姿态_MPU6050 模块

销售与服务联系

东莞野火科技有限公司

地址：东莞市大岭山镇石大路 2 号艺华综合办公大楼 301 1 2 3 4 楼
官网：https://embedfire.com
论坛：http://www.firebbs.cn
资料：https://doc.embedfire.com
天猫：https://yehuosm.tmall.com
京东：https://yehuo.jd.com/
邮箱：embedfire@embedfire.com
电话：0769-33894118

第一章 产品介绍

1.1 模块简介

YH-MPU6050 是野火科技推出的六轴传感器模块，它采用 InvenSense 公司的 MPU6050 作为主芯片，能同时检测三轴加速度、三轴陀螺仪(三轴角速度)的运动数据以及温度数据。利用 MPU6050 芯片内部的 DMP 模块（Digital Motion Processor 数字运动处理器），可对传感器数据进行滤波、融合处理，直接通过 IIC 接口向主控器输出姿态解算后的数据，降低主控器的运算量。其姿态解算频率最高可达 200Hz，非常适合用于对姿态控制实时要求较高的领域。常见应用于手机、智能手环、四轴飞行器、计步器等的姿态检测。

1.2 参数特性

参数	说明
供电	3.3V-5V
通讯接口	IIC 协议，支持的 IIC 时钟最高频率为 400KHz
DMP 姿态解算频率	最高 200Hz
测量维度	加速度：3 维　　陀螺仪：3 维
ADC 分辨率	加速度：16 位　　陀螺仪：16 位
加速度测量范围	±2g、±4g、±8g、±16g　　其中 g 为重力加速度常数，g=9.8m/s ²
加速度最高分辨率	16384 LSB/g
加速度测量精度	0.1g
加速度输出频率	最高 1000Hz
陀螺仪测量范围	±250 °/s 、±500 °/s 、±1000 °/s、±2000 °/s、
陀螺仪最高分辨率	131 LSB/(°/s)
陀螺仪测量精度	0.1 °/s
陀螺仪输出频率	最高 8000Hz
温度传感器测量范围	-40~ +85℃
温度传感器分辨率	340 LSB/℃
温度传感器精度	±1℃
工作温度	-40~ +85℃
功耗	500uA~3.9mA (工作电压 3.3V)

第二章 使用说明

2.1 模块原理/概念说明

MPU6050 的工作原理可以简单描述如下：

三轴加速度计：测量物体在 X、Y、Z 三个轴向上的加速度，根据牛顿定律和重力加速度，可以推导出物体的倾斜角度。

三轴陀螺仪：测量物体绕 X、Y、Z 三个轴向的角速度，通过积分可以得到物体在各轴上的旋转角度变化。

结合加速度计和陀螺仪的数据，可以使用一种称为“互补滤波”的算法来获得更准确的姿态信息。互补滤波算法通过结合加速度计和陀螺仪的数据，利用它们各自的优势来消除彼此的误差，从而得到相对准确的姿态信息。

2.2 接口/通讯简介

I²C（Inter-Integrated Circuit）是一种串行通信接口，用于在多个设备之间进行数据传输和通信。它由飞利浦（Philips）公司开发，并在现代电子设备中广泛应用。以下是关于 I²C 接口的简要介绍：

总线拓扑结构：I²C 接口采用主从结构。其中，主设备（Master）负责控制通信的发起和管理，而从设备（Slave）则被动地响应主设备的指令和数据。

通信方式：I²C 使用两根线进行通信，分别是数据线（SDA）和时钟线（SCL）。数据线用于传输实际的数据，而时钟线用于同步数据传输的时序。

地址分配：每个从设备在总线上都有一个唯一的 7 位或 10 位地址，主设备通过发送地址来选择要与之通信的特定从设备。

数据传输：数据传输分为两种模式：传输字节和传输位。在传输字节模式下，主设备将一个字节的数据发送给从设备，而在传输位模式下，主设备发送或接收一个单独的位。

起始和停止条件：I²C 通信的开始和结束都有特定的条件。起始条件是主设备在时钟线为高电平时，数据线从高电平跳变到低电平。停止条件是主设备在时钟线为高电平时，数据线从低电平跳变到高电平。

