MPU6050 教程

1.1 MPU6050 简介

如果你想玩四轴，想搞什么空中鼠标，平衡车等待，那么 MPU6050 真的是太强大了，能做很多东西。

玩 MPU6050 的步骤：

1.学习 I2C，I2C 就是 MPU6050 传送数据到单片机的一种协议，类似于 USB，当然 USB 还是比较有难度的。

2.了解 MPU6050 相关寄存器，有中文版本的，一边学一边看例程就可以获取数据了。

3.把获取的数据进行各种处理。

1.2 IIC 简介

IIC 可以去看下我们野火相关的教程，在这里只是简单地介绍下，先看下我们的书或者教程，从 EEPROM 里面写入和读取数据，因为 EEPROM 写入和读取数据也是根据 I2C 协议来的。I2C 有分软件和硬件，软件就你通过对 I2C 的时钟线和数据线，可能你不知道时钟线和数据线，那还是先去学 I2C 的基础教程。

软件模拟 I2C 就是根据下面的图然后再适当的时候给时钟线和数据线高低，具体可以看 I2C 的协议见图 1-1。

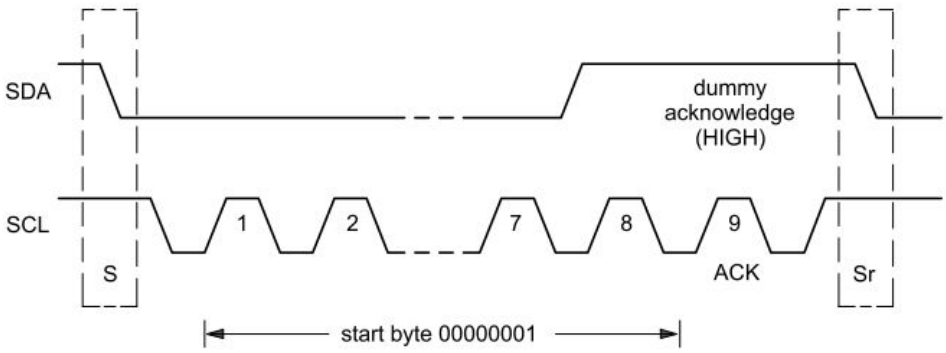

图 18　起始字节过程

图 1-1 IIC 起始字节时序图

这部分学习的诀窍就是：先写下 I2C 通讯的一个环节就好了，比如起始字节，其他的也是大同小异，直接上网找例程就好了，想要用软件模拟出全部的时序当然也可以。

还有一个方式可以用 I2C 读写数据，就是硬件 I2C，硬件 I2C 就是单片机内部的电路，可以将 I2C 的时序用硬件电路搞出来，这样子你读写数据就方便很多了。STM32 硬件 I2C 可以去看我们野火的教程。

1.3 读取 MPU6050 原始数据

我们先来认识下 MPU6050 的硬件，这是 MPU6050 模块的图片，注意是模块，中间那个才是 MPU6050，只有 MPU6050 是不够的，还要有一些外围电路才行，这就跟 51 芯片跟最小系统的区别一样。

图 1- 2 MPU6050 模块正面

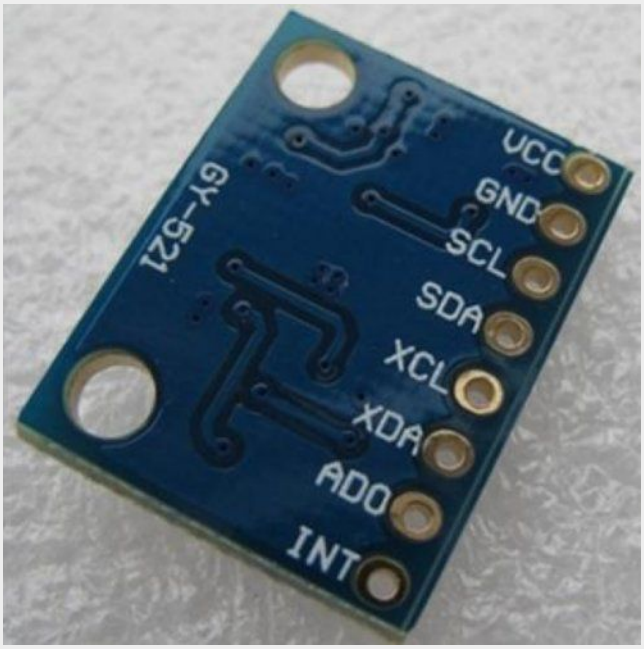

图 1- 3 MPU6050 模块背面

管脚名称	说明
VCC	3.3-5V（内部有稳压芯片）
GND	地线
SCL	MPU6050 作为从机时 IIC 时钟线
SDA	MPU6050 作为从机时 IIC 数据线
XCL	MPU6050 作为主机时 IIC 时钟线
XDA	MPU6050 作为主机时 IIC 数据线
AD0	地址管脚，该管脚决定了 IIC 地址的最低一位
INT	中断引脚

