```
/*
  10-5-11
  Copyright Spark Fun Electronics© 2011
  Aaron Weiss
  aaron at sparkfun dot com

      9DOF-Razor-IMU-AHRS compatible

      ATMega328@3.3V w/ external 8MHz resonator
      High fuse 0xDA
  Low fuse 0xFF
      EXT fust 0xF8

      Default Baud: 57600bps

      Revision Notes:
      Hardware v22
      Firmware:
      v18 took self test off of menu, explain how to use it in help menu
      v19 added baud rate selection, default to 57600bps, various bug fixes
      v19i fixed baud menu return bugs
      v20 using ITG3200 gyro
      v21 added auto self test upon startup (see notes)
      v22 corrected HMC output, x and y registers are different in the HMC5883

      ADXL345: Accelerometer
      HMC5883: Magnetometer
      ITG3200: Pitch, Roll, and Yaw Gyro

      Notes:

      -To get out of autorun, hit ctrl-z
      -max baud rate @8MHz is 57600bps
      -self-test startup: LED blinks 5 times then OFF = GOOD, LED ON = BAD

*/

#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "types.h"
#include "defs.h"
#include "i2c.h"

#define STATUS_LED 5 //stat LED is on PB5
```

```
#define sbi(var, mask)   ((var) |= (uint8_t)(1 << mask))
#define cbi(var, mask)   ((var) &= (uint8_t)~(1 << mask))

#define ITG3200_R 0xD1      // ADD pin is pulled low
#define ITG3200_W 0xD0

///============Initialize Prototypes=====/////////////////
void init(void);
unsigned int UART_Init(unsigned int ubrr);
uint8_t uart_getchar(void);
static int uart_putchar(char c, FILE *stream);
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
_FDEV_SETUP_WRITE);
void i2cInit(void);

///============Function Prototypes=========////////////////
void accelerometer_init(void);
void auto_raw(void);
void baud_menu(void);
void check_baud(void);
void config_menu(void);
void config_read(void);
void gyro_init(void);
void help(void);
void magnetometer(void);
void magnetometer_init(void);
void print_adxl345(void);
void print_hmc5883(void);
void print_itg3200(void);
void raw(void);
void self_test(void);
uint16_t x_accel(void);
uint16_t y_accel(void);
uint16_t z_accel(void);
uint16_t x_gyro(void);
uint16_t y_gyro(void);
uint16_t z_gyro(void);
uint16_t sqrtTest(int16_t InitialGuess, int16_t Number);

///============EEPROM Protoypes============/////////////////
void write_to_EEPROM(unsigned int Address, unsigned char Data);
unsigned char read_from_EEPROM(unsigned int Address);

///============Display Strings============/////////////////
```

```
const char wlcm_str[] PROGMEM = "\n\n\r9DOF IMU Firmware v22
\n\r========================";
const char accel[] PROGMEM = "\n\r[1]Accelerometer: ADXL345 \n\r";
const char mag[] PROGMEM = "[2]Magnetometer: HMC5883 \n\r";
const char gyro[] PROGMEM = "[3]Gyroscope: ITG-3200 \n\r";
const char raw_out[] PROGMEM = "[4]Raw Output\n\r";
const char baud_change[] PROGMEM = "[5]Change Baud Rate: ";
const char autorun[] PROGMEM = "[Ctrl+z]Toggle Autorun\n\r";
const char help_[] PROGMEM = "[?]Help\n\r";

///============Global Vars=========/////////////////
uint16_t x_mag, y_mag, z_mag;//InitialGuess,Number,sqrt_err,min_sqrt_err; //x, y,
and z magnetometer values
long baud;
//float Xo;//result;
//int32_t  iteration_count,max_iterations;

/////===========MAIN====================//////////////////
int main(void)
{
        //init();
        //self_test();
 int16_t x,y,z,i;
 uint16_t sqr,srt;
 init();
 self_test();
 UART_Init(16);
 baud = 57600;
//while(1)
 {

  //check to see if autorun is set, if it is don't print the menu
  //if(read_from_EEPROM(1) == 48) config_read();
 //else config_menu();
 }
        while(1)
        {

                //check to see if autorun is set, if it is don't print the menu
                //if(read_from_EEPROM(1) == 48) config_read();
                //else config_menu();
 x=0;
 y=0;
 z=0;
 for(i=0;i<16;i++)
 {
```

```
  //x+= i+100;
  //y+= 2*i+10;
  //z+= i*i-10*i;
   x += x_accel();
   y += y_accel();