速度和模式：I²C 支持不同的时钟频率，常见的有标准模式（100 kHz）和快速模式（400 kHz）。一些设备还支持更高的速度，如高速模式（3.4 MHz）和超高速模式。

碰撞检测：I²C 具有碰撞检测机制，能够检测到多个设备同时尝试在总线上传输数据时的冲突，并采取相应的措施进行处理。

2.3 程序流程

例程：野火小智 STM32F103C8 核心板_HAL——串口发送

根据芯片手册以及原理图，可知以下信息：

- SDA 引脚：该引脚为输入输出引脚，用于写/读数据
- SCL 引脚：该引脚为输入引脚，接收时钟信号，用于同步数据的传输。
- INT 引脚：该引脚为中断信号输出引脚，MPU6050 在触发相关中断后会产生电平差。
- ADO 引脚：该引脚用于控制 MPU6050 的地址，默认接低电平，地址为 0x68。

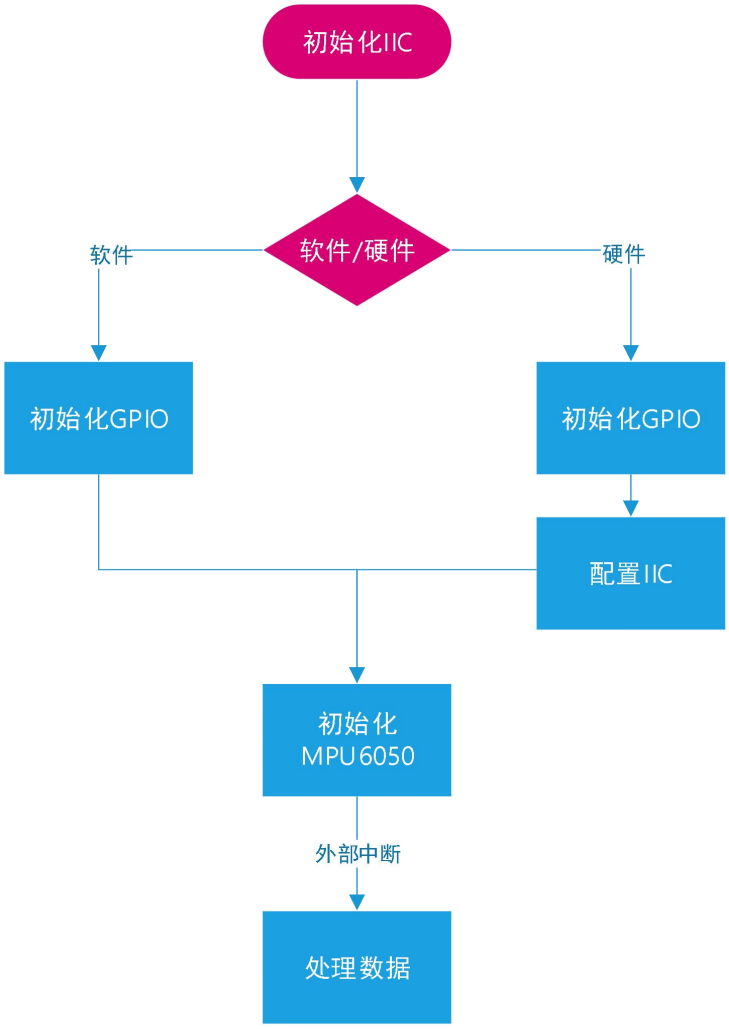
根据以上信息，可以初步确定编程步骤：

第一步：根据软/硬件 IIC 初始化 GPIO 以及 IIC；

第二步：初始化 MPU6050；

第三步：通过 INT 引脚判断是否接收到数据，对数据进行处理；

第四步：打印数据。



更多详细内容可参考芯片数据手册

第三章 程序介绍

3.1 通用接线方式

编号	主控	MPU6050 模块
1	3.3V	3.3V
2	GND	GND
3	SDA	SDA
4	SCL	SCL
5	外部中断引脚	INT
6	3.3V/GND(默认)	ADO

