AN1807 ATK-MPU6050 六轴传感器模块使用说明

本应用文档（AN1807，对应 ALIENTEK NANO STM32F103 开发板扩展实验 6）将教大家如何在 ALIENTEK NANO STM32F103 开发板上使用 ATK-MPU6050 六轴传感器模块。

本文档分为如下几部分：

1， MPU6050 简介
2， 硬件连接
3， 软件实现
4， 验证

1、MPU6050 简介

本节，我们将分 2 个部分介绍：1，MPU6050 基础介绍。2，DMP 使用简介。

1.1 MPU6050 基础介绍

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 1.1.1 所示：

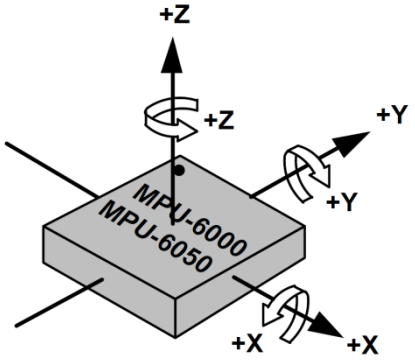

图 1.1.1 MPU6050 检测轴及其方向

MPU6050 的内部框图如图 1.1.2 所示：

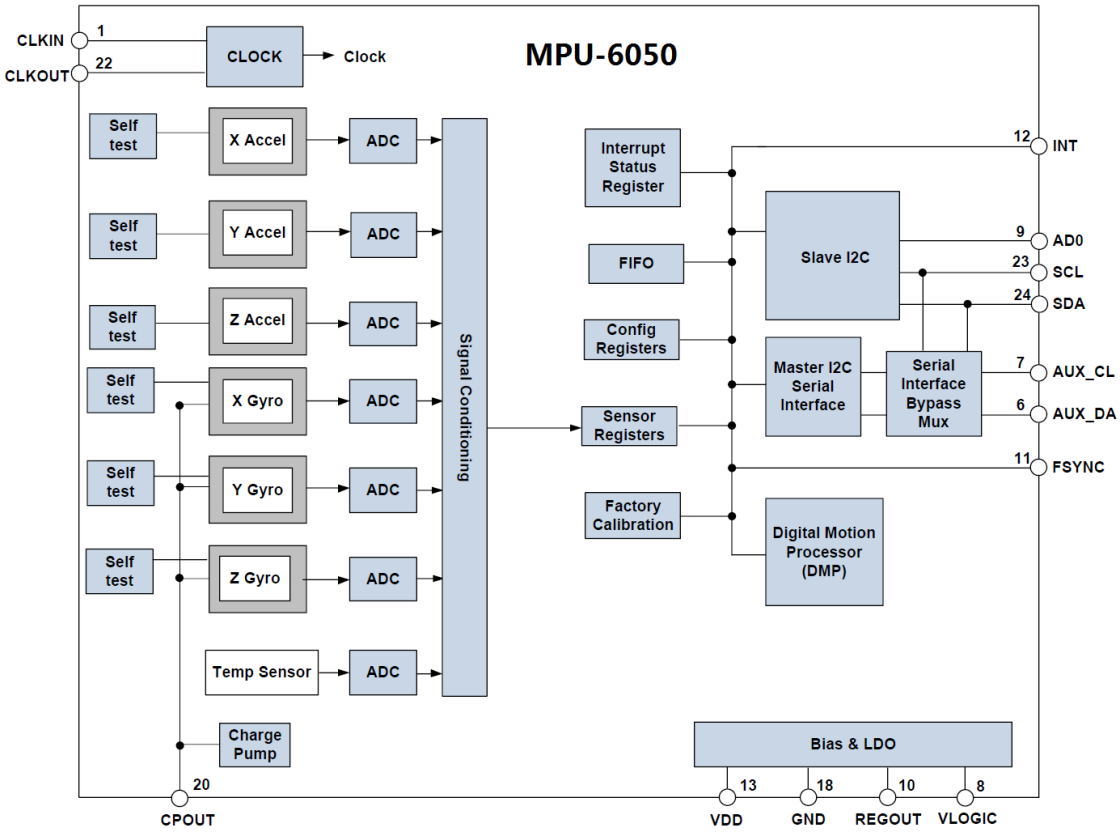

图 1.1.2 MPU6050 框图

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）！！

接下来，我们介绍一下利用 STM32F1 读取 MPU6050 的加速度和角度传感器数据（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 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。

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（0X1B）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，各位描述如图 1.1.3 所示：

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

图 1.1.3 电源管理寄存器 1 各位描述

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，选择关系如表 1.1.1 所示：

CLKSEL[2:0]	时钟源
000	内部 8M RC 晶振
001	PLL，使用 X 轴陀螺作为参考
010	PLL，使用 Y 轴陀螺作为参考
011	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

