REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

During this lab, the main issue I encountered was that SerialPlot displayed erratic and spiking accelerometer data. This happened because I had configured SerialPlot for uint16 instead of int16, which caused incorrect interpretation of the signed two's complement values. I also ran into a problem where the IMU was not generating interrupts, which I resolved by performing a dummy read after initialization to wake up the sensor. Additionally, I included an extra dummy write in my LSM_read function which caused the incorrect timing and prevented valid data from being received. I also passed the wrong parameter into one of my functions and ran into linker/complier errors when trying to incorporate a header file to use for lab6_5.c and the extracredit.c files. Ultimately, I resolved all these errors and the lab works as intended with the extra credit functioning too.

FUTURE WORK/APPLICATIONS

With additional time and resources, I could extend this project to record and analyze motion patterns over time rather than just displaying live data. The IMU readings could be stored in memory or sent to a computer for further processing, allowing for movement classification or activity detection. This system could then be applied to applications like step counting, vibration monitoring, or motion-based user input.

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.

   a. Which device(s) should be given the role of master and which device(s) should be given the role of student?
      i. The ATxmega128A1U should be given the role of master and the IMU should be given the role of student.

   b. 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?
      i. The student device will be enabled by driving its chip-select line low through a digital output pin on the ATxmegaA1U.
      ii. If a student select is utilized, rather than just have the device be permanently enabled the ~CS/I2C_EN (PF4) pin will be used.

   c. What is the order of data transmission? Is the MSb or LSb transmitted first?
      i. For the order of data transmission, the MSb is transmitted first.

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

   e. What is the maximum serial clock frequency that can be utilized by the relevant devices?
      i. The maximum serial clock frequency that can be utilized by the relevant devices is 10 MHz (limited by the IMU).

ii. Why is it a better idea to modify global flag variables inside of ISRs instead of doing everything inside of them?
   a. It is a better idea to modify global variables inside of ISRs instead of doing everything inside of them because ISRs should be kept as short and efficient as possible. Long or complex code inside an ISR can block other interrupts from being serviced and cause timing issues. By simply setting a global flag in the ISR, the main loop can check this flag and perform longer tasks, such as data processing or serial transmission, outside of the interrupt context.

iii. To output two unsigned 32-bit values 0xA4B3C2D1 [CH1] and 0x37012DEA [CH2] to SerialPlot, list all the bytes in the order you would send them via UART.
   a. To output two unsigned 32-bit values 0xA4B3C2D1 and 0x37012DEA to SerialPlot you would use the following byte order because SerialPlot takes data in little-endian format:
      i. D1, C2, B3, A4, EA, 2D, 01, 37

iv. What is the most positive value that can be received from the accelerometer (in decimal)? What about the most negative?
   a. The most positive value that can be received from the accelerometer is +32,767.
   b. The most negative value that can be received from the accelerometer is −32,768.
   c. This is because the accelerometer outputs signed 16-bit data in 2’s complement form.

PSEUDOCODE/FLOWCHARTS

SECTION 1

spi.c

Used spi.c skeleton

lab6_2.c

MAIN:

//init the SPI

SPI_INIT()

//configure CS pin as an output (PF4)

//set high initially

While(1)

{

Enable CS (PF4 to low)

Spi_write(0x89)

Disable CS (PF4 to high)

}

void LSM_write(uint8_t reg_addr, uint8_t data)

;enable the imu, pull cs low PF4

;send first byte with R/W bit and address

;bit 0 (msb) = low for write

;spi_write(reg_addr & 0x7f)

;send byte 2 with data

;spi_write(data)

;disable imu pull CS high PF4

uint8_t LSM_read(uint8_t reg_addr)

;enable the imu, pull cs low PF4

;send first byte with R/W bit and address

;bit 0 (msb) = high for read

;spi_write(reg_addr | (1<<7)

;send a dummy byte so the data transfers into our data register

;spi_write 0x55

;read data byte from IMU and store in variable

;received byte = spi_read()

;disable imu pull CS high PF4

;return received byte

Lab6_3.c

//include files

//initialize variables

//uint8_t who = 0

//init spi (call spi init)

//set CS (PF4) high initially

//set CS (PF4) as output

//who = LSM_read(WHO_AM_I);

//nop for breakpoint

While(1)

//no op

void LSM_init(void){

//perform software reset on IMU, 0x01 to CTRL3_C

```
//poll reset pin until it is 0
//configure CTRL9_XL to 0xE0 to enable accel data
//configure CTRL1_XL to 2G (00 to bits 23) and 104hz (high nibble to 4)
//configure INT1_CTRL to enable accel interrupts (set bit 0)
//configure I/O port Interrupt to active low input sense for PORTC pin 6
// PC6 as input
// trigger on low level
// mask PC6 for interrupt 0
// low-level interrupt priority
}
```

FOR PART 5:

void usartd0_out_bytes(uint8_t *data, uint8_t length)

```
{
   // initialize loop counter
   // loop through all bytes in array
   while (i < length)
   {
      // send one byte through UART
      // increment counter to move to next byte
   }
   // when all bytes sent, exit function
}
```

void send_accel_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H)

```
{
   // create temporary 6-byte array to hold all accelerometer data
   // store bytes in little-endian order (low byte first for each axis)
```

```
    // call helper function to send all 6 bytes through serial port
    // exit function
}
```

