



开源共享

携手共进

深圳普中科技有限公司

官方网站: www.prechin.cn

技术论坛: www.prechin.net

技术 QQ: 2489019400

咨询电话: 0755-61139052

PZ-MPU6050 六轴传感器模块开发手册

本手册教大家学习当下最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。通过本手册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 TFTLCD 显示。学习本章可以参考模块参考资料。本章分为如下几部分内容：

- 1 MPU6050 传感器介绍
- 2 利用 DMP 进行姿态解算
- 3 硬件设计
- 4 软件设计
- 5 实验现象

普中科技STM32开发板

1 MPU6050 传感器介绍

1.1 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%

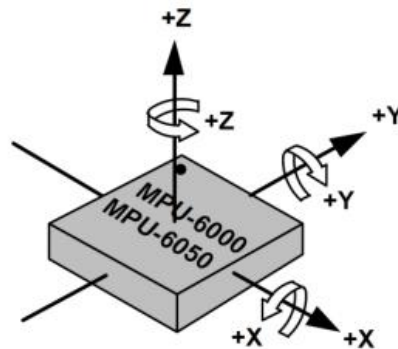
11. 陀螺仪工作电流：5mA，陀螺仪待机电流：5uA；加速器工作电流：500uA，
加速器省电模式电流：40uA@10Hz

