PZ-MPU6050 六轴传感器模块开发手册

这一章我们来学习当下非常流行的一款六轴传感器：MPU6050，该传感器广泛用于四轴、平衡车和空中鼠标等设计，具有非常广泛的应用范围。通过本章的介绍让大家学会如何使用这款功能强大的六轴传感器。本章要实现的功能是：使用 STM32 来驱动 MPU6050，读取其原始数据，并利用其自带的 DMP 实现姿态解算，结合匿名四轴上位机软件和串口助手显示。学习本章可以参考“\4--参考资料”。本章分为如下几部分内容：

1 PZ-MPU6050 模块介绍

2 模块使用说明

3 硬件设计

4 软件设计

5 实验现象

1 PZ-MPU6050 模块介绍

1.1 特性参数

PZ-MPU6050 是深圳普中科技推出的一款高性能三轴加速度+三轴陀螺仪的六轴传感器模块。该模块采用 InvenSense 公司的 MPU6050 芯片作为核心，该芯片内部整合了 3 轴陀螺仪和 3 轴加速度传感器，并可利用自带的数字运动处理器（DMP: Digital Motion Processor）硬件加速引擎，通过主 IIC 接口，向应用端输出姿态解算后的数据。有了 DMP，我们可以使用 InvenSense 公司提供的运动处理资料库，非常方便的实现姿态解算，降低了运动处理运算对操作系统的负荷，同时大大降低了开发难度。

PZ-MPU6050 模块具有：体积小、自带 DMP、自带温度传感器、支持 IIC 从机地址设置和中断、兼容 3.3V/5V 系统、使用方便等特点。

PZ-MPU6050 模块块各项参数如下图所示。

项目	说明
接口特性	3.3V/5V
通讯接口	IIC 接口
通讯速率	400Khz(Max)
测量维度	加速度：3 维 陀螺仪：3 维
加速度测量范围	±2/±4/±8/±16g
陀螺仪测量范围	±250/±500/±1000/±2000° /秒
ADC 位数	16 位
分辨率	加速度：16384LSB/g(Max) 陀螺仪：131LSB/(° /s)(Max)
输出速率	加速度：1Khz (Max) 陀螺仪：8Khz (Max)
姿态解算输出速率	200Hz (Max)
温度传感器测量范围	-40℃~85℃
温度传感器精度	±1℃
工作温度	-40℃~85℃
模块尺寸	21.34mm*29.59mm
电源电压	3.3V/5V
IO 口电平②	3.3V LVTTL
功耗	5mA

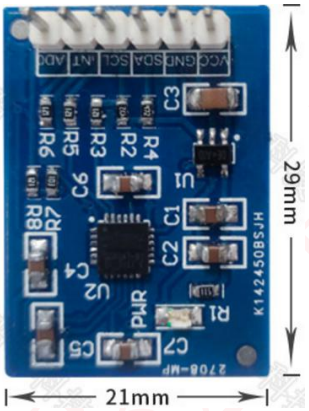
注②：模块 IO 电压是 3.3V，不过我们做了 5V 兼容性处理（串 120R 电阻），所以也可以直接连接 5V 的 MCU 使用。

2 模块使用说明

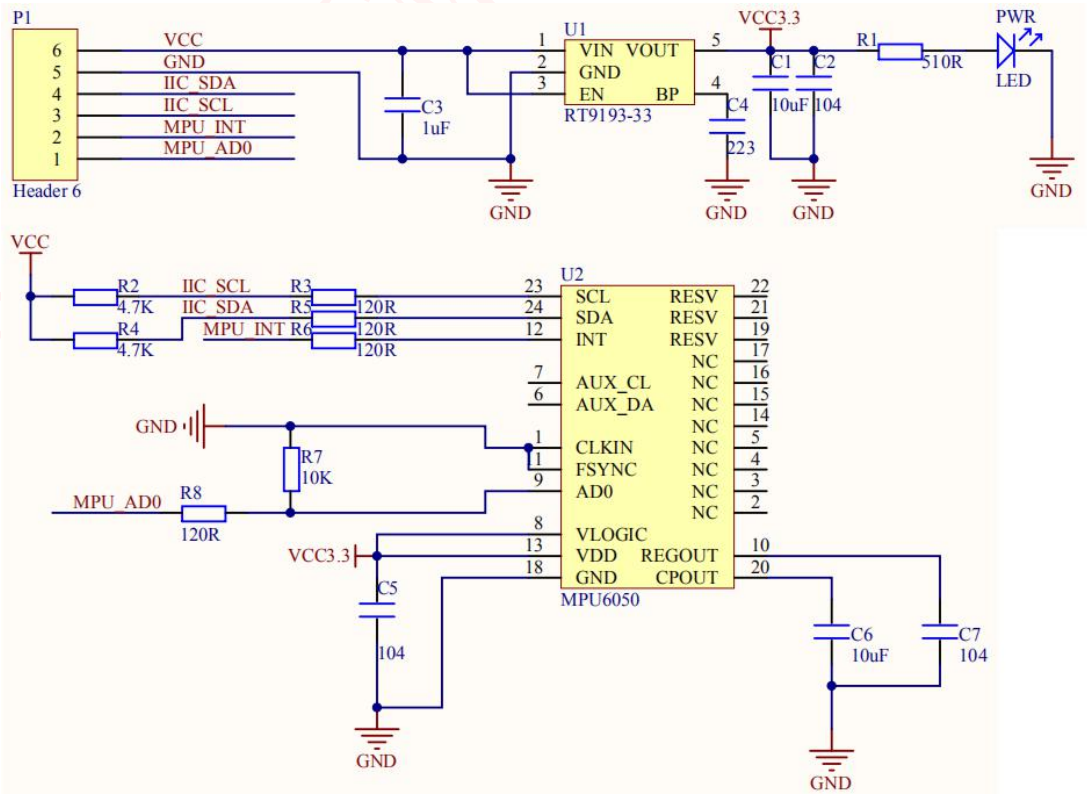
2.1 模块引脚说明

PZ-MPU6050 六轴传感器模块通过 1*6 的排针（2.54mm 间距）同外部连接，模块可以使用杜邦线与普中 T100&T200 等开发板连接进行测试。对应开发板都提供有相应例程，用户可以直接在这些开发板上，对模块进行测试。

PZ-MPU6050 六轴传感器模块外观如下图所示：



PZ-MPU6050 模块原理图如下图所示：



从图中可以看出，模块自带了 3.3V 超低压差稳压芯片，给 MPU6050 供电，因此外部供电可以选择：3.3V / 5V 都可以。模块通过 P1 排针与外部连接，引出了 VCC、GND、IIC_SDA、IIC_SCL、MPU_INT 和 MPU_AD0 等信号，其中，IIC_SDA 和 IIC_SCL 带了 4.7K 上拉电阻，外部可以不用再加上拉电阻了，另外 MPU_AD0 自带了 10K 下拉电阻，当 AD0 悬空时，默认 IIC 地址为（0X68）。

PZ-MPU6050 六轴传感器模块通过一个 1*6 的排针（P1）同外部电路连接，各引脚的详细描述如下图所示：

序号	名称	说明
1	VCC	3.3V/5V 电源输入
2	GND	地线
3	IIC_SDA	IIC 通信数据线
4	IIC_SCL	IIC 通信时钟线
5	MPU_INT	中断输出引脚
6	MPU_AD0	IIC 从机地址设置引脚； ID：0X68(悬空/接 GND) ID：0X69(接 VCC)

模块仅通过一个 IIC 接口与外部通信，并可以通过 MPU_AD0 设置模块的 IIC 地址，当 MPU_AD0 悬空/接 GND 的时候，模块的 IIC 从机地址为：0X68；当 MPU_AD0 接 VCC 的时候，模块的 IIC 从机地址为：0X69。

2.2 MPU6050 传感器介绍

2.2.1 MPU6050 传感器简介

MPU6050 是 InvenSense 公司推出的全球首款整合性 6 轴运动处理组件，相较于多组件方案，免除了组合陀螺仪与加速器时之轴间差的问题，减少了安装空间。

