Lab 6: SPI, IMU Accelerometer

OBJECTIVES

- Learn how to use SPI communication to interact with an external IMU sensor package.
- Stream and plot real-time XYZ acceleration data with USART to Atmel Studio's data visualizer.

REQUIRED MATERIALS

- uPAD and Robotics Backpack
- Robotics Backpack schematic
- NAD/DAD (NI/Diligent Analog Discovery) kit
- LSM330 Datasheet
- `LSM330.h` Header file
- XMEGA documents
  - doc8331: XMEGA Manual, section 22
  - avr151_spiAVR151: SPI Application Note

YOU WILL NOT BE ALLOWED INTO YOUR LAB SECTION WITHOUT THE REQUIRED PRE-LAB.

PRELAB REQUIREMENTS

You must adhere to the *Lab Rules and* Policies document for **every** lab. Your clock should be running at **32 MHz**. **ALL** of your code should be written in C.

INTRODUCTION

In this lab, you will setup communication with an *LSM330 Inertial Measurement Unit* (**IMU**) chip that contains a MEMS **3D digital accelerometer** and **3D digital gyroscope**. (You will **not** use the gyroscope in this lab.) To do this, you will use the SPI module of the XMEGA to configure as well as receive the sensor data from the IMU.

Once SPI communication with the IMU is properly established, you will use the USART system (like previous labs) to stream the sensor data to Atmel Studio's data visualizer via your PC's COM port to generate real-time plots of the linear acceleration data.

PART A: INITIALIZING SPI

In this part, you will setup the SPI module of the XMEGA in order to communicate with the IMU. Carefully read sections 22.2, 22.3, 22.5 and 22.7 of the doc8331 manual. Also read the LSM330 datasheet. Some important things in the datasheet are Table 2 (focus on the SPI functions of the pins), Table 6, Section 4, Table 9, **Section 6.2**, Section 7, and Section 8. Pay more attention to the accelerometer functionality as opposed to the gyroscope, since we will not use the gyroscope this semester.

For this lab, you will need to configure the SPI system in the XMEGA as well as configuring the LSM330 device. Therefore, you will need to have SPI working on the XMEGA before you can even begin configuring the IMU.

Here are some of the main things to considering when configuring the SPI module:

- Order of data transmission (MSB or LSB transmitted first)
- XMEGA should be the bus master and the IMU should be a slave
- Transfer mode – This determines when the data is latched (often referred to as "phase" and "polarity")
- Serial clock speed – The speed at which data is transferred. The serial clock is referred to as **SCK** by Atmel and **SPC** in the LSM330 datasheet.

For Part A, you will write a function called **`spi_init()`** that initializes the XMEGA's SPI module for communication with the LSM330. You will need to initialize the correct **control signals** and **SPI signals**. Refer to the Robotics backpack schematic for specific ports and pins for these signals. You will also need to properly configure the `SPIF.CTRL` register because we are using **PORTF** to send/receive data to the LSM330.

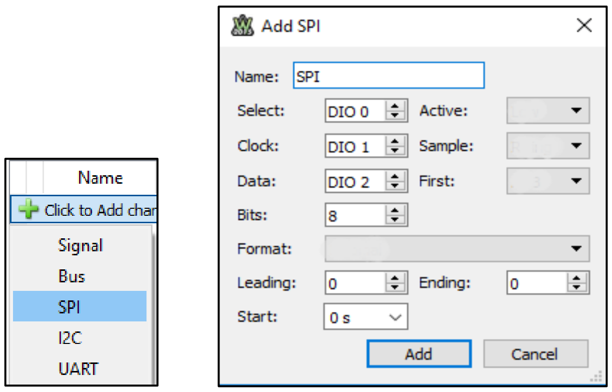
If you aren't sure how to configure the bits in the `SPIF.CTRL` register, for example, the "DORD" (Data Order) bit, then refer to the LSM330 datasheet. The SPI timing diagrams there will tell you whether the LSB or the MSB should be transmitted first. Also **make sure** you use a clock frequency that isn't too fast for the LSM330.

YOU NEED TO WRITE AND SUBMIT PSEUDOCODE FOR ALL PARTS WHERE YOU WROTE CODE, EVEN IF YOU DO NOT HAVE TO SUBMIT A SOURCE (.c) FILE.

PART B: SPI COMMUNICATION TESTING

Now you will test your SPI configuration to verify that it is working. The easiest way to do this is to use the LSA of your DAD/NAD board. Make sure you have the latest version of Waveforms, and that there is **NO backpack connected** to your uPAD. At the time of writing this document, the latest version is 3.5.4. This version includes a way to easily analyze and view SPI communication (using the LSA).

