NAME: Candace Caimol

NSHE ID: 5001848054

TITLE: Accelerometer to Wi-Fi Module

GOAL:

* Have Atmega328P microcontroller obtain readings from MPU6050 Accelerometer via I2C interface
* Connect to ESP8266 Wi-Fi module and upload data onto cloud. Module will be connected to PC via FTDI 232R and will be programmed using ESPlorer application.
* Data will be recorded as points on a line graph. The data sets will be recorded and separated between x-coordinates, y-coordinates, and z-coordinates.

DELIVERABLES:

* Have the program capable of basic device start-up (waking the MPU6050 from sleep)
* Read and record accelerometer measurements as actual numerical
* Upload these values onto cloud service via Wi-Fi (assuming working Wi-Fi module). The chosen cloud service is Thingspeak.com and the values are to be displayed as points on a line graph

LITERATURE SURVEY:

Based on the Project presentation materials and a quick literature survey - explain the need, contribution, and sustainability of your project.

This project as a standalone device doesn’t seem to serve much of a purpose other than reading and recording its orientation in relation to the Earth’s axis. On the other hand, this device could be integrated into bigger/more complex systems and its functionality could work hand-in-hand with some of the other mechanisms available in these systems. The fact that the overall cost of implementing this device is relatively cheap could make it a go to design for some of those costly projects. Also, it’s important to note that I2C is the only type of interfacing possible when it comes to using the MPU6050 sensor. This is known to be a more tedious and time-consuming process to set up compared to SPI. On the other hand, once implemented, it has been known to run more efficiently, especially since this type of interface only requires two wires, which serve as busses to send a clock signal (SCL) and data signal (SDA).  As for the ESP8266, it’s significantly more difficult to use than the Bluetooth or USART. On the other hand, once the firmware has been successfully flashed on, the module will automatically connect to the network that it was set up with.

COMPONENTS:

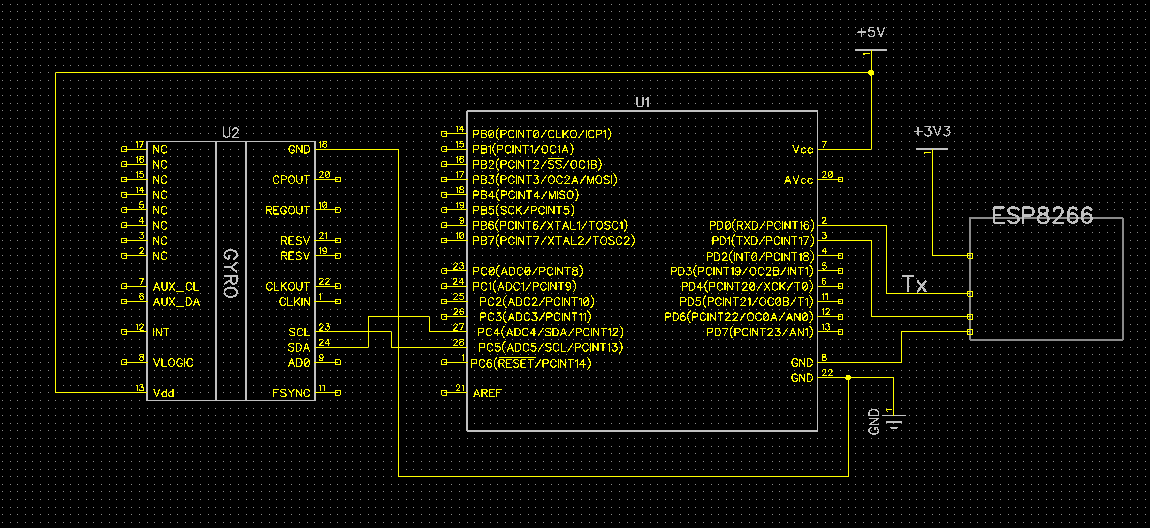
Explain the main characteristics, interface, and limitation of the components used

The components used in this project are the Atmega328P microcontroller, MPU6050 Accelerometer + Gyroscope, the FTDI 232R, and the ESP8266 Wi-Fi module. The Atmega328P microcontroller is the central component of this project. It will interface with the MPU6050 via I2C, where the microcontroller is the master and the accelerometer + gyroscope is the slave. That is, the Atmega328P will operate in the Master Receiver and Master Transmitter modes. By using this interface, the slave (MPU6050) will be read from/written to using addressing. This process will involve both the address of the device and the addresses of the internal registers to be written to/read from. The read/write process of the I2C interface is as follows (according to https://timothymcpherson.wordpress.com/2015/09/07/interfacing-over-i2c-with-the-atmega328p/):

