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

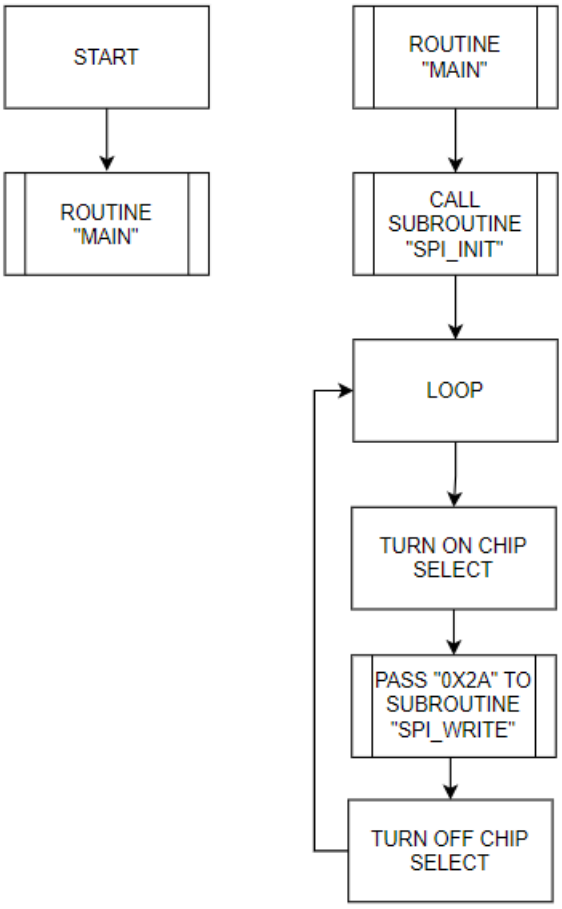

Figure 1: Flowchart for “lab6_2.C”

Section 3

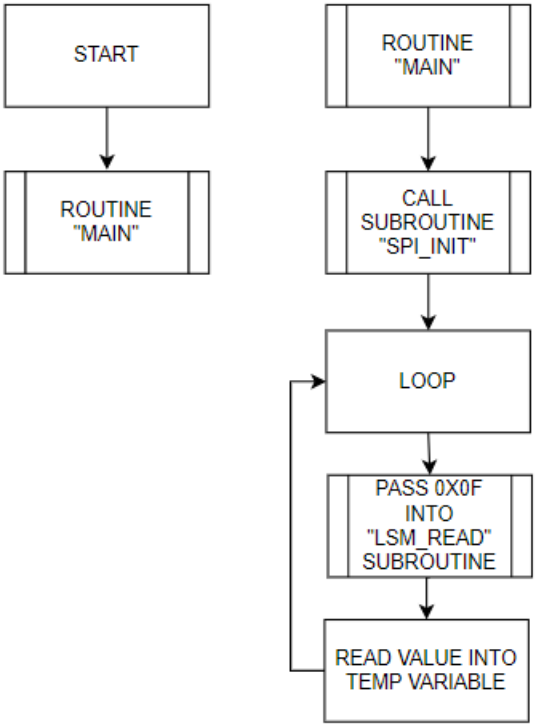

Figure 2: Flowchart for "lab6_3.C"

