**FM发射模块QN8027软件android实现分析**

一，kernel层中的驱动：(主要为厂家提供，主要配置对应的I2C口线)

由三个文件组成：

1，  Makefile   /\*驱动的编译文件，让该驱动被编译到内核中去\*/

2，  qn8027.c   /\*驱动的源文件，厂家提供，具体实现不分析，这里主要关注如何与JAVA层的通讯\*/

3，  qn8027.h  /\*驱动头文件，厂家提供。定义通讯的IO\*/

Makefile文档：

include $(srctree)/drivers/misc/mediatek/Makefile.custom

obj-y += qn8027.o

qn8027.c文档：

#include<linux/disp\_assert\_layer.h>

#include<linux/miscdevice.h>

#include<linux/fs.h>

#include<linux/file.h>

#include<linux/cdev.h>

#include<asm/tlbflush.h>

#include<asm/page.h>

#include<linux/slab.h>

//#include<linux/autoconf.h>

#include<linux/module.h>

#include<linux/mm.h>

#include<linux/init.h>

#include<linux/fb.h>

#include<linux/delay.h>

#include<linux/device.h>

#include<linux/platform\_device.h>

#include<linux/dma-mapping.h>

#include<linux/earlysuspend.h>

#include<linux/kthread.h>

#include<linux/rtpm\_prio.h>

#include<asm/uaccess.h>

#include<asm/atomic.h>

#include<asm/mach-types.h>

#include<asm/cacheflush.h>

#include<asm/io.h>

#include<mach/dma.h>

#include<mach/irqs.h>

#include<linux/vmalloc.h>

#include<mach/mt\_gpio.h>

#include"cust\_gpio\_usage.h"

#include"qn8027.h"

staticsize\_t qn8027\_iic\_log\_on = false;

#defineQN8027\_IIC\_LOG(fmt, arg...) \

        do { \

                if (qn8027\_iic\_log\_on){printk("[qn8027]%s,#%d ", \_\_func\_\_, \_\_LINE\_\_); printk(fmt, ##arg);}\

        }while (0)

#defineQN8027\_IIC\_FUNC()       \

    do { \

        if(qn8027\_iic\_log\_on)printk("[qn8027] %s\n", \_\_func\_\_); \

    }while (0)

/\*----------------------------------------------------------------------------\*/

// qn8027device information

/\*----------------------------------------------------------------------------\*/

#defineMAX\_TRANSACTION\_LENGTH 8

#defineQN8027\_DEVICE\_NAME           "FMtransmitter"

#defineQN8027\_I2C\_SLAVE\_ADDR        0x58

#defineIIC\_BUSNUM              2

#defineGPIO\_EN\_QN8027\_PIN          (GPIO31 |0x80000000)

/\*EINT\_AMP Y23 \*/

#defineGPIO\_SPEAKER\_EN\_PIN                         (GPIO88| 0x80000000)

staticbool QN8027\_FLAG = FALSE;

externbool accdet\_plug\_state();

/\*----------------------------------------------------------------------------\*/

staticint qn8027\_i2c\_probe(struct i2c\_client \*client, const struct i2c\_device\_id\*id);

staticint qn8027\_i2c\_remove(struct i2c\_client \*client);

staticstruct i2c\_client \*qn8027\_i2c\_client = NULL;

staticconst struct i2c\_device\_id qn8027\_i2c\_id[] = {{QN8027\_DEVICE\_NAME,0},{}};

staticstruct i2c\_board\_info \_\_initdata i2c\_qn8027 = {I2C\_BOARD\_INFO(QN8027\_DEVICE\_NAME, (QN8027\_I2C\_SLAVE\_ADDR>>1))};

externvoid set\_pwm(int vol);

/\*----------------------------------------------------------------------------\*/

structi2c\_driver qn8027\_i2c\_driver = {

    .probe = qn8027\_i2c\_probe,

    .remove = qn8027\_i2c\_remove,

    .driver = { .name = QN8027\_DEVICE\_NAME,},

    .id\_table = qn8027\_i2c\_id,

};

structqn8027\_i2c\_data {

      struct i2c\_client \*client;

        uint16\_t addr;

        int use\_reset;  //use RESET flag

        int use\_irq;            //use EINT flag

        int retry;