图 1.1.1 CLKSEL 选择列表

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，各位描述如图 1.1.4 所示：

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

图 1.1.4 陀螺仪配置寄存器各位描述

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，各位描述如图 1.1.5 所示：

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

图 1.1.5 加速度传感器配置寄存器各位描述

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：0X1C，各位描述如图 1.1.6 所示：

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

图 1.1.6 FIFO 使能寄存器各位描述

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，各位描述如图 1.1.7 所示：

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

图 1.1.7 陀螺仪采样率分频寄存器各位描述

该寄存器用于设置 MPU6050 的陀螺仪采样频率，计算公式为：

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，各位描述如图 1.1.8 所示：

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

图 1.1.8 配置寄存器各位描述

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表 1.1.2 所示：

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

图 1.1.2 DLPF_CFG 配置表

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，各位描述如图 1.1.9 所示：

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

图 1.1.9 电源管理寄存器 2 各位描述

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 8 个寄存器组成，地址为：0X43~0X48，通过读取这 8 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 8 个，地址为：0X3B~0X40，通过读取这 8 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 的基础介绍，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：4，MPU6050 参考资料→ MPU-6000 and MPU-6050 Product

Specification.pdf 和 MPU-6000 and MPU-6050 Register Map and Descriptions.pdf 这两个文档，另外该目录还提供了部分 MPU6050 的中文资料，供大家参考学习。

1.2 DMP 使用简介

经过 1.1 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 的 30 次方倍。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 的 30 次方，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 的 30 次方，其中 q30 是一个常量：1073741824，即 2 的 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，我们需要将其移植一下，才可以用到 STM32F1 上面，官方原版驱动在光盘：6，硬件资料→MPU6050 资料→DMP 资料→Embedded_MotionDriver_5.1.rar，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 Driver V5.1.1 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，我们放在本例程→HARDWARE→MPU6050→eMPL 文件夹内，总共 6 个文件，如图 1.2.1 所示：

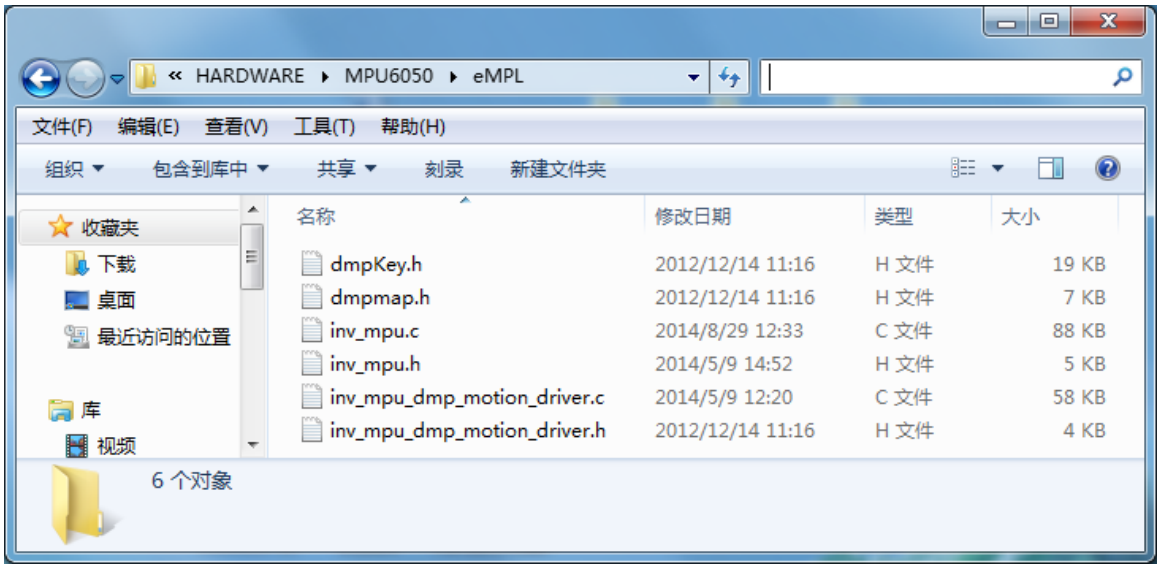

图 1.2.1 移植后的驱动库代码

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);//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
        |DMP_FEATURE_ANDROID_ORIENT|DMP_FEATURE_SEND_RAW_ACCEL
        |DMP_FEATURE_SEND_CAL_GYRO|DMP_FEATURE_GYRO_CAL);
        //设置 dmp 功能
        if(res)return 6;
```