   z += z_accel();

  //printf("x=%6d   ",x);
  //printf("y=%6d   ",y);
  //printf("z=%6d   \r",z);
  delay_ms(62);
 }
  x=x>>4;
  y=y>>4;
  z=z>>4;
  sqr= (x*x)+(y*y)+(z*z);
  //srt= sqrtTest(x,sqr);
  printf("%d,",x);
  printf("%d,",y);
  printf("%d\r\n",z);
  //printf("sqr=%6u\r   ",sqr);
  //printf("srt=%6u\r   ",srt);
        }

}

void accelerometer_init(void)
{

        //initialize
        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0xA6);    //write to ADXL
        i2cWaitForComplete();
        i2cSendByte(0x2D);    //power register
        i2cWaitForComplete();
        i2cSendByte(0x08);    //measurement mode
        i2cWaitForComplete();
        i2cSendStop();

        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0xA6);    //write to ADXL
        i2cWaitForComplete();
        i2cSendByte(0x31);    //data format
        i2cWaitForComplete();
```

```
	i2cSendByte(0x08);    //full resolution
	i2cWaitForComplete();
	i2cSendStop();

}

void print_adxl345(void)
{
	printf("x=%4d, ", x_accel());
	printf("y=%4d, ", y_accel());
	printf("z=%4d \n\r", z_accel());
	delay_ms(20);
}

void auto_raw(void)
{
	unsigned int ticks = 0;
	//while there is not a button pressed
	while(!(UCSR0A & (1 << RXC0)))
	{
		//prints the raw vaues with a '$' start and '#\n\r' end
		printf("$");
		printf("%d,", x_accel());
		printf("%d,", y_accel());
		printf("%d,", z_accel());
		/*printf("%d,", x_gyro());
		printf("%d,", y_gyro());
		printf("%d,", z_gyro());*/
		/*if (ticks++ % 20 == 0) // Only once each 20 ticks, i.e. 400ms
			magnetometer();
		printf("%d,", x_mag);
		printf("%d,", y_mag);
		printf("%d", z_mag);
		printf("#\n\r");*/
		delay_ms(20);
	}

	//if a button is pressed and that button is ctrl-z, reset autorun, display menu
	if(uart_getchar() == 0x1A)
	{
		write_to_EEPROM(1,0);
		config_menu();
	}

	auto_raw();
}
```

```
void baud_menu(void)
{
	printf("\n\rBaud Rate Select Menu\n\r");
	printf("[1] 4800\n\r");
	printf("[2] 9600\n\r");
	printf("[3] 19200\n\r");
	printf("[4] 38400\n\r");
	printf("[5] 57600\n\r");

	uint8_t choicer=0;

	while(1)
	{
		choicer = uart_getchar();
		putchar('\n');
		putchar('\r');

		if(choicer=='1') //4800
		{
			//outside of default flag: used to notify init not to run default
baud value
			write_to_EEPROM(2, 99);

			//clear other EEPROM values
			write_to_EEPROM(4, 0);
			write_to_EEPROM(5, 0);
			write_to_EEPROM(6, 0);
			write_to_EEPROM(7, 0);

			write_to_EEPROM(3, 4); //baud change flag
			printf("!change baud rate to 4800bps, reset board!");
			delay_ms(50);
			UART_Init(207);
			while(1);
		}
		if(choicer=='2') //9600
		{
			write_to_EEPROM(2, 99); //outside of default flag

			//clear other EEPROM values
			write_to_EEPROM(3, 0);
			write_to_EEPROM(5, 0);
			write_to_EEPROM(6, 0);
			write_to_EEPROM(7, 0);
```

```
            write_to_EEPROM(4, 9); //baud change flag
            printf("!change baud rate to 9600bps, reset board!");
            delay_ms(50);
            UART_Init(103);
            while(1);
    }
    if(choicer=='3') //19200
    {
            write_to_EEPROM(2, 99); //outside of default flag

            //clear other EEPROM values
            write_to_EEPROM(3, 0);
            write_to_EEPROM(4, 0);
            write_to_EEPROM(6, 0);
            write_to_EEPROM(7, 0);

            write_to_EEPROM(5, 19); //baud change flag
            printf("!change baud rate to 19200bps, reset board!");
            delay_ms(50);
            UART_Init(51);
            while(1);
    }
    if(choicer=='4') //38400
    {
            write_to_EEPROM(2, 99); //outside of default flag

            //clear other EEPROM values
            write_to_EEPROM(3, 0);
            write_to_EEPROM(4, 0);
            write_to_EEPROM(5, 0);
            write_to_EEPROM(7, 0);

            write_to_EEPROM(6, 38); //baud change flag
            printf("!change baud rate to 38400bps, reset board!");