};

staticstruct qn8027\_i2c\_data \*obj\_i2c\_data = NULL;

boolqn8027\_status()

{

return QN8027\_FLAG;

}

EXPORT\_SYMBOL(qn8027\_status);

/\*----------------------------------------------------------------------------\*/

intqn8027\_i2c\_read(u16 addr, u32 \*data)

{

    u8 rxBuf[1] = {0};

    int ret = 0;

    struct i2c\_client \*client = qn8027\_i2c\_client;

    u8 lens;

    client->addr = (client->addr &I2C\_MASK\_FLAG);

    client->timing = 400;

    client->ext\_flag = I2C\_WR\_FLAG;

        rxBuf[0] = addr;

        lens = 1;

        ret = i2c\_master\_send(client,&rxBuf[0], (1 << 8) | lens);

    if (ret < 0)

    {

               QN8027\_IIC\_LOG("qn8027\_i2c\_read reg[0x%X] fail, Error code [0x%X]\n", addr, ret);

        return -EFAULT;

    }

        //ret = i2c\_master\_recv(client,&rxBuf[0], 0x1);

   \*data = rxBuf[0];

        QN8027\_IIC\_LOG("qn8027\_i2c\_readreg[0x%X] = 0x%X\n", addr, rxBuf[0]);

    return 0;

}

/\*----------------------------------------------------------------------------\*/

EXPORT\_SYMBOL\_GPL(qn8027\_i2c\_read);

/\*----------------------------------------------------------------------------\*/

intqn8027\_i2c\_write(u16 addr, u32 data)

{

    struct i2c\_client \*client =qn8027\_i2c\_client;

    u8 buffer[8];

        u8 write\_data[2] = {0};

    int ret = 0;

    struct i2c\_msg msg =

    {

        .addr = client->addr &I2C\_MASK\_FLAG,

        .flags = 0,

        .len = 2,

        .buf = write\_data,

        .timing = 400,

    };

        write\_data[0] = addr;

    write\_data[1] = data;

        //ret = i2c\_master\_send(client,write\_data, 0x2);

    ret = i2c\_transfer(client->adapter,&msg, 1);

    if (ret < 0)

    {

               QN8027\_IIC\_LOG("qn8027\_write reg[0x%X] fail, Error code [0x%X]\n", addr, ret);

        return -EFAULT;

    }

        QN8027\_IIC\_LOG("[QN8027]qn8027\_i2c\_write reg[0x%X] = 0x%X\n", addr, data);

    return 0;

}

/\*----------------------------------------------------------------------------\*/

EXPORT\_SYMBOL\_GPL(qn8027\_i2c\_write);

voidqn8027\_reg\_init()

{

        qn8027\_i2c\_write(0x00, 0x81);

        msleep(20);

        //qn8027\_i2c\_write(0x03, 0x50);

        qn8027\_i2c\_write(0x04, 0x32);

        qn8027\_i2c\_write(0x00, 0x41);

        qn8027\_i2c\_write(0x00, 0x01);

        msleep(20);

        //qn8027\_i2c\_write(0x18, 0xe4);

        //qn8027\_i2c\_write(0x1b, 0xf0);

        qn8027\_i2c\_write(0x01, 0x00); //sendrate 88.8M

        qn8027\_i2c\_write(0x02, 0xb9);

        qn8027\_i2c\_write(0x00, 0x21);

}

staticvoid start\_transfer()

{

        qn8027\_i2c\_write(0x00, 0x81);

        msleep(20);

        qn8027\_i2c\_write(0x04, 0x32);

        qn8027\_i2c\_write(0x00, 0x41);

        qn8027\_i2c\_write(0x00, 0x01);

        msleep(20);

        qn8027\_i2c\_write(0x01, 0x00); //sendrate 88.8M

        qn8027\_i2c\_write(0x02, 0xb9);

        qn8027\_i2c\_write(0x00, 0x21);

}

staticvoid stop\_transfer()

{

        qn8027\_i2c\_write(0x00, 0x81);

        msleep(20);

}

staticvoid set\_transfer\_rate(int freq)

{

        int freqH;

        int freqL;

        if(freq < 760 || freq > 1080)

        {

                printk("fmtrans: the freqis not right\n");

        }

        else

        {

                freq = (freq - 760)\*10 / 5;

                if(freq > 0xff)