MPU6050 内部整合了 3 轴陀螺仪和 3 轴加速度传感器，并且含有一个第二 IIC 接口，可用于连接外部磁力传感器，并利用自带的数字运动处理器（DMP: Digital Motion Processor）硬件加速引擎，通过主 IIC 接口，向应用端输出完整的 9 轴融合演算数据。有了 DMP，我们可以使用 InvenSense 公司提供的运动处理资料库，非常方便的实现姿态解算，降低了运动处理运算对操作系统的负荷，同时大大降低了开发难度。

MPU6050 的特点包括：

1.以数字形式输出 6 轴或 9 轴（需外接磁传感器）的旋转矩阵、四元数(quaternion)、欧拉角格式(Euler Angle forma)的融合演算数据（需 DMP 支持）

2.具有 131 LSBs/°/sec 敏感度与全格感测范围为±250、±500、±1000 与±2000°/sec 的 3 轴角速度感测器(陀螺仪)

3.集成可程序控制，范围为±2g、±4g、±8g 和±16g 的 3 轴加速度传感器

4.移除加速器与陀螺仪轴间敏感度，降低设定给予的影响与感测器的飘移

5.自带数字运动处理(DMP: Digital Motion Processing)引擎可减少 MCU 复杂的融合演算数据、感测器同步化、姿势感应等的负荷

6.内建运作时间偏差与磁力感测器校正演算技术，免除了客户须另外进行校正的需求

7.自带一个数字温度传感器

8.带数字输入同步引脚(Sync pin)支持视频电子影相稳定技术与 GPS

9.可程序控制的中断(interrupt)，支持姿势识别、摇摄、画面放大缩小、滚动、快速下降中断、high-G 中断、零动作感应、触击感应、摇动感应功能

10.VDD 供电电压为 2.5V±5%、3.0V±5%、3.3V±5%；VLOGIC 可低至 1.8V ± 5%

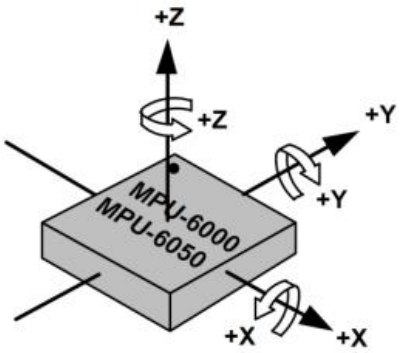
11.陀螺仪工作电流：5mA，陀螺仪待机电流：5uA；加速器工作电流：500uA，加速器省电模式电流：40uA@10Hz

12.自带 1024 字节 FIFO，有助于降低系统功耗

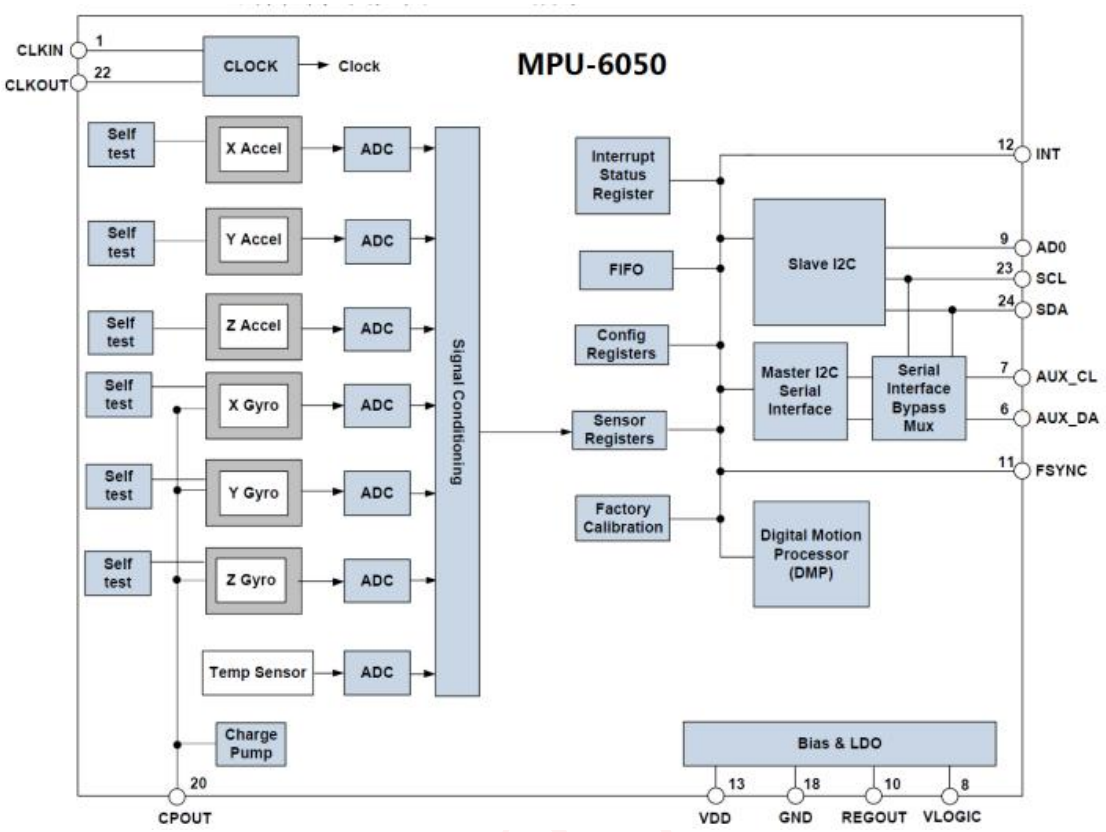
13.高达 400Khz 的 IIC 通信接口

14.超小封装尺寸：4x4x0.9mm（QFN）

MPU6050 传感器的检测轴方向如下图所示：



MPU6050 的内部框图如下图所示：



其中，SCL 和 SDA 是连接 MCU 的 IIC 接口，MCU 通过这个 IIC 接口来控制 MPU6050，另外还有一个 IIC 接口：AUX_CL 和 AUX_DA，这个接口可用来连接外部从设备，比如磁传感器，这样就可以组成一个九轴传感器。VLOGIC 是 IO 口电压，该引脚最低可以到 1.8V，我们一般直接接 VDD 即可。AD0 是从 IIC 接口（接 MCU）的地址控制引脚，该引脚控制 IIC 地址的最低位。如果接 GND，则 MPU6050 的 IIC 地址是：0X68，如果接 VDD，则是 0X69，注意：这里的地址是不包含数据传输的最低位的（最低位用来表示读写）。

我们教程中，AD0 是接 GND 的，所以 MPU6050 的 IIC 地址是 0X68（不含最低位），IIC 通信的时序在开发板基础实验“I2C-EEPROM 实验”已经介绍过，这里就不再多说了。

2.2.2 MPU6050 传感器的使用步骤

2.2.3 MPU6050 传感器的使用步骤

通过前面的介绍，我们知道了 MPU6050 是采用 IIC 通信，接下来，我们介绍使用 STM32F1 读取 MPU6050 的加速度和角度传感器数据（本实验采用非中断方式）需要哪些初始化步骤：

（1）初始化 IIC 接口

MPU6050 采用 IIC 接口与 STM32F1 通信，所以我们需要先初始化与 MPU6050 连接的 SDA 和 SCL 数据线。这个在前面的 I2C-EEPROM 实验章节已经介绍过了，这里 MPU6050 与 AT24C02 共用一个 IIC，所以初始化 IIC 完全一模一样。

（2）复位 MPU6050

这一步让 MPU6050 内部所有寄存器恢复默认值，通过对电源管理寄存器 1（0X6B）的 bit7 写 1 实现。复位后，电源管理寄存器 1 恢复默认值(0X40)，然后必须设置该寄存器为 0X00，以唤醒 MPU6050，进入正常工作状态。