            delay_ms(50);
            UART_Init(25);
            while(1);
    }
    if(choicer=='5') //57600
    {
            write_to_EEPROM(2, 99); //outside of default flag

            //clear other EEPROM values
            write_to_EEPROM(3, 0);
            write_to_EEPROM(4, 0);
            write_to_EEPROM(5, 0);
```

```
                write_to_EEPROM(6, 0);

                write_to_EEPROM(7, 57); //baud change flag
                printf("!change baud rate to 57600bps, reset board!");
                delay_ms(50);
                UART_Init(16);
                while(1);
            }
            if((choicer < 0X31) || (choicer > 0x35))config_menu(); //if choice is
not #s 1-5 goto conig menu
        }
        config_menu();
}

void check_baud(void)
{
        if(read_from_EEPROM(2) == 99) //check to see if the baud rate has been
changed by user
        {
            //read baud rate selection
            if(read_from_EEPROM(3) == 4)
            {
                baud = 4800;
                UART_Init(207);
            }
            if(read_from_EEPROM(4) == 9)
            {
                baud = 9600;
                UART_Init(103);
            }
            if(read_from_EEPROM(5) == 19)
            {
                baud = 19200;
                UART_Init(51);
            }
            if(read_from_EEPROM(6) == 38)
            {
                baud = 38400;
                UART_Init(25);
            }
            if(read_from_EEPROM(7) == 57)
            {
                baud = 57600;
                UART_Init(16);
            }
        }
```

```
	else
	{
		//57600bps Default UART
		UART_Init(16);
		baud = 57600;
	}
}

void config_menu(void)
{
	i2cInit();
	accelerometer_init();
	/*magnetometer_init();*/

	printf_P(wlcm_str);
	printf_P(accel);
	printf_P(mag);
	printf_P(gyro);
	printf_P(raw_out);
	printf_P(baud_change);
	printf("%ldbps\n\r", baud);
	printf_P(autorun);
	printf_P(help_);

	config_read();
}

void config_read(void)
{
	uint8_t choice=0;

	if(read_from_EEPROM(1) != 48)
	{
		while(1)
		{
			choice = uart_getchar();
			putchar('\n');
			putchar('\r');

			if(choice=='1')
			{
				while(!(UCSR0A & (1 << RXC0)))print_adxl345();
				config_menu();
			}
			if(choice=='2')
			{
```

```
                                        while(!(UCSR0A & (1 << RXC0)))
                                        {
                                                /*print_hmc5883();*/
                                                delay_ms(350);//at least 100ms interval
between measurements
                                        }
                                        config_menu();
                                }
                                if(choice=='3')
                                {
                                        /*while(!(UCSR0A & (1 << RXC0)))print_itg3200();*/
                                        config_menu();
                                }
                                if(choice=='4')
                                {
                                        while(!(UCSR0A & (1 << RXC0)))raw();
                                        config_menu();
                                }
                                if(choice=='5')
                                {
                                        baud_menu();
                                        config_menu();
                                }
                                if(choice==0x10) //if ctrl-p
                                {
                                        self_test();
                                }
                                if(choice==0x1A) //if ctrl-z
                                {
                                        write_to_EEPROM(1,48);
                                        auto_raw();
                                }
                                if(choice==0x3F) //if ?