                {

                        freqH = 0x20 |(freq>>8 & 0x3);

                        freqL = freq&0xff;

                        printk("set thefreqL 0x%x\n", freqL);

                        qn8027\_i2c\_write(0x01,freqL);

                        printk("set thefreqH 0x%x\n", freqH);

                        qn8027\_i2c\_write(0x00,freqH);

                }

                else

                {

                        qn8027\_i2c\_write(0x01,freq);

                        qn8027\_i2c\_write(0x00,0x20);

                }

        }

}

staticint qn8027\_open(struct inode \*inode, struct file \*file)

{

        file->private\_data =qn8027\_i2c\_client;

        return nonseekable\_open(inode, file);

}

staticint qn8027\_release(struct inode \*inode, struct file \*file)

{

        file->private\_data = NULL;

        return 0;

}

staticlong qn8027\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg)/\*IOCTL        接口，用于接受和处理上层发送过来的IO命令。注意具体命令格式和参数要对应。\*/

{

        struct i2c\_client \*client = (structi2c\_client\*)file->private\_data;

        void \_\_user \*data;

        int freq = 0;

        printk("jimmy: qn8027\_ioctl cmd =%d\n", cmd);

        switch(cmd)

        {

                caseFMTRANS\_IOCTL\_START:/\*开启FM发送模块的命令\*/

                                                //mt\_set\_gpio\_mode(GPIO\_SPEAKER\_EN\_PIN,GPIO\_MODE\_00);

                                                //mt\_set\_gpio\_dir(GPIO\_SPEAKER\_EN\_PIN,GPIO\_DIR\_OUT);

                                                //mt\_set\_gpio\_out(GPIO\_SPEAKER\_EN\_PIN,GPIO\_OUT\_ZERO);

                                                QN8027\_FLAG= TRUE;

                                                set\_pwm(0);/\*本地声音静音处理\*/

                                                msleep(200);

                                                /\*下面是控制QN8027的上电\*/

                                                mt\_set\_gpio\_mode(GPIO\_EN\_QN8027\_PIN,GPIO\_MODE\_00);

                                                mt\_set\_gpio\_dir(GPIO\_EN\_QN8027\_PIN,GPIO\_DIR\_OUT);

                                                mt\_set\_gpio\_out(GPIO\_EN\_QN8027\_PIN,GPIO\_OUT\_ONE);

                                                msleep(200);

                        start\_transfer();/\*I2C写入对应寄存器，让QN8027工作起来，具体寄存器不讨论，可以看datasheet，或者直接找FAE就好了\*/

                                                 printk("FMTRANS\_IOCTL\_START:QN8027\_FLAG =%d\n", QN8027\_FLAG);

                        break;

                case FMTRANS\_IOCTL\_STOP:

                        stop\_transfer();

                                                msleep(200);

                                               QN8027\_FLAG= FALSE;

                                                mt\_set\_gpio\_mode(GPIO\_EN\_QN8027\_PIN,GPIO\_MODE\_00);

                                                mt\_set\_gpio\_dir(GPIO\_EN\_QN8027\_PIN,GPIO\_DIR\_OUT);

                                                mt\_set\_gpio\_out(GPIO\_EN\_QN8027\_PIN,GPIO\_OUT\_ZERO);

                                                msleep(200);

                                                //if(!accdet\_plug\_state())

                                                {

                                                         //mt\_set\_gpio\_mode(GPIO\_SPEAKER\_EN\_PIN,GPIO\_MODE\_00);

                                               //mt\_set\_gpio\_dir(GPIO\_SPEAKER\_EN\_PIN,GPIO\_DIR\_OUT);

                                               //mt\_set\_gpio\_out(GPIO\_SPEAKER\_EN\_PIN,GPIO\_OUT\_ONE);

                                               set\_pwm(100);

                                                }

                                                printk("FMTRANS\_IOCTL\_STOP:QN8027\_FLAG= %d\n", QN8027\_FLAG);

                        break;

                case FMTRANS\_IOCTL\_SET\_RATE:

                        /\*

                        data = (void \_\_user\*)arg;

                       copy\_from\_user(&freq, data, sizeof(freq));

                        \*/

                        \_\_get\_user(freq, (int\*)arg);

                        printk("get therate %d\n", freq);

                       set\_transfer\_rate(freq);

                        break;

                default :

