

君正®

X1000 软件开发手册

Version: 4.3

Date: 2017.12.18



北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

君正@

Linux X1000 软件开发手册

Release history

Date	Revision	Revision History
2016-03-28	1.0	- First released
2016-09-26	2.0	- Add loglevel
2016-11-22	3.0	- Add fastboot
2017-01-12	3.0	-Modify description of wifi
2017-04-24	4.0	-Modify Audio && Camera && VPU
2017-05-24	4.1	-Modify Camera&&Add button && Add TouchScreen
2017-05-27	4.1	-Add Network Operating mode && QEMU
2017-07-06	4.1	-Modify Security Module
2017-07-20	4.1	-Modify take pictures and preview commands
2017-09-14	4.2	-Modify Audio && USB Mass_Storage&& Bluetooth
2017-09-22	4.3	-Add Minigui demo
2017-10-03	4.3	-Delete LCD user space description
2017-11-23	4.3	-Add LCD user space
2017-12-14	4.3	-Modify grab && USB Mass_Storage && USB Audio Gadget
2017-12-18	4.3	-Add USB Debug Description

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10, Xibeiwang East Road, Haidian Dist., Beijing, China, 100193

Tel: 86-10-56345000

Fax: 86-10-56345001

<http://www.ingenic.com>

1. 前言.....	4
1.1. 文档目的及背景.....	4
2. 搭建开发环境.....	5
3. Uboot 配置和使用.....	6
3.1. uboot 编译.....	6
3.2. uboot 常用命令.....	7
3.3. uboot 通过 SPI flash 加载内核镜像.....	8
3.4. uboot 通过 tftp 加载内核镜像.....	9
3.5. uboot 通过 fastboot 命令加载内核镜像.....	10
3.6. uboot 挂载文件系统.....	11
3.6.1. 挂载 Jffs2 文件系统.....	11
3.6.2. 挂载网络文件系统.....	11
3.7. log 打印级别 loglevel.....	12
4. Linux 内核驱动和应用.....	13
4.1. 内核配置和编译.....	13
4.2. Xburst 板级介绍.....	13
4.3. GPIO 模块.....	14
4.3.1. 内核空间.....	15
4.3.2. 用户空间.....	15
4.4. I2C 模块.....	18
4.4.1. 内核空间.....	18
4.4.2. 用户空间.....	19
4.5. SPI 模块.....	20
4.5.1. 内核空间.....	20
4.5.2. 用户空间.....	21
4.6. Audio 模块.....	23
4.6.1. 内核空间.....	23
4.6.2. 用户空间.....	25
4.7. Camera 模块(CIM).....	36
4.7.1. 添加 sensor 流程.....	36
4.7.2. 调试方法.....	45
4.7.3. FAQ.....	49
4.8. SD 卡模块.....	51
4.8.1. 内核空间.....	51
4.8.2. 用户空间.....	51
4.9. LCD 模块.....	52
4.9.1. 内核空间.....	52
4.10. MiniGUI demo 的使用.....	54
4.10.1. 驱动配置及编译.....	54
4.10.2. 运行图形界面.....	55
4.11. VPU 模块.....	56
4.11.1. 内核空间.....	56
4.11.2. 用户空间.....	56

4.12. USB 模块.....	58
4.12.1. 内核空间.....	58
4.12.2. usb-host.....	58
4.12.3. usb-device.....	61
4.13. 蓝牙-WIFI 模块.....	65
4.13.1. 蓝牙.....	65
4.13.2. WIFI.....	67
4.14. WIFI AP 模式.....	70
4.14.1. AP 模式使用方法.....	70
4.14.2. AP 模式的验证.....	70
4.14.3. AP 模式切换 wifi 模式.....	72
4.15. Network 工作模式.....	73
4.15.1. 桥接模式.....	73
4.15.2. NAT 模式.....	75
4.16. TouchScreen.....	77
4.16.1. 内核配置.....	77
4.16.2. 代码目录及描述.....	77
4.17. Button.....	79
4.17.1. Kernel 的配置.....	79
4.17.2. Button demo.....	79
4.18. SECURITY 加密模块.....	80
4.18.1. 驱动配置.....	80
4.18.2. IOCTL 命令定义.....	80
4.18.3. 驱动结构体描述.....	81
4.18.4. 编程指导.....	82
4.18.5. 用户 API.....	82
4.19. 语音唤醒.....	84
4.19.1. Voice trigger 驱动配置方法.....	84
4.19.2. 验证方法.....	84
4.20. 快速启动配置.....	86
5. QEMU.....	87
5.1. User Mode.....	87
5.2. System Mode.....	88
5.2.1. 启动脚本.....	88
6. 烧录工具使用.....	90

1. 前言

1.1. 文档目的及背景

君正处理器是高集成度、高性能和低功耗的 32 位 RISC 处理器，带有 MMU 和数据及指令 Cache，以及丰富的外围设备，可以运行 Linux 操作系统。本文将向读者介绍基于君正处理器平台进行 Linux 内核的配置方法和开发过程，引导开发人员快速进行 Linux 开发。本文档为君正内核 3.10 版本开发文档，基于芯片 X1000，不针对具体开发板，文中如有涉及具体开发板型号，是为了说明方便。

在阅读该文档前，需要具备以下基本技能：

- 1.会使用 Linux 系统进行开发，最好是 ubuntu。
- 2.知道嵌入式开发基本流程。如 uboot，linux，文件系统制作等。

阅读该文档，会提供以下帮助：

- 1.帮助理解君正 BSP 基本组成。（uboot,linux,文件系统）
- 2.提供基于君正开发平台创建自己的应用程序方法。
- 3.提供应用程序访问驱动的基本测试用例。

2. 搭建开发环境

在发布 SDK 时，可以使用君正提供的开发平台，该平台包含了 uboot 源码，kernel 源码，交叉工具链和一些测试程序等，基于该平台，可以方便第三方库的添加，方便应用程序开发。详细参考文档《[Manhattan_platform 编译系统使用指南.pdf](#)》完成开发环境搭建。

3. Uboot 配置和使用

Linux 内核需要 U-Boot 来引导。U-Boot 是为嵌入式平台提供的开放源代码的引导程序,它提供串行口、以太网等多种下载方式,提供 NOR 和 NAND 闪存和环境变量管理等功能,支持网络协议栈、JFFS2/EXT2/FAT 文件系统,同时还支持多种设备驱动如 MMC/SD 卡、USB 设备、LCD 驱动等。

3.1. uboot 编译

在进行此步骤前,请确保已经正确配置好交叉编译环境。

针对不同开发板的配置,uboot 的编译配置也不相同,在发布的 uboot 中,编译配置由开发板型号[BOARD_NAME],内核镜像格式[IMAGE_FMT]和启动方式[BOOT]组成,格式如下

[BOARD_NAME]_[IMAGE_FMT]_[BOOT]

具体配置在 uboot/boards.cfg 文件中,可以通过以下命令快速查看开发板支持的编译配置

```
$ cat boards.cfg | grep [BOARD_NAME]
```

根据以上方法,找到对应开发板的编译选项按照以下方式进行编译:

```
$ make distclean
```

```
$ make [BOARD_NAME]_[IMAGE_FMT]_[BOOT]
```

例如开发板 phoenix 的编译配置如下:

phoenix_uImage_msc0 支持 sd 卡启动 uImage 的配置

phoenix_uImage_sfc_nor 支持 nor flash 启动 uImage 的配置

```
$ make distclean
```

```
$ make phoenix_uImage_sfc_nor
```

例如开发板 halley2 的编译配置如下:

halley2_uImage_sfc_nor 支持 nor flash 启动 uImage 的配置

```
$ make distclean
```

```
$ make halley2_uImage_sfc_nor
```

编译完成后会在当前目录下生成 u-boot-with-spl.bin 文件。即最终烧录所需的 uboot 文件。

3.2. uboot 常用命令

打开调试串口，在 uboot 启动过程中，敲击任意按键，打断 uboot 引导镜像过程，进入 uboot shell 环境，uboot 常用命令如下：

“help”命令:该命令查看所有命令,其中“help command”查看具体命令的格式。

“printenv”命令:该命令查看环境变量。

“setenv”命令:该命令设置环境变量。

“saveenv”命令:该命令保存环境变量。

“bootp”命令:该命令动态获取 IP。

“tftpboot”命令:该命令通过 TFTP 协议从网络下载文件运行。

“bootm”命令:该命令从 memory 运行 u-boot 映像。

“go”命令:该命令从 memory 运行应用程序。

“boot”命令:该命令运行 bootcmd 环境变量指定的命令。

“reset”命令:该命令复位 CPU。

“md”命令:显示内存数据。

“mw”命令:修改内存数据。

“cp”命令:内存拷贝命令。

sfc nor 命令:

: “sfcnor read” 从 spi nor flash 中读取数据到内存。

“sfcnor write” 从内存中写数据到 spi nor flash。

“sfcnor erase” spi nor flash 擦出。

Sd 卡命令:

“mmc read” 从 sd 卡中读取数据。

“mmc write” 写数据到 sd 卡。

3.3. uboot 通过 SPI flash 加载内核镜像

1. 一种方法，可以在 uboot 运行的时候，进入 uboot shell，修改 bootcmd 变量如下：

```
# set bootcmd 'sfcnor read 0x40000 0x300000 0x80800000 ;bootm 0x80800000'  
# saveenv
```

2. 另一种方法，修改 uboot 源码，include/configs/[BOARD_NAME].h

例如修改 include/configs/halley2.h 文件中的

```
#define CONFIG_BOOTCOMMAND "sfcnor read 0x40000 0x300000 0x80800000 ;bootm  
0x80800000"
```

注意需要编译支持 Nor flash 启动的 uboot。

3.4. uboot 通过 tftp 加载内核镜像

1. PC 服务器端环境配置

```
$ sudo apt-get install tftpd-hpa
```

```
$ sudo service tftpd-hpa start
```

在 /var/lib/tftpboot 下的文件都可以通过 TFTP 协议下载，可以通过 tftp 客户端程序测试 tftp 服务器是否可以访问。

```
$ sudo apt-get install tftp-hpa          #安装客户端程序
```

```
$tftp 127.0.0.1
```

```
tftp> get test.txt
```

2. 开发板端配置

进入 uboot 环境，根据网络情况设置参数，将需要的 uImage 放入上述服务器文件夹下执行以下命令将 uImage 下载到内存 0x80800000 位置，并启动 tftp 服务，根据实际的 ip 情况，设置 uboot 各个环境变量如下：

```
$set ipaddr 192.168.4.145
```

```
$set serverip 192.168.4.146
```

```
$tftp 0x80800000 uImage
```

```
$bootm 0x80800000
```

3.5. uboot 通过 fastboot 命令加载内核镜像

Fastboot 是使用 USB 端口把 uImage 下载到内存并启动 kernel 命令，只用于调试 kernel 镜像不支持 android fastboot 其他功能。

使用方法：

在 PC 端使用以下命令安装 fastboot 应用程序：

```
$ sudo apt-get install android-tools-fastboot
```

(1) 进入开发板调试串口，在 uboot 启动过程中，敲击任意按键，打断 uboot 引导镜像过程，进入 uboot shell 模式，输入#fastboot 命令：

```
#fastboot
```

(2) 在 PC 端存放 kernel 镜像文件（uImage）的路径下，执行以下命令：

```
$ sudo fastboot boot uImage
```

