**REQUIREMENTS NOT MET**

N/A

**PROBLEMS ENCOUNTERED**

N/A

**FUTURE WORK/APPLICATIONS**

Applications of this includes development of communications for basically all microcontrollers and peripheral devices, as SPI is faster than UART.

**PRE-LAB EXERCISES**

**i. In regard to SPI communication that is to exist between the relevant ATxmega128A1U and IMU chips, answer each of the questions within the previously given bulleted list**

**Which device(s) should be given the role of master and which device(s) should be given the role of student?**

**The IMU should be the slave, and the ATX should be the master**

**How will the student device(s) be enabled? If a student select is utilized, rather than just have the device(s) be permanently enabled, which pin(s) will be used?**

**The slave will be enabled using its chip select.  
 The chip select of the slave (pin 12) will be connected to the slave select of the ATX(port F pin 5).**

**What is the order of data transmission? Is the MSb or LSb transmitted first?**

**The data should be transmitted MSB first**

**In regard to the relevant clock signal, should data be latched on a rising edge or on a falling edge?**

**The IMU transmits and receives data on a rising clock edge. So the data should be latched on a falling edge.**

**What is the maximum serial clock frequency that can be utilized by the relevant devices?**

**The ATX can transmit/receive data at a max rate of 1MHZ.  
 However, the IMU can transmit at a max rate of 10MHZ**

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

**The ISR subroutines are meant to be short as possible. Since were doing live data logging in this lab, outputting data in an ISR may slow down the speed of the logging software.**

**iii. To output two unsigned 32-bit values 0x30680905 [CH1] and 0x02225196 [CH2] to SerialPlot, list all the bytes in the order you would send them via UART.**

**0x05**

**0x09**

**0x68**

**0x30**

**0x96**

**0x51**

**0x22**

**0x02**

**iv. What is the most positive value that can be received from the accelerometer (in decimal)? What about the most negative?**

**Since the registers are based on 16-bit signed numbers, 15 bits of data are available to represent the the actual number.  
 That means that, theoretically, the highest number that can be represented is 32767.  
 The lowest number that can be represented is, theoretically, -32767.**

**PSEUDOCODE/FLOWCHARTS**

**Section 2**

**A diagram of a program

Description automatically generated  
Figure 1: Flowchart for “lab6\_2.C”**

**Section 3**

**A diagram of a program

Description automatically generated  
Figure 2: Flowchart for “lab6\_3.C”**

**Section 5**

**A diagram of a software company

Description automatically generated  
Figure 3: Flowcharts for “la6\_5.C”**

**PROGRAM CODE**

**SECTION 2**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Lab 6, Section 2

//Name: Steven Miller

//Class #: 11318

//PI Name: Anthony Stross

//Description: transmits 0x2a over mosi

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include "spi.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN PROGRAM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

//init spi

spi\_init();

//transmit 0x2a

while(1)

{

//turn on chip select

PORTF.OUTCLR = SS\_bm;

//write out to mosi

spi\_write(0x2a);

//turn off chip select

PORTF.OUTSET = SS\_bm;

}

return 0;

}

**SECTION 3**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Lab 6, Section 3

//Name: Steven Miller

//Class #: 11318

//PI Name: Anthony Stross

//Description: gets imus id number

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "lsm6dsl.h"

#include "lsm6dsl\_registers.h"

#include <avr/io.h>

#include "spi.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

spi\_init();

//read "who am i?" register

while(1)

{

*uint8\_t* identity = lsm\_read(WHO\_AM\_I);

}

//uint8\_t identity = lsm\_read(WHO\_AM\_I);

return 0;

}

**SECTION 5**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Lab 6, Section 5

//Name: Steven Miller

//Class #: 11318

//PI Name: Anthony Stross

//Description: gets imus acceleration and gyroscopic data and outputs it to the computer

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "lsm6dsl.h"

#include "lsm6dsl\_registers.h"

#include <avr/io.h>

#include "spi.h"

#include "usart.h"

#include <avr/interrupt.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//flags

volatile *uint8\_t* accel\_flag;

void intr\_init(void);

int main(void)

{

accel\_flag = 0;

spi\_init();

usartd0\_init();

LSM\_init();

intr\_init();

wakeup\_accel();

//wakeup gyro

read\_gyro();

while(1)

{

if(accel\_flag == 1)

{

/\*read accelerometer data\*/

*uint8\_t* data = 0;

data = LSM\_read((OUTX\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTX\_H\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_H\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_H\_XL));

usartd0\_out\_char(data);