```
            res=dmp_set_fifo_rate(DEFAULT_MPU_HZ);//设置 DMP 输出速率(最大 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系列操作，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;
        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_stm32f103xb.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。DMP 部分，我们就介绍到这。

2、硬件连接

采用 NANO STM32F103 开发板连接 ATK-MPU6050 模块，模块共有两个实验例程，分别是串口助手打印和上位机显示。

1，串口助手打印：将读取的温度传感器、加速度传感器、陀螺仪、DMP 姿态解算后的欧拉角等数据通过串口输出在串口调试助手上。同时 DS0 闪烁，以表示程序正在运行。

2，上位机显示：将读取的加速度传感器、陀螺仪、DMP 姿态结算后的欧拉角等数据，通过串口上报给上位机，利用上位机软件（ANO_Tech 匿名四轴上位机_V2.6.exe），可以实时显示 MPU6050 的传感器状态曲线，并显示 3D 姿态。同时 DS0 闪烁，以表示程序正在运行。

所要用到的硬件资源如下：

1） 指示灯 DS0

2） 串口 1

3） KEY_UP 按键

4） ATK-MPU6050 模块

前 2 个，在之前的实例已经介绍过了，这里我们介绍下 ATK-MPU6050 模块与 NANO STM32F103 开发板的连接。ATK-MPU6050 模块原理图如图 2.1 所示：

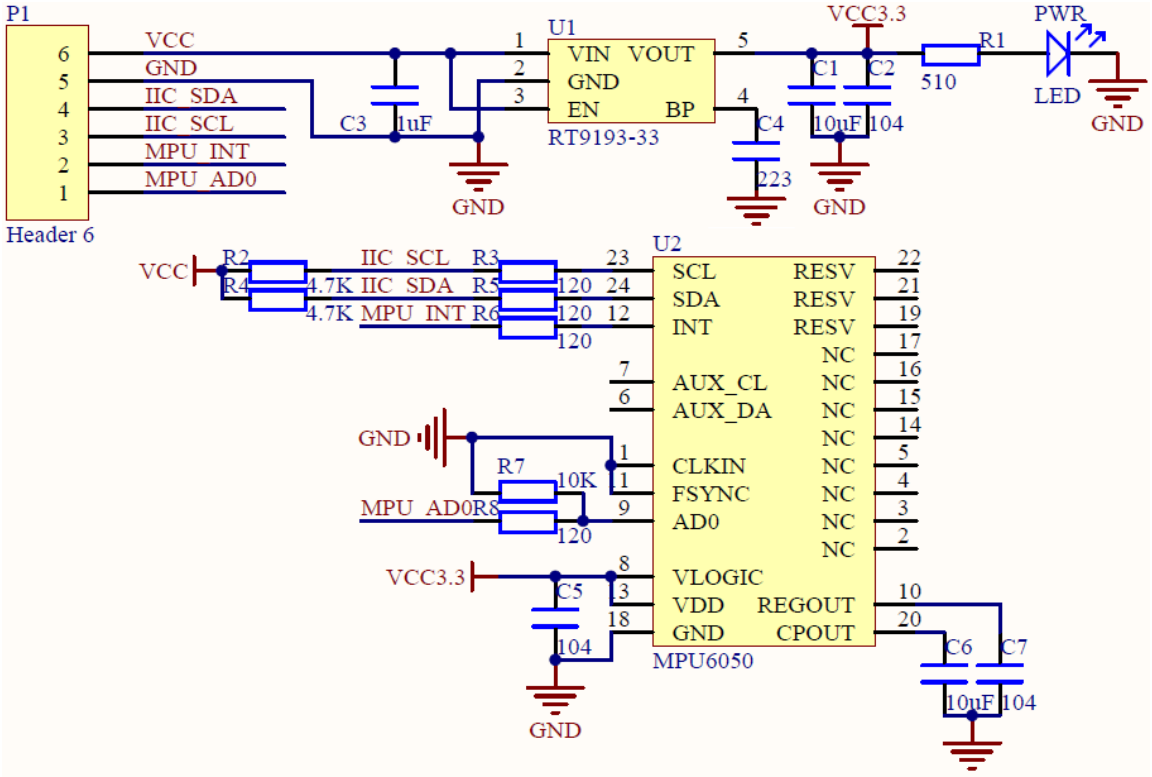

图 2.1 ATK-MPU6050 模块原理图

从上图可知，ATK-MPU6050 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）。

因为 NANO STM32 开发板并没有自带 ATK-MODULE 接口，所以，只能通过杜邦线连接开发板和 ATK-MPU6050 模块。连接关系如表 2.1 所示：

ATK-MPU6050 模块与 NANO STM32 开发板连接关系				
ATK-MPU6050 模块	VCC	GND	SDA	SCL
NANO STM32F103 开发板	3.3V/5V	GND	PA3	PA2

