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).**

1. 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.**

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

**10 MHZ**

1. 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.**

1. 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.**

1. 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**

/\* 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.

\*/

#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>

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



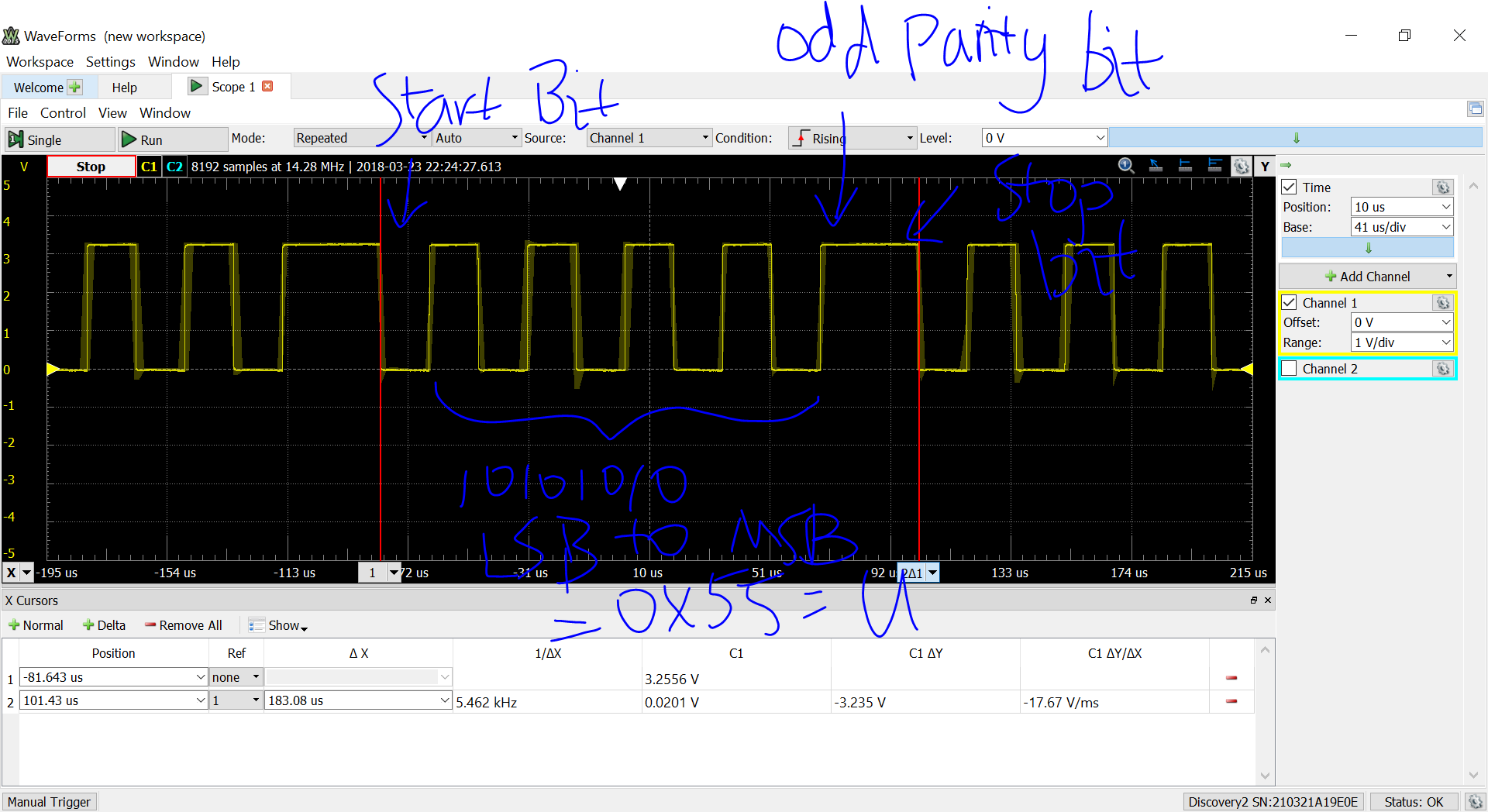
**Figure 1: USART Single Bit Transmission.**