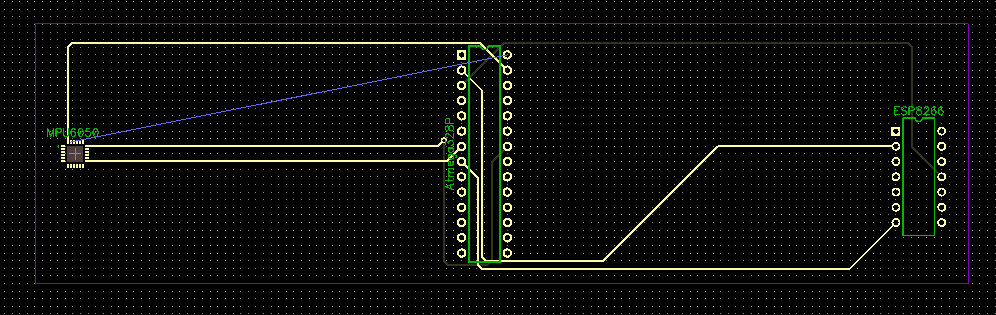
1. Send a “start” signal to inform the MPU6050 (slave) to listen to a command
2. Send the slave address plus a read/write bit
3. Send the address to read at
4. Send a stop signal before sending another start signal (repeated start)
5. Send address plus read byte
6. Read the data byte and acknowledge receipt to slave
7. Read another data byte. Slave will automatically increment address. Read won’t be acknowledged (meaning last byte to read).
8. Stop signal sent

One of the advantages of using the MPU6050 is that it is both compact and inexpensive. For an affordable price (around $5), it is capable of tracking a device’s absolute orientation with high precision. The MPU6050 contains a Digital Motion Processor (DMP) which allows 6-axis sensor fusion calculations for be performed at a fixed rate of 200Hz. When it comes to its physical connections to the Atmega328P, its only requirements are a connections of 5V and ground and the SFA and SCL pins (according to https://github.com/kriswiner/MPU-6050/wiki/Affordable-9-DoF-Sensor-Fusion).  Meanwhile, the ESP8266 Wi-Fi module will interface with the microcontroller via the TX and RX ports. This firmware needs to initially be flashed onto the Wi-Fi module using a flasher, and during this step, the Wi-Fi module needs to constantly be connected to the PC via FTDI. The ESPlorer application will be used to program the Wi-Fi module in order to have the Wi-Fi module connect to a network and have it upload values onto the cloud. Once the Wi-Fi module has been connected to a network, the only connections needed are TX, RX, VCC, and GND. It’s important to note that the ESP8266 cannot receive a voltage of more than 3.3V and it needs a current of at least 200mA in order to properly operate. The advantage of using both of these is that it is easier for first-time users to use.

SCHEMATICS: (exception - include image)



INITIAL PCB:



IMPLEMENTATION:

Based on the Project presentation materials and a quick literature survey - explain implementation details of the project. Make bullet points

Note: only part of the project has been worked on so far, so this may not be a complete/accurate representation

* Setting up I2C interfacing between the Atmega328P microcontroller and the MPU6050. Here, data is read in when SCL is high. The SCL period can be set by changing the configuration of the TWBR register (equation for precise period can be found in provided I2C pdf).
* Within the I2C interface, include the proper steps for reading/writing (described above in “COMPONENTS” section).
* Main purposes for I2C interfacing: waking up device (MPU6050), reading accelerometer and gyroscope values
* Place and connect the Wi-Fi module between the Atmega328P and the FTDI 232R. Configure the ESP8266 Wi-Fi module in order to upload the recorded data onto a free cloud service (Thingspeak.com). AT firmware will be used and the ESPlorer application will be used to program it. Implement the design such that the actual numerical values are stored (not as a graph)

CODE: (with comments)

Note: The following includes two different sets of code. The first set is the finalized code, where majority of it is based on Dr. Venki’s helper code (this is the code I got to work). The second set is the code I attempted to write completely from scratch (this code is unfinished and does not work).

FINALIZED CODE (ALL OF IT WORKS):

MPU6050

/\*

 \* Final\_Project.c

 \*

 \* Created: 4/23/2016 5:12:22 PM

 \* Author : user

 \*/

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <util/twi.h>

#include "I2C.h"

#include <math.h>

#include <string.h>

#define MPU60501 0xD0 // (0x68 << 1) I2C slave address

unsigned char ret; // return value

char outs[80];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void usart\_init (void)

{

//synchronous usart, transmit 8-bit data

UCSR0C = ((1<<UCSZ01)|(1<<UCSZ00)|(1<<UMSEL00));

//9600 Baud Rate from 8MHz clock

UBRR0H = 0x00;

UBRR0L = 33;

UCSR0B = (1<<TXEN0); //enable transmitter

}

void USART\_tx\_string (char \*data)

{

while((\*data!='\0')){

while(!(UCSR0A&(1<<UDRE0))); //wait until transmit register is empty

UDR0 = \*data;

data++;

}

}

void MPU6050\_writereg(uint8\_t accel, uint8\_t reg, uint8\_t val)

{

i2c\_start(accel+I2C\_WRITE);

i2c\_write(reg); // go to register e.g. 106 user control

i2c\_write(val); // set value e.g. to 0100 0000 FIFO enable

i2c\_stop(); // set stop condition = release bus

}

uint16\_t MPU6050\_readreg(uint8\_t accel, uint8\_t reg)//read unsigned 16 bits

{

i2c\_start\_wait(accel+I2C\_WRITE); // set device address and write mode

i2c\_write(reg); // ACCEL\_OUT

i2c\_rep\_start(accel+I2C\_READ); // set device address and read mode

int raw = i2c\_readAck(); // read one intermediate byte

raw = (raw<<8) | i2c\_readNak(); // read last byte

i2c\_stop();

return raw;

}

int16\_t MPU6050\_signed\_readreg(uint8\_t accel, uint8\_t reg)//read signed 16 bits