3.2 例程接线与操作

1. 例程的具体引脚分配与接线方式请查看例程文件夹内的引脚分配表，结合实际源码内容查看。

2. 例程操作：

例程：野火小智 STM32F103C8 核心板——串口发送

因为本次实验涉及到串口，需要先将 STM32C8T6 核心板上的串口 1 连接至 CH340。若核心板上有 PA10、PA9 与 TXD、RXD 相邻的排针，则可以通过跳线帽与之相连；若核心板上没有此排针，需要一个单独的 CH340 模块与串口 1 的引脚相连，从而保证串口输出。

将电脑连接上串口，并且打开串口助手，例程的串口参数为：波特率 115200、无校验位、8 位数据位、1 位停止位。确认电脑端能查看到该串口设备。在烧录完代码后，可以在串口助手看到相关信息。如果串口助手没有打印信息，建议按下复位键，因为可以烧录的配置中未设置烧录完成后复位。

例程：野火小智 STM32F103C8 核心板——上位机

本次实验同样要用到串口，串口的硬件配置同上。但是本次实验不能直接通过查看串口助手打印的数据来得到有效信息，因为用到了上位机——**“匿名上位机 V7”**，串口传来的数据都根据上位机的数据协议包装过的。

代码烧录完成后，连接串口至电脑端，在文件夹**.\ebf_mpu6050\1-硬件资料_参考资料_模块手册\例程工具上位机**中可以找到 **匿名上位机 V7.exe**。双击打开软件，可以看见有四个大的板块，分别为：基本收发、协议解析、数据波形、飞控数据。点击右下角的“连接设置”配置串口的相关参数，串口号直接使用自己的相应串口号即可（可以通过设备管理器查看），波特率实验例程设置的是 115200。配置完成后点击右下角的“打开连接”，随后点击右下角“功能界面”后面的第四个图标（类似于一个小飞机），也就是飞控数据，可以看到左侧有个 3D 无人机，右侧是各项参数。至此就可以看到 3D 无人机和右侧数据根据 MPU6050 的姿态变化而变化。

3.3 例程关键代码部分讲解

例程：野火小智 STM32F103C8 核心板_HAL——串口发送

IIC 代码讲解

```
1.  //选择软件 I2C，注释掉可选择硬件 I2C
2.  #define  i2c_soft
3.  //#define  i2c_hard
```

此代码部分在 MPU6050.h 中，用于选择当前代码使用的是软件 IIC 和硬件 IIC。

初始化 IIC 的 GPIO

```
1.  void i2c_GPIO_Config(void)
2.  {
3.   GPIO_InitTypeDef GPIO_InitStructure;
4.
5.      //开启 I2C 的 GPIO 时钟
6.      SOFT_I2C_GPIO_CLK_ENABLE();
7.
8.      //初始化 GPIO
9.      GPIO_InitStructure.Pin   = SOFT_I2C_SCL_PIN | SOFT_I2C_SDA_PIN;
10.     GPIO_InitStructure.Mode  = GPIO_MODE_OUTPUT_OD;//复用开漏输出
11.     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
12.     GPIO_InitStructure.Pull  = GPIO_NOPULL;
13.     HAL_GPIO_Init(GPIO_PORT_SOFT_I2C,&GPIO_InitStructure);
14.
15.  /* 给一个停止信号，复位 I2C 总线上的所有设备到待机模式 */
16.  i2c_Stop();
17. }
```

此处以软件 IIC 举例，配置 GPIO 为开漏输出。利用开漏输出的特性，开漏输出可以方便地与外部上拉电阻结合，从而实现电平的转换。因为在 I2C 总线中，数据线上的高电平是由外部上拉电阻提供的，设备只需要将线拉低即可，不需要提供高电平。这样就可以实现不同电平之间的适配。

软件 IIC 的起始信号

```
1.  void i2c_Start(void)
2.  {
3.   /* 当 SCL 高电平时，SDA 出现一个下跳沿表示 I2C 总线启动信号 */
```

```
4.   I2C_SDA_1();
5.   I2C_SCL_1();
6.   i2c_Delay();
7.   I2C_SDA_0();
8.   i2c_Delay();
9.   I2C_SCL_0();
10.  i2c_Delay();
11. }
```