Lab6_5.c

```
//include files
//declare volatile global flag variable

void usartd0_out_bytes(uint8_t *data, uint8_t length)
{
    // initialize loop counter
    // loop through all bytes in array
    while (i < length)
    {
        // send one byte through UART
        // increment counter to move to next byte
    }
    // when all bytes sent, exit function
}

void send_accel_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H)
{
    // create temporary 6-byte array to hold all accelerometer data
    // store bytes in little-endian order (low byte first for each axis)
    // call helper function to send all 6 bytes through serial port
    // exit function
}

// interrupt service routine triggered when IMU pulls INT1 low
// sets flag to signal new data available and exits quickly
```

```
ISR(PORTC_INT0_vect)
{
   // disable further interrupts temporarily
   // set accel_flag = 1
   // exit ISR immediately (no SPI or UART operations)
}

// main routine: initializes everything and sends accelerometer data to SerialPlot
int main(void)
{
   // initialize USART with highest baud rate possible
   // initialize SPI using spi_init()
   // configure IMU using LSM_init()
   // perform one dummy read from accelerometer to wake up IMU
   // enable global interrupts (sei())

   // continuously loop
   while (1)
   {
      // check if accel_flag is set
      if (accel_flag == 1)
      {
         // clear flag
         accel_flag = 0;

         // read accelerometer data from IMU registers
         X_L = LSM_read(OUTX_L_XL);
         X_H = LSM_read(OUTX_H_XL);
         Y_L = LSM_read(OUTY_L_XL);
         Y_H = LSM_read(OUTY_H_XL);
```

```
            Z_L = LSM_read(OUTZ_L_XL);
            Z_H = LSM_read(OUTZ_H_XL);

            // send data to SerialPlot
            send_accel_data(X_L, X_H, Y_L, Y_H, Z_L, Z_H);

            // re-enable interrupt for next measurement
        }
    }
}
```

EXTRA CREDIT PSEUDOCODE:

```
// include files
#include <avr/io.h>
#include "spi.h"
#include "lsm6dsl.h"
#include "lsm6dsl_registers.h"
#include "usart.h"
#include <avr/interrupt.h>

// global flag
volatile uint8_t gyro_flag = 0;

// function prototypes
void usartd0_out_bytes(uint8_t *data, uint8_t length);
void send_gyro_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H);

// main
```

```
int main(void)
{
   uint8_t X_L, X_H, Y_L, Y_H, Z_L, Z_H;

   // initialize USART, SPI, and IMU
   usartd0_init();
   spi_init();
   LSM_init_gyro();   // (new function)

   // dummy read to wake up IMU
   LSM_read(OUTX_L_G);

   // enable interrupts
   PMIC.CTRL = PMIC_LOLVLEN_bm;
   sei();

   while (1)
   {
      if (gyro_flag == 1)
      {
         gyro_flag = 0;

         // read gyro data registers
         X_L = LSM_read(OUTX_L_G);
         X_H = LSM_read(OUTX_H_G);
         Y_L = LSM_read(OUTY_L_G);
         Y_H = LSM_read(OUTY_H_G);
         Z_L = LSM_read(OUTZ_L_G);
         Z_H = LSM_read(OUTZ_H_G);
```

```
            // send via UART to SerialPlot
            send_gyro_data(X_L, X_H, Y_L, Y_H, Z_L, Z_H);

            // re-enable interrupt
            PORTC.INTCTRL = PORT_INT0LVL_LO_gc;
        }
    }
}

// ISR (INT0)
ISR(PORTC_INT0_vect)
{
    PORTC.INTFLAGS = PORT_INT0IF_bm;
    gyro_flag = 1;
}

void LSM_init_gyro(void)
{
    // Configure INT input (same as accel setup)
    // Software reset

    // Common control setup (active-low interrupt, auto-increment)
    #define CTRL3C_BDU_bm (1<<6)
    #define CTRL3C_H_LACTIVE (1<<5)
    #define CTRL3C_IF_INC_bm (1<<2)
    LSM_write(CTRL3_C, CTRL3C_BDU_bm | CTRL3C_IF_INC_bm | CTRL3C_H_LACTIVE);

    // Enable gyroscope data-ready interrupt

    // Configure gyroscope (CTRL2_G)
```

```
    // 0x40 = 104 Hz
}

void usartd0_out_bytes(uint8_t *data, uint8_t length)
{
    for (uint8_t i = 0; i < length; i++)
        usartd0_out_char(data[i]);
}

void send_gyro_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H)
{
    uint8_t gyro_bytes[6] = {X_L, X_H, Y_L, Y_H, Z_L, Z_H};
    usartd0_out_bytes(gyro_bytes, 6);
}
```

PROGRAM CODE

SECTION 2

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: 13 Oct 2025
-----------------------------------------------------------------------------*/
/*****************************************************
 * Lab 6, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Initializes and provides functions for SPI communication
 *   on the ATxmega128A1U. Includes initialization, write,
 *   and read routines used to communicate with the LSM6DSL
 *   IMU over Port F (Mode 3, MSB first, ≤10 MHz).
 *****************************************************/


