
I2C 设备应用笔记

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019



WWW.RT-THREAD.ORG

Friday 28th September, 2018

目录

目录	i
1 本文的目的和结构	1
1.1 本文的目的和背景	1
1.2 本文的结构	1
2 I2C 设备驱动框架简介	1
3 运行 I2C 设备驱动示例代码	2
3.1 示例代码软硬件平台	2
3.2 启用 I2C 设备驱动	4
3.3 运行示例代码	6
4 I2C 设备驱动接口详解	7
4.1 查找设备	8
4.2 数据传输	9
4.2.1. 发送数据	10
4.2.2. 接收数据	13
4.3 I2C 设备驱动应用	16

!!! abstract “摘要” 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。

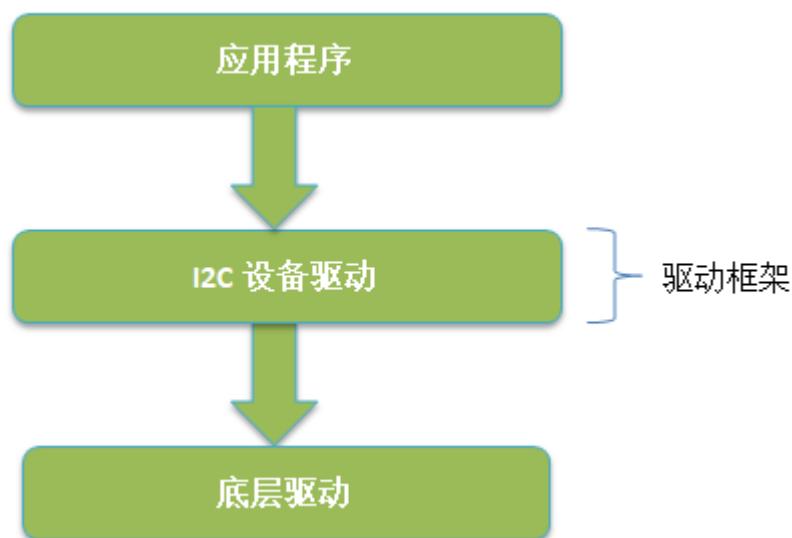


图 1: RT-Thread I2C 设备驱动框架

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](#)
5. [I2C 示例代码](#)