（3）设置角速度传感器（陀螺仪）和加速度传感器的满量程范围

这一步，我们设置两个传感器的满量程范围(FSR)，分别通过陀螺仪配置寄存器（0X1B）和加速度传感器配置寄存器（0X1C）设置。我们一般设置陀螺仪的满量程范围为±2000dps，加速度传感器的满量程范围为±2g。

（4）设置其他参数

这里，我们还需要配置的参数有：关闭中断、关闭 AUX IIC 接口、禁止 FIFO、设置陀螺仪采样率和设置数字低通滤波器（DLPF）等。本章我们不用中断方式读取数据，所以关闭中断，然后也没用到 AUX IIC 接口外接其他传感器，所以也关闭这个接口。分别通过中断使能寄存器（0X38）和用户控制寄存器（0X6A）控制。MPU6050 可以使用 FIFO 存储传感器数据，不过本章我们没有用到，所以关闭所有 FIFO 通道，这个通过 FIFO 使能寄存器（0X23）控制，默认都是 0（即禁止 FIFO），所以用默认值就可以了。陀螺仪采样率通过采样率分频寄存器（0X19）控制，这个采样率我们一般设置为 50 即可。数字低通滤波器(DLPF)则通过配置寄存器（0X1A）设置，一般设置 DLPF 为带宽的 1/2 即可。

（5）配置系统时钟源并使能角速度传感器和加速度传感器

系统时钟源同样是通过电源管理寄存器 1（0X6B）来设置，该寄存器的最低三位用于设置系统时钟源选择，默认值是 0（内部 8M RC 震荡），不过我们一

般设置为 1，选择 x 轴陀螺 PLL 作为时钟源，以获得更高精度的时钟。同时，使能角速度传感器和加速度传感器，这两个操作通过电源管理寄存器 2（0X6C）来设置，设置对应位为 0 即可开启。

至此， MPU6050 的初始化就完成了，可以正常工作了（其他未设置的寄存器全部采用默认值即可），接下来，我们就可以读取相关寄存器，得到加速度传感器、角速度传感器和温度传感器的数据了。不过，我们先简单介绍几个重要的寄存器。

首先，我们介绍电源管理寄存器 1，该寄存器地址为 0X6B，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

其中，DEVICE_RESET 位用来控制复位，设置为 1，复位 MPU6050，复位结束后， MPU 硬件自动清零该位。SLEEEP 位用于控制 MPU6050 的工作模式，复位后，该位为 1，即进入了睡眠模式（低功耗），所以我们要清零该位，以进入正常工作模式。TEMP_DIS 用于设置是否使能温度传感器，设置为 0，则使能。最后 CLKSEL[2:0]用于选择系统时钟源，选择关系如下：

CLKSEL[2:0]	时钟源
0	内部 8M RC 晶振
1	PLL，使用 X 轴陀螺作为参考
10	PLL，使用 Y 轴陀螺作为参考
11	PLL，使用 Z 轴陀螺作为参考
100	PLL，使用外部 32.768Khz 作为参考
101	PLL，使用外部 19.2Mhz 作为参考
110	保留
111	关闭时钟，保持时序产生电路复位状态

默认是使用内部 8M RC 晶振的，精度不高，所以我们一般选择 X/Y/Z 轴陀螺作为参考的 PLL 作为时钟源，一般设置 CLKSEL=001 即可。

接着，我们看陀螺仪配置寄存器，该寄存器地址为：0X1B，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

该寄存器我们只关心 FS_SEL[1:0]这两个位，用于设置陀螺仪的满量程范围： 0，±250°/S；1，±500° /S； 2，±1000° /S； 3，±2000° /S；我

们一般设置为 3，即±2000° /S，因为陀螺仪的 ADC 为 16 位分辨率，所以得到灵敏度为： 65536/4000=16.4LSB/(° /S)。

接下来，我们看加速度传感器配置寄存器，寄存器地址为： 0X1C，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

该寄存器我们只关心 AFS_SEL[1:0]这两个位，用于设置加速度传感器的满量程范围：0，±2g； 1，±4g； 2，±8g； 3，±16g；我们一般设置为 0，即±2g，因为加速度传感器的 ADC 也是 16 位，所以得到灵敏度为：65536/4=16384LSB/g。

接下来，我看看 FIFO 使能寄存器，寄存器地址为： 0X23，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
23	35	TEMP_ FIFO_EN	XG_ FIFO_EN	YG_ FIFO_EN	ZG_ FIFO_EN	ACCEL _FIFO_EN	SLV2 _FIFO_EN	SLV1 _FIFO_EN	SLV0 _FIFO_EN

该寄存器用于控制 FIFO 使能，在简单读取传感器数据的时候，可以不用 FIFO，设置对应位为 0 即可禁止 FIFO，设置为 1，则使能 FIFO。注意加速度传感器的 3 个轴，全由 1 个位（ACCEL_FIFO_EN）控制，只要该位置 1，则加速度传感器的三个通道都开启 FIFO 了。

接下来，我们看陀螺仪采样率分频寄存器，寄存器地址为： 0X19，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV[7:0]							

该寄存器用于设置 MPU6050 的陀螺仪采样频率，计算公式是：

采样频率 = 陀螺仪输出频率 / (1+SMPLRT_DIV)

这里陀螺仪的输出频率，是 1Khz 或者 8Khz，与数字低通滤波器（ DLPF）的设置有关，当 DLPF_CFG=0/7 的时候，频率为 8Khz，其他情况是 1Khz。而且 DLPF 滤波频率一般设置为采样率的一半。采样率，我们假定设置为 50Hz，那么 SMPLRT_DIV=1000/50-1=19。

接下来，我们看配置寄存器，寄存器地址为：0X1A，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

这里，我们主要关心数字低通滤波器（DLPF）的设置位，即：DLPF_CFG[2:0]，加速度计和陀螺仪，都是根据这三个位的配置进行过滤的。DLPF_CFG 不同配置对应的过滤情况如下：

DLPF_CFG[2:0]	加速度传感器 Fs=1Khz		角速度传感器（陀螺仪）		
	带宽(Hz)	延迟（ms）	带宽(Hz)	延迟（ms）	Fs(Khz)
000	260	0	256	0.98	8
001	184	2.0	188	1.9	1
010	94	3.0	98	2.8	1
011	44	4.9	42	4.8	1
100	21	8.5	20	8.3	1
101	10	13.8	10	13.4	1
110	5	19.0	5	18.6	1
111	保留		保留		8

这里的加速度传感器，输出速率（Fs）固定是 1Khz，而角速度传感器的输出速率（Fs），则根据 DLPF_CFG 的配置有所不同。一般我们设置角速度传感器的带宽为其采样率的一半，如前面所说的，如果设置采样率为 50Hz，那么带宽就应该设置为 25Hz，取近似值 20Hz，就应该设置 DLPF_CFG=100。

接下来，我们看电源管理寄存器 2，寄存器地址为：0X6C，各位描述如下：

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6C	108	LP_WAKE_CTRL[1:0]		STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG

该寄存器的 LP_WAKE_CTRL 用于控制低功耗时的唤醒频率，本章用不到。剩下的 6 位，分别控制加速度和陀螺仪的 x/y/z 轴是否进入待机模式，这里我们全部都不进入待机模式，所以全部设置为 0 即可。

接下来，我们看看陀螺仪数据输出寄存器，总共有 6 个寄存器组成，地址为： 0X43~0X48，通过读取这 6 个寄存器，就可以读到陀螺仪 x/y/z 轴的值，比如 x 轴的数据，可以通过读取 0X43（高 8 位）和 0X44（低 8 位）寄存器得到，其他轴以此类推。

同样，加速度传感器数据输出寄存器，也有 6 个，地址为： 0X3B~0X40，通过读取这 6 个寄存器，就可以读到加速度传感器 x/y/z 轴的值，比如读 x 轴