When you add an SPI "bus" using the DAD/NAD menu shown in Figure 1a, you will be prompted with the menu in Figure 1b.



Name: SPI
Select: DIO 0
Clock: DIO 1
Data: DIO 2
Bits: 8
Leading: 0
Ending: 0
Start: 0 s

Figure 1: **a)** (left figure); **b)** (right figure)

Lab 6: SPI, IMU Accelerometer

This feature only looks at three signals: Select, Clock, and Data. You will connect either the MOSI **or** the MISO signal to the DIO wire you choose for the "Data" line, depending on which signal you want to observe. For this part of the lab, an XMEGA SPI write cycle is used to verify that it works by using the SPI feature of your DAD/NAD board's LSA. This means you will need to connect the "**MOSI**" signal from **PORTF** to the DIO wire that you choose for the "Data" SPI signal in Waveforms.

To initiate an SPI transmission, you need to write data to the **`SPIF.DATA`** register. Then, wait for the "**IF**" bit to be set in the **`SPIF.STATUS`** register. The "**IF**" bit is set when a full byte of data has been completely shifted.

Write function,
`uint8_t spiWrite(uint8_t data)`
that writes a **single byte** of data and **waits** for the transmission to be complete. This function should also **return** whatever is in the **`SPIF.DATA`** register. Remember that when you write data over SPI, data gets shifted in **AND** shifted out at the same time. After the "**IF**" bit is set in the **`SPIF.STATUS`** register, there is data in the **`SPIF.DATA`** register that is ready to be read.

You should also write a function that is called
`uint8_t spiRead()`
that returns the result of `spiWrite(0xFF)`. In this case, the value 0xFF is irrelevant. With SPI, data is shifted in at the **same time** data is shifted out. To be able to **READ** a byte, we have to **SEND** a byte. This function should essentially write 0xFF via SPI and then return the result, since your `spiWrite()` function should already return the contents of the **`SPIF.DATA`** register. This will be used in the following parts of the lab.

It is highly recommended that you create source and header files to go along with your main program. For example, you could create an spi.c and spi.h pair of files that contain the `spiWrite()` and `spiRead()` functions. Then you can add more functions to them in the next part of the lab.

Write a program, **`Lab6b.c`**, that calls your **`spi_init()`** function and then repeatedly calls your **`spiWrite()`** function, sending the data **0x53**.

While your program is running, use the SPI functionality of your LSA to view the SPI data. You can use the SCK (aka SPC) signal as a trigger source.

Take a screenshot of your LSA showing a full byte of data being transmitted as well as the clock signal. Include this screenshot in your report's Appendix.

PART C: COMMUNICATING WITH THE LSM330

Now that you have the ability to send and receive data via the SPI module, you can configure the *LSM330 IMU*. This is the part where you really need to make sure you have **read and understand** the LSM330 datasheet. At this point you can go ahead and **connect** the Robotics backpack to your uPAD.

The LSM330 has a plethora of configuration registers, exactly like the XMEGA, other microprocessors, and other peripherals (like the LCDs that we discussed in class). The only difference is that you have to use SPI to write/read to/from them (instead of the data bus for an LCD). For example, one of the registers is called `CTRL_REG5_A`, and it is at address 0x20 (see Table 17 in the LSM330 datasheet). This register configures the output data rate as well as enables the 3 different axes (X, Y, Z) for the accelerometer. The "_A" at the end of the register name indicates that it modifies the accelerometer. Registers ending in "_G" would modify the gyroscope.

It is also highly recommended that you download and #include the **`LSM330.h`** header file that is on the website. It contains definitions similar to the XMEGA's include file that allows you to refer to the names of the registers in your code as opposed to just the addresses. For example, see Figure 2 below.

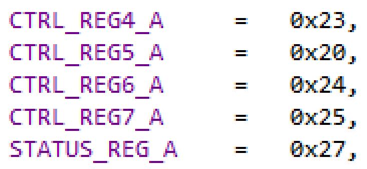

```
CTRL_REG4_A     =   0x23,
CTRL_REG5_A     =   0x20,
CTRL_REG6_A     =   0x24,
CTRL_REG7_A     =   0x25,
STATUS_REG_A    =   0x27,
```

Figure 2: **Code in LSM330.h**

Create functions that allow you to **write** to the LSM330 accelerometer as well as **read** from it. If you look at the Robotics backpack schematic, you will see that there are three digital switches on page 3. Two of these switches are controlled by the **PROTOCOL_SEL** signal which is just a pin on PORTA (see page 1). This signal determines which protocol you will use to communicate to the IMU (I2C or SPI). This semester we will use only SPI in the course, so you should set PROTOCOL_SEL to zero, i.e., PROTOCOL_SEL = 0. This will connect the SCK signal from the XMEGA to the SCL pin of the LSM330.