命令执行成功后，即可下载 kernel 镜像（uImage）到内存并启动 kernel。

3.6. uboot 挂载文件系统

在进行开发时,可以将文件系统存放在不同的介质中,比如 nfs 网络文件系统,spi nor flash jffs2 文件系统, SD 卡 ext4 文件系统等等。针对不同的文件系统,uboot 需要向内核传递参数,即设置 bootargs 环境变量,在内核启动的时候,会根据参数去挂载相应的文件系统。

3.6.1. 挂载 Jffs2 文件系统

```
# set bootargs 'console=ttyS2,115200n8 mem=31M@0x0  
ip=off init=/linuxrc rootfstype=jffs2 root=/dev/mtdblock2 rw'  
以上命令在同一行。  
# saveenv
```

3.6.2. 挂载网络文件系统

3.6.2.1. PC 端配置

挂载网络文件系统,需要在 PC 端安装 NFS 服务器,并且已经将其导出。以 ubuntu 为例,设置如下:

安装 nfs 服务器:

```
$ sudo apt-get install nfs-kernel-server
```

导出/home/user/nfs_root 文件夹

```
$ sudo vi /etc/exports
```

添加以下行:

```
/home/user/nfs_root    *(rw,insecure,no_root_squash,no_subtree_check)
```

启动 nfs 服务

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

测试 nfs 服务

```
mount 192.168.4.146:/home/user/nfs_root/ /mnt
```

如果 mnt 下的文件和/home/user/nfs_root 目录下文件一致说明 nfs 服务器安装成功

3.6.2.2. 开发板端配置

具体 ip 地址以实际开发环境为准。注意内核需要选中 nfs 网络文件系统支持,才可以通过网络挂载文件系统。

```
# set bootargs 'console=ttyS2,115200n8 mem=31M@0x0  
ip=192.168.4.145:192.168.4.1:192.168.4.1:255.255.255.0  
nfsroot=192.168.4.146:/home/user/nfs_root rw'  
以上命令在同一行。  
# saveenv
```

3.7. log 打印级别 loglevel

log 打印共有 8 个级别，从 0~7 依次递增，0 为最低级别，7 为最高级别，相关描述文件在以下路径：

/kernel/Documentation/kernel-parameters.txt

```

1481 loglevel= All Kernel Messages with a loglevel smaller than the
1482 console loglevel will be printed to the console. It can
1483 also be changed with klogd or other programs. The
1484 loglevels are defined as follows:
1485
1486 0 (KERN_EMERG) system is unusable
1487 1 (KERN_ALERT) action must be taken immediately
1488 2 (KERN_CRIT) critical conditions
1489 3 (KERN_ERR) error conditions
1490 4 (KERN_WARNING) warning conditions
1491 5 (KERN_NOTICE) normal but significant condition
1492 6 (KERN_INFO) informational
1493 7 (KERN_DEBUG) debug-level messages
1494
1495

```

在板级头文件定义中有如下的信息（以 halley2 开发板为例）：

板级文件路径：u-boot/include/configs/halley2.h

```

#define CONFIG_DDR_64M 64 /*DDR size 64M*/
#define CONFIG_DDR_32M 32 /*DDR size 32M*/

#define BOOTARGS_COMMON "console=ttyS2,115200n8 mem=32M@0x0 loglevel=7"
#if defined(CONFIG_SPL_NOR_SUPPORT) || defined(CONFIG_SPL_SFC_SUPPORT)

```

默认的 uboot 配置中定义 log 打印级别 loglevel 等级为最高等级 7，可以通过修改相应板级文件的 loglevel 等级来提升启动速度，缩短启动时间，置为 0 时，启动时间可缩短至 100ms 左右。

4. Linux 内核驱动和应用

本章主要介绍内核的各个模块的驱动和相应的用户空间测试方法。通过阅读本章节，旨在对引导内核编译，对内核的各个驱动有基本的了解，对相应的测试程序有一定的了解。

以下介绍的模块，在发布的内核源码中不一定全部包含，可以根据需要，按照文档的说明，自己添加相关驱动的编译配置。

本章节不限定在某个固定的平台，但是为了描述方便，会以 halley2 nor 开发板为例进行介绍。

4.1. 内核配置和编译

在君正发布的 BSP 中，会根据发布的开发板型号，组成 defconfig。内核的 defconfig 一般组成格式如下：

`[board_name]_[media_type]_linux_defconfig`

`[board_name]`：发布开发板名。

`[media_type]`：一般是开发板所使用的存储介质名。例如，nor,spinand 等。

具体使用哪一个作为开发板的默认配置，按照发布为准。

在 arch/mips/configs/目录下可以找到相应的配置文件。

例如，针对 halley2_nor_v10 的开发板，内核提供的默认配置为

`halley2_nor_v10_linux_defconfig`

在 PC 开发环境下执行

```
$ make halley2_nor_v10_linux_defconfig
```

```
$ make uImage
```

会生成 arch/mips/boot/uImage

可以在 defconfig 的基础上，通过 make menuconfig 添加自己的驱动模块，或为内核添加其它的特性。例如：

```
$ make halley2_nor_v10_linux_defconfig
```

```
$ make menuconfig
```

4.2. Xburst 板级介绍

在发布的内核版本中，针对不同的芯片型号，会在 arch/mips/xburst 目录下进行添加，该目录

基本介绍如下：

```

common/          #所有芯片公共部分
core/            #xburst 核心文件
Kconfig
lib/
Makefile
Platform
soc-4775/        #4775 系列板级
soc-m200/        #m200 系列板级
soc-x1000/       #x1000 系列板级
以 halley2 开发板为例，其板级定义在
soc-x1000/chip-x1000/halley2

```

该目录重要文件介绍如下：

```

├── common
│   ├── 43438_bt_power_control.c      #蓝牙电源管理
│   ├── 43438_wlan_device.c
│   ├── 43438_wlan_power_control.c    #wifi 电源管理
│   ├── board_base.c                 # 各个 platform_device 注册
│   ├── board_base.h
│   ├── cim_board.c                  # camera 定义
│   ├── i2c_bus.c                    # 内核 i2c 设备描述
│   ├── keyboard_gpio.c
│   ├── lcd                          # 不同 LCD 屏幕参数配置
│   │   ├── lcd-kfm701a21_1a.c
│   │   └── lcd-truly_tft240240_2_e.c
│   ├── misc.c
│   ├── mmc.c                        # sd 控制器设备描述
│   ├── spi_bus.c                    # 内核 spi 设备描述
│   └── ssv6051_wlan_power_control.c
├── halley2_v10
│   └── board.h                      #所有使用的 GPIO 定义。

```

4.3. GPIO 模块

X1000 的 gpio 控制器有五组，A，B，C，D，Z。

支持输入输出和设备复用功能。内核的 gpio 驱动程序是基于 gpio 子系统架构编写的。应用程序可以使用 gpio_demo 进行测试。内核驱动可以在内核空间使用，也可以通过导出 gpio sys 节点到用户空间，在用户空间进行操作。

4.3.1. 内核空间

4.3.1.1. 文件介绍

gpio 一般在进行开发板设计的时候就已经固定好了，有的 gpio 只能作为设备复用功能管脚，有的 gpio 作为普通的输入输出和中断检测功能，对于固定设备复用的功能管脚在以下文件中定义：

arch/mips/xburst/soc-x1000/include/mach/platform.h

在 arch/mips/xburst/soc-x1000/common/platform.c 会根据驱动配置，选中相应的设备功能管脚。

内核的 gpio 驱动基于 gpio 子系统实现，所以其它驱动程序可以通过内核提供的 libgpio 接口，很方便的进行 gpio 控制，例如 gpio_request_one, gpio_get_value, gpio_set_value 等。

gpio 驱动文件所在位置：

arch/mips/xburst/soc-x1000/common/gpio.c

4.3.1.2. 编译配置

内核通过配置 CONFIG_GPIOLIB 选项可以使用 gpio 功能，默认必须选上。

CONFIG_GPIOLIB:

Symbol: GPIOLIB [=y]

Type : boolean

Prompt: GPIO Support

Location:

-> Device Drivers

Defined at drivers/gpio/Kconfig:38

Depends on: ARCH_WANT_OPTIONAL_GPIOLIB [=n] || ARCH_REQUIRE_GPIOLIB [=y]

通过配置 CONFIG_GPIO_SYSFS 选项，可以将 gpio 导出到用户节点/sys/class/gpio 下，对该节点下的文件操作，可以控制 gpio 输入输出。

Symbol: GPIO_SYSFS [=y]

Type : boolean

Prompt: /sys/class/gpio/... (sysfs interface)

Location:

-> Device Drivers

-> GPIO Support (GPIOLIB [=y])

Defined at drivers/gpio/Kconfig:69

Depends on: GPIOLIB [=y] && SYSFS [=y]

4.3.2. 用户空间

在内核导出 gpio 节点的前提下，可以操作/sys/class/gpio 节点，控制 gpio 输入输出。

/sys/class/gpio/

"export" ... 用户空间可以通过写其编号到这个文件，要求内核导出

一个 GPIO 的控制到用户空间。

例如: 如果内核代码没有申请 GPIO #19, "echo 19 > export" 将会为 GPIO #19 创建一个 "gpio19" 节点。

"unexport" ... 导出到用户空间的逆操作。

例如: "echo 19 > unexport" 将会移除使用 "export" 文件导出的 "gpio19" 节点。

GPIO 信号的路径类似 /sys/class/gpio/gpio42/ (对于 GPIO #42 来说), 并有如下的读/写属性:

/sys/class/gpio/gpioN/

"direction" ... 读取得到 "in" 或 "out"。这个值通常运行写入。

写入 "out" 时, 其引脚的默认输出为低电平。为了确保无故障运行, "low" 或 "high" 的电平值应该写入 GPIO 的配置, 作为初始输出值。注意: 如果内核不支持改变 GPIO 的方向, 或者在导出时内核代码没有明确允许用户空间可以重新配置 GPIO 方向, 那么这个属性将不存在。

"value" ... 读取得到 0 (低电平) 或 1 (高电平)。如果 GPIO 配置为输出, 这个值允许写操作。任何非零值都以高电平看待。

如果引脚可以配置为中断信号, 且如果已经配置了产生中断的模式 (见 "edge" 的描述), 你可以对这个文件使用轮询操作 (poll(2)), 且轮询操作会在任何中断触发时返回。如果你使用轮询操作 (poll(2)), 请在 events 中设置 POLLPRI 和 POLLERR。如果你使用轮询操作 (select(2)), 请在 exceptfds 设置你期望的文件描述符。在轮询操作 (poll(2)) 返回之后, 既可以通过 lseek(2) 操作读取 sysfs 文件的开始部分, 也可以关闭这个文件并重新打开它来读取数据。

"edge" ... 读取得到 "none"、"rising"、"falling" 或者 "both"。将这些字符串写入这个文件可以选择沿触发模式, 会使得轮询操作 (select(2)) 在 "value" 文件中返回。

这个文件仅有在这个引脚可以配置为可产生中断输入引脚时, 才存在。

"active_low" ... 读取得到 0 (假) 或 1 (真)。写入任何非零值可以翻转这个属性的 (读写) 值。已存在或之后通过 "edge" 属性设置了

"rising"

