Shabrya Lott

CPE 301 – SPRING 2016

FINAL REPORT

TITLE: BLE Hand Visualization

GOAL:

- To learn how to capture data from IMU sensor (6 DOF- MPU 6050) to estimate orientation and movement of the human hand.
- To learn how to filter and display collected data in a visual manner
- To learn how to make device wireless by using a Bluetooth module (HC-06).

DELIVERABLES:

- AVR C code
- Schematics
- PCB layout
- PowerPoint Presentation
- Any other documentation
- Visual presentation of captured data
- YouTube link

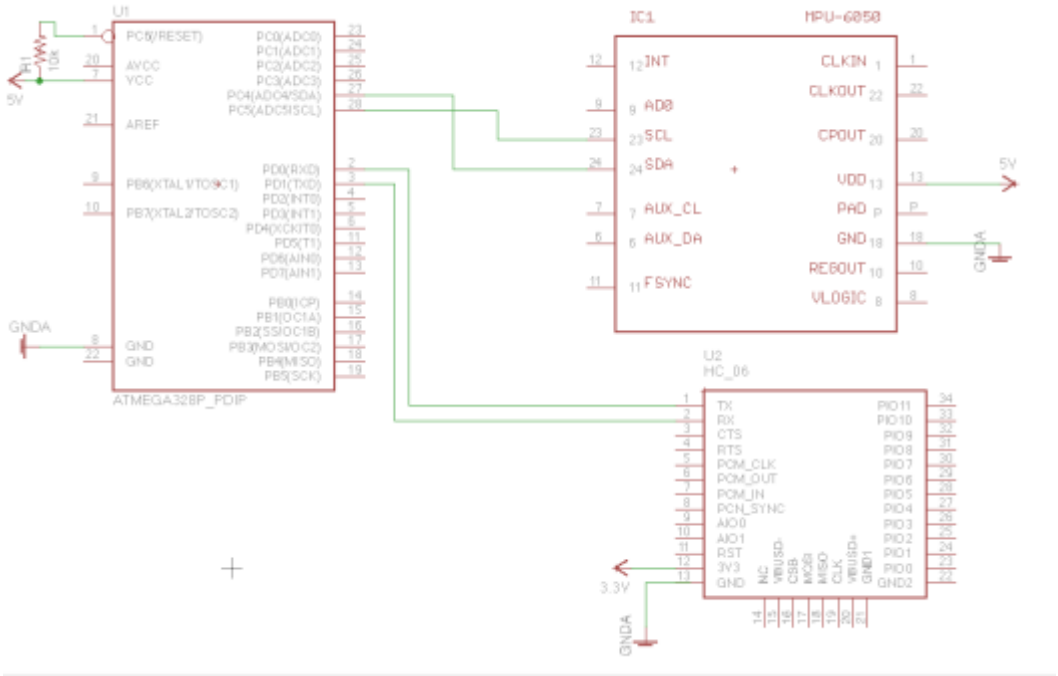
LITERATURE SURVEY:

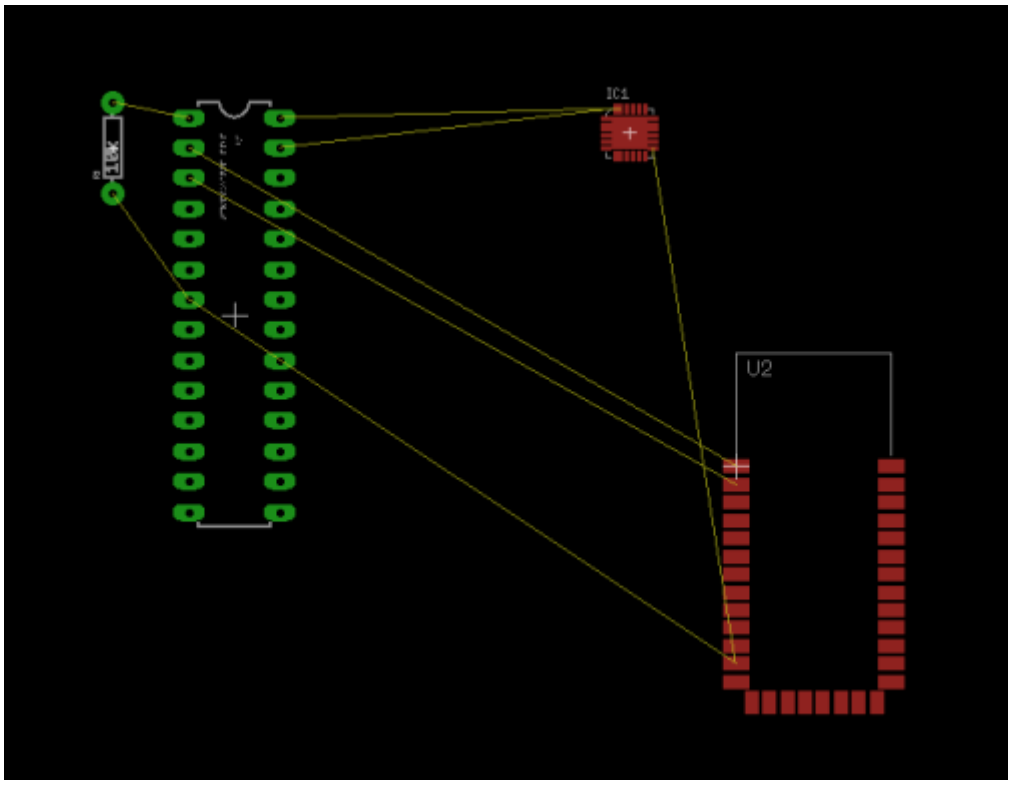
Being able to capture data from hand movements and transmit them wirelessly could be useful for controlling other systems remotely. This project could also be useful for finding the correlation between hand activity and carpal tunnel syndrome for workers whose jobs cause them to do repetitive hand movements in the field. (Ex: factory workers, farmers, construction workers, secretaries who type a lot, etc.)

