Project 6: Interfacing the XMega32e5 Xplained with an Accelerometer and Gyroscope

Jacob Hollenbeck, Boise State University 04/29/2016

Contents

1 Overview

This document describes the interfacing of the MPU6050 Accelerometer with the ATXMega32e5 Xplained evaluation board. This project was sponsored by Dr. Loo's Lab. The contents of this document describe the hardware setup and software setup. Please note that Atmel ASF was used in the development of the firmware. This was done so that the group that uses this code will be able to easily implement the designed firmware with their code. Figure 1 shows the XMega connected to the MPU6050 on a flat surface. The LCD readout shows the accelerometer values and gyroscope values are all set to 0.

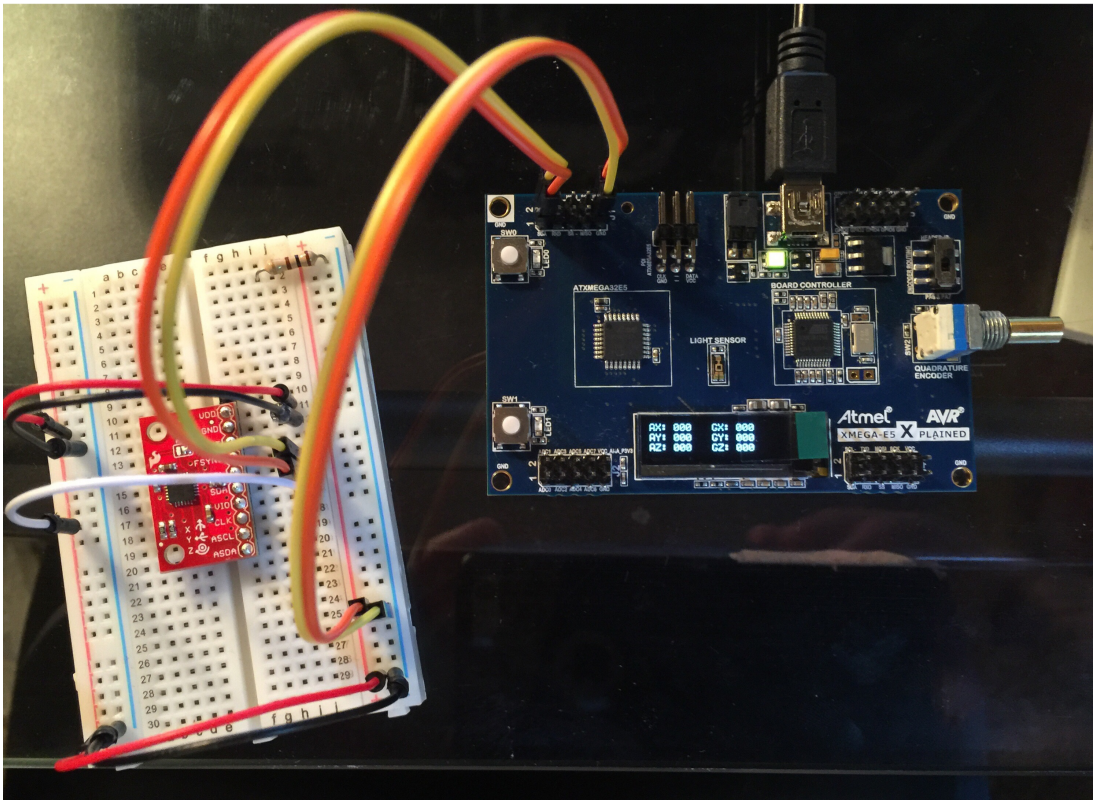

Figure 1: MPU and XMega on flat surface

Figure 2 displays the MPU6050 tilted vertically. The readout on the LCD screen shows the AY value has changed to -16 and the AZ value has changed to -20.

