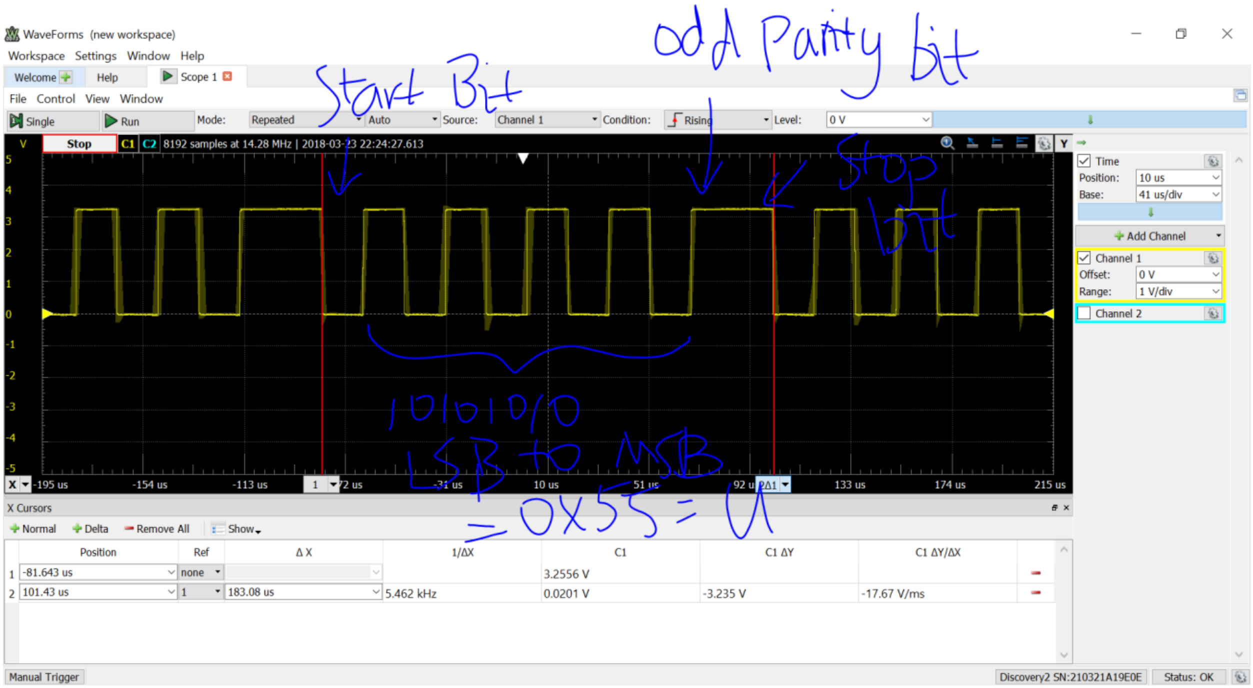

Figure 3: USART Single Frame Transmission Annotated

