

# AN0003 I2C设备应用指南

!!! abstract "摘要"

1 | 本应用笔记以驱动I2C接口的6轴传感器MPU6050为例，说明了如何使用I2C设备驱动接口开发应用程序，并详细讲解了RT-Thread I2C设备驱动框架及相关函数。

## 1 本文的目的和结构

### 1.1 本文的目的和背景

I2C（或写作i2c、IIC、iic）总线是由Philips公司开发的一种简单、双向二线制（时钟SCL、数据SDA）同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息，是半导体芯片使用最为广泛的通信接口之一。RT-Thread中引入了I2C设备驱动框架，I2C设备驱动框架提供了基于GPIO模拟和硬件控制器的2种底层硬件接口。

### 1.2 本文的结构

本文首先描述了RT-Thread I2C设备驱动框架的基本情况，然后详细描述了I2C设备驱动接口，并使用I2C设备驱动接口编写MPU6050的驱动程序，并给出了在正点原子STM32F4探索者开发板上验证的代码示例。

## 2 I2C设备驱动框架简介

在使用MCU进行项目开发的时候，往往需要用到I2C总线。一般来说，MCU带有I2C控制器（硬件I2C），也可以使用MCU的2个GPIO自行编写程序模拟I2C总线协议实现同样的功能。

RT-Thread提供了一套I/O设备管理框架，它把I/O设备分成了三层进行处理：应用层、I/O设备管理层、底层驱动。I/O设备管理框架给上层应用提供了统一的设备操作接口和I2C设备驱动接口，给下层提供的是底层驱动接口。应用程序通过I/O设备模块提供的标准接口访问底层设备，底层设备的变更不会对上层应用产生影响，这种方式使得应用程序具有很好的可移植性，应用程序可以很方便的从一个MCU移植到另外一个MCU。

本文以6轴惯性传感器MPU6050为例，使用RT-Thread I2C设备驱动框架提供的GPIO模拟I2C控制器的方式，阐述了应用程序如何使用I2C设备驱动接口访问I2C设备。

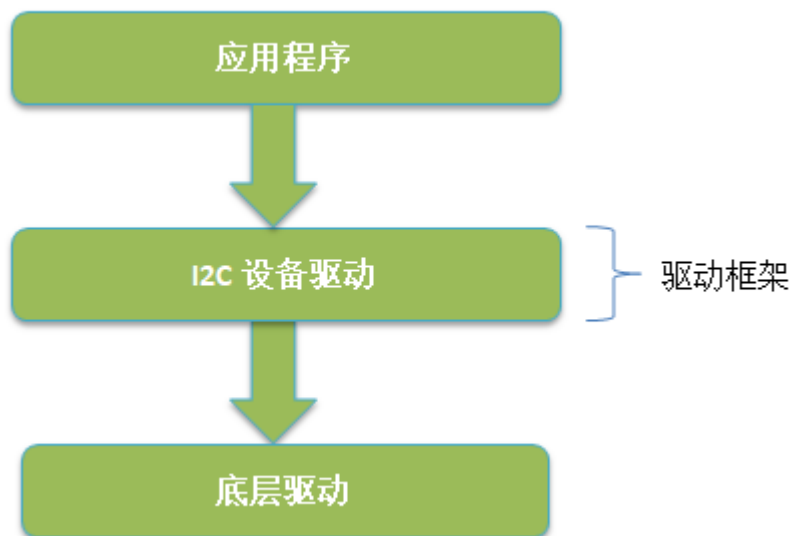


图2-1 RT-Thread I2C设备驱动框架

## 3 运行I2C设备驱动示例代码

### 3.1 示例代码软硬件平台

1. [正点原子STM32F4探索者开发板](#)
2. GY-521 MPU-6050模块
3. MDK5
4. [RT-Thread 源码](#)

正点原子探索者STM32F4 开发板的MCU是STM32F407ZGT6，本示例使用USB串口（USART1）发送数据及供电，使用SEGGER JLINK连接JTAG调试。

本次实验用的GY521模块是一款6轴惯性传感器模块，板载MPU6050。我们使用开发板的PD6（SCL）、PD7（SDA）作为模拟I2C管脚，用杜邦线将GY521模块的SCL硬件连接到PD6、SDA连接到PD7、GND连接到开发板的GND、VCC连接到3.3V。