12. 自带 1024 字节 FIFO，有助于降低系统功耗

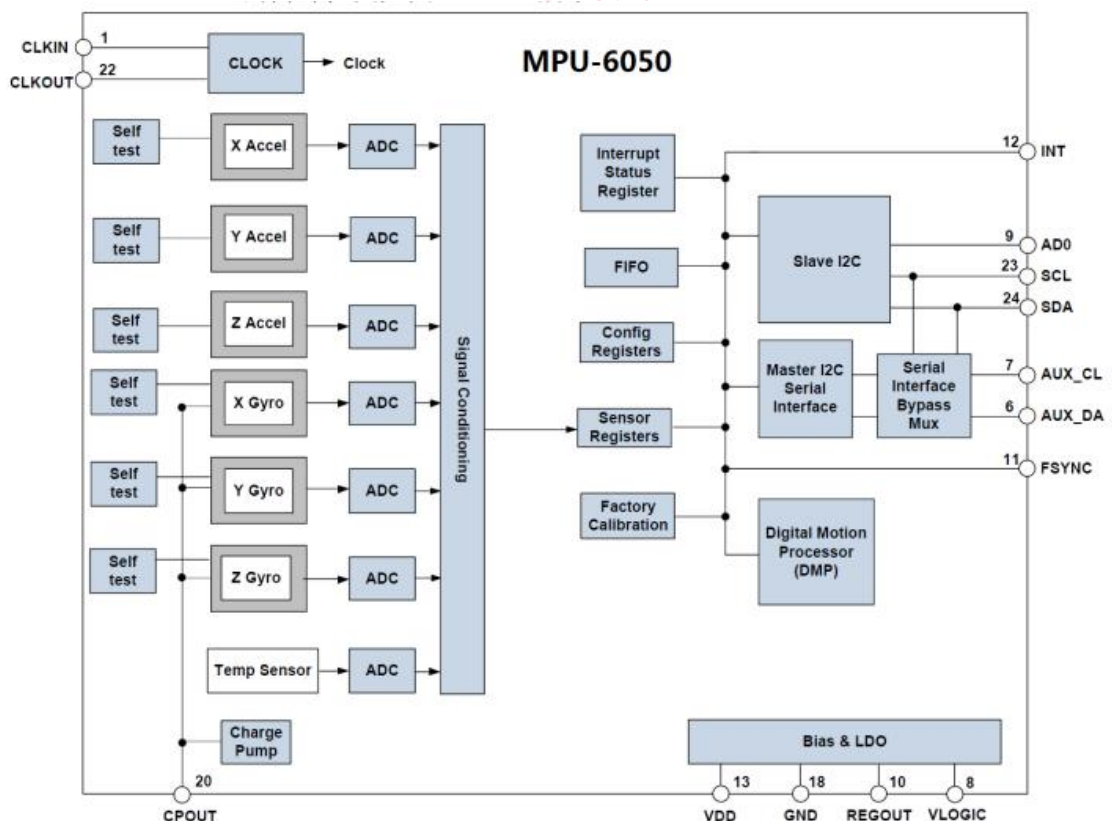
13. 高达 400Khz 的 IIC 通信接口

14. 超小封装尺寸：4x4x0.9mm（QFN）

MPU6050 传感器的检测轴如图所示：



MPU6050 的内部框图如图所示：



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）。

本教程我们将 AD0 是接 GND 的，所以 MPU6050 的 IIC 地址是 0X68（不含最低位），IIC 通信的时序我们在开发攻略文档中已经介绍过，这里就不再多说了。

1.2 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 STM32 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 STM32 通信，所以我们需要先初始化与 MPU6050 连接的 SDA 和 SCL 数据线。

（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。SLEEP 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， $\pm 250^{\circ}/S$ ；1， $\pm 500^{\circ}/S$ ；2， $\pm 1000^{\circ}/S$ ；3， $\pm 2000^{\circ}/S$ ；我们一般设置为 3，即 $\pm 2000^{\circ}/S$ ，因为陀螺仪的 ADC 为 16 位分辨率，所以得到灵敏度为： $65536/4000=16.4LSB/(^{\circ}/S)$ 。

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， $\pm 2g$ ；1， $\pm 4g$ ；2， $\pm 8g$ ；3， $\pm 16g$ ；我们一般设置为 0，即 $\pm 2g$ ，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	SAMPLRT_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：

$$\text{采样频率} = \text{陀螺仪输出频率} / (1 + \text{SMPLRT_DIV})$$

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：

$$\text{Temperature} = 36.53 + \text{regval}/340$$

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，请参考模块资料。MPU6050 涉及的知识非常多，所以要耐心学习。

