

1. 目录

HCRTOS i2c userguide

1. 目录
2. 文档履历
3. 概述
 - 3.1 编写目的
 - 3.2 读者对象
4. 模块介绍
5. 模块接口说明
 - 5.1 gpio-i2c的使用
 - 5.1.1 设备树的配置
 - 5.1.2 menuconfig配置
 - 5.1.3 编译和生成
 - 5.1.4 代码使用
 - 5.2 硬件i2c的使用
 - 5.2.1 设备树的配置
 - 5.2.2 menuconfig配置
 - 5.2.3 编译和生成
 - 5.2.4 代码使用
 - 5.3 在中断处理函数中读写i2c设备
6. 常见问题

2. 文档履历

版本号	日期	制/修订人	制/修订记录
1.0	2023.04.06	邱浩佳	新增文档说明

3. 概述

3.1 编写目的

指导i2c功能的开发和使用。

3.2 读者对象

软件开发工程师和技术支持工程师。

4. 模块介绍

- 本驱动存放在SDK的路径为：hcrtos/components/kernel/source/drivers/i2c。
- 该目录下：hc_i2c_bitbang.c为gpio-i2c驱动；hc-i2c.c为硬件i2c驱动；hdmi_i2c_slave.c与hdmi相关，具体查看hdmi相关文档。
- gpio-i2c为gpio模拟i2c波形，波特率只有100k，硬件i2c支持常用的波特率选择：100k，200k和400k。
- 本模块读写i2c的方式和Linux下进行i2c的读写操作一致，**需要注意的一点是不可以在中断中读写i2c**，可以通过创建一个work去完成读写，具体操作可以参考该文件里读写i2c的操作：
hcrtos/components/kernel/source/drivers/input/tp/hy46xx/hy46xx_ts.c。
- 硬件i2c和gpio-i2c都提供四组。
- i2c访问的地址为目标器件的七位地址。

5. 模块接口说明

5.1 gpio-i2c的使用

5.1.1 设备树的配置

在板子设备树添加如下所示节点，gpio-i2c有0~3组可以配置，下面以第0组为例子。

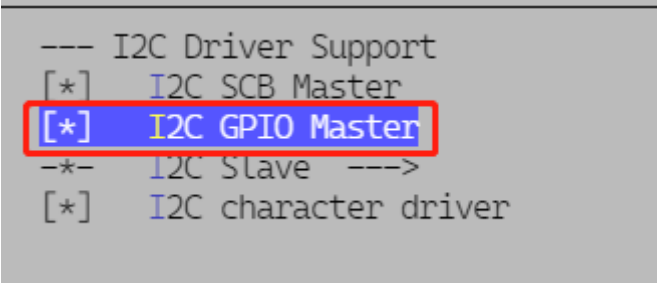
```
1  gpio-i2c@0 {  
2      sda-pinmux = <PINPAD_L19>;  
3      scl-pinmux = <PINPAD_L20>;  
4      status = "okay";  
5      simulate;  
6  };
```

注意：sda-pinmux和scl-pinmux所配置的引脚要和硬件电路与器件连接的引脚对应，注意线序。

5.1.2 menuconfig配置

在sdk根目录根据下面路径进行配置，输入make menuconfig进行配置gpio-i2c驱动。

```
1 Location:
2   -> Components
3     -> kernel (BR2_PACKAGE_KERNEL [=y])
4       -> Drivers
5         -> I2C Driver Support (CONFIG_I2C [=y])
```



```
--- I2C Driver Support
[*]   I2C SCB Master
[*]   I2C GPIO Master
--*-- I2C Slave  --->
[*]   I2C character driver
```

5.1.3 编译和生成

配置完设备树和驱动后，输入make kernel-rebuild all进行编译，烧录编译生成的文件，在串口终端输入对应的命令，即可查看到对应的节点。

```

hc1600a@dbc3000v10#
hc1600a@dbc3000v10# nsh
hc1600a@dbc3000v10(nsh)# ls
/:
 dev/
hc1600a@dbc3000v10(nsh)# cd dev
hc1600a@dbc3000v10(nsh)# ls
/dev:
 auddec
 audsink
 avsync0
 avsync1
 bus/
 dis
 efuse
 fb0
 ge
 gpio-i2c0
 input/
 lvds
 mmz
 mtd0
 mtd1
 mtd2
 mtd3
 mtd4
 mtdblock0
 mtdblock1
 mtdblock2
 mtdblock3
 mtdblock4
 persistentmem
 pq
 sndC0i2si
 sndC0i2so
 sndC1spin
 tv_decoder
 uart2
 uart_dummy
 viddec
 vidsink
hc1600a@dbc3000v10(nsh)# █

```

5.1.4 代码使用

gpio-i2c提供和Linux一样读写的i2c接口，需要构建i2c_transfer_s结构体。

```

1  int fd;
2  char *writebuf, int writelen, char *readbuf, int readlen;
3
4  fd = open("/dev/gpio-i2c0", O_RDWR);
5  if(fd < 0)
6      return -1;
7
8  struct i2c_transfer_s xfer_read;
9  struct i2c_msg_s i2c_msg_read[2] = {0};
10
11  i2c_msg_read[0].addr = (uint8_t)addr;
12  i2c_msg_read[0].flags = 0x0;           //写
13  i2c_msg_read[0].buffer = writebuf;
14  i2c_msg_read[0].length = writelen;
15