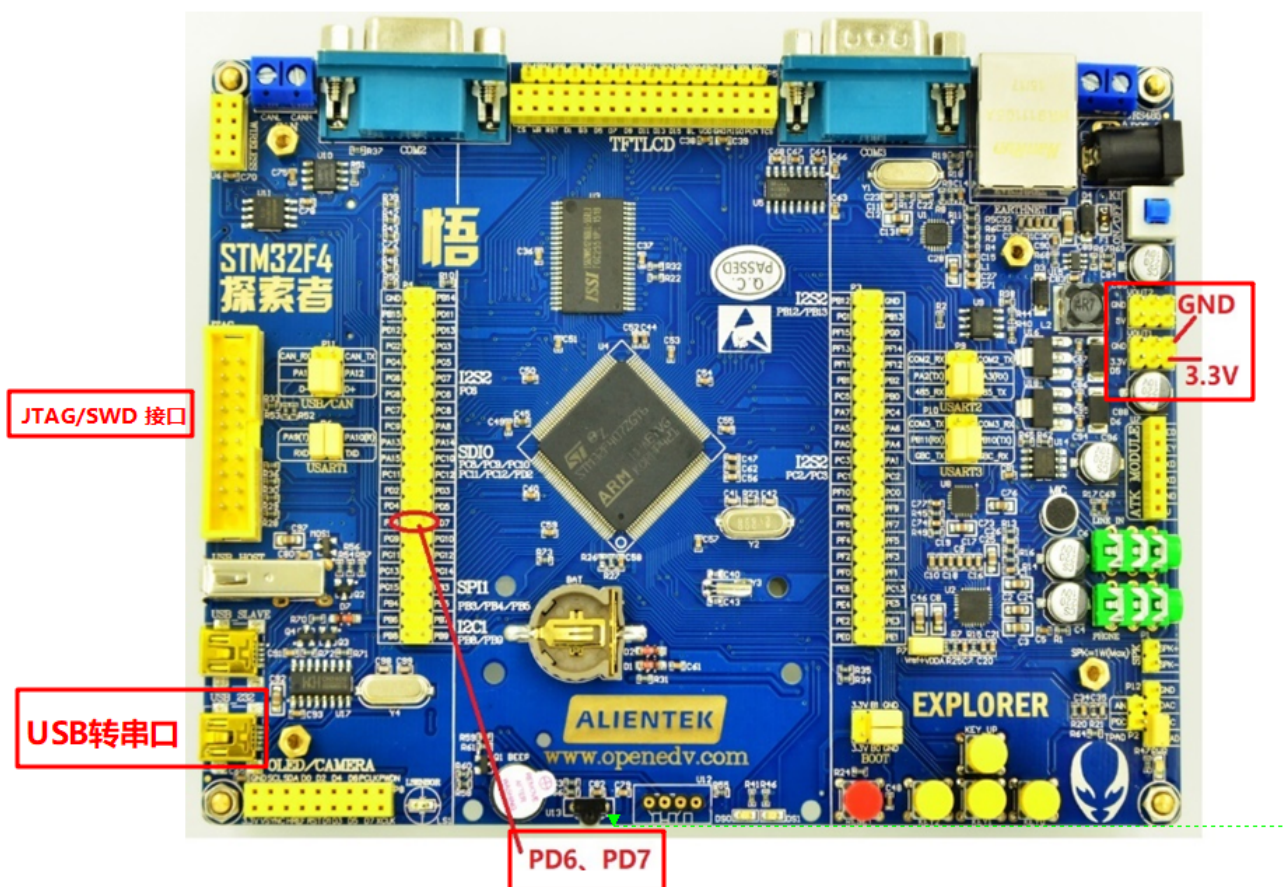


图3.1-1 正点原子开发板

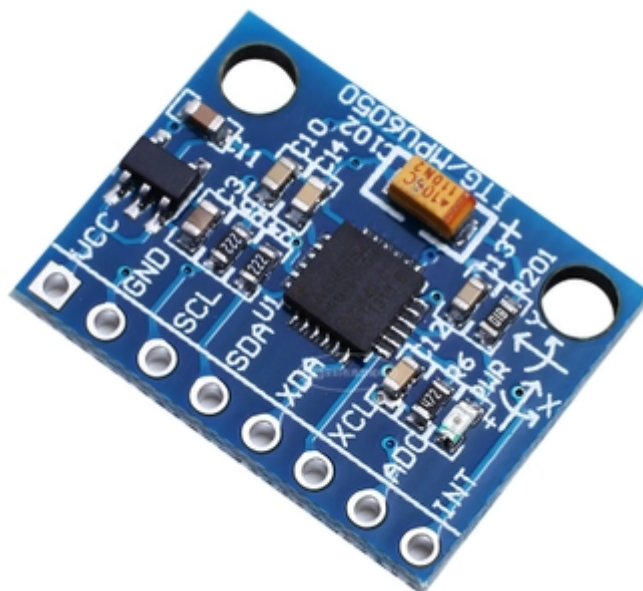


图3.1-2 GY521模块

本文基于正点原子STM32F4探索者开发板，给出了底层I2C驱动（GPIO模拟方式）的添加方法和I2C设备的具体应用示例代码（以驱动MPU6050为例），包含寄存器读、写操作方法。由于RT-Thread上层应用API的通用性，因此这些代码不局限于具体的硬件平台，用户可以轻松将它移植到其它平台上。

### 3.2 启用I2C设备驱动

- 1. 使用env工具命令行进入 rt-thread\bsp\stm32f4xx-HAL 目录，然后输入 menuconfig 命令进入配置界面。
- 2. 配置shell使用串口1：选中Using UART1，进入RT-Thread Kernel ---> Kernel Device Object菜单，修改the device name for console为uart1。
- 3. 进入RT-Thread Components ---> Device Drivers菜单，选中 Using I2C device drivers，本示例使用GPIO模拟I2C，因此还要开启 Use GPIO to simulate I2C。