2 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：

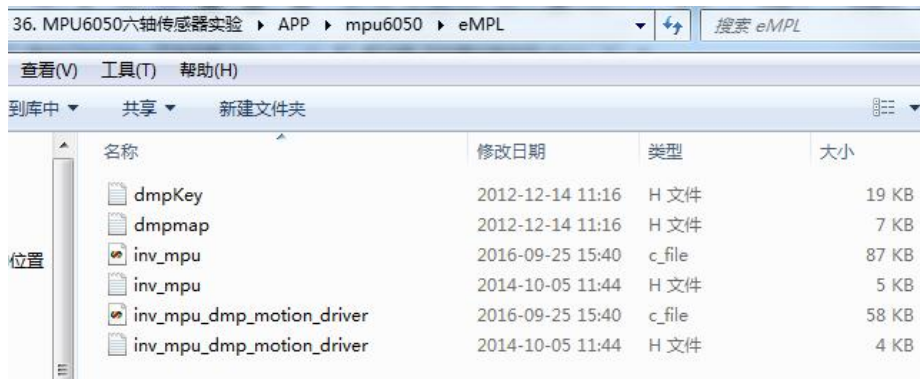
```
q0=quat[0] / q30; //q30 格式转换为浮点数
q1=quat[1] / q30;
q2=quat[2] / q30;
q3=quat[3] / q30;
//计算得到俯仰角/横滚角/航向角
pitch=asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; //俯仰角
roll=atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)*
57.3; //横滚角
yaw=atan2(2*(q1*q2 + q0*q3), q0*q0+q1*q1-q2*q2-q3*q3) * 57.3; //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 $180/\pi$ ，这样得到的结果就是以度($^{\circ}$)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 STM32F1 上面，官方原版驱动在模块资料“\MPU6050 参考资料\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：



名称	修改日期	类型	大小
dmpKey	2012-12-14 11:16	H 文件	19 KB
dmpmap	2012-12-14 11:16	H 文件	7 KB
inv_mpu	2016-09-25 15:40	c_file	87 KB
inv_mpu	2014-10-05 11:44	H 文件	5 KB
inv_mpu_dmp_motion_driver	2016-09-25 15:40	c_file	58 KB
inv_mpu_dmp_motion_driver	2014-10-05 11:44	H 文件	4 KB

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_ori
entation)); //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_SEN
D_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_stm32f40_41xxx.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。

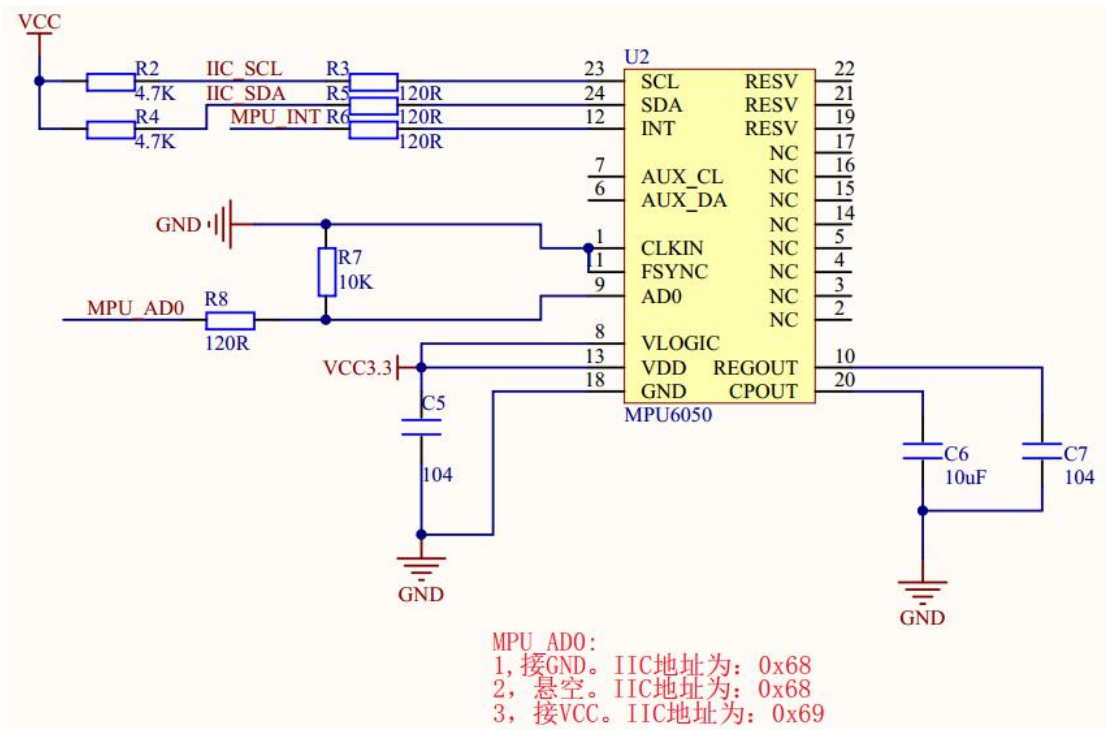
利用这两个函数，我们就可以读取到姿态解算后的欧拉角，使用非常方便。

3 硬件设计

本实验使用到硬件资源如下：

- (1) D1 指示灯
- (2) K_UP 按键
- (3) 串口 1

(5) PZ-MPU6050 六轴传感器模块

[illegible]

PZ-MPU6050 模块与 STM32F1 开发板的连接管脚如图所示:

PZ-MPU6050模块	普中STM32F1开发板 (IO)
MPU_ADO	GND
MPU_INT	悬空
IIC_SCL	PB10
IIC_SDA	PB11
GND	GND
VCC	5V

D1 指示灯用来提示系统运行状态，K_UP 按键用来控制数据（加速度传感器、陀螺仪、DMP 姿态解算后的欧拉角等数据）上传功能，如果开启上传数据功能，这些数据将通过串口 1 发送给上位机，使用“\PZ-MPU6050 六轴传感器模块\调试工具\匿名四轴上位机” ANO_Tech 匿名四轴上位机_V2.6.exe 软件即可观察 3D 姿态变化。同时 TFTLCD 模块也可显示温度及欧拉角等信息。

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 K_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“PZ-MPU6050 六轴传感器模块程序\PZ-MPU6050 六轴传感器模块—STM32F1 程序”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:

```
//初始化 MPU6050
//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, bu
f);
    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();
    IIC_Send_Byte((addr<<1)|1); //发送器件地址+读命令
    IIC_Wait_Ack();    //等待应答
    while(len)

```

```

    {
        if(len==1)*buf=IIC_Read_Byte(0); //读数据, 发送 nACK
        else *buf=IIC_Read_Byte(1);      //读数据, 发送 ACK
        len--;
        buf++;
    }

    IIC_Stop(); //产生一个停止条件

    return 0;
}

```

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 MPU6050 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, 即最终需要求得欧拉角。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:

```

//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;
}

```

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：

```

//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
}

```

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：

```

int main()
{
    u8 i=0;
    u8 key;
    u8 report=1;

    float pitch,roll,yaw;           //欧拉角
    short aacx,aacy,aacz;           //加速度传感器原始数据
    short gyrox,gyroy,gyroz;        //陀螺仪原始数据
    short temp;                     //温度
    u8 res;

    SysTick_Init(72);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //中断优先级

```

分组 分2组

```
LED_Init();
USART1_Init(256000);
TFTLCD_Init();           //LCD 初始化
KEY_Init();
MPU6050_Init();           //初始化 MPU6050

FRONT_COLOR=BLACK;
LCD_ShowString(10, 10, tftlcd_data.width, tftlcd_data.height, 16, "
PRECHIN STM32F1");
LCD_ShowString(10, 30, tftlcd_data.width, tftlcd_data.height, 16, "
www.prechin.net");
LCD_ShowString(10, 50, tftlcd_data.width, tftlcd_data.height, 16, "
MPU6050 Test");

FRONT_COLOR=RED;
res=mpu_dmp_init();
printf("res=%d\r\n", res);
while(mpu_dmp_init())
{
    printf("MPU6050 Error!\r\n");

    LCD_ShowString(10, 130, tftlcd_data.width, tftlcd_data.height, 16, "MP
U6050 Error!");
    delay_ms(200);
}
printf("MPU6050 OK!\r\n");
LCD_ShowString(10, 130, tftlcd_data.width, tftlcd_data.height, 16,
"MPU6050 OK!");
```



```
LCD_ShowString(10, 150, tftlcd_data.width, tftlcd_data.height, 16, "K_UP:UPLOAD ON/OFF");
```

```
POINT_COLOR=BLUE;//设置字体为蓝色
```

```
LCD_ShowString(10, 170, tftlcd_data.width, tftlcd_data.height, 16, "UPLOAD ON ");
```

```
LCD_ShowString(10, 200, tftlcd_data.width, tftlcd_data.height, 16, "Temp:    . C");
```

```
LCD_ShowString(10, 220, tftlcd_data.width, tftlcd_data.height, 16, "Pitch:    . C");
```

```
LCD_ShowString(10, 240, tftlcd_data.width, tftlcd_data.height, 16, "Roll:    . C");
```

```
LCD_ShowString(10, 260, tftlcd_data.width, tftlcd_data.height, 16, "Yaw :    . C");
```

```
while(1)
{
    key=KEY_Scan(0);
    if(key==KEY_UP)
    {
        report=!report;
```