COMPONENTS:

- Atmega328p
- 6 DOF Gyro/Accelerometer - MPU 6050 (Use I2C to Interface w/ Atmega328p)
- Bluetooth Module HC-06 (UART interface)
- Android Tablet with “Bluetooth Terminal/Graphics” App

SCHEMATICS:





INITIAL PCB*:

Shabrya Lott

IMPLEMENTATION

- Learn to use I2C to interface 6 DOF Gyro/Accelerometer - MPU 6050 with Atmeg328p
- Collect data (angular velocity and forces on object) from accelerometer and gyroscope and turn into degrees to obtain angular position.
- Make device wireless by adding HC-06 bluetooth and using UART interface
- Graph data (Gyro Z, pitch- rotation about X-axis, roll - rotations about y-axis) on Android tablet using "Bluetooth Terminal/Graphics" App

SNAPSHOTS/SCREENSHOTS:

DEMO VIDEO: https://youtu.be/_NsUO8qaSoY

CODE: (with comments)

```
/*
 * Final.c
 *
 * Created: 5/4/2015 7:12:58 PM
 * Used code sample from class website. Only had to modify the UART init
 * and main to average raw values and print them in a format readable for the app
 * used to graph.
 */
```

```
#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[50];

//*******************************//



void uart_init (void)
{
	//asynchronous uart, transmit 8-bit data
	UCSR0C = ((1<<UCSZ01)|(1<<UCSZ00));
	//9600 Baud Rate
	UBRR0L = 0x33;
	UCSR0B = (1<<TXEN0); //enable transmitter
}
void uart_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");
		uart_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
		snprintf(outs,sizeof(outs),"done start \n\r");
		uart_tx_string(outs);
	}
	i2c_stop();
}



int main(){
	int16_t xi1 = 0;
	int16_t yi1 = 0;
	int16_t zi1 = 0;
	int xa1, ya1, za1;
	DDRD = 0xF0;
	DDRC = 0x00;
	//declare average calibrated accelerometer values
```

```
	//initialize calibarition values
	//declare accelerometer value strings

	uart_init();//initialize uart
	i2c_init();     // init I2C interface
	_delay_ms(200);  // Wait for 200 ms.
	Init_MPU6050(MPU60501);    // sensor init
	_delay_ms(200);     // Wait for 200 ms.
	snprintf(outs,sizeof(outs),"6050 initialized \n\r");
	uart_tx_string(outs);
	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;

	//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_signe
d_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_signe
d_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_signe
d_readreg(MPU60501,0x3F);
		za1 = ((za1/3)-zi1)/2048.00;

		//convert values to printable strings (
		//displayed as ints for better    graphing visualization
		//print out the values in a special format required
		//by the Android App. Format:"EValue1,Value2,Value3.....\n"

		snprintf(outs,sizeof(outs),"E%i, ", xa1); //send x-value
		uart_tx_string(outs);
		snprintf(outs,sizeof(outs),"%i, ", ya1);  //send y-value
		uart_tx_string(outs);
		snprintf(outs,sizeof(outs),"%i\n", za1);  //send z-value
		uart_tx_string(outs);

	}
```

```
	return 0;
}



/*
 * 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 4000000UL
#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 */
```

```
#ifndef _I2CMASTER_H
#define _I2CMASTER_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
```

REFERENCE:

Class website : https://tinyurl.com/unlvcpe301s16
GITHUB sample code: https://github.com/YifanJiangPolyU/MPU6050