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 IMU transmits data MSB first.
The ATX can transmit data either LSB or 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 then.

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
Therefore, the ATX will need to receive at its max frequency of 1MHZ

PSEUDOCODE/FLOWCHARTS

SECTION 2

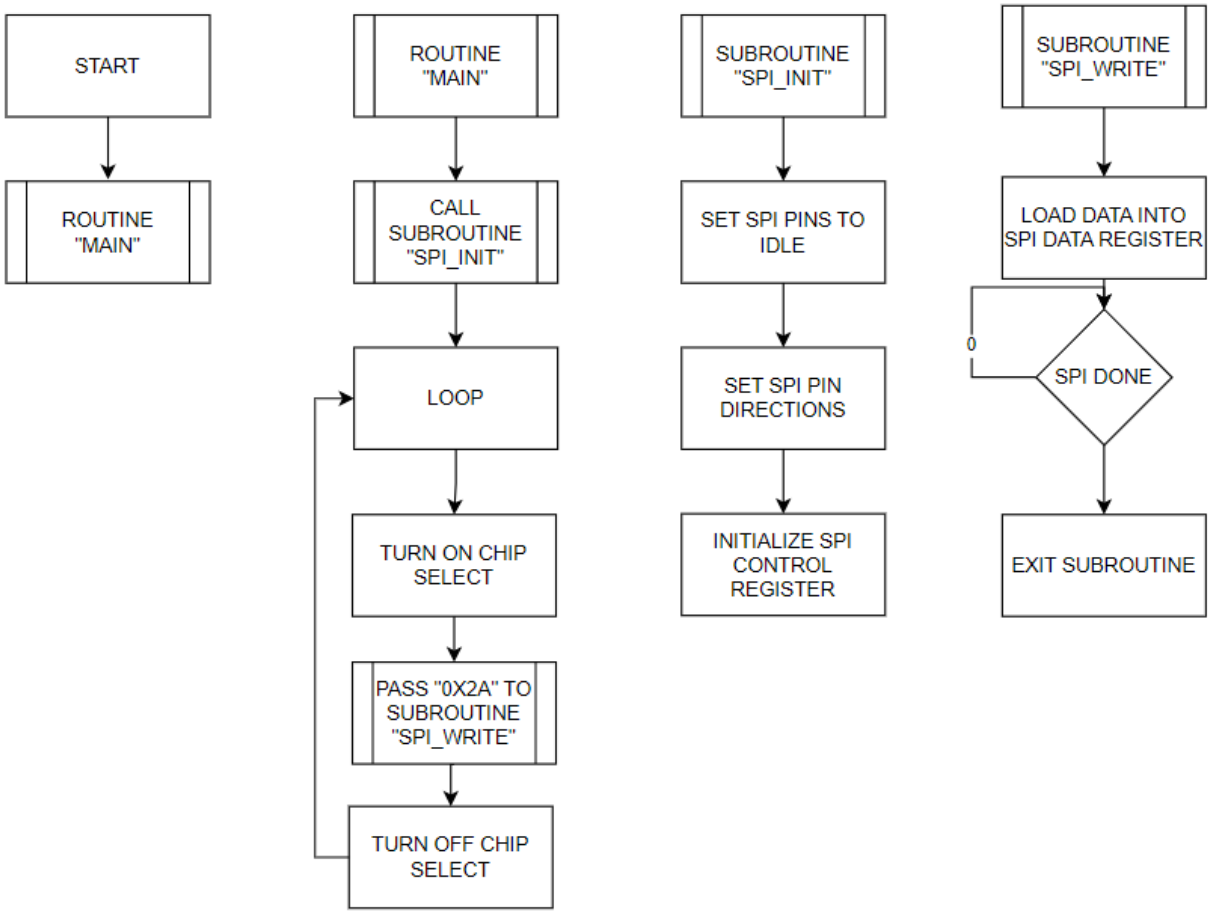

Figure 1: Flowchart for “lab6_2.C”

PROGRAM CODE

SECTION 2

```
//****************************************
//Lab 6, Section 2
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: continuously sends data over spi
//****************************************

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

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

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


/*************************MAIN**********************/
int main(void)
{
	spi_init();
	while(1)
	{
		//turn on chip select
		PORTF.OUTCLR = SS_bm;
		spi_write(0x2a);
		//turn off chip select
		PORTF.OUTSET = SS_bm;
	}
	return 0;
}
```

APPENDIX

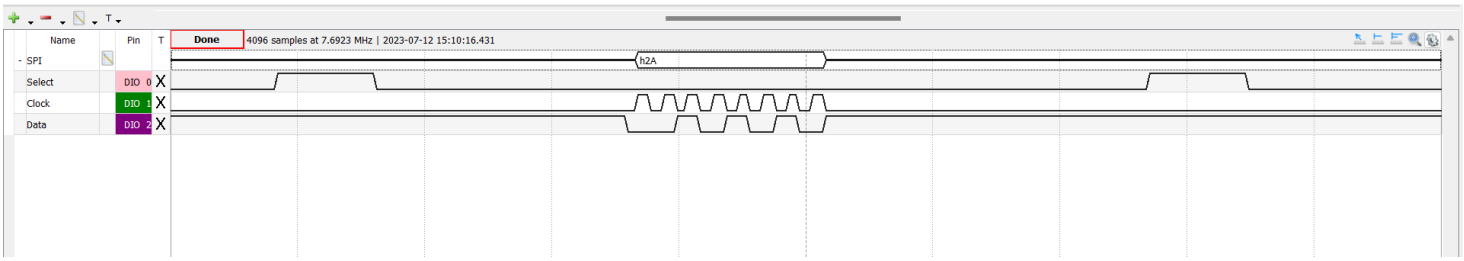

Figure 2: Measurement of "lab6_2.C"

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.
```

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|MOSI_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_0_gc|SPI_ENABLE_bm| SPI_CLK2X_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 while we wait
       }
  /* 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 while we wait
  }

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

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