/*********************************DEPENDENCIES*********************************/

#include <avr/io.h>
#include "spi.h"

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


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

/*****************************************************
* Name: spi_init
* Purpose: Initialize SPI (master, mode 3, MSB first).
* Inputs:  None
* Outputs: None
*****************************************************/
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 =  PIN4_bm | PIN7_bm;

  /* Configure the pin direction of relevant SPI signals. */
  PORTF.DIRSET = PIN7_bm | PIN5_bm | PIN4_bm;
  PORTF.DIRCLR = PIN6_bm;

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

```
  SPIF.CTRL   =     SPI_PRESCALER_DIV16_gc               |
                                    SPI_MASTER_bm      |
                                    SPI_MODE_3_gc        |
                                    SPI_ENABLE_bm;
}

/*****************************************************
* Name: spi_write
* Purpose: Transmit one byte via SPI.
* Inputs:  data - byte to send
* Outputs: None
*****************************************************/
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));

  /* 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. */
}

/*****************************************************
* Name: spi_read
* Purpose: Read one byte from SPI (sends dummy 0x37).
* Inputs:  None
* Outputs: Returns received byte
*****************************************************/
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));

  /* After the transmission, return the data that was received. */
  return SPIF.DATA;
}

/*************************END OF FUNCTION DEFINITIONS*************************/
```

Lab6_2.c

```
/*
 * lab6_2.c
 *
 * Created: 10/25/2025 3:52:22 PM
 *  Author: arist
 */
/*****************************************************
 * Lab 6, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Main program for verifying SPI transmit
 *   functionality by sending 0x89 repeatedly
 *   with chip-select toggle on PF4.
 *****************************************************/


#include <avr/io.h>

#include "spi.h"
int main(void)
{

//init the SPI
 spi_init();
//configure CS pin as an output (PF4)
PORTF.OUTSET = PIN4_bm;
//set high initially
PORTF.DIRSET = PIN4_bm;

while(1)
{
       //Enable CS (PF4 to low)
       PORTF.OUTCLR = PIN4_bm;

 spi_write(0x89);

       //Disable CS (PF4 to high)
       PORTF.OUTSET = PIN4_bm;
}

return 0;
}
```

Lab6_3.c

```c
/*
 * lab6_3.c
 *
 * Created: 11/1/2025 12:53:43 PM
 *  Author: arist
 */

/*****************************************************
 * Lab 6, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Reads the WHO_AM_I register from the LSM6DSL IMU
 *   over SPI to verify receive functionality. Initializes
 *   the SPI interface, configures chip select on PF4,
 *   and repeatedly reads the device ID for verification.
 *****************************************************/


//include files
#include <avr/io.h>
#include "spi.h"
#include "lsm6dsl.h"
#include "lsm6dsl_registers.h"

int main(void)
{

	//initialize variables
	uint8_t who = 0xFF;

	//init spi (call spi init)
	spi_init();
	//set CS (PF4) high initially
	PORTF.OUTSET = PIN4_bm;
	//set CS (PF4) as output
	PORTF.DIRSET = PIN4_bm;

	//read from who am I
	who = LSM_read(WHO_AM_I);


	while(1)
	{
		//for dad captures
		who = LSM_read(WHO_AM_I);
	}

	return 0;
}
```