```

```

16 i2c_msg_read[1].addr = (uint8_t)addr;
17 i2c_msg_read[1].flags = 0x1;           //读
18 i2c_msg_read[1].buffer = readbuf;
19 i2c_msg_read[1].length = readlen;
20
21 xfer_read.msgv = i2c_msg_read;
22 xfer_read.msgc = 2;
23
24 ret = ioctl(fd, I2CIOCTransfer, &xfer_read);
25
26 if (ret < 0)
27     dev_err(&client->dev, "%s: i2c read error.\n", __func__);

```

5.2 硬件i2c的使用

5.2.1 设备树的配置

在板子设备树添加如下所示节点，硬件i2c有0~3组可以配置，下面以第1组为例子。

```

1 i2c@1 {
2     pinmux-active = <PINPAD_B23 10 PINPAD_B24 10>;
3     devpath = "/dev/i2c1";
4     baudrate = <100000>;
5     mode = "master";
6     status = "okay";
7 };

```

pinmux-active: PINPAD_B23代表所使用的io引脚，10表示将该引脚配置成i2c1_scl，并不是所有的引脚配置成i2c1的值是10，具体配置根据所使用引脚参考 [components/kernel/source/include/uapi/hcuapi/pinmux/hc16xx_pinmux.h](#)。

devpath: i2c设备节点生成的路径，用于代码访问。

baudrate: i2c的速率，可选的有100k、200k和400k。

mode: 为master模式。

status: okay为启用，disabled为不启用。

5.2.2 menuconfig配置

i2c的驱动默认开启。

```

1 Location:
2     -> Components
3         -> kernel (BR2_PACKAGE_KERNEL [=y])
4             -> Drivers
5                 -> I2C Driver Support (CONFIG_I2C [=y])

```

```
--- I2C Driver Support
[*] I2C SCB Master
[*] I2C GPIO Master
-* I2C Slave --->
[*] I2C character driver
```

5.2.3 编译和生成

配置完设备树和驱动后，输入make kernel-rebuild all进行编译，烧录编译生成的文件，在串口终端输入对应的命令，即可查看到对应的节点。

```
hc1600a@dbc3000v10#
hc1600a@dbc3000v10# nsh
hc1600a@dbc3000v10(nsh)# cd dev
hc1600a@dbc3000v10(nsh)# ls
/dev:
auddec
audsink
avsync0
avsync1
bus/
dis
efuse
fb0
ge
gpio-i2c0
i2c3
input/
lvds
mmz
mtd0
mtd1
mtd2
mtd3
mtd4
mtdblock0
mtdblock1
mtdblock2
mtdblock3
mtdblock4
persistentmem
pq
sndC0i2si
sndC0i2so
sndClspin
tv_decoder
uart2
uart_dummy
viddec
vidsink
hc1600a@dbc3000v10(nsh)#
```

5.2.4 代码使用

```
1  int fd;
2  char *writebuf, int writelen, char *readbuf, int readlen;
3
4  fd = open("/dev/i2c3", O_RDWR);
5  if (fd < 0)
6      return -1;
7
8  struct i2c_transfer_s xfer_read;
9  struct i2c_msg_s i2c_msg_read[2] = {0};
10
11 i2c_msg_read[0].addr = (uint8_t)addr;
12 i2c_msg_read[0].flags = 0x0;          //写
13 i2c_msg_read[0].buffer = writebuf;
14 i2c_msg_read[0].length = writelen;
15
16 i2c_msg_read[1].addr = (uint8_t)addr;
17 i2c_msg_read[1].flags = 0x1;          //读
18 i2c_msg_read[1].buffer = readbuf;
19 i2c_msg_read[1].length = readlen;
20
21 xfer_read.msgv = i2c_msg_read;
22 xfer_read.msgc = 2;
23
24 ret = ioctl(fd, I2CIOCTransfer, &xfer_read);
25
26 if (ret < 0)
27     dev_err(&client->dev, "f%s: i2c read error.\n", __func__);
```

5.3 在中断处理函数中读写i2c设备

i2c的读写会涉及中断，不能在其它中断中直接调用i2c的读写，会造成死锁现象。需要进行一下特殊处理，创建个高优先级的work去完成i2c的读取。下面以一个gpio中断读取i2c为例子，代码存放路径：
hcrtos/components/kernel/source/drivers/input/tp/hy46xx/hy46xx_ts.c

```
1  /*高优先级work*/
2  static void hy46xx_work(void *param)
3  {
4      struct hy46xx_ts_data *hy46xx_ts = (struct hy46xx_ts_data*)param;
5
6      hy46xx_read_Touchdata(hy46xx_ts);          //读写i2c的操作
7      hy46xx_report_value(hy46xx_ts);
8
9      return;
10 }
11
12 /*gpio中断服务函数: */
13 static void hy46xx_irq(uint32_t param)
14 {
15     struct hy46xx_ts_data *hy46xx_ts = (struct hy46xx_ts_data*)param;
16
17     if (work_available(&hy46xx_ts->work)) {
```

```
18         work_queue(HPWORK, &hy46xx_ts->work, hy46xx_work, (void
19         *)hy46xx_ts, 0);
20     }
```

6. 常见问题

Q：配置gpio-i2c有波形，但器件一直没有ack。

A：检查设备树中gpio-i2c节点的i2c引脚顺序会不会搞反了。

Q：硬件i2c配置完，没有发出波形。

A：[检查将io配置成i2c的值是否正确](#)，或者有无存在引脚复用。