                        break;

        }

        return 0;

}

staticstruct file\_operations qn8027\_fops = {

        .owner = THIS\_MODULE,

        .open = qn8027\_open,

        .release = qn8027\_release,

        .unlocked\_ioctl = qn8027\_ioctl,

};

staticstruct miscdevice qn8027\_device = {

        .minor = MISC\_DYNAMIC\_MINOR,

        .name = "fmtransmitter",

        .fops =&qn8027\_fops,

};

/\*----------------------------------------------------------------------------\*/

// IICProbe

/\*----------------------------------------------------------------------------\*/

staticint qn8027\_i2c\_probe(struct i2c\_client \*client, const struct i2c\_device\_id \*id)/\*I2C设备对应的probe,设备注册成功后会调用。\*/

{

    int ret = -1;

    struct qn8027\_i2c\_data \*obj;

    QN8027\_IIC\_FUNC();

    obj =kzalloc(sizeof(\*obj), GFP\_KERNEL);

        if (obj == NULL) {

                ret = -ENOMEM;

               QN8027\_IIC\_LOG(QN8027\_DEVICE\_NAME ": Allocate ts memoryfail\n");

                return ret;

    }

    obj\_i2c\_data = obj;

          client->timing = 400;

    obj->client = client;

    qn8027\_i2c\_client = obj->client;

    i2c\_set\_clientdata(client, obj);

        if(misc\_register(&qn8027\_device))/\*关键的地方，注册了一个misc类型的device,注册成功后，会在 /dev目录下生产一个对应的设备文件，后面就是通过这个设备文件，用IOCTL的方式来实现kennel成的驱动与JAVA层进行通信的，中间还经过了JNI层的转换\*/

        {

                ret = -ENOMEM;

               QN8027\_IIC\_LOG(QN8027\_DEVICE\_NAME ": misc\_register error!\n");

                return ret;

        }

        //qn8027\_reg\_init();

    QN8027\_IIC\_LOG("MediaTek QN8027 i2cprobe success\n");

    return 0;

}

/\*----------------------------------------------------------------------------\*/

staticint qn8027\_i2c\_remove(struct i2c\_client \*client)

{

    QN8027\_IIC\_FUNC();

    qn8027\_i2c\_client = NULL;

    i2c\_unregister\_device(client);

    kfree(i2c\_get\_clientdata(client));

        misc\_deregister(&qn8027\_device);

    return 0;

}

intg\_reg\_value\_qn8027=0;

staticunsigned int reg\_address = 0;

staticssize\_t show\_qn8027\_access(struct device \*dev,struct device\_attribute \*attr,char \*buf)

{

        qn8027\_i2c\_read(reg\_address,&g\_reg\_value\_qn8027);

        printk("[show\_qn8027\_access]0x%x\n", g\_reg\_value\_qn8027);

        return sprintf(buf,"0x%x=0x%x\n", reg\_address, g\_reg\_value\_qn8027);

}

staticssize\_t store\_qn8027\_access(struct device \*dev,struct device\_attribute \*attr,const char \*buf, size\_t size)

{

        int ret=0;

        char \*pvalue = NULL;

        unsigned int reg\_value = 0;

        printk("[store\_qn8027\_access]\n");

        if(buf != NULL && size != 0)

        {

                printk("[store\_qn8027\_access] buf is %sand size is %d \n",buf,size);

                reg\_address =simple\_strtoul(buf,&pvalue,16);

                if(size > 5)

                {

                        reg\_value =simple\_strtoul((pvalue+1),NULL,16);

                       printk("[store\_qn8027\_access] write qn8027 reg 0x%x with value 0x%x!\n",reg\_address,reg\_value);

                       qn8027\_i2c\_write(reg\_address, reg\_value);

                }

        }

        return size;

}

staticstruct kobj\_attribute reg\_attribute = {

        .attr = {.name ="qn8027\_reg", .mode = 0666},

        .show = show\_qn8027\_access,

        .store = store\_qn8027\_access,

};

/\*----------------------------------------------------------------------------\*/

staticint \_\_init qn8027\_init(void)/\*驱动模块的入口\*/

{

    QN8027\_IIC\_FUNC();

    i2c\_register\_board\_info(IIC\_BUSNUM,&i2c\_qn8027, 1);/\*设置I2C的口线\*/

    if(i2c\_add\_driver(&qn8027\_i2c\_driver)) /\*如名字，把qn8027\_i2c\_driver加入到i2c的驱动中去\*/