Lab6_5.c

```
/*
 * lab6_5.c
 *
 * Created: 11/1/2025 5:34:20 PM
 *  Author: arist
 */

/*****************************************************
 * Lab 6, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Streams real-time 3-axis accelerometer data from the
 *   LSM6DSL IMU to SerialPlot using SPI and USARTD0.
 *   Initializes SPI, USART, and IMU; uses an interrupt
 *   service routine on PORTC to set a flag when new data
 *   is available; transmits accelerometer data in
 *   little-endian format for plotting.
 *****************************************************/

//include files
#include <avr/io.h>
#include "spi.h"
#include "lsm6dsl.h"
#include "lsm6dsl_registers.h"
#include <avr/interrupt.h>
#include "usart.h"
//#include "lab6_5.h"

// global flag to indicate when new IMU data is ready
volatile uint8_t accel_flag = 0;

//function prototypes
//void usartd0_out_bytes(uint8_t *data, uint8_t length);
//void send_accel_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H);



/*****************************************************
* Name: usartd0_out_bytes
* Purpose: Transmit a sequence of bytes via USARTD0.
* Inputs:
*   - data: pointer to an array of bytes to transmit
*   - length: number of bytes to send
* Outputs: None
*****************************************************/
void usartd0_out_bytes(uint8_t *data, uint8_t length)
{
       // loop through all bytes in array
 for (uint8_t i = 0; i < length; i++)
 {
        // send one byte through UART
        usartd0_out_char(data[i]);
 }
}


/*****************************************************
```

```
* Name: send_accel_data
* Purpose: Send 3-axis accelerometer data to SerialPlot
*          in little-endian format (6 bytes total).
* Inputs:
*   - X_L, X_H: low/high bytes of X-axis data
*   - Y_L, Y_H: low/high bytes of Y-axis data
*   - Z_L, Z_H: low/high bytes of Z-axis data
* Outputs: None
*****************************************************/
void send_accel_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H)
{
	// create temporary 6-byte array to hold all accelerometer data
	// store bytes in little-endian order (low byte first for each axis)
	   uint8_t accel_bytes[6] = {X_L, X_H, Y_L, Y_H, Z_L, Z_H};
	// call helper function to send all 6 bytes through serial port
	usartd0_out_bytes(accel_bytes, 6);

}

// main routine: initializes everything and sends accelerometer data to SerialPlot
int main(void)
{
	uint8_t X_L, X_H, Y_L, Y_H, Z_L, Z_H;
	// initialize USART with highest baud rate possible
	usartd0_init();
	// initialize SPI using spi_init()
	spi_init();
	// configure IMU using LSM_init()
	LSM_init();
	// perform one dummy read from accelerometer to wake up IMU
	LSM_read(OUTX_L_XL);
	// enable pmic and global interrupts (sei())
	PMIC.CTRL = PMIC_LOLVLEN_bm;
	sei();
	// continuously loop
	while (1)
	{
		// check if accel_flag is set
		if (accel_flag == 1)
		{
			// clear flag
			accel_flag = 0;

			// read accelerometer data from IMU registers
			X_L = LSM_read(OUTX_L_XL);
			X_H = LSM_read(OUTX_H_XL);
			Y_L = LSM_read(OUTY_L_XL);
			Y_H = LSM_read(OUTY_H_XL);
			Z_L = LSM_read(OUTZ_L_XL);
			Z_H = LSM_read(OUTZ_H_XL);

			// send data to SerialPlot
			send_accel_data(X_L, X_H, Y_L, Y_H, Z_L, Z_H);

			// re-enable interrupt for next measurement
			PORTC.INTCTRL = PORT_INT0LVL_LO_gc;
		}
	}
}
```

```
/*****************************************************
* Name: ISR(PORTC_INT0_vect)
* Purpose: Handle interrupt from IMU INT1 pin (active low)
*          to indicate new accelerometer data is ready.
* Inputs:  None
* Outputs: None
*****************************************************/
// interrupt service routine triggered when IMU pulls INT1 low
// sets flag to signal new data available and exits quickly
ISR(PORTC_INT0_vect)
{
	// clear interrupt flag first
	PORTC.INTFLAGS = PORT_INT0IF_bm;
	// set accel_flag = 1
	accel_flag = 1;
	// exit ISR immediately
}
```

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: 13 Oct 2025
    Last modified by: Arion Stern
    Last modified on: 2 Nov 2025

       /*****************************************************
 * Lab 6, Section 22
 * Name: Arion Stern
 * Class #: 11303
 * PI Name: Ian Santamauro
 * Description:
 *   Defines functions for communicating with and configuring
 *   the LSM6DSL IMU over SPI. Includes read/write routines,
 *   accelerometer initialization, and gyroscope setup for
 *   extra credit.
 *****************************************************/
-----------------------------------------------------------------------------*/

/*********************************DEPENDENCIES*********************************/

#include <avr/io.h>
#include "lsm6dsl.h"
#include "lsm6dsl_registers.h"

#define CTRL3C_BDU_bm     (1 << 6)
#define CTRL3C_H_LACTIVE  (1 << 5)
#define CTRL3C_IF_INC_bm  (1 << 2)

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


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

/*****************************************************
* Name: LSM_write
* Purpose: Write a single byte to an LSM6DSL register via SPI.
* Inputs:
*   - reg_addr: 8-bit address of target register
*   - data: byte value to write
* Outputs: None
*****************************************************/
void LSM_write(uint8_t reg_addr, uint8_t data)
{
       //enable the imu, pull cs low PF4
       PORTF.OUTCLR = PIN4_bm;
       //;send first byte with R/W bit and address
       //bit 0 (msb) = low for write
       //spi_write(reg_addr & 0x7f)
       spi_write(reg_addr & 0x7F);
```

```
	//send byte 2 with data
	//spi_write(data)
	spi_write(data);
	//disable imu pull CS high PF4
	PORTF.OUTSET = PIN4_bm;

}

/*****************************************************
* Name: LSM_read
* Purpose: Read a single byte from an LSM6DSL register via SPI.
* Inputs:
*   - reg_addr: 8-bit address of target register
* Outputs:
*   - Returns the 8-bit value read from the IMU register
*****************************************************/
uint8_t LSM_read(uint8_t reg_addr)
{
	//enable the imu, pull cs low PF4
	PORTF.OUTCLR = PIN4_bm;
	//send first byte with R/W bit and address
	//bit 0 (msb) = high for read
	//spi_write(reg_addr | (1<<7)
	spi_write(reg_addr | 0x80);
	//send a dummy byte so the data transfers into our data register
	//spi_write 0x55
	//spi_write(0x55);
	//read data byte from IMU and store in variable
	//received byte = spi_read()
	uint8_t byte_rec = spi_read();
	//disable imu pull CS high PF4
	PORTF.OUTSET = PIN4_bm;
	//return received byte
	return byte_rec;
}