的数据，可以通过读取 0X3B（高 8 位）和 0X3C（低 8 位）寄存器得到，其他轴以此类推。

最后，温度传感器的值，可以通过读取 0X41（高 8 位）和 0X42（低 8 位）寄存器得到，温度换算公式为：

```
Temperature = 36.53 + regval/340
```

其中，Temperature 为计算得到的温度值，单位为℃，regval 为从 0X41 和 0X42 读到的温度传感器值。

关于 MPU6050 的基本使用，我们就介绍到这。MPU6050 的详细资料和相关寄存器介绍，请参考资料了解。MPU6050 涉及的知识非常多，所以要耐心学习。

2.3 利用 DMP 进行姿态解算

经过上一节的介绍，我们可以读出 MPU6050 的加速度传感器和角速度传感器的原始数据。不过这些原始数据，对想搞四轴之类的初学者来说，用处不大，我们期望得到的是姿态数据，也就是欧拉角：航向角（yaw）、横滚角（roll）和俯仰角（pitch）。有了这三个角，我们就可以得到当前四轴的姿态，这才是我们想要的结果。

要得到欧拉角数据，就得利用我们的原始数据，进行姿态融合解算，这个比较复杂，知识点比较多，初学者不易掌握。而 MPU6050 自带了数字运动处理器，即 DMP，并且，InvenSense 提供了一个 MPU6050 的嵌入式运动驱动库，结合 MPU6050 的 DMP，可以将我们的原始数据，直接转换成四元数输出，而得到四元数之后，就可以很方便的计算出欧拉角，从而得到 yaw、roll 和 pitch。

使用内置的 DMP，大大简化了四轴的代码设计，且 MCU 不用进行姿态解算过程，大大降低了 MCU 的负担，从而有更多的时间去处理其他事件，提高系统实时性。

使用 MPU6050 的 DMP 输出的四元数是 q30 格式的，也就是浮点数放大了 2^30 倍。在换算成欧拉角之前，必须先将其转换为浮点数，也就是除以 2^30 ，然后再进行计算，计算公式为：

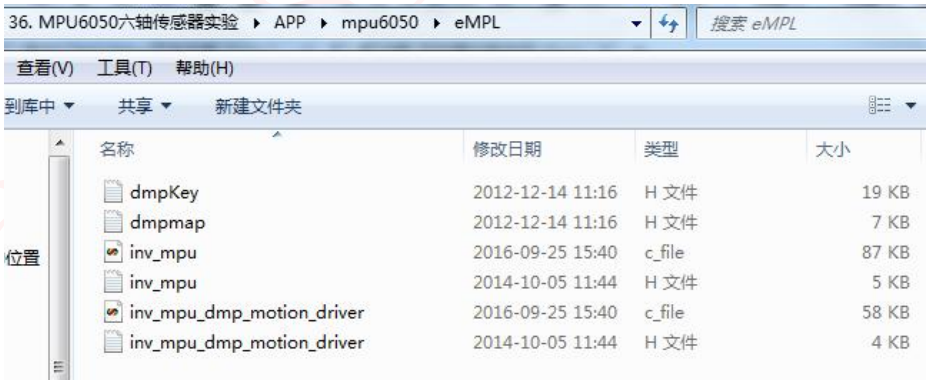
```
1.  q0=quat[0] / q30; //q30 格式转换为浮点数
2.  q1=quat[1] / q30;
3.  q2=quat[2] / q30;
4.  q3=quat[3] / q30;
```

```
//计算得到俯仰角/横滚角/航向角
pitch=asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; //俯仰角
roll=atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; //横滚角
yaw=atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3; //航向角
```

其中 quat[0]~ quat[3]是 MPU6050 的 DMP 解算后的四元数， q30 格式，所以要除以一个 2^30 ，其中 q30 是一个常量：1073741824，即 2 ^30 次方，然后带入公式，计算出欧拉角。上述计算公式的 57.3 是弧度转换为角度，即 180/π，这样得到的结果就是以度（°）为单位的。关于四元数与欧拉角的公式推导，这里我们不进行讲解，感兴趣的朋友，可以自行查阅相关资料学习。

InvenSense 提供的 MPU6050 运动驱动库是基于 MSP430 的，我们需要将其移植到 STM32F7 上面，官方原版驱动在资料“\MPU6050 资料\DMP 资料\Embedded_MotionDriver_5.1”，这就是官方原版的驱动，代码比较多，不过官方提供了两个资料供大家学习：Embedded Motion Driver V5.1.1 API 说明.pdf 和 Embedded Motion DriverV5.1.1 教程.pdf，这两个文件都在 DMP 资料文件夹里面，大家可以阅读这两个文件，来熟悉官方驱动库的使用。

官方 DMP 驱动库移植起来，还是比较简单的，主要是实现这 4 个函数：i2c_write， i2c_read，delay_ms 和 get_ms，具体细节，我们就不详细介绍了，移植后的驱动代码，我们放在本实验例程的“\APP\mpu6050\eMPL”文件夹内，总共 6 个文件，如下所示：



该驱动库，重点就是两个 c 文件： inv_mpu.c 和 inv_mpu_dmp_motion_driver.c。其中我们在 inv_mpu.c 添加了几个函数，方便我们使用，重点是两个函数：mpu_dmp_init 和 mpu_dmp_get_data 这两个函数，这里我们简单介绍下这两个函数。

mpu_dmp_init，是 MPU6050 DMP 初始化函数，该函数代码如下：

```
//mpu6050,dmp 初始化
//返回值:0,正常
//    其他,失败
u8 mpu_dmp_init(void)
{
    u8 res=0;
    IIC_Init();         //初始化 IIC 总线
    if(mpu_init()==0)   //初始化 MPU6050
    {
        res=mpu_set_sensors(INV_XYZ_GYRO|INV_XYZ_ACCEL);//设置所需要的传感器
        if(res)return 1;
        res=mpu_configure_fifo(INV_XYZ_GYRO | INV_XYZ_ACCEL);//设置 FIFO
        if(res)return 2;
        res=mpu_set_sample_rate(DEFAULT_MPU_HZ);    //设置采样率
        if(res)return 3;
        res=dmp_load_motion_driver_firmware();      //加载 dmp 固件
        if(res)return 4;
        res=dmp_set_orientation(inv_orientation_matrix_to_scalar(gyro_orientation));//设置陀螺仪方向
        if(res)return 5;
        res=dmp_enable_feature(DMP_FEATURE_6X_LP_QUAT|DMP_FEATURE_TAP|  //设置 dmp 功能
            DMP_FEATURE_ANDROID_ORIENT|DMP_FEATURE_SEND_RAW_ACCEL|DMP_FEATURE_SEND_CAL_GYRO
|
            DMP_FEATURE_GYRO_CAL);
        if(res)return 6;
        res=dmp_set_fifo_rate(DEFAULT_MPU_HZ);  //设置 DMP 输出速率(最大不超过 200Hz)
        if(res)return 7;
        res=run_self_test();        //自检
        if(res)return 8;
        res=mpu_set_dmp_state(1);   //使能 DMP
        if(res)return 9;
    }
    return 0;
}
```