{

i2c\_start\_wait(accel+I2C\_WRITE); // set device address and write mode

i2c\_write(reg); // ACCEL\_OUT

i2c\_rep\_start(accel+I2C\_READ); // set device address and read mode

char raw1 = i2c\_readAck(); // read one intermediate byte

int16\_t raw2 = (raw1<<8) | i2c\_readNak(); // read last byte

i2c\_stop();

return raw2;

}

void Init\_MPU6050(uint8\_t accel)

{

ret = i2c\_start(accel+I2C\_WRITE); // set device address and write mode

if ( ret )

{

snprintf(outs,sizeof(outs),"failed to issue start condition \n\r");

USART\_tx\_string(outs);

i2c\_stop();

}

else

{

/\* issuing start condition ok, device accessible \*/

MPU6050\_writereg(accel, 0x6B, 0x00); // reg 107 set value to 0000 0000 and wake up sensor

MPU6050\_writereg(accel, 0x19, 0x07); // reg 25 sample rate divider set value to 0000 1000 for 1000 Hz

MPU6050\_writereg(accel, 0x1C, 0x18); // reg 28 acceleration configuration set value to 0001 1000 for 16g

MPU6050\_writereg(accel, 0x23, 0xF8); // reg 35 FIFO enable set value to 1111 1000 for all sensors not slave

MPU6050\_writereg(accel, 0x37, 0x10); // reg 55 interrupt configuration set value to 0001 0000 for logic level high and read clear

MPU6050\_writereg(accel, 0x38, 0x01); // reg 56 interrupt enable set value to 0000 0001 data ready creates interrupt

MPU6050\_writereg(accel, 0x6A, 0x40); // reg 106 user control set value to 0100 0000 FIFO enable

}

i2c\_stop();

}

int main(){

int16\_t xi1 = 0;

int16\_t yi1 = 0;

int16\_t zi1 = 0;

float xa1, ya1, za1;

DDRD = 0xF0;

DDRC = 0x00;

//declare average calibrated accelerometer values

//initialize calibarition values

//declare accelerometer value strings

usart\_init();//initialize usart

i2c\_init(); // init I2C interface

\_delay\_ms(200); // Wait for 200 ms.

Init\_MPU6050(MPU60501); // sensor init

\_delay\_ms(200); // Wait for 200 ms.

for(int i = 0; i<10; i++)//get values for initial calibration

{

// read raw X acceleration from fifo

xi1 += MPU6050\_signed\_readreg(MPU60501,0x3B);

// read raw Y acceleration from fifo

yi1 += MPU6050\_signed\_readreg(MPU60501,0x3D);

// read raw Z acceleration from fifo

zi1 += MPU6050\_signed\_readreg(MPU60501,0x3F);

}

//average values for calibration

xi1 = xi1/10;

yi1 = yi1/10;

zi1 = zi1/10;

snprintf(outs,sizeof(outs),"AT+CIPMUX=1\r\n"); //allow multiple connections w/ CIPMUX

USART\_tx\_string(outs);

\_delay\_ms(5000); //delay for 5 seconds

//Start infinite loop

while(1){

//grab 3 values, average, subtract calibration value, and divide by MSB

// read raw X acceleration from fifo

xa1 = MPU6050\_signed\_readreg(MPU60501,0x3B)+MPU6050\_signed\_readreg(MPU60501,0x3B)+MPU6050\_signed\_readreg(MPU60501,0x3B);

xa1 = ((xa1/3)-xi1)/2048.00;

// read raw Y acceleration from fifo

ya1 = MPU6050\_signed\_readreg(MPU60501,0x3D)+MPU6050\_signed\_readreg(MPU60501,0x3D)+MPU6050\_signed\_readreg(MPU60501,0x3D);

ya1 = ((ya1/3)-yi1)/2048.00;

// read raw Z acceleration from fifo

za1 = MPU6050\_signed\_readreg(MPU60501,0x3F)+MPU6050\_signed\_readreg(MPU60501,0x3F)+MPU6050\_signed\_readreg(MPU60501,0x3F);

za1 = ((za1/3)-zi1)/2048.00;

int x\_int; //integer value of accel x-coordinate

int x\_dec; //decimal value of accel x-coordinate

int y\_int; //integer value of accel y-coordinate

int y\_dec; //decimal value of accel x-coordinate

int z\_int; //integer value of accel z-coordinate

int z\_dec; //decimal value of accel x-coordinate

x\_int = (int)xa1; //get integer part of x-coordinate

xa1 = xa1 - x\_int; //subtract integer part from float value

x\_dec = abs((int)(xa1 \* 100)); //get decimal part of x-coordinate

y\_int = (int)ya1; //get integer part of y-coordinate

ya1 = ya1 - y\_int; //subtract integer part from float value

y\_dec = abs((int)(ya1 \* 100)); //get decimal part of y-coordinate

z\_int = (int)za1; //get integer part of z-coordinate

za1 = za1 - z\_int; //subtract integer part from float value

z\_dec = abs((int)(za1 \* 100)); //get decimal part of z-coordinate

snprintf(outs,sizeof(outs),"AT+CIPSTART=4,\"TCP\",\"184.106.153.149\",80\r\n"); //pass a channel 4

USART\_tx\_string(outs);

\_delay\_ms(4000); //delay for 4 seconds

snprintf(outs,sizeof(outs),"AT+CIPSEND=4,80\r\n"); //size of string to be sent (80 characters)