                                {
                                        help();
                                        while(!(UCSR0A & (1 << RXC0)));
                                        config_menu();
                                }
                                if(choice==0xFF) config_read();
                        }
                }else auto_raw();

}

void gyro_init(void)
{
```

```
	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write 0xB4
	i2cWaitForComplete();
	i2cSendByte(0x3E);  // write register address
	i2cWaitForComplete();
	i2cSendByte(0x80);
	i2cWaitForComplete();
	i2cSendStop();

	delay_ms(10);

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write 0xB4
	i2cWaitForComplete();
	i2cSendByte(0x16);  // write register address
	i2cWaitForComplete();
	i2cSendByte(0x18);  // DLPF_CFG = 0, FS_SEL = 3
	i2cWaitForComplete();
	i2cSendStop();

	delay_ms(10);

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write 0xB4
	i2cWaitForComplete();
	i2cSendByte(0x3E);  // write register address
	i2cWaitForComplete();
	i2cSendByte(0x00);
	i2cWaitForComplete();
	i2cSendStop();
}

void help(void)
{
	printf("HELP MENU\n\r");
	printf("[1] send ascii 1 to get output from the accelerometer(x,y,z). Hit any
key to return to menu.\n\r");
	/*printf("[2] send ascii 2 to get output from the magnetometer(x,y,z). Hit any
key to return to menu.\n\r");
	printf("[3] send ascii 3 to get output from the gyroscope(x,y,z). Hit any key to
return to menu.\n\r");
	printf("[4] send ascii 4 to get the raw output from all of the sensors. Hit any
key to return to menu.\n\r");
```

```
	printf("*** Raw format
'$accelx,accely,accelz,gyrox,gyroy,gyroz,magx,magy,magz#\n\r");
	printf("[5] send ascii 5 to get the 5 choices for baud rates. Reset terminal and
board after change. Hit any key to return to menu.\n\r");
	printf("[ctrl-p] ctrl-p tests the accelerometer and magnetometer.\n\r");
	printf("[ctrl-z] ctrl+z at anytime will toggle between raw output and the
menu\n\r");*/
}

/*void print_hmc5883(void)
{
	magnetometer();
	printf("x=%4d, ", x_mag);
	printf("y=%4d, ", y_mag);
	printf("z=%d\n\r", z_mag);
}*/

/*void magnetometer(void)
{
	/*
		The magnetometer values must be read consecutively
		in order to move the magnetometer pointer. Therefore the x, y, and z
		outputs need to be kept in this function. To read the magnetometer
		values, call the function magnetometer(), then global vars
		x_mag, y_mag, z_mag.
	*/

	/*magnetometer_init();

	uint8_t xh, xl, yh, yl, zh, zl;

	//must read all six registers plus one to move the pointer back to 0x03

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(0x3D);    //write to HMC
	i2cWaitForComplete();
	i2cReceiveByte(TRUE);
	i2cWaitForComplete();
	xh = i2cGetReceivedByte();  //x high byte
	i2cWaitForComplete();

	i2cReceiveByte(TRUE);
	i2cWaitForComplete();
	xl = i2cGetReceivedByte();   //x low byte
	i2cWaitForComplete();
```

```
        x_mag = xl|(xh << 8);

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zh = i2cGetReceivedByte();
        i2cWaitForComplete();     //z high byte

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zl = i2cGetReceivedByte();   //z low byte
        i2cWaitForComplete();
        z_mag = zl|(zh << 8);

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        yh = i2cGetReceivedByte();  //y high byte
        i2cWaitForComplete();

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        yl = i2cGetReceivedByte();   //y low byte
        i2cWaitForComplete();
        y_mag = yl|(yh << 8);

        i2cSendByte(0x3D);        //must reach 0x09 to go back to 0x03
        i2cWaitForComplete();

        i2cSendStop();
}

void magnetometer_init(void)
{
        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0x3C);    //write to HMC
        i2cWaitForComplete();
        i2cSendByte(0x00);    //mode register
        i2cWaitForComplete();
        i2cSendByte(0x70);    //8 average, 15Hz, normal measurement
        i2cWaitForComplete();
        i2cSendStop();

        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0x3C);    //write to HMC
        i2cWaitForComplete();
```

```
          i2cSendByte(0x01);    //mode register
          i2cWaitForComplete();
          i2cSendByte(0xA0);    //gain = 5
          i2cWaitForComplete();
          i2cSendStop();

          i2cSendStart();
          i2cWaitForComplete();
          i2cSendByte(0x3C);    //write to HMC