此函数首先通过 IIC_Init(需外部提供)初始化与 MPU6050 连接的 IIC 接口，然后调用 mpu_init 函数，初始化 MPU6050，之后就是设置 DMP 所用传感器、 FIFO、采样率和加载固件等一些列操作，在所有操作都正常之后，最后通过 mpu_set_dmp_state(1)使能 DMP 功能，在使能成功以后，我们便可以通过 mpu_dmp_get_data 来读取姿态解算后的数据了。

mpu_dmp_get_data 函数代码如下：

```
//得到 dmp 处理后的数据(注意,本函数需要比较多堆栈,局部变量有点多)
```

```
    //pitch:俯仰角 精度:0.1°   范围:-90.0° <---> +90.0°
    //roll:横滚角  精度:0.1°   范围:-180.0°<---> +180.0°
    //yaw:航向角   精度:0.1°   范围:-180.0°<---> +180.0°
    //返回值:0,正常
    //    其他,失败
    u8 mpu_dmp_get_data(float *pitch,float *roll,float *yaw)
    {
        float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;
        unsigned long sensor_timestamp;
        short gyro[3], accel[3], sensors;
        unsigned char more;
        long quat[4];
        if(dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors,&more))return 1;
        /* Gyro and accel data are written to the FIFO by the DMP in chip frame and hardware units.
         * This behavior is convenient because it keeps the gyro and accel outputs of dmp_read_fifo and mpu_read_fifo consistent.
        **/
        /*if (sensors & INV_XYZ_GYRO )
        send_packet(PACKET_TYPE_GYRO, gyro);
        if (sensors & INV_XYZ_ACCEL)
        send_packet(PACKET_TYPE_ACCEL, accel); */
        /* Unlike gyro and accel, quaternions are written to the FIFO in the body frame, q30.
         * The orientation is set by the scalar passed to dmp_set_orientation during initialization.
        **/
        if(sensors&INV_WXYZ_QUAT)
        {
            q0 = quat[0] / q30; //q30 格式转换为浮点数
            q1 = quat[1] / q30;
            q2 = quat[2] / q30;
            q3 = quat[3] / q30;
            //计算得到俯仰角/横滚角/航向角
            *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;   //yaw
        }else return 2;
        return 0;
    }
```

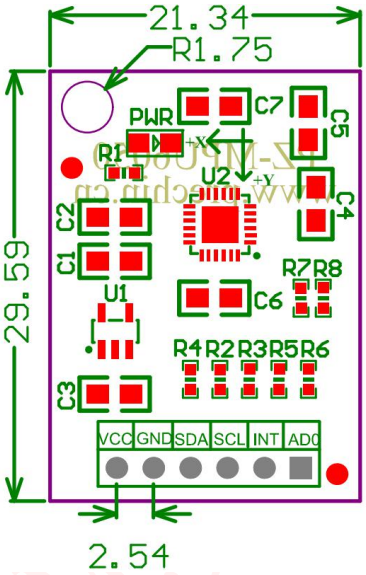
此函数用于得到 DMP 姿态解算后的俯仰角、横滚角和航向角。不过本函数局部变量有点多，大家在使用的时候，如果死机，那么请设置堆栈大一点(在 startup_stm32f10x.s 里面设置，默认是 400)。这里就用到了我们前面介绍的

四元数转欧拉角公式，将 dmp_read_fifo 函数读到的 q30 格式四元数转换成欧拉角。

利用这两个函数，我们就可以读取到姿态解算后的欧拉角，使用非常方便。

2.4 结构尺寸

PZ-MPU6050 六轴传感器模块的尺寸结构如下图所示：



3 硬件设计

3.1 硬件准备

本实验所需要的硬件资源如下：

①Z100&Z200 开发板 1 个

②PZ-MPU6050 模块 1 个

③USB 线一条（用于供电和模块与电脑串口调试助手通信）

3.2 模块与开发板连接

下面我们介绍下 PZ-MPU6050 模块与 Z100&Z200 开发板的连接关系，如下图所示：（PZ-MPU6050 模块-->STM32 IO）

```
VCC-->5V
GND-->GND
SDA-->PB9
SCL-->PB8
```

模块与开发板只需要连接 4 根线即可通信。因为 MPU_AD0 自带了 10K 下拉电阻，当 AD0 悬空时，默认 IIC 地址为（0X68）。MPU_INT 我们没有用到。

4 软件设计

本章所要实现的功能是：使用 STM32 来驱动 MPU6050，读取其原始数据，并利用其自带的 DMP 实现姿态解算，结合匿名四轴上位机软件和 TFTLCD 显示。本章实验我们使用 KEY_UP 键来控制数据（加速度传感器、陀螺仪、 DMP 姿态解算后的欧拉角等数据）上传，程序框架如下：

（1）初始化 MPU6050

（2）读取 MPU6050 的加速度、陀螺仪原始数据和温度

（4）使用 DMP 进行姿态解算获取欧拉角

（5）将 DMP 解算后的数据打包上传给上位机

（6）编写主函数

这些步骤前面我们都已介绍。下面我们打开实验工程，在 APP 工程组中可以看到添加了 mpu6050.c、inv_mpu.c、inv_mpu_dmp_motion_driver.c 文件（里面包含了 MPU6050 及 DMP 解算的驱动程序），在 StdPeriph_Driver 工程组中并未添加新的库文件，仍然采用的是一个工程的模板。记住添加原文件时，还要包含对应的头文件路径。

这里我们分析几个重要函数，其他部分程序大家可以打开工程查看。

4.1 MPU6050 初始化函数

要使用 MPU6050 首先就要对他进行初始化，初始化步骤在前面 MPU6050 的使用已介绍，具体代码如下：

```
1.  //初始化 MPU6050
2.  //返回值:0,成功
```

```
//    其他,错误代码
u8 MPU6050_Init(void)
{
    u8 res;
    IIC_Init();//初始化 IIC 总线
    MPU6050_Write_Byte(MPU6050_PWR_MGMT1_REG,0X80); //复位 MPU6050
    delay_ms(100);
    MPU6050_Write_Byte(MPU6050_PWR_MGMT1_REG,0X00); //唤醒 MPU6050
    MPU6050_Set_Gyro_Fsr(3);                    //陀螺仪传感器,±2000dps
    MPU6050_Set_Accel_Fsr(0);                   //加速度传感器,±2g
    MPU6050_Set_Rate(50);                       //设置采样率 50Hz
    MPU6050_Write_Byte(MPU6050_INT_EN_REG,0X00);    //关闭所有中断
    MPU6050_Write_Byte(MPU6050_USER_CTRL_REG,0X00); //I2C 主模式关闭
    MPU6050_Write_Byte(MPU6050_FIFO_EN_REG,0X00);   //关闭 FIFO
    MPU6050_Write_Byte(MPU6050_INTBP_CFG_REG,0X80); //INT 引脚低电平有效
    res=MPU6050_Read_Byte(MPU6050_DEVICE_ID_REG);
    if(res==MPU6050_ADDR)//器件 ID 正确
    {
        MPU6050_Write_Byte(MPU6050_PWR_MGMT1_REG,0X01); //设置 CLKSEL,PLL X 轴为参考
        MPU6050_Write_Byte(MPU6050_PWR_MGMT2_REG,0X00); //加速度与陀螺仪都工作
        MPU6050_Set_Rate(50);                       //设置采样率为 50Hz
    }else return 1;
    return 0;
}
```

4.2 获取 MPU6050 原始值和温度值函数

MPU 初始化成功后接下来就可以读取传感器数据。具体代码如下：

```
//得到温度值
//返回值:温度值(扩大了 100 倍)
short MPU6050_Get_Temperature(void)
{
    u8 buf[2];
    short raw;
    float temp;
    MPU6050_Read_Len(MPU6050_ADDR,MPU6050_TEMP_OUTH_REG,2,buf);
    raw=((u16)buf[0]<<8)|buf[1];
    temp=36.53+((double)raw)/340;
    return temp*100;;
}
//得到陀螺仪值(原始值)
//gx,gy,gz:陀螺仪 x,y,z 轴的原始读数(带符号)
//返回值:0,成功
```

```
//    其他,错误代码
u8 MPU6050_Get_Gyroscope(short *gx,short *gy,short *gz)
{
    u8 buf[6],res;
    res=MPU6050_Read_Len(MPU6050_ADDR,MPU6050_GYRO_XOUTH_REG,6,buf);
    if(res==0)
    {
        *gx=((u16)buf[0]<<8)|buf[1];
        *gy=((u16)buf[2]<<8)|buf[3];
        *gz=((u16)buf[4]<<8)|buf[5];
    }
    return res;;
}
//得到加速度值(原始值)
//gx,gy,gz:陀螺仪 x,y,z 轴的原始读数(带符号)
//返回值:0,成功
//    其他,错误代码
u8 MPU6050_Get_Accelerometer(short *ax,short *ay,short *az)
{
    u8 buf[6],res;
    res=MPU6050_Read_Len(MPU6050_ADDR,MPU6050_ACCEL_XOUTH_REG,6,buf);
    if(res==0)
    {
        *ax=((u16)buf[0]<<8)|buf[1];
        *ay=((u16)buf[2]<<8)|buf[3];
        *az=((u16)buf[4]<<8)|buf[5];
    }
    return res;;
}
```

