目录

[1 Rk3308 简介： 2](#_Toc7110388)

[2使用adb连接板子 2](#_Toc7110389)

[2.1 adb安装 2](#_Toc7110390)

[2.2 使用putty连接板子 2](#_Toc7110391)

[2.3 上传下载文件到开发板 2](#_Toc7110392)

[2.4 更新内核 3](#_Toc7110393)

[3编译下载内核 3](#_Toc7110394)

[3.1 编译uboot(32位) 3](#_Toc7110395)

[3.2 编译内核 3](#_Toc7110396)

[3.3 编译驱动的Makefile（32位内核） 3](#_Toc7110397)

[3.4 板子匹配的设备树 4](#_Toc7110398)

[3.5 添加lcd驱动到内核 4](#_Toc7110399)

[3.6 编译应用的编译器(32位编译器) 4](#_Toc7110400)

[3.7 编译配置buildroot 5](#_Toc7110401)

[3.8 mkfirmware.sh生成固件 5](#_Toc7110402)

[3.9 parameter的配置和含义 6](#_Toc7110403)

[3.10 userdata.img生成 6](#_Toc7110404)

[4 GPIO口使用 6](#_Toc7110405)

[5 spi 接口Lcd驱动 7](#_Toc7110406)

[5.1 设计思路 7](#_Toc7110407)

[5.2 硬件信息 7](#_Toc7110408)

[4.2.1 rk3308 SPI功能 7](#_Toc7110409)

[5.2.2配置内核支持瑞芯微spi 8](#_Toc7110410)

[5.2.3 lcd 屏接口模式配置 8](#_Toc7110411)

[5.2.4 lcd屏spi接口引脚描述 8](#_Toc7110412)

[5.2 dts配置 8](#_Toc7110413)

[5.3 驱动例程 9](#_Toc7110414)

[5.4 测试 9](#_Toc7110415)

[6 触摸屏ft6236驱动 10](#_Toc7110416)

[6.1 设计思路 10](#_Toc7110417)

[6.2 配置dts 10](#_Toc7110418)

[6.3驱动例程 10](#_Toc7110419)

[6.4 测试 10](#_Toc7110420)

[7 配置minigui支持 11](#_Toc7110421)

[8 单独移植minigui 12](#_Toc7110422)

[9移植openssl和curl 15](#_Toc7110423)

[10 显示logo 16](#_Toc7110424)

[10.1 uboot显示logo 16](#_Toc7110425)

[10.3 测试uboot的spi驱动是否可用 17](#_Toc7110426)

[10.2 内核显示logo 18](#_Toc7110427)

[11 裁剪内存 18](#_Toc7110428)

[12 PWM背光 19](#_Toc7110429)

[12.1 配置内核支持pwm背光驱动 19](#_Toc7110430)

[12.2 配置pwm背光设备节点 19](#_Toc7110431)

[12.2.3 背光驱动 20](#_Toc7110432)

[13 wifi 配置 20](#_Toc7110433)

[13.1配置设备树 20](#_Toc7110434)

[13.2 配置内核 20](#_Toc7110435)

[13.3.1 buildroot选择配置文件 20](#_Toc7110436)

[13.3.2 串口终端输入 21](#_Toc7110437)

[13.4 驱动源码 21](#_Toc7110438)

[14 红外感应开关 21](#_Toc7110439)

[14.1 设计思路 21](#_Toc7110440)

[14.2 设备树配置 21](#_Toc7110441)

[错误解决 22](#_Toc7110442)

# 1 Rk3308 简介：

四核64位核心处理器，整合高性能CODEC音频编解码器，VAD。支持国内外多种人工智能及物联网操作系统，可配置6个麦克风阵列及POE扩展板。可用于智能音频方案、语音识别等项目。

**处理器核心：**cortex-a35核心配置，主频1.3GHZ,

**开源开发板：**ROC-RK3308-CC 是一款集多种功能为一体的AI+IOT开源主板。

**vad:(静音抑制)：**语音活动检测(Voice Activity Detection,VAD)又称语音端点检测,语音边界检测。目的是从声音信号流里识别和消除长时间的静音期，以达到在不降低业务质量的情况下节省[话路](https://baike.baidu.com/item/%E8%AF%9D%E8%B7%AF/5922604)资源的作用，它是IP电话应用的重要组成部分。静音抑制可以节省宝贵的[带宽](https://baike.baidu.com/item/%E5%B8%A6%E5%AE%BD/266879)资源，可以有利于减少用户感觉到的端到端的时延。

# 2使用adb连接板子

## 2.1 adb安装

下载adb安装包，然后将system32里面的cmd命令行拷贝到adb文件夹里面。打开cmd工具执行adb shell登陆到板子的根目录中。如果能进入根目录证明配置成功

## 2.2 使用putty连接板子

下载putty的adb版本，放入adb文件夹里面，打开选择adb模式，Host name里面填写transport-usb,然后点击open进入到板子的根目录。

## 2.3 上传下载文件到开发板

打开adb目录下的cmd命令行工具，不要执行adb shell 登陆到板子。

adb push <本地路径> <远程路径>

adb push "E:\hello.txt" "userdata/mytest/" //板子data区可读写，注意/是linux下，\window

//下使用。

adb pull <远程路径> <本地路径>

adb push "userdata/mytest/ hello.txt " "E:\ "

## 2.4 更新内核

瑞芯微开发工具软件下载，首先按住reset 插入usb线，当软件提示发现一个ADB

设备后点击切换按钮进入loader模式，点击执行开始下载。

该软件可以单独更新某个固件。如果不知道下载地址，那么可以读取设备内部的分区表，分区表就是nandflash存放uboot，kernel，rootfs等的分区信息：