4.4. I2C 模块

X1000 的 i2c 控制器为 SMB，SMB 支持的接口有 i2c,支持 100 Kb/s 和 400 Kb/s。

I2C 接口可连接至 pmu, camera, 通过 i2c 接口进行配置。内核的 i2c 驱动程序是基于 i2c 子系统编写的。应用程序可以使用 i2c_demo 进行测试。可以在内核空间通过 i2c 驱动操作 i2c 接口，也可以在用户空间通过 i2c_msg 操作 i2c 接口。

4.4.1. 内核空间

4.4.1.1. 文件介绍

内核驱动基于 i2c 驱动架构,所有的硬件资源在板级定义。

按照 i2c 驱动编写规范，需要提供 i2c_board_info。

以下介绍驱动对应在内核中的路径。

板级资源定义路径:

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/i2c_bus.c

这里是各种使用 i2c 的设备，比如 camera，pmu。详情见本文档相应章节。

与 X1000 I2C 相关的驱动所在目录和文件说明,忽略目录中存在的其他文件。
driver/i2c/

```

|—— algos
|—— busses
|   |—— i2c-v12-jz.c
|—— i2c-core.c
|—— i2c-dev.c
|—— i2c-smbus.c

```

4.4.1.2. 编译配置

一般发布的软件版本中会默认配置 i2c 驱动,如果需要自己更改 i2c 的编译选项,可以通过以下方式进行配置。

配置以下选项:

CONFIG_I2C_V12_JZ

Symbol: I2C_V12_JZ [=y]

Type : tristate

Prompt: Ingenic SoC based on Xburst arch's I2C controller Driver support

Location:

-> Device Drivers

-> I2C support (I2C [=y])

-> I2C Hardware Bus support

Defined at drivers/i2c/busses/Kconfig:855

Depends on: I2C [=y] && HAS_IOMEM [=y] && MACH_XBURST [=y]

如果需要在用户控件编写 i2c 设备驱动,需要将 i2c 设备节点导出到用户空间 dev/

下,配置以下选项:

CONFIG_I2C_CHARDEV:

Symbol: I2C_CHARDEV [=n]

Type : tristate

Prompt: I2C device interface

Location:

->Device Drivers

-> I2C support (I2C [=y])

Defined at drivers/i2c/Kconfig:38

Depends on: I2C [=y]

4.4.2. 用户空间

在 packages/example/Sample/i2c 下,提供了 i2c 操作的接口。

通过填写 i2c_msg 结构体,通过 ioctl 在用户空间进行 i2c 设备的读写操作。

4.5. SPI 模块

X1000 的 spi 控制器为 SSI, SSI 支持的接口有 Microwire, SSP, SPI。SPI 接口可连接至 spi nor, 支持 spi 读写功能。内核的 spi 驱动程序是基于 spi 子系统架构编写的。应用程序可以使用 spi_demo 进行测试。Spi 总线下, 可以挂普通的 char 设备, 也可以挂 mtd 设备。可以在内核空间通过 spi 驱动操作 spi 接口, 也可以在用户空间通过 spi_ioc_transfer 操作 spi 接口。

4.5.1. 内核空间

4.5.1.1. 文件介绍

内核驱动基于 spi 驱动架构,所有的硬件资源在板级定义。

按照 spi 驱动编写规范, 需要提供 spi_board_info。

以下介绍驱动对应在内核中的路径

板级资源定义路径:

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/spi_bus.c, 这个文件下面是 spi 的设备。

与 X1000 spi 相关的驱动所在目录和文件说明,忽略目录中存在的其他文件。

drivers/spi/

- ├── jz_spi.c
- ├── jz_spi.h
- ├── spi-bitbang.c
- ├── spi.c
- ├── spidev.c

4.5.1.2. 编译配置

一般发布的软件版本中会默认配置 spi 驱动,如果需要自己更改 spi 的编译选项,可以通过以下方式进行配置。

在工作电脑上执行:

```
$ make menuconfig
```

配置以下选项:

(1)CONFIG_JZ_SPI:

SPI driver for Ingenic JZ series SoCs

Symbol: JZ_SPI [=y]

Type : tristate

Prompt: Ingenic JZ series SPI driver

Location:

-> Device Drivers

-> SPI support (SPI [=y])

Defined at drivers/spi/Kconfig:96

Depends on: SPI [=y] && SPI_MASTER [=y] && MACH_XBURST [=y]

Selects: SPI_BITBANG [=y]

配置以下选项，可以把 spi 的设备节点导出到/dev 下，供用户空间使用。

(2)CONFIG_SPI_SPIDEV:

Symbol: SPI_SPIDEV [=y]

Type : tristate

Prompt: User mode SPI device driver support

Location:

-> Device Drivers

-> SPI support (SPI [=y])

Defined at drivers/spi/Kconfig:613

Depends on: SPI [=y] && SPI_MASTER [=y]

4.5.2. 用户空间

在 packages/example/Sample/spi 下，提供了 spi 操作的 demo。

```
#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include "spidev.h"
#include "spi.h"
```

```
class NorDev{
public:
    int InitSpi(unsigned short mode, unsigned short bits, unsigned int speed);
};
```

```
SpiDev spinorflash(0,0);
```

```
int NorDev::InitSpi(unsigned short mode, unsigned short bits, unsigned int speed)
```

```
{
```

```
spinorflash.SpiSetMode(mode);
spinorflash.SpiSetBitsPerWord(bits);
spinorflash.SpiSetMaxSpeed(speed);
}

int main()
{
    unsigned short mode = 0;
    unsigned short bits = 8;
    unsigned int speed = 1000000;
    unsigned int delay = 500;
    unsigned char send_buf[1] = {0x9f,};
    unsigned char recv_buf[3] = {0};

    NorDev nor;

    nor.InitSpi(mode, bits, speed);
    spinorflash.SpiMessageTransfer(send_buf, recv_buf, sizeof(send_buf) + sizeof(recv_buf));

    unsigned int id = (recv_buf[1] << 16) | (recv_buf[2] << 8) | recv_buf[3];
    printf("recv_buf[0]=%x    recv_buf[1]=%x  recv_buf[2]=%x  recv_buf[3]=%x\n",recv_buf[0],
recv_buf[1], recv_buf[2], recv_buf[3]);
    printf("id=%06x\n", id);
}
```

4.6. Audio 模块

X1000 Audio 音频控制器为 AIC, AIC 支持的接口有 I2S, SPDIF。I2S 接口可连接至内部 Codec, 也可连接至外部 Codec。DMIC 控制器支持 DMIC 录音。PCM 控制器支持播放和录音, 一般用作蓝牙通话方面。内核的音频驱动程序是基于 alsa 音频驱动架构编写的。应用程序可以使用 alsa-lib 库编程。

4.6.1. 内核空间

4.6.1.1. 文件介绍

内核驱动基于 alsa 驱动架构, 所有的硬件资源在板级定义。按照 alsa 音频驱动编写规范, 实现声卡驱动需要提供 struct snd_soc_card, 和声卡驱动用于传输数据的 snd_soc_platform_driver, 以及不同的播放或录音设备所需要的 snd_soc_component_driver, 以下介绍驱动对应在内核中的路径

板级资源定义路径:

arch/mips/xburst/soc-x1000/common/platform.c

与 X1000 音频相关的驱动所在目录和文件说明, 忽略目录中存在的其他文件。

sound/soc/ingenic/

```
|—— asoc-board
|   |—— canna_icdc.c
|   |—— canna_icdc.h
|   |—— dorado_icdc.c
|   |—— extcodec-npcp215.c
|   |—— mensa_icdc.c
|   |—— mensa_spdif.c
|   |—— newton2_plus_icdc.c
|   |—— phoenix_icdc.c
|   |—— phoenix_spdif.c
|   |—— watch_icdc.c
|—— asoc-v12
|   |—— asoc-aic-v12.c
|   |—— asoc-aic-v12.h
|   |—— asoc-dma-v12.c
|   |—— asoc-dma-v12.h
|   |—— asoc-dmic-v12.c
|   |—— asoc-dmic-v12.h
|   |—— asoc-i2s-v12.c
|   |—— asoc-pcm-v12.c
|   |—— asoc-pcm-v12.h
```



```

|   └── asoc-spdif-v12.c
└── asoc-v13
|   ├── alsa_play.c
|   ├── asoc-aic-v13.h
|   ├── asoc-dma-dmic.c    #使用 dmic 语音唤醒时 snd_soc_platform_driver 实现。
|   ├── asoc-dma-v13.c    #通用 snd_soc_platform_driver 实现。
|   ├── asoc-dma-v13.h
|   ├── asoc-dmic-module.c #使用 dmic 语音唤醒的 snd_soc_component_driver 实现。
|   ├── asoc-dmic-v13.c    #不使用 dmic 语音唤醒 snd_soc_component_driver 实现。
|   ├── asoc-dmic-v13.h
|   ├── asoc-i2s-v13.c     #i2s 控制器的 snd_soc_component_driver
|   ├── asoc-pcm-v13.c     #pcm 控制器的 snd_soc_component_driver 实现
|   ├── asoc-pcm-v13.h
|   └── asoc-spdif-v13.c
└── icodec
|   ├── dmic_dump.c
|   ├── icdc_d1.c
|   ├── icdc_d1.h
|   ├── icdc_d2.c
|   ├── icdc_d2.h
|   ├── icdc_d3.c         #内部 codec 操作具体实现
|   ├── icdc_d3.h
|   ├── pcm_dump.c
|   └── spdif_dump.c
└── Kconfig
└── Makefile

```

4.6.1.2. 编译配置

一般发布的软件版本中会默认配置音频驱动，如果需要自己更改音频的编译选项，可以通过以下方式进行配置。

在工作电脑上执行：

```
$ make menuconfig
```

配置以下选项：

```
CONFIG_SND_ASOC_INGENIC:
```

Say 'Y' to enable Alsa drivers of xburst.