USART\_tx\_string(outs);

\_delay\_ms(4000); //delay for 4 seconds

snprintf(outs,sizeof(outs),"GET /update?key=3GQ0DYTCL5BC4BNT&field1=%i.%i&field2=%i.%i&field3=%i.%i\r\n", x\_int,x\_dec,y\_int,y\_dec,z\_int,z\_dec); //plot coordinates onto cloud graphs

USART\_tx\_string(outs);

USART\_tx\_string(outs);

\_delay\_ms(4000); //delay for 4 seconds

//convert doubles to printable strings

//print out the values

/\*snprintf(outs,sizeof(outs),"Accelerometer Coordinates: %i.%i, ", x\_int,x\_dec);

USART\_tx\_string(outs); //send string to be transmitted

snprintf(outs,sizeof(outs),"%i.%i, ", y\_int,y\_dec);

USART\_tx\_string(outs); //send string to be transmitted

snprintf(outs,sizeof(outs),"%i.%i\n\r", z\_int, z\_dec);

USART\_tx\_string(outs); //send string to be transmitted

\_delay\_ms(500); //update every half second\*/

}

return 0;

}

I2C.h

#ifndef \_I2C\_H

#define \_I2C\_H   1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Title:    C include file for the I2C master interface

\*           (i2cmaster.S or twimaster.c)

\* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury

\* File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $

\* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3

\* Target:   any AVR device

\* Usage:    see Doxygen manual

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef DOXYGEN

/\*\*

 @defgroup pfleury\_ic2master I2C Master library

 @code #include <i2cmaster.h> @endcode

 @brief I2C (TWI) Master Software Library

 Basic routines for communicating with I2C slave devices. This single master

 implementation is limited to one bus master on the I2C bus.

 This I2c library is implemented as a compact assembler software implementation of the I2C protocol

 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with built-in TWI hardware (twimaster.c).

 Since the API for these two implementations is exactly the same, an application can be linked either against the

 software I2C implementation or the hardware I2C implementation.

 Use 4.7k pull-up resistor on the SDA and SCL pin.

 Adapt the SCL and SDA port and pin definitions and eventually the delay routine in the module

 i2cmaster.S to your target when using the software I2C implementation !

 Adjust the  CPU clock frequence F\_CPU in twimaster.c or in the Makfile when using the TWI hardware implementaion.

 @note

    The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected and adapted

    to GNU assembler and AVR-GCC C call interface.

    Replaced the incorrect quarter period delays found in AVR300 with

    half period delays.

 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury

 @par API Usage Example

  The following code shows typical usage of this library, see example test\_i2cmaster.c

 @code

 #include <i2cmaster.h>

 #define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet

 int main(void)

 {

     unsigned char ret;

     i2c\_init();                             // initialize I2C library

     // write 0x75 to EEPROM address 5 (Byte Write)

     i2c\_start\_wait(Dev24C02+I2C\_WRITE);     // set device address and write mode

     i2c\_write(0x05);                        // write address = 5

     i2c\_write(0x75);                        // write value 0x75 to EEPROM

     i2c\_stop();                             // set stop conditon = release bus

     // read previously written value back from EEPROM address 5

     i2c\_start\_wait(Dev24C02+I2C\_WRITE);     // set device address and write mode

     i2c\_write(0x05);                        // write address = 5

     i2c\_rep\_start(Dev24C02+I2C\_READ);       // set device address and read mode

     ret = i2c\_readNak();                    // read one byte from EEPROM

     i2c\_stop();

     for(;;);

 }

 @endcode

\*/

#endif /\* DOXYGEN \*/

/\*\*@{\*/

#if (\_\_GNUC\_\_ \* 100 + \_\_GNUC\_MINOR\_\_) < 304

#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler !"

#endif

#include <avr/io.h>

/\*\* defines the data direction (reading from I2C device) in i2c\_start(),i2c\_rep\_start() \*/

#define I2C\_READ    1

/\*\* defines the data direction (writing to I2C device) in i2c\_start(),i2c\_rep\_start() \*/

#define I2C\_WRITE   0

/\*\*

 @brief initialize the I2C master interace. Need to be called only once

 @param  void

 @return none

 \*/

extern void i2c\_init(void);

/\*\*

 @brief Terminates the data transfer and releases the I2C bus

 @param void

 @return none

 \*/

extern void i2c\_stop(void);

/\*\*

 @brief Issues a start condition and sends address and transfer direction

 @param    addr address and transfer direction of I2C device

 @retval   0   device accessible

 @retval   1   failed to access device

 \*/

extern unsigned char i2c\_start(unsigned char addr);

/\*\*

 @brief Issues a repeated start condition and sends address and transfer direction

 @param   addr address and transfer direction of I2C device

 @retval  0 device accessible

 @retval  1 failed to access device

 \*/

extern unsigned char i2c\_rep\_start(unsigned char addr);

/\*\*

 @brief Issues a start condition and sends address and transfer direction

 If device is busy, use ack polling to wait until device ready

 @param    addr address and transfer direction of I2C device

 @return   none

 \*/

extern void i2c\_start\_wait(unsigned char addr);