/*****************************************************
* Name: LSM_init
* Purpose: Initialize the LSM6DSL accelerometer and configure
*          interrupt behavior for data-ready signaling.
* Inputs:  None
* Outputs: None
*****************************************************/
void LSM_init(void)
{
//configure I/O port Interrupt to active low input sense for PORTC pin 6
// PC6 as input
PORTC.DIRCLR = PIN6_bm;
// trigger on low level
PORTC.PIN6CTRL = PORT_ISC_LEVEL_gc;
// mask PC6 for interrupt 0
PORTC.INT0MASK = PIN6_bm;
// low-level interrupt priority
PORTC.INTCTRL = PORT_INT0LVL_LO_gc;

//perform software reset on IMU, 0x01 to CTRL3_C
LSM_write(CTRL3_C, 0x01);
//poll reset pin until it is 0
while(LSM_read(CTRL3_C) & 0x01);

// After reset completes:
```

```
LSM_write(CTRL3_C, CTRL3C_BDU_bm | CTRL3C_IF_INC_bm | CTRL3C_H_LACTIVE);

//configure CTRL9_XL to 0xE0 to enable accel data
LSM_write(CTRL9_XL, 0xE0);
//configure CTRL1_XL to 2G (00 to bits 23) and 104hz (high nibble to 4)
LSM_write(CTRL1_XL,  0x30);
//configure INT1_CTRL to enable accel interrupts (set bit 0)
LSM_write(INT1_CTRL, 0x01);


}


/*****************************************************
* Name: LSM_init_gyro
* Purpose: Configure the LSM6DSL gyroscope for output and
*          data-ready interrupts (extra credit).
* Inputs:  None
* Outputs: None
*****************************************************/
void LSM_init_gyro(void)
{
    // configure I/O port interrupt to active-low input sense on PC6
    PORTC.DIRCLR = PIN6_bm;              // PC6 input
    PORTC.PIN6CTRL = PORT_ISC_LEVEL_gc;  // trigger on low level
    PORTC.INT0MASK = PIN6_bm;            // mask PC6
    PORTC.INTCTRL = PORT_INT0LVL_LO_gc;  // low-level interrupt priority

    // perform IMU software reset
    LSM_write(CTRL3_C, 0x01);
    while (LSM_read(CTRL3_C) & 0x01); // wait until reset finishes

    // enable Block Data Update, auto-increment, active-low interrupt
    LSM_write(CTRL3_C, (1 << 6) | (1 << 5) | (1 << 2));

    // enable gyroscope axes (X, Y, Z)
    LSM_write(CTRL10_C, 0x00);

    // set gyro full-scale = 104 hz
    LSM_write(CTRL2_G, 0x40);

    // route gyro data-ready interrupt to INT1 (bit 1)
    LSM_write(INT1_CTRL, 0x02);
}


/* INSERT YOUR LSM6DSL FUNCTION DEFINITIONS BELOW. */

/*************************END OF FUNCTION DEFINITIONS************************/
```

extracredit.c

```
/*
 * extracredit.c
 *
 * Created: 11/1/2025 9:38:34 PM
 *  Author: arist
 */

/*

 *
 * Created: 11/1/2025 10:45:00 PM
 *  Author: Arion Stern
 *  Description:
 *    Reads gyroscope data (X, Y, Z) from the LSM6DSL IMU
 *    and transmits it via USART to SerialPlot in real-time
 *    using Simple Binary (3 channels, int16, little-endian).
 */

#include <avr/io.h>
#include "spi.h"
#include "lsm6dsl.h"
#include "lsm6dsl_registers.h"
#include "usart.h"
#include "lab6_5.h"
#include <avr/interrupt.h>

// global flag for new gyro data
volatile uint8_t gyro_flag = 0;

// transmit an array of bytes through USART
void usartd0_out_bytes(uint8_t *data, uint8_t length)
{
	for (uint8_t i = 0; i < length; i++)
	{
		usartd0_out_char(data[i]);
	}
}

// send all 3 gyroscope axis values (6 bytes total) through USART
void send_accel_data(uint8_t X_L, uint8_t X_H, uint8_t Y_L, uint8_t Y_H, uint8_t Z_L, uint8_t Z_H)
{
	uint8_t gyro_bytes[6] = {X_L, X_H, Y_L, Y_H, Z_L, Z_H};
	usartd0_out_bytes(gyro_bytes, 6);
}


/*********************************************
 * MAIN
 *********************************************/
int main(void)
{
    uint8_t X_L, X_H, Y_L, Y_H, Z_L, Z_H;

    // initialize communication modules
    usartd0_init();
    spi_init();
    LSM_init_gyro();
```

```
    // perform dummy read to wake up device
   LSM_read(OUTX_L_G);

    // enable PMIC and global interrupts
    PMIC.CTRL = PMIC_LOLVLEN_bm;
    sei();

    while (1)
    {
        if (gyro_flag == 1)
        {
            gyro_flag = 0;

            // read all 3 gyro axes
            X_L = LSM_read(OUTX_L_G);
            X_H = LSM_read(OUTX_H_G);
            Y_L = LSM_read(OUTY_L_G);
            Y_H = LSM_read(OUTY_H_G);
            Z_L = LSM_read(OUTZ_L_G);
            Z_H = LSM_read(OUTZ_H_G);