          i2cWaitForComplete();
          i2cSendByte(0x02);    //mode register
          i2cWaitForComplete();
          i2cSendByte(0x00);    //continuous measurement mode
          i2cWaitForComplete();
          i2cSendStop();
}*/
uint16_t sqrtTest(int16_t InitialGuess, int16_t Number)
     {
        uint16_t max_iterations = 20;

        int16_t sqrt_err = 1000;
        int16_t min_sqrt_err = 1;
        int16_t Xo = InitialGuess;

        int16_t iteration_count;

        uint16_t result = 0.0F;
        if (Xo<0)
        {
         Xo = Xo*(-1);
        }
       iteration_count = 0;
       printf("Xo=%4d\r",Xo);
       while (iteration_count < max_iterations && ((sqrt_err > min_sqrt_err ||
sqrt_err < -min_sqrt_err)))
        {
           if(Number == 0) // zero check
           {
              return 0;
           }
           else
           {
              result = (uint16_t)(Xo - (Xo * Xo - Number) / (2 * Xo));
           }

           sqrt_err = (int16_t)(result - Xo);
```

```
            Xo = (int16_t)(result);
            //printf("sqrt_err=%4d",sqrt_err);
            //printf("Xo=%4d",Xo);
            //printf("result=%4u\r",result);

            iteration_count++;
          }
          return (uint16_t)result;

      }  // end of sqrtTest

void raw(void)
{
          //prints the raw vaues with a '$' start and '#\n\r' end
          printf("$");
          printf("%d,", x_accel());
          printf("%d,", y_accel());
          printf("%d,", z_accel());
          /*printf("%d,", x_gyro());
          printf("%d,", y_gyro());
          printf("%d,", z_gyro());*/
          /*magnetometer();
          printf("%d,", x_mag);
          printf("%d,", y_mag);
          printf("%d", z_mag);
          printf("#\n\r");*/
          delay_ms(350);//at least 100ms interval between mag measurements
}

void self_test(void)
{
          //MAGNETOMETER

          //uint8_t xh, xl, yh, yl, zh, zl, hmc_flag = 0;
          //x_mag = 0;
          //y_mag = 0;
          //z_mag = 0;

          /*magnetometer_init();

          //must read all six registers plus one to move the pointer back to 0x03

          i2cSendStart();
          i2cWaitForComplete();
          i2cSendByte(0x3D);    //write to HMC
          i2cWaitForComplete();
```

```
        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        xh = i2cGetReceivedByte();  //x high byte
        i2cWaitForComplete();

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        xl = i2cGetReceivedByte();   //x low byte
        i2cWaitForComplete();
        x_mag = xl|(xh << 8);

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zh = i2cGetReceivedByte();
        i2cWaitForComplete();     //z high byte

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zl = i2cGetReceivedByte();   //z low byte
        i2cWaitForComplete();
        z_mag = zl|(zh << 8);

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        yh = i2cGetReceivedByte();  //y high byte
        i2cWaitForComplete();

        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        yl = i2cGetReceivedByte();   //y low byte
        i2cWaitForComplete();
        y_mag = yl|(yh << 8);

        i2cSendByte(0x3D);       //must reach 0x09 to go back to 0x03
        i2cWaitForComplete();

        i2cSendStop();
        */
        //if it gets to this point and there are values in x,y,z_mag, we can assume part
is responding correctly
//      if((x_mag == y_mag) && (y_mag == z_mag)) hmc_flag = 0xFF;
//      else hmc_flag = 0;

        //ACCELEROMETER

        uint8_t x, dummy;
```

```
//0x32 data registers
i2cSendStart();
i2cWaitForComplete();
i2cSendByte(0xA6);    //write to ADXL
i2cWaitForComplete();
i2cSendByte(0x00);    //X0 data register
i2cWaitForComplete();

i2cSendStop();                    //repeat start
i2cSendStart();

i2cWaitForComplete();
i2cSendByte(0xA7);    //read from ADXL
i2cWaitForComplete();
i2cReceiveByte(TRUE);
i2cWaitForComplete();
x = i2cGetReceivedByte();
i2cWaitForComplete();
i2cReceiveByte(FALSE);
i2cWaitForComplete();
dummy = i2cGetReceivedByte();
i2cWaitForComplete();
i2cSendStop();

////////////////////////////////
// Gyro//////////////////////////
////////////////////////////////

char data;

/*cbi(TWCR, TWEN);            // Disable TWI
sbi(TWCR, TWEN);    // Enable TWI

i2cSendStart();
i2cWaitForComplete();
i2cSendByte(ITG3200_W);   // write 0xD2