/\*\*

 @brief Send one byte to I2C device

 @param    data  byte to be transfered

 @retval   0 write successful

 @retval   1 write failed

 \*/

extern unsigned char i2c\_write(unsigned char data);

/\*\*

 @brief    read one byte from the I2C device, request more data from device

 @return   byte read from I2C device

 \*/

extern unsigned char i2c\_readAck(void);

/\*\*

 @brief    read one byte from the I2C device, read is followed by a stop condition

 @return   byte read from I2C device

 \*/

extern unsigned char i2c\_readNak(void);

/\*\*

 @brief    read one byte from the I2C device

 Implemented as a macro, which calls either i2c\_readAck or i2c\_readNak

 @param    ack 1 send ack, request more data from device<br>

               0 send nak, read is followed by a stop condition

 @return   byte read from I2C device

 \*/

extern unsigned char i2c\_read(unsigned char ack);

#define i2c\_read(ack)  (ack) ? i2c\_readAck() : i2c\_readNak();

/\*\*@}\*/

#endif

I2C.c

/\*

 \* modified version of I2C master library

 \* added a timeout variable for non blocking i2c

 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Title:    I2C master library using hardware TWI interface

\* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury

\* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $

\* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3

\* Target:   any AVR device with hardware TWI

\* Usage:    API compatible with I2C Software Library i2cmaster.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <inttypes.h>

#include <compat/twi.h>

#include "I2C.h"

/\* define CPU frequency in Mhz here if not defined in Makefile \*/

#ifndef *F\_CPU*

#define *F\_CPU* 8000000UL

#endif

/\* I2C clock in Hz \*/

#define SCL\_CLOCK  100000L

/\* I2C timer max delay \*/

#define I2C\_TIMER\_DELAY 0xFF

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Initialization of the I2C bus interface. Need to be called only once

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void i2c\_init(void)

{

  /\* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 \*/

  TWSR = 0;                         /\* no prescaler \*/

  TWBR = ((*F\_CPU*/SCL\_CLOCK)-16)/2;  /\* must be > 10 for stable operation \*/

}/\* i2c\_init \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Issues a start condition and sends address and transfer direction.

  return 0 = device accessible, 1= failed to access device

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char i2c\_start(unsigned char address)

{

*uint32\_t*  i2c\_timer = 0;

*uint8\_t*   twst;

// send START condition

TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

// wait until transmission completed

i2c\_timer = I2C\_TIMER\_DELAY;

while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

if(i2c\_timer == 0)

return 1;

// check value of TWI Status Register. Mask prescaler bits.

twst = *TW\_STATUS* & 0xF8;

if ( (twst != *TW\_START*) && (twst != *TW\_REP\_START*)) return 1;

// send device address

TWDR = address;

TWCR = (1<<TWINT) | (1<<TWEN);

// wail until transmission completed and ACK/NACK has been received

i2c\_timer = I2C\_TIMER\_DELAY;

while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

if(i2c\_timer == 0)

return 1;

// check value of TWI Status Register. Mask prescaler bits.

twst = *TW\_STATUS* & 0xF8;

if ( (twst != *TW\_MT\_SLA\_ACK*) && (twst != *TW\_MR\_SLA\_ACK*) ) return 1;

return 0;

}/\* i2c\_start \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Issues a start condition and sends address and transfer direction.

 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void i2c\_start\_wait(unsigned char address)

{

*uint32\_t*  i2c\_timer = 0;

*uint8\_t*   twst;

    while ( 1 )

    {

    // send START condition

    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // wait until transmission completed

    i2c\_timer = I2C\_TIMER\_DELAY;

    while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

    // check value of TWI Status Register. Mask prescaler bits.

    twst = *TW\_STATUS* & 0xF8;

    if ( (twst != *TW\_START*) && (twst != *TW\_REP\_START*)) continue;

    // send device address

    TWDR = address;

    TWCR = (1<<TWINT) | (1<<TWEN);

    // wail until transmission completed

    i2c\_timer = I2C\_TIMER\_DELAY;

    while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

    // check value of TWI Status Register. Mask prescaler bits.

    twst = *TW\_STATUS* & 0xF8;

    if ( (twst == *TW\_MT\_SLA\_NACK* )||(twst ==*TW\_MR\_DATA\_NACK*) )

    {

        /\* device busy, send stop condition to terminate write operation \*/

        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

        // wait until stop condition is executed and bus released

        i2c\_timer = I2C\_TIMER\_DELAY;

        while((TWCR & (1<<TWSTO)) && i2c\_timer--);

        continue;

    }

    //if( twst != TW\_MT\_SLA\_ACK) return 1;

    break;

     }

}/\* i2c\_start\_wait \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible

          1 failed to access device

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char i2c\_rep\_start(unsigned char address)

{

    return i2c\_start( address );

}/\* i2c\_rep\_start \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Terminates the data transfer and releases the I2C bus

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void i2c\_stop(void)

{

*uint32\_t*  i2c\_timer = 0;

    /\* send stop condition \*/

TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

// wait until stop condition is executed and bus released

i2c\_timer = I2C\_TIMER\_DELAY;

while((TWCR & (1<<TWSTO)) && i2c\_timer--);

}/\* i2c\_stop \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Send one byte to I2C device

  Input:    byte to be transfered

  Return:   0 write successful

            1 write failed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char i2c\_write( unsigned char data )

{

*uint32\_t*  i2c\_timer = 0;

