**IoT With ATMega328P, A Movement Detector**

CPE 301 FINAL PROJECT

Instructor: Venki Muthukumar

Brandon Wade

5/12/2016

**Project Goal:**

* Read accelerometer values from MPU 6050 vis I2C
* ATMega328P used to read data and process data
* From the ATMega328P device, data will be sent to ESP8266 via USART
* The ESP8266 will then upload the data to ThingSpeak cloud service where the data will be graphically represented.

**Deliverables:**

The project will result in an IoT (Internet of Things) device capable of uploading sensor read data to the internet via WiFi. The ATMega328P chip was used to interface between the (MPU 6050) sensor and the (ESP8266) WiFi module.

**Literature Survey:**

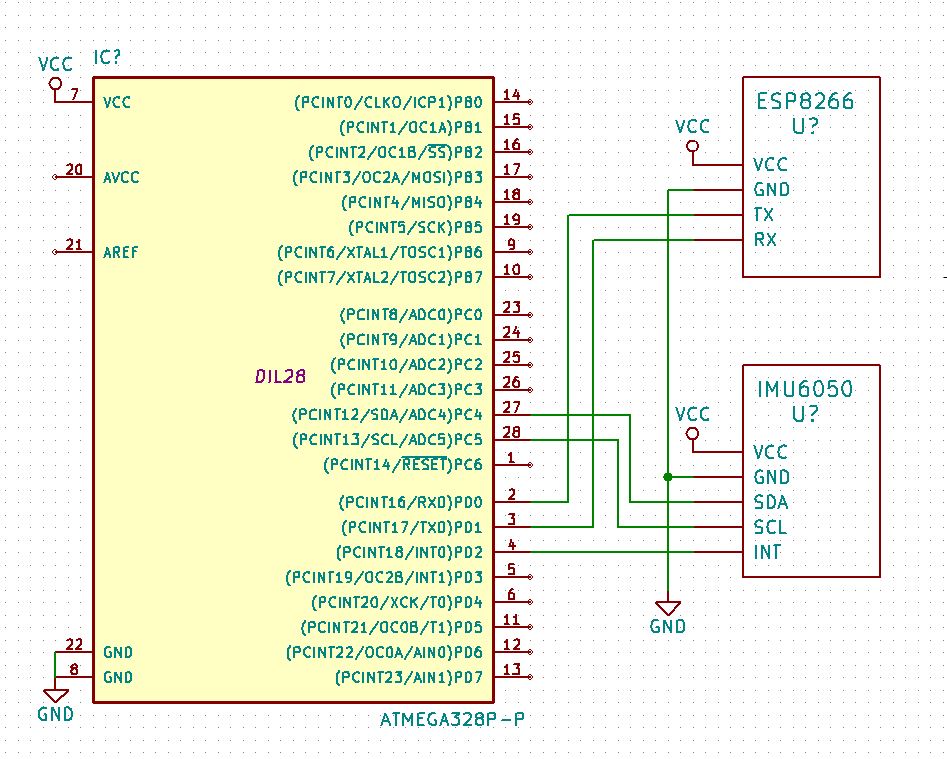
This type of project is useful for any monitoring type applications. The use of WiFi allows sensor read data to be accessed via anywhere one has a connection to the internet. This IoT idea allows data to therefore be shared across the world where anyone with access to the relevant upload site can process the data on their own. The same type of application also allows automated systems to respond to certain physical stimuli. For example, setting up this type of device with the use of a thermometer sensor could allow one to receive a text message if the temperature read reaches a certain threshold. For my particular application, because only raw values were read, the graph could show when something is disturbed, perhaps a front door opening. Therefor connecting this device to a door in your house, for example, one can monitor if and when their door has been opened.