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。

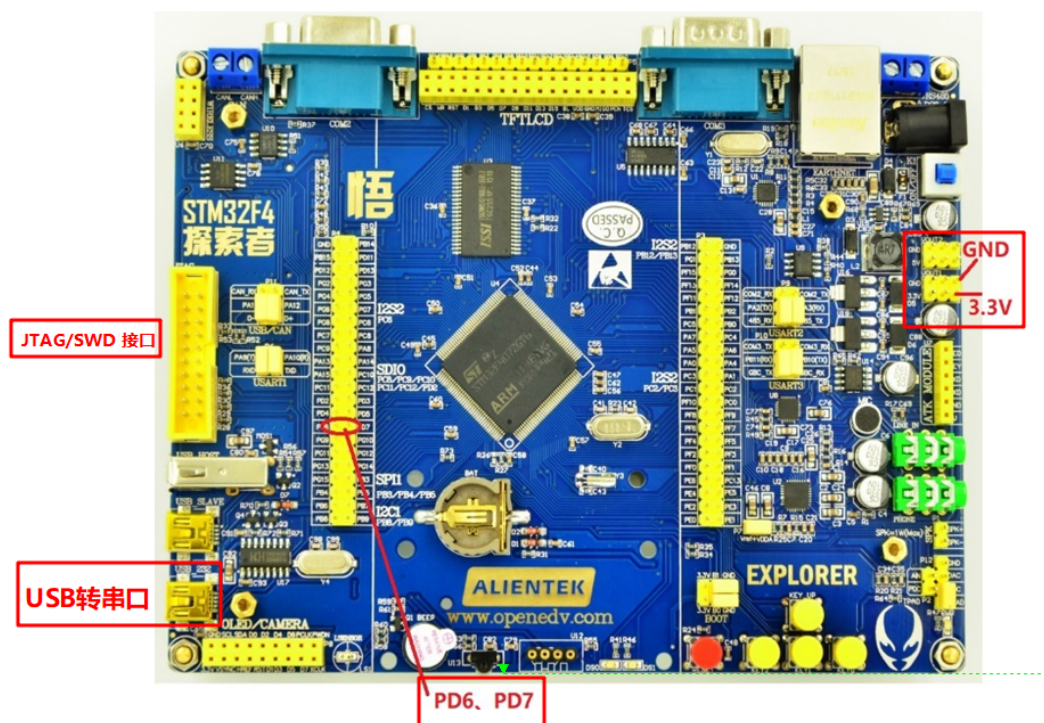


图 2: 正点原子开发板

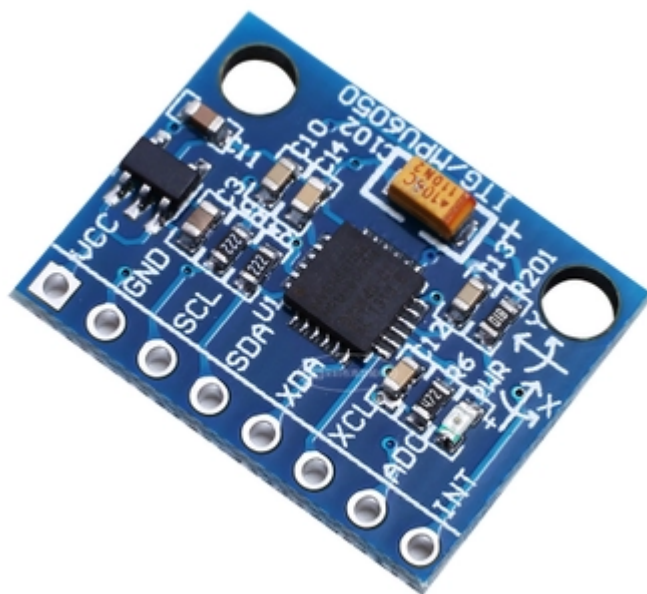


图 3: GY521 模块