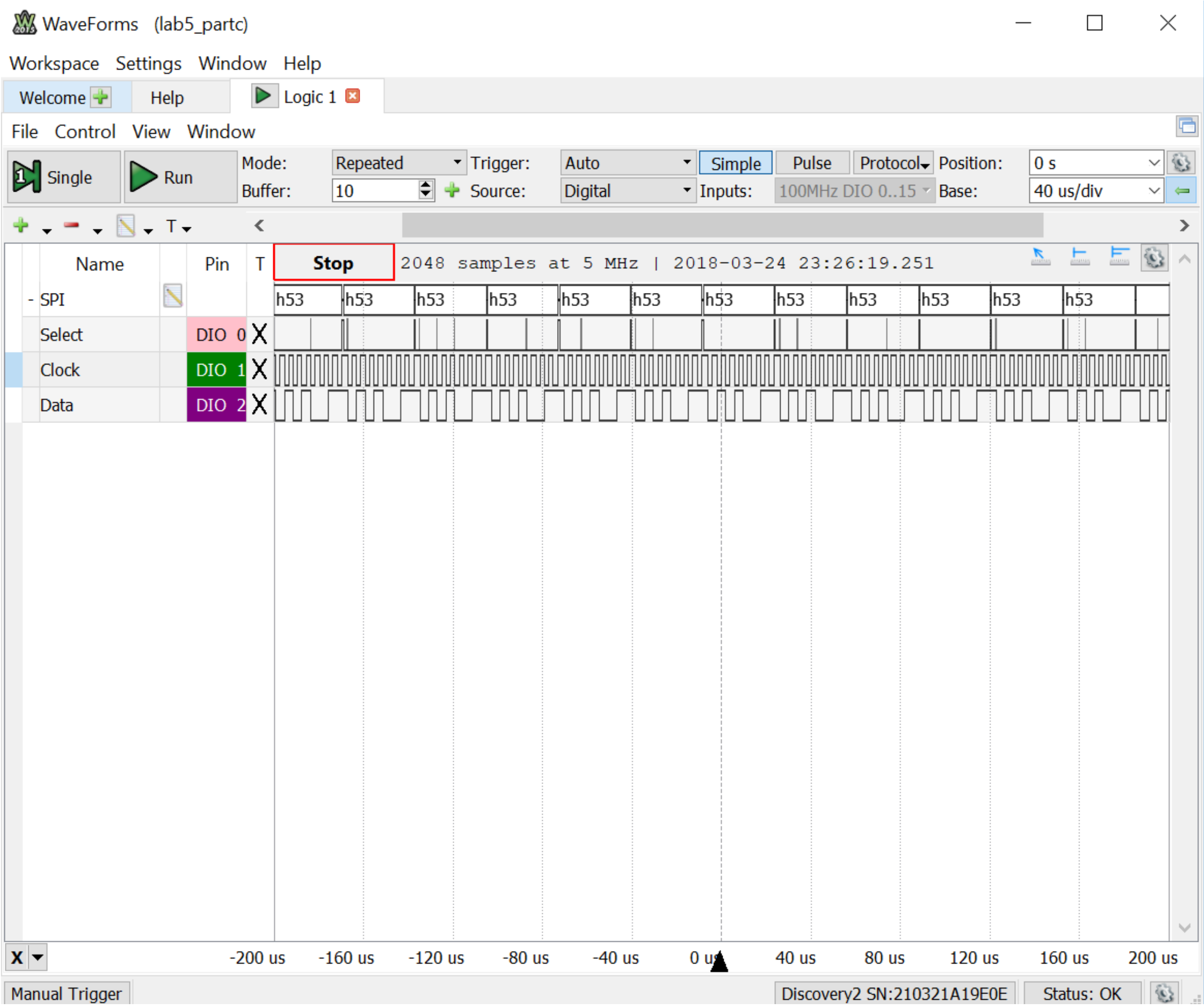

Figure 4: SPI Transmission (Part C)