通过 SCL 高电平时，SDA 产生下降沿来表达。

软件 IIC 的停止信号

```
1.  void i2c_Stop(void)
2.  {
3.   /* 当SCL 高电平时，SDA 出现一个上跳沿表示 I2C 总线停止信号 */
4.   I2C_SDA_0();
5.   I2C_SCL_1();
6.   i2c_Delay();
7.   I2C_SDA_1();
8.  }
```

通过 SCL 高电平时，SDA 产生上升沿来表达。

软件 IIC 发送一个字节

```
1.  void i2c_SendByte(uint8_t _ucByte)
2.  {
3.   uint8_t i;
4.
5.   /* 先发送字节的高位 bit7 */
6.   for (i = 0; i < 8; i++)
7.   {
8.    if (_ucByte & 0x80)
9.    {
10.    I2C_SDA_1();
11.   }
12.   else
13.   {
14.    I2C_SDA_0();
15.   }
16.   i2c_Delay();
17.   I2C_SCL_1();
```

```
    i2c_Delay();
    I2C_SCL_0();
    if (i == 7)
    {
      I2C_SDA_1(); // 释放总线
    }
    _ucByte <<= 1; /* 左移一个 bit */
    i2c_Delay();
  }
}
```

通过和 0x80 相与，得到最高位并且选择相应的 SDA 电平，电平转换过程都是在 SCL 为低电平的时间里进行的，当 SCL 为高电平时，SDA 应当保持稳定，否则会出现通讯错误。每发送完一位数据，字节左移一位，重复上面的操作，直到发送完一个字节，最后 SDA 拉高，释放总线。

IIC 读取一个字节

```
uint8_t i2c_ReadByte(uint8_t ack)
{
  uint8_t i;
  uint8_t value;

  /* 读到第 1 个 bit 为数据的 bit7 */
  value = 0;
  for (i = 0; i < 8; i++)
  {
    value <<= 1;
    I2C_SCL_1();
    i2c_Delay();
    if (I2C_SDA_READ())
    {
      value++;
    }
    I2C_SCL_0();
    i2c_Delay();
  }
  if(ack==0)
    i2c_NAck();
  else
    i2c_Ack();
  return value;
```

```
25. }
```

当 SCL 为高电平时，读取 SDA 引脚的电平状态。如果为高电平则加 1，低电平则不变，随后将数据左移一位，如此重复，直到读完一个字节。最后判断传入的形参，选择非应答信号或应答信号。

MPU6050 代码讲解

写 MPU6050 寄存器

```
1.  int MPU6050_WriteReg(unsigned char slave_addr, unsigned char reg_addr, unsign
    ed short len,unsigned char *data_ptr)
2.  {
3.      unsigned short i = 0;
4.      slave_addr <<= 1;
5.      i2c_Start();
6.      i2c_SendByte(slave_addr);
7.
8.      if(i2c_WaitAck())
9.          I2Cx_Error(slave_addr);
10.
11.     i2c_SendByte(reg_addr);
12.
13.     if(i2c_WaitAck())
14.         I2Cx_Error(slave_addr);
15.
16.     for(i=0;i<len;i++)
17.     {
18.         i2c_SendByte(data_ptr[i]);
19.         if(i2c_WaitAck())
20.             I2Cx_Error(slave_addr);
21.     }
22.     i2c_Stop();
23.     return 0;
24. }
```

先发送起始信号，随后发送 MPU6050 的器件地址。每发送一个数据都要等待一次应答信号，若未得到应答信号则报错。再发送寄存器地址、发送数据。发送完数据后再发送停止信号。