*uint8\_t*   twst;

// send data to the previously addressed device

TWDR = data;

TWCR = (1<<TWINT) | (1<<TWEN);

// wait until transmission completed

i2c\_timer = I2C\_TIMER\_DELAY;

while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

if(i2c\_timer == 0)

return 1;

// check value of TWI Status Register. Mask prescaler bits

twst = *TW\_STATUS* & 0xF8;

if( twst != *TW\_MT\_DATA\_ACK*) return 1;

return 0;

}/\* i2c\_write \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Read one byte from the I2C device, request more data from device

 Return:  byte read from I2C device

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char i2c\_readAck(void)

{

*uint32\_t*  i2c\_timer = 0;

TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);

i2c\_timer = I2C\_TIMER\_DELAY;

while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

if(i2c\_timer == 0)

return 0;

    return TWDR;

}/\* i2c\_readAck \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char i2c\_readNak(void)

{

*uint32\_t*  i2c\_timer = 0;

TWCR = (1<<TWINT) | (1<<TWEN);

i2c\_timer = I2C\_TIMER\_DELAY;

while(!(TWCR & (1<<TWINT)) && i2c\_timer--);

if(i2c\_timer == 0)

return 0;

    return TWDR;

}/\* i2c\_readNak \*/

MY ORIGINAL CODE (this attempt did not work):

/\*

 \* Final\_Project.c

 \*

 \* Created: 4/23/2016 5:12:22 PM

 \* Author : user

 \*/

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

//TWI bit mask definitions (TW\_MT\_ = master transmitter; TW\_MR = master receiver)

//Listed in http://www.nongnu.org/avr-libc/user-manual/group\_\_util\_\_twi.html

#define TW\_START 0x08

#define TW\_REP\_START 0x10

#define TW\_MT\_SLA\_ACK 0x18

#define TW\_MT\_SLA\_NACK 0x20

#define TW\_MT\_DATA\_ACK 0x28

#define TW\_MT\_DATA\_NACK 0x30

#define TW\_MT\_ARB\_LOST 0x38

#define TW\_MR\_ARB\_LOST 0x38

#define TW\_MR\_SLA\_ACK 0x40

#define TW\_MR\_SLA\_NACK 0x48

#define TW\_MR\_DATA\_ACK 0x50

#define TW\_MR\_DATA\_NACK 0x58

//MPU 6050  addresses; listed in https://www.olimex.com/Products/Modules/Sensors/MOD-MPU6050/resources/RM-MPU-60xxA\_rev\_4.pdf//

#define MPU6050\_ADDR\_WRITE 0b1101000//MPU6050 slave address (0xD0) (write end bit); AD0 pulled low

#define MPU6050\_ADDR\_READ 0b1101001//MPU6050 slave address (0xD1) (read end bit); AD0 pulled high