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。

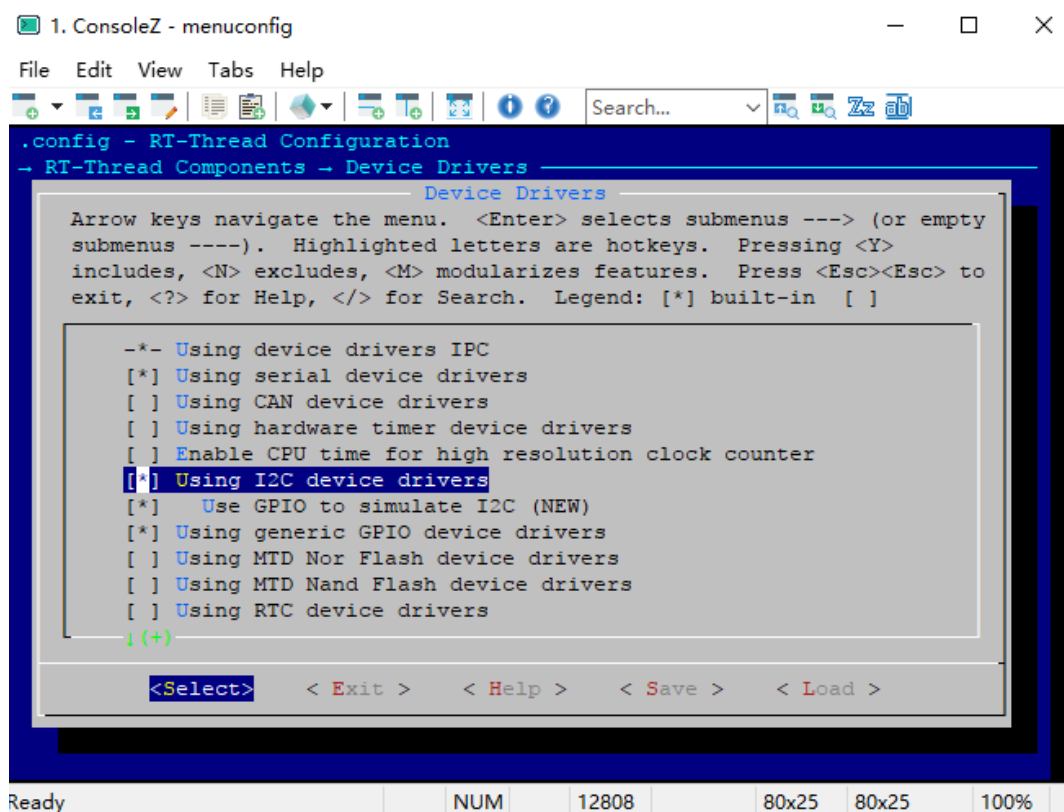


图 4: 使用 `menuconfig` 开启 `i2c`

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