单次写入时序								
主机	S	AD+W		RA		DATA		P
从机			ACK		ACK		ACK	

连续写入时序

主机	S	AD+W		RA		DATA		DATA		P
从机			ACK		ACK		ACK		ACK	

PS：

符号	S	AD	RA	W	R	ACK	NACK	DATA	P
意义	开始信号	寄存器地址	寄存器地址	写信号	读信号	应答信号	非应答信号	数据	停止信号

读取寄存器数据

```
1.  int MPU6050_ReadData(unsigned char slave_addr, unsigned char reg_addr, unsign
    ed short len,unsigned char *data_ptr)
2.  {
3.   unsigned char i;
4.   slave_addr <<= 1;
5.
6.   i2c_Start();
7.   i2c_SendByte(slave_addr);
8.
9.   if(i2c_WaitAck())
10.         I2Cx_Error(slave_addr);
11.
12.  i2c_SendByte(reg_addr);
13.
14.     if(i2c_WaitAck())
15.         I2Cx_Error(slave_addr);
16.
17.  i2c_Start();
18.  i2c_SendByte(MPU6050_SLAVE_ADDRESS+1);
19.
20.     if(i2c_WaitAck())
21.         I2Cx_Error(slave_addr);
22.
23.  for(i=0;i<(len-1);i++){
24.   *data_ptr=i2c_ReadByte(1);
25.   data_ptr++;
26.  }
27.  *data_ptr=i2c_ReadByte(0);
28.  i2c_Stop();
29.     return 0;
30. }
```

发送开始信号后，先发送 MPU6050 的器件地址，再发送寄存器地址。此后重新发送开始信号，发送 MPU6050 器件地址加一，表示读出数据。因为器件地址只由七位数据组成，最后一位为读写位，0 表示写、1 表示读。发送完地址后开始接收数据，接收完数据后，主机需要发送非应答和停止位，表示读出数据完成。

单次读出时序											
主机	S	AD+W		RA		S	AD+R			NACK	P
从机			ACK		ACK			ACK	DATA		

连续读出时序													
主机	S	AD+W		RA		S	AD+R			ACK		NACK	P
从机			ACK		ACK			ACK	DATA		DATA		

PS：

符号	S	AD	RA	W	R	ACK	NACK	DATA	P
意义	开始信号	寄存器地址	寄存器地址	写信号	读信号	应答信号	非应答信号	数据	停止信号

MPU6050 初始化

```
1.  void MPU6050_Init(void)
2.  {
3.    int i=0,j=0;
4.    uint8_t reg[5]={0x00,0x07,0x06,0x01,0x18};
5.    //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.    {
8.      for(j=0;j<1000;j++)
9.      {
10.       ;
11.     }
12.   }
13.   MPU6050_WriteReg(MPU6050_SLAVE_ADDRESS,MPU6050_RA_PWR_MGMT_1,1, &reg[0]);
        //解除休眠状态
14.   MPU6050_WriteReg(MPU6050_SLAVE_ADDRESS,MPU6050_RA_SMPLRT_DIV ,1, &reg[1]);
        //陀螺仪采样率
15.   MPU6050_WriteReg(MPU6050_SLAVE_ADDRESS,MPU6050_RA_CONFIG , 1,&reg[2]);
16.   MPU6050_WriteReg(MPU6050_SLAVE_ADDRESS,MPU6050_RA_ACCEL_CONFIG ,1, &reg[3]);
        //配置加速度传感器工作在 4G 模式
17.   MPU6050_WriteReg(MPU6050_SLAVE_ADDRESS,MPU6050_RA_GYRO_CONFIG,1, &reg[4]);
        //陀螺仪自检及测量范围，典型值：0x18(不自检，2000deg/s)
18. }
```

在短暂延迟后，给寄存器配置相应的值，这些值都可以根据不同的需求去更改。更多详细内容可参考芯片数据手册。

DMP 代码配置

DMP（Digital Motion Processor）库是由 InvenSense 公司开发的一种软件库，是 MPU 芯片自带的一种功能。它允许开发者轻松地获取物体的姿态、方向和速度等高级别的信息，而无需编写复杂的算法。要移植该软件库我们需要为它提供 I2C 读写接口、定时服务以及 MPU6050 的数据更新标志。