GPT part: 0, name: uboot, start:0x2000, size:0x800

GPT part: 1, name: trust, start:0x2800, size:0x800

GPT part: 2, name: misc, start:0x3000, size:0x800

GPT part: 3, name: recovery, start:0x3800, size:0x6000

GPT part: 4, name: boot, start:0x9800, size:0x3000

GPT part: 5, name: rootfs, start:0xc800, size:0x11800

GPT part: 6, name: oem, start:0x1e000, size:0xe000

GPT part: 7, name: userdata, start:0x2c000, size:0x97df

# 3编译下载内核

## 3.1 编译uboot(32位)

source buildroot/build/envsetup.sh //选择3,32位debug版本

Cd uboot/

make ARCH=arm menuconfig

./make.sh evb-aarch32-rk3308

## 3.2 编译内核

source buildroot/build/envsetup.sh //选择3,32位debug版本

cd kernel

make ARCH=arm rk3308\_linux\_aarch32\_debug\_defconfig

make ARCH=arm menuconfig

make ARCH=arm savedefconfig

cp defconfig arch/arm/configs/rk3308\_linux\_aarch32\_debug\_defconfig

make ARCH=arm rk3308-voice-module-board-v10-aarch32.img //使用这个

//make ARCH=arm rk3308-voice-module-amic-mainboard-v10-aarch32.img

## 3.3 编译驱动的Makefile（32位内核）

编译驱动的终端必须是配置内核的终端，而且使用root用户。

KERN\_DIR = /home/chao/rk3308\_linux/kernel

all:

make -C $(KERN\_DIR) M=`pwd` ARCH=arm modules //指定是32位

clean:

make -C $(KERN\_DIR) M=`pwd` ARCH=arm modules clean

rm -rf modules.order

obj-m += gpio\_keys.o

## 3.4 板子匹配的设备树

配置1：

**rk3308-voice-module-amic-mainboard-v10-aarch32.**

#include "rk3308-voice-module-v10-aarch32.dtsi"

#include "rk3308-voice-module-mainboard-v10-aarch32.dtsi"

32位dts也包含arm64/boot/dts/rk3308.dtsi

**配置2：rk3308-voice-module-board-v10-aarch32**

#include "rk3308-voice-module-v10-aarch32.dtsi"

## 3.5 添加lcd驱动到内核

以我编写的spilcd驱动为例

1. driver/char/目录下新建目录spilcd
2. 把spilcd.c拷贝到spilcd目录里面
3. 创建Makefile和Kconfig两个文件//Kconfig用于图像配置，Makefile用于编译
4. Kconfig内容如下：

menu "RK3308 SPILCD"

config RK3308\_SPILCD

tristate "the driver of spilcd for rk3308"

depends on CPU\_RK3308 //表示CPU\_RK3308选中才能配置该选项

default Y

help

this is a driver for spi lcd

endmenu

1. 修改Makefile添加如下行

obj-$(CONFIG\_RK3308\_SPILCD) += spilcd.o

// CONFIG\_RK3308\_SPILCD就是前面的config RK3308\_SPILCD 中的RK3308\_SPILCD加上//CONFIG\_

1. 修改Drivers/char目录下面的Kconfig

source "drivers/char/spilcd/Kconfig"//建立和子目录的联系

endmenu

1. 修改Drivers/char目录下面的Makefile,在最后添加

obj-$(CONFIG\_RK3308\_SPILCD) += spilcd/ //建立和子目录的联系

## 3.6 编译应用的编译器(32位编译器)

/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_release/host/bin/arm-rockchip-linux-gnueabihf-gcc

## 3.7 编译配置buildroot

source buildroot/build/envsetup.sh

选择3,32位调试版本

cd buildroot/output/

cd rockchip\_rk3308\_32\_debug/

make menuconfig

make savedefconfig

make //编译buildroot

## 3.8 mkfirmware.sh生成固件

需要配置device/rockchip/BoardConfig.mk文件。BoardConfig.mk是mkfirmware.sh执行时的环境变量配置文件，里面包括了生成img文件的源文件路径。

可以复制BoardConfig\_32bit.mk为BoardConfig.mk然后再更改

下面是生成32位系统debug版系统镜像的配置：

#!/bin/bash

# Target arch

export RK\_ARCH=arm

# Uboot defconfig

export RK\_UBOOT\_DEFCONFIG=evb-aarch32-rk3308

# Kernel defconfig

export RK\_KERNEL\_DEFCONFIG=rk3308\_linux\_aarch32\_debug\_defconfig

# Kernel dts

export RK\_KERNEL\_DTS=rk3308-voice-module-amic-mainboard-v10-aarch32

# boot image type

export RK\_BOOT\_IMG=zboot.img

# parameter for GPT table

export RK\_PARAMETER=parameter-32bit.txt

# Buildroot config

export RK\_CFG\_BUILDROOT=rockchip\_rk3308\_32\_debug

# Recovery config

export RK\_CFG\_RECOVERY=rockchip\_rk3308\_recovery

# Pcba config

export RK\_CFG\_PCBA=rockchip\_rk3308\_pcba

# Build jobs

export RK\_JOBS=12

# target chip

export RK\_TARGET\_PRODUCT=rk3308

# Set rootfs type, including ext2 ext4 squashfs

export RK\_ROOTFS\_TYPE=squashfs

# rootfs image path

export RK\_ROOTFS\_IMG=buildroot/output/$RK\_CFG\_BUILDROOT/images/rootfs.$RK\_ROOTFS\_TYPE