这里重点讲解 AD0 的作用，I2C 通讯中从机是要有地址的，以区别多个从机。当 AD0 管脚接低电平的时候，从机地址是 0xD0。从 MPU6050 的寄存器中我们可以得到答案，MPU6050 作为一个 IIC 从机设备的时候，有 8 位地址，高 7 位的地址是固定的，就是 WHO AM I 寄存器的默认——0x68，最低的一位是由 AD0 的连线决定的。

2进制	8进制	10进制	16进制	24进制	60
1101000	150	104	68	48	1c
11010000	320	208	D0	8G	3S

图 1-4 0x68 和 0xD0 的二进制

4.34 REGISTER 117 – WHO AM I 我是谁

WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

说明：这个寄存器用于标识设备的身份。**WHO_AM_I** 的内容是 **MPU-60X0** 的 **7** 位 **I2C** 地址的头 **6** 位。最后一位地址由 **AD0** 引脚确定。**AD0** 引脚的值跟寄存器无关。

寄存器的默认值为 **0x68**。

位 **0** 和 **7** 保留。（硬编码为 **0**）。

图 1-5 WHO AM I 寄存器

读取原始数据这个过程中一个很重要的思路就是一步一步，确保每步都对之后就很容易读出正确的数据。我们对 MPU6050 进行读写传感器数据就是对 MPU6050 的寄存器用 I2C 进行读写。我们还要了解下 MPU6050 的寄存器，这个过程跟学习 52 单片机没有什么区别，就是配置寄存器，读取数据。

先来看下软件 IIC 读取 MPU6050 例程里面的初始化 MPU6050 的函数，

```
1  void MPU6050_Init(void)
2  {
3      int i=0,j=0;
4      //在初始化之前要延时一段时间，若没有延时，则断电后再上电数据可能会出错
6      for (i=0; i<1000; i++) {
7          for (j=0; j<1000; j++) {
8              ;
9          }
10     }
11     PMU6050_WriteReg(MPU6050_RA_PWR_MGMT_1, 0x00);
12                         //解除休眠状态
13     PMU6050_WriteReg(MPU6050_RA_SMPLRT_DIV , 0x07);
14                         //陀螺仪采样率，1KHz
15     PMU6050_WriteReg(MPU6050_RA_CONFIG , 0x06);
16                         //低通滤波器的设置，截止频率是 1K，带宽是 5K
17
18     PMU6050_WriteReg(MPU6050_RA_ACCEL_CONFIG , 0x00);
19                         //配置加速度传感器工作在 2G 模式，不自检
20
21     PMU6050_WriteReg(MPU6050_RA_GYRO_CONFIG, 0x18);
22                         //陀螺仪自检及测量范围，典型值：0x1
23                         8(不自检，2000deg/s)
24 }
```

下面这句为什么可以解除休眠状态呢？至于为什么要接触休眠状态就要看 MPU6050 的 datasheet。

```
1  PMU6050_WriteReg(MPU6050_RA_PWR_MGMT_1, 0x00);
```

```
2                       //解除休眠状态
```

首先 Single_WriteI2C();是给 MPU6050 寄存器写入数据的一个函数，需要有寄存器的地址，关于寄存器的描述，教程的文件夹有两个文档。一个是中文版，一个是英文版，中文版看得不是很明白，可能翻译得不好，英语可以的可以看下英文版本的。

首先找到 PWR_MGMT_1 是某个寄存器的地址，其值如下

```
1 #define MPU6050_RA_PWR_MGMT_1       0x6B
```

我们在文档中找到地址为 0x6B 的寄存器，看下说明，可以知道为什么发送 0x00 给这个寄存器。慢慢地这样子去读，配合 MPU6050 的寄存器文档。获取数据也是差不多的。这个过程自己慢慢去看 MPU6050 的 datasheet 就好了，方法已经介绍。

如果整个过程顺利的话，你可以用串口获得类似下面的数据。怎么检验这些数据是不是对的，我直接将手放在 MPU6050 上面，可以看到温度稍微会升高，而且我移动 MPU6050 的时候，数据会变化。初步证明数据是对的。