/\*read gyroscope data\*/

read\_gyro();

PORTC.INTCTRL = (PORT\_INT0LVL\_MED\_gc);

accel\_flag = 0;

}

}

return 0;

}

ISR(PORTC\_INT0\_vect)

{

//disable interrupt

PORTC.INTCTRL = (0);

accel\_flag = 1;

}

void intr\_init(void)

{

//enable medium level interrupts

PMIC.CTRL = (PMIC\_MEDLVLEN\_bm);

sei();

}

void wakeup\_accel(void)

{

*uint8\_t* data = 0;

data = LSM\_read((OUTX\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTX\_H\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_H\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_L\_XL));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_H\_XL));

}

//put gyroscope data here

void read\_gyro()

{

/\*read gyroscope data\*/

*uint8\_t* data = 0;

data = LSM\_read((OUTX\_L\_G));

usartd0\_out\_char(data);

data = LSM\_read((OUTX\_H\_G));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_L\_G));

usartd0\_out\_char(data);

data = LSM\_read((OUTY\_H\_G));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_L\_G));

usartd0\_out\_char(data);

data = LSM\_read((OUTZ\_H\_G));

usartd0\_out\_char(data);

}

**APPENDIX**

**A screenshot of a computer

Description automatically generated  
Figure 4: Measurement of “lab6\_2.C”**

**A screenshot of a graph

Description automatically generated  
Figure 5: Screenshot of accelerometer plots**

**A screenshot of a graph

Description automatically generated  
Figure 6: Screenshot of gyroscope plot**

**SPI.C**

/\*------------------------------------------------------------------------------

spi.c --

Description:

Provides useful definitions for manipulating the relevant SPI

module of the ATxmega128A1U.

Author(s): Dr. Eric M. Schwartz, Christopher Crary, Wesley Piard

Last modified by: Dr. Eric M. Schwartz

Last modified on: 8 Mar 2023

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include "spi.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void spi\_init(void)

{

/\* Initialize the relevant SPI output signals to be in an "idle" state.

\* Refer to the relevant timing diagram within the LSM6DSL datasheet.

\* (You may wish to utilize the macros defined in `spi.h`.) \*/

PORTF.OUTSET = (SS\_bm|SCK\_bm);

/\* Configure the pin direction of relevant SPI signals. \*/

PORTF.DIRSET = (SS\_bm|MOSI\_bm|SCK\_bm);

PORTF.DIRCLR = (MISO\_bm);

/\* Set the other relevant SPI configurations. \*/

SPIF.CTRL =(SPI\_PRESCALER\_DIV4\_gc|SPI\_MASTER\_bm|SPI\_MODE\_3\_gc|SPI\_ENABLE\_bm);

}

void spi\_write(*uint8\_t* data)

{

/\* Write to the relevant DATA register. \*/

SPIF.DATA = data;

/\* Wait for relevant transfer to complete. \*/

while(!(SPIF.STATUS & SPI\_IF\_bm))

{

//do nothing

}

/\* In general, it is probably wise to ensure that the relevant flag is

\* cleared at this point, but, for our contexts, this will occur the

\* next time we call the `spi\_write` (or `spi\_read`) routine.

\* Really, because of how the flag must be cleared within

\* ATxmega128A1U, it would probably make more sense to have some single

\* function, say `spi\_transceive`, that both writes and reads

\* data, rather than have two functions `spi\_write` and `spi\_read`,

\* but we will not concern ourselves with this possibility

\* during this semester of the course. \*/

}

*uint8\_t* spi\_read(void)

{

/\* Write some arbitrary data to initiate a transfer. \*/

SPIF.DATA = 0x37;

/\* Wait for relevant transfer to be complete. \*/

while(!(SPIF.STATUS & SPI\_IF\_bm))

{

//do nothing

}

/\* After the transmission, return the data that was received. \*/

return SPIF.DATA;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**SPI.H**

#ifndef SPI\_H\_ // Header guard.

#define SPI\_H\_

/\*------------------------------------------------------------------------------

spi.h --

Description:

Provides function prototypes and macro definitions for utilizing the SPI

system of the ATxmega128A1U.