Symbol: SND_ASOC_INGENIC [=y]

Type : tristate

Prompt: ASoC support for Ingenic

Location:

```
-> Device Drivers
```

-> Sound card support (SOUND [=y])

-> Advanced Linux Sound Architecture (SND [=y])

-> ALSA for SoC audio support (SND_SOC [=y])

4.6.2. 用户空间

开源项目 `alsa-project`, 为基于 `alsa` 编写的音频驱动程序提供了用户空间访问的库 `alsa-lib` 和基于 `alsa-lib` 编写的 `alsa-utils` 工具, 可以用来测试音频的播放和录音功能, 具体使用方法可参考下文。

可访问 <http://www.alsa-project.org> 获取关于 `alsa` 的资料。

`alsa` 库整体占用资源较多, 考虑 X1000 内存资源, 可使用 `tinypalsa` 库替换 `alsa` 库, 实现录音和播放功能。

4.6.2.1. alsa 库

通过交叉编译 `alsa-lib` 和 `alsa-utils`, 可以生成 `amixer`, `aplay` 程序, `amixer` 用于配置复杂的混音功能, 通道切换等, `aplay` 用于播放 `wav` 格式音频文件, 将 `aplay` 重命名为 `arecord`, 使用 `arecord` 即可实现录音功能。

4.6.2.1.1. alsa 工具使用

`amixer`, `aplay`, `arecord` 的使用方法如下:

假设播放的音频文件为 `play.wav`, 录音生成的文件为 `record.wav`, 在开发板上按照以下操作。

(1) 播放音频文件

```
# aplay play.wav
```

(2) 通过 `amic` 录音

```
# arecord -D hw:0,0 -c 2 -f S16_LE -r 44100 -d 10 record.wav
```

单独使用该命令可以在当前目录下生成 10 秒的录音文件 `record.wav`。一般在执行 `arecord` 命令之前, 会通过 `amixer` 设置录音的通道和参数, 命令如下:

```
# amixer cset numid=18,iface=MIXER,name='ADC Mux' 0
# amixer cset numid=4,iface=MIXER,name='Digital Capture Volume' 20
# amixer cset numid=6,iface=MIXER,name='Mic Volume' 3
```

(3) 通过 `LINE-IN` 录音

```
# amixer cset numid=9,iface=MIXER,name='Mix0' 0
# amixer cset numid=10,iface=MIXER,name='Mix1' 3
# amixer cset numid=11,iface=MIXER,name='Mix2' 3
# amixer cset numid=12,iface=MIXER,name='Mix3' 0
# amixer cset numid=13,iface=MIXER,name='Mux0' 1
```

```
# amixer cset numid=14,iface=MIXER,name='Mux2' 0
# amixer cset numid=17,iface=MIXER,name='VDIGITAL BYPASS Switch' 1
```

按照上述配好 LINE-IN 通路后，开发板插上 line in 线，手机播放音乐可通过开发板放出声音，插入耳机敲打 AIC 亦可以从耳机听到声音。

(4) 通过 dmics 录音

通过 dmics 录 8K 采样率的双通道音频数据：

```
# arecord -D hw:0,2 -c 2 -f S16_LE -r 8000 -d 10 record.wav
```

(5) 混响

DAC 数据叠加上ADC 通路的数据流(放音时将录音通路的数据做为背景)：

```
# amixer cset numid=9,iface=MIXER,name='Mix0' 3
# amixer cset numid=10,iface=MIXER,name='Mix1' 0
# amixer cset numid=11,iface=MIXER,name='Mix2' 0
# amixer cset numid=12,iface=MIXER,name='Mix3' 3
# amixer cset numid=13,iface=MIXER,name='Mux0' 0
# amixer cset numid=14,iface=MIXER,name='Mux2' 1
```

录音播放：

```
# aplay xp.wav &
# arecord -D hw:0,0 -c 2 -f S16_LE -r 8000 -d 10 a.wav&
```

此时会将播放的音乐文件xp.wav 录入到开发板中，录入文件名称为a.wav。重启开发板恢复amixer默认通路配置后，播放音频文件：

```
# aplay a.wav &
```

也可不重启开发板，执行以下命令恢复amixer 默认配置后直接播放音频文件：

```
# amixer cset numid=9,iface=MIXER,name='Mix0' 0
# amixer cset numid=10,iface=MIXER,name='Mix1' 0
# amixer cset numid=11,iface=MIXER,name='Mix2' 0
# amixer cset numid=12,iface=MIXER,name='Mix3' 3
# amixer cset numid=13,iface=MIXER,name='Mux0' 0
# amixer cset numid=14,iface=MIXER,name='Mux2' 0
# aplay a.wav &
```

(6) audio 测试工具说明：

1. arecord 录音

-D 参数用于指定音频设备PCM

hw 的第一个参数用来指定声卡号，第二个参数用于指定设备号

-c 用于指定声道数

-f 用于指定数据格式

-r 用于指定采样频率

-d 用于指定录音时间

--help 获取帮助

2. aplay 放音

参数设置与 arecord 一致。

3. amixer 配置选项：

(1) 查看配置选项命令: # amixer controls

```
numid=1,iface=MIXER,name='Master Playback Volume'
numid=6,iface=MIXER,name='Mic Volume'
numid=7,iface=MIXER,name='ADC High Pass Filter Switch'
numid=18,iface=MIXER,name='ADC Mux'
numid=3,iface=MIXER,name='Playback Mixer Volume'
numid=5,iface=MIXER,name='Digital Capture Mixer Volume'
numid=4,iface=MIXER,name='Digital Capture Volume'
numid=8,iface=MIXER,name='Digital Playback mute'
numid=9,iface=MIXER,name='Mix0'
numid=10,iface=MIXER,name='Mix1'
numid=11,iface=MIXER,name='Mix2'
numid=12,iface=MIXER,name='Mix3'
numid=13,iface=MIXER,name='Mux0'
numid=14,iface=MIXER,name='Mux2'
numid=2,iface=MIXER,name='TITANIUM Playback Volume'
numid=17,iface=MIXER,name='VDIGITAL BYPASS Switch'
numid=15,iface=MIXER,name='dmic gain'
numid=16,iface=MIXER,name='dmic-dma soft mute'
```

(2) 查看配置选项及参数命令: # amixer contents

```
numid=1,iface=MIXER,name='Master Playback Volume'
; type=INTEGER,access=rw---R--,values=2,min=0,max=31,step=0
: values=31,31
| dBscale-min=-31.00dB,step=0.00dB,mute=0
numid=6,iface=MIXER,name='Mic Volume'
; type=INTEGER,access=rw---R--,values=1,min=0,max=4,step=0
: values=0
| dBscale-min=0.00dB,step=1.00dB,mute=0
numid=7,iface=MIXER,name='ADC High Pass Filter Switch'
; type=BOOLEAN,access=rw-----,values=1
: values=off
numid=18,iface=MIXER,name='ADC Mux'
; type=ENUMERATED,access=rw-----,values=1,items=2
; Item #0 'AMIC ON'
; Item #1 'DMIC ON'
: values=0
numid=3,iface=MIXER,name='Playback Mixer Volume'
; type=INTEGER,access=rw---R--,values=2,min=0,max=31,step=0
: values=0,0
| dBscale-min=-31.00dB,step=0.00dB,mute=0
numid=5,iface=MIXER,name='Digital Capture Mixer Volume'
; type=INTEGER,access=rw---R--,values=2,min=0,max=31,step=0
```

```

: values=0, 0
| dBscale-min=-31.00dB, step=0.00dB, mute=0
numid=4, iface=MIXER, name='Digital Capture Volume'
: type=INTEGER, access=rw---R--, values=2, min=0, max=43, step=0
: values=0, 0
| dBscale-min=0.00dB, step=1.00dB, mute=0
numid=8, iface=MIXER, name='Digital Playback mute'
: type=BOOLEAN, access=rw-----, values=1
: values=on
numid=9, iface=MIXER, name='Mix0'
: type=ENUMERATED, access=rw-----, values=1, items=4
: Item #0 'normal inputs'
: Item #1 'cross inputs'
: Item #2 'mixed inputs'
: Item #3 '0 inputs'
: values=0
numid=10, iface=MIXER, name='Mix1'
: type=ENUMERATED, access=rw-----, values=1, items=4
: Item #0 'normal inputs'
: Item #1 'cross inputs'
: Item #2 'mixed inputs'
: Item #3 '0 inputs'
: values=0
numid=11, iface=MIXER, name='Mix2'
: type=ENUMERATED, access=rw-----, values=1, items=4
: Item #0 'normal inputs'
: Item #1 'cross inputs'
: Item #2 'mixed inputs'

: Item #3 '0 inputs'
: values=16
numid=12, iface=MIXER, name='Mix3'
: type=ENUMERATED, access=rw-----, values=1, items=4
: Item #0 'normal inputs'
: Item #1 'cross inputs'
: Item #2 'mixed inputs'
: Item #3 '0 inputs'
: values=0
numid=13, iface=MIXER, name='Mux0'
: type=ENUMERATED, access=rw-----, values=1, items=2
: Item #0 'Playback DAC only'
: Item #1 'Playback DAC + ADC'
: values=0
numid=14, iface=MIXER, name='Mux2'

```

```
; type=ENUMERATED, access=rw-----, values=1, items=2
; Item #0 'Record input only'
; Item #1 'Record input + DAC'
: values=16
numid=2, iface=MIXER, name='TITANIUM Playback Volume'
; type=INTEGER, access=rw---R--, values=2, min=0, max=31, step=0
: values=31, 31
| dBscale-min=-31.00dB, step=0.00dB, mute=0
numid=17, iface=MIXER, name='VDIGITAL BYPASS Switch'
; type=BOOLEAN, access=rw-----, values=1
: values=off
numid=15, iface=MIXER, name='dmic gain'
; type=INTEGER, access=rw---R--, values=1, min=0, max=15, step=0
: values=8
| dBscale-min=0.00dB, step=3.00dB, mute=0
numid=16, iface=MIXER, name='dmic-dma soft mute'
; type=BOOLEAN, access=rw-----, values=1
: values=off
```

4.6.2.1.2. 基于 alsa 库应用程序编写

alsa 为标准的音频架构，开源网站 <http://www.alsa-project.org> 有详细的教程可供参考，这里从网站摘取部分源码，仅供参考。

一个典型的应用程序一般按照以下流程，调用 alsa lib 接口。

```
open_the_device();
set_the_parameters_of_the_device();
while (!done) {
    /* one or both of these */
    receive_audio_data_from_the_device();
    deliver_audio_data_to_the_device();
}
close the device
```

以下提供一个简单的播放程序源码和录音程序源码，基本上使用了 alsa-lib 的常用 API。

一个简单的基于 alsa-lib 的播放程序，配置为立体声，16bit，44.1KHZ。

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
```

```
main (int argc, char *argv[])
{
    int i;
    int err;
    short buf[128];
    snd_pcm_t *playback_handle;
    snd_pcm_hw_params_t *hw_params;

    if ((err = snd_pcm_open (&playback_handle, argv[1], SND_PCM_STREAM_PLAYBACK,
0)) < 0) {
        fprintf(stderr, "cannot open audio device %s (%s)\n",
            argv[1],
            snd_strerror (err));
        exit (1);
    }

    if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
        fprintf(stderr, "cannot allocate hardware parameter structure (%s)\n",
            snd_strerror (err));
        exit (1);
    }

    if ((err = snd_pcm_hw_params_any (playback_handle, hw_params)) < 0) {
        fprintf(stderr, "cannot initialize hardware parameter structure (%s)\n",
            snd_strerror (err));
        exit (1);
    }

    if ((err = snd_pcm_hw_params_set_access (playback_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
        fprintf(stderr, "cannot set access type (%s)\n",
            snd_strerror (err));
        exit (1);
    }

    if ((err = snd_pcm_hw_params_set_format (playback_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
        fprintf(stderr, "cannot set sample format (%s)\n",
            snd_strerror (err));
        exit (1);
    }

    if ((err = snd_pcm_hw_params_set_rate_near (playback_handle, hw_params, 44100, 0)) < 0)
{
```