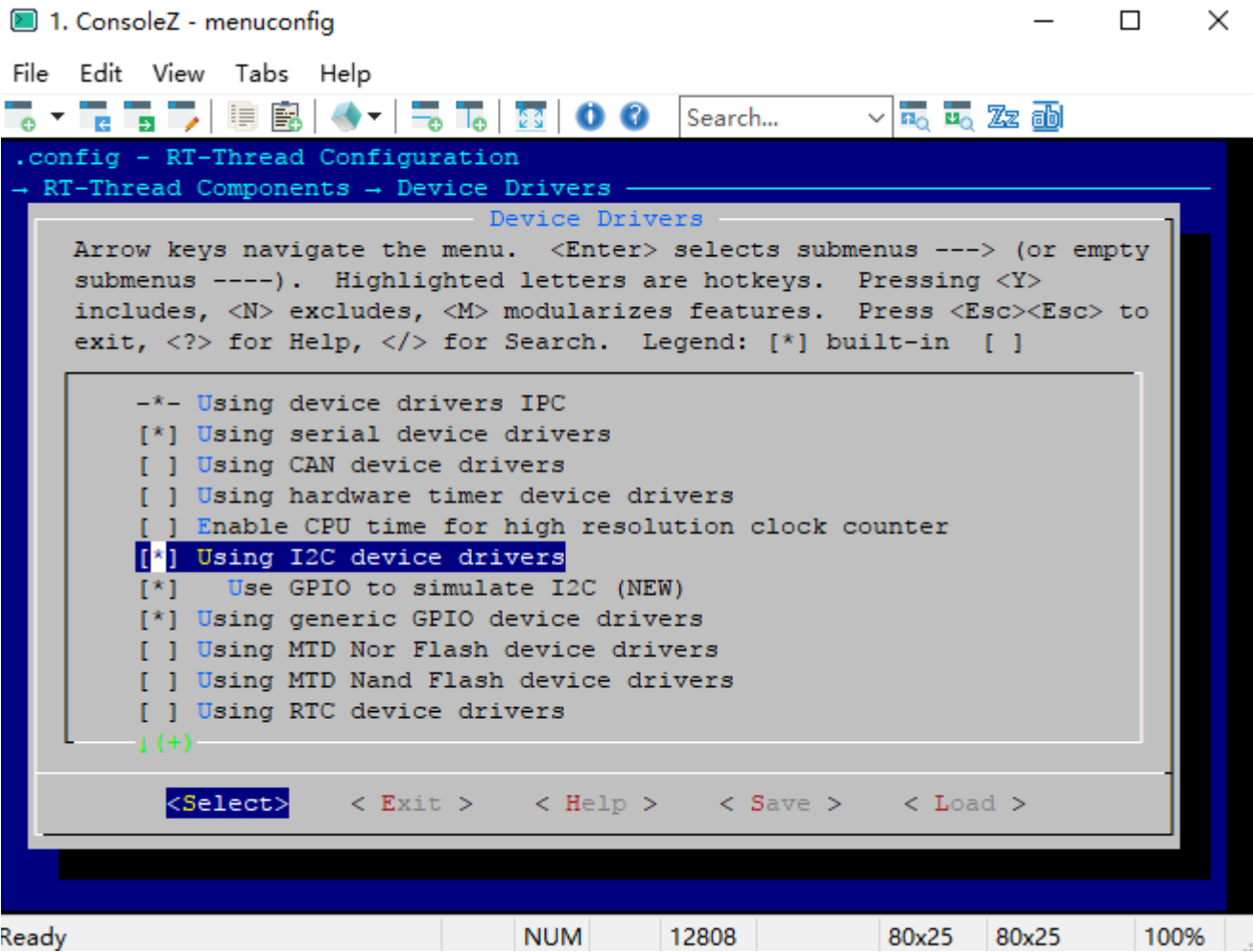


图3.2-1 使用menuconfig开启i2c

- 4. 退出menuconfig配置界面并保存配置，在env命令行输入 scons --target=mdk5 -s 命令生成mdk5工程，新工程名为project。使用MDK5打开工程，修改MCU型号为STM32F407ZGTx，修改调试选项为J-LINK。