**Components**

|  |  |
| --- | --- |
| Hardware | Software |
| ATMega328P | Atmel Studio 6.0 |
| MPU 6050 Sensor | Putty |
| ESP8266 | ESPlorer |
| SparkFun USB to Serial Breakout – FT232RL | DipTrace |
| BreadBoard | ThingSpeak |

* The project was implemented on a breadboard with all devices attached to it except for the MPU 6050 sensor (to allow for easy movement of the sensor).
* Interfaces included I2C and USART
  + I2C used to interface ATMega328P with MPU6050 sensor
  + USART used to interface with ATMega328P and ESP8266
  + ESP8266 connected to mobile hotspot via WiFi
* ThingSpeak was used to collect data. Limitation: Only updates every 15 seconds
* ATMega328P only had one set of I2C interfaces and therefor would have only been able to collecting data from one device using this interface.
* Image of Circuit Components: <http://imgur.com/QnVYNWf>

**Schematic and PCB**



<http://imgur.com/zrvD4rW>

**Implementation**

* Project was implemented on a breadboard with the MPU6050 sensor free to dangle (allowing for easier control in moving it)
* ESP8266 device was configured with AT firmware. Using ESPlorer, the device was set up to automatically connect to the mobile hotspot hosted from my phone when powered on.
* ATMega328P chip was programmed to read sensor data from MPU6050 and process the data into printable strings. The chip was also then programmed to print these strings (AT commands) continuously to USART which was also connected to the ESP8266 RX pin.
* The ESP8266 then connected to ThingSpeak. Data was uploaded by visiting a URL with the relevant data embedded (this was done by the ATMega328P chip).
* As ThingSpeak could only be updated every 15 seconds, the loop issuing these commands to the ESP8266 was delayed to keep in sync with the ThingSpeak server.

**Snapshots of Experiment**

* + Graphs show data updates over time. First graph is acceleration in the x direction, second graph is acceleration in y direction, and third graph is acceleration in the z direction. Spike in graphs indicated movement in the relevant direction while flat lines indicate resting position. Link: <http://imgur.com/Avt1Kj8>
  + Physical Circuit Snapshot: <http://imgur.com/QnVYNWf>

Final Demonstration: <https://www.youtube.com/watch?v=RngmWc71sFk>

**Main.c (Highlighted parts indicated changes from provided helper code)**

/\* Brandon Wade \*/

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <util/twi.h>

#include "I2Cmaster.h"

#include <math.h>

#include <string.h>

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

unsigned char ret; // return value

char outs[75];

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

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; // BAUD OF 115200

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

//snprintf(outs,sizeof(outs),"done start \n\r");

//USART\_tx\_string(outs);

}

i2c\_stop();

}

int main(){

int16\_t xi1 = 0;

int16\_t yi1 = 0;

int16\_t zi1 = 0;

float xa1, ya1, za1;

int x0, x1, x2, y0, y1, y2, z0, z1, z2;

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(7000); // Wait for 7s.

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; // average values…

yi1 = yi1/10;

zi1 = zi1/10;

snprintf(outs, sizeof(outs),"AT+CIPMUX=1\r\n");

USART\_tx\_string(outs);

\_delay\_ms(5000);

//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;

// Send AT commands to open channel for input, and update given fields

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

USART\_tx\_string(outs);

\_delay\_ms(5000); // delay to allow ESP to connect

snprintf(outs, sizeof(outs),"AT+CIPSEND=4,75\r\n");

USART\_tx\_string(outs);

\_delay\_ms(5000);

x0 = (int)xa1%10; // take value and find ones, tenths, and hundredths place for each x y and z position

x1 = abs((int)(xa1 \*10)%10);

x2 = abs((int)(xa1 \*100)%10);

y0 = (int)ya1%10;

y1 = abs((int)(ya1 \*10)%10);

y2 = abs((int)(ya1 \*100)%10);

z0 = (int)xa1%10;

z1 = abs((int)(za1 \*10)%10);

z2 = abs((int)(za1 \*100)%10);

snprintf(outs, sizeof(outs),"GET /update?key=7PA0P9DMZOLYS1JR&field1=%i.%i%i&field2=%i.%i%i&field3=%i.%i%i\r\n", x0,x1,x2,y0,y1,y2,z0,z1,z2); // send to my channels API key

USART\_tx\_string(outs);

USART\_tx\_string(outs); // double send to make sure values get updated

\_delay\_ms(5000);

snprintf(outs, sizeof(outs),"AT+CIPCLOSE\r\n");

USART\_tx\_string(outs);

\_delay\_ms(5000);

}

return 0;

}

**References**

<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/>

<https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf>

<http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf>

MPU6050\_helper codes provided by you