```
        fprintf(stderr, "cannot set sample rate (%s)\n",
                snd_strerror(err));
        exit(1);
    }

    if ((err = snd_pcm_hw_params_set_channels(playback_handle, hw_params, 2)) < 0) {
        fprintf(stderr, "cannot set channel count (%s)\n",
                snd_strerror(err));
        exit(1);
    }

    if ((err = snd_pcm_hw_params(playback_handle, hw_params)) < 0) {
        fprintf(stderr, "cannot set parameters (%s)\n",
                snd_strerror(err));
        exit(1);
    }

    snd_pcm_hw_params_free(hw_params);

    if ((err = snd_pcm_prepare(playback_handle)) < 0) {
        fprintf(stderr, "cannot prepare audio interface for use (%s)\n",
                snd_strerror(err));
        exit(1);
    }

    for (i = 0; i < 10; ++i) {
        if ((err = snd_pcm_writei(playback_handle, buf, 128)) != 128) {
            fprintf(stderr, "write to audio interface failed (%s)\n",
                    snd_strerror(err));
            exit(1);
        }
    }

    snd_pcm_close(playback_handle);
    exit(0);
}
```

一个简单的录音程序示例:

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

main (int argc, char *argv[])
{
```



```
int i;
int err;
short buf[128];
snd_pcm_t *capture_handle;
snd_pcm_hw_params_t *hw_params;

if ((err = snd_pcm_open (&capture_handle, argv[1], SND_PCM_STREAM_CAPTURE, 0)) <
0) {
    fprintf(stderr, "cannot open audio device %s (%s)\n",
            argv[1],
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf(stderr, "cannot allocate hardware parameter structure (%s)\n",
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params_any (capture_handle, hw_params)) < 0) {
    fprintf(stderr, "cannot initialize hardware parameter structure (%s)\n",
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params_set_access (capture_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf(stderr, "cannot set access type (%s)\n",
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params_set_format (capture_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf(stderr, "cannot set sample format (%s)\n",
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params_set_rate_near (capture_handle, hw_params, 44100, 0)) < 0) {
    fprintf(stderr, "cannot set sample rate (%s)\n",
            snd_strerror (err));
    exit (1);
}
```

```
}

if ((err = snd_pcm_hw_params_set_channels (capture_handle, hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count (%s)\n",
            snd_strerror (err));
    exit (1);
}

if ((err = snd_pcm_hw_params (capture_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n",
            snd_strerror (err));
    exit (1);
}

snd_pcm_hw_params_free (hw_params);

if ((err = snd_pcm_prepare (capture_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
            snd_strerror (err));
    exit (1);
}

for (i = 0; i < 10; ++i) {
    if ((err = snd_pcm_readi (capture_handle, buf, 128)) != 128) {
        fprintf (stderr, "read from audio interface failed (%s)\n",
                snd_strerror (err));
        exit (1);
    }
}

snd_pcm_close (capture_handle);
exit (0);
}
```

4.6.2.2. tinyalsa 库

4.6.2.2.1. 源码介绍

Tinyalsa 是上层基于 alsa 接口实现的简易 alsa 库，能够进行播放和录音，库的体积较小，适合在内存资源比较紧张的系统上运行。

tinlyalsa 编译后的库的大小：libtinyalsa.so 约为 30kB。

Tinyalsa 可以使用以下命令从 github 上下载：

```
git clone https://github.com/tinyalsa/tinyalsa.git
```

交叉编译 tinyalsa 后，会生成 tinymix, tinypcminfo, tinyplay, tinycap 可执行程序 and libtinyalsa.so 库，通过这些执行程序，可以进行录音和播放测试。例如：

录音：

```
# tinycap record.wav -D 0 -d 2 -c 2 -r 8000 -b 16
```

播放：

```
# tinyplay play.wav
```

4.6.2.2.2. 基于 tinyalsa 应用程序编写

与 alsa 库的应用程序编写流程一样，都是按照

1. 打开设备文件
2. 配置参数
3. 读数据或写数据
4. 结束

具体 API 使用方法可以参考源文件 tinymix.c, tinypcminfo.c, tinyplay.c, tinycap.c。

以下贴出代码片段，详细源码请查看上述源文件。

```
void play_sample(FILE *file, unsigned int card, unsigned int device, unsigned int channels,
                 unsigned int rate, unsigned int bits, unsigned int period_size,
                 unsigned int period_count)
{
    struct pcm_config config;
    struct pcm *pcm;
    char *buffer;
    int size;
    int num_read;

    memset(&config, 0, sizeof(config));
    config.channels = channels;
    config.rate = rate;
    config.period_size = period_size;
    config.period_count = period_count;
    if (bits == 32)
```

```
        config.format = PCM_FORMAT_S32_LE;
    else if (bits == 16)
        config.format = PCM_FORMAT_S16_LE;
    config.start_threshold = 0;
    config.stop_threshold = 0;
    config.silence_threshold = 0;

    if (!sample_is_playable(card, device, channels, rate, bits, period_size, period_count)) {
        return;
    }

    pcm = pcm_open(card, device, PCM_OUT, &config);
    if (!pcm || !pcm_is_ready(pcm)) {
        fprintf(stderr, "Unable to open PCM device %u (%s)\n",
                device, pcm_get_error(pcm));
        return;
    }

    size = pcm_frames_to_bytes(pcm, pcm_get_buffer_size(pcm));
    buffer = malloc(size);
    if (!buffer) {
        fprintf(stderr, "Unable to allocate %d bytes\n", size);
        free(buffer);
        pcm_close(pcm);
        return;
    }

    printf("Playing sample: %u ch, %u hz, %u bit\n", channels, rate, bits);

    /* catch ctrl-c to shutdown cleanly */
    signal(SIGINT, stream_close);

    do {
        num_read = fread(buffer, 1, size, file);
        if (num_read > 0) {
            if (pcm_write(pcm, buffer, num_read)) {
                fprintf(stderr, "Error playing sample\n");
                break;
            }
        }
    } while (!close && num_read > 0);
    free(buffer);
    pcm_close(pcm);
}
```

4.7. Camera 模块(CIM)

X1000 camera 摄像头控制器为 cim, x1000 平台使用的接口为 dvp, 8 位数据并行传输, 硬件上用 hsync, vsync, pclk, 等控制信号控制 camera 数据的传输, cim 控制器支持输入格式为 yuv422, 输出格式支持 yuv422., 内核的 camera 控制器驱动 和 sensor 驱动是基于 v4l2 驱动架构编写的。应用程序可以使用 cimutils 进行测试。以下会对如何添加一款新 sensor 进行简要的说明。

4.7.1. 添加 sensor 流程

4.7.1.1. 硬件 (Hardware)

4.7.1.1.1. CIM 控制器

1. 首先请先了解下 CIM 控制器, 以确定是否能支持当前所需支持 sensor。

通过查看 X1000 的 PM 手册得知:

5.1.1 Features

- Input image size up to 2M pixels
- Integrated DMA
- Supported data format: YCbCr 4:2:2
- Supports ITU656 (YCbCr 4:2:2) input
- Configurable VSYNC and HSYNC signals: active high/low
- Configurable PCLK: active edge rising/falling
- 128x66 image data receive FIFO (RXFIFO)
- PCLK max. 80MHz
- Configurable output order

5.1.2 Pin Description

Table 5-1 Camera Interface Pins Description

Name	I/O	Width	Description
MCLK	O	1	CIM work clock
PCLK	I	1	Pixel clock from Image Sensor
VSYNC	I	1	Vertical synchronous from Image Sensor
HSYNC	I	1	Horizontal synchronous from Image Sensor
DATA	I	8	Data bus from Image Sensor

其中主要几个信息：

- PCLK 最大支持 80MHZ 输入。
- 图像采集尺寸最大 2M。
- 输入数据格式为 YCbCr。
- 控制器和 sensor 之间的连线为 12 根，其中数据线宽度为 8 根。

2. CIM 控制器的功能 PIN 设置

CIM 控制器的功能 pin 定义在 arch/mips/xburst/soc-x1000/include/mach/platform.h 下，一般不需要改动。

通过查看手册可知：CIM 控制器的 GPIO 组为 portA, function 为 2, 对应管脚为 PA08 到 PA19 共 12 个 pin。

Table 19-2 GPIO Port A summary

#PAD ID	PAD TYPE	AD PULL AC	PULL RST	ST RST	SL RST	DS RST	FUNCTION2
PA00	PRUW08DGZ G	0	1	-	-	-	smb1_sck(i-o-1)
PA01	PRUW08DGZ G	0	1	-	-	-	smb1_sda(i-o-1)
PA02	PRUW08DGZ G	0	1	-	-	-	uart2_rxd(i-1)
PA03	PRUW08DGZ G	0	1	-	-	-	uart2_txd(o)
PA04	PRUW08DGZ G	0	1	-	-	-	uart1_rxd(i-1)
PA05	PRUW08DGZ G	0	1	-	-	-	uart1_txd(o)
PA06	PRUW08DGZ G	0	1	-	-	-	(i-0)
PA07	PRUW08DGZ G	0	1	-	-	-	(i-0)
PA08	PRUW08DGZ G	0	1	-	-	-	cim_pclk i(i-0)
PA09	PRUW08DGZ G	0	1	-	-	-	cim_hsyn i(i-0)
PA10	PRUW08DGZ G	0	1	-	-	-	cim_vsyn i(i-0)
PA11	PRUW08DGZ G	0	1	-	-	-	cim_mclk o(o)
PA12	PRUW08DGZ G	0	1	-	-	-	cim_d7 i(i-0)
PA13	PRUW08DGZ G	0	1	-	-	-	cim_d6 i(i-0)
PA14	PRUW08DGZ G	0	1	-	-	-	cim_d5 i(i-0)
PA15	PRUW08DGZ G	0	1	-	-	-	cim_d4 i(i-0)
PA16	PRUW08DGZ G	0	1	-	-	-	cim_d3 i(i-0)
PA17	PRUW08DGZ G	0	1	-	-	-	cim_d2 i(i-0)
PA18	PRUW08DGZ G	0	1	-	-	-	cim_d1 i(i-0)
PA19	PRUW08DGZ G	0	1	-	-	-	cim_d0 i(i-0)

19.5.1 Port Function Guide

INT	MASK	PAT1	PAT0	Port Description
1	0	0	0	Port is low level triggered interrupt input.
1	0	0	1	Port is high level triggered interrupt input.
1	0	1	0	Port is fall edge triggered interrupt input.
1	0	1	1	Port is rise edge triggered interrupt input.
1	1	0	0	Port is low level triggered interrupt input. Interrupt is masked. Flag is recorded.
1	1	0	1	Port is high level triggered interrupt input. Interrupt is masked. Flag is recorded.
1	1	1	0	Port is fall edge triggered interrupt input. Interrupt is masked. Flag is recorded.
1	1	1	1	Port is rise edge triggered interrupt input. Interrupt is masked. Flag is recorded.
0	0	0	0	Port is pin of device 0.
0	0	0	1	Port is pin of device 1.
0	0	1	0	Port is pin of device 2.
0	0	1	1	Port is pin of device 3.
0	1	0	0	Port is GPIO output 0.
0	1	0	1	Port is GPIO output 1.
0	1	1	?	Port is GPIO input.