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

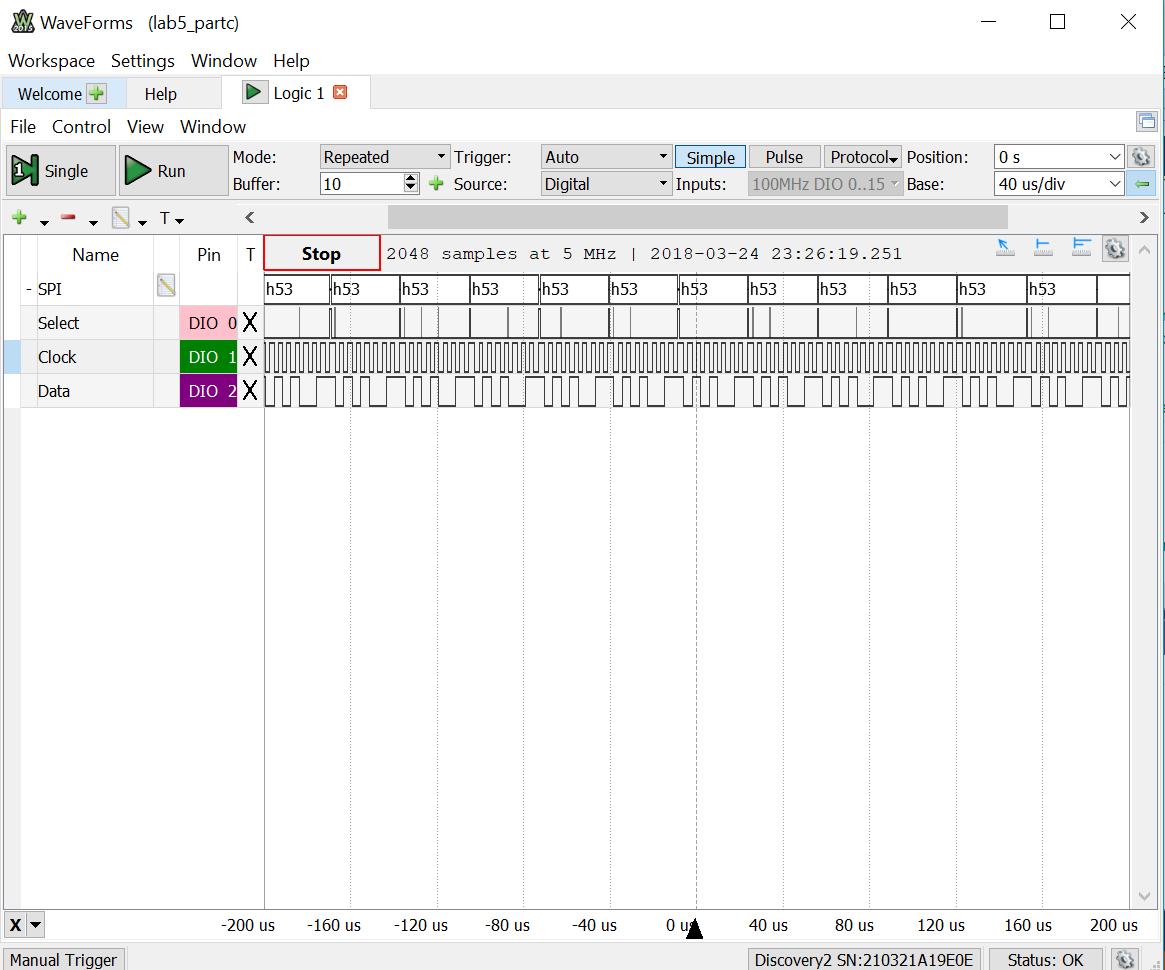
**\*1/(.000017227)=57680.1 =57600 Hz required in the doc**