i2cWaitForComplete();
i2cSendByte(0x00);  // who am i
i2cWaitForComplete();

i2cSendStart();

i2cSendByte(ITG3200_R);    // write 0xD3
i2cWaitForComplete();
i2cReceiveByte(FALSE);
```

```
        i2cWaitForComplete();

        data = i2cGetReceivedByte();        // Get MSB result
        i2cWaitForComplete();
        i2cSendStop();

        cbi(TWCR, TWEN);    // Disable TWI
        sbi(TWCR, TWEN);    // Enable TWI
        */
        //int gyro_flag = 0;
        //int mag_flag = 0;
        int accel_flag = 0;

        if(data == 0x69)
        {
                //printf("ITG: GOOD\n\r");
//              gyro_flag = 1;
        }//else printf("ITG: BAD\n\r");

        //if(hmc_flag == 0)
        {
                //printf("HMC: GOOD\n\r");
//              mag_flag = 1;
        }//else printf("HMC: BAD\n\r");

        if(x == 0xE5)
        {
                //printf("ADXL: GOOD\n\r");
                accel_flag = 1;
        }//else printf("ADXL: BAD\n\r");

//      if(gyro_flag ==1 && mag_flag == 1 && accel_flag == 1)
        {
                sbi(PORTB, 5);
                delay_ms(1000);
                cbi(PORTB, 5);
                delay_ms(1000);
                sbi(PORTB, 5);
                delay_ms(1000);
                cbi(PORTB, 5);
                delay_ms(1000);
                sbi(PORTB, 5);
                delay_ms(1000);
                cbi(PORTB, 5);
                delay_ms(1000);
                sbi(PORTB, 5);
```

```
                delay_ms(1000);
                cbi(PORTB, 5);
                delay_ms(1000);
                sbi(PORTB, 5);
                delay_ms(1000);
                cbi(PORTB, 5);

//              gyro_flag = 0;
//              mag_flag = 0;
                accel_flag = 0;

        }//else sbi(PORTB, 5);

        //while(!(UCSR0A & (1 << RXC0)));
        //config_menu();
}

/*void print_itg3200(void)
{

        printf("x= %4d, ", x_gyro());
        printf("y= %4d, ", y_gyro());
        printf("z= %4d\n\r", z_gyro());

        delay_ms(20);
}*/

/*uint16_t x_gyro(void)
{
        uint16_t  xh, xl, data;

        cbi(TWCR, TWEN);    // Disable TWI
        sbi(TWCR, TWEN);    // Enable TWI

        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(ITG3200_W);   // write
        i2cWaitForComplete();
        i2cSendByte(0x1D);     // x high address
        i2cWaitForComplete();
        i2cSendStart();

        i2cSendByte(ITG3200_R);    // read
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();
```

```
	xh = i2cGetReceivedByte();  // Get MSB result
	i2cWaitForComplete();
	i2cSendStop();

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write
	i2cWaitForComplete();
	i2cSendByte(0x1E);      // x low address
	i2cWaitForComplete();
	i2cSendStart();

	i2cSendByte(ITG3200_R);    // read
	i2cWaitForComplete();
	i2cReceiveByte(FALSE);
	i2cWaitForComplete();

	xl = i2cGetReceivedByte();   // Get LSB result
	i2cWaitForComplete();
	i2cSendStop();

	data = xl|(xh << 8);

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI

	return data;
}

uint16_t y_gyro(void)
{
	uint16_t  xh, xl, data;

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write
	i2cWaitForComplete();
	i2cSendByte(0x1F);  // y high address
	i2cWaitForComplete();
```

```
	i2cSendStart();

	i2cSendByte(ITG3200_R);     // read
	i2cWaitForComplete();
	i2cReceiveByte(FALSE);
	i2cWaitForComplete();

	xh = i2cGetReceivedByte();  // Get MSB result
	i2cWaitForComplete();
	i2cSendStop();

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI

	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(ITG3200_W);   // write
	i2cWaitForComplete();
	i2cSendByte(0x20);  // y low address
	i2cWaitForComplete();
	i2cSendStart();

	i2cSendByte(ITG3200_R);    // read
	i2cWaitForComplete();
	i2cReceiveByte(FALSE);
	i2cWaitForComplete();

	xl = i2cGetReceivedByte();   // Get LSB result
	i2cWaitForComplete();
	i2cSendStop();

	data = xl|(xh << 8);

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI

	return data;
}

uint16_t z_gyro(void)
{
	uint16_t  xh, xl, data;

	cbi(TWCR, TWEN);    // Disable TWI
	sbi(TWCR, TWEN);    // Enable TWI
```

```
        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(ITG3200_W);   // write
        i2cWaitForComplete();
        i2cSendByte(0x21);  // z high address
        i2cWaitForComplete();
        i2cSendStart();

        i2cSendByte(ITG3200_R);    // read
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();