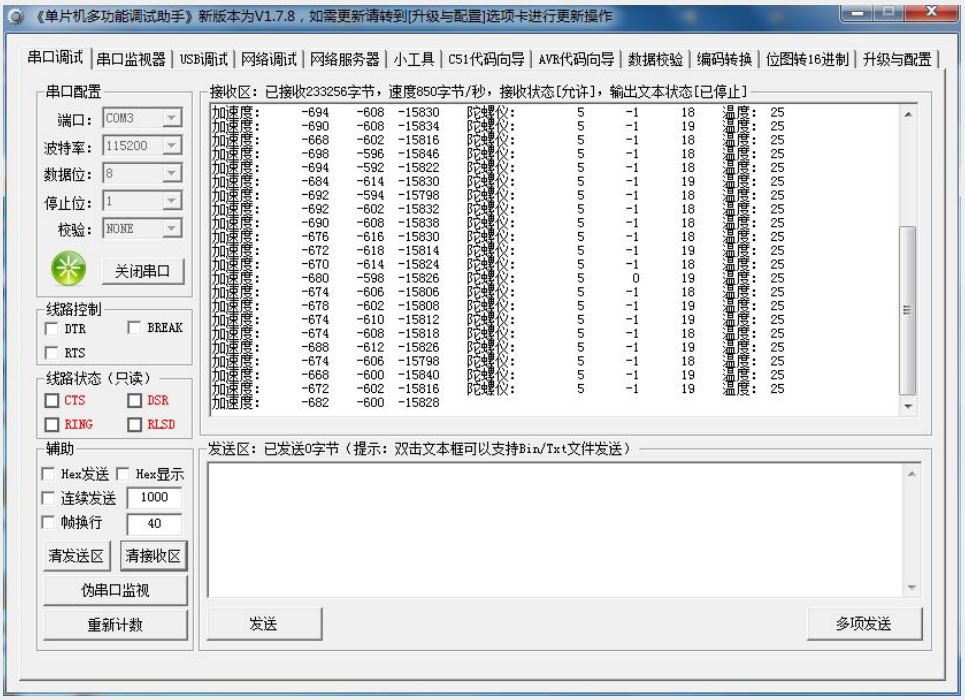

图 1-6 串口数据读取原始数据

1.4 读取 MPU6050 原始数据分析

为什么要获取这些数据呢？看完下一个小节你就会明白。这个要配合一个第七实验室的文档《MPU6050 原始数据分析》。

我们看在初始化化 MPU6050 时配置 MPU6050

```
1     PMU6050_WriteReg(MPU6050_RA_ACCEL_CONFIG , 0x00);
2                       //配置加速度传感器工作在 2G 模式，不自检
3     PMU6050_WriteReg(MPU6050_RA_GYRO_CONFIG, 0x18);
4                       //陀螺仪自检及测量范围，典型值：0x1
5                       8(不自检，2000deg/s)
```

查英文寄存器的手册可以知道我配置的陀螺仪的量程是±2000°/s，再查可知灵敏度是 16.4LSB°/s。这里还要知道量程越大，测量精度越低。LSB 的意思是最小有效位，为数字输出方式下使用；一般我们可以用 mg/LSB 来表示灵敏度，例如：mpu6050 输出的位数为 16 位(2 的 16 次方共 65536 个 LSB)对应满量程，当量程为±8g 时对应灵敏度就为 16g/65536LSB=0.244140625mv/g，取倒数为 4096LSB/g，因为 mpu6050 只能 16 位输出，所以测量范围越大，对应精度就越低。

根据第七实验室的描述一除可以知道我的陀螺仪绕 3 个轴的角速度最小是不够 1°/s，最多是 6°/s 左右。当时我是保持静止测得的。

FS_SEL selects the full scale range of the gyroscope outputs according to the following table.

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

图 1-7 陀螺仪配置跟量程的关系

从ADC值到 dps

通过I2C接口读出来的转换结果ADC值，并不是以度每秒为单位。一般按以下公式进行转换：
Anglerate = ADCrate /灵敏度

以量程为±1000°/s为例，说明如何转换。假设读取x轴的ADC值为200，从上表中得知在±1000°/s下的灵敏度为32.8LSB/(°/s) 。根据上面的公式：

Anglerate = 200/32.8 = 6.09756°/s

这就是说，MPU6050检测到模块正在以约6度每秒的速度绕X轴(或者叫在YZ平面上)旋转。
ADC值并不都是正的，请注意，当出现负数时，意味着该设备从现有的正方向相反的方向旋转。