Author(s): Dr. Eric M. Schwartz, Christopher Crary, Wesley Piard

Last modified by: Dr. Eric M. Schwartz

Last modified on: 8 Mar 2023

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SS\_bm (1<<4)

#define MOSI\_bm (1<<5)

#define MISO\_bm (1<<6)

#define SCK\_bm (1<<7)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------

spi\_init --

Description:

Initializes the relevant SPI module to communicate with the LSM6DSL.

Input(s): N/A

Output(s): N/A

------------------------------------------------------------------------------\*/

void spi\_init(void);

/\*------------------------------------------------------------------------------

spi\_write --

Description:

Transmits a single byte of data via the relevant SPI module.

Input(s): `data` - 8-bit value to be written via the relevant SPI module.

Output(s): N/A

------------------------------------------------------------------------------\*/

void spi\_write(*uint8\_t* data);

/\*------------------------------------------------------------------------------

spi\_read --

Description:

Reads a byte of data via the relevant SPI module.

Input(s): N/A

Output(s): 8-bit value read from the relevant SPI module.

------------------------------------------------------------------------------\*/

*uint8\_t* spi\_read(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif // End of header guard.

**LSM6DSL.C**

/\*------------------------------------------------------------------------------

lsm6dsl.c --

Description:

Brief description of file.

Extended description, if appropriate.

Author(s):

Last modified by: Dr. Eric M. Schwartz

Last modified on: 8 Mar 2023

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include "lsm6dsl.h"

#include "lsm6dsl\_registers.h"

#include "spi.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void lsm\_write(*uint8\_t* reg\_addr, *uint8\_t* data)

{

//enable imu by enabling chip select

PORTF.OUTCLR = SS\_bm;

//send over the address bits

//keep in mind that writing to an address in the imu takes 16 cycles(16 bits of data need to be shifted)

//the first bit of the 8 bit address is the strobe bit. Which tells the imu whether we wanna read or write.

//1= read, 0 = write

spi\_write(reg\_addr|LSM6DSL\_SPI\_WRITE\_STROBE\_bm);

//our spi master data register is now filled with junk data,

//we now send out the data we wanna store in the imu.

spi\_write(data);

//disable imu by disabling chip select

PORTF.OUTSET = SS\_bm;

}

*uint8\_t* LSM\_read(*uint8\_t* reg\_addr)

{

//enable imu by enabling chip select

PORTF.OUTCLR = SS\_bm;

//send over the address bits

//keep in mind that reading from an address in the imu takes 16 cycles(16 bits of data need to be shifted)

//the first bit of the 8 bit address is the strobe bit. Which tells the imu whether we wanna read or write.

//1= read, 0 = write

spi\_write(reg\_addr|LSM6DSL\_SPI\_READ\_STROBE\_bm);

//our spi master data register is now filled with junk data,

//we need to perform another read so we can activate the clock and recieve our desired data

spi\_read();

//disable imu by disabling chip select

PORTF.OUTSET = SS\_bm;

return SPIF.DATA;

}

void LSM\_init(void)

{

/\*enable interrupt detection on port c PIN 6 of atx\*/

//set pin 6 as input

PORTC.DIRCLR = (PORTC.DIRCLR|PIN6\_bm);

//enable interrupts on pin 6

PORTC.INT0MASK = (PORTC.INT0MASK|PIN6\_bm);

//make it sense low level

PORTC.PIN6CTRL= (PORTC.PIN6CTRL|PORT\_ISC\_LEVEL\_gc);

//make it medium priority

PORTC.INTCTRL = (PORTC.INTCTRL|PORT\_INT0LVL\_MED\_gc);

//restart device

lsm\_write(CTRL3\_C,LSM6DSL\_RESET\_DEVICE\_BM);

//make interrupts active low

lsm\_write(CTRL3\_C,LSM6DSL\_INT1\_MAKE\_ACTIVE\_LOW);

//enable all axes

lsm\_write(CTRL9\_XL,LSM6DSL\_ENABLE\_ALLAXIS);

//output data rate and scale setting

lsm\_write((CTRL1\_XL),(LSM6DSL\_208HZ|LSM6DSL\_SCALE\_2));

//enable interrupt 1 for accel

lsm\_write(INT1\_CTRL,LSM6DSL\_DRDY\_XL\_EN\_BM);

//output data rate and scale setting

lsm\_write((CTRL2\_G),(LSM6DSL\_208HZ|LSM6DSL\_SCALE\_2));

//enable interrupt 2 for gyro

lsm\_write(INT2\_CTRL,LSM6DSL\_DRDY\_GY\_EN\_BM);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**LSM6DSL.H**

#ifndef LSM6DSL\_H\_ // Header guard.