表 2.1 ATK-MPU6050 模块与 NANO STM32 开发板连接关系表

如表 2.1 所示，我们的 NANO STM32 开发板与 ATK-MPU6050 模块的连接，只用了 4 根线：VCC,GND,SDA,SCL。模块的 MPU_INT 和 MPU_AD0 并没有用到，所以，模块的地址为：0X68。

3、软件实现

本扩展例程（扩展实验 6 ATK-MPU6050 六轴传感器模块测试实验），我们在 NANO STM32 开发板：IIC 实验的基础上进行修改。两个例程中底层驱动都是一样，唯独是 main.c 文件上的应用不一样。下面我们分别讲解两个例程：

3.1 串口助手打印

例程中没有用到 24C02，所以，先去掉 24cxx.c 和 myiic.c。然后，在 HARDWARE 文件夹下新建一个 MPU6050 的文件夹。然后新建一个 mpu6050.c 和 mpu6050.h 的文件保存在 MPU6050 文件夹下，并将这个文件夹加入头文件包含路径。

同时，将 DMP 驱动库代码：见本例程源码：扩展实验 6 ATK-MPU6050 六轴传感器实验\HARDWARE\MPU6050\ eMPL，里面的 eMPL 文件夹，拷贝到本例程 MPU6050 文件夹里面，将 eMPL 文件夹也加入头文件包含路径，然后将 eMPL 文件夹里面的两个 c 文件：inv_mpu.c 和 inv_mpu_dmp_motion_driver.c 加入 HARDWARE 组。

由于 mpu6050.c 里面代码比较多，这里我们就不全部列出来了，仅介绍几个重要的函数。

首先是：MPU_Init，该函数代码如下：

```
//初始化 MPU6050
//返回值:0,成功
//      其他,错误代码
u8 MPU_Init(void)
{
    u8 res;
    MPU_IIC_Init();                    //初始化 IIC 总线
    MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X80);//复位 MPU6050
    delay_ms(100);
    MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X00);//唤醒 MPU6050
    MPU_Set_Gyro_Fsr(3);                    //陀螺仪传感器,±2000dps
    MPU_Set_Accel_Fsr(0);                   //加速度传感器,±2g
    MPU_Set_Rate(50);                       //设置采样率 50Hz
    MPU_Write_Byte(MPU_INT_EN_REG,0X00);    //关闭所有中断
    MPU_Write_Byte(MPU_USER_CTRL_REG,0X00); //I2C 主模式关闭
    MPU_Write_Byte(MPU_FIFO_EN_REG,0X00);   //关闭 FIFO
    MPU_Write_Byte(MPU_INTBP_CFG_REG,0X80); //INT 引脚低电平有效
```

```
        res=MPU_Read_Byte(MPU_DEVICE_ID_REG);
        if(res==MPU_ADDR)//器件 ID 正确
        {
            MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X01);
            //设置 CLKSEL,PLL X 轴为参考
            MPU_Write_Byte(MPU_PWR_MGMT2_REG,0X00);//加速度与陀螺仪都工作
            MPU_Set_Rate(50);                                    //设置采样率为 50Hz
        }else return 1;
        return 0;
    }
```

该函数就是按我们在 1.1 节介绍的方法，对 MPU6050 进行初始化，该函数执行成功后，便可以读取传感器数据了。

然后再看 MPU_Get_Temperature、MPU_Get_Gyroscope 和 MPU_Get_Accelerometer 等三个函数，源码如下：

```
    //得到温度值
    //返回值:温度值(扩大了 100 倍)
    short MPU_Get_Temperature(void)
    {
        u8 buf[2];
        short raw; float temp;
        MPU_Read_Len(MPU_ADDR,MPU_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 MPU_Get_Gyroscope(short *gx,short *gy,short *gz)
    {
        u8 buf[6],res;
        res=MPU_Read_Len(MPU_ADDR,MPU_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 MPU_Get_Accelerometer(short *ax,short *ay,short *az)
{
    u8 buf[6],res;
    res=MPU_Read_Len(MPU_ADDR,MPU_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;;
}
```

其中 MPU_Get_Temperature 用于获取 MPU6050 自带温度传感器的温度值，然后 MPU_Get_Gyroscope 和 MPU_Get_Accelerometer 分别用于读取陀螺仪和加速度传感器的原始数据。

