REQUIREMENTS NOT MET

Section 3,4,5 not completed

PROBLEMS ENCOUNTERED

I don’t know why but I just couldn’t seem to grasp this lab. I read the documentation and watched all of the videos but was stuck on Section 2 for the longest time. I gave myself plenty of time to do the lab, but I think that I expected the lab to be easier than it was. I thought it would be easier because we were writing in C instead of assembly. Also, the instructions seemed very straightforward, so when I ran into a problem, I didn’t think it would be too bad. But I felt completely lost just on section 2. Because of that, I didn’t go on to any other section. I ended up getting Section 2 to work at the last minute, but it was too late to really do any other part of the lab. My problem was simply the Prescaler amount. One thing that was annoying though, is that I looked at the wrong backpack schematic for the longest time. It wasn’t making sense to me and I thought that was where a lot of my problems were. Then I found out there was another schematic that was the correct one.

Sorry that this is a bad lab report. I don’t like turning in work like this, but I figured I would rather turn this in than not turn in anything.

FUTURE WORK/APPLICATIONS

Once I complete this I feel like I can answer this question than I can now, but my current answer is that understanding communication is super important. Understanding this lab can give a programmer powerful tools that allow data to flow smoothly and quickly which is always a good thing.

PRE-LAB EXERCISES

Section 1

i. Which device(s) should be given the role of master and which device(s) should be given the role of slave?

The ATxmega128A1U should have the role of master, and the LSM6DS3 should have the role of slave.

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

Because there is only one slave device, we can just always enable that Slave enable. A better way is to use the CS for our slave device. We can turn it on and off when we want to use it.

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

The order of data transmission is MSb first

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

Data should be latched of a falling edge.

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

I honestly don't know. I thought that it was around 10MHz, but now I really don't know.

PSEUDOCODE/FLOWCHARTS

N/A

PROGRAM CODE

SECTION 1

spi.c code:

```
/*------------------------------------------------------------------------------
  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: Christopher Crary
  Last modified on: 18 July 2020
------------------------------------------------------------------------------*/

/********************************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 LSM6DS3 datasheet.
   * (You may wish to utilize the macros defined in `spi.h`.) */
       PORTF.OUTSET =      SS_bm  |
                                  SCK_bm |
                                  MOSI_bm;//idle is high

  /* Configure the pin direction of relevant SPI signals. */
       PORTF.DIRSET =      SS_bm  |
                                  SCK_bm |
                                  MOSI_bm;

       PORTF.DIRCLR =      MISO_bm; //only input for master

       /* Set the other relevant SPI configurations. */
       SPIF.CTRL     =             SPI_PRESCALER_DIV128_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 != 0x80);
```

```c
  /* 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);

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

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

spi.h code:

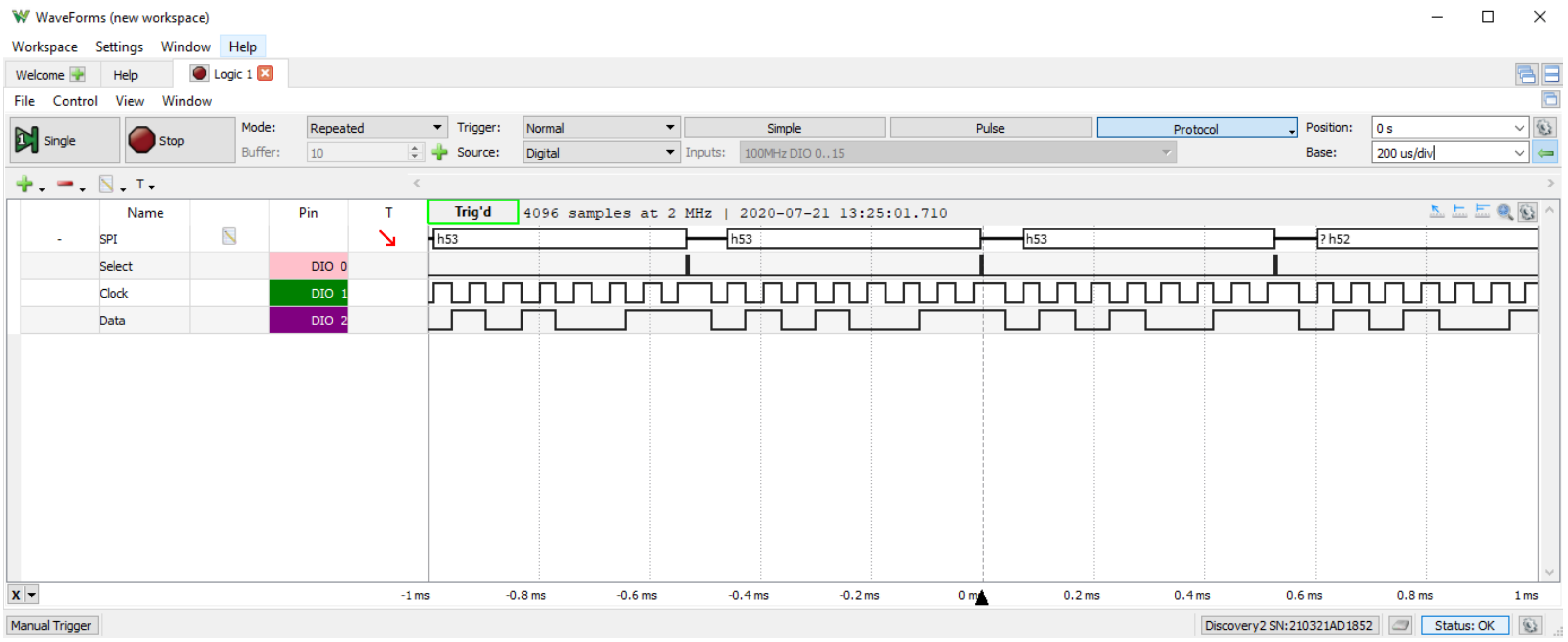
```
#ifndef SPI_H_              // Header guard.
#define SPI_H_

/*------------------------------------------------------------------------------
  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: Christopher Crary
  Last modified on: 18 July 2020
------------------------------------------------------------------------------*/

/********************************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 LSM6DS3.

  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.
```

lab6_2.c code:

```
/*
 * Lab6.c
 *
 * Created: 7/20/2020 6:16:31 PM
 * Author : Koby
 */

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



int main(void)
{
    spi_init();

       while(1)
       {
              //assert low
              PORTF.OUTCLR = 0b00010000;

              spi_write(0x53);

              //assert high
              PORTF.OUTSET = 0b00010000;
       }
}
```

APPENDIX

Section 2 DAD logic picture:



Mode: Repeated
Buffer: 10
Trigger: Normal
Source: Digital
Inputs: 100MHz DIO 0..15
Position: 0 s
Base: 200 us/div

```
Trig'd 4096 samples at 2 MHz | 2020-07-21 13:25:01.710
```

Discovery2 SN:210321AD1852
Status: OK