            // send in little-endian order (6 bytes total)
            send_accel_data(X_L, X_H, Y_L, Y_H, Z_L, Z_H);

            // re-enable interrupt for next measurement
            PORTC.INTCTRL = PORT_INT0LVL_LO_gc;
        }
    }
}

/*********************************************
 * INTERRUPT
 *********************************************/
ISR(PORTC_INT0_vect)
{
    // clear interrupt flag
    PORTC.INTFLAGS = PORT_INT0IF_bm;
    // set gyro flag
    gyro_flag = 1;
}
```

APPENDIX

Supporting ASM/C Code and Additional Information

- Included/Referenced Files and Headers:
    - The files/headers contained in the lab6_files.zip
- Additional Screenshots/Images:

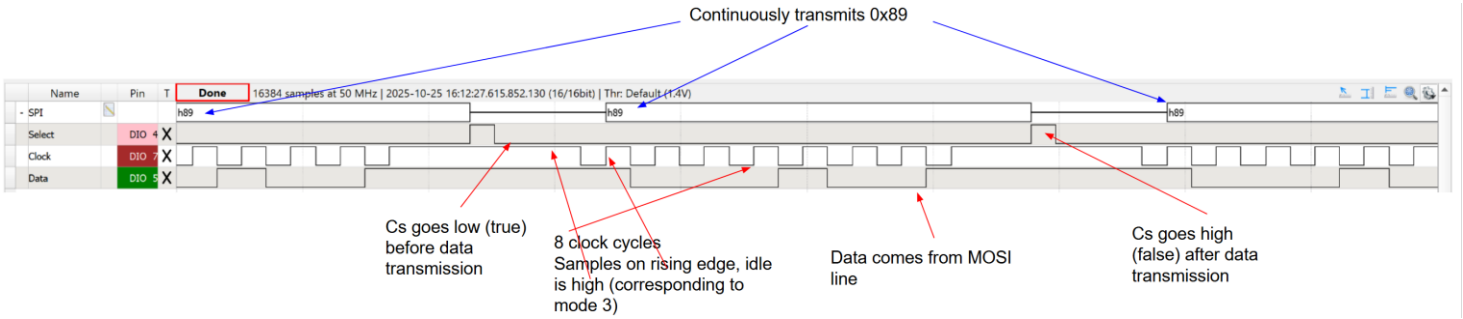

Figure 1: Logic Analyzer capture verifying SPI transmission of 0x89 from the ATxmega128A1U master to the IMU student device. The