# Set oem partition type, including ext2 squashfs

export RK\_OEM\_FS\_TYPE=ext2

# Set userdata partition type, including ext2, fat

export RK\_USERDATA\_FS\_TYPE=ext2

# Set flash type. support <emmc, nand, spi\_nand, spi\_nor>

export RK\_STORAGE\_TYPE=nand

#OEM config: /oem/dueros/aispeech-6mic-64bit/aispeech-2mic-64bit/aispeech-4mic-32bit/aispeech-2mic-32bit/aispeech-2mic-kongtiao-32bit/iflytekSDK/CaeDemo\_VAD/smart\_voice

export RK\_OEM\_DIR=oem//表示不打包算法程序

#userdata config

export RK\_USERDATA\_DIR=userdata\_empty //定义userdata.img源路径，此处为不生成。

MIC\_NUM=2

## 3.9 parameter的配置和含义

FIRMWARE\_VER:8.1

MACHINE\_MODEL:RK3308

MACHINE\_ID:007

MANUFACTURER: RK3308

MAGIC: 0x5041524B

ATAG: 0x00200800

MACHINE: 3308

CHECK\_MASK: 0x80

PWR\_HLD: 0,0,A,0,1

TYPE: GPT

CMDLINE:mtdparts=rk29xxnand:0x00000800@0x00002000(uboot),0x00000800@0x00002800(trust),0x00000800@0x00003000(misc),0x00006000@0x00003800(recovery),0x00003000@0x00009800(boot),0x00015000@0x0000C800(rootfs),0x0000E000@0x00021800(oem),-@0x0002F800(userdata:grow)

uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9

parameter是用来给nandflash分区的，单位是512子节。0x00000800@0x00002000(uboot)就是设置uboot分区的。起始地址在0x00002000\*512子节处，大小为0x00000800\*512B。

其中绿色部分为更改后的值，因为文件系统较大，原来预留的空间不足，修改增大了空间。

否则会导致rootfs.img下载失败。

## 3.10 userdata.img生成

Mkfirmware.sh调用 device/rockchip/common里面的mk-userdata.sh生成镜像。镜像的源文件位置device/rockchip/userdata里面。该路径是由device/rockchip/rk3308/BoardConfig.mk里面指示。

# 4 GPIO口使用

Firefly-RK3399的dts:

对引脚的描述与Firefly-RK3288有所区别，GPIO0\_B4被描述为：<&gpio0 12 GPIO\_ACTIVE\_HIGH>，这里的12来源于：8+4=12，其中8是因为GPIO0\_B4是属于GPIO0的B组，如果是A组的话则为0，如果是C组则为16，如果是D组则为24，以此递推，而4是因为B4后面的4。 GPIO\_ACTIVE\_HIGH表示高电平有效，如果想要低电平有效，可以改为：GPIO\_ACTIVE\_LOW，这个属性将被驱动所读取。

Rk3308 dts:

一般按键引脚配置：

gpio-keys {

compatible = "gpio-keys";

autorepeat;

pinctrl-names = "default";

pinctrl-0 = <&pwr\_key>;

power {

//gpio0 A组的第6个引脚

gpios = <&gpio0 RK\_PA6 GPIO\_ACTIVE\_LOW>;

linux,code = <KEY\_POWER>;

label = "GPIO Key Power";

wakeup-source;

debounce-interval = <100>;

};

};

&pinctrl {

pinctrl-names = "default";

pinctrl-0 = <&rtc\_32k>;

buttons {

pwr\_key: pwr-key {

rockchip,pins = <0 RK\_PA6 RK\_FUNC\_GPIO &pcfg\_pull\_up>;

};

};

};

};

# 5 spi 接口Lcd驱动

## 5.1 设计思路

使用帧缓冲区来操作，驱动中申请缓冲区，应用层映射该缓冲区，应用层像缓冲区中写入数据，驱动每隔30ms读取缓冲区数据发送到lcd设备。定时时间间隔发送采用延时工作队列delayed\_worker。先编程实现在硬件驱动中刷新lcd,然后实现支持framebuffer的驱动，从应用层测试。

## 5.2 硬件信息

### 4.2.1 rk3308 SPI功能

摩托罗拉SPI协议

支持8位和16位。

时钟频率50MHZ

SPI 4中传输模式配置

### 5.2.2配置内核支持瑞芯微spi

Device Drivers ‐‐‐>  
[\*] SPI support ‐‐‐>  
<\*> Rockchip SPI controller driver

### 5.2.3 lcd 屏接口模式配置

接口类型总共有两种：8080 MCU接口和串口。

通过外部IM[3:0]引脚来控制。1110模式下配置为四线spi接口，SCL,SDI,D/CX,SDO,CS。 1101模式下配置为3线9位接口，SCL,SDI,SDO,CSX。 0101模式下三线9位串行接口，SCL,SDA,CSX, SDA线同时作为输入和输出口。0110四线8位模式，SCL,SDA,D/CX,CSX,SDA线同时作为输入和输出口。

原理图采用的是三线9位接口。Rk3308只能支持四线8位。

### 5.2.4 lcd屏spi接口引脚描述

CSX :片选信号，低电平有效。

WRX(D/CX) :四线系统命令数据选择。低电平表示命令，高电平表示数据。

D/CX(SCL):串行时钟

SDI/SDA:四线输入信号，三线输入输出信号

SDO: 四线输出信号。主机master输出数据端口

位传输顺序：最高位先传输。9位模式下，D/CX先传输。

主机通过SDA发送读命令后，芯片会在随后的时钟位发送数据到SDA。

## 5.2 dts配置

直接修改/arch/arm64/boot/dts/rk3308.dtsi

spi2: spi@ff140000 {

compatible = "rockchip,rk3308-spi", "rockchip,rk3066-spi";

reg = <0x0 0xff140000 0x0 0x1000>;

interrupts = <GIC\_SPI 17 IRQ\_TYPE\_LEVEL\_HIGH>;

#address-cells = <1>;

#size-cells = <0>;

clocks = <&cru SCLK\_SPI2>, <&cru PCLK\_SPI2>;

clock-names = "spiclk", "apb\_pclk";

max-freq = <80000000>;

dmas = <&dmac1 16>, <&dmac1 17>;

dma-names = "tx", "rx";

pinctrl-names = "default", "high\_speed";

pinctrl-0 = <&spi2\_clk &spi2\_csn0 &spi2\_miso &spi2\_mosi>;

pinctrl-1 = <&spi2\_clk\_hs &spi2\_csn0 &spi2\_miso\_hs &spi2\_mosi\_hs>;

status = "okay";

spilcd@0{

status="okay";

compatible="spilcd";

reg=<0>;

spi-max-frequency=<40000000>;

reset-gpios=<&gpio0 RK\_PC4 GPIO\_ACTIVE\_LOW>;

dc-gpios=<&gpio0 RK\_PC1 GPIO\_ACTIVE\_LOW>;

