RK 平台 I2C 应用说明

文档版本	日期	作者	说明
1.0	2016/12	Chen Qiwei	创建

1.节点权限确认

RK 平台支持的 I2C 节点位于/dev 下：
ll /dev/i2c*

```
crw------- root     root      89,   0 2011-01-01 20:00 i2c-0
crw------- root     root      89,   1 2011-01-01 20:00 i2c-1
crw------- root     root      89,   2 2011-01-01 20:00 i2c-2
crw------- root     root      89,   3 2011-01-01 20:00 i2c-3
crw------- root     root      89,   4 2011-01-01 20:00 i2c-4
```

默认没有访问权限，如果要在应用中访问，请联系 RYD 修改。

2.应用访问 I2C 节点的参考例程（以 I2C0 为例）

```
#include <stdio.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>

#include "ql_commfunc.h"
#define I2C_FILE_NAME "/dev/i2c-0"


static int get_i2c_register(int file,
                            unsigned short addr,
                            unsigned char reg,
```

```
                        unsigned char *val,
                           int len) {
    //unsigned char inbuf, outbuf;
    unsigned char outbuf;
    struct i2c_rdwr_ioctl_data packets;
    struct i2c_msg messages[2];

    /*
     * In order to read a register, we first do a "dummy write" by writing
     * 0 bytes to the register we want to read from.  This is similar to
     * the packet in set_i2c_register, except it's 1 byte rather than 2.
     */
    outbuf = reg;
    messages[0].addr  = addr;
    messages[0].flags = 0;
    messages[0].len   = sizeof(outbuf);
    messages[0].buf   = &outbuf;

    /* The data will get returned in this structure */
    messages[1].addr  = addr;
    messages[1].flags = I2C_M_RD/* | I2C_M_NOSTART*/;
    messages[1].len   = len;
    messages[1].buf   = val;

    /* Send the request to the kernel and get the result back */
    packets.msgs      = messages;
    packets.nmsgs     = 2;
    if(ioctl(file, I2C_RDWR, &packets) < 0) {
        perror("Unable to send data");
        return 1;
    }
    //*val = inbuf;

    return 0;
}


static int set_i2c_register(int file,
                        unsigned char addr,
                        unsigned char reg,
                        unsigned char *value,
                           int len) {

    unsigned char *outbuf = (unsigned char *)malloc(sizeof(unsigned char)*(len+1));
```

```
        if(outbuf==NULL)
        {
                perror("MALLOC");
                return -1;
        }
    struct i2c_rdwr_ioctl_data packets;
    struct i2c_msg messages[1];

    messages[0].addr  = addr;
    messages[0].flags = 0;
    messages[0].len   = len+1;
    messages[0].buf   = outbuf;


    /* The first byte indicates which register we'll write */
    outbuf[0] = reg;

    /*
     * The second byte indicates the value to write.  Note that for many
     * devices, we can write multiple, sequential registers at once by
     * simply making outbuf bigger.
     */
//    outbuf[1] = value;
memcpy(outbuf+1, value, len);

    /* Transfer the i2c packets to the kernel and verify it worked */
    packets.msgs  = messages;
    packets.nmsgs = 1;
    if(ioctl(file, I2C_RDWR, &packets) < 0) {
        perror("Unable to send data");
        return 1;
    }

    return 0;
}



int ql_get_i2c_register(unsigned short slave_addr, unsigned char reg, unsigned char *outbuf, int
buf_len )
{

        int fd;
        // Open a connection to the I2C userspace control file.
```

```
    if ((fd = open(I2C_FILE_NAME, O_RDWR)) < 0) {
        PERROR("Unable to open i2c control file");
        return -1;
        //exit(1);
    }
    if(get_i2c_register(fd, slave_addr, reg, outbuf, buf_len))
    {
        DEBUG("");
        return -2;
    }
    close(fd);
    return 0;
}

int ql_set_i2c_register(unsigned short slave_addr, unsigned char reg, unsigned char *inbuf, int
buf_len )
{
    int fd;
    // Open a connection to the I2C userspace control file.
    if ((fd = open(I2C_FILE_NAME, O_RDWR)) < 0) {
        PERROR("Unable to open i2c control file");
        return -1;
        //exit(1);
    }
    if(set_i2c_register(fd, slave_addr, reg, inbuf, buf_len))
    {
        DEBUG("");
        return -4;
    }
    close(fd);
    return 0;

}
```