Select (CS) line goes low before transmission and returns high afterward. Eight clock pulses are generated on the Clock line with data sampled on the rising edge and idle high—confirming correct SPI Mode 3 operation. The Data (MOSI) line shows the transmitted byte 0x89.

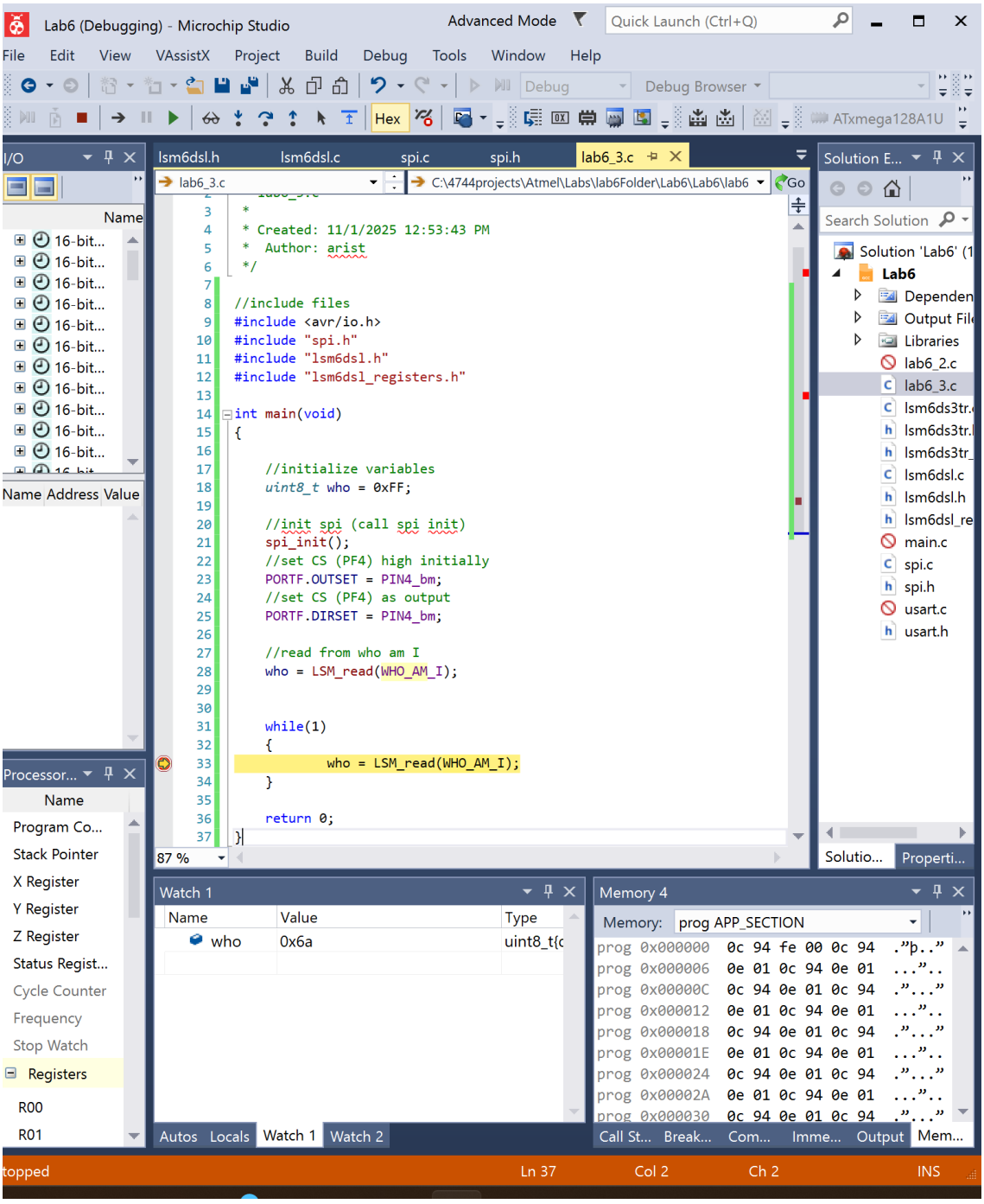

Figure 2: Debug session in Microchip Studio verifying the WHO_AM_I register read from the LSM6DSL IMU. The who variable, displayed in the Watch window, holds the value 0x6A, confirming correct SPI communication and device identification. The variable was initialized to 0xFF prior to the read, and the updated value verifies successful data transfer from the IMU.

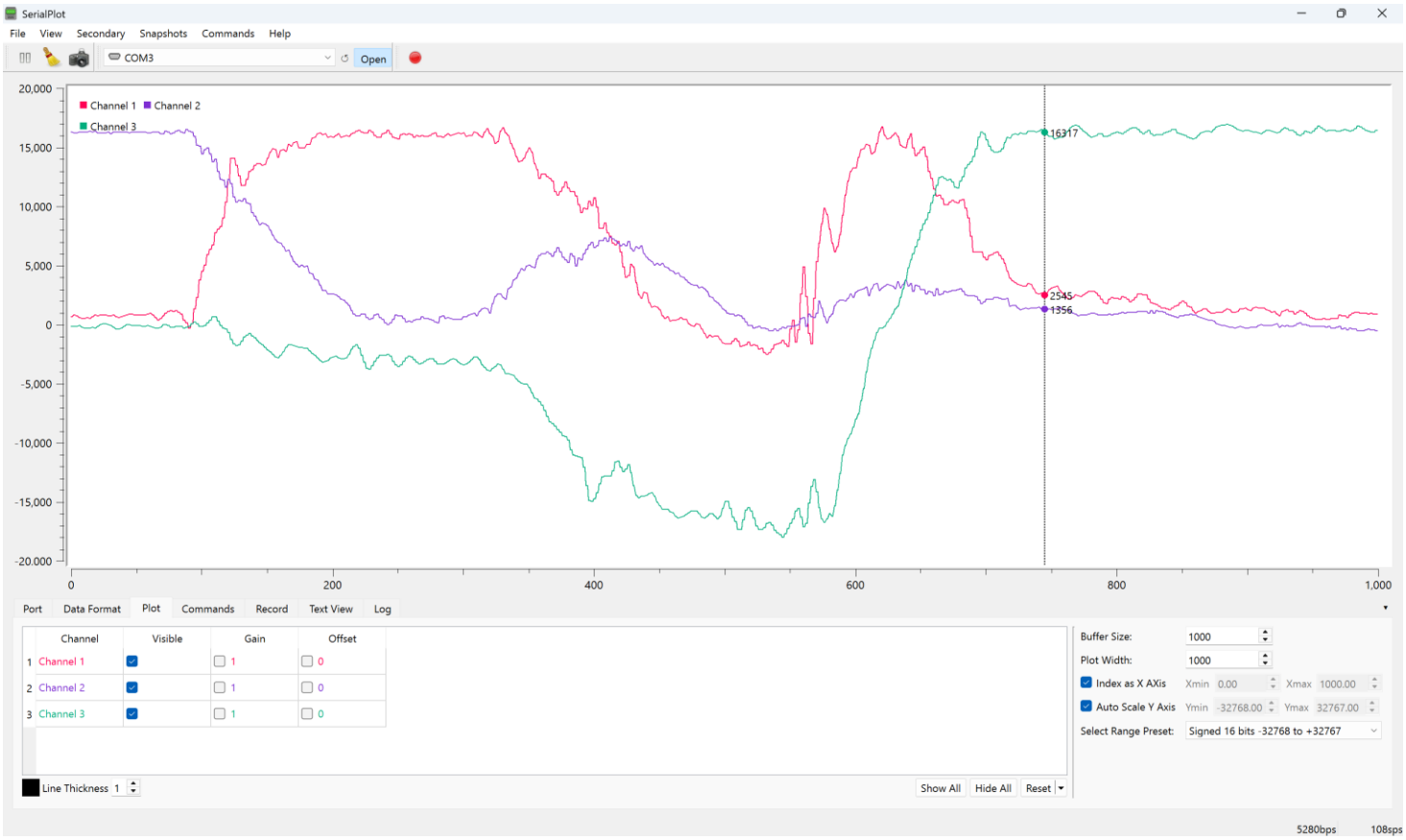