};

};

引用修改：

rk3308-voice-module-mainboard-v10-aarch32.dtsi中引用rk3308.dtsi

rk3308保持官方文件不变。只把需要更改的重新设置。这样就更改了rk3308.dtsi中的设定值。这样做的好处是，由于多个板子公用rk3308.dtsi文件，那么改变了rk3308.dtsi就改变了所有板子的配置。这不是我们想要的，我们只想更改我们自己的板子配置。

&spi2{

status = "okay";

max-freq = <48000000>;

spilcd@0{

status="okay";

compatible="spilcd";

reg=<0>;

spi-max-frequency=<12000000>;

dc-gpios=<&gpio0 RK\_PC1 GPIO\_ACTIVE\_LOW>;

reset-gpios=<&gpio0 RK\_PC4 GPIO\_ACTIVE\_LOW>;

};

};

## 5.3 驱动例程

完整例程查看rk3308驱动myfb.c ，里面有详细的注释。

## 5.4 测试

打开adb模式cmd命令行，在没有登录板子情况下执行：

adb push “myfb.ko” “tmp”

adb push “HZK16” “tmp”

adb push “show\_font” “tmp”

连接调试串口

adb shell 登录到板子

cd /tmp/

insmod myfb.ko

chmod 777 show\_font

./show\_font 发现板子上面显示字符A和汉字中

# 6 触摸屏ft6236驱动

## 6.1 设计思路

Ft6236有数据时会产生中断。然后通过i2c接口读取数据。

## 6.2 配置dts

引用配置

rk3308-voice-module-mainboard-v10-aarch32.dtsi中引用rk3308.dtsi

&i2c3{

status = "okay";

ft6236:ft6236@0x38{

compatible = "ft6236";

status = "okay";

reg = <0x38>;

touchscreen-size-x=<240>;

touchscreen-size-y=<320>;

interrupts = <&gpio0 RK\_PC2 IRQ\_TYPE\_EDGE\_RISING>;

reset-gpios=<&gpio2 RK\_PA3 GPIO\_ACTIVE\_LOW>;

};

};

rK3308.dtsi中i2c3配置，不修改

i2c3: i2c@ff070000 {

compatible = "rockchip,rk3399-i2c";

reg = <0x0 0xff070000 0x0 0x1000>;

clocks = <&cru SCLK\_I2C3>, <&cru PCLK\_I2C3>;

clock-names = "i2c", "pclk";

interrupts = <GIC\_SPI 14 IRQ\_TYPE\_LEVEL\_HIGH>;

pinctrl-names = "default";

pinctrl-0 = <&i2c3m0\_xfer>;

#address-cells = <1>;

#size-cells = <0>;

status = "disabled";

};

## 6.3驱动例程

完整例程查看rk3308驱动ft6236.c ，里面有详细的注释

## 6.4 测试

Insmod ft6236.ko

点击触摸屏会打印坐标点数据。

# 7 配置minigui支持

make menuconfig

搜索minigui 找到配置项

1、配置打开minigui

Target packages

rockchip BSP packages

[\*] Minigui core lib

2、make savedefconfig

3、make minigui

提示信息查找lib位置

/bin/mkdir -p '/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib/pkgconfig'

/bin/mkdir -p '/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/include/minigui'

/usr/bin/install -c -m 644 minigui.pc '/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib/pkgconfig'

/usr/bin/install -c -m 644 mgconfig.h '/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/include/minigui'

在下面文件夹找到了库：

rockchip\_rk3308\_32\_debug/target/usr/lib

ls -l libminigui\_ths\* 共找到四个文件。

lrwxrwxrwx 1 root root 27 Apr 2 01:12 libminigui\_ths-3.2.so.0 -> libminigui\_ths-3.2.so.0.0.0

-rwxr-xr-x 1 root root 1260268 Apr 2 01:12 libminigui\_ths-3.2.so.0.0.0

-rwxr-xr-x 1 root root 955 Apr 2 01:12 libminigui\_ths.la

lrwxrwxrwx 1 root root 27 Apr 2 01:12 libminigui\_ths.so -> libminigui\_ths-3.2.so.0.0.0

可见使用的都是libminigui\_ths-3.2.so.0.0.0

决定复制该库到开发板

root@ubuntu:/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib# ls libjpeg\* -l

-rwxr-xr-x 1 root root 901 Apr 2 01:10 libjpeg.la

lrwxrwxrwx 1 root root 16 Apr 2 01:10 libjpeg.so -> libjpeg.so.8.1.2

lrwxrwxrwx 1 root root 16 Apr 2 01:10 libjpeg.so.8 -> libjpeg.so.8.1.2

-rwxr-xr-x 1 root root 172852 Apr 2 01:10 libjpeg.so.8.1.2

可见都是指向 libjpeg.so.8.1.2，使用该库到开发板

turbojpeg针对ARM和X86对了优化，宣称其速度是libjpeg的2到4倍。

root@ubuntu:/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib# ls libturbojpeg\* -l

-rwxr-xr-x 1 root root 931 Apr 2 01:10 libturbojpeg.la

lrwxrwxrwx 1 root root 21 Apr 2 01:10 libturbojpeg.so -> libturbojpeg.so.0.1.0

lrwxrwxrwx 1 root root 21 Apr 2 01:10 libturbojpeg.so.0 -> libturbojpeg.so.0.1.0

-rwxr-xr-x 1 root root 193180 Apr 2 01:10 libturbojpeg.so.0.1.0

使用libturbojpeg.so.0.1.0

root@ubuntu:/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib# ls libpng\* -l

