目录

[1 Rk3308 简介： 1](#_Toc5279081)

[2使用adb连接板子 2](#_Toc5279082)

[2.1 adb安装 2](#_Toc5279083)

[2.2 使用putty连接板子 2](#_Toc5279084)

[2.3 上传下载文件到开发板 2](#_Toc5279085)

[2.4 更新内核 2](#_Toc5279086)

[3编译下载内核 2](#_Toc5279087)

[3.1 编译uboot(32位) 3](#_Toc5279088)

[3.2 编译内核 3](#_Toc5279089)

[3.3 编译驱动的Makefile（32位内核） 3](#_Toc5279090)

[3.4 板子匹配的设备树 3](#_Toc5279091)

[3.5 添加lcd驱动到内核 3](#_Toc5279092)

[3.6 编译应用的编译器(32位编译器) 4](#_Toc5279093)

[3.7 编译配置buildroot 4](#_Toc5279094)

[3.8 mkfirmware.sh生成固件 4](#_Toc5279095)

[3.9 5](#_Toc5279096)

[4 GPIO口使用 5](#_Toc5279097)

[5 spi 接口Lcd驱动 6](#_Toc5279098)

[5.1 设计思路 6](#_Toc5279099)

[5.2 硬件信息 6](#_Toc5279100)

[4.2.1 rk3308 SPI功能 6](#_Toc5279101)

[5.2.2配置内核支持瑞芯微spi 6](#_Toc5279102)

[5.2.3 lcd 屏接口模式配置 7](#_Toc5279103)

[5.2.4 lcd屏spi接口引脚描述 7](#_Toc5279104)

[5.2 dts配置 7](#_Toc5279105)

[5.3 驱动例程 8](#_Toc5279106)

[5.4 测试 8](#_Toc5279107)

[6 触摸屏ft6236驱动 8](#_Toc5279108)

[6.1 设计思路 8](#_Toc5279109)

[6.2 配置dts 9](#_Toc5279110)

[6.3驱动例程 9](#_Toc5279111)

[6.4 测试 9](#_Toc5279112)

[配置minigui支持 9](#_Toc5279113)

[配置zlog 11](#_Toc5279114)

# 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.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

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

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

## 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

#userdata config

export RK\_USERDATA\_DIR=userdata\_empty

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。

# 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=<&gpio0 RK\_PB6 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。

## 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

## 配置zlog

# 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/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

**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 --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

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"

找不到jpeglib.h,jerror.h

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

#include </home/chao/minigui/build/includejerror.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”