可以确认设置 GPIO 的四个寄存器值为 0 0 1 0，寄存器地址可以在下图中得到

▼ Registers Description	426
▼ PORT A Register Group	429
PORT A PIN Level Registers (PAPINL,0x00)	429
PORT A Interrupt Registers (PAINT,0x10)	430
PORT A Interrupt Set Registers (PAINTS,0x14)	430
PORT A Interrupt Clear Registers (PAINTC,0x18)	430
PORT A Mask Registers (PAMSK,0x20)	431
PORT A Mask Set Registers (PAMSKS,0x24)	431
PORT A Mask Clear Registers (PAMSKC,0x28)	432
PORT A PAT1/Direction Registers (PAPAT1,0x30)	432
PORT A PAT1/Direction Set Registers (PAPAT1S,0x34)	433
PORT A PAT1/Direction Clear Registers (PAPAT1C,0x38)	433
PORT A PAT0/Data Registers (PAPAT0,0x40)	433
PORT A PAT0/Data Set Registers (PAPAT0S,0x44)	434
PORT A PAT0/Data Clear Registers (PAPAT0C,0x48)	434
PORT A FLAG Registers (PAFLG,0x50)	435
PORT A FLAG Clear Registers (PAFLGC,0x58)	435
PORT A PULL Disable Registers (PAPEN,0x70)	436
PORT A PULL Set Registers (PAPENS,0x74)	436
PORT A PULL Clear Registers (PAPENC,0x78)	436

Table 19-6 Registers Memory Map-Address Base

Name	Base	Description
PA	0x10010000	Address base of GPIO Port A
PB	0x10010100	Address base of GPIO Port B
PC	0x10010200	Address base of GPIO Port C
PD	0x10010300	Address base of GPIO Port D
PZ	0x10010700	Address base of GPIO Group Z (Shadow group)

4.7.1.1.2. 添加 sensor

以 halley2 标配的 HZGP2697 为例

通过查看 HZGP2697-CS-Model.pdf, 可以得到以下关键信息:

sensor 型号	gc2155	
管脚信息	SDA	连接到 X1000 的一组 I2C 通道上 (I2C0, I2C1, I2C2)
	SCL	
	PWDN	sensor 的电源管理 PIN
	RESET	复位
		以下为和 CIM 控制器对接的 PIN 脚
	HSYNC (HREF)	行同步信号
	VSYNC	帧同步信号
	MCLK	sensor 参考时钟输入
	PCLK	像素时钟
	D0 - D7	数据 pin

再根据 sensor 手册《GC2155 COB DataSheet release V1.0_20131128.pdf》, 可以得到以下关键信息:

- 上下电时序
- clk data 时序
- MCLK 频率范围
- HSYNC 默认有效极性
- VSYNC 默认有效极性
- PCLK 默认采样沿

一般情况下, 我们不需要从零开始配置 sensor, 厂家的 FAE 会给一个配置, 对于此配置厂家会告知其输出特性, 最基本的有:

- 分辨率
- 帧率

*有些 sensor 的管脚极性是可以配置的, 在厂家提供的配置时, 可以咨询厂家当前配置是否有极性调整。

Note: 为确保后续编写驱动顺利进行, 请务必确认好以上信息。

4.7.1.2.1. 代码路径及描述

sensor 相关的代码分布在以下目录及文件中，以 halley2 为例：

```
arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/cim_board.c
arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/i2c_bus.c
arch/mips/xburst/soc-x1000/chip-x1000/halley2/halley2_v10/board.h
arch/mips/xburst/soc-x1000/include/mach/platform.h
arch/mips/xburst/soc-x1000/common/platform.c
drivers/media/i2c/soc_camera/          /*sensor 设备驱动目录，如 gc2155.c*/
/*X1000 CIM 控制器驱动,基于 V4L2 架构，并采用 videobuf2。便于上层开发。*/
drivers/media/platform/soc_camera/jz_camera_v13.c
drivers/media/platform/soc_camera/soc_camera.c    /* soc_camera 子系统*/
```

以下为添加 sensor 必改的文件，请同时参考代码一起熟悉：

[arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/cim_board.c](#)

主要用于注册 sensor 设备信息，和 CIM 控制器设备信息。

其中主要数据结构为：

1) static **struct soc_camera_link** iclink_front

.bus_id	CIM 控制器 ID，由于 X1000 只集成了一个 CIM 控制器，所以固定为 0。
.i2c_adapter_id	sensor 所连接 I2C 控制器的通道号。
.board_info sensor	sensor 名称及 i2c 设备地址信息，代码位置在： arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/i2c_bus.c
.power	上电接口。
.reset	复位接口。
	gc2155 的上电和复位涉及到了 3 个宏： CAMERA_VDD_EN CAMERA_SENSOR_RESET CAMERA_FRONT_SENSOR_PWDN 这些需要按照实际的硬件连接来设置，代码位置在： arch/mips/xburst/soc-x1000/chip-x1000/halley2/halley2_v10/board.h

2) struct **jz_camera_pdata** camera_pdata

.mclk_10khz	指定输出的 MCLK，单位为 10khz。
其他参数没有使用	

Sensor 设备驱动:

[drivers/media/i2c/soc_camera/gc2155.c](#)

主要的数据结构为:

1) static struct **v4l2_subdev_core_ops** gc2155_subdev_core_ops

.s_power	负责 sensor 初始化, 主要工作是 sensor 上电, 复位, 通过 i2c 将 gc2155_init_regs 配置给 sensor。gc2155_init_regs 为 gc2155 厂家提供的初始化寄存器表。
----------	--

2) static struct **v4l2_subdev_video_ops** gc2155_subdev_video_ops

.s_mbus_fmt	在启动启动 sensor 前, 上层会通过 v4l2 接口配置所需分辨率信息到 sensor, 最终也是通过 i2c 将 gc2155_vga_reg、sgc2155_qvga_regs、gc2155_720P_regs 配置给 sensor。												
.g_mbus_config	<p>非常关键</p> <p>在 gc2155_g_mbus_config 中, 会设置 sensor 的 3 个管脚属性给控制器, 就是我们从 gc2155 手册中得到的信息</p> <p>cfg->flags =</p> <p>V4L2_MBUS_PCLK_SAMPLE_RISING // 上升沿采样</p> <p>V4L2_MBUS_VSYNC_ACTIVE_HIGH // vsync 有效极性为高电平</p> <p>V4L2_MBUS_HSYNC_ACTIVE_HIGH // hsync 有效极性为高电平</p> <p>V4L2_MBUS_MASTER //固定标识, 必须有</p> <p>V4L2_MBUS_DATA_ACTIVE_HIGH; //固定标识, 必须有</p> <p>这些宏定义在/include/media/v4l2-mediabus.h 中</p> <table border="1"> <tr> <td>V4L2_MBUS_PCLK_SAMPLE_RISING</td><td>CIM 控制器在 pclk 上升沿处采样。</td></tr> <tr> <td>V4L2_MBUS_PCLK_SAMPLE_FALLING</td><td>CIM 控制器在 pclk 下降沿处采样。</td></tr> <tr> <td>V4L2_MBUS_VSYNC_ACTIVE_HIGH</td><td>CIM 控制器在 vsync 翻转为高电平时开始启动新的一帧采集。</td></tr> <tr> <td>V4L2_MBUS_VSYNC_ACTIVE_LOW</td><td>CIM 控制器在 vsync 翻转为低电平时开始启动新的一帧采集。</td></tr> <tr> <td>V4L2_MBUS_HSYNC_ACTIVE_HIGH</td><td>CIM 控制器在 hsync 为高电平时, 采集期间每一个有效的 PCLK。</td></tr> <tr> <td>V4L2_MBUS_HSYNC_ACTIVE_LOW</td><td>CIM 控制器在 hsync 为低电平时, 采集期间每一个有效的 PCLK。</td></tr> </table>	V4L2_MBUS_PCLK_SAMPLE_RISING	CIM 控制器在 pclk 上升沿处采样。	V4L2_MBUS_PCLK_SAMPLE_FALLING	CIM 控制器在 pclk 下降沿处采样。	V4L2_MBUS_VSYNC_ACTIVE_HIGH	CIM 控制器在 vsync 翻转为高电平时开始启动新的一帧采集。	V4L2_MBUS_VSYNC_ACTIVE_LOW	CIM 控制器在 vsync 翻转为低电平时开始启动新的一帧采集。	V4L2_MBUS_HSYNC_ACTIVE_HIGH	CIM 控制器在 hsync 为高电平时, 采集期间每一个有效的 PCLK。	V4L2_MBUS_HSYNC_ACTIVE_LOW	CIM 控制器在 hsync 为低电平时, 采集期间每一个有效的 PCLK。
V4L2_MBUS_PCLK_SAMPLE_RISING	CIM 控制器在 pclk 上升沿处采样。												
V4L2_MBUS_PCLK_SAMPLE_FALLING	CIM 控制器在 pclk 下降沿处采样。												
V4L2_MBUS_VSYNC_ACTIVE_HIGH	CIM 控制器在 vsync 翻转为高电平时开始启动新的一帧采集。												
V4L2_MBUS_VSYNC_ACTIVE_LOW	CIM 控制器在 vsync 翻转为低电平时开始启动新的一帧采集。												
V4L2_MBUS_HSYNC_ACTIVE_HIGH	CIM 控制器在 hsync 为高电平时, 采集期间每一个有效的 PCLK。												
V4L2_MBUS_HSYNC_ACTIVE_LOW	CIM 控制器在 hsync 为低电平时, 采集期间每一个有效的 PCLK。												

驱动中使用的两个 i2c 读写接口:

i2c_smbus_write_byte_data

i2c_smbus_read_byte_data

4.7.1.2.2. 编译配置

一般发布的软件版本中会默认已经配置了 camera 驱动，可以基于已有配置进行熟悉更改，如 arch/mips/configs/halley2_nor_v10_linux_defconfig

在工作机终端下执行：

```
$ make halley2_nor_v10_linux_defconfig; make menuconfig
```

kernel 需配置以下选项：

Symbol: VIDEO_JZ_CIM_HOST_V13 [=y] #选择 x1000 camera 的控制器驱动

Type : tristate

Prompt: ingenic cim driver used on camera x1000

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

-> SoC camera support (SOC_CAMERA [=y])

Symbol: SOC_CAMERA_GC2155 [=y]

#选择具体 camera 型号

Type : tristate

Prompt: gc2155 camera support

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> Sensors used on soc_camera driver

Symbol: JZ_IMEM [=y]

#为 camera use_ptr 模式提供 vpu 编码所需要的 memory

Type : boolean

Prompt: jz imem

Location:

-> Machine selection

-> SOC type (SOC_TYPE [=y])

CONFIG_JZ_VPU:

#使用 vpu（硬件编码）作为编码的方式

Symbol: VIDEO_INGENIC_VPU_FOR_V4L2 [=y]

Type : tristate

Prompt: V4L2 driver for Ingenic VPU

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

Symbol: VIDEO_INGENIC_X1000_JPEG [=y]

Type : tristate

Prompt: X1000 JPEG codec driver

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

-> V4L2 driver for Ingenic VPU (VIDEO_INGENIC_VPU_FOR_V4L2 [=y])

Symbol: I2C0_V12_JZ [=y] #选择 i2c 控制器，根据实际 sensor 所连接的 I2C 通道进行选择

Type : tristate

Prompt: JZ_v12 i2c controler 0 Interface support

Location:

-> Device Drivers

-> I2C support (I2C [=y])

-> I2C Hardware Bus support

Symbol: I2C0_SPEED [=100]

#设置 i2c 控制器为 100KHz

Type : integer

Range : [100 400]

Prompt: Jz_v12 i2c0 speed in KHZ

Location:

-> Device Drivers

-> I2C support (I2C [=y])

-> I2C Hardware Bus support

4.7.2. 调试方法

4.7.2.1. 调试命令

当代码编辑完毕并编译无误后，开始进行调试。

在我们的工程下提供了 camera 例程 `cimutils`，其操作 camera 使用的是标准的 V4L2 接口。

`cimutils` 可以拍照和预览，拍照可保存成为 jpeg, bmp, 和 raw 格式的文件，预览可以把 camera 获取到的数据显示到显示屏上。`cimutils` 应用可以动态设置拍照的大小和格式以及保存的格式和名字，目前 `cimutils` 支持 user 和 mmap 两种模式。用户也可以自行编写例程。