#define LSM6DSL\_H\_

/\*------------------------------------------------------------------------------

lsm6dsl.h --

Description:

Provides custom data types to make it easier to handle any data

read from the LSM6DSL IMU.

The LSM6DSL can output accelerometer and gyroscope data. Data from both

of these sensors is represented in a 16-bit signed format.

Author(s): Wesley Piard & Leslye Castillo & Dr. Eric M. Schwartz

Last modified by: Dr. Eric M. Schwartz

Last modified on: 29 June 2022

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define LSM6DSL\_SPI\_READ\_STROBE\_bm 0x80

#define LSM6DSL\_SPI\_WRITE\_STROBE\_bm 0x00

//Miller, july 12 2023: added macros here:

//UTILITIES

#define LSM6DSL\_RESET\_DEVICE\_BM (0x01<<0)

#define LSM6DSL\_INT1\_MAKE\_ACTIVE\_LOW (0X01<<5)

#define LSM6DSL\_SCALE\_2 ((0X00 <<3) |(0X00 << 2))

#define LSM6DSL\_208HZ (5<<4)

//ACCELEROMETER

#define LSM6DSL\_ENABLE\_XAXIS (0X01 << 7)

#define LSM6DSL\_ENABLE\_YAXIS (0X01 << 6)

#define LSM6DSL\_ENABLE\_ZAXIS (0X01 << 5)

#define LSM6DSL\_ENABLE\_ALLAXIS (LSM6DSL\_ENABLE\_XAXIS|LSM6DSL\_ENABLE\_YAXIS|LSM6DSL\_ENABLE\_ZAXIS)

#define LSM6DSL\_DRDY\_XL\_EN\_BM (0X01<<0)

//GYROSCOPE

#define LSM6DSL\_DRDY\_GY\_EN\_BM (0x01<<1)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Used to differentiate the accelerometer and gyroscope within the LSM6DSL. \*/

typedef enum {LSM6DSL\_ACCEL, LSM6DSL\_GYRO} lsm6dsl\_module\_t;

/\* Can be used to contain the separated bytes of data as they are read from

\* the LSM6DSL. \*/

typedef struct lsm6dsl\_data\_raw

{

*uint8\_t* accel\_x\_low, accel\_x\_high;

*uint8\_t* accel\_y\_low, accel\_y\_high;

*uint8\_t* accel\_z\_low, accel\_z\_high;

*uint8\_t* gyro\_x\_low, gyro\_x\_high;

*uint8\_t* gyro\_y\_low, gyro\_y\_high;

*uint8\_t* gyro\_z\_low, gyro\_z\_high;

}lsm6dsl\_data\_raw\_t;

/\* Contains the full concatenated signed 16-bit words of data. \*/

typedef struct lsm6dsl\_data\_full

{

*int16\_t* accel\_x, accel\_y, accel\_z;

*int16\_t* gyro\_x, gyro\_y, gyro\_z;

}lsm6dsl\_data\_full\_t;

/\* Provides the ability to choose how to access the LSM6DSL data. \*/

typedef union lsm6dsl\_data

{

lsm6dsl\_data\_full\_t word;

lsm6dsl\_data\_raw\_t byte;

}lsm6dsl\_data\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void LSM\_write(*uint8\_t* reg\_addr, *uint8\_t* data);

*uint8\_t* LSM\_read(*uint8\_t* reg\_addr);

void LSM\_init(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif // End of header guard.

**LSM6DSL\_REGISTERS.H**

#ifndef LSM6DSL\_REGISTERS\_H\_ // Header guard.

#define LSM6DSL\_REGISTERS\_H\_

/\*------------------------------------------------------------------------------

lsm6dsl\_registers.h --

Description:

Provides useful macro definitions and symbols that can be used

when accesing registers of the LSM6DSL IMU.