-rwxr-xr-x 1 root root 921 Apr 2 01:11 libpng16.la

lrwxrwxrwx 1 root root 19 Apr 2 01:11 libpng16.so -> libpng16.so.16.34.0

lrwxrwxrwx 1 root root 19 Apr 2 01:11 libpng16.so.16 -> libpng16.so.16.34.0

-rwxr-xr-x 1 root root 144604 Apr 2 01:11 libpng16.so.16.34.0

lrwxrwxrwx 1 root root 11 Apr 2 01:11 libpng.la -> libpng16.la

lrwxrwxrwx 1 root root 11 Apr 2 01:11 libpng.so -> libpng16.so

最终都是使用libpng16.so.16.34.0

具体复制哪几个库还不确定

等到实验后确定到底需要哪几个库。

编译程序需要的头文件

root@ubuntu:/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/include# ls

jconfig.h jerror.h jmorecfg.h jpeglib.h libpng16 minigui pngconf.h png.h pnglibconf.h turbojpeg.h

# 8 单独移植minigui

**Zlib 移植**

./configure --host= arm-rockchip-linux --prefix=/home/chao/minigui/build

Make

Make install

**Freetype移植：**

1> ./configure --host= arm-rockchip-linux --prefix=/home/chao/minigui/build

2>gedit Makefile

CC=arm-rockchip-linux-gnueabihf-gcc

CXX=arm-rockchip-linux-gnueabihf-g++

LD=arm-rockchip-linux-gnueabihf-ld

AS=arm-rockchip-linux-gnueabihf-as

3> make

4> make install

**Jpeg-7移植**

1>

./configure --enable-shared --host=arm-rockchip-linux --prefix=/home/chao/minigui/jpeg-7/libtest CC=arm-rockchip-linux-gnueabihf-gcc CXX=arm-rockchip-linux-gnueabihf-g++ LD=arm-rockchip-linux-gnueabihf-ld AS=arm-rockchip-linux-gnueabihf-as

**移植libpng**

1>

export CFLAGS="-I/home/chao/minigui/build/include/"

export LDFLAGS="-L/home/chao/minigui/build/lib"

2>

./configure --host=arm-rockchip-linux CC=arm-rockchip-linux-gnueabihf-gcc LD=arm-rockchip-linux-gnueabihf-ld AS=arm-rockchip-linux-gnueabihf-as AR=arm-rockchip-linux-gnueabihf-ar CXX=arm-rockchip-linux-gnueabihf-g++ --enable-shared --prefix=/home/chao/minigui/build

3> make

4> make install

如果使用过程发现格式不支持错误，用更新版本库替换

**移植minigui-res-be-3.0.12**

./configure --host=arm-rockchip-linux CC=arm-rockchip-linux-gnueabihf-gcc LD=arm-rockchip-linux-gnueabihf-ld AS=arm-rockchip-linux-gnueabihf-as AR=arm-rockchip-linux-gnueabihf-ar CXX=arm-rockchip-linux-gnueabihf-g++ --prefix=/home/chao/minigui/build

Make

Make install //安装到build/share里面

**移植libminigui**

./configure --prefix=/home/chao/minigui/build --host=arm-rockchip-linux --enable-share --with-targetname=fbcon --enable-video-fbcon --disable-static --enable-lite --enable-rbf16 --enable-tinyscreen --disable-cursor --disable-textmode --enable-tslibial  --enable-pngsupport --enable-jpegsupport --disable-svescreen CC=arm-rockchip-linux-gnueabihf-gcc LD=arm-rockchip-linux-gnueabihf-ld AS=arm-rockchip-linux-gnueabihf-as AR=arm-rockchip-linux-gnueabihf-ar CXX=arm-rockchip-linux-gnueabihf-g++

修改：gedit src/newgal/pcxvfb/Makefile 的194行，将/usr/include去掉

INCLUDES = -I$(abs\_top\_srcdir)/src/include -I$(abs\_top\_srcdir)/include \

-I$(abs\_top\_srcdir)/src/newgal/ **-I/usr/include/**

否则编译pcxvfb.c将出错

修改Makefile:添加

跟踪代码发现，这里应该要显示飞漫的启动动画的。 配置png支持的宏没有打开，最好先编译安装飞漫官方的png库， 配置\_MGIMAGE\_PNG, 重新编译核心库gpl，

./configure --enable-pngsupport  --enable这里其实jpeg库也没支持，先不管，用到再说，这时候编译same会出错，因为Makefile中

没有添加png的链接，Makefile中添加  LIBS +=-lpng -lts –ljpeg

上面是支持png,tslib和jpeg

为了解决如下错误

NEWGAL>FBCON:Error when setting the terminal to graphics mode: Inappropriate ioctl for device

添加--disable-textmode：--disable-textmode

配置文件mgconfig.h添加如下支持jpg和png

#define \_MGIMAGE\_PNG 1

#define \_MGIMAGE\_JPG 1

所有包含\_MGIMAGE\_PNG 和\_MGIMAGE\_JPG的文件都要加上#inlcude “mgconfig.h”

而且每次配置完毕都要检查mgconfig.里面这两个宏是否定义，发现重配置后mgconfig会改变

要记住同时删除库里面的include/minigui目录和libminigui库

编译程序出现

未定义引用错误WndClientRect，找到方法了：  
$cd libminigui-3.0.12-linux  
$grep -r "WndClientRect" .  
发现WndClientRect在libminigui-3.0.12-linux/src/newgdi/gdi.c中，将inline void WndRect(HWND hWnd, PRECT prc)和inline void WndClientRect(HWND hWnd, PRECT prc)的inline去掉，再make clean;make;make install重新生成库，再去编例子就ok了。还有一个函数pix2rgb一样方法处理。

找不到tslib.h，在tslibal.c添加绝tslib.h的绝对路径

/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/build/tslib-1.15/src

Jpeg.c里面

找不到jpeglib.h,jerror.h

</home/chao/minigui/build/include/jpeglib.h>

#include </home/chao/minigui/build/include/jerror.h>

编译应用程序时提示：

Unable find “ts\_open” “ts\_fd”

修改应用run.sh编译脚本添加：

arm-rockchip-linux-gnueabihf-gcc hello.c -I/home/chao/minigui/build/include/ -L/home/chao/minigui/build/lib -lminigui\_ths -ljpeg -lm -lpthread -ldl -lpng -L/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/usr/lib -lts

如果出现如下错误：

KERNEL>InitGUI: Count not init mouse cursor!