Section 5

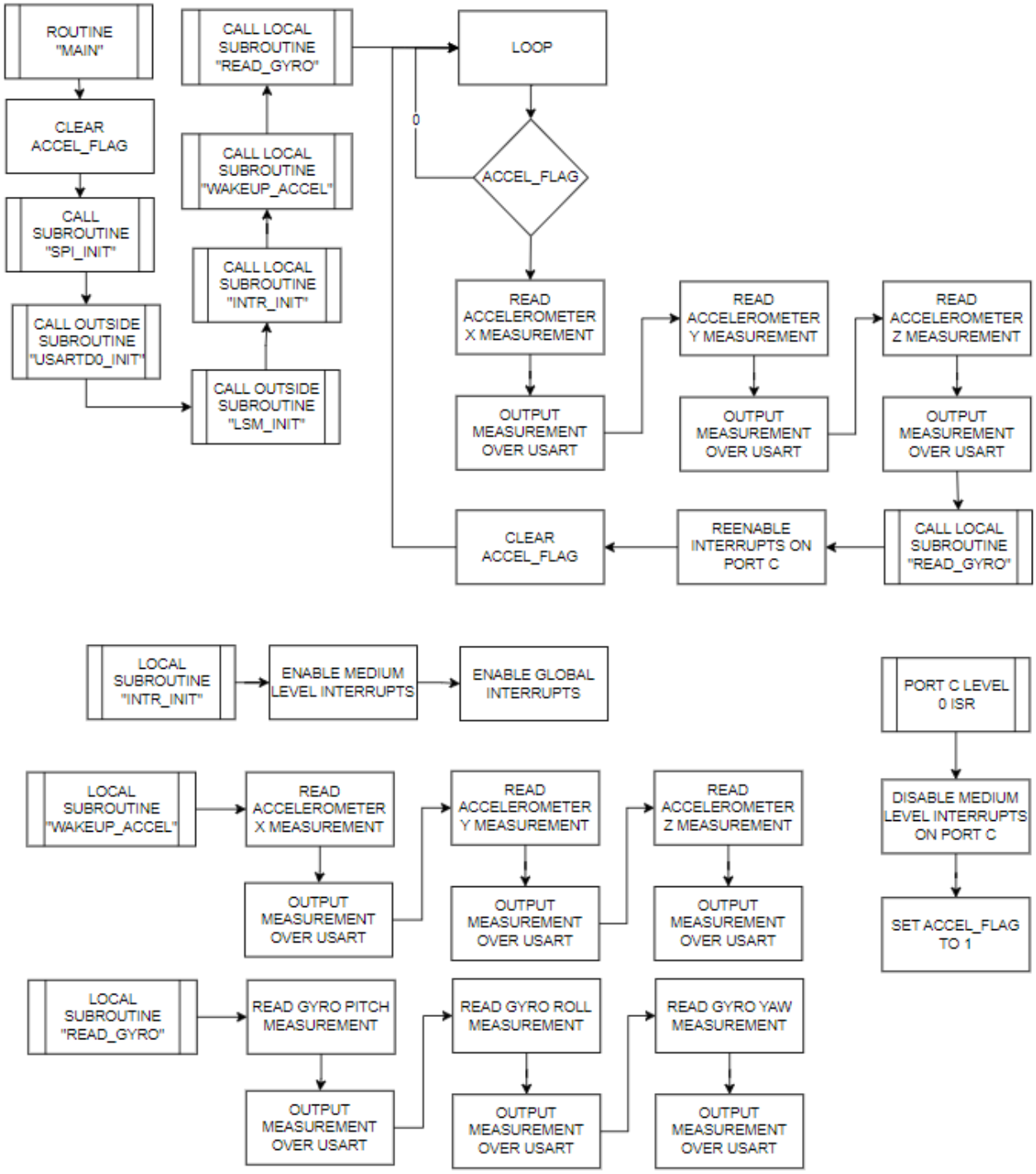

Figure 3: Flowcharts for "la6_5.C"

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

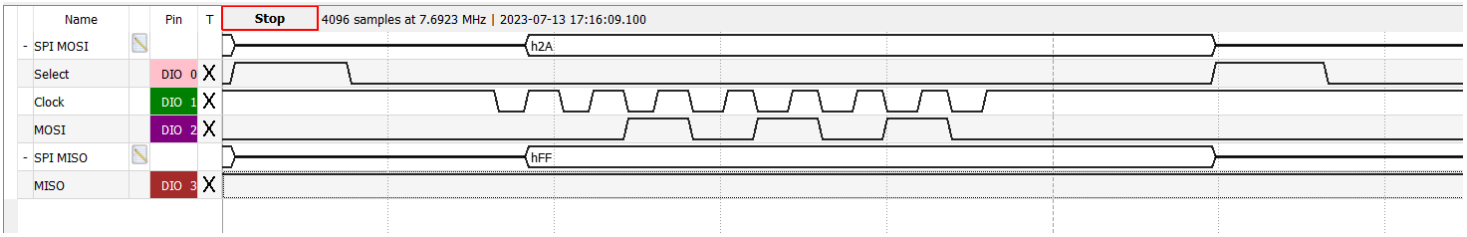

Figure 4: Measurement of "lab6_2.C"