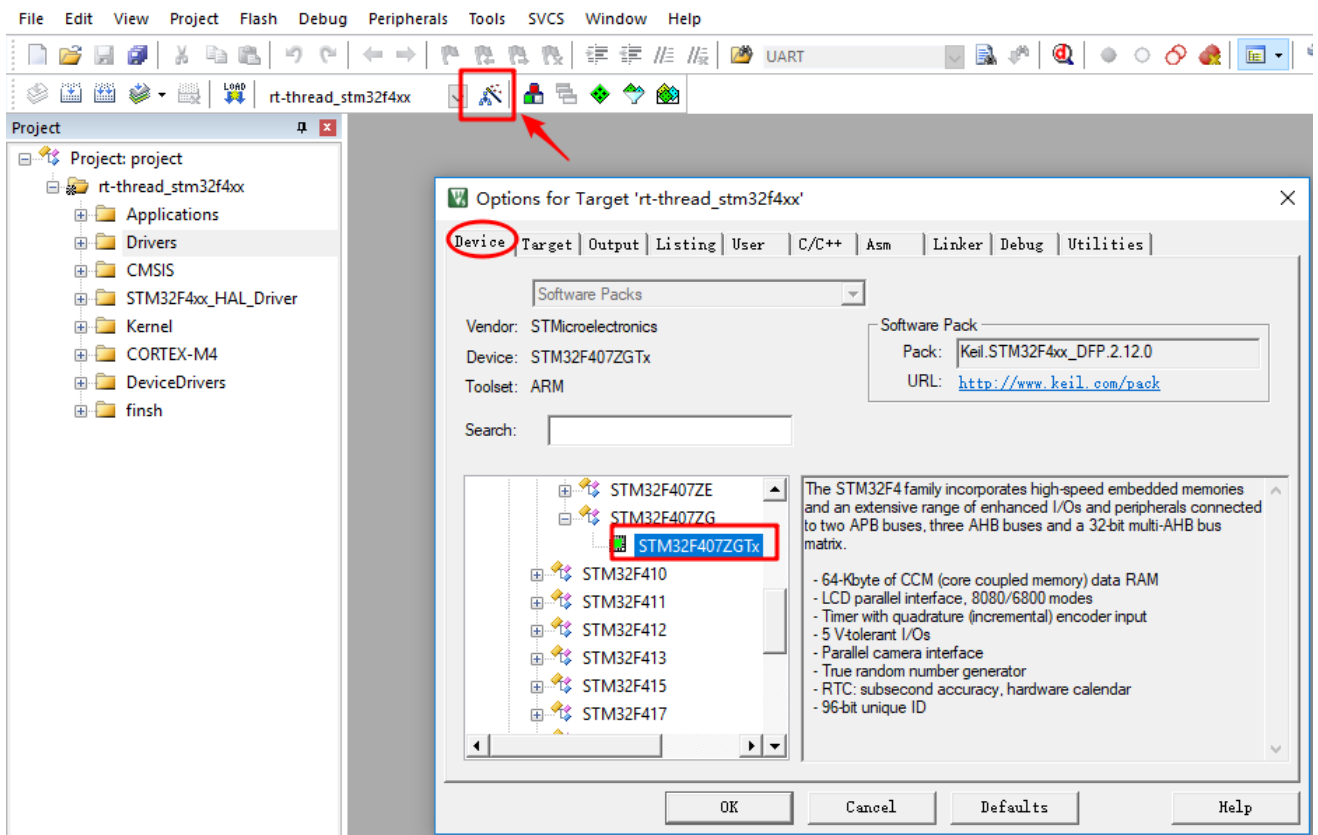


图3.2-2 修改MCU

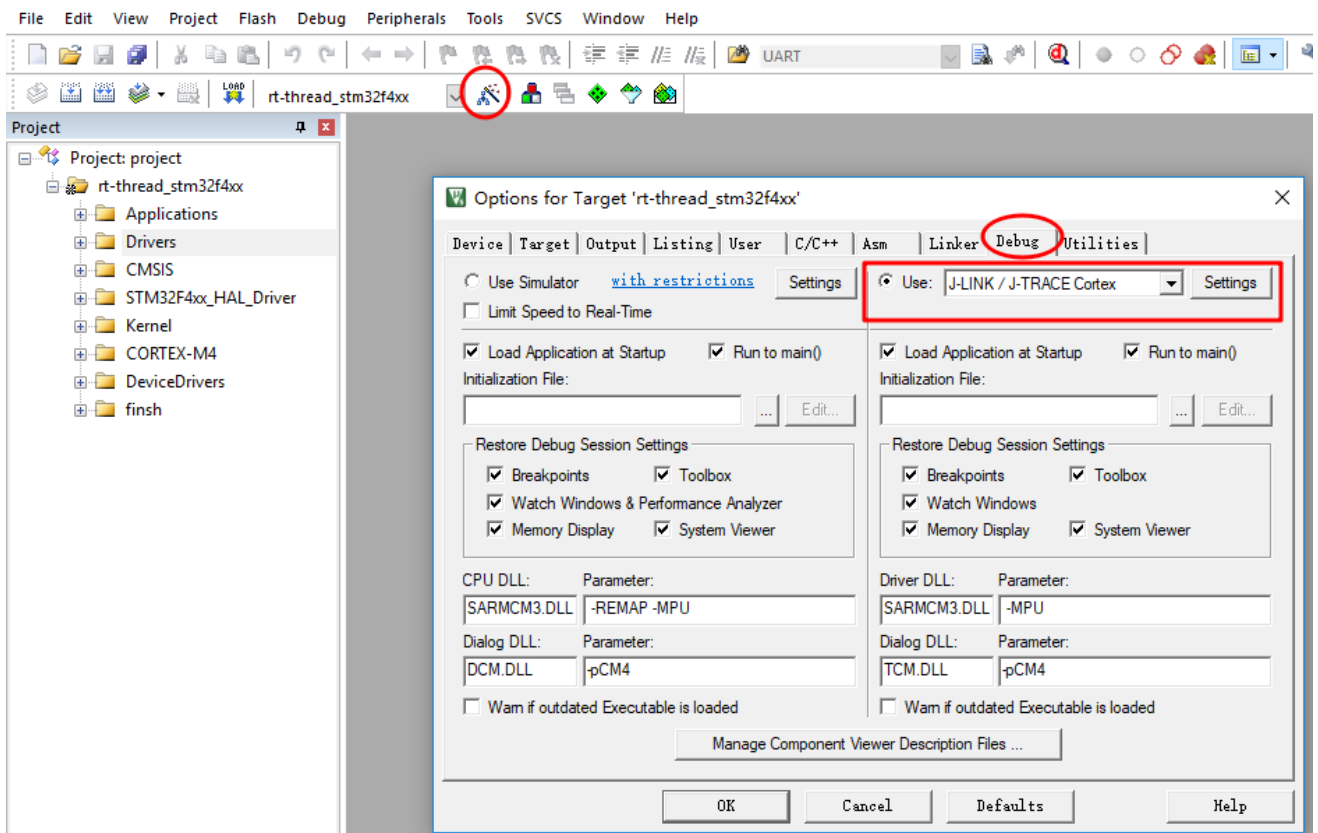
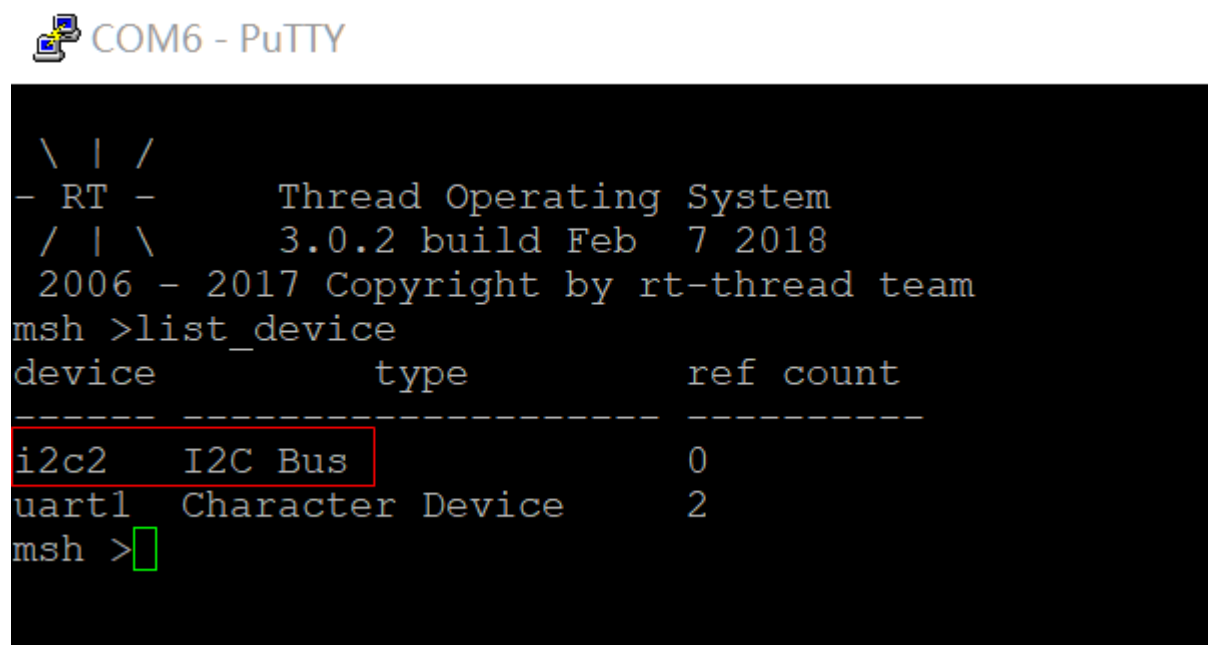


图3.2-3 修改调试选项

5. 编译工程后下载程序至开发板运行。在终端PuTTY(打开对应端口，波特率配置为115200)输入 `list_device` 命令可以看到名为i2c2的设备，设备类型是I2C Bus，说明I2C设备驱动添加成功了。如图所示：



```
\ | /
- RT -      Thread Operating System
/ | \      3.0.2 build Feb  7 2018
2006 - 2017 Copyright by rt-thread team
msh >list_device
device          type          ref count
-----
i2c2    I2C Bus          0
uart1   Character Device  2
msh >
```

图3.2-4使用list\_device命令查看i2c总线

### 3.3 运行示例代码

将I2C示例代码里的 `main.c` 拷贝到 `\rt-thread\bsp\stm32f4xx-HAL\applications` 目录，替换原有的 `main.c`。  
`drv_mpu6050.c`、`drv_mpu6050.h` 拷贝到 `\rt-thread\bsp\stm32f4xx-HAL\drivers` 目录，并将它们添加到工程中对应分组。如图所示：