图 1-8 第七实验室文档中对于读取陀螺仪数据的意义的解析

接下来是加速度的分析，同样查英文寄存器手册可以知道我设置的加速度例程是±2g，灵敏度是 16384LSB/g，这样子下来加速度在 y 轴上面是 1g，这个是重力加速度，说明我放的时候是 y 轴在竖直方向上的，其他的两周加速度

是接近于 0，说明我放得很垂直啊，你也可以换其他的轴在竖直方向上，看那个轴显示出来的值是否是 16384 左右，我试了，结果是。更加说明我们的结果很对啦。

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

图 1-9 加速度配置跟量程的关系

在此提出一个问题， 加速度计如何将这些信息告诉我们？
目前市面上的加速度计从输出上区分为两种，一种是数字的，另一种是模拟的. miniAHRS 使用的是MPU6050三轴加速度计，是I2C接口的数字传感器. 通过特定的命令可以配置加速度的量程, 并将内部ADC的转换结果读出来.
现在，我们有我们的加速度计的读数，以LSB为单位的，它仍然不是g（9. 8米/秒^2），需要最后的转换，我们要知道加速度计灵敏度，通常表示为LSB /g。比方说当我们选择2g的量程时，对应的灵敏度= 16384 LSB/ G 。为了得到最终的力值，单位为g，我们用下面的公式：

RX = ADCRx /灵敏度
也就是说 当x轴的计数为ADCRx 时，那么对应的加速度值就是（ADCRx/16384）g.

图 1-10 第七实验室文档中对于读取加速度数据的意义的解析

软件 IIC 读取原始数据的例程接线和验证方法如下：

VCC：连接到板子 5V 或者 3.3V

GND：连接到板子的地

SCL：连接到板子的 B10

SDA：连接到板子的 B11

AD0：连接到地

然后用串口助手看数据即可

硬件 IIC 读取原始数据的例程接线由于硬件 IIC 是固定的两个硬件管脚，B6 和 B7，所以要改下上面的 SCL 和 SDA 到 B6 和 B7 即可

1.5 圆点博士 DMP 移植

先让大家看一个 DMP 数据上传到电脑通过上位机来显示的视频

http://v.youku.com/v_show/id_XNzEyNjQ3ODEy.html

你如果是初学者，只是获取了原始数据，那只能告诉你，其实原始数据没有处理没有多大的用处。我们还需要将那些数据根据数学方面的东西转化为姿

态有关的四元数和欧拉角。这个过程有两种办法，一种你可以去学下数学，然后编程把我们上面获取的原始数据转化为四元数和欧拉角；另一种是直接用 MPU6050 内部的 DMP。这里我们介绍后者。

DMP 是什么意思？ DMP 就是指 MPU6050 内部集成的处理单元，可以直接运算出四元数和姿态，而不再需要另外进行数学运算。DMP 的使用大大简化了四轴的代码设计。DMP 是数字运动处理器的缩写，顾名思义 mpu6050 并不单单是一款传感器，其内部还包含了可以独立完成姿态解算算法的处理单元。如在设计中使用 DMP 来实现传感器融合算法优势很明显。首先，invensense 官方提供的姿态解算算法应该比像楼主这样的小白要可靠的多。其次，由 DMP 实现姿态解算算法将单片机从算法处理的压力中解放出来，单片机所要做的是等待 DMP 解算完成后产生的外部中断，在外部中断里去读取姿态解算的结果。这样单片机有大量的时间来处理诸如电机调速等其他任务，提高了系统的实时性。直接上 E 文吧。标准一点。

DMP and DMP features: The DMP is aunique hardware feature of InvenSenseMPU devices which is capable of computing quaternion data from sensor readings, performing device calibration,and also includes application specific features such as pedometerstep-counting. The DMP image (firmware) is held in volatile memory on the MPUand needs to be updated to the DMP every time the chip powers up to leveragethis functionality.

四元数和欧拉角是什么？我简单理解，简单解释。标准的自己去查下概念。四元数就是 4 个数，经过 DMP 或 数学+软件 你就可以得到四元数，四元数就是 4 个数，可以表征姿态，经过几个数学公式之后就可以的出姿态，姿态包括 pitch，roll，yaw。

我们移植的并不是官方的库，而是圆点博士的库，还包括他的 I2C，两者是配套使用的。这个移植要比官方的简单。添加例程文件里面的圆点博士的文件到工程里面，该 include 的就 include。

先进行初始化

```
1 uart_init(115200);          //串口初始化为 115200
2 //引用圆点博士的 I2C 程序，这里跟我们平常没有什么区
3 别
4 ANBT_I2C_Configuration();                    //IIC 初始化
5 AnBT_DMP_MPU6050_Init();                          //6050DMP 初始化
```