    {

        QN8027\_IIC\_LOG("unable to addqn8027 i2c driver.\n");

        return -1;

    }

        sysfs\_create\_file(kernel\_kobj,&reg\_attribute.attr);

/\*增加一个node节点，主要是方便用adb命令对寄存器进行调试\*/

    return 0;

}

/\*----------------------------------------------------------------------------\*/

staticvoid \_\_exit qn8027\_exit(void)

{

    i2c\_del\_driver(&qn8027\_i2c\_driver);

}

/\*----------------------------------------------------------------------------\*/

module\_init(qn8027\_init);

module\_exit(qn8027\_exit);

qn8027.h文档：/\*定义IO口命令，FMTRANS这个定义为0x58，不知道为什么。好像是定义的这个器件的地址。估计只是为了区分开别的设备，定义一个值而已吧\*/

#include<linux/ioctl.h>

#defineFMTRANS                                        0x58

#defineFMTRANS\_IOCTL\_START                    \_IO(FMTRANS, 0x01)

#defineFMTRANS\_IOCTL\_STOP                     \_IO(FMTRANS, 0x02)

#define FMTRANS\_IOCTL\_SET\_RATE          \_IOW(FMTRANS,0x03, int)

二，JNI层的实现：

Android.mk   /\*编译文件，主要生产一个的libfm\_qn8027\_jni.so库，让JAVA层调用\*/

fm\_qn8027.cpp    /\*JNI具体的转换文件\*/

fm\_qn8027.h    /\*头文件，与kernel层的想对应\*/

Android.mk文档：

LOCAL\_PATH:=$(call my-dir)

include$(CLEAR\_VARS)

LOCAL\_MODULE\_TAGS:= optional

LOCAL\_CERTIFICATE:= platform

LOCAL\_SRC\_FILES:=\

         fm\_qn8027.cpp

LOCAL\_SHARED\_LIBRARIES:= \

    libandroid\_runtime \

    libnativehelper \

    libcutils \

    libutils \

    liblog \

    libhardware

LOCAL\_C\_INCLUDES+= \

    $(JNI\_H\_INCLUDE) \

    libcore/include \

LOCAL\_MODULE:=libfm\_qn8027\_jni

ALL\_DEFAULT\_INSTALLED\_MODULES+= $(LOCAL\_MODULE)

include $(BUILD\_SHARED\_LIBRARY)

fm\_qn8027.cpp文档：

#include<jni.h>

#include<utils/Log.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/ioctl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<unistd.h>

#include<termios.h>

#include<pthread.h>

#include<signal.h>

#include<errno.h>

#include<dlfcn.h>

#include"fm\_qn8027.h"

#ifdefLOG\_TAG

#undefLOG\_TAG

#endif

#defineLOG\_TAG "QN8027\_control"

usingnamespace android;

//#defineFMTRANS\_IOCTL\_START         0x5801

//#defineFMTRANS\_IOCTL\_STOP          0x5802

//#defineFMTRANS\_IOCTL\_SET\_RATE      0x40045803

staticint fm\_fd;

staticint jni\_pid;

staticint fm\_status;

//MutexmLock;

// mustbe kept in sync with the values in WindowManager

enumFmControlType {

         FM\_START = 0,

         FM\_STOP = 1,

         FM\_SET\_RATE = 2

};

enumFmControlStatus {

         UNKNOW = 0,

         FM\_S\_START = 1,

         FM\_s\_STOP = 2

};

staticjint qn8027\_init(JNIEnv \*env, jobject thiz)

{

         ALOGD("qn8027\_init --->");

         return 0;

}

staticjint qn8027\_finish(JNIEnv \*env, jobject thiz)

{

         ALOGD("qn8027\_finish--->");

         return 0;

}

staticjint qn8027\_open(JNIEnv \*env, jobject thiz)

{

         fm\_fd = open(DEVICE\_NAME,O\_RDWR);        //打开设备

         if(fm\_fd < 0)

         {

                   ALOGE("qn8027\_open fm\_fd= %d", fm\_fd);

         }

         return fm\_fd;

}

staticjint qn8027\_close(JNIEnv \*env, jobject thiz)