最后看 MPU_Write_Len 和 MPU_Read_Len 这两个函数，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 MPU_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()){IIC_Stop();return 1;}//等待应答
    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()) {IIC_Stop();return 1;}//等待 ACK
    }
    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 MPU_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()){ IIC_Stop();return 1; }  //等待应答
    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;
}
```

MPU_Write_Len 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 MPU_Read_Len 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。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 delay.c 里面的 delay_ms 实现，get_ms 则直接提供一个空函数即可。

关于 mpu6050.c 我们就介绍到这，将 mpu6050.c 加入 HARDWARE 组下，另外 mpu6050.h 的代码，我们这里就不再贴出了，大家看光盘源码即可。

最后在 main.c 里面修改代码如下：

```
int main(void)
{

    u8 t=0;
    u8 key=0;
    u8 GetData=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

    HAL_Init();                             //初始化 HAL 库
    Stm32_Clock_Init(RCC_PLL_MUL9); //设置时钟,72M
    delay_init(72);                         //初始化延时函数
    uart_init(115200);                      //初始化串口 115200
```

```
LED_Init();                          //初始化 LED
KEY_Init();                          //初始化按键
usmart_init(72);                     //USMART 初始化
printf("ALIENTEK NANO STM32\r\n");
printf("MPU6050 TEST\r\n");
while(mpu_dmp_init())//MPU DMP 初始化
{
    printf("MPU6050 Error!!!\r\n");
    delay_ms(500);
    LED0=!LED0;//DS0 闪烁
}
printf("MPU6050 OK\r\n");
while(1)
{

    key=KEY_Scan(0);
    if(key==WKUP_PRES)
    {
        GetData=!GetData;
        if(GetData)printf("GETDATA ON\r\n");
        else printf("GETDATA OFF\r\n");
    }
    if(mpu_dmp_get_data(&pitch,&roll,&yaw)==0)
    {
        temp=MPU_Get_Temperature();  //得到温度值
        MPU_Get_Accelerometer(&aacx,&aacy,&aacz); //得到加速度传感器数据
        MPU_Get_Gyroscope(&gyrox,&gyroy,&gyroz); //得到陀螺仪数据
        if(GetData)//GetData=0 时 用于 USMART 调试 MPU6050 寄存器
        {

            if((t%10)==0)
            {
                //temp 值
                if(temp<0)
                {
                    temp=-temp;         //转为正数
                    printf(" Temp:   -%d.%dC\r\n",temp/100,temp%10);
                }else
                    printf(" Temp:   %d.%dC\r\n",temp/100,temp%10);

                //pitch 值
                temp=pitch*10;
                if(temp<0)
                {
```

```
                temp=-temp;          //转为正数
                printf(" Pitch: -%d.%dC\r\n",temp/10,temp%10);
            }else
                printf(" Pitch:    %d.%dC\r\n",temp/10,temp%10);

            //roll 值
            temp=roll*10;
            if(temp<0)
            {
                temp=-temp;          //转为正数
                printf(" Roll:    -%d.%dC\r\n",temp/10,temp%10);
            }else
                printf(" Roll:    %d.%dC\r\n",temp/10,temp%10);

            //yaw 值
            temp=yaw*10;
            if(temp<0)
            {
                temp=-temp;          //转为正数
                printf(" Yaw:    -%d.%dC\r\n",temp/10,temp%10);
            }else
                printf(" Yaw:     %d.%dC\r\n",temp/10,temp%10);

            printf("\r\n");
            t=0;
            LED0=!LED0;//LED 闪烁
        }

    }

    }
    t++;
  }
}
```

该文件就一个 main 函数，main 函数也非常简单，先对用到的外设（串口，按键）、MPU6050 传感器等初始化，然后将获取的温度、加速度、陀螺仪的数据输出到串口上。

最后，我们将 MPU_Write_Byte、MPU_Read_Byte 和 MPU_Get_Temperature 等三个函数加入 USMART 控制，这样，我们就可以通过串口调试助手，改写和读取 MPU6050 的寄存器数据了，并可以读取温度传感器的值，方便大家调试（注意在 USMART 调试的时候，最好通过按 KEY_UP，先关闭数据获取功能，然后再调试）。

3.2 上位机显示

该例程与串口助手打印例程底层函数一样，这里不再做讲解，下面我们说下 main.c 文件，main.c 文件代码如下：

```
//串口 1 发送 1 个字符
//c:要发送的字符
void usart1_send_char(u8 c)
{
    while((USART1->SR&0X40)==0);//等待上一次发送完毕
    USART1->DR=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
}

int main(void)
{
    u8 t=0;
    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