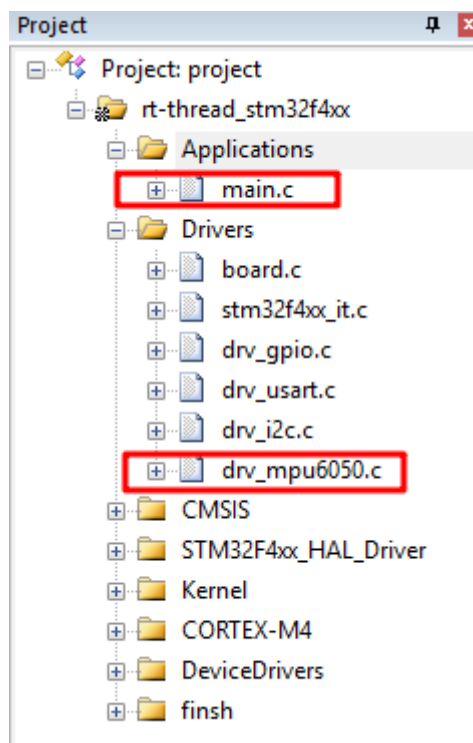
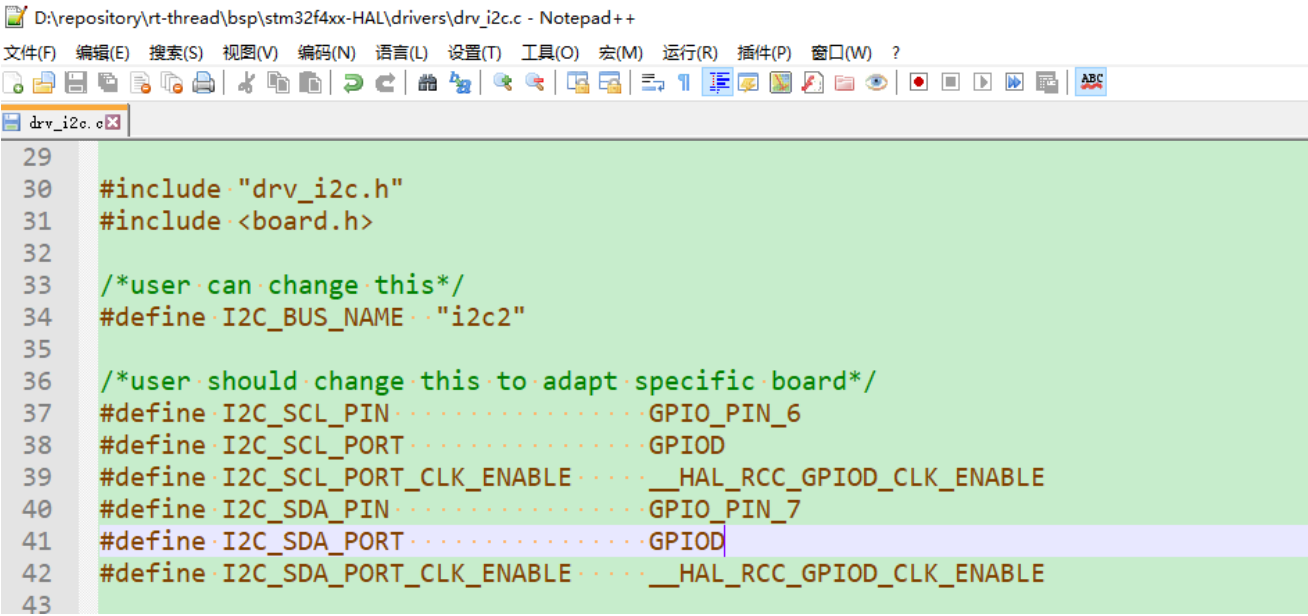


图3.3-1 添加驱动



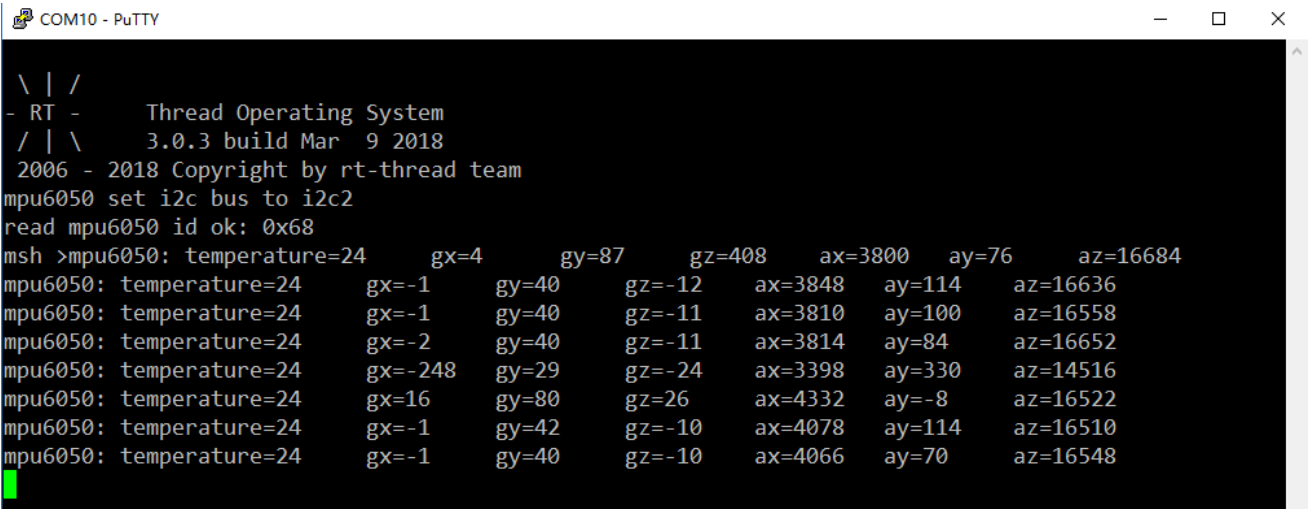
本例使用GPIO PD6作为SCL、GPIO PD7作为SDA，I2C总线名字是i2c2，读者可根据需要修改drv\_i2c.c 件中如下参数以适配自己的板卡，确保 drv\_mpu6050.c 中定义的宏MPU6050\_I2C\_BUS\_NAME与drv\_i2c.c 中的宏I2C\_BUS\_NAME相同。本示例需要将drv\_i2c.c 默认驱动端口GPIOB改为GPIOD，如下图所示：



```
29
30 #include "drv_i2c.h"
31 #include <board.h>
32
33 /*user can change this*/
34 #define I2C_BUS_NAME "i2c2"
35
36 /*user should change this to adapt specific board*/
37 #define I2C_SCL_PIN ..... GPIO_PIN_6
38 #define I2C_SCL_PORT ..... GPIOD
39 #define I2C_SCL_PORT_CLK_ENABLE ..... __HAL_RCC_GPIOD_CLK_ENABLE
40 #define I2C_SDA_PIN ..... GPIO_PIN_7
41 #define I2C_SDA_PORT ..... GPIOD
42 #define I2C_SDA_PORT_CLK_ENABLE ..... __HAL_RCC_GPIOD_CLK_ENABLE
43
```

图3.3-2 drv\_i2c.c中的i2c板级配置

连接好MPU6050模块和开发板，编译工程并下载程序至开发板，复位MCU，终端PuTTY会打印出读取到的MPU6050传感器数据，依次是温度，三轴加速度，三轴角速度：



```
\ | /
- RT -      Thread Operating System
/ | \      3.0.3 build Mar  9 2018
2006 - 2018 Copyright by rt-thread team
mpu6050 set i2c bus to i2c2
read mpu6050 id ok: 0x68
msh >mpu6050: temperature=24      gx=4      gy=87      gz=408      ax=3800      ay=76      az=16684
mpu6050: temperature=24      gx=-1      gy=40      gz=-12      ax=3848      ay=114      az=16636
mpu6050: temperature=24      gx=-1      gy=40      gz=-11      ax=3810      ay=100      az=16558
mpu6050: temperature=24      gx=-2      gy=40      gz=-11      ax=3814      ay=84      az=16652
mpu6050: temperature=24      gx=-248      gy=29      gz=-24      ax=3398      ay=330      az=14516
mpu6050: temperature=24      gx=16      gy=80      gz=26      ax=4332      ay=-8      az=16522
mpu6050: temperature=24      gx=-1      gy=42      gz=-10      ax=4078      ay=114      az=16510
mpu6050: temperature=24      gx=-1      gy=40      gz=-10      ax=4066      ay=70      az=16548
```