{

         jint ret;

    ret = close(fm\_fd);

         return ret;

}

staticjint qn8027\_start(JNIEnv \*env, jobject thiz)

{

         jint ret;

         ret = ioctl(fm\_fd,FMTRANS\_IOCTL\_START);

         if (ret < 0)

         {

                   ALOGE("qn8027\_start ret= %d", ret);

                   return ret;

         }

         return 0;

}

staticjint qn8027\_stop(JNIEnv \*env, jobject thiz)

{

         jint ret;

         ret = ioctl(fm\_fd, FMTRANS\_IOCTL\_STOP);

         if (ret < 0)

         {

                   ALOGE("qn8027\_stop ret =%d", ret);

                   return ret;

         }

         return 0;

}

staticjint qn8027\_set\_rate(JNIEnv \*env, jobject thiz, jint a)

{

         static jint set\_rate = 0;

         jint ret;

         set\_rate = a;             //760 <= set\_rate <= 1080

         ALOGD("--->set\_rate = %d",set\_rate);

         ret = ioctl(fm\_fd,FMTRANS\_IOCTL\_SET\_RATE, &set\_rate);

         if (ret < 0)

         {

                   ALOGE("qn8027\_set\_rateret = %d", ret);

                   return ret;

         }

         return 0;

}

//com/hcn/fmemitter

staticconst char \*classPathName = "com/hcn/fmemitter/MainActivity";

/\*增加的具体方法\*/

static JNINativeMethod methods[] = {

//  {"add","(II)I", (void\*)add },

  {"FmInit","()I", (void\*)qn8027\_init },

 {"FmFinish", "()I", (void\*)qn8027\_finish},

  {"FmOpen","()I", (void\*)qn8027\_open },

  {"FmClose","()I", (void\*)qn8027\_close},

  {"FmStart","()I", (void\*)qn8027\_start },

  {"FmStop","()I", (void\*)qn8027\_stop },

// {"FmStatus", "()I", (void\*)qn8027\_status },

 {"FmSetRate", "(I)I", (void\*)qn8027\_set\_rate },

};

/\*

 \* Register several native methods for oneclass.

 \*/

staticint registerNativeMethods(JNIEnv\* env, const char\* className,

    JNINativeMethod\* gMethods, int numMethods)

{

    jclass clazz;

    clazz = env->FindClass(className);

    if (clazz == NULL) {

        ALOGE("Native registration unableto find class '%s'", className);

        return JNI\_FALSE;

    }

    if (env->RegisterNatives(clazz,gMethods, numMethods) < 0) {

        ALOGE("RegisterNatives failed for'%s'", className);

        return JNI\_FALSE;

    }

    return JNI\_TRUE;

}

/\*

 \* Register native methods for all classes weknow about.

 \*

 \* returns JNI\_TRUE on success.

 \*/

staticint registerNatives(JNIEnv\* env)

{

/\*注册一个本地的方法，classPathName就是需要增加这些方法的class\*/

  if (!registerNativeMethods(env,classPathName,

                 methods, sizeof(methods) /sizeof(methods[0]))) {

   return JNI\_FALSE;

  }

  return JNI\_TRUE;

}

//----------------------------------------------------------------------------

/\*

 \* This is called by the VM when the sharedlibrary is first loaded.

 \*/

typedefunion {

    JNIEnv\* env;

    void\* venv;

}UnionJNIEnvToVoid;

jintJNI\_OnLoad(JavaVM\* vm, void\* reserved)/\*JNI的入口，具体流程待学习\*/

{

    UnionJNIEnvToVoid uenv;

    uenv.venv = NULL;

    jint result = -1;

    JNIEnv\* env = NULL;

    ALOGI("JNI\_OnLoad");

    if (vm->GetEnv(&uenv.venv,JNI\_VERSION\_1\_4) != JNI\_OK) {

        ALOGE("ERROR: GetEnvfailed");

        goto bail;

    }

    env = uenv.env;

    if (registerNatives(env)!= JNI\_TRUE) {

        ALOGE("ERROR: registerNativesfailed");

        goto bail;

    }

    result = JNI\_VERSION\_1\_4;

bail:

    return result;