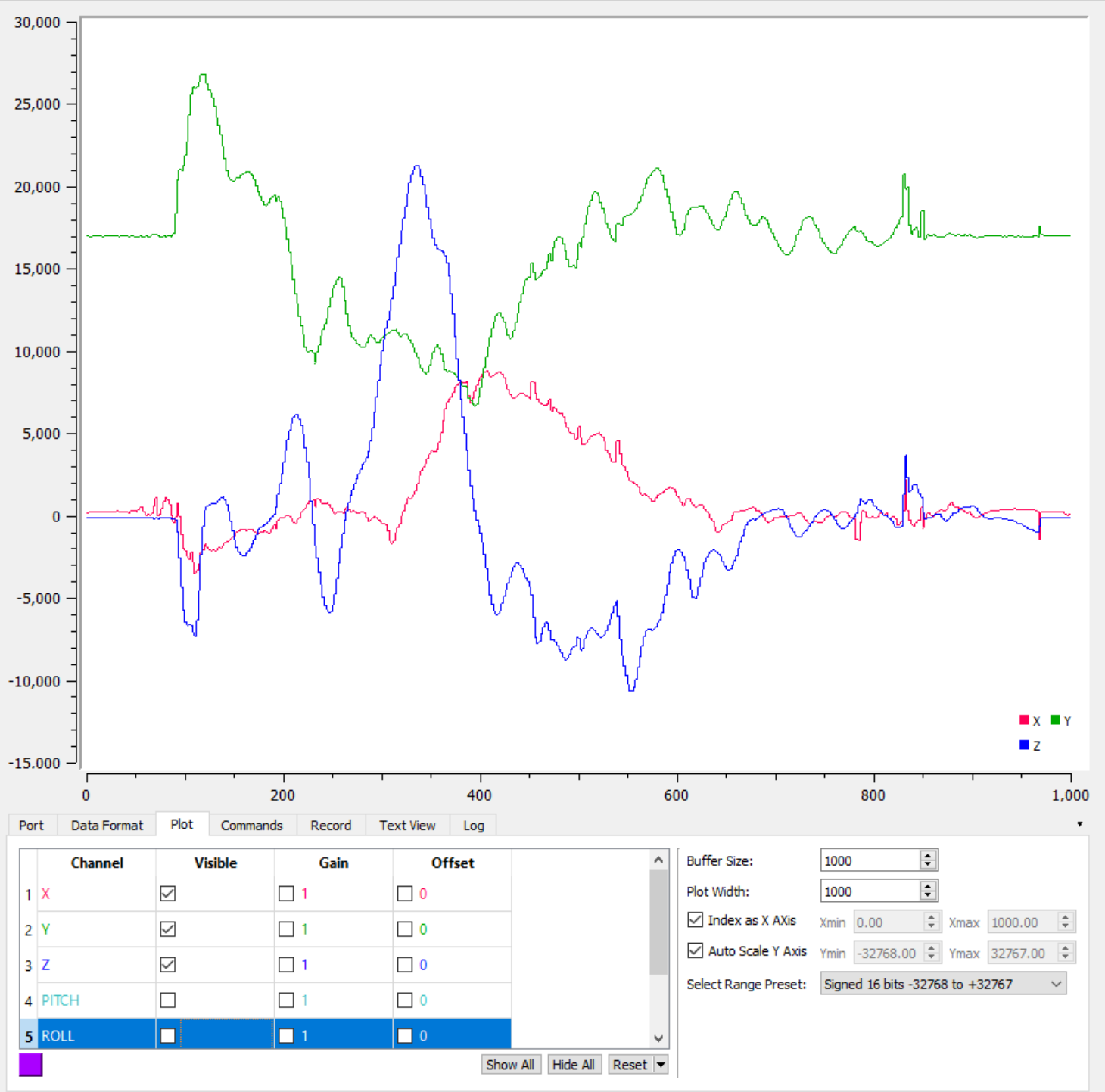

Figure 5: Screenshot of accelerometer plots

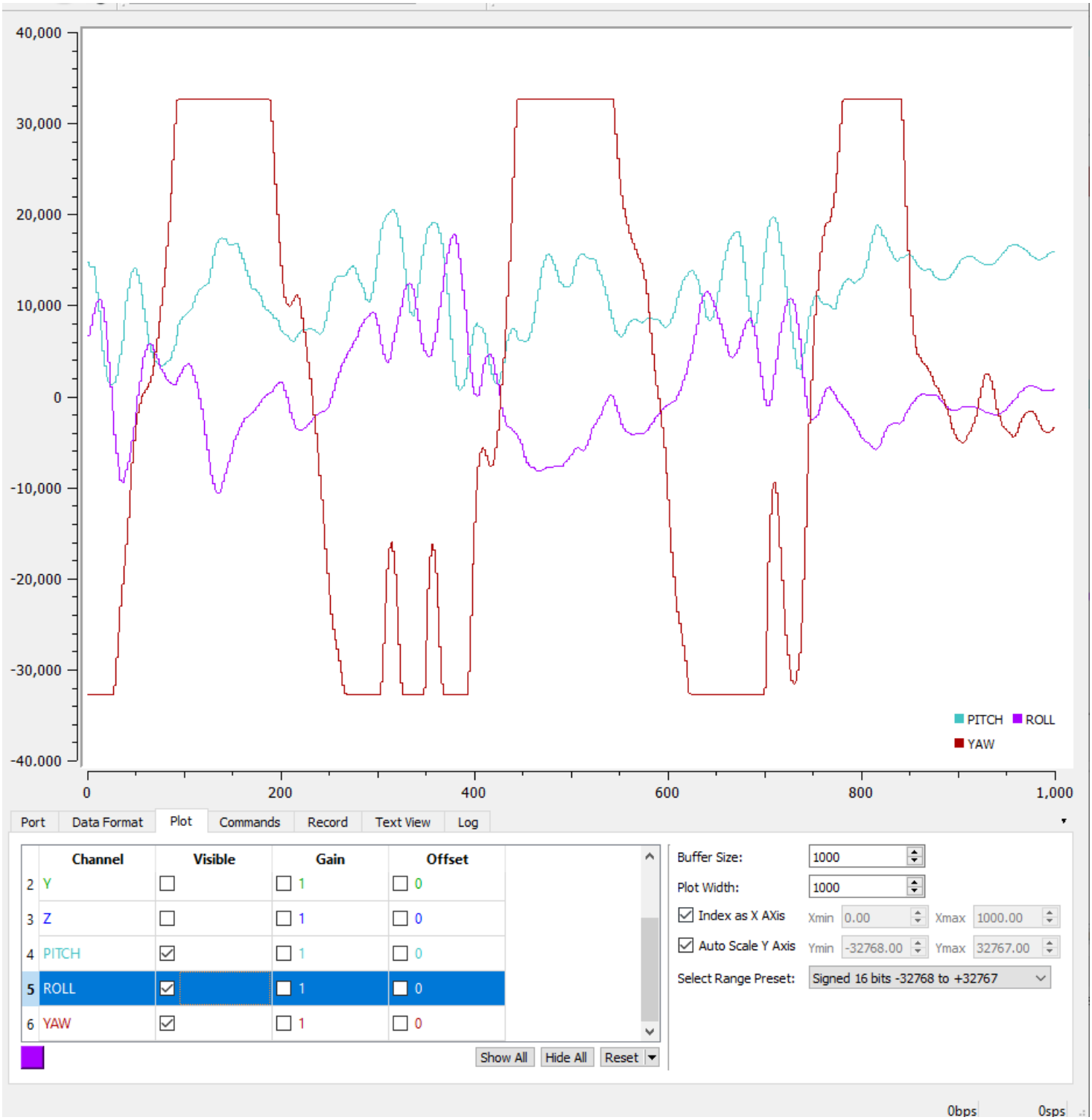

Figure 6: Screenshot of gyroscope plot

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.
```