#define SMPLRT\_DIV 0x19

#define CONFIG 0x1A

#define GYRO\_CONFIG 0x1B

#define ACCEL\_CONFIG 0x1C

#define MOT\_THR 0x1F

#define FIFO\_EN 0x23

#define I2C\_MST\_CTRL 0x24

#define I2C\_SLV0\_ADDR 0x25

#define I2C\_SLV0\_REG 0x26

#define I2C\_SLV0\_CTRL 0x27//goes until SL4; add to list if needed

#define I2C\_MST\_STATUS 0x36

#define INT\_PIN\_CFG 0x37

#define INT\_ENABLE 0x38

#define INT\_STATUS 0x3A

#define ACCEL\_XOUT\_H 0x3B

#define ACCEL\_XOUT\_L 0x3C

#define ACCEL\_YOUT\_H 0x3D

#define ACCEL\_YOUT\_L 0x3E

#define ACCEL\_ZOUT\_H 0x3F

#define ACCEL\_ZOUT\_L 0x40

#define TEMP\_OUT\_H0x41

#define TEMP\_OUT\_L 0x42

#define GYRO\_XOUT\_H 0x43

#define GYRO\_XOUT\_L 0x44

#define GYRO\_YOUT\_H 0x45

#define GYRO\_YOUT\_L 0x46

#define GYRO\_ZOUT\_H 0x47

#define GYRO\_ZOUT\_L 0x48

#define EXT\_SENS\_DATA\_00 0x49

#define EXT\_SENS\_DATA\_01 0x4A

#define EXT\_SENS\_DATA\_02 0x4B

#define EXT\_SENS\_DATA\_03 0x4C

#define EXT\_SENS\_DATA\_04 0x4D

#define EXT\_SENS\_DATA\_05 0x4E

#define EXT\_SENS\_DATA\_06 0x4F

#define EXT\_SENS\_DATA\_07 0x50

#define EXT\_SENS\_DATA\_08 0x51

#define EXT\_SENS\_DATA\_09 0x52

#define EXT\_SENS\_DATA\_10 0x53

#define EXT\_SENS\_DATA\_11 0x54

#define EXT\_SENS\_DATA\_12 0x55

#define EXT\_SENS\_DATA\_13 0x56

#define EXT\_SENS\_DATA\_14 0x57

#define EXT\_SENS\_DATA\_15 0x58

#define EXT\_SENS\_DATA\_16 0x59

#define EXT\_SENS\_DATA\_17 0x5A

#define EXT\_SENS\_DATA\_18 0x5B

#define EXT\_SENS\_DATA\_19 0x5C

#define EXT\_SENS\_DATA\_20 0x5D

#define EXT\_SENS\_DATA\_21 0x5E

#define EXT\_SENS\_DATA\_22 0x5F

#define EXT\_SENS\_DATA\_23 0x60

#define I2C\_SLV0\_DO 0x63

#define I2C\_SLV1\_DO 0x64

#define I2C\_SLV2\_DO 0x65

#define I2C\_SLV3\_DO 0x66

#define I2C\_MST\_DELAY\_CTRL 0x67

#define SIGNAL\_PATH\_RESET 0x68

#define MOT\_DETECT\_CTRL 0x69

#define USER\_CTRL 0x6A

#define POWER\_MGMT\_1 0x6B

#define POWER\_MGMT\_2  0x6C

#define FIFO\_COUNTH 0x72

#define FIFO\_COUNTL 0x73

#define FIFO\_R\_W 0x74

#define WHO\_AM\_I 0x75

void I2C\_init()

{

TWSR = 0x00;//prescaler value set to 1

TWBR = 0x47;//SCL frequency is 50K for XTAL = 8MHz

TWCR |= (1 <<TWEN);//TWI enabled

}

//pattern every time something is done with TWI:

//Set TWCR, wait for TWINT, check TWSR

unsigned char I2C\_start(void)

{

TWCR |= (1 << TWINT) | (1 << TWSTA) | (1 << TWEN) | (1 << TWEA);

//TWINT set by hardware when TWI has finished current job

//TWINT flag must be cleared by software by writing 1 to it

//Application writes TWSTA to 1 when it desires to become a Master

//TWEN = 1, TWI takes control over I/O pins connected to SCL and SDA

//enables send "acknowledge" after data packet is received

while (!(TWCR & (1 << TWINT)));//loop while TWINT flag isn't set

if ((TWSR & 0xF8) != TW\_START)//wait for TWINT bit to be set again, and return from program if expected result doesn't show up

return 1;

return 0;

}

void I2C\_stop()

{

TWCR |= (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);

//TWINT set by hardware when TWI has finished current job

//TWINT flag must be cleared by software by writing 1 to it

//Setting TWSTO to 1 in Master mode will generate stop condition; cleared automatically

//TWEN enables TWI operation and activates TWI interface

//TWEN = 1, TWI takes control over I/O pins connected to SCL and SDA

//TWEN = 0, TWI switched off and TWI transmissions terminated

}

//used with MPU6050 address

//MPU6050 acknowledges it's addressed and ready to receive data

unsigned char I2C\_write\_slave(unsigned char addr)

{

TWDR = addr;//next data byte to be transmitted

TWCR |= (1 << TWINT) | (1 << TWEN);

//TWINT set by hardware when TWI has finished current job

//TWEN enables TWI operation and activates TWI interface

//TWEN = 1, TWI takes control over I/O pins connected to SCL and SDA

while (!(TWCR & (1 << TWINT)));  //loop while TWINT flag isn't set

if ((TWSR & 0xF8) != TW\_MT\_SLA\_ACK)//MPU acknowledges it's addressed as slave

return 1;

return 0;

}

//Real data written to accelerometer/gyroscope to wake it up

//First value sent: address of register to write (internal to MPU6050)

//Second value sent: actual value is written

unsigned char I2C\_write\_data(unsigned char data)

{

TWDR = data;//next data byte to be transmitted

TWCR |= (1 << TWINT) | (1 << TWEN);

//TWINT set by hardware when TWI has finished current job

//TWEN enables TWI operation and activates TWI interface

//TWEN = 1, TWI takes control over I/O pins connected to SCL and SDA

while (!(TWCR & (1 << TWINT)));  //loop while TWINT flag isn't set

if ((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)//slave acknowledges data receipt

return 1;

return 0;

}

unsigned char I2C\_read\_data(unsigned char LastByte, unsigned char DevAddr, unsigned char RegAddr)

{

I2C\_start();

I2C\_write\_slave(DevAddr);//address of MPU6050 (indicate if read or write)

I2C\_write\_data(RegAddr);//address of register to be read from

//checks if last byte because MPU6050 data values are 16 bits

if (LastByte == 0)//if want to read more than one byte

TWCR |= (1 << TWINT) | (1 << TWEN) | (1 << TWEA);

//TWINT set by hardware when TWI has finished current job

//TWINT flag must be cleared by software by writing 1 to it

//TWEN = 1, TWI takes control over I/O pins connected to SCL and SDA

//TWEA = 1, ACK pulse generated on TWI bus

else                   //if want to read only one byte

TWCR |= (1 << TWINT) | (1 << TWEN);

while (!(TWCR & (1 << TWINT)));//loop while TWINT flag isn't set

I2C\_stop();

*\_delay\_ms*(1);

return TWDR;//return read data byte in data register

}

void MPU6050\_init(void)

{

//I2C write: exit sleep mode

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(POWER\_MGMT\_1);//send address of POWER\_MGMT\_1 to wake up MPU6050

I2C\_write\_data(0x00);//set value of POWER\_MGMT\_1

I2C\_stop();

*\_delay\_ms*(1);//allow time for stop to send

//I2C write: LPF, bandwidth = 184 (accel) and 188 (gyro)

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(CONFIG);