`。

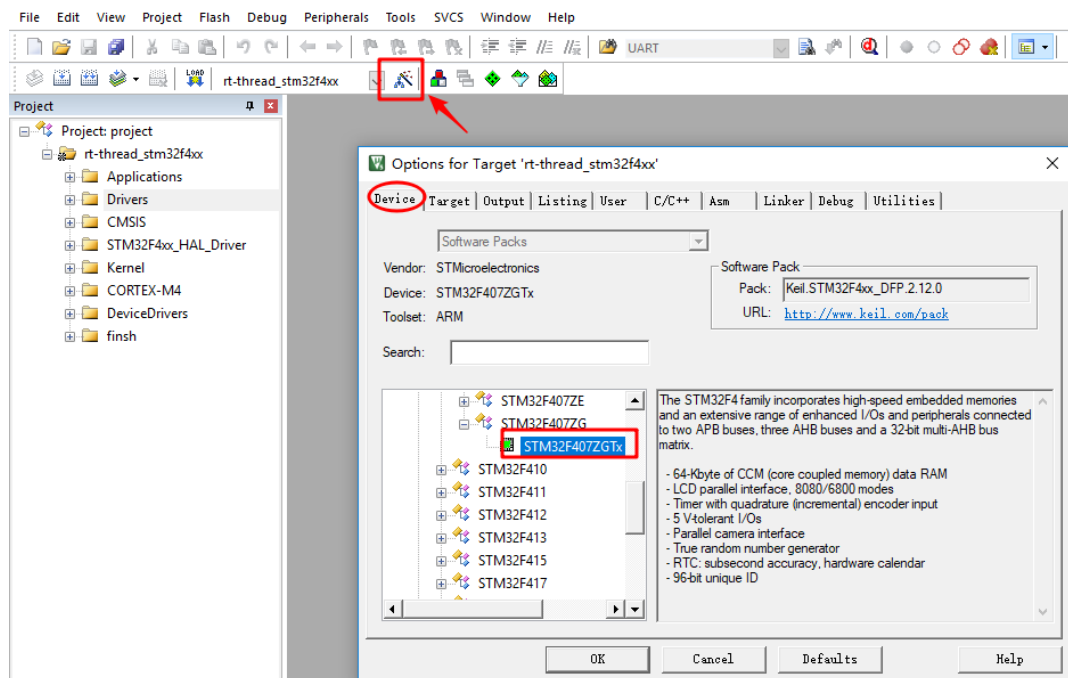


图 5: 修改 MCU

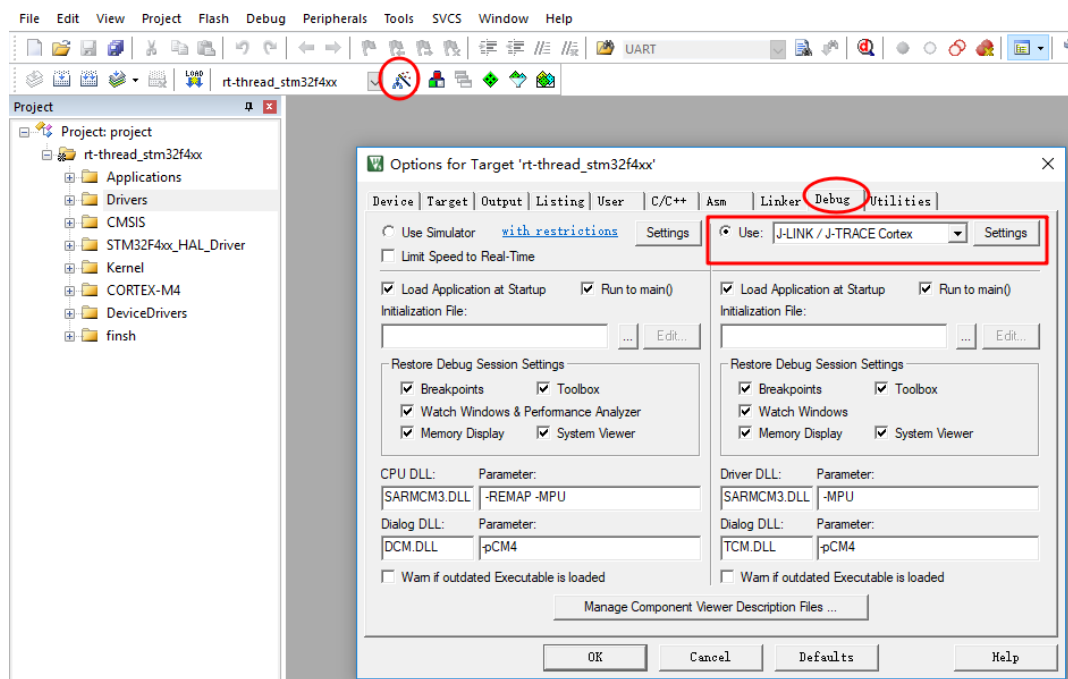
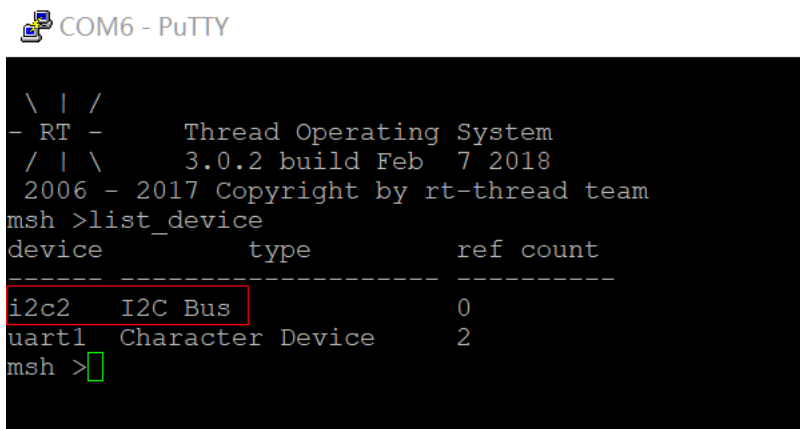