“inv_mpu.c”文件

```
#define i2c_write MPU6050_WriteReg
#define i2c_read MPU6050_ReadData
#define delay_ms HAL_Delay
#define get_ms get_ms_user
```

在“inv_mpu.c”文件的最上方提供了宏定义，根据我们前面写的函数以及 HAL 库提供的函数去更改宏定义的内容。IIC 的读写即为前面 MPU6050 的寄存器读写；毫秒级的延时由 HAL 库中的延时函数提供；获取毫秒级的时间戳本质上也由 HAL 库提供，此处只是封装成了一个函数，不多赘述。

“log_stm32.c”文件

```
//================移植修改================
extern UART_HandleTypeDef UartHandle;
#define PRINT_UART_HANDLE UartHandle
```

在“log_stm32.c”文件的最上方提供了串口的宏定义，将当前文件夹中的串口句柄外部声明至此，修改宏定义即可。

```
void EXTI15_10_IRQHandler(void)
{
    MPU6050_data_ready_cb();
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_11);
}
void MPU6050_data_ready_cb(void) //中断回调函数
{
    hal.new_gyro = 1;
}
```

MPL 代码库使用了 MPU6050 的 INT 中断信号，为此我们要给提供中断接口。本例程中选择了 PA11 作为了外部中断引脚，触发中断则代表有新数据传出，触发中断回调函数，将标志位置一。

Main 函数

```
int main(void)
```

```
{
    unsigned char new_temp = 0;
    unsigned long timestamp;

    /* 配置系统时钟为72 MHz */
    SystemClock_Config();
    /* 初始化USART1 配置模式为 115200 8-N-1 */
    DEBUG_USART_Config();
    /* 初始化LED */
    LED_GPIO_Config();
    /* 初始化MPU6050 的INT 引脚 */
    MPU_INT_GPIO_Init();
    /* 根据宏定义选择软件还是硬件I2C */
    #ifdef i2c_soft
        i2c_GPIO_Config();
    #else
        MPU_I2C_Config();
    #endif


    //初始化MPU6050
    MPU6050_mpu_init();
    MPU6050_mpl_init();
    MPU6050_config();

    printf("欢迎使用野火——MPU6050 例程\n");
    HAL_Delay(500);

    while (1)
    {
        unsigned long sensor_timestamp;
        int new_data = 0;

        /* 接收到INT 传来的中断信息后继续往下 */
        if (!hal.sensors || !hal.new_gyro)
        {
            continue;
        }