```
        if(report)LCD_ShowString(10, 170, tftlcd_data.width, tftlcd_data.height, 16, "UPLOAD ON ");
        else
```

```
LCD_ShowString(10, 170, tftlcd_data.width, tftlcd_data.height, 16, "UPLOAD  
OFF");
```

```
}
```

```
if(mpu_dmp_get_data(&pitch,&roll,&yaw)==0)
```

```
{
```

```
temp=MPU6050_Get_Temperature(); //得到温度值
```

```
MPU6050_Get_Accelerometer(&aacx,&aacy,&aacz); //得到加  
速度传感器数据
```

```
MPU6050_Get_Gyroscope(&gyrox,&gyroy,&gyroz); //得到陀  
螺仪数据
```

```
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
```

```
if(report)usart1_report_imu(aacx, aacy, aacz, gyrox, gyroy, gyroz, (int)  
(roll*100), (int)(pitch*100), (int)(yaw*10));
```

```
if((i%10)==0)
```

```
{
```

```
if(temp<0)
```

```
{
```

```
LCD_ShowChar(10+48, 200, '-', 16, 0); //显示负  
号
```

```
temp=-temp; //转为正数
```

```
}else LCD_ShowChar(10+48, 200, ' ', 16, 0); //去掉  
负号
```

```
printf("检测温度为: %d 度\r\n", temp);
```

```
LCD_ShowNum(10+48+8, 200, temp/100, 3, 16); //显示  
整数部分
```

```

LCD_ShowNum(10+48+40, 200, temp%10, 1, 16);          //显示
小数部分

temp=pitch*10;
if(temp<0)
{
    LCD_ShowChar(10+48, 220, '-', 16, 0);          //显示负
号

    temp=-temp;          //转为正数
}else LCD_ShowChar(10+48, 220, ' ', 16, 0);          //去掉
负号

printf("Pitch: %d\r\n", temp);
LCD_ShowNum(10+48+8, 220, temp/10, 3, 16);          //显示整
数部分

LCD_ShowNum(10+48+40, 220, temp%10, 1, 16);          //显示
小数部分

temp=roll*10;
if(temp<0)
{
    LCD_ShowChar(10+48, 240, '-', 16, 0);          //显示负
号

    temp=-temp;          //转为正数
}else LCD_ShowChar(10+48, 240, ' ', 16, 0);          //去掉
负号

printf("Roll: %d\r\n", temp);
LCD_ShowNum(10+48+8, 240, temp/10, 3, 16);          //显示整
数部分

LCD_ShowNum(10+48+40, 240, temp%10, 1, 16);          //显示
小数部分

temp=yaw*10;

```

```

        if(temp<0)
        {
            LCD_ShowChar(10+48, 260, '-', 16, 0);    // 显示负
号
            temp=-temp;    //转为正数
        }else LCD_ShowChar(10+48, 260, ' ', 16, 0);    // 去掉
负号

        printf("Yaw: %d\r\n", temp);
        LCD_ShowNum(10+48+8, 260, temp/10, 3, 16);    // 显示整
数部分

        LCD_ShowNum(10+48+40, 260, temp%10, 1, 16);    //显示
小数部分

        led1=!led1;
    }
}
i++;
}
}

```

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，中断分组，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，TFTLCD 会显示“MPU6050 Error!”，如果成功就会接着往下执行，TFTLCD 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 K_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 TFTLCD 上。并且 D1 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

将工程程序编译后下载到开发板内，可以看到 D1 指示灯不断闪烁，表示程序正常运行。同时在 TFTLCD 模块上可以看到解算后的欧拉角及温度数据值。如图所示：



实验说明：屏幕显示了 MPU6050 的温度、俯仰角（pitch）、横滚角（roll）和航向角（yaw）的数值。我们可以晃动模块，看看各角度的变化。同时我们可以通过 K_UP 开启数据上传功能，然后打开“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。如图所示：

