NAME: Candace Caimol

NSHE ID: 5001848054

TITLE: Accelerometer + Gyroscope to Wi-Fi Module

GOAL:

* Have Atmega328P microcontroller obtain readings from MPU6050 Accelerometer + Gyroscope 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 actual numerical values. This will consist of x, y, and z coordinates (from both the accelerometer and gyroscope)

DELIVERABLES:

* Have the program capable of basic device start-up (waking the MPU6050 from sleep)
* Read and record accelerometer and gyroscope measurements as actual numerical values
* Upload these values onto cloud service via Wi-Fi (assuming working Wi-Fi module)

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).

COMPONENTS:

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

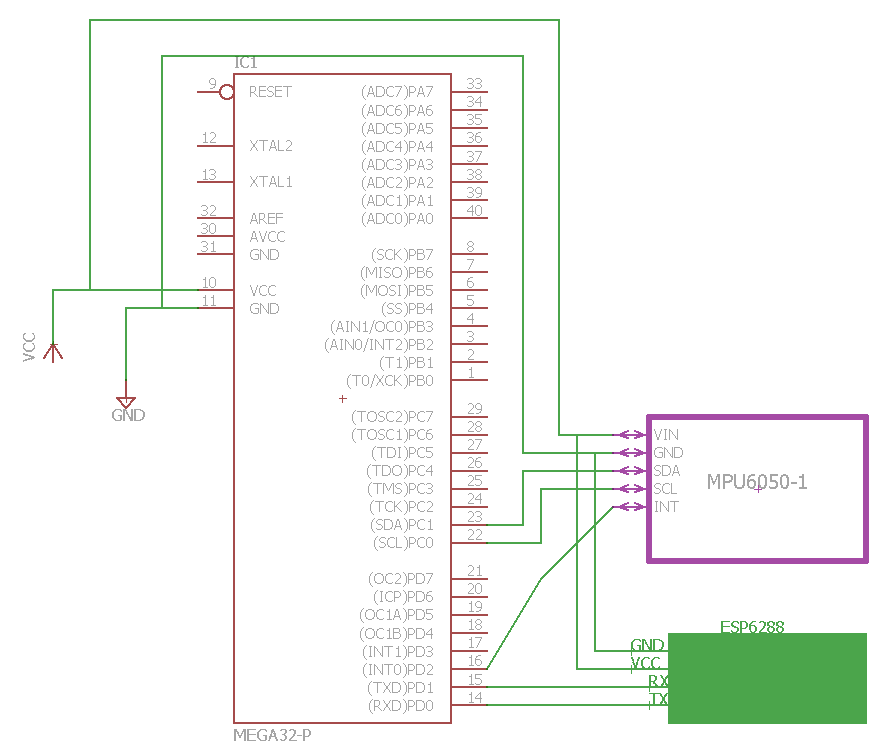
The components used in this project are the Atmega328P microcontroller, MPU6050 Accelerometer + Gyroscope, the FTDI 232R, and the ESP8266 Wi-Fi module. The Atmega328P microcontroller is the central component of this project. It will interface with the MPU6050 via I2C, where the microcontroller is the 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. It will also be connected directly to the PC using the FTDI 232R because this constant connection is required when using the AT firmware for the Wi-Fi module. The ESPlorer application will be used to program the Wi-Fi module. The advantage of using both of these is that it is easier for first-time users to use.

SCHEMATICS: (exception - include image)

* Note: The microcontroller shown below is the Atmega32, which was the closest thing available on Eagle. Therefore, the placements of the SCL and SDA pins are incorrect. Instead of pins PC0 and PC1, the locations of these pins on the Atmega328P are on PC4 and PC5.



IIMPLEMENTATION:

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. 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) [progress made so far]

/\*

\* 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\_H 0x41

#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/>

PS: \* - can be omitted in the Pre-Final Report.