KERNEL>InitGUI: Init failure, please check your MiniGUI configuration or resource.

缺少资源文件,资源路径不对，可能是最后少一个/，应用层缺少

修改MiniGui.cfg

# Edit following line to specify cursor files path

cursorpath=/userdata/minigui/res/cursor/

[resinfo]

respath=/userdata/minigui/

拷贝minigui到userdata/minigui命令

adb push “minigui/” “userdata/minigui”

禁止鼠标，启动画面和屏保

./configure --prefix=/home/chao/minigui/build --host=arm-rockchip-linux --with-targetname=fbcon --disable-static --disable-cursor --disable-textmode --enable-tslibial  --enable-pngsupport --enable-jpgsupport --disable-savescreen --disable-splash --disable-screensaver --disable-mousecalibrate --disable-savebitmap --disable-consoleps2 --disable-consoleimps2 --disable-consolems --disable-consolems3 --disable-consolegpm CC=arm-rockchip-linux-gnueabihf-gcc LD=arm-rockchip-linux-gnueabihf-ld AS=arm-rockchip-linux-gnueabihf-as AR=arm-rockchip-linux-gnueabihf-ar CXX=arm-rockchip-linux-gnueabihf-g++

cp src/newgal/pcxvfb/Makefilebackup src/newgal/pcxvfb/Makefile

cp Makefilebackup Makefile

# 9移植openssl和curl

编译openssl1.0.0g

因为curl需要openssl1.0.0，不能用1.1.0版本。而版本过低会导致

找不到SSL\_COMP\_free\_compression\_methods();

1>./config no-asm shared --prefix=/home/chao/curl/build/

2>修改Makefile

CC= arm-rockchip-linux-gnueabihf-gcc

去掉所有-m64,

3> make

4> make install

no-asm： 是在交叉编译过程中不使用汇编代码代码加速编译过程，原因是它的汇编代码是对arm格式不支持的。

shared ：生成动态连接库。

--prefix ：指定make install后生成目录的路径

如果在linux下安装openssl，执行config和make之后，在执行make install时如果出现下面的错误

cms.pod around line 457: Expected text after =item, not a number

cms.pod around line 461: Expected text after =item, not a number   
cms.pod around line 465: Expected text after =item, not a number   
cms.pod around line 470: Expected text after =item, not a number   
cms.pod around line 474: Expected text after =item, not a number

POD document had syntax errors at /usr/bin/pod2man [line](http://cpro.baidu.com/cpro/ui/uijs.php?c=news&cf=1001&ch=0&di=128&fv=17&jk=51aa8e399b449d99&k=line&k0=line&kdi0=0&luki=1&n=10&p=baidu&q=wangyuead_cpr&rb=0&rs=1&seller_id=1&sid=999d449b398eaa51&ssp2=1&stid=0&t=tpclicked3_hc&tu=u1828791&u=http%3A%2F%2Fwww.myexception.cn%2Fprogram%2F1811081.html&urlid=0) 69.

则在root权限下，执行rm -f /usr/bin/pod2man  然后重新make install

Makefile里面的

LIBDIR=lib64改为lib

删除linux-x86\_64

#CONFIGURE\_ARGS=linux-x86\_64 no-asm shared --prefix=/home/chao/minigui/curl/build/

CONFIGURE\_ARGS= no-asm shared --prefix=/home/chao/minigui/curl/build/

编译curl7.50

./configure --enable-shared --host=arm-rockchip-linux --prefix=/home/chao/curl/build/ CC=arm-rockchip-linux-gnueabihf-gcc CXX=arm-rockchip-linux-gnueabihf-g++

Make&&Make install

# 10 显示logo

## 10.1 uboot显示logo

函数调用流程分析：

Uboot 中显示logo的函数在

int board\_late\_init(void)

{

#ifdef CONFIG\_DRM\_ROCKCHIP

rockchip\_show\_logo();

#endif

}

由于我们不使用DRM,而logo是必需的，因此需要去掉CONFIG\_DRM\_ROCKCHIP宏限制

static init\_fnc\_t init\_sequence\_r[] = {

#ifdef CONFIG\_BOARD\_LATE\_INIT

board\_late\_init,

#endif

}

void board\_init\_r(gd\_t \*new\_gd, ulong dest\_addr)

{

initcall\_run\_list(init\_sequence\_r)

}

10.2 分析显示logo的函数

void rockchip\_show\_logo(void)

{

struct display\_state \*s;

list\_for\_each\_entry(s, &rockchip\_display\_list, head) {

s->logo.mode = s->logo\_mode;

//查找logo

if (load\_bmp\_logo(&s->logo, s->ulogo\_name))

printf("failed to display uboot logo\n");

else

display\_logo(s);

/\* Load kernel bmp in rockchip\_display\_fixup() later \*/

}

}

如果想使用这套函数就只能移植到自己的驱动上面，比较繁琐。

## 10.3 测试uboot的spi驱动是否可用

Drivers/spi/rk\_spi.c添加如下：

static const struct udevice\_id rockchip\_spi\_ids[] = {

{ .compatible = "rockchip,rk3288-spi" },

{ .compatible = "rockchip,rk3368-spi" },

{ .compatible = "rockchip,rk3399-spi" },

{ .compatible = "rockchip,rk3308-spi" },

{ }

};