    HAL_Init();                     //初始化 HAL 库
```

```
    Stm32_Clock_Init(RCC_PLL_MUL9); //设置时钟,72M
    delay_init(72);                 //初始化延时函数
    uart_init(500000);              //初始化串口 500000
    LED_Init();                     //初始化 LED
    KEY_Init();                     //初始化按键
    usmart_init(72);                //USMART 初始化
    printf("ALIENTEK NANO STM32\r\n");
    printf("MPU6050 TEST\r\n");
    while(mpu_dmp_init())//MPU DMP 初始化
    {
        printf("MPU6050 Error!!!\r\n");
        delay_ms(500);
        LED0=!LED0;//DS0 闪烁
    }
    printf("MPU6050 OK\r\n");
    while(1)
    {
        key=KEY_Scan(0);
        if(key==WKUP_PRES)
        {
            GetData=!GetData;
            if(GetData)printf("GETDATA ON\r\n");
            else printf("GETDATA OFF\r\n");
        }
        if(mpu_dmp_get_data(&pitch,&roll,&yaw)==0)
        {
            MPU_Get_Accelerometer(&aacx,&aacy,&aacz); //得到加速度传感器数据
            MPU_Get_Gyroscope(&gyrox,&gyroy,&gyroz); //得到陀螺仪数据
            if(GetData)//(GetData=0 时  用于 USMART 调试 MPU6050 寄存器)
            {
                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