Created by: OOTB-LT

Created on: 14 September 2019

Author(s): Leslye Castillo & Dr. Eric M. Schwartz

Last modified by: Dr. Eric M. Schwartz

Last modified on: 29 June 2022

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef enum LSM6DSL\_ACCEL\_REGISTERS

{

FUNC\_CFG\_ACCESS = 0x01,

SENSOR\_SYNC\_TIME\_FRAME = 0x04,

SENSOR\_SYNC\_RES\_RATIO = 0x05,

FIFO\_CTRL1 = 0x06,

FIFO\_CTRL2 = 0x07,

FIFO\_CTRL3 = 0x08,

FIFO\_CTRL4 = 0x09,

FIFO\_CTRL5 = 0x0A,

DRDY\_PULSE\_CFG\_G = 0x0B,

INT1\_CTRL = 0x0D,

INT2\_CTRL = 0x0E,

WHO\_AM\_I = 0x0F,

CTRL1\_XL = 0x10,

CTRL2\_G = 0x11,

CTRL3\_C = 0x12,

CTRL4\_C = 0x13,

CTRL5\_C = 0x14,

CTRL6\_C = 0x15,

CTRL7\_G = 0x16,

CTRL8\_XL = 0x17,

CTRL9\_XL = 0x18,

CTRL10\_C = 0x19,

MASTER\_CONFIG = 0x1A,

WAKE\_UP\_SRC = 0x1B,

TAP\_SRC = 0x1C,

D6D\_SRC = 0x1D,

STATUS\_REG = 0x1E,

OUT\_TEMP\_L = 0x20,

OUT\_TEMP\_H = 0x21,

OUTX\_L\_G = 0x22,

OUTX\_H\_G = 0x23,

OUTY\_L\_G = 0x24,

OUTY\_H\_G = 0x25,

OUTZ\_L\_G = 0x26,

OUTZ\_H\_G = 0x27,

OUTX\_L\_XL = 0x28,

OUTX\_H\_XL = 0x29,

OUTY\_L\_XL = 0x2A,

OUTY\_H\_XL = 0x2B,

OUTZ\_L\_XL = 0x2C,

OUTZ\_H\_XL = 0x2D,

SENSORHUB1\_REG = 0x2E,

SENSORHUB2\_REG = 0x2F,

SENSORHUB3\_REG = 0x30,

SENSORHUB4\_REG = 0x31,

SENSORHUB5\_REG = 0x32,

SENSORHUB6\_REG = 0x33,

SENSORHUB7\_REG = 0x34,

SENSORHUB8\_REG = 0x35,

SENSORHUB9\_REG = 0x36,

SENSORHUB10\_REG = 0x37,

SENSORHUB11\_REG = 0x38,

SENSORHUB12\_REG = 0x39,

FIFO\_STATUS1 = 0x3A,

FIFO\_STATUS2 = 0x3B,

FIFO\_STATUS3 = 0x3C,

FIFO\_STATUS4 = 0x3D,

FIFO\_DATA\_OUT\_L = 0x3E,

FIFO\_DATA\_OUT\_H = 0x3F,

TIMESTAMP0\_REG = 0x40,

TIMESTAMP1\_REG = 0x41,

TIMESTAMP2\_REG = 0x42,

STEP\_TIMESTAMP\_L = 0x49,

STEP\_TIMESTAMP\_H = 0x4A,

STEP\_COUNTER\_L = 0x4B,

STEP\_COUNTER\_H = 0x4C,

SENSORHUB13\_REG = 0x4D,

SENSORHUB14\_REG = 0x4E,

SENSORHUB15\_REG = 0x4F,

SENSORHUB16\_REG = 0x50,

SENSORHUB17\_REG = 0x51,

SENSORHUB18\_REG = 0x52,

FUNC\_SRC1 = 0x53,

FUNC\_SRC2 = 0x54,

WRIST\_TILT\_IA = 0x55,

TAP\_CFG = 0x58,

TAP\_THS\_6D = 0x59,

INT\_DUR2 = 0x5A,

WAKE\_UP\_THS = 0x5B,

WAKE\_UP\_DUR = 0x5C,

FREE\_FALL = 0x5D,

MD1\_CFG = 0x5E,

MD2\_CFG = 0x5F,

MASTER\_CMD\_CODE = 0x60,

SENS\_SYNC\_SPI\_ERROR\_CODE = 0x61,

OUT\_MAG\_RAW\_X\_L = 0x66,

OUT\_MAG\_RAW\_X\_H = 0x67,

OUT\_MAG\_RAW\_Y\_L = 0x68,

OUT\_MAG\_RAW\_Y\_H = 0x69,

OUT\_MAG\_RAW\_Z\_L = 0x6A,

OUT\_MAG\_RAW\_Z\_H = 0x6B,

X\_OFS\_USR = 0x73,

Y\_OFS\_USR = 0x74,

Z\_OFS\_USR = 0x75

}LSM6DSL\_REGA\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif // End of header guard.