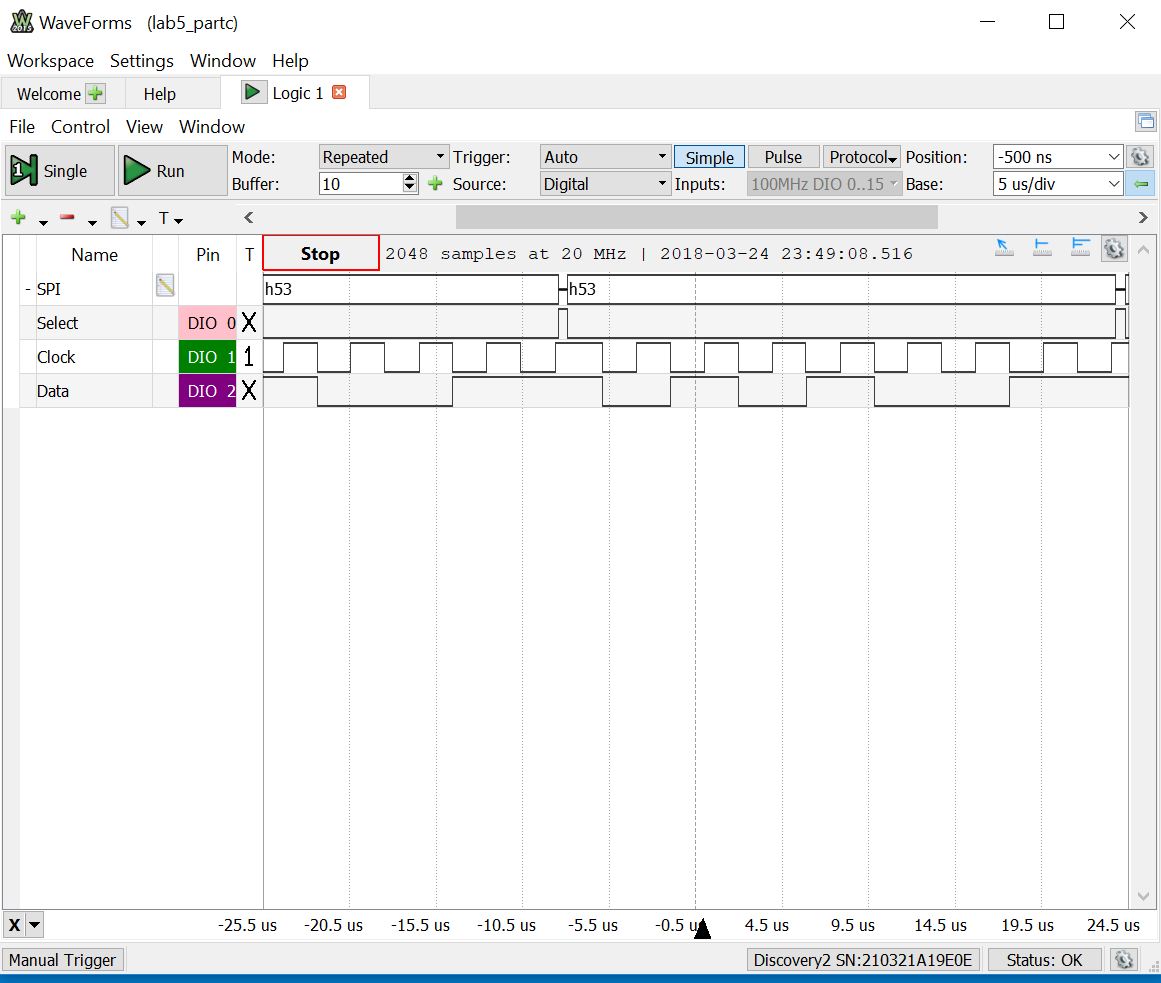
**Figure 2: USART Single Frame Transmission**