Figure 3: Real-time 3-axis accelerometer data from the LSM6DSL IMU displayed in SerialPlot. Channel 1 (red), Channel 2 (purple), and Channel 3 (green) represent the X, Y, and Z acceleration axes respectively. The continuous waveforms confirm that the SPI and USART interfaces are functioning correctly to stream signed 16-bit acceleration data in little-endian format. Variations in each axis correspond to changes in board orientation and gravitational influence.

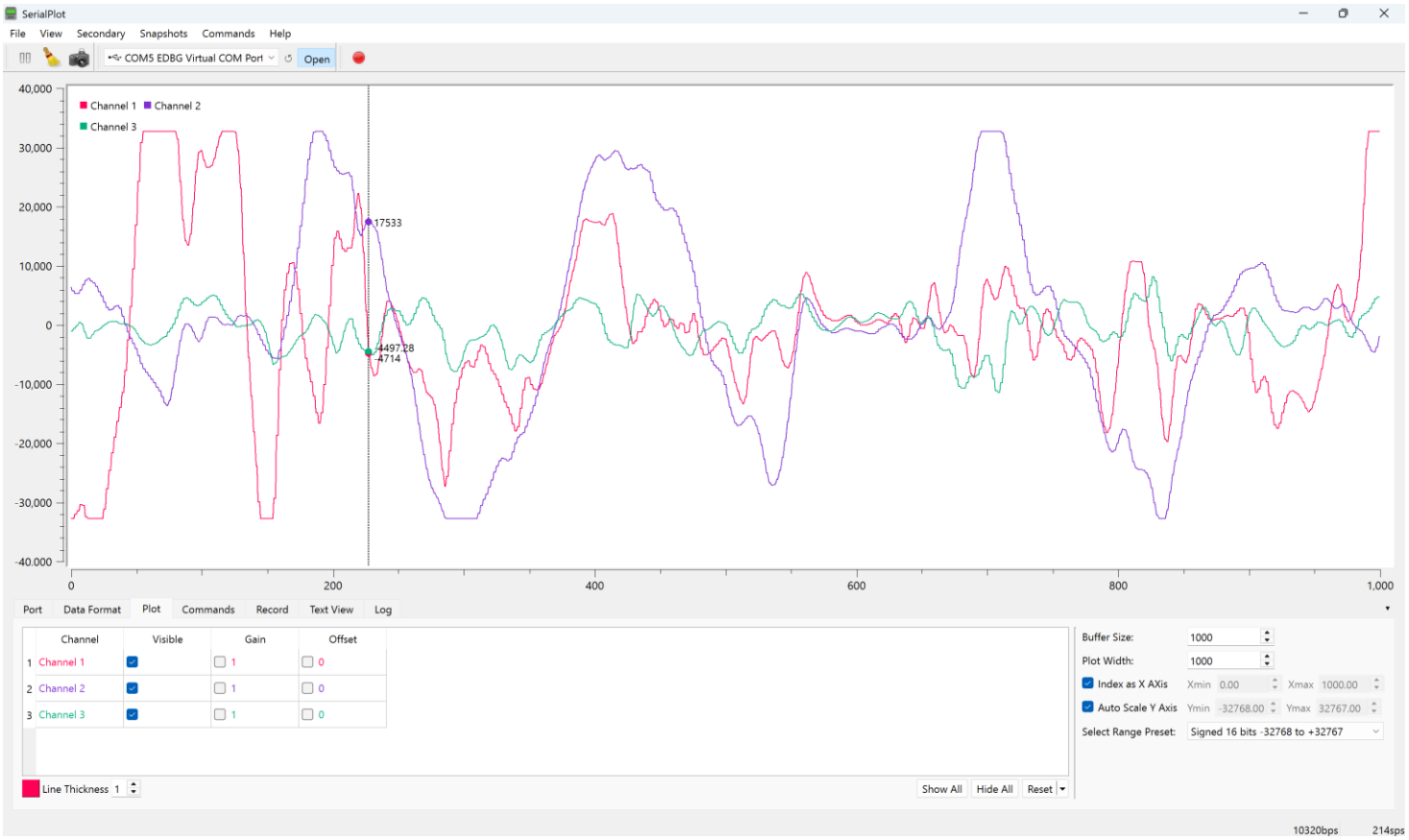

Figure 4: Real-time 3-axis gyroscope angular velocity data from the LSM6DSL IMU displayed in SerialPlot. Channel 1 (red), Channel 2 (purple), and Channel 3 (green) represent the X, Y, and Z rotational axes respectively. The waveforms confirm that the gyroscope was successfully initialized and configured for 104 Hz output and SPI-based data streaming. Variations in each axis correspond to changes in rotational motion of the µPAD and Robotics Backpack assembly.