**USART.C**

/\*------------------------------------------------------------------------------

usart.c --

Description:

Provides some useful definitions regarding the USART system of the

ATxmega128A1U.

Author(s): Dr. Eric Schwartz, Christopher Crary, Wesley Piard

Last modified by: Dr. Eric M. Schwartz

Last modified on: 8 Mar 2023

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include "usart.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* At 2 MHz SYSclk, 5 BSEL, -6 BSCALE corresponds to 115200 bps \*/

#define BSEL (5)

#define BSCALE (-6)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char usartd0\_in\_char(void)

{

/\* intentionally left blank \*/

}

void usartd0\_in\_string(char \* buf)

{

/\* intentionally left blank \*/

}

void usartd0\_init(void)

{

/\* Configure relevant TxD and RxD pins. \*/

PORTD.OUTSET = PIN3\_bm;

PORTD.DIRSET = PIN3\_bm;

PORTD.DIRCLR = PIN2\_bm;

/\* Configure baud rate. \*/

USARTD0.BAUDCTRLA = (*uint8\_t*)BSEL;

USARTD0.BAUDCTRLB = (*uint8\_t*)((BSCALE << 4)|(BSEL >> 8));

/\* Configure remainder of serial protocol. \*/

/\* (In this example, a protocol with 8 data bits, no parity, and

\* one stop bit is chosen.) \*/

USARTD0.CTRLC = (USART\_CMODE\_ASYNCHRONOUS\_gc |USART\_PMODE\_DISABLED\_gc| USART\_CHSIZE\_8BIT\_gc)&(~USART\_SBMODE\_bm);

/\* Enable receiver and/or transmitter systems. \*/

USARTD0.CTRLB = USART\_RXEN\_bm | USART\_TXEN\_bm;

/\* Enable interrupt (optional). \*/

/\* USARTD0.CTRLA = USART\_RXCINTLVL\_MED\_gc; \*/

}

void usartd0\_out\_char(char c)

{

while(!(USARTD0.STATUS & USART\_DREIF\_bm));

USARTD0.DATA = c;

}

void usartd0\_out\_string(const char \* str)

{

while(\*str) usartd0\_out\_char(\*(str++));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION DEFINITIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**USART.H**

#ifndef USART\_H // Header guard.

#define USART\_H

/\*------------------------------------------------------------------------------

usart.h --

Description:

Provides some useful declarations regarding the USART system of the

ATxmega128A1U.

Author(s): Dr. Eric Schwartz, Christopher Crary, Wesley Piard

Last modified by: Dr. Eric M. Schwartz

Last modified on: 8 Mar 2023

------------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF DEPENDENCIES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF MACROS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF CUSTOM DATA TYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------------------------------------------------------

usartd0\_in\_char --

Description:

Returns a single character via the receiver of the USARTD0 module.

Input(s): N/A

Output(s): Character received from USARTD0 module.

------------------------------------------------------------------------------\*/

char usartd0\_in\_char(void);

/\*------------------------------------------------------------------------------

usartd0\_in\_string --

Description:

Reads in a string with the receiever of the USARTD0 module.

The string is to be stored within a pre-allocated buffer, accessible

via the character pointer `buf`.

Input(s): `buf` - Pointer to character buffer.

Output(s): N/A

------------------------------------------------------------------------------\*/

void usartd0\_in\_string(char \* buf);

/\*------------------------------------------------------------------------------

usartd0\_init --

Description:

Configures the USARTD0 module for a specific asynchronous serial protocol.

Input(s): N/A

Output(s): N/A

------------------------------------------------------------------------------\*/

void usartd0\_init(void);

/\*------------------------------------------------------------------------------

usartd0\_out\_char --

Description:

Outputs a character via the transmitter of the USARTD0 module.

Input(s): `c` - Read-only character.

Output(s): N/A

------------------------------------------------------------------------------\*/

void usartd0\_out\_char(char c);

/\*------------------------------------------------------------------------------

usartd0\_out\_string --

Description:

Outputs a string via the transmitter of the USARTD0 module.

The string is to be stored within a pre-allocated buffer, accessible

via the character pointer `str`.

Input(s): `str` - Pointer to read-only character string.

Output(s): N/A

------------------------------------------------------------------------------\*/

void usartd0\_out\_string(const char \* str);

void usartd0\_out\_string\_no\_null(const char\* str);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF FUNCTION PROTOTYPES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif // End of header guard.