The remaining digital switch determines which device is connected to the MISO signal – either the accelerometer or the gyroscope. If you were using the gyroscope, you would need to change this signal accordingly every time you access the accel/gyro registers. For example, if you wanted to read from the accelerometer you would need to make SENSOR_SEL = 1. This connects the XMEGA's MISO signal on PORTF to the LSM330's MISO_A pin.

Lab 6: SPI, IMU Accelerometer

Here is the order of events that need to happen when creating a function that **writes** to the LSM330:

1. Make the slave select signal **LOW** for the accelerometer (SSA).
2. Set the SENSOR_SEL bit (1=accel, 0=gyro). This selects the accelerometer as the device with which you want to communicate.
3. Now you need to use your `spiWrite()` function to write two bytes of data to the LSM330. See the SPI timing diagram in Figure 3.
   a. According to this diagram, the first byte that you write is the address byte. It contains RW, MS, and AD[5:0]. RW is the read/write bit. For a write cycle, we want it to be a 0. The MS bit is set when you want to do multiple simultaneous reads/writes at successive addresses. We won't be using this feature for this lab. Make sure the MS bit is 0. The AD[5:0] bits are the address bits. Each register in the LSM330 is addressed by these 6 bits. For example, if you are writing to the CTRL_REG5_A register, these bits would need to be 0b100000 (0x20). If you use the LSM330.h header file you can simply use `CTRL_REG5_A` when calling your write function.
   b. The second byte that you need to write (**immediately** after the first byte) is the DATA byte (DI[7:0]). This is the data that you are writing **to** the designated register.
4. After both bytes have been written, set the slave select signal back **HIGH** to signal the end of transmission.

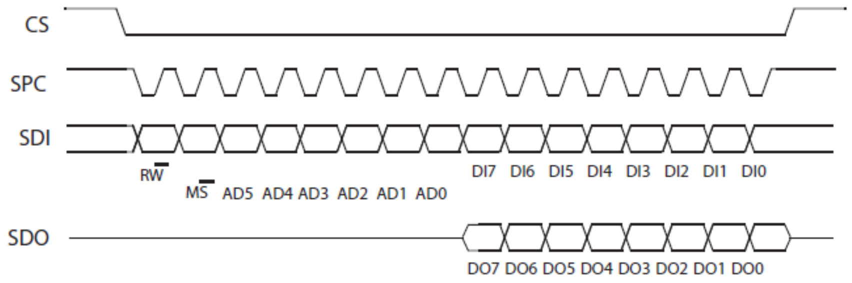

CS
SPC
SDI
RW
MS
AD5 AD4 AD3 AD2 AD1 AD0
DI7 DI6 DI5 DI4 DI3 DI2 DI1 DI0
SDO
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

Figure 3: LSM330 SPI Timing Diagram

You will need to create functions that read data **FROM** the accelerometer and gyroscope as well as write to them. The steps are mostly the same as write cycles. You still have to select which sensor you are communicating with (accel) as well as make the SSA signal **LOW**. Then, you still **write** the first byte which determines which address you are referencing, and whether you want to read data or write data. The difference is after you write the first byte, you have to use your spiRead() function to read the data that was at the designated address (register).

PART D: INITIALIZING THE LSM330

Now that you have a set of functions that allow you to configure the LSM330's registers, you need to actually configure them!

Create a function, `accel_init()`, that initializes the LSM330's accelerometer.

It is always best to reset a system before you use it.

First set the XMEGA SPI slave selects for the accelerometer and gyro to false (high). Then set the serial system for SPI (and not I2C). Then initialize the SPI system.

After setting up the SPI system, reset the LSM330. Reset the accelerometer by sending 0x01 to `CTRL_REG4_A`. Do all of this before you initialize the accelerometer.

You will initialize the LSM330 **accelerometer** as follows:

1. XMEGA: Enable a falling edge interrupt on PORTC, pin 7. This is the INT1A signal. This will trigger an interrupt signaling data is ready to be read.
2. LSM330: Use the `CTRL_REG4_A` register to route the DRDY signal to `INT1_A` and enable `INT1` with a rising edge (active high).
3. LSM330: Use the `CTRL_REG5_A` register to configure the accelerometer to have the highest possible output data rate as well as enable the X, Y, and Z axes.