MPU_Get_Temperature 函数用于获取 MPU6050 自带温度传感器的温度值，通过读取相应寄存器获取温度数据。这些寄存器的宏都在 mpu6050.h 文件中定义了。MPU_Get_Gyroscope 和 MPU_Get_Accelerometer 函数分别用于读取陀螺仪和加速度传感器的原始数据。

这些函数内部还调用了 MPU6050_Write_Len 和 MPU6050_Read_Len 函数，具体代码如下：

```
//IIC 连续写
//addr:器件地址
//reg:寄存器地址
//len:写入长度
//buf:数据区
//返回值:0,正常
```

```
//    其他,错误代码
u8 MPU6050_Write_Len(u8 addr,u8 reg,u8 len,u8 *buf)
{
    u8 i;
    IIC_Start();
    IIC_Send_Byte((addr<<1)|0);//发送器件地址+写命令
    if(IIC_Wait_Ack())  //等待应答
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(reg); //写寄存器地址
    IIC_Wait_Ack();     //等待应答
    for(i=0;i<len;i++)
    {
        IIC_Send_Byte(buf[i]);  //发送数据
        if(IIC_Wait_Ack())      //等待 ACK
        {
            IIC_Stop();
            return 1;
        }
    }
    IIC_Stop();
    return 0;
}
//IIC 连续读
//addr:器件地址
//reg:要读取的寄存器地址
//len:要读取的长度
//buf:读取到的数据存储区
//返回值:0,正常
//    其他,错误代码
u8 MPU6050_Read_Len(u8 addr,u8 reg,u8 len,u8 *buf)
{
    IIC_Start();
    IIC_Send_Byte((addr<<1)|0);//发送器件地址+写命令
    if(IIC_Wait_Ack())  //等待应答
    {
        IIC_Stop();
        return 1;
    }
    IIC_Send_Byte(reg); //写寄存器地址
    IIC_Wait_Ack();     //等待应答
    IIC_Start();
```

```
51.     IIC_Send_Byte((addr<<1)|1);//发送器件地址+读命令
52.     IIC_Wait_Ack();     //等待应答
53.     while(len)
54.     {
55.         if(len==1)*buf=IIC_Read_Byte(0);//读数据,发送 nACK
56.         else *buf=IIC_Read_Byte(1);     //读数据,发送 ACK
57.         len--;
58.         buf++;
59.     }
60.     IIC_Stop(); //产生一个停止条件
61.     return 0;
62. }
```

MPU6050_Write_Len 函数用于指定器件和地址，连续写数据，可用于实现 DMP 部分的： i2c_write 函数。而 MPU6050_Read_Len 函数用于指定器件和地址，连续读数据，可用于实现 DMP 部分的： i2c_read 函数。前面介绍的 DMP 移植部分的 4 个函数，这里就实现了 2 个，剩下的 delay_ms 就直接采用我们 SysTick.c 里面的 delay_ms 实现，get_ms 则直接提供一个空函数即可。

4.3 获取 DMP 解算后的欧拉角函数

我们从 MOU6050 传感器读取出来的是原始数据，如果要将这些应用到项目中，通过都是需要进行 DMP 姿态解算的，即最终需要或得欧拉角。实现 DMP 解算过程在前面部分已介绍，这里我们看下主要代码，如下：

```
1.  //得到 dmp 处理后的数据(注意,本函数需要比较多堆栈,局部变量有点多)
2.  //pitch:俯仰角 精度:0.1°   范围:-90.0° <---> +90.0°
3.  //roll:横滚角  精度:0.1°   范围:-180.0°<---> +180.0°
4.  //yaw:航向角   精度:0.1°   范围:-180.0°<---> +180.0°
5.  //返回值:0,正常
6.  //    其他,失败
7.  u8 mpu_dmp_get_data(float *pitch,float *roll,float *yaw)
8.  {
9.      float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;
10.     unsigned long sensor_timestamp;
11.     short gyro[3], accel[3], sensors;
12.     unsigned char more;
13.     long quat[4];
14.     if(dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors,&more))return 1;
15.     /* Gyro and accel data are written to the FIFO by the DMP in chip frame and hardware un
    its.
```

```
16.         * This behavior is convenient because it keeps the gyro and accel outputs of dmp_read_
    fifo and mpu_read_fifo consistent.
17.       **/
18.       /*if (sensors & INV_XYZ_GYRO )
19.       send_packet(PACKET_TYPE_GYRO, gyro);
20.       if (sensors & INV_XYZ_ACCEL)
21.       send_packet(PACKET_TYPE_ACCEL, accel); */
22.       /* Unlike gyro and accel, quaternions are written to the FIFO in the body frame, q30.
23.        * The orientation is set by the scalar passed to dmp_set_orientation during initializa
    tion.
24.       **/
25.       if(sensors&INV_WXYZ_QUAT)
26.       {
27.           q0 = quat[0] / q30; //q30 格式转换为浮点数
28.           q1 = quat[1] / q30;
29.           q2 = quat[2] / q30;
30.           q3 = quat[3] / q30;
31.           //计算得到俯仰角/横滚角/航向角
32.           *pitch = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; // pitch
33.           *roll  = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; //
    roll
34.           *yaw   = atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3;   //yaw
35.       }else return 2;
36.       return 0;
37.   }
```