I2C\_write\_data(0x01);

I2C\_stop();

*\_delay\_ms*(1);

//I2C write: gyro ADC scale: 1000 deg/s

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(GYRO\_CONFIG);

I2C\_write\_data(1 << 4);

I2C\_stop();

*\_delay\_ms*(1);

//I2C write: accelerometer ADC scale: 2 g

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(ACCEL\_CONFIG);

I2C\_write\_data(0x00);

I2C\_stop();

*\_delay\_ms*(1);

//I2C write: enable data ready interrupt

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(INT\_ENABLE);

I2C\_write\_data(0x00);

I2C\_stop();

*\_delay\_ms*(1);

//I2C write: don't reset signal path

I2C\_start();

I2C\_write\_slave(MPU6050\_ADDR\_WRITE);

I2C\_write\_data(SIGNAL\_PATH\_RESET);

I2C\_write\_data(0x00);

I2C\_stop();

*\_delay\_ms*(1);

}

unsigned char Read\_Gyro\_X\_H()

//reads higher byte of gyro x coordinate

{

unsigned char gyro\_x\_h;//higher byte of gyro x coordinate

gyro\_x\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, GYRO\_XOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return gyro\_x\_h;

}

unsigned char Read\_Gyro\_X\_L()

//reads lower byte of gyro y coordinate

{

unsigned char gyro\_x\_l;//lower byte of gyro x coordinate

gyro\_x\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, GYRO\_XOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return gyro\_x\_l;

}

unsigned char Read\_Gyro\_Y\_H()

{

unsigned char gyro\_y\_h;//higher byte of gyro y coordinate

gyro\_y\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, GYRO\_YOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return gyro\_y\_h;

}

unsigned char Read\_Gyro\_Y\_L()

{

unsigned char gyro\_y\_l;//lower byte of gyro y coordinate

gyro\_y\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, GYRO\_YOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return gyro\_y\_l;

}

unsigned char Read\_Gyro\_Z\_H()

{

unsigned char gyro\_z\_h;//higher byte of gyro z coordinate

gyro\_z\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, GYRO\_ZOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return gyro\_z\_h;

}

unsigned char Read\_Gyro\_Z\_L()

{

unsigned char gyro\_z\_l;//lower byte of gyro z coordinate

gyro\_z\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, GYRO\_ZOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return gyro\_z\_l;

}

unsigned char Read\_Accel\_X\_H()

{

unsigned char accel\_x\_h;//higher byte of accel x coordinate

accel\_x\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, ACCEL\_XOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return accel\_x\_h;

}

unsigned char Read\_Accel\_X\_L()

{

unsigned char accel\_x\_l;//lower byte of accel x coordinate

accel\_x\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, ACCEL\_XOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return accel\_x\_l;

}

unsigned char Read\_Accel\_Y\_H()

{

unsigned char accel\_y\_h;//higher byte of accel y coordinate

accel\_y\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, ACCEL\_YOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return accel\_y\_h;

}

unsigned char Read\_Accel\_Y\_L()

{

unsigned char accel\_y\_l;//lower byte of accel y coordinate

accel\_y\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, ACCEL\_YOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return accel\_y\_l;

}

unsigned char Read\_Accel\_Z\_H()

{

unsigned char accel\_z\_h;//higher byte of accel z coordinate

accel\_z\_h = I2C\_read\_data(0, MPU6050\_ADDR\_READ, ACCEL\_ZOUT\_H);

//0 indicates that this is not the last byte to be read

//address of the device and address of the register to be read from

return accel\_z\_h;

}

unsigned char Read\_Accel\_Z\_L()

{

unsigned char accel\_z\_l;//lower byte of accel z coordinate

accel\_z\_l = I2C\_read\_data(1, MPU6050\_ADDR\_READ, ACCEL\_ZOUT\_L);

//1 indicates that this is the last byte to be read

//address of the device and address of the register to be read from

return accel\_z\_l;

}

int main(void)

{

    while (1)

    {

    }

}

REFERENCE:

<https://timothymcpherson.wordpress.com/2015/09/07/interfacing-over-i2c-with-the-atmega328p/>

<https://github.com/kriswiner/MPU-6050/wiki/Affordable-9-DoF-Sensor-Fusion>

<https://docs.google.com/viewer?a=v&pid=sites&srcid=dW5sdi5lZHV8dW5sdmNwZTMwMXxneDozN2E5YTU1NjcxNmNjMjYz>

<http://www.robotshop.com/en/6-dof-gyro-accelerometer-imu-mpu6050.html>

<https://docs.google.com/viewer?a=v&pid=sites&srcid=dW5sdi5lZHV8dW5sdmNwZTMwMXxneDo0NjExZTg4YTgwMjAzNjZm>

<http://playground.arduino.cc/Main/MPU-6050>

<http://www.instructables.com/id/ESP8266-Wi-fi-module-explain-and-connection/?ALLSTEPS>

<http://www.i2cdevlib.com/forums/topic/4-understanding-raw-values-of-accelerometer-and-gyrometer/>

<http://rancidbacon.com/files/kiwicon8/ESP8266_WiFi_Module_Quick_Start_Guide_v_1.0.4.pdf>

<https://mcuoneclipse.com/2014/12/14/tutorial-iot-datalogger-with-esp8266-wifi-module-and-frdm-kl25z/>