Now that you have a function that initializes the accelerometer, you need to decide how to manage the data that is stored on the LSM330. There is a GPIO triggered interrupt that is triggered when the accelerometer has data ready to be read. One method would be to have a global flag variable such as `accelDataReady`. Then, when your ISR gets triggered, all you have to do is set the flag. Don't forget to declare global variables as volatile! For example:

```
volatile uint8_t globalVariable;
```

In your main `while(1)` loop, you would be constantly checking if the flag is set. If it is set, then you read the data from the specific registers in the LSM330.

The accelerometer has 3 axes (X, Y, and Z). Each axis has data contained in two registers (low byte AND high byte). For example, the high and low bytes of data for the X axis of the accelerometer is stored in the `OUT_X_L_A` and `OUT_X_H_A` registers, respectively.

When the `accelDataReady` flag is set, you should read the data from all 3 axes and store it at known locations in memory. Make sure you can distinguish between low and high bytes. For example, name 8-bit variables something like `ACCEL_X_H`, or 16-bit variables something like `ACCEL_X`. It doesn't matter how you organize the data as long as you know how to properly access it later.

PART E: USART DATA STREAM

Now that you have the data organized (and constantly updating), you need to use the USART system to stream the data to Atmel Studio via the USB serial port. This will allow you to stream the data to a graph in real time!

Initialize the XMEGA's USART system (USARTD0) just like previous labs, but use a baud rate of **1 Mbps**. Use no parity, one stop bit, and 8-bit character size.

Now to stream the data, you will be using Atmel Studio's Data Streamer Protocol. This gives you a nice way to organize all of the data you will be streaming via USART.

See Atmel's website for more information:
http://www.atmel.com/webdoc/dv/dv.Modules.DataStreamer.html

For this protocol, there is a **specific order** in which you need to transmit the data, as shown in Figure 4. Note that in this figure, the values in between the "start" and "end" bytes are just arbitrary example values.

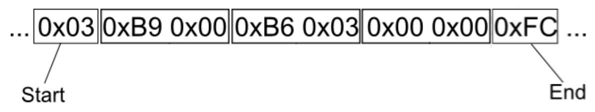


Figure 4: Data Stream Order of Bytes

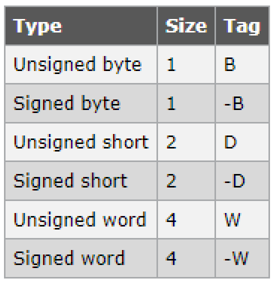
As you can see, there is a "start byte" and an "end byte". These are for synchronization of the data, similar to USART start and stop bits. Notice how the data is organized in **2-byte** blocks. According to the "Stream Format" section of the above linked documentation, "All data must be given as little-endian values, meaning that the lowest byte must be sent first."

To use the Data Streamer in Atmel Studio's Data Visualizer, you must first open it by going to Configuration > Protocols > Data Streamer. Then, by clicking the "…" button, you have to load a configuration file. The config file is just a .txt file that specifies the amount, type, and order of data that is going to be streamed. Download the **`lab6_stream_protocol.txt`** file from the class website, shown in Figure 5.

```
-D,1,1,ACCEL_X
-D,2,1,ACCEL_Y
-D,3,1,ACCEL_Z
```

Figure 5: Data Streamer config file, `lab6_stream_protocol.txt`

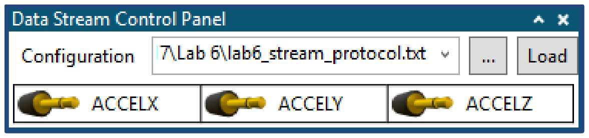
Each row of this file defines a specific stream of data. In our case, we are splitting the data up into each axis of the accelerometer. This gives us 3 streams of data. The first column defines the type of data for its particular stream. These types are shown in Figure 6. For data from the LSM330, we will need 2-bytes for each stream. For example, the accelerometer's x-axis data is stored in two bytes (from LSM330 registers `OUT_X_L_A` and `OUT_X_H_A`). We are using the "-D" tag because it corresponds to "signed short" values which are two bytes wide.