图3.3-3 实验现象

## 4 I2C设备驱动接口详解

按照前文的步骤，相信读者能很快的将RT-ThreadI2C设备驱动运行起来，那么如何使用I2C设备驱动接口开发应用程序呢？

RT-Thread I2C设备驱动目前只支持主机模式，使用RT-Thread I2C设备驱动需要使用menuconfig工具开启宏RT\_USING\_DEVICE和RT\_USING\_I2C，如果要使用GPIO模拟I2C还需开启宏RT\_USING\_I2C\_BITOPS。

使用I2C设备驱动的大致流程如下：

- 1. 用户可以在msh shell输入 `list_device` 命令查看已有的I2C设备，确定I2C设备名称。
- 2. 查找设备使用 `rt_i2c_bus_device_find()` 或者 `rt_device_find()`，传入I2C设备名称获取i2c总线设备句柄。
- 3. 使用 `rt_i2c_transfer()` 即可以发送数据也可以接收数据，如果主机只发送数据可以使用 `rt_i2c_master_send()`，如果主机只接收数据可以使用 `rt_i2c_master_recv()`。

接下来本章将详细讲解I2C设备驱动接口的使用。

### 4.1 查找设备

应用程序要使用已经由操作系统管理的I2C设备需要调用查找设备函数，找到I2C设备后才可以对该设备进行信息传送。

**函数原型:** `struct rt_i2c_bus_device *rt_i2c_bus_device_find(const char *bus_name)`

参数	描述
bus_name	I2C设备名称

**函数返回：**I2C设备存在则返回I2C设备句柄，否则返回RT\_NULL。

本文示例代码底层驱动 `drv_mpu6050.c` 中 `mpu6050_hw_init()` 查找设备源码如下：

```
1  #define MPU6050_I2CBUS_NAME  "i2c2" /* I2C设备名称,必须和drv_i2c.c注册的I2C设备名称一致 */
2  static struct rt_i2c_bus_device *mpu6050_i2c_bus; /* I2C设备句柄 */
3  ...
4  ...
5
6  int mpu6050_hw_init(void)
7  {
8      rt_uint8_t res;
9
10     mpu6050_i2c_bus = rt_i2c_bus_device_find(MPU6050_I2CBUS_NAME); /*查找I2C设备*/
11
12     if (mpu6050_i2c_bus == RT_NULL)
13     {
14         MPUDEBUG("can't find mpu6050 %s device\r\n",MPU6050_I2CBUS_NAME);
15         return -RT_ERROR;
16     }
17
18     ...
19     ...
20 }
21
```

### 4.2 数据传输

RT-Thread I2C设备驱动的核心API是 `rt_i2c_transfer()`，它传递的消息是链式结构的。可以通过消息链，实现调用一次完成多次数据的收发，此函数既可以用于发送数据，也可以用于接收数据。

**函数原型:**



```

1 rt_size_t rt_i2c_transfer(struct rt_i2c_bus_device *bus,
2                           struct rt_i2c_msg      msgs[],
3                           rt_uint32_t            num)

```

参数	描述
bus	I2C总线设备句柄
msgs[]	I2C消息数组
num	消息数组的数量

**函数返回：** 成功传输的消息数组的数量

消息数组msgs[]类型为

```

1 struct rt_i2c_msg
2 {
3     rt_uint16_t addr;    //从机地址
4     rt_uint16_t flags;   //标志，读、写等
5     rt_uint16_t len;     //读写数据字节数
6     rt_uint8_t  *buf;    //读写数据指针
7 }

```

addr从机地址支持7位和10位二进制地址（flags |= RT\_I2C\_ADDR\_10BIT）。RT-Thread的I2C设备驱动接口使用的从机地址均为不包含读写位的地址，读写位对应修改flags。

flags标志可选值为 i2c.h 文件中定义的宏，发送数据赋值 RT\_I2C\_WR，接收数据赋值RT\_I2C\_RD，根据需要可以与其他宏使用位运算“|”组合起来使用。

```

1 #define RT_I2C_WR          0x0000
2 #define RT_I2C_RD          (1u << 0)
3 #define RT_I2C_ADDR_10BIT (1u << 2) /* this is a ten bit chip address */
4 #define RT_I2C_NO_START    (1u << 4)
5 #define RT_I2C_IGNORE_NACK (1u << 5)
6 #define RT_I2C_NO_READ_ACK (1u << 6) /* when I2C reading, we do not ACK */

```

### 4.2.1 发送数据

用户可以调用I2C设备驱动接口 `rt_i2c_master_send()` 或者 `rt_i2c_transfer()` 发送数据。函数调用关系如下：

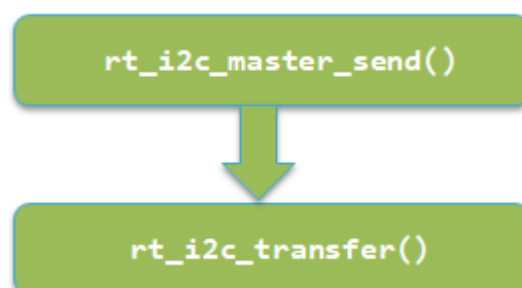


图4.2.1-1 发送数据函数调用关系

`drv_mpu6050.c` 中的 `mpu6050_write_reg()` 函数是MCU向mpu6050寄存器写数据。此函数的实现共有2种，分别调用了I2C设备驱动接口 `rt_i2c_transfer()` 和 `rt_i2c_master_send()` 实现。

本文示例使用的MPU6050数据手册中提到7位从机地址是110100X，X由芯片的AD0管脚决定，GY521模块的AD0连接到了GND，因此MPU6050作为从机时地址是1101000，16进制形式是0x68。写MPU6050某个寄存器，主机首先发送从机地址MPU6050\_ADDR、读写标志 R/W 为 RT\_I2C\_WR（0 为写，1 为读），然后主机发送从机寄存器地址reg及数据data。