        xh = i2cGetReceivedByte();  // Get MSB result
        i2cWaitForComplete();
        i2cSendStop();

        cbi(TWCR, TWEN);    // Disable TWI
        sbi(TWCR, TWEN);    // Enable TWI

        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(ITG3200_W);   // write
        i2cWaitForComplete();
        i2cSendByte(0x22);  // z low address
        i2cWaitForComplete();
        i2cSendStart();

        i2cSendByte(ITG3200_R);    // read
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();

        xl = i2cGetReceivedByte();   // Get LSB result
        i2cWaitForComplete();
        i2cSendStop();

        data = xl|(xh << 8);

        cbi(TWCR, TWEN);    // Disable TWI
        sbi(TWCR, TWEN);    // Enable TWI

        return data;
}*/

uint16_t x_accel(void)
```

```
{
	//0xA6 for a write
	//0xA7 for a read

	uint8_t dummy, xh, xl;
	uint16_t xo;

	//0x32 data registers
	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(0xA6);    //write to ADXL
	i2cWaitForComplete();
	i2cSendByte(0x32);    //X0 data register
	i2cWaitForComplete();

	i2cSendStop();                  //repeat start
	i2cSendStart();

	i2cWaitForComplete();
	i2cSendByte(0xA7);    //read from ADXL
	i2cWaitForComplete();
	i2cReceiveByte(TRUE);
	i2cWaitForComplete();
	xl = i2cGetReceivedByte();   //x low byte
	i2cWaitForComplete();
	i2cReceiveByte(FALSE);
	i2cWaitForComplete();
	dummy = i2cGetReceivedByte();     //must do a multiple byte read?
	i2cWaitForComplete();
	i2cSendStop();

	//0x33 data registers
	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(0xA6);    //write to ADXL
	i2cWaitForComplete();
	i2cSendByte(0x33);    //X1 data register
	i2cWaitForComplete();

	i2cSendStop();                  //repeat start
	i2cSendStart();

	i2cWaitForComplete();
	i2cSendByte(0xA7);    //read from ADXL
	i2cWaitForComplete();
	i2cReceiveByte(TRUE);
```

```
        i2cWaitForComplete();
        xh = i2cGetReceivedByte();  //x high byte
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();
        dummy = i2cGetReceivedByte();     //must do a multiple byte read?
        i2cWaitForComplete();
        i2cSendStop();
        xo = xl|(xh << 8);
        return xo;
}

uint16_t y_accel(void)
{
        //0xA6 for a write
        //0xA7 for a read

        uint8_t dummy, yh, yl;
        uint16_t yo;

        //0x34 data registers
        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0xA6);    //write to ADXL
        i2cWaitForComplete();
        i2cSendByte(0x34);    //Y0 data register
        i2cWaitForComplete();

        i2cSendStop();                  //repeat start
        i2cSendStart();

        i2cWaitForComplete();
        i2cSendByte(0xA7);    //read from ADXL
        i2cWaitForComplete();
        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        yl = i2cGetReceivedByte();   //x low byte
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();
        dummy = i2cGetReceivedByte();     //must do a multiple byte read?
        i2cWaitForComplete();
        i2cSendStop();

        //0x35 data registers
        i2cSendStart();
```

```
	i2cWaitForComplete();
	i2cSendByte(0xA6);    //write to ADXL
	i2cWaitForComplete();
	i2cSendByte(0x35);    //Y1 data register
	i2cWaitForComplete();

	i2cSendStop();                  //repeat start
	i2cSendStart();

	i2cWaitForComplete();
	i2cSendByte(0xA7);    //read from ADXL
	i2cWaitForComplete();
	i2cReceiveByte(TRUE);
	i2cWaitForComplete();
	yh = i2cGetReceivedByte();  //y high byte
	i2cWaitForComplete();
	i2cReceiveByte(FALSE);
	i2cWaitForComplete();
	dummy = i2cGetReceivedByte();     //must do a multiple byte read?
	i2cWaitForComplete();
	i2cSendStop();
	yo = yl|(yh << 8);
	return yo;
}

uint16_t z_accel(void)
{
	//0xA6 for a write
	//0xA7 for a read

	uint8_t dummy, zh, zl;
	uint16_t zo;

	//0x36 data registers
	i2cSendStart();
	i2cWaitForComplete();
	i2cSendByte(0xA6);    //write to ADXL
	i2cWaitForComplete();
	i2cSendByte(0x36);    //Z0 data register
	i2cWaitForComplete();

	i2cSendStop();                  //repeat start
	i2cSendStart();

	i2cWaitForComplete();
	i2cSendByte(0xA7);    //read from ADXL
```

```
        i2cWaitForComplete();
        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zl = i2cGetReceivedByte();   //z low byte
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();
        dummy = i2cGetReceivedByte();     //must do a multiple byte read?