Type	Size	Tag
Unsigned byte	1	B
Signed byte	1	-B
Unsigned short	2	D
Signed short	2	-D
Unsigned word	4	W
Signed word	4	-W

Figure 6: Stream Data Types

You can think of the 2nd and 3rd columns as a way to visually organize the streams within the Data Visualizer. Figure 7 shows what the resulting data streamer should look like using this file.



Configuration 7\Lab 6\lab6_stream_protocol.txt

ACCELX ACCELY ACCELZ

Figure 7: Data Streamer control panel

Notice how the 2nd and 3rd columns of the config file determine the column and row of the stream in the Data Streamer control panel.

The last column of the config file gives a name to the stream. You **MUST** transmit the stored data in the **SAME ORDER** as it appears in this configuration file. "ACCELX" data should be transmitted **FIRST**, and "ACCELZ" data should be transmitted **LAST**, etc.

Lab 6: SPI, IMU Accelerometer

Write a function that transmits the stream of sensor data via the USB Serial Port (USARTD0). You can use functions you have already created such as `OUT_CHAR()`, `OUT_STRING()`, etc. This function should do the following:

1. Transmit the start byte. The example in Figure 4 used 0x03. You can use other values, but the stop byte **HAS** to be the opposite of the start byte.
2. Transmit the sensor data in the same order as the streams are listed in the configuration file. Remember to send the **LOWER BYTES FIRST**.
3. Transmit the stop byte. Remember this must be the opposite as the start byte. If the start byte is 0x03, the stop byte must be 0xFC, as seen in Figure 4.

The overall process is as follows:
1. Read and store sensor data from the LSM330 when it is available.
2. Transmit the large stream of data from the XMEGA to your computer's Serial COM port using USARTD0. Only transmit data when there is NEW data available. Do NOT waste time sending old data.
3. The large stream of data is then filtered by the Data Streamer into 3 separate streams, one for each axis of the sensor.

PART F: REAL TIME DATA PLOTTING

Now that you have 3 streams of separated sensor data, you can do some cool things with it! You will write a program, **`Lab6f.c`**, in this section.

For this part, you will feed the sensor data streams to the graph tool in Atmel Studio's Data Visualizer. This part is very simple. To start, open up **ONE** "Graph" window by going to Configuration > Visualization > Graph. This graph will be used to display the accelerometer data. Then, open the Data Streamer and the Serial Port control panels.

The source of all of the data is the Serial Port. Drag the **PLUG** from the Serial Port window and connect it to the **JACK** in the Data Stream Control Panel. This is making the Serial Port (USARTD0) stream the source **TO** the Data Streamer. Now you have 3 streams of separate data (see Figure 8) that you can use. See the `Visualizer_for_SPI.gif` GIF on our website.

Start by connecting the "ACCELX" plug to the "**new plot**" jack of the first graph. Do the same thing for "ACCELY" and "ACCELZ". Now you should have the three accelerometer streams of data being plotted in real time on your first graph!

Save this program as **`Lab6f.c`**, and take a screenshot of your graph. Include it in the Appendix of your lab report.

PART G: VISUALIZING DATA USING RGB

In this part you will modify the RGB LED using PWM to visualize the accelerometer data instead of the Data Visualizer.

Use your code from Part F, but remove all of the serial functionality. Start with a program that continuously updates the ACCEL data variables that you created.

For this part it is assumed that you already know how to configure PWM. Add the following to your program (and call it **`Lab6g.c`**):

The RGB LED should be mostly the color that is experiencing the highest force. When the uPAD is standing flat on a surface, the Z axis is perpendicular to the surface and therefore gravity is affecting it more than the X or Y axes. You must map the colors as such: Red:X, Green:Y, Blue:Z. So when the uPAD is standing straight up on a normal table, the BLUE LED should be illuminated the most. You can use the previous part of your lab to familiarize yourself with which axes are which.

Remember that you MUST use PWM for this. When you move the board around or hold it diagonally, the RGB LED should be a blend of multiple intensities of each color LED. Note: Treat all negative accelerometer data values as positive.

PRE-LAB QUESTIONS

1. What is the highest speed of communication the IMU can handle?
2. In which order should you transmit data to the LSM330? LSB first or MSB first?
3. How are the accelerometer and gyroscope enabled?
4. When using SPI, why do we have to write data in order to be able to read data?
5. Why is it a good idea to modify global flag variables inside ISRs instead of doing everything inside of them?

IN-LAB REQUIREMENTS

1. Demo Part F – Data being plotted from the USART data stream. If you can't get the graphs to work, try outputting values to a terminal window for partial credit.
2. Demo Part G – RGB LED changing while moving uPAD.