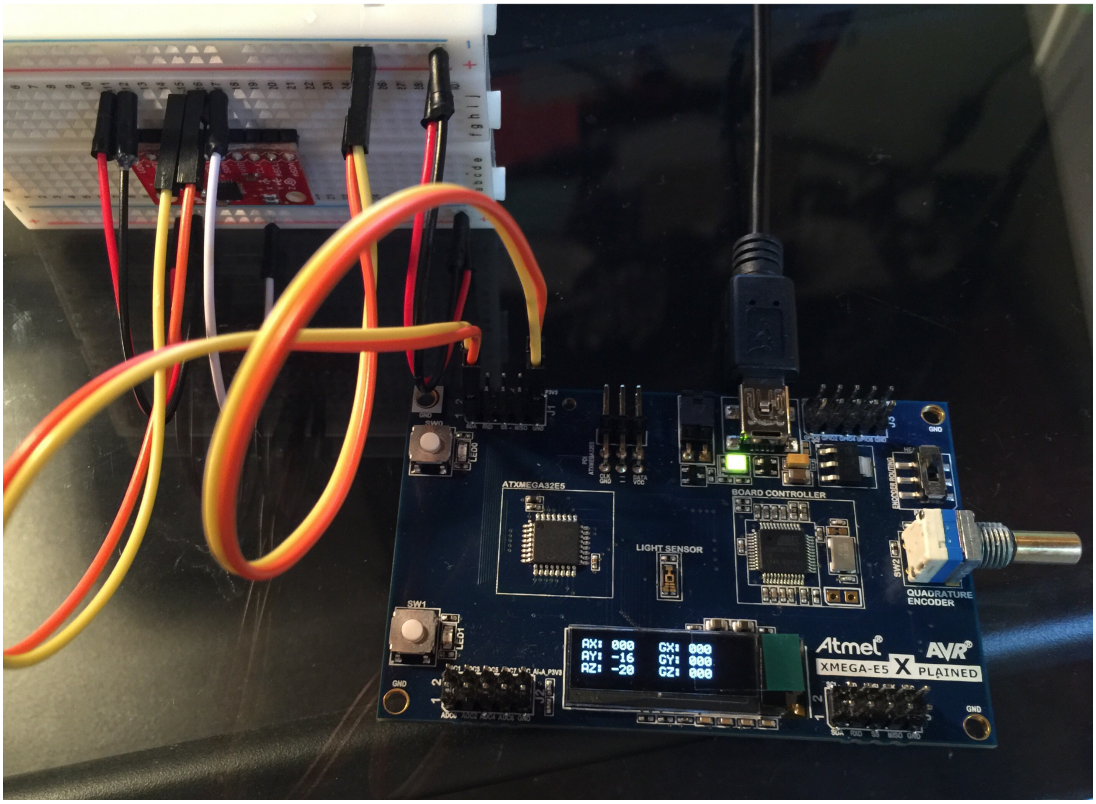

Figure 2: MPU and XMega on tilted

2 Supplies

- AtXMega32e5 Xplained Evaluation Kit
- Xplained power cable
- MPU6050 Sparkfun breakout board
- Atmel-ICE programmer with ISP cable
- Breadboard
- Jumper wires

3 Hardware

The MPU6050 can be communicated with using either SPI or I2C protocol. In this project, I2C was used. The MPU6050 sparkfun breakout board features 12 pins as shown in figure 3.

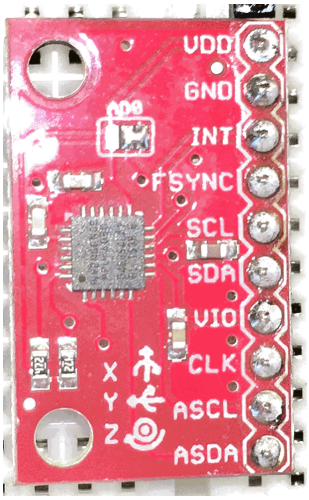

Figure 3: MPU6050

The board can be powered at 3.3 to 5.0 V. To power the board up, connect VDD to a power source and GND to the respective ground. The VIO pin needs to be connected to the same power source as VDD. The SCL and SDA are to be wired to the respective pins on the MCU. The following figure demonstrates how to configure the device.