### 使用rt\_i2c\_transfer()发送数据

本文示例代码底层驱动 `drv_mpu6050.c` 发送数据源码如下：

```
1  #define MPU6050_ADDR          0X68
2
3  //写mpu6050单个寄存器
4  //reg:寄存器地址
5  //data:数据
6  //返回值: 0,正常 / -1,错误代码
7  rt_err_t mpu6050_write_reg(rt_uint8_t reg, rt_uint8_t data)
8  {
9      struct rt_i2c_msg msgs;
10     rt_uint8_t buf[2] = {reg, data};
11
12     msgs.addr = MPU6050_ADDR; /* 从机地址 */
13     msgs.flags = RT_I2C_WR;   /* 写标志 */
14     msgs.buf = buf;           /* 发送数据指针 */
15     msgs.len = 2;
16
17     if (rt_i2c_transfer(mpu6050_i2c_bus, &msgs, 1) == 1)
18     {
19         return RT_EOK;
20     }
21     else
22     {
23         return -RT_ERROR;
24     }
25 }
26
```

以本文示例代码其中一次调用 `rt_i2c_transfer()` 发送数据为例，从机MPU6050地址16进制值为0X68，寄存器地址reg 16进制值为0X6B，发送的数据data 16进制值为0X80。示例波形如下图所示，第一个发送的数据是0XD0，第一个数据的高7位是从机地址，最低位是读写位为写（值为0），所以第一个数据为：0X68 << 1 | 0 = 0XD0，然后依次发送寄存器地址0X6B和数据0X80。

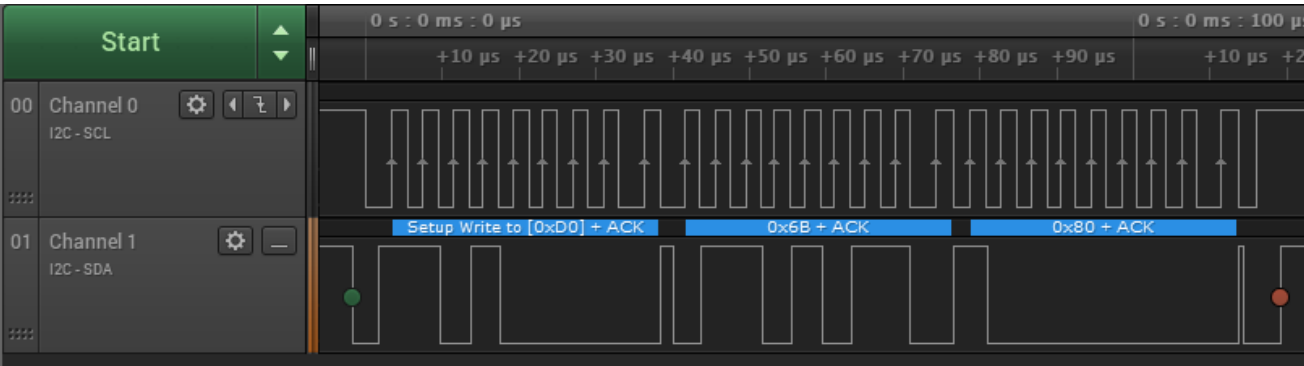


图4.2.1-2 I2C发送数据波形示例

使用rt\_i2c\_master\_send()发送数据

函数原型:

```
1  rt_size_t rt_i2c_master_send(struct rt_i2c_bus_device *bus,
2                               rt_uint16_t             addr,
3                               rt_uint16_t             flags,
4                               const rt_uint8_t         *buf,
5                               rt_uint32_t             count)
6
```

参数	描述
bus	I2C总线设备句柄
addr	从机地址，不包含读写位
flags	标志，读写标志为写。只支持10位地址选择RT_I2C_ADDR_10BIT
buf	指向发送数据的指针
count	发送数据字节数

**函数返回：**成功发送的数据字节数。

此函数是对 `rt_i2c_transfer()` 的简单封装。

本文示例代码底层驱动 `drv_mpu6050.c` 发送数据源码如下：

```
1  #define MPU6050_ADDR          0X68
2
3  //写mpu6050单个寄存器
4  //reg:寄存器地址
5  //data:数据
6  //返回值: 0,正常 / -1,错误代码
7  rt_err_t mpu6050_write_reg(rt_uint8_t reg, rt_uint8_t data)
8  {
9      rt_uint8_t buf[2];
10
```

```

11     buf[0] = reg;
12     buf[1] = data;
13
14     if (rt_i2c_master_send(mpu6050_i2c_bus, MPU6050_ADDR, 0, buf, 2) == 2)
15     {
16         return RT_EOK;
17     }
18     else
19     {
20         return -RT_ERROR;
21     }
22 }

```

## 4.2.2 接收数据

用户可以调用I2C设备驱动接口 `rt_i2c_master_recv()` 或者 `rt_i2c_transfer()` 接受数据。函数调用关系如下：

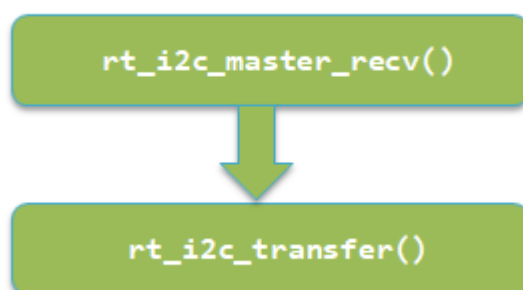


图4.2.2-1 接收数据函数调用关系

本文示例代码 `drv_mpu6050.c` 中的 `mpu6050_read_reg()` 函数是MCU从MPU6050寄存器读取数据，此函数的实现同样有2种方式，分别调用了I2C设备驱动接口 `rt_i2c_transfer()` 和 `rt_i2c_master_recv()` 实现。

读MPU6050某个寄存器，主机首先发送从机地址MPU6050\_ADDR、读写标志 R/W 为 RT\_I2C\_WR (0 为写，1 为读)、从机寄存器地址reg之后才能开始读设备。然后发送从机地址MPU6050\_ADDR、读写标志 R/W 为 RT\_I2C\_RD (0 为写，1 为读)、保存读取数据指针。

### 使用rt\_i2c\_transfer()接收数据

本文示例代码底层驱动 `drv_mpu6050.c` 接收数据源码如下：

```

1  #define MPU6050_ADDR          0X68
2
3  //读取寄存器数据
4  //reg:要读取的寄存器地址
5  //len:要读取的数据字节数
6  //buf:读取到的数据存储区
7  //返回值: 0,正常 / -1,错误代码
8  rt_err_t mpu6050_read_reg(rt_uint8_t reg, rt_uint8_t len, rt_uint8_t *buf)
9  {
10     struct rt_i2c_msg msgs[2];
11
12     msgs[0].addr = MPU6050_ADDR; /* 从机地址 */
13     msgs[0].flags = RT_I2C_WR; /* 写标志 */