                usart1_report_imu(aacx,aacy,aacz,gyrox,gyroy,gyroz,
              (int)(roll*100),(int)(pitch*100),(int)(yaw*10));//发送数据到上位机
            }
            if((t%10)==0)
            {
                t=0;
                LED0=!LED0;//LED 闪烁
            }
        }
        t++;
    }
}
```

此部分代码除了 main 函数，还有几个函数，用于上报数据给上位机软件，利用上位机软件显示传感器波形，以及 3D 姿态显示，有助于更好的调试 MPU6050。上位机软件使用：ANO_Tech 匿名四轴上位机_V2.6.exe，该软件在：ATK-MPU6050 六轴传感器模块资料→ 3，配套软件→匿名四轴上位机 文件夹里面可以找到，该软件的使用方法，见该文件夹下的 README.txt，这里我们不做介绍。其中，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。

这里，main 函数是比较简单的，大家看代码即可，不过需要注意的是，为了高速上传数据，这里我们将串口 1 的波特率设置为 500Kbps 了，测试的时候要注意下。

最后，我们将 MPU_Write_Byte、MPU_Read_Byte 和 MPU_Get_Temperature 等三个函数加入 USMART 控制，这样，我们就可以通过串口调试助手，改写和读取 MPU6050 的寄存器数据了，并可以读取温度传感器的值，方便大家调试（注意在 USMART 调试的时候，最好通过按 KEY_UP，先关闭数据上传上位机，然后再调试，因为不关闭会显示一堆乱码）。

至此，我们的两个例程的软件设计部分讲解就结束了。

4、验证

4.1 串口助手打印

本例程测试需要自备 ATK-MPU6050 模块一个，并按第 2 节所示连接方式，连接模块和开发板。在代码编译成功之后，我们通过下载代码到 ALIENTEK NANO STM32F103 开发板上，MPU6050 初始化成功后，可以看到串口助手打印显示如图 4.1.1 所示的内容：

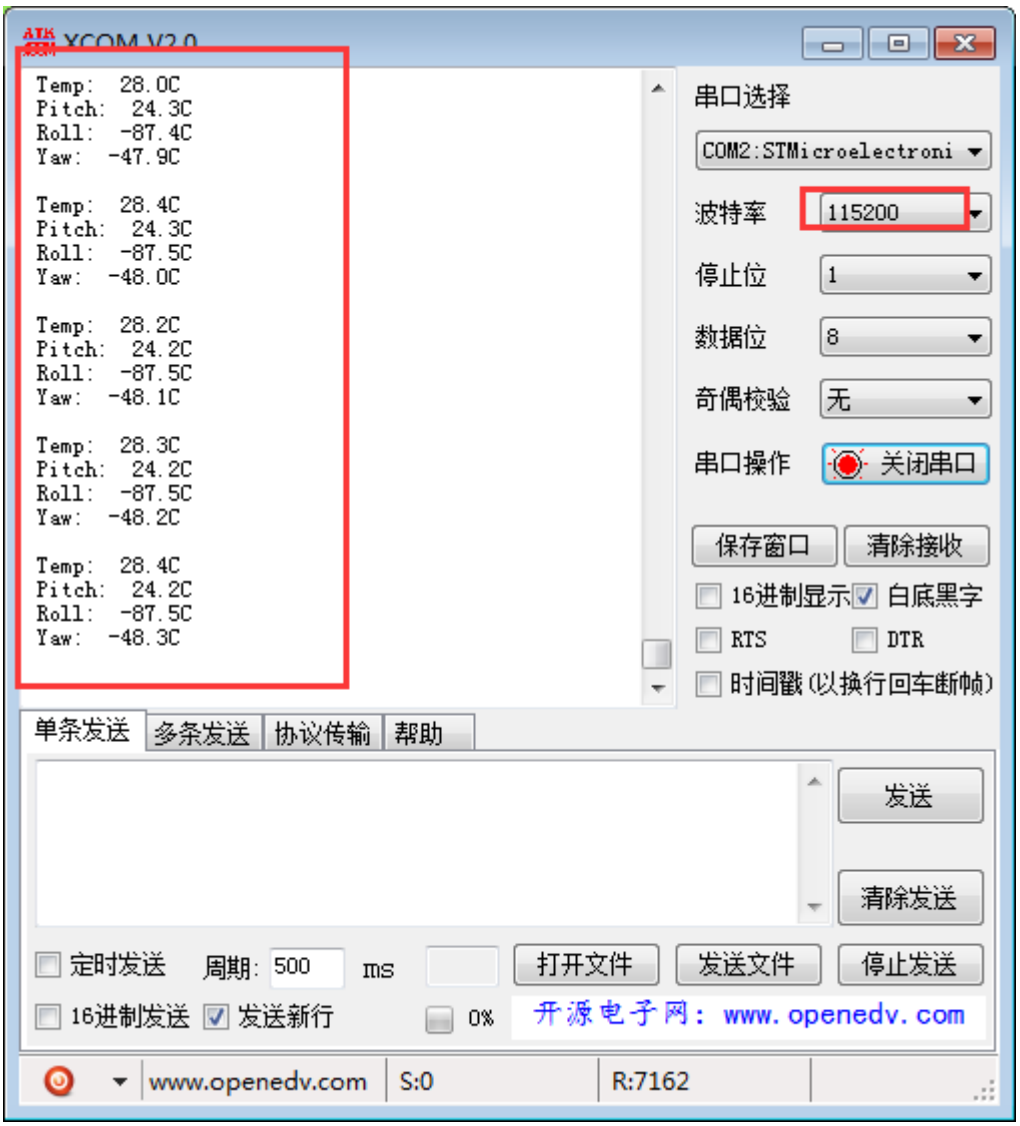

图 4.1 程序运行时数据串口打印

特别注意：因为 NANO STM32 和 MPU6050 模块是通过杜邦线连接的，容易受干扰，如果出现“MPU6050 Error”的报错：