}

fm\_qn8027.h文档：

#ifndef\_\_QN8027\_H\_\_

#define\_\_QN8027\_H\_\_

#include<linux/ioctl.h>

#include<linux/time.h>

#defineFMTRANS                     0x58

#defineFMTRANS\_IOCTL\_START         \_IO(FMTRANS,0x01)

#defineFMTRANS\_IOCTL\_STOP          \_IO(FMTRANS,0x02)

#defineFMTRANS\_IOCTL\_SET\_RATE     \_IOW(FMTRANS,0x03, unsigned int)

#defineDEVICE\_NAME "/dev/fmtransmitter"

#endif

三，JAVA层的调用：

加载LIB，然后申明这个接口为本地的方法。后面就可以当做内部函数来调用了。

 static

  {

   System.loadLibrary("fm\_qn8027\_jni");

  }

 private static native int FmClose();

 private static native int FmFinish();

 private static native int FmInit();

 private static native int FmOpen();

 private static native int FmSetRate(int paramInt);

 private static native int FmStart();

 private static native int FmStop();

四，FM发送对应sh命令的制作。

Android.mk    /\*编译文档\*/

fmtest.c    /\*命令的实现\*/

Android.mk文档：

LOCAL\_PATH:= $(call my-dir)

include$(CLEAR\_VARS)

LOCAL\_SRC\_FILES:= fmtest.c

LOCAL\_SHARED\_LIBRARIES:= \

        libcutils

LOCAL\_CERTIFICATE:= platform

LOCAL\_MODULE:= fmtest

include $(BUILD\_EXECUTABLE)

fmtest.c文档：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        > File Name: fmtest.c

        > Created by JinQuan, Time: Fri 19Jun 2015 02:05:57 PM CST

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

 #include <sys/ioctl.h>

 #define FMTRANS                     0x58

 #define FMTRANS\_IOCTL\_START         \_IO(FMTRANS, 0x01)

 #define FMTRANS\_IOCTL\_STOP          \_IO(FMTRANS, 0x02)

 #define FMTRANS\_IOCTL\_SET\_RATE      \_IOW(FMTRANS,0x03, unsigned int)

 /\*

 fmtest cmd:

 0: stop

 1: start

 other: set rate

关闭FM发射adb命令：

Fmtest /dev/fmtransmitter 0

打开FM发送：

Fmtest /dev/fmtransmitter 1

设置FM发送的频率：例如频率为90.0MHZ

Fmtest /dev/fmtransmitter 2 900

 \*/

 int main(int argc, char \*\*argv)

 {

         int fd;

         int ret;

         int cmd = argv[2][0];

         if(argc < 2)

         {

                 printf("please entercmd\n");

                 return 0;

         }

//               fd = open("/dev/tty",O\_RDWR | O\_NOCTTY | O\_NDELAY);

                   printf("1-->>%s\n",argv[0]);

                   printf("2-->>%s\n",argv[1]);

//               fd =open("/dev/ttyMT1", O\_RDWR );

                   fd = open(argv[1], O\_RDWR );

//         fd =open("/dev/fmtransmitter", O\_RDONLY);

         if(fd <=0 )

         {

                 printf("open deviceserror!fd=%d\n",fd);

                 return -1;

         } else {

                             printf("open success!fd=%d\n",fd);

                    }

                    sleep(5);

         switch(cmd)

         {

                 case '0':

                         printf("FMTRANS\_IOCTL\_STOP\n");

                         ret = ioctl(fd,FMTRANS\_IOCTL\_STOP);

                         printf("ioctlreturn is %d\n", ret);

                         break;

                 case '1':

                        printf("FMTRANS\_IOCTL\_START\n");

                         ioctl(fd, FMTRANS\_IOCTL\_START);

                         break;

                 case '2':

                         if(argc < 3)

                         {

                                 printf("parameter notright\n");

                                 return -1;

                         }

                         int freq =strtoul(argv[3], 0, 0);

                        printf("FMTRANS\_IOCTL\_SET\_RATE %d\n", freq);

                         ioctl(fd,FMTRANS\_IOCTL\_SET\_RATE, &freq);

                         break;

                 default:

                         break;

         }

         close(fd);

         return 0;

 }

---------------------

作者：liyanfei123456

来源：CSDN

原文：https://blog.csdn.net/liyanfei123456/article/details/53516741

版权声明：本文为博主原创文章，转载请附上博文链接！