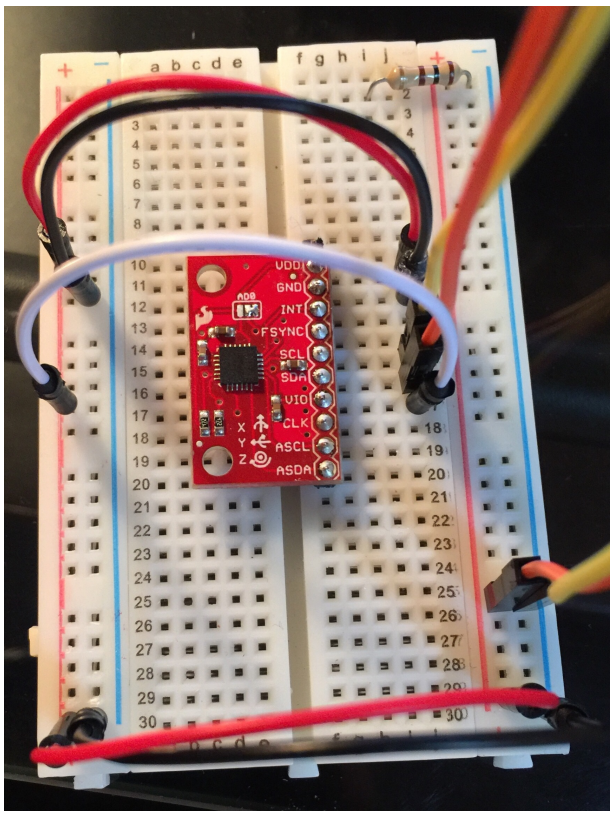

Figure 4: MPU wiring

4 Software

The MCU featured has an on-chip, LCD screen. This screen is used to display the output of the data. The data can also be displayed on a VT100 terminal by uncommenting the VT100_print code. To communicate between the MCU and the MPU6050, I2C was selected. The code was broken into 4 files. They are as listed:

- main.c
- MPU6050_Fuctions.*
- my_twi.*
- MPU6050.h

Whereas the .* indicates both a header and a C file were created. Please note that the MPU6050.h file is from a third party source. The definitions used to access the MPU were available from [2]. The program was broken up into three layers. my_twi.* was built on top of the ASF two-wire functionality. This file contains functions used by MPU6050_Functions. The MPU6050_Functions file contains accesser and initialization functions for the device. These are called from main.c

4.1 TWI

To use the ASF TWI, the two wire interface needs to be initialized. First the master device and the slave device must be declared. Next, the device options need to be set. These options

include the transfer speed between the two devices, the master address, and the TWI baud rate. The TWI baud rate is calculated in the following code:

```
    #define TWI_MASTER          TWIC        // Port locations
    #define TWI_MASTER_PORT     PORTC
    #define TWI_SLAVE           TWIC
    #define TWI_SPEED           400000      // Transfer speed
    #define TWI_MASTER_ADDR     0x50        // XMega address
    #define TWI_SLAVE_ADDR      0x68        // MPU6050 address

    // Declare speed options
      twi_options_t m_options = {
        .speed      = TWI_SPEED,
        .chip       = TWI_MASTER_ADDR,
        .speed_reg = TWI_BAUD(sysclk_get_cpu_hz(), TWI_SPEED)
    };
```

After the device options have been configured, the master and slave clocks must be initialized, the master must be initialized and enabled, and fast mode must be initialized. This is done as follows:

```
    sysclk_enable_peripheral_clock(&TWI_MASTER);
    twi_fast_mode_enable(&TWI_MASTER);
    twi_slave_fast_mode_enable(&TWI_SLAVE);
    twi_master_init(&TWI_MASTER, &m_options);
    twi_master_enable(&TWI_MASTER);
    sysclk_enable_peripheral_clock(&TWI_SLAVE);
```

Because the accelerometer is a large device, the user needs to be able to access multiple registers inside the device. To do this, readByte and writeByte functions were created. The ASF TWI function has read and write functionality that is difficult to use. To get around this, the functions were designed to be as simple as possible. Each function instantiates a twi_package structure and assigns the needed values. This is done as follows:

```

// Write a single byte to the given register
uint8_t writeByte(uint8_t reg_addr, uint8_t byte)
{
    twi_package_t p;
    p.chip = TWI_SLAVE_ADDR;        // device address
    p.addr[0] = reg_addr;           // register address
    p.addr_length = 1;              // size of reg_address
    p.buffer= &byte;                // data
    p.length = 1;                   // size of data

    // twi function
  return twi_master_write(&TWI_MASTER, &p);
}

// Read a single byte from the given register
uint8_t readByte(uint8_t reg_addr, uint8_t * byte)
{
```

```
  twi_package_t p;
  p.chip = TWI_SLAVE_ADDR;        // device address
  p.addr[0] = reg_addr;           // register address
  p.addr_length = 1;              // size of register address
  p.buffer = byte;                // where you want data to go
  p.length = 1;                   // size of 1 byte

    // twi function
  return twi_master_read(&TWI_MASTER, &p);
}
```

The user will also need to access specific bits inside of the MPU6050 as well as words within the device. These functions can be seen within the included code.