```

```

14  msgs[0].buf      = &reg;                /* 从机寄存器地址 */
15  msgs[0].len      = 1;                    /* 发送数据字节数 */
16
17  msgs[1].addr     = MPU6050_ADDR;         /* 从机地址 */
18  msgs[1].flags    = RT_I2C_RD;           /* 读标志 */
19  msgs[1].buf      = buf;                  /* 读取数据指针 */
20  msgs[1].len      = len;                  /* 读取数据字节数 */
21
22  if (rt_i2c_transfer(mpu6050_i2c_bus, msgs, 2) == 2)
23  {
24      return RT_EOK;
25  }
26  else
27  {
28      return -RT_ERROR;
29  }
30  }

```

以本文示例代码其中一次调用rt\_i2c\_transfer()接收数据为例，从机MPU6050地址16进制值为0X68，寄存器地址reg 16进制值为0X75。示例波形如下图所示，第一个发送的数据是0XD0，第一个数据的高7位是从机地址，最低位是读写位是写（值为0），所以第一个数据值为：0X68 << 1 | 0 = 0XD0，然后发送寄存器地址0X75。第二次发送的第一个数据为0XD1，读写位是读（值为1），值为：0X68 << 1 | 1 = 0XD1，然后收到读取到的数据0X68。

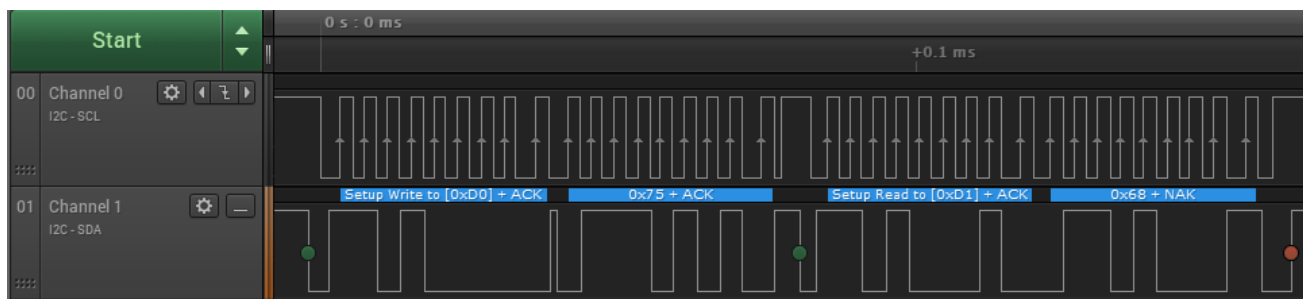


图4.2.2-2 I2C发送数据波形示例

### 使用 rt i2c master recv()接收数据

### 函数原型:

[illegible]

参数	描述
bus	I2C总线设备句柄
addr	从机地址，不包含读写位
flags	标志，读写标志为读，只支持10位地址选择RT_I2C_ADDR_10BIT
buf	接受数据指针
count	接收数据字节数

**函数返回：**成功接收的数据字节数。

此函数是对 `rt_i2c_transfer()` 的简单封装，只能读取数据（接收数据）。

本文示例代码底层驱动 `drv_mpu6050.c` 接收数据源码如下：

```
1  #define MPU6050_ADDR          0X68
2
3  //读取寄存器数据
4  //reg:要读取的寄存器地址
5  //len:要读取的数据字节数
6  //buf:读取到的数据存储区
7  //返回值：0,正常 / -1,错误代码
8  rt_err_t mpu6050_read_reg(rt_uint8_t reg, rt_uint8_t len, rt_uint8_t *buf)
9  {
10     if (rt_i2c_master_send(mpu6050_i2c_bus, MPU6050_ADDR, 0, &reg, 1) == 1)
11     {
12         if (rt_i2c_master_recv(mpu6050_i2c_bus, MPU6050_ADDR, 0, buf, len) == len)
13         {
14             return RT_EOK;
15         }
16         else
17         {
18             return -RT_ERROR;
19         }
20     }
21     else
22     {
23         return -RT_ERROR;
24     }
25 }
26 }
```

### 4.3 I2C设备驱动应用

通常I2C接口芯片的只读寄存器分为2种情况，一种是单一功能寄存器，另一种是地址连续，功能相近的寄存器。例如MPU6050的寄存器0X3B、0X3C、0X3D、0X3E、0X3F、0X40依次存放的是三轴加速度X、Y、Z轴的高8位、低8位数据。

本文示例代码底层驱动 `drv_mpu6050.c` 使用 `mpu6050_read_reg()` 函数读取MPU6050的3轴加速度数据：



```
1  #define MPU_ACCEL_XOUTH_REG    0X3B    //加速度值,X轴高8位寄存器
2
3  //得到加速度值(原始值)
4  //gx,gy,gz:陀螺仪x,y,z轴的原始读数(带符号)
5  //返回值:0,成功/ -1,错误代码
6  rt_err_t mpu6050_accelerometer_get(rt_int16_t *ax, rt_int16_t *ay, rt_int16_t *az)
7  {
8      rt_uint8_t buf[6], ret;
9
10     ret = mpu6050_read_reg(MPU_ACCEL_XOUTH_REG, 6, buf);
11     if (ret == 0)
12     {
13         *ax = ((rt_uint16_t)buf[0] << 8) | buf[1];
14         *ay = ((rt_uint16_t)buf[2] << 8) | buf[3];
15         *az = ((rt_uint16_t)buf[4] << 8) | buf[5];
16
17         return RT_EOK;
18     }
19     else
20     {
21         return -RT_ERROR;
22     }
23 }
24
```

## 5 参考

### 本文所有相关的API

API	头文件
rt_i2c_transfer()	rt-thread\components\drivers\include\drivers\i2c.h
rt_i2c_master_send()	rt-thread\components\drivers\include\drivers\i2c.h
rt_i2c_master_recv()	rt-thread\components\drivers\include\drivers\i2c.h
mpu6050_hw_init()	drv_mpu6050.h
mpu6050_write_reg()	drv_mpu6050.h
mpu6050_read_reg()	drv_mpu6050.h
mpu6050_temperature_get()	drv_mpu6050.h
mpu6050_gyroscope_get()	drv_mpu6050.h
mpu6050_accelerometer_get()	drv_mpu6050.h

## 讨论和反馈

欢迎登陆[RT-Thread开发者社区](#)进行交流

## 参考文献

[RT-Thread编程手册](#)

## 关注RT-Thread公众号



扫一扫关注RT-Thread微信公众号