图 6: 修改调试选项

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:



```
COM6 - PuTTY

\ | /
- 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 7: 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:

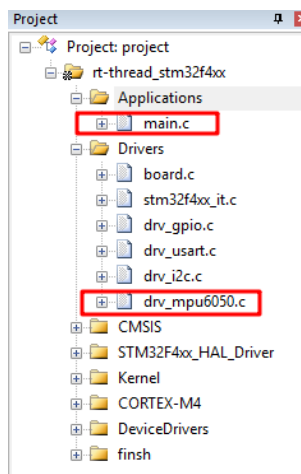
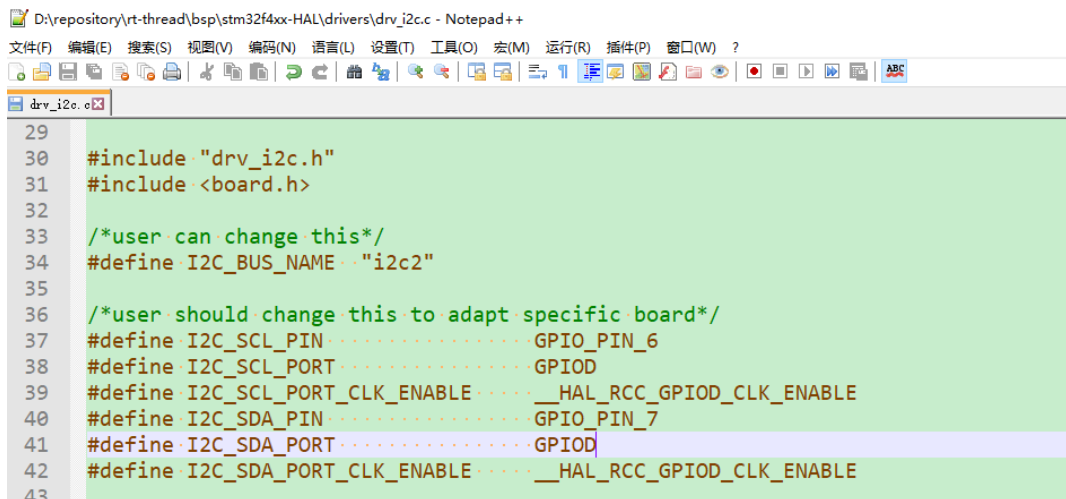


图 8: 添加驱动

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:



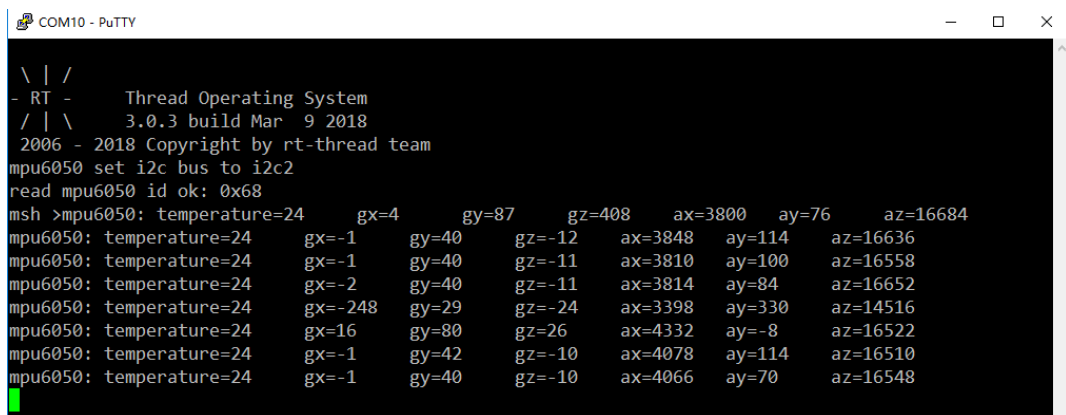
```

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 9: *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 10: 终端打印信息

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：

```
#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 */
static struct rt_i2c_bus_device *mpu6050_i2c_bus; /* I2C 设备句柄 */
...
...

int mpu6050_hw_init(void)
{
    rt_uint8_t res;

    mpu6050_i2c_bus = rt_i2c_bus_device_find(MPU6050_I2CBUS_NAME); /* 查
        找 I2C 设备 */

    if (mpu6050_i2c_bus == RT_NULL)
    {
        MPUDEBUG("can't find mpu6050 %s device\r\n",MPU6050_I2CBUS_NAME);
        return -RT_ERROR;
    }

    ...
}
```

```
... ..
}
```

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:

```
rt_size_t rt_i2c_transfer(struct rt_i2c_bus_device *bus,
                          struct rt_i2c_msg      msgs[],
                          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

```
struct rt_i2c_msg
{
    rt_uint16_t addr;    // 从机地址
    rt_uint16_t flags;   // 标志，读、写等
    rt_uint16_t len;     // 读写数据字节数
    rt_uint8_t  *buf;    // 读写数据指针
}
```

`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。

```

#define RT_I2C_WR          0x0000
#define RT_I2C_RD          (1u << 0)
#define RT_I2C_ADDR_10BIT (1u << 2) /* this is a ten bit chip address */
#define RT_I2C_NO_START    (1u << 4)
#define RT_I2C_IGNORE_NACK (1u << 5)
#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：