        i2cWaitForComplete();
        i2cSendStop();

        //0x37 data registers
        i2cSendStart();
        i2cWaitForComplete();
        i2cSendByte(0xA6);   //write to ADXL
        i2cWaitForComplete();
        i2cSendByte(0x37);   //Z1 data register
        i2cWaitForComplete();

        i2cSendStop();                  //repeat start
        i2cSendStart();

        i2cWaitForComplete();
        i2cSendByte(0xA7);   //read from ADXL
        i2cWaitForComplete();
        i2cReceiveByte(TRUE);
        i2cWaitForComplete();
        zh = i2cGetReceivedByte();  //z high byte
        i2cWaitForComplete();
        i2cReceiveByte(FALSE);
        i2cWaitForComplete();
        dummy = i2cGetReceivedByte();     //must do a multiple byte read?
        i2cWaitForComplete();
        i2cSendStop();
        zo = zl|(zh << 8);
        return zo;
}

/*********************
 ****Initialize****
 *********************/

void init (void)
{
   //1 = output, 0 = input
        DDRB = 0b01100000; //PORTB4, B5 output for stat LED
```

```
   DDRC = 0b00010000; //PORTC4 (SDA), PORTC5 (SCL), PORTC all others are
inputs
   DDRD = 0b00000010; //PORTD (TX output on PD1)
        PORTC = 0b00110000; //pullups on the I2C bus

        cbi(PORTB, 5);

        i2cInit();
        accelerometer_init();
        /*magnetometer_init();
        gyro_init();*/

        check_baud();
}

unsigned int UART_Init(unsigned int ubrr)
{
        int ubrr_new;
        // set baud rate
        ubrr_new = ubrr;
        UBRR0H = ubrr_new>>8;
        UBRR0L = ubrr_new;

        // Enable receiver and transmitter
        UCSR0A = (1<<U2X0); //double the speed
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);

        // Set frame format: 8 bit, no parity, 1 stop bit,
        UCSR0C = (1<<UCSZ00)|(1<<UCSZ01);

        stdout = &mystdout; //Required for printf init
        return(ubrr);
}

uint8_t uart_getchar(void)
{
   while( !(UCSR0A & (1<<RXC0)) );
   return(UDR0);
}

static int uart_putchar(char c, FILE *stream)
{
   if (c == '\n') uart_putchar('\r', stream);

   loop_until_bit_is_set(UCSR0A, UDRE0);
   UDR0 = c;
```

```
   return 0;
}

/*********************
 **EEPROM Map and Functions***
 *********************/
//Address Map for EEAR
//0x01-0x07 used

//Description: Writes an unsigned char(Data)  to the EEPROM at the given Address
//Pre: Unsigned Int Address contains address to be written to
//        Unsigned Char Data contains data to be written
//Post: EEPROM "Address" contains the "Data"
//Usage: write_to_EEPROM(0, 'A');
void write_to_EEPROM(unsigned int Address, unsigned char Data)
{
   //Interrupts are globally disabled!

        while(EECR & (1<<EEPE)); //Wait for last Write to complete
        //May need to wait for Flash to complete also!
        EEAR = Address;                 //Assign the Address Register with
"Address"
        EEDR=Data;                      //Put "Data" in the Data Register
        EECR |= (1<<EEMPE);         //Write to Master Write Enable
        EECR |= (1<<EEPE);          //Start Write by setting EE Write Enable
}

//Description: Reads the EEPROM data at "Address" and returns the character
//Pre: Unsigned Int Address is the address to be read
//Post: Character at "Address" is returned
//Usage:        unsigned char Data;
//              Data=read_from_EEPROM(0);
unsigned char read_from_EEPROM(unsigned int Address)
{
        //Interrupts are globally disabled!

        while(EECR & (1<<EEPE));  //Wait for last Write to complete
        EEAR = Address;                         //Assign the Address Register
with "Address"
        EECR |= (1<<EERE);              //Start Read by writing to EER
        return EEDR;                    //EEPROM Data is returned
}
```