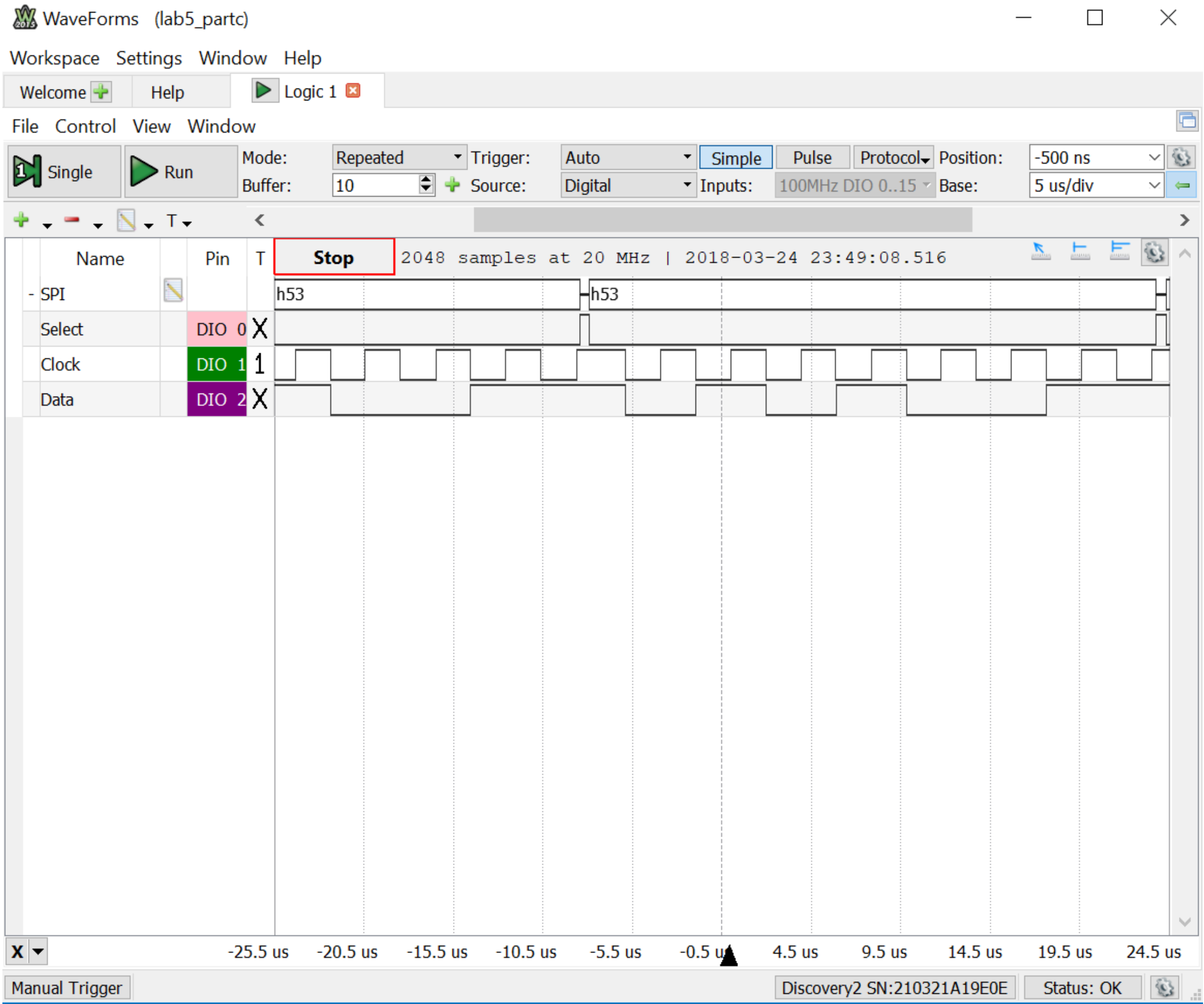

Figure 5: SPI Transmission (Part C)- Close View

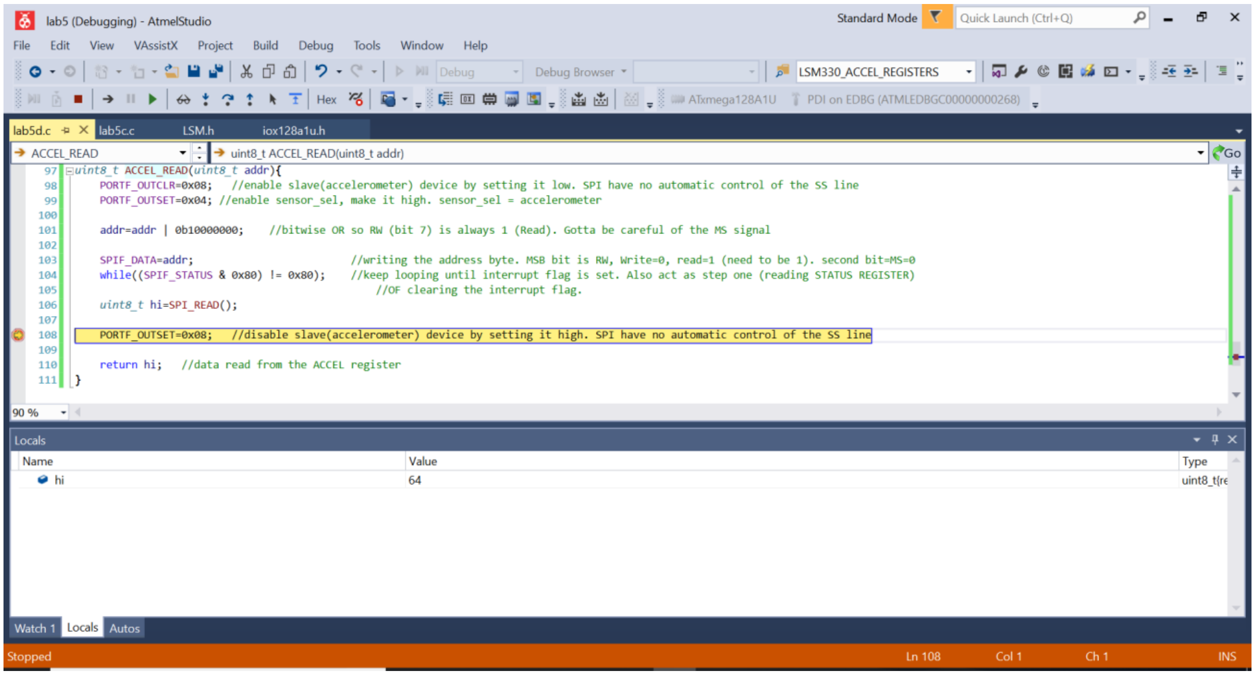

Figure 6: WHO_AM_I_A register data confirmation. 