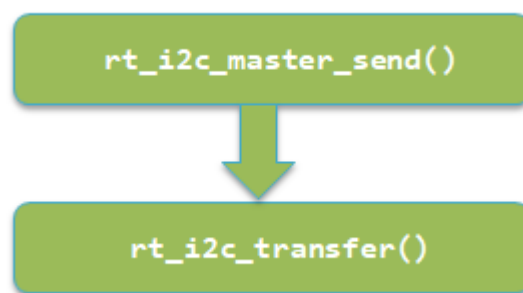


图 11: 发送数据函数调用关系

`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。

1) 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：

```

#define MPU6050_ADDR          0X68

// 写 mpu6050 单个寄存器
// reg: 寄存器地址

```

```

//data: 数据
// 返回值: 0, 正常 / -1, 错误代码
rt_err_t mpu6050_write_reg(rt_uint8_t reg, rt_uint8_t data)
{
    struct rt_i2c_msg msgs;
    rt_uint8_t buf[2] = {reg, data};

    msgs.addr = MPU6050_ADDR;    /* 从机地址 */
    msgs.flags = RT_I2C_WR;      /* 写标志 */
    msgs.buf = buf;              /* 发送数据指针 */
    msgs.len = 2;

    if (rt_i2c_transfer(mpu6050_i2c_bus, &msgs, 1) == 1)
    {
        return RT_EOK;
    }
    else
    {
        return -RT_ERROR;
    }
}

```

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。

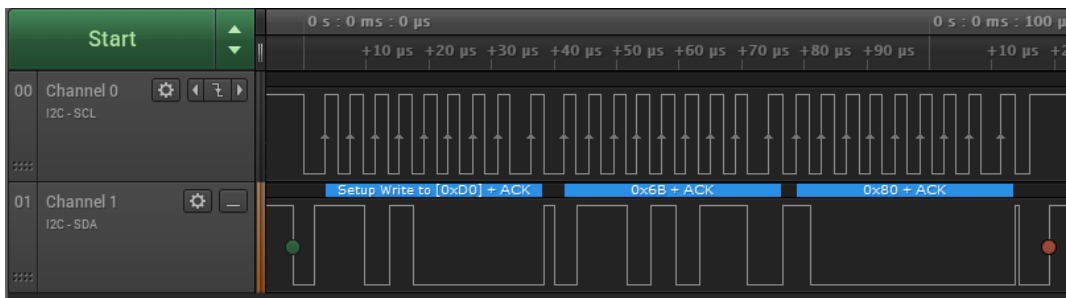


图 12: I2C 发送数据波形示例

2) 使用 `rt_i2c_master_send()` 发送数据

函数原型:

```

rt_size_t rt_i2c_master_send(struct rt_i2c_bus_device *bus,
                             rt_uint16_t                addr,

```

<code>rt_uint16_t</code>	<code>flags,</code>
<code>const rt_uint8_t</code>	<code>*buf,</code>
<code>rt_uint32_t</code>	<code>count)</code>

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：

```
#define MPU6050_ADDR          0X68

// 写 mpu6050 单个寄存器
//reg: 寄存器地址
//data: 数据
// 返回值: 0, 正常 / -1, 错误代码
rt_err_t mpu6050_write_reg(rt_uint8_t reg, rt_uint8_t data)
{
    rt_uint8_t buf[2];

    buf[0] = reg;
    buf[1] = data;

    if (rt_i2c_master_send(mpu6050_i2c_bus, MPU6050_ADDR, 0, buf ,2) ==
        2)
    {
        return RT_EOK;
    }
    else
    {
        return -RT_ERROR;
    }
}
```