函数内将获取的原始数据通过前面介绍 DMP 的计算公式获取欧拉角。

4.4 上传解算后的数据函数

通过 DMP 解算后，我们就可以将这些数据上传到上位机软件了，这里我们将解算后的数据以及原始数据通过串口 1 上传，具体代码如下：

```
1.    //串口 1 发送 1 个字符
2.    //c:要发送的字符
3.    void usart1_send_char(u8 c)
4.    {
5.        while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
6.        USART_SendData(USART1,c);
7.
8.    }
9.
10.   //传送数据给匿名四轴上位机软件(V2.6 版本)
11.   //fun:功能字. 0XA0~0XAF
```

```
//data:数据缓存区,最多 28 字节!!
//len:data 区有效数据个数
void usart1_niming_report(u8 fun,u8*data,u8 len)
{
    u8 send_buf[32];
    u8 i;
    if(len>28)return;    //最多 28 字节数据
    send_buf[len+3]=0;  //校验数置零
    send_buf[0]=0X88;   //帧头
    send_buf[1]=fun;    //功能字
    send_buf[2]=len;    //数据长度
    for(i=0;i<len;i++)send_buf[3+i]=data[i];          //复制数据
    for(i=0;i<len+3;i++)send_buf[len+3]+=send_buf[i];    //计算校验和
    for(i=0;i<len+4;i++)usart1_send_char(send_buf[i]);   //发送数据到串口 1
}

//发送加速度传感器数据和陀螺仪数据
//aacx,aacy,aacz:x,y,z 三个方向上面的加速度值
//gyrox,gyroy,gyroz:x,y,z 三个方向上面的陀螺仪值
void mpu6050_send_data(short aacx,short aacy,short aacz,short gyrox,short gyroy,short gyroz)

{
    u8 tbuf[12];
    tbuf[0]=(aacx>>8)&0XFF;
    tbuf[1]=aacx&0XFF;
    tbuf[2]=(aacy>>8)&0XFF;
    tbuf[3]=aacy&0XFF;
    tbuf[4]=(aacz>>8)&0XFF;
    tbuf[5]=aacz&0XFF;
    tbuf[6]=(gyrox>>8)&0XFF;
    tbuf[7]=gyrox&0XFF;
    tbuf[8]=(gyroy>>8)&0XFF;
    tbuf[9]=gyroy&0XFF;
    tbuf[10]=(gyroz>>8)&0XFF;
    tbuf[11]=gyroz&0XFF;
    usart1_niming_report(0XA1,tbuf,12);//自定义帧,0XA1
}

//通过串口 1 上报结算后的姿态数据给电脑
//aacx,aacy,aacz:x,y,z 三个方向上面的加速度值
//gyrox,gyroy,gyroz:x,y,z 三个方向上面的陀螺仪值
//roll:横滚角.单位 0.01 度。  -18000 -> 18000 对应 -180.00  ->  180.00 度
//pitch:俯仰角.单位 0.01 度。-9000 - 9000 对应 -90.00 -> 90.00 度
//yaw:航向角.单位为 0.1 度 0 -> 3600  对应 0 -> 360.0 度
```

```
55.  void usart1_report_imu(short aacx,short aacy,short aacz,short gyrox,short gyroy,short gyroz,
     short roll,short pitch,short yaw)
56.  {
57.      u8 tbuf[28];
58.      u8 i;
59.      for(i=0;i<28;i++)tbuf[i]=0;//清 0
60.      tbuf[0]=(aacx>>8)&0XFF;
61.      tbuf[1]=aacx&0XFF;
62.      tbuf[2]=(aacy>>8)&0XFF;
63.      tbuf[3]=aacy&0XFF;
64.      tbuf[4]=(aacz>>8)&0XFF;
65.      tbuf[5]=aacz&0XFF;
66.      tbuf[6]=(gyrox>>8)&0XFF;
67.      tbuf[7]=gyrox&0XFF;
68.      tbuf[8]=(gyroy>>8)&0XFF;
69.      tbuf[9]=gyroy&0XFF;
70.      tbuf[10]=(gyroz>>8)&0XFF;
71.      tbuf[11]=gyroz&0XFF;
72.      tbuf[18]=(roll>>8)&0XFF;
73.      tbuf[19]=roll&0XFF;
74.      tbuf[20]=(pitch>>8)&0XFF;
75.      tbuf[21]=pitch&0XFF;
76.      tbuf[22]=(yaw>>8)&0XFF;
77.      tbuf[23]=yaw&0XFF;
78.      usart1_niming_report(0XAF,tbuf,28);//飞控显示帧,0XAF
79.  }
```

usart1_niming_report 函数用于将数据打包、计算校验和，然后上报给匿名四轴上位机软件。mpu6050_send_data 函数用于上报加速度和陀螺仪的原始数据，可用于波形显示传感器数据，通过 A1 自定义帧发送。而 usart1_report_imu 函数，则用于上报飞控显示帧，可以实时 3D 显示 MPU6050 的姿态，传感器数据等。

4.5 主函数

编写好 MPU6050 初始化、DMP 解算、上传 DMP 解算数据函数后，接下来就可以编写主函数了，代码如下：

```
1.  #include "system.h"
2.  #include "SysTick.h"
3.  #include "led.h"
4.  #include "usart.h"
```

```c
#include "key.h"
#include "mpu6050.h"
#include "inv_mpu.h"
#include "inv_mpu_dmp_motion_driver.h"


//串口 1 发送 1 个字符
//c:要发送的字符
void usart1_send_char(u8 c)
{
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)==RESET);
    USART_SendData(USART1,c);

}

//传送数据给匿名四轴上位机软件(V2.6 版本)
//fun:功能字. 0XA0~0XAF
//data:数据缓存区,最多 28 字节!!
//len:data 区有效数据个数
void usart1_niming_report(u8 fun,u8*data,u8 len)
{
    u8 send_buf[32];
    u8 i;
    if(len>28)return;    //最多 28 字节数据
    send_buf[len+3]=0;  //校验数置零
    send_buf[0]=0X88;   //帧头
    send_buf[1]=fun;    //功能字
    send_buf[2]=len;    //数据长度
    for(i=0;i<len;i++)send_buf[3+i]=data[i];          //复制数据
    for(i=0;i<len+3;i++)send_buf[len+3]+=send_buf[i];    //计算校验和
    for(i=0;i<len+4;i++)usart1_send_char(send_buf[i]);   //发送数据到串口 1
}

//发送加速度传感器数据和陀螺仪数据
//aacx,aacy,aacz:x,y,z 三个方向上面的加速度值
//gyrox,gyroy,gyroz:x,y,z 三个方向上面的陀螺仪值
void mpu6050_send_data(short aacx,short aacy,short aacz,short gyrox,short gyroy,short gyroz)

{
    u8 tbuf[12];
    tbuf[0]=(aacx>>8)&0XFF;
    tbuf[1]=aacx&0XFF;
    tbuf[2]=(aacy>>8)&0XFF;
    tbuf[3]=aacy&0XFF;
```

```
        tbuf[4]=(aacz>>8)&0XFF;
        tbuf[5]=aacz&0XFF;
        tbuf[6]=(gyrox>>8)&0XFF;
        tbuf[7]=gyrox&0XFF;
        tbuf[8]=(gyroy>>8)&0XFF;
        tbuf[9]=gyroy&0XFF;
        tbuf[10]=(gyroz>>8)&0XFF;
        tbuf[11]=gyroz&0XFF;
        usart1_niming_report(0XA1,tbuf,12);//自定义帧,0XA1
    }

    //通过串口 1 上报结算后的姿态数据给电脑
    //aacx,aacy,aacz:x,y,z 三个方向上面的加速度值
    //gyrox,gyroy,gyroz:x,y,z 三个方向上面的陀螺仪值
    //roll:横滚角.单位 0.01 度。 -18000 -> 18000 对应 -180.00  ->  180.00 度
    //pitch:俯仰角.单位 0.01 度。-9000 - 9000 对应 -90.00 -> 90.00 度
    //yaw:航向角.单位为 0.1 度 0 -> 3600  对应 0 -> 360.0 度
    void usart1_report_imu(short aacx,short aacy,short aacz,short gyrox,short gyroy,short gyroz,
short roll,short pitch,short yaw)
    {
        u8 tbuf[28];
        u8 i;
        for(i=0;i<28;i++)tbuf[i]=0;//清 0
        tbuf[0]=(aacx>>8)&0XFF;
        tbuf[1]=aacx&0XFF;
        tbuf[2]=(aacy>>8)&0XFF;
        tbuf[3]=aacy&0XFF;
        tbuf[4]=(aacz>>8)&0XFF;
        tbuf[5]=aacz&0XFF;
        tbuf[6]=(gyrox>>8)&0XFF;
        tbuf[7]=gyrox&0XFF;
        tbuf[8]=(gyroy>>8)&0XFF;
        tbuf[9]=gyroy&0XFF;
        tbuf[10]=(gyroz>>8)&0XFF;
        tbuf[11]=gyroz&0XFF;
        tbuf[18]=(roll>>8)&0XFF;
        tbuf[19]=roll&0XFF;
        tbuf[20]=(pitch>>8)&0XFF;
        tbuf[21]=pitch&0XFF;
        tbuf[22]=(yaw>>8)&0XFF;
        tbuf[23]=yaw&0XFF;
        usart1_niming_report(0XAF,tbuf,28);//飞控显示帧,0XAF
    }