64=0x40. (Part D)

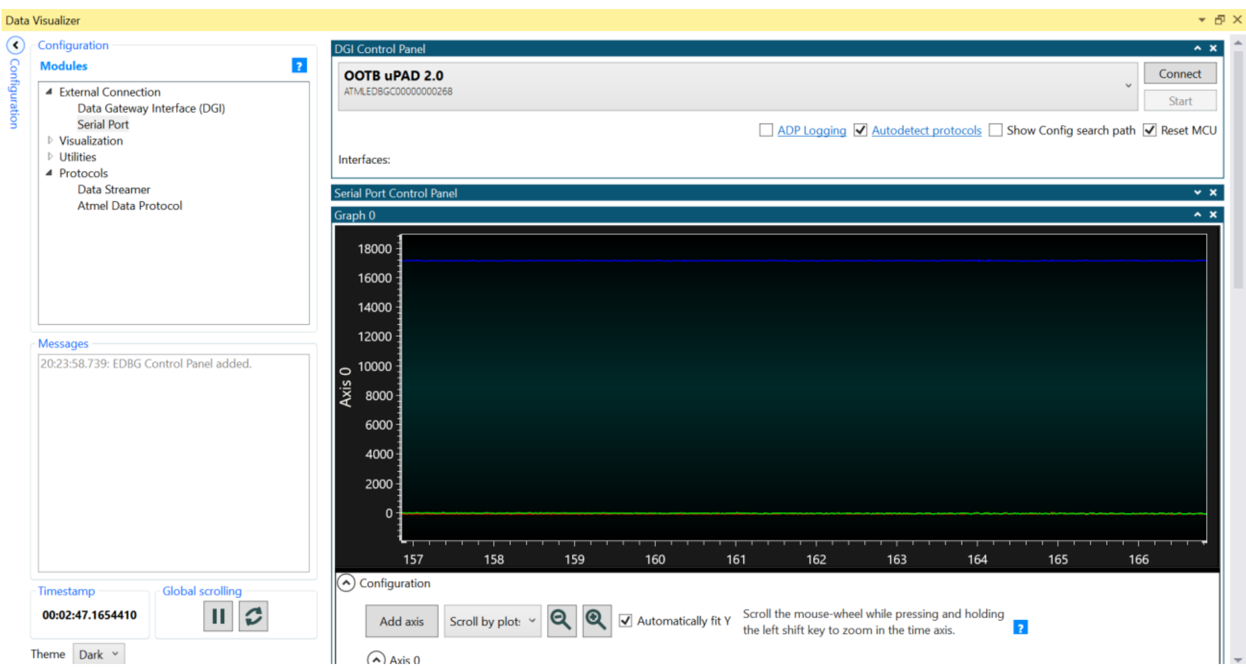

Figure 7: Part F Real Time Plot Static View

Figure 7: Part F Real-Time Plot Dynamic View