配置uboot

[\*] Rockchip SPI driver

Rk3308\_common.h里面打开显示调试信息

#define DEBUG

以后要去掉，影响启动速度。

## 10.2 内核显示logo

10.2.1 采用将图片数据直接加载到代码中的显示方式，这样方式简单。

# 11 裁剪内存

Cache和buffer

Cache是读磁盘时缓存，数据从硬盘发送到缓存，应用从缓存获取数据。

Buffer：是写磁盘的缓存。应用写的数据先写到buffer,然后从buffer中再写到硬盘。

因为cpu和硬盘速度差异过大，使用内存作为缓冲。

# free -m

total used free shared buffers cached

Mem: 59320 34200 25120 136 6564 10640

-/+ buffers/cache: 16996 42324

第一部分Mem行：

total：内存总数

used：已经使用的内存数

free：空闲的内存数

shared：可用的共享内存

buffers：内存缓冲数

cached：内存缓存数

第二部分:(-/+ buffers/cache)

used: 除去被用作buffers和cache内存后已用的内存

free：用作buffers和cache的内存加上Mem部分空闲的内存数

第三部分:（Swap）

用一部分磁盘当做内存用的“内存”

在我们来做一下数据统计

total=Mem\_used+Mem\_free

Mem\_used(34200)=Mem\_buffers(10640+6564)+Mem\_cached+(-/+buffers/cache)\_used1(6996)

Mem\_free=(-/+ buffers/cache)\_free-Mem\_buffers-Mem\_cached

如何减少缓存的占用呢？

在/etc/init.d/rcS里面添加如下

 echo 1 > /proc/sys/vm/drop\_caches

ulimit -s 4096 指定线程栈4096

修改后板子内存分配

# free -m

total used free shared buffers cached

Mem: 59320 25000 34320 132 404 6932

-/+ buffers/cache: 17664 41656

Swap: 0 0 0

删除adbd:adb调试使用

ntpd：时间同步

# 12 PWM背光

## 12.1 配置内核支持pwm背光驱动

驱动文件所在位置： drivers/video/backlight/pwm\_bl.c

Graphics support

[\*] Backlight & LCD device support

<\*> Lowlevel Backlight controls

<\*> Generic PWM based Backlight Driver

## 12.2 配置pwm背光设备节点

rk3308-voice-module-amic-mainboard-v10-aarch32设备树文件中添加背光设备节点：

在根节点配置。具体可以看rk的手册解释了pwm背光如何使用。

/ {

backlight: backlight {

compatible = "pwm-backlight";

pwms = <&pwm0 0 25000 0>;

brightness-levels = <

0 1 2 3 4 5 6 7

8 9 10 11 12 13 14 15

16 17 18 19 20 21 22 23

24 25 26 27 28 29 30 31

32 33 34 35 36 37 38 39

40 41 42 43 44 45 46 47

48 49 50 51 52 53 54 55

56 57 58 59 60 61 62 63

64 65 66 67 68 69 70 71

72 73 74 75 76 77 78 79

80 81 82 83 84 85 86 87

88 89 90 91 92 93 94 95

96 97 98 99 100 101 102 103

104 105 106 107 108 109 110 111

112 113 114 115 116 117 118 119

120 121 122 123 124 125 126 127

128 129 130 131 132 133 134 135

136 137 138 139 140 141 142 143

144 145 146 147 148 149 150 151

152 153 154 155 156 157 158 159

160 161 162 163 164 165 166 167

168 169 170 171 172 173 174 175

176 177 178 179 180 181 182 183

184 185 186 187 188 189 190 191

192 193 194 195 196 197 198 199

200 201 202 203 204 205 206 207

208 209 210 211 212 213 214 215

216 217 218 219 220 221 222 223

224 225 226 227 228 229 230 231

232 233 234 235 236 237 238 239

240 241 242 243 244 245 246 247

248 249 250 251 252 253 254 255>;

default-brightness-level = <100>;};

}

## 12.2.3 背光驱动

# 13 wifi 配置

## 13.1配置设备树

rk3308-voice-module-mainboard-v10-aarch32.dtsi wireless-wlan {

compatible = "wlan-platdata";

rockchip,grf = <&grf>;

clocks = <&cru SCLK\_WIFI>;

clock-names = "clk\_wifi";

ref-clock-frequency = <24000000>;

pinctrl-names = "default";

pinctrl-0 = <&wifi\_wake\_host>;

//wifi\_chip\_type = "rtl8723ds";

wifi\_chip\_type = "rtl8189fs";//具体名字通过内核搜索rtl8189\*找到

WIFI,host\_wake\_irq = <&gpio0 RK\_PA0 GPIO\_ACTIVE\_LOW>;

status = "okay";

}

## 13.2 配置内核

Device Drivers

[\*] Network device support

[\*] Wireless LAN

[\*] Network device support

[\*] Wireless LAN

[\*] Rockchip Wireless LAN support

<\*> Realtek 8189F SDIO WiFi

13.3 命令行配网

### 13.3.1 buildroot选择配置文件

根据对应WiFi选择相应配置：

Target packages

[\*] rockchip BSP packages

[\*] rkwifibt

wifi chip support (RTL8189FS)

### 13.3.2 串口终端输入

1>查看是否启动了wpa\_supplicant，并且找到配置文件

ps|grep wpa\_supplicant

282 root 3548 S wpa\_supplicant -B -i wlan0 -c /data/cfg/wpa\_supplica

2>修改配置文件

#cd /data/cfg/wpa\_supplicant.conf

# cat wpa\_supplicant.conf

ctrl\_interface=/var/run/wpa\_supplicant

update\_config=1

ap\_scan=1

network={

ssid="ZGWLCS"

psk="ZGWLceshi"

key\_mgmt=WPA-PSK

}

3>重新连接wifi

wpa\_cli reconnect

# 13.4 驱动源码

rfkill-wlan.c

# 14 高精度定时器hrtimer

### 14.1 hrtimer函数的简单介绍

**hrtimer\_init()**

说明：初始化定时器工作模式。

原型：hrtimer\_init(&timer, CLOCK\_MONOTONIC, HRTIMER\_MODE\_REL);