定义一些相关的变量

```
1 short gyro[3], accel[3], sensors;
2 //陀螺仪存放数组，加速度存放数组，返回状态量
3 unsigned char more;
4 long quat[4];//四元数存放数组
5 float Yaw=0.00,Roll,Pitch;//欧拉角
6 float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;
7          //计算姿态过程用到的变量
```

接着用库函数读出四元数

```
1 dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors,&more);
```

计算欧拉角，里面包含了上面所说的将四元数转化为欧拉角的公式。

```
1 if (sensors&INV_WXYZ_QUAT)
```

```
{
    q0=quat[0] / q30;
    q1=quat[1] / q30;
    q2=quat[2] / q30;
    q3=quat[3] / q30;
    Pitch = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; // pitch
    Roll = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; // roll
    Yaw =          atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3;
    //          printf("pitch: %.2f    roll:%.2f yaw:%.2f\r\n",Pitch,Roll,Yaw);
                          //普通串口输出
    Data Send Status(Pitch,Roll,Yaw,gyro,accel);
    delay init(72);
 }
```

最后写一个符合你上位机通讯协议的发送程序，我的是匿名四轴最新版上位机，匿名四轴有在他的讲解视频中讲到怎么去写通讯协议。

http://www.amobbs.com/forum.php?mod=viewthread&tid=5580829&highlight=%E5%8C%BF%E5%90%8D%E5%9B%9B%E8%BD%B4

函数如下，注意这是针对匿名最新的上位机程序的

```
/*函数功能：根据匿名最新上位机协议写的显示姿态的程序
 *具体原理看匿名的讲解视频
 */

void Data Send Status(float Pitch,float Roll,float Yaw,int16 t *gyro,int16 t *accel)
{
    unsigned char i=0;
    unsigned char  cnt=0,sum = 0;
    unsigned int  temp;
    u8 data to send[50];

    data to send[ cnt++]=0xAA;
    data to send[ cnt++]=0xAA;
    data to send[ cnt++]=0x01;
    data to send[ cnt++]=0;

     temp = (int)(Roll*100);
    data to send[ cnt++]=BYTE1( temp);
    data to send[ cnt++]=BYTE0( temp);
     temp = 0-(int)(Pitch*100);
    data to send[ cnt++]=BYTE1( temp);
    data to send[ cnt++]=BYTE0( temp);
     temp = (int)(Yaw*100);
    data to send[ cnt++]=BYTE1( temp);
    data to send[ cnt++]=BYTE0( temp);

    data to send[3] =  cnt-4;
    //和校验
    for (i=0; i< cnt; i++)
        sum+= data to send[i];
    data to send[ cnt++]=sum;

    //串口发送数据
    for (i=0; i< cnt; i++)
        AnBT Uart1 Send Char(data to send[i]);
}
```

移植成功一开始在串口输出可以看到，后面的效果就是跟视频连接的一样，如果对匿名上位机还有不懂的地方可以去看下他的上位机演示教程。

图 1-11 DMP 移植成功串口打印内容

可能有些还想要在上位机上打印出波形如下，我建议同样是去看匿名上位机的教程就可以了。

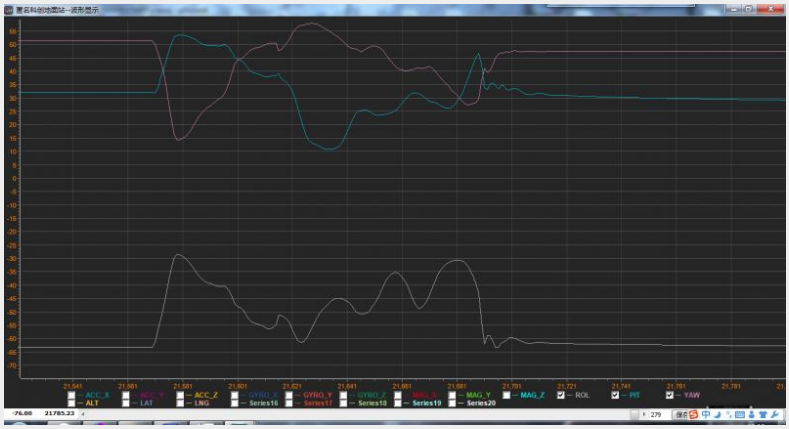

图 1-12 匿名上位机打印出波形

除了例程文件夹里面的资料，这里再附上一些网络上的资料：

DMP 官方工程

http://pan.baidu.com/s/1pJoF4pl

一个英文讲解加速度计和陀螺仪的网站：

http://www.starlino.com/imu_guide.html