1，当该报错闪烁比较快的时候，约 1 秒钟 1 次，则说明读取 ID 错误了，此时建议给模块断电，然后重新上电。

2，当该报错闪烁比较慢的时候，约 3 秒钟 1 次，则说明 ID 读取正常，DMP 初始化异常，此时，可以用手抓住杜邦线，将模块放平，然后稍等片刻，即可完成初始化。

串口调试助手打印显示了 MPU6050 的温度、俯仰角（pitch）、横滚角（roll）和航向角（yaw）的数值。然后，我们可以晃动 MPU6050 模块，看看各角度的变化。

同时，我们还可以用 USMART 读写 MPU6050 的任何寄存器，来调试代码，这里我们就不做演示了，大家自己测试即可。最后，建议大家用 USMART 调试的时候，先按 KEY0 关闭数据获取功能，然后再去调试。

4.2 上位机显示

下载代码到 ALIENTEK NANO STM32F103 开发板上，MPU6050 初始化成功后，我们可以打开：ANO_Tech 匿名四轴上位机_V2.6.exe，这个软件，接收 STM32F1 上传的数据，从而图形化显示传感器数据以及飞行姿态，如图 4.2.1 和图 4.2.2 所示：

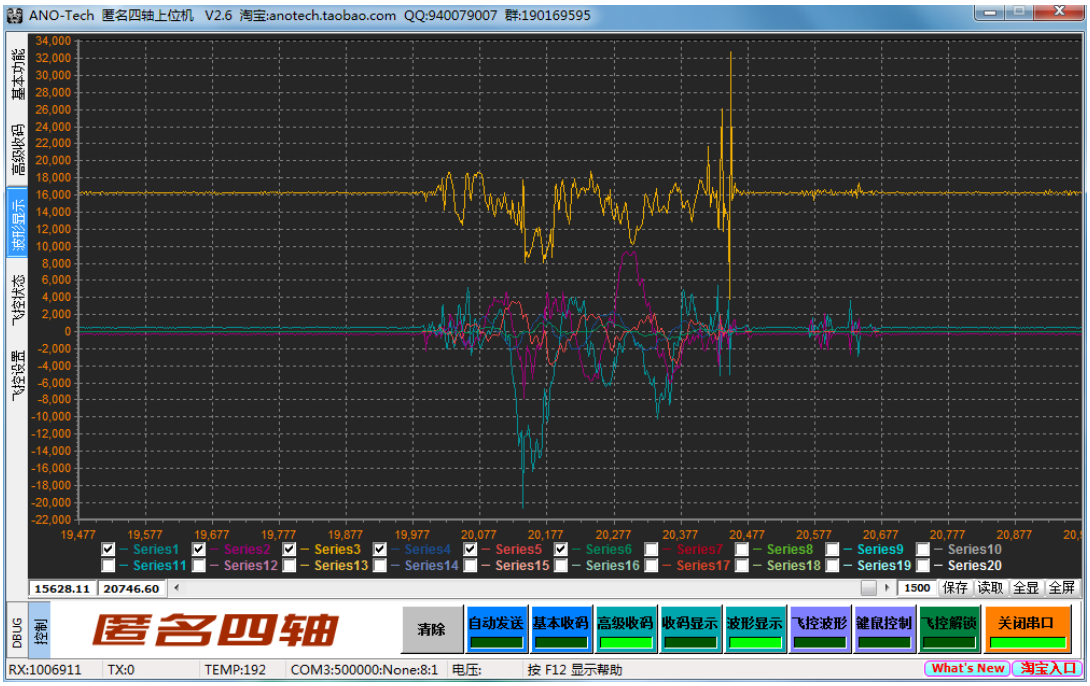

图 4.2.1 传感器数据波形显示

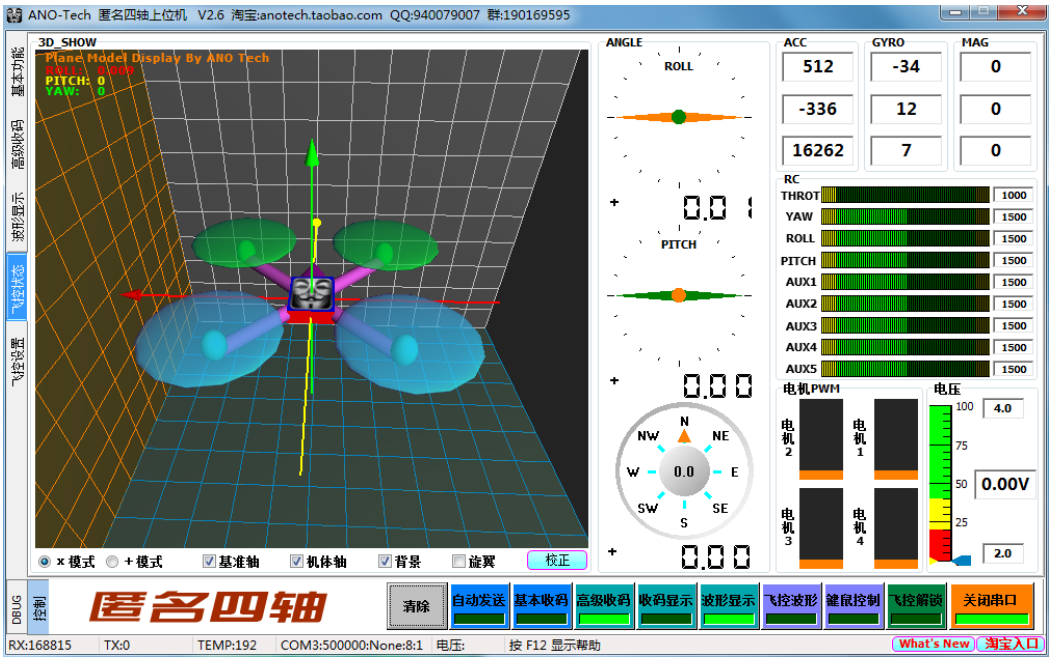

图 4.2.2 飞控状态显示

图 4.2.1 就是波形化显示我们通过 mpu6050_send_data 函数发送的数据，采用 A1 功能帧发送，总共 6 条线（Series1~6）显示波形，全部来自 A1 功能帧，int16 数据格式，Series1~6 分别代表：加速度传感器 x/y/z 和角速度传感器（陀螺仪）x/y/z 方向的原始数据。

图 4.2.2 则 3D 显示了我们开发板的姿态，通过 usart1_report_imu 函数发送的数据显示，采用飞控显示帧格（AF）式上传，同时还显示了加速度陀螺仪等传感器的原始数据。

最后，我们还可以用 USMART 读写 MPU6050 的任何寄存器，来调试代码，这里我们就不做演示了，大家自己测试即可。最后，建议大家用 USMART 调试的时候，先按 KEY0 关闭数据上传功能，否则会收到很多乱码！！，注意波特率设置为：500Kbps（设置方法：XCOM 在关闭串口状态下，选择自定义波特率，然后输入：500000，再打开串口就可以了）。

正点原子@ALIENTEK
公司网址：www.alientek.com
技术论坛：www.openedv.com
电话：020-38271790
传真：020-36773971