参数：

CLOCK\_MONOTONIC：从系统启动这一刻起开始计时,不受系统时间被用户改变的影响。

CLOCK\_REALTIME:系统实时时间,随系统实时时间改变而改变。

HRTIMER\_MODE\_REL：相对时间

设置回调函数：

timer.function = timer\_func;

定时器到时该函数将被调用。

回调函数原型：static enum hrtimer\_restart timer\_func(struct hrtimer \*timer)

//注：该回调函数为原子操作不能被中断

**hrtimer\_start()**

说明：启动定时器

原型：hrtimer\_start hrtimer\_start(&timer, ktime\_set(value / 1000, (value % 1000) \* 1000000),HRTIMER\_MODE\_REL);.

返回值：enum hrtimer\_restart {

HRTIMER\_NORESTART,  /\* Timer is not restarted \*/

HRTIMER\_RESTART,  /\* Timer must be restarted \*/

};

**hrtimer\_cancel()**

说明：取消一个hrtimer

原型：

int hrtimer\_cancel(struct hrtimer \*timer);

### 14.2 hrtimer结构

**hrtimer结构体**

struct hrtimer {

struct timerqueue\_node node;

ktime\_t \_softexpires;

enum hrtimer\_restart (\*function)(struct hrtimer \*);

struct hrtimer\_clock\_base \*base;

unsigned **long** state;

......

};

**State：**标识当前定时器状态：

#define HRTIMER\_STATE\_INACTIVE 0x00 // 定时器未激活

#define HRTIMER\_STATE\_ENQUEUED 0x01 // 定时器已经被排入红黑树中

#define HRTIMER\_STATE\_CALLBACK 0x02 // 定时器的回调函数正在被调用

#define HRTIMER\_STATE\_MIGRATE 0x04 // 定时器正在CPU之间做迁移

hrtimer的到期时间：

可以基于以下几种时间基准系统

enum hrtimer\_base\_type {

HRTIMER\_BASE\_MONOTONIC,  // 单调递增的monotonic时间，不包含休眠时间

HRTIMER\_BASE\_REALTIME,  // 平常使用的墙上真实时间

HRTIMER\_BASE\_BOOTTIME,  // 单调递增的boottime，包含休眠时间

HRTIMER\_MAX\_CLOCK\_BASES,  // 用于后续数组的定义

};

# 14.3 hrtimer使用

添加一个定时器：

void hrtimer\_init(struct hrtimer \*timer, clockid\_t which\_clock,

enum hrtimer\_mode mode);

which\_clock可以是CLOCK\_REALTIME、CLOCK\_MONOTONIC、CLOCK\_BOOTTIME中的一种，mode则可以是相对时间HRTIMER\_MODE\_REL，也可以是绝对时间HRTIMER\_MODE\_ABS。设定回调函数：

timer.function = hr\_callback;

如果定时器无需指定一个到期范围，可以在设定回调函数后直接使用hrtimer\_start激活该定时器：

**int** hrtimer\_start(struct hrtimer \*timer, ktime\_t tim,

const enum hrtimer\_mode mode);

如果需要指定到期范围，则可以使用hrtimer\_start\_range\_ns激活定时器

hrtimer\_start\_range\_ns(struct hrtimer \*timer, ktime\_t tim,

unsigned **long** range\_ns, const enum hrtimer\_mode mode);

要取消一个hrtimer，使用hrtimer\_cancel：

**int** hrtimer\_cancel(struct hrtimer \*timer);

以下两个函数用于推后定时器的到期时间：

extern u64 hrtimer\_forward(struct hrtimer \*timer, ktime\_t now, ktime\_t interval);

/\* Forward a hrtimer so it expires after the hrtimer's current now \*/

static inline u64 hrtimer\_forward\_now(struct hrtimer \*timer,

ktime\_t interval)

{

return hrtimer\_forward(timer, timer->base->get\_time(), interval);

}

# 15 异步通知

Linux设备驱动开发详解和其源码

# 16 红外感应开关

## 16.1 设计思路

红外发射电路以1Khz频率发射红外线，红外接收器通过接收的红外线拉高拉低电平，如果在1s内接收到800以上的电平反转，就认为有人。

发射端驱动程序采用高精度定时器，500us翻转一次IO口电平。

接收端采用中断方式对io口计数，上升沿中断计数一次。

1s判断一次计数值。实现方法为在高精度定时器里面做时间计数，每当计数到2000次就是1s，此时就判断一次中断计数，如果计数值超过800就认为检测到人。

## 16.2 设备树配置

infrared: infrared {

compatible = "infrared-switch";

interrupts = <&gpio0 RK\_PC2 IRQ\_TYPE\_EDGE\_RISING>;

reset-gpios=<&gpio2 RK\_PA3 GPIO\_ACTIVE\_LOW>;

}

## 16.3 驱动设计

## 16.4 测试

检测到设备树

创建了设备节点

定时器定时

Pwm脉冲正常

发出了异步通知

# 17 SDIO 卡驱动

# 错误解决

Sh:vbetool not found 发现和input-event-daemon有关

/home/chao/rk3308\_linux/buildroot/output/rockchip\_rk3308\_32\_debug/target/etc/ input-event-daemon.conf是编译时临时从下面文件拷贝到当前文件的

/home/chao/rk3308\_linux/buildroot/board/rockchip/rk3308/fs-overlay/etc/input-event-daemon.conf

因此需要更改

/home/chao/rk3308\_linux/buildroot/board/rockchip/rk3308/fs-overlay/etc/input-event-daemon.conf

注释掉空闲事件

[Idle]

#1h 30m = vbetool dpms off

#reset = vbetool dpms on