```

```
41.         /* 每过500ms 读取一次温度 */
42.         get_tick_count(&timestamp);
43.         if (timestamp > hal.next_temp_ms)
44.         {
45.             hal.next_temp_ms = timestamp + TEMP_READ_MS;
46.             new_temp = 1;
47.         }
48.
49.         /* 接收到新数据 并且 开启DMP */
50.         if (hal.new_gyro && hal.dmp_on)
51.         {
52.             short gyro[3], accel_short[3], sensors;
53.             unsigned char more;
54.             long accel[3], quat[4], temperature;
55.             /* 当DMP 正在使用时，该函数从 FIFO 获取新数据 */
56.             dmp_read_fifo(gyro, accel_short, quat, &sensor_timestamp, &sensor
    s, &more);
57.             /* 如果more=0，则数据被获取完毕 */
58.             if (!more)
59.                 hal.new_gyro = 0;
60.
61.             /* 读取相对应的新数据，推入 MPL */
62.             //四元数数据
63.             if (sensors & INV_WXYZ_QUAT)
64.             {
65.                 inv_build_quat(quat, 0, sensor_timestamp);
66.                 new_data = 1;
67.             }
68.
69.             //陀螺仪数据
70.             if (sensors & INV_XYZ_GYRO)
71.             {
72.                 inv_build_gyro(gyro, sensor_timestamp);
73.                 new_data = 1;
74.                 if (new_temp)
75.                 {
76.                     new_temp = 0;
77.                     /* 获取温度，仅用于陀螺温度比较 */
78.                     mpu_get_temperature(&temperature, &sensor_timestamp);
```

```
79.                     inv_build_temp(temperature, sensor_timestamp);
80.                 }
81.             }
82.
83.             //加速度计数据
84.             if (sensors & INV_XYZ_ACCEL)
85.             {
86.                 accel[0] = (long)accel_short[0];
87.                 accel[1] = (long)accel_short[1];
88.                 accel[2] = (long)accel_short[2];
89.                 inv_build_accel(accel, 0, sensor_timestamp);
90.                 new_data = 1;
91.             }
92.         }
93.
94.         if (new_data)
95.         {
96.             long data[9];
97.             int8_t accuracy;
98.             /* 处理接收到的数据 */
99.             if (inv_execute_on_data())
100.                {
101.                    printf("数据错误\n");
102.                }
103.
104.                float float_data[3];
105.                printf("\r\n\r\n 以下是 eMPL_outputs.c 中的函数获取的数据
    \r\n");
106.                if (inv_get_sensor_type_euler(data, &accuracy, (inv_time_t
    *)&timestamp))
107.                {
108.                    float_data[0] = data[0] * 1.0 / (1 << 16);
109.                    float_data[1] = data[1] * 1.0 / (1 << 16);
110.                    float_data[2] = data[2] * 1.0 / (1 << 16);
111.                    printf("欧拉角
    (rad)\t\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], float_
    data[2]);
112.                }
```

```
113.                 if (inv_get_sensor_type_accel(data, &accuracy, (inv_time_t
    *)&timestamp))
114.                 {
115.                     float_data[0] = data[0] * 1.0 / (1 << 16);
116.                     float_data[1] = data[1] * 1.0 / (1 << 16);
117.                     float_data[2] = data[2] * 1.0 / (1 << 16);
118.                     printf("加速度
    (g/s)\t\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], float_
    data[2]);
119.                 }
120.
121.                 if (inv_get_sensor_type_gyro(data, &accuracy, (inv_time_t *)
    &timestamp))
122.                 {
123.                     float_data[0] = data[0] * 1.0 / (1 << 16);
124.                     float_data[1] = data[1] * 1.0 / (1 << 16);
125.                     float_data[2] = data[2] * 1.0 / (1 << 16);
126.                     printf("角速度
    (rad/s)\t\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], floa
    t_data[2]);
127.                 }
128.
129.                 printf("\r\n\r\n 以下是 hal_outputs.c 中的函数获取的数据\r\n");
130.                 inv_get_sensor_type_orientation(float_data, &accuracy, (inv
    _time_t *)&timestamp);
131.                 printf("伪方位角
    (rad/s)\t: %7.5f\t %7.5f\t %7.5f\t \r\n", float_data[0],
132.                         float_data[1], float_data[2]);
133.
134.                 if (inv_get_sensor_type_accelerometer(float_data, &accuracy,
     (inv_time_t *)&timestamp))
135.                 {
136.                     printf("加速度
    (m/s^2)\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], float_
    data[2]);
137.                 }
138.                 if (inv_get_sensor_type_linear_acceleration(float_data, &ac
    curacy, (inv_time_t *)&timestamp))
139.                 {
```

```
140.                     printf("线性加速度
    (m/s^2)\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], float_
    data[2]);
141.                 }
142.                 if (inv_get_sensor_type_gyroscope(float_data, &accuracy, (i
    nv_time_t *)&timestamp))
143.                 {
144.                     printf("校准后角速度
    (rad/s)\t: %7.5f\t %7.5f\t %7.5f\t\r\n", float_data[0], float_data[1], float_
    data[2]);
145.                 }
146.                 HAL_Delay(100);
147.             }
148.
149.         }
150.
151.     }
```

Main 函数主要流程如下：

1. 初始话系统时钟、串口、LED、MPU6050；
2. 等待 INT 引脚触发外部中断，也就是接收 MPU6050 传来新数据；
3. 每过 500ms 读取一次温度
4. 若接收到数据并且 DMP 开启正常，进行数据处理
5. 从 FIFO 中读取数据，将其一一对应的推入 MPL，其中包括：四元数数据、陀螺仪数据、加速度计数据；
6. 只要第五步推入了新的数据，处理相关数据并且将其打印出来。