例如：

user_ptr:

```
# ./cimutils -I 1 -C -x 640 -y 480 -f RTF_ingenic_userptr.jpg -v    #拍照命令
# ./cimutils -I 1 -P -x 320 -y 240 -v    #预览命令
```

mmap:

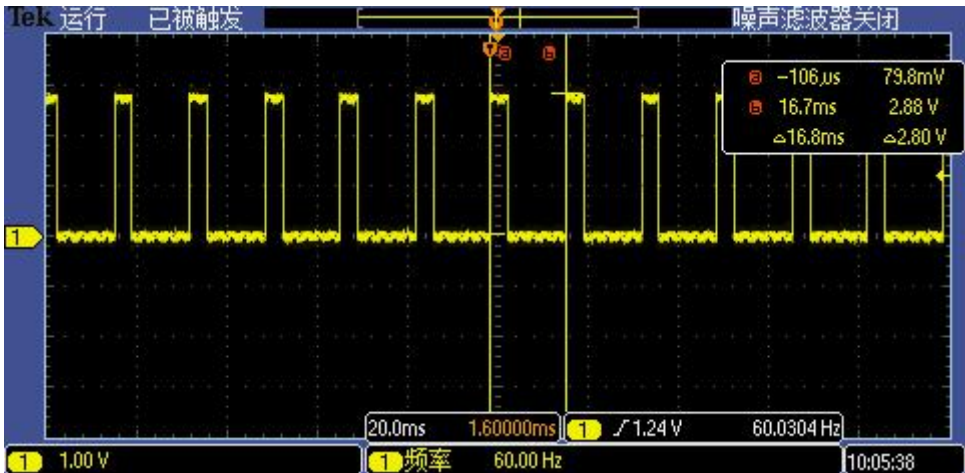
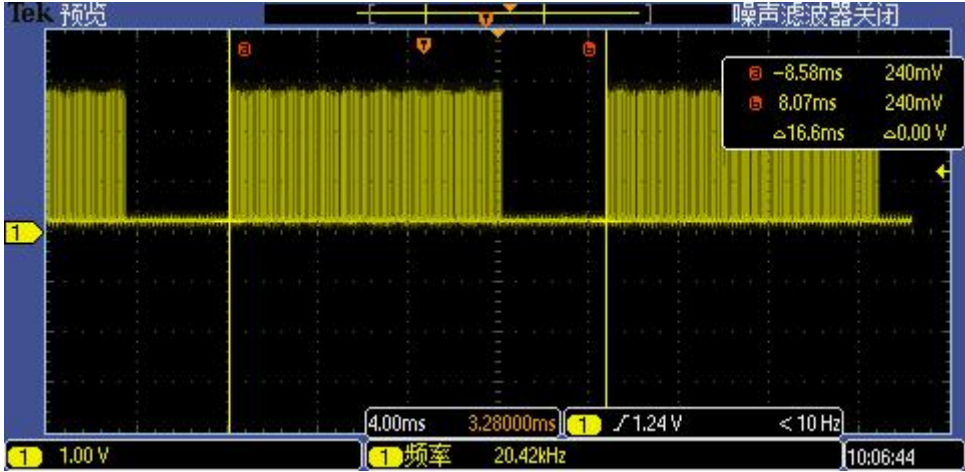
```
# ./cimutils -I 1 -C -x 640 -y 480 -f RTF_ingenic_mmap.jpg    #拍照命令
# ./cimutils -I 1 -P -x 320 -y 240    #预览命令
```

首先在终端下执行：

```
# cimutils -I 1 -P -x 320 -y 240
```

此命令会启动 sensor 并在 halley2 板载屏幕进行预览。若无问题，屏幕会有正确的 sensor 图像显示。

4.7.2.2. 常见问题及解决方法

<p>I2C 报 NO ACK</p> <p>终端下无异常， 但没有图像输出</p>	<p>检查 sensor 是否已正常上电，是否已处于 I2C 可以正常通信状态。</p> <p>首先确保 sensor 能够正常输出，再确认 CIM 控制器是否正常工作。</p> <p>利用示波器对 sensor 的各路信号进行测量，通过抓取波形，确认如下信息是否正确：</p> <p>mclk 频率</p> <p>vsync 有效极性 帧率</p> <p>hsync 有效极性，通过 hsync 估算行数</p> <p>pclk 采样沿，频率（小于 80M），通过 pclk 估算行宽</p>  <p>图 1 vsync 可以看出当前的帧率为 60 帧。</p>
	 <p>图 2 hsync 大致可以估算一帧内 hsync 占用时间为 12ms</p>

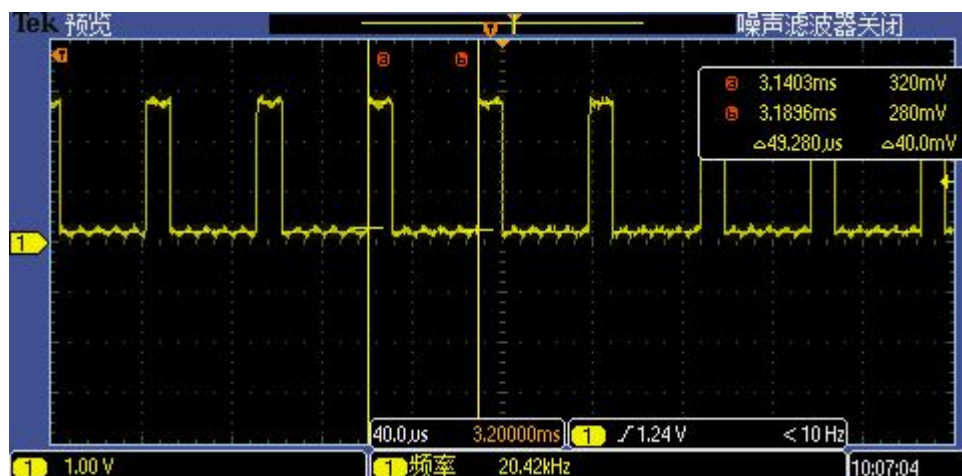


图 3 一个 hsync 大约耗时 49us，可以估算一帧内行数为 $12\text{ms}/49\mu\text{s} \approx 240$ 行

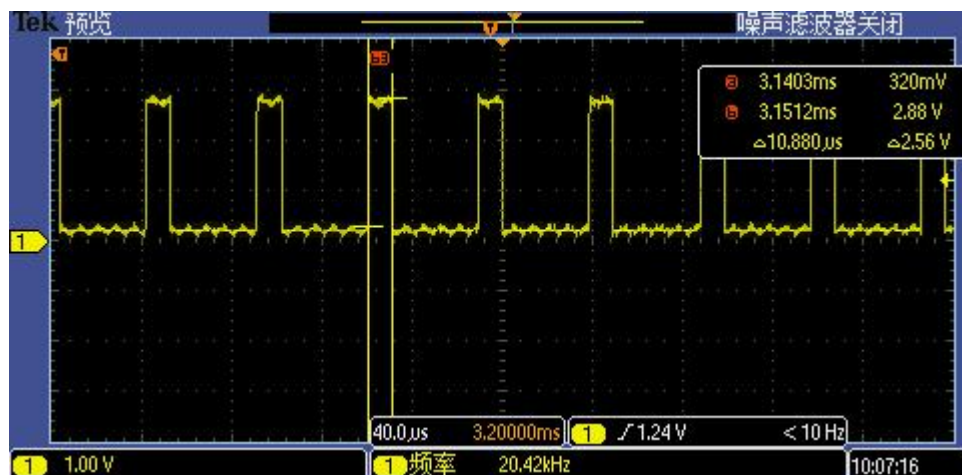


图 4 hsync 一个行信号中有效时间大约 11us

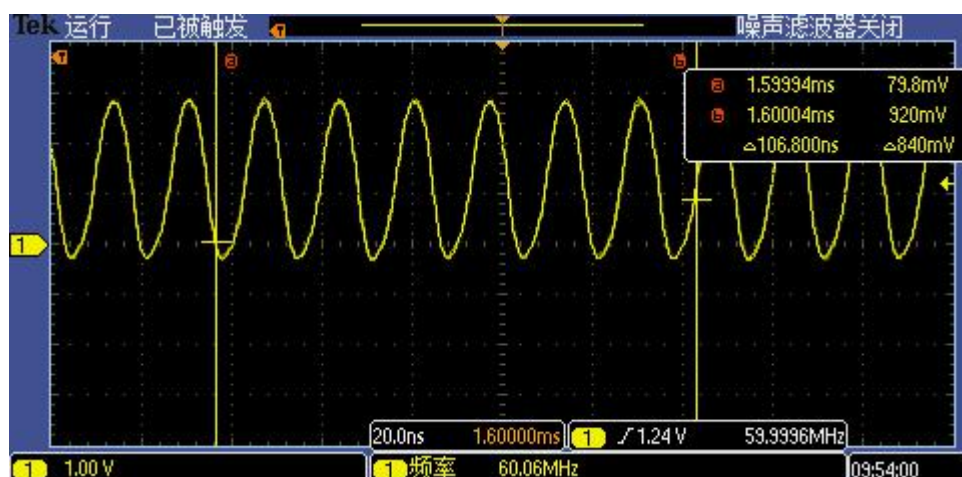


图 5 pclk，可以计算出一行的有效 pixel 数为 $11\mu\text{s}/60\text{MHz} \approx 640\text{pclk} = 320\text{pixel}$ (两个 pclk 为一个像素)

由此可以确认是否 sensor 输出正常，如果不是预期的尺寸，那可以跟厂家确认其

	<p>提供的配置是否正确,也可以把每一条配置在从 sensor 中读回来以确认是否正确。</p>
	<p>如果 sensor 输出正常,那么接下来可以分析 CIM 控制器是否配置正确 输出 CIM 控制器的寄存器值</p> <p>如下为 GC2155 输出正常时的寄存器值,可以参考</p> <pre> REG_CIM_CFG = 0x00001c42 REG_CIM_CTRL = 0x00000085 REG_CIM_CTRL2 = 0x00c00015 REG_CIM_STATE = 0x00000002 REG_CIM_IMR = 0x0000009c REG_CIM_IID = 0x00000000 REG_CIM_DA = 0x00c8c020 REG_CIM_FA = 0x00880000 REG_CIM_FID = 0x00000000 REG_CIM_CMD = 0x48009600 //REG_CIM_CMD 中, LEN 为 WORD 单位, 0x9600 = 320*240*2/WORD REG_CIM_WSIZE = 0x00000000 REG_CIM_WOFFSET = 0x00000000 REG_CIM_FS = 0x00ef413f //REG_CIM_FS 中, DMA 尺寸为 240x320, 每像素 2 字节 </pre>
应用程序 select time out	<p>可以尝试关闭 FSC 校验位来调试,在 FAQ 中有介绍。</p>

4.7.3. FAQ

1. 如何查看 CIM 控制器寄存器内容？

`cat /sys/devices/platform/jz-cim.0/debug/dump_cim_reg` 可以打印当前控制器的寄存器, 要求控制器必须处于工作状态, 否则控制器的时钟会关闭, 从而读不到值。可以后台执行 `cimutils`。

例: `cimutils -I 1 -P -x 320 -y 240 &`

2. FSC 是什么？

帧尺寸校验, 此功能默认开启, 目的是由于 sensor 在和 CIM 控制器通信过程中, 如果由于干扰或者线上噪声过大, 导致一帧数据有丢失的话, 那么此帧数据会被控制器主动抛弃, 此种情况下保证应用层能够接收到完整图像。此功能建议打开。

在调试过程中, 可能 sensor 输出尺寸和预期不符, 此时如果 FSC 校验位打开, 那么控制器一帧数据都无法成功接收, 此时, 在应用层来看, 就是会发生 `select time out`。可以, 暂时将此校验位关闭, 应用层将可以捕获图像, 然后在根据图像内容来分析, 这是一种调试方法, 不过还是测量所有 sync 脚的波形更方便些。

帧校验是可以产生中断的, 可以通过中断来确认是否有丢帧事件的发生。以下为 FSC 中断补丁:

```
diff --git a/drivers/media/platform/soc_camera/jz_camera_v13.c b/drivers/media/platform/soc_camera/
jz_camera_v13.c
index b4217a2..469e1a8 100644
--- a/drivers/media/platform/soc_camera/jz_camera_v13.c
+++ b/drivers/media/platform/soc_camera/jz_camera_v13.c
@@ -628,6 +628,11 @@ static int jz_camera_set_bus_param(struct soc_camera_device *icd) {
    writel(ctrl2_reg, pcdev->base + CIM_CTRL2);
    writel(fs_reg, pcdev->base + CIM_FS);

+    /* enable fsc interrupt */
+    temp = readl(pcdev->base + CIM_IMR);
+    temp &= (~CIM_IMR_FSEM);
+    writel(temp, pcdev->base + CIM_IMR);
+
    /* enable end of frame interrupt */
    temp = readl(pcdev->base + CIM_IMR);
    temp &= (~CIM_IMR_EOFM);
@@ -795,6 +800,18 @@ static irqreturn_t jz_camera_irq_handler(int irq, void *data) {
    pcdev->dma_stopped = 1;
}

+    if(status & CIM_STATE_SIZE_ERR) {
+        /* clear dma interrupt status */
+        temp = readl(pcdev->base + CIM_STATE);
```

```
+         temp &= (~CIM_STATE_SIZE_ERR);  
+         writel(temp, pcdev->base + CIM_STATE);  
+         dev_err(pcdev->dev, "Frame size check error!\n");  
+     }  
+  
+     if(status & CIM_STATE_DMA_EOF) {  
+         /* clear dma interrupt status */  
+         temp = readl(pcdev->base + CIM_STATE);
```

3.如何测量当前的帧率？

为了方便调试，CIM 控制器驱动里提供了两个测试功能，可以根据需求进行开启：

drivers/media/platform/soc_camera/jz_camera_v13.c

#define CIM_DEBUG_FPS //显示当前帧率

#define PRINT_CIM_REG //启动控制器前，输出当前寄存器值。

测试用例 cimutils，也提供了 debug 功能，可以 cimutils --help 查看，其中包括帧率测试功能，在命令中加入参数 -d 即可。

另外在网络上有很多 sensor 调试文档，可供参考。

4.8. SD 卡模块

4.8.1. 内核空间

1. 板级配置文件:

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/mmc.c

其中:

struct jzmmc_platform_data tf_pdata 描述 sd 卡设备。

struct jzmmc_platform_data sdio_pdata 描述 sdio 类设备, 一般指 sdio-wifi。

2. 电路连接:

sd 卡接到 MSC0, sdio-wifi 接到 MSC1。

1. 驱动程序文件:

drivers/mmc/host/jzmmc_v12.c

驱动配置:

Symbol: JZMMC_V12 [=y]

Type : tristate

Prompt: Ingenic(XBurst) MMC/SD Card Controller(MSC) v1.2 support

Location:

-> Device Drivers

-> MMC/SD/SDIO card support (MMC [=y])

Defined at drivers/mmc/host/Kconfig:7

Depends on: MMC [=y] && (SOC_M200 [=n] || SOC_X1000 [=y])

4.8.2. 用户空间

MSC 支持 SD 设备, SDIO 设备, MMC 设备, 以 SD 卡设备为例。SD 卡作为一个标准的块设备驱动程序, 在 dev/ 下会产生/dev/mmcblk*p*类似的设备节点。使用 mount 命令能够将分区 mount 到目录

例如:

```
$ mount -t ext4 /dev/mmcblk0p7 /mnt
```

如果 mount 失败, 一般是由于内核没有支持 ext4 文件系统选项, 或者分区不是 ext4 文件系统类型。

4.9. LCD 模块

X1000 SLCD Controller 支持 16/18/24 bit 6800/8080 并行接口, 9 bit 6800/8080 2 次并行接口, 8 bit 6800/8080 1/2/3 次并行接口, 8/16/18/24 bit 串行接口。液晶屏

4.9.1. 内核空间

4.9.1.1. 文件介绍

内核驱动基于 framebuffer 驱动架构, 所有的硬件资源在板级定义。按照 framebuffer 驱动编写规范, 实现 LCD 驱动需要提供 struct fb_info, 以及用作显示的 platform_driver 以下介绍驱动对应在内核中的路径:

板级资源定义路径:

halley2 开发板的板级:

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/lcd/

phoenix 开发板的板级:

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/lcd/

与 X1000 LCD 相关的驱动所在目录和文件说明, 忽略目录中存在的其他文件。

driver/video/jz_fb_v13/

└─ regs.h #包含 lcdc 的所有寄存器。

└─ jz_fb.h #定义了 struct jzfb, 包含驱动过程中的所有参数, 放到 fb_info->par; 定义了 ioctl 以及 ioctl 相关的结构体。

└─ jz_fb.c #驱动实现的文件。

4.9.1.2. 编译配置

一般发布的软件版本中会默认配置 LCD 驱动, 如果需要自己更改 LCD 屏幕, 可以通过以下方式进行配置。

在工作电脑上执行:

```
$ make menuconfig
```

配置以下选项:

以 slcd truly240240 为例

配置

CONFIG_LCD_TRULY_TFT240240_2_E

Location:

-> Device Drivers

-> Graphics support

-> Backlight & LCD device support (BACKLIGHT_LCD_SUPPORT [=y])
-> Lowlevel LCD controls (LCD_CLASS_DEVICE [=y])

如需添加新的 LCD 屏幕，需要在板级添加相关文件和配置，以 SLCD truly240240 为例：

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/lcd/

添加 lcd-truly_tft240240_2_e.c 文件，填写屏幕参数；

Makefile 中添加：

```
obj-$(CONFIG_LCD_TRULY_TFT240240_2_E) += lcd-truly_tft240240_2_e.o
```

driver/video/backlight/

添加 truly_tft240240_2_e.c 文件，配置 gpio、电源配置等；

Kconfig 中添加：

```
config LCD_TRULY_TFT240240_2_E
```

```
    tristate "SLCD TRULY TFT240240-2-E with control IC st7789s (240x240)"
```

```
    depends on BACKLIGHT_CLASS_DEVICE
```

```
    default n
```

4.9.2. 用户空间

调试 LCD 的过程中，可在用户空间查看 LCD 的状态信息：

```
# cd /sys/devices/platform/jz-fb/graphics/fb0/
```

```
# ls
```

bits_per_pixel	device	power	uevent
blank	mode	rotate	virtual_size
console	modes	state	
cursor	name	stride	
dev	pan	subsystem	

查看当前分辨率及帧率信息：

```
# cat mode
```

```
U:240x240p-60
```

当前分辨率 240*240 帧率为 60fps。

查看位宽：

```
# cat bits_per_pixel
```

```
16
```

当前位宽为 16bit。

4.10. MiniGUI demo 的使用

MiniGUI 是一款面向嵌入式系统的高级窗口系统（Windowing System）和图形用户界面（Graphical User Interface, GUI）支持系统，目前 Manhatton 工程中移植了 MiniGUI 的部分功能用于展示 x1000 的图形界面显示功能。

相关代码路径如下：

```
packages/example/Sample/minigui
external/minigui
```

4.10.1. 驱动配置及编译

（1）kernel 配置选项：

There is no help available for this option.

Symbol: SLCDC_CONTINUA [=y]

Type : tristate

Prompt: SLCDC CONTINUA TRANFER

Location:

-> Device Drivers

-> Graphics support

-> Backlight & LCD device support (BACKLIGHT_LCD_SUPPORT [=y])

-> Lowlevel LCD controls (LCD_CLASS_DEVICE [=y])

默认代码此选项是选中状态，无需手动选择。

（2）整体编译：

文件路径：

device/halley2/halley2_base.mk

在此文件中添加如下选项：

```
18 #the device applications & test demo
19 PRODUCT_MODULES +=$(RUNTESTDEMO_UTILS)\
20                 $(CAMERA_UTILS)\
21                 $(ALSA_UTILS)\
22                 minigui demo
23 #
24                 $(TINYALSA_UTILS)
```

保存并退出，整体编译 manhatton：

```
$ cd <Your_Manhatton_Project>
```

```
$ source build/envsetup.sh
```

```
$ lunch
```

<以 halley2 nor flash 为例，选中：halley2_norflash-eng>

```
$ make
```

编译完成会将 minigui_demo 的相关代码（包括其依赖的库）编译到文件系统中，生成文件路径：`out/product/halley2/image/system.jffs2`

加入 minigui_demo 整体编译生成的文件系统为 14MB 左右，若是大容量的存储介质无需关心其 size 变大的影响，但若是小容量的 flash,如 halley2 开发板默认为 16MB 的 Nor flash,需要裁减文件系统的其他功能来给文件系统瘦身在 12MB 以下方能正常烧录启动，也可不裁减任何功能直接网络挂载文件系统进行 minigui_demo 的操作（关于网络挂载文件系统的详细步骤请参考文档《nfs 的搭建.pdf》，本章节不再赘述）。

4.10.2. 运行图形界面

关于 minigui_demo 的使用，下面以网络挂载文件系统的方式且系统已成功启动开始说明。
网络文件系统启动后，进入 minigui 的 demo 目录,并运行 minigui_demo:

```
# cd testsuite/minigui_demo
# ./minigui_demo
```

命令运行成功后在 lcd 小板上会有“hello world”的图形界面显示。

4.11. VPU 模块

Vpu 是视屏处理单元，在 x1000 中用到了 vpu 中的 jpeg 模块，JPEG 模块是一个 JPEG 编码单元，用来将 yuv 数据转换成 jpeg 格式。目前 jpeg 支持输入数据格式为 yuv422，输出格式 jpeg，目前支持 3 个级别 jpeg 的格式输出，分别是低，中，高。

4.11.1. 内核空间

板级资源定义路径：

arch/mips/xburst/soc-x1000/common/platform.c 申请资源

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/board_base.c 注册资源

与 X1000 vpu 相关的驱动所在目录和文件说明，忽略目录中存在的其他文件。

drivers/media/platform/jz_vpu/x1000

在发布版本中默认已经添加 vpu 驱动，如果想要自己改变 vpu 的编译配置可以通过以下进行配置：

CONFIG_JZ_VPU:

Symbol: VIDEO_INGENIC_VPU_FOR_V4L2 [=y]

Type : tristate

Prompt: V4L2 driver for Ingenic VPU

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

Symbol: VIDEO_INGENIC_X1000_JPEG [=y]

Type : tristate

Prompt: X1000 JPEG codec driver

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

-> V4L2 driver for Ingenic VPU (VIDEO_INGENIC_VPU_FOR_V4L2 