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：

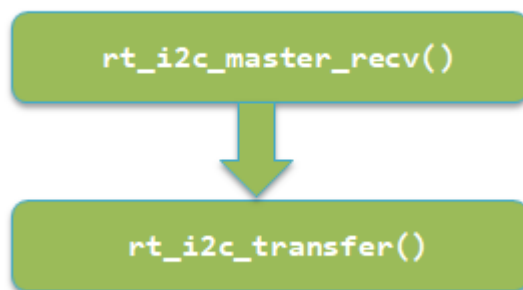


图 13: 接收数据函数调用关系

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。

1) 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：

```
#define MPU6050_ADDR          0X68

// 读取寄存器数据
//reg: 要读取的寄存器地址
//len: 要读取的数据字节数
//buf: 读取到的数据存储区
// 返回值: 0, 正常 / -1, 错误代码
rt_err_t mpu6050_read_reg(rt_uint8_t reg, rt_uint8_t len, rt_uint8_t *buf)
{
    struct rt_i2c_msg msgs[2];

    msgs[0].addr = MPU6050_ADDR;    /* 从机地址 */
    msgs[0].flags = RT_I2C_WR;      /* 写标志 */
```

```

    msgs[0].buf    = &reg;          /* 从机寄存器地址 */
    msgs[0].len    = 1;             /* 发送数据字节数 */

    msgs[1].addr   = MPU6050_ADDR;  /* 从机地址 */
    msgs[1].flags  = RT_I2C_RD;     /* 读标志 */
    msgs[1].buf    = buf;           /* 读取数据指针 */
    msgs[1].len    = len;           /* 读取数据字节数 */

    if (rt_i2c_transfer(mpu6050_i2c_bus, msgs, 2) == 2)
    {
        return RT_EOK;
    }
    else
    {
        return -RT_ERROR;
    }
}

```

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。

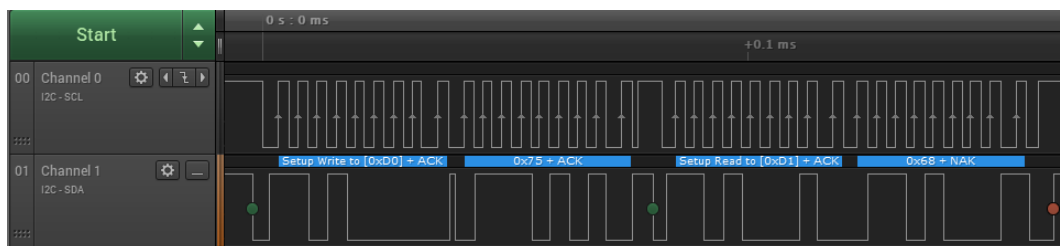


图 14: I2C 发送数据波形示例

2) 使用 `rt_i2c_master_recv()` 接收数据

函数原型:

```

rt_size_t rt_i2c_master_recv(struct rt_i2c_bus_device *bus,
                             rt_uint16_t             addr,
                             rt_uint16_t             flags,
                             rt_uint8_t               *buf,
                             rt_uint32_t             count)

```


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：

```
#define MPU6050_ADDR          0X68

// 读取寄存器数据
//reg: 要读取的寄存器地址
//len: 要读取的数据字节数
//buf: 读取到的数据存储区
// 返回值: 0, 正常 / -1, 错误代码
rt_err_t mpu6050_read_reg(rt_uint8_t reg, rt_uint8_t len, rt_uint8_t *buf)
{
    if (rt_i2c_master_send(mpu6050_i2c_bus, MPU6050_ADDR, 0, &reg, 1) ==
        1)
    {
        if (rt_i2c_master_recv(mpu6050_i2c_bus, MPU6050_ADDR, 0, buf, len)
            == len)
        {
            return RT_EOK;
        }
        else
        {
            return -RT_ERROR;
        }
    }
    else
    {
        return -RT_ERROR;
    }
}
```

```
}
```

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：

```
#define MPU_ACCEL_XOUTH_REG    0X3B    // 加速度值，X 轴高 8 位寄存器

// 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 / -1，错误代码
rt_err_t mpu6050_accelerometer_get(rt_int16_t *ax, rt_int16_t *ay,
    rt_int16_t *az)
{
    rt_uint8_t buf[6], ret;

    ret = mpu6050_read_reg(MPU_ACCEL_XOUTH_REG, 6, buf);
    if (ret == 0)
    {
        *ax = ((rt_uint16_t)buf[0] << 8) | buf[1];
        *ay = ((rt_uint16_t)buf[2] << 8) | buf[3];
        *az = ((rt_uint16_t)buf[4] << 8) | buf[5];

        return RT_EOK;
    }
    else
    {
        return -RT_ERROR;
    }
}
```