4.2 MPU6050

The MPU6050 features 75 registers and outputs the data in micrometers per second squared. The device needs to be initialized before anything can be done with it. To do this, the clock source needs to be set and the sleep state needs to be disabled.

```
void MPU6050_init(void) {
  // Read that the GYRO clock is slightly better than default clock
    set_ClockSource(0x1);

    // Set sensitivity of both accel and gyro
    set_Sensitivity(4);

    // Disable the default sleep mode
    set_SleepEnabled(0x0);

    // Default offset is 1, so need to apply an offset of 1
    set_ZAccelOffset(0x1);
}
```

The above function also sets the bit resolution and applys an offset to the accelerometer z axis. The accelerometer z axis begins at 1. We would like to normalize that to 0. The bit resolution can be set to 4 different levels of resolution for both the gyroscope and the accelerometer. A function was designed so that the user can input the numbers 1 to 4 to control the resolution whereas 1 is the most sensitive and 4 is the least. To customize it, please refer to the user manual [1]. Note that the device comes set with the device beginning in sleep mode. This needs to be disabled by writing a 0 to it. After the device has been initialized, a read can occur. To do this, the device register for each of the axis motions are simply read. These registers can be found inside of the user manual. Note that the 7 registers containing the desired data are sequential. The read that is used inside of this code reads in the 14 bytes proceeding 0x3B (the first register containing data). Each data register occupies to full 8 bit registers. The data from these buffers are then stored inside of 16 bit unsigned integers. This can be seen as follows:

```
/* Grab data from MPU beginning at 0x3B. Each takes up 16 bits so:
  0x3B AX
```

```
  0x3D AY
  0x3F AZ
  0x41
  0x43 GX
  0x45 GY
  0x47 GZ
*/
void get_6AxisData(accel_6AxisData_t * p)
{
  readBytes(MPU6050_RA_ACCEL_XOUT_H, buffer, BUFFER_SIZE);
  // Want to see negative values so data is converted to int
  // Acclerometer data
    p->ax = ((((int16_t)buffer[0]) << 8) | buffer[1])/SCALE;     // SCALE = 1000
    p->ay = ((((int16_t)buffer[2]) << 8) | buffer[3])/SCALE;
    p->az = ((((int16_t)buffer[4]) << 8) | buffer[5])/SCALE;

    // Gyroscope data
    p->gx = ((((int16_t)buffer[8]) << 8) | buffer[9])/SCALE;
    p->gy = ((((int16_t)buffer[10]) << 8) | buffer[11])/SCALE;
    p->gz = ((((int16_t)buffer[12]) << 8) | buffer[13])/SCALE;
}
```

Because the data is in micro-g's, the data is scaled by 1000. The data is cast into signed integers so that the user can see the negative values. These functions are used inside of main as follows:

```
#include <asf.h>
#include "MPU6050_Functions.h"

int main(void)
{
  sysclk_init();
  board_init();
  gfx_mono_init();

  // My functions
  twi_init();
  MPU6050_init();

  // Set sensitivity
  set_Sensitivity(4);

  // create structure
  accel_6AxisData_t data;

  // init 6 axis data to 0
  init_6AxisData(&data);

  while (true) {

    get_6AxisData(&data);
    print_6AxisData(&data);
```

```
  }
}
```

5 Compilation

Please note that ASF was used with linux operating system. This was very difficult and resulted in many headaches. After watching some videos online, I would highly recommend that the user use a virtual machine with windows. This will allow the user to use atmel studios with ASF. If the user does desire to use linux, then the following steps must be taken. First download the ASF from atmel studios. Next, copy the example directory found with the ASF. For example:

```
$ cd avr-toolchain/xdk-asf-3.30.0/xmega/applications/
$ cp -rf xmega_e5_xplained_demp my_dir
```

Next, navigate to the gcc directory located at within the atxmega32e5_xmega_e5_xplained directory. Open the config.mk file and update all CSRS paths and INC_PATHS to your current directory. Exit this file. The new file structure is as follows: the my_dir is the top level. This is where the user will include the user designed sources. The next dir, atxmega32e5_xmega_e5_xplained, contains the configuration files and peripherals that the user wishes to use. To compile, navigate to the gcc folder and use the make command. To load to the board use the following command:

```
$ avrdude -c atmelice_pdi -p x32e5 -e -U flash:w:test.elf
```

References

[1] *INVESENSE ACCELEROMETER*. URL: https://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf.

[2] Jeff Rowberg. URL: https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050.h.