```

```
91.
92.
93.
94.  int main()
95.  {
96.      u8 i=0;
97.      u8 key;
98.      u8 report=0;
99.      float pitch,roll,yaw;       //欧拉角
100.     short aacx,aacy,aacz;       //加速度传感器原始数据
101.     short gyrox,gyroy,gyroz;    //陀螺仪原始数据
102.     short temp;                 //温度
103.
104.     SysTick_Init(168);
105.     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  //中断优先级分组 分 2 组
106.     LED_Init();
107.     USART1_Init(256000);
108.     KEY_Init();
109.     MPU6050_Init();         //初始化 MPU6050
110.
111.     printf("MPU6050 Test\r\n");
112.     while(mpu_dmp_init())
113.     {
114.         printf("MPU6050 Error!\r\n");
115.         delay_ms(500);
116.     }
117.     printf("MPU6050 OK!\r\n");
118.     printf("K_UP:UPLOAD ON/OFF\r\n");
119.
120.     while(1)
121.     {
122.         key=KEY_Scan(0);
123.         if(key==KEY_UP_PRESS)
124.         {
125.             report=!report;
126.             if(report)printf("UPLOAD ON\r\n");
127.             else printf("UPLOAD OFF\r\n");
128.         }
129.
130.         if(mpu_dmp_get_data(&pitch,&roll,&yaw)==0)
131.         {
132.             temp=MPU6050_Get_Temperature(); //得到温度值
133.             MPU6050_Get_Accelerometer(&aacx,&aacy,&aacz);    //得到加速度传感器数据
134.             MPU6050_Get_Gyroscope(&gyrox,&gyroy,&gyroz);     //得到陀螺仪数据
```

```
135.                if(report)mpu6050_send_data(aacx,aacy,aacz,gyrox,gyroy,gyroz);//用自定义帧发送加速度和陀螺仪原始数据
136.                if(report)usart1_report_imu(aacx,aacy,aacz,gyrox,gyroy,gyroz,(int)(roll*100),(int)(pitch*100),(int)(yaw*10));
137.            if((i%10)==0)
138.            {
139.                printf("检测温度为：%d 度\r\n",temp/100);
140.                temp=pitch*10;
141.                printf("Pitch：%d\r\n",temp);
142.                temp=roll*10;
143.                printf("Roll：%d\r\n",temp);
144.                temp=yaw*10;
145.                printf("Yaw：%d\r\n",temp);
146.                LED1=!LED1;
147.            }
148.        }
149.        i++;
150.    }
151. }
```

主函数实现的功能很简单，首先调用之前编写好的硬件初始化函数，包括 SysTick 系统时钟，LED 初始化等。然后调用我们前面编写的 MPU6050_Init 函数，然后再调用 mpu_dmp_init 函数用来初始化传感器内部的 DMP，并且不断循环检测初始化是否成功，如果失败，串口助手会显示“MPU6050 Error!”，如果成功就会接着往下执行，串口助手显示"MPU6050 OK!"。然后进入 while 循环，调用 mpu_dmp_get_data 函数读取解算后的欧拉角，并且读取传感器的原始数据和温度数据，通过 KEY_UP 键可以控制解算后的数据和原始数据是否上传，可通过匿名四轴上位机软件查看，同时将解算后的欧拉角及温度数据显示在串口助手上。并且 DS0 指示灯不断闪烁，提示系统正常运行。

这里要提醒下大家：为了能高速上传数据，这里我们将串口 1 的波特率设置为 256000bps，使用上位机软件测试的时候也要改成这个波特率值。

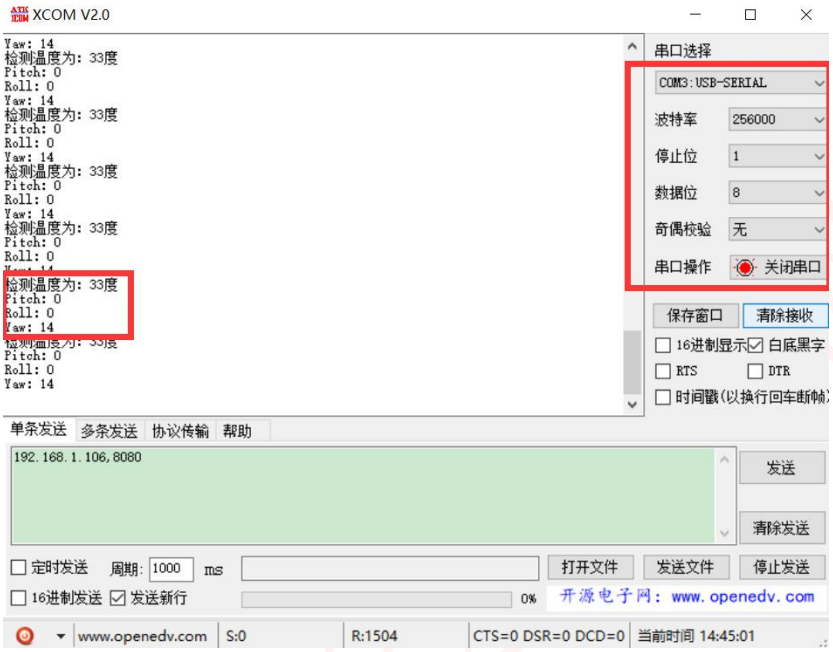
5 实验现象

首先，请先确保硬件都已经连接好：

①PZ-MPU6050 模块与 T100&T200 开发板连接（可参考硬件设计小节）

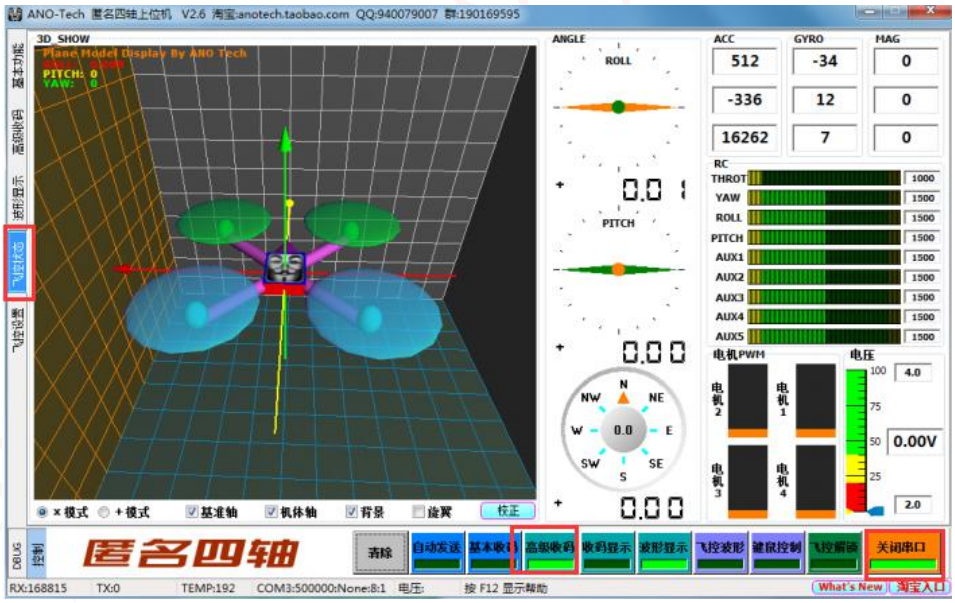
②使用 USB 线给开发板供电，并且通过串口助手查看输出信息。

将工程程序编译后下载到开发板内，可以看到 DS0 指示灯不断闪烁，表示程序正常运行。打开串口调试助手可以看到串口助手显示解算后的欧拉角及温度数据值。如下图所示：



实验说明：串口助手显示了 MPU6050 的温度、俯仰角（pitch）、横滚角（roll）和航向角（yaw）的数值。我们可以晃动开发板，看看各角度的变化。同时我们可以通过 KEY_UP 开启数据上传功能，然后打开“\软件工具\匿名四轴上位机”软件（该软件双击后，会弹出一个蓝色的小界面，直接关闭后即可进入主界面）即可观察 3D 姿态图形。**注意：一定要将串口波特率设置为程序中的 256000bps 值**，然后打开串口如下图所示：

串口波特率设置好后，我们就可以切换到“飞控状态”界面，选择“高级接收码”，这时我们摆动开发板就可以在 3D 姿态图形上观察姿态变化。如下图所示：



6 其他

（1）购买地址（普中授权店铺）

http://www.prechin.net/forum.php?mod=viewthread&tid=38746&extra=

（2）资料下载

http://prechin.net/forum.php?mod=viewthread&tid=35264&extra=page%3D1

（3）技术支持

普中官网：www.prechin.cn

普中论坛：www.prechin.net

技术电话：0755-36564227（转技术）