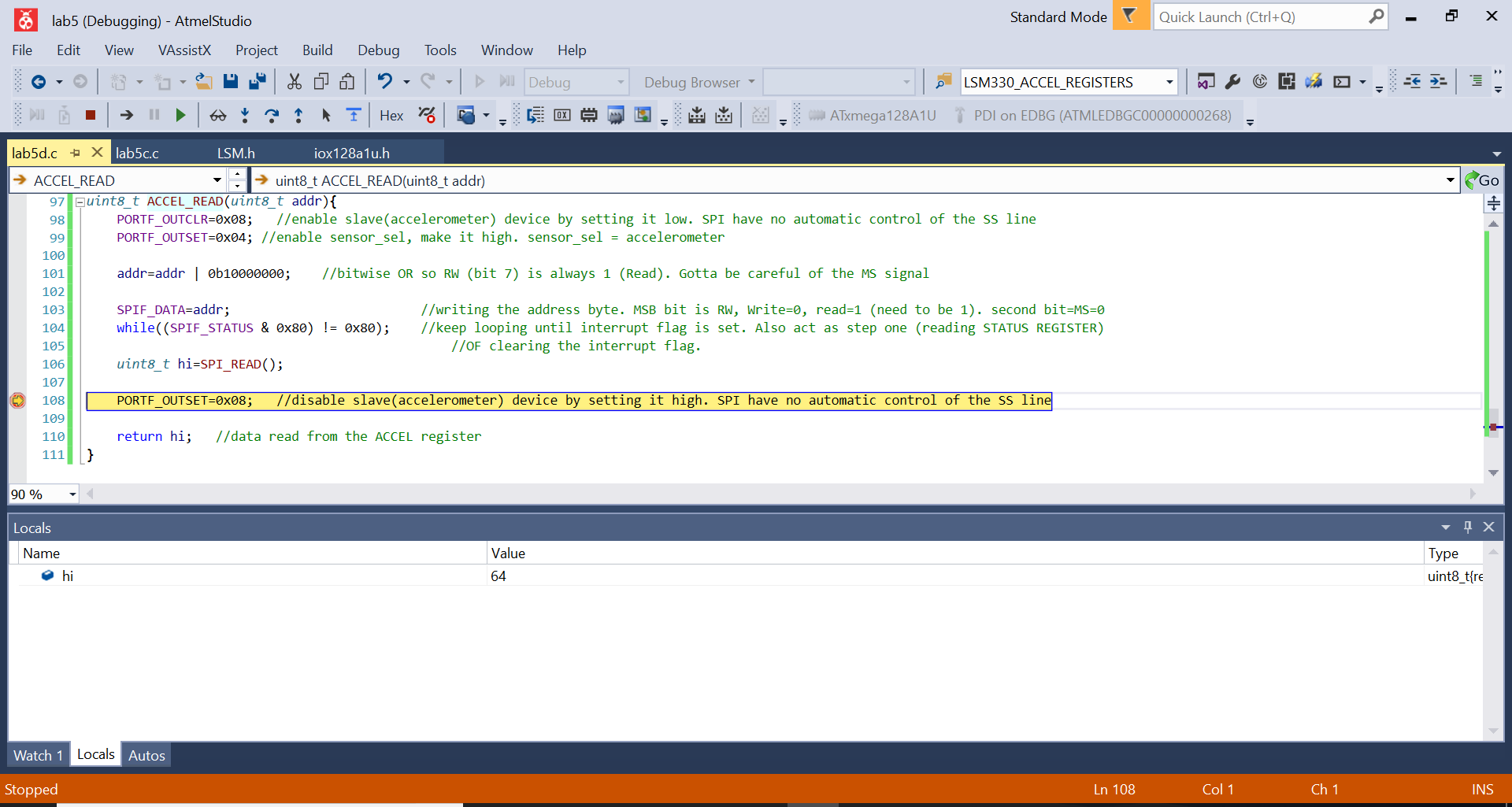
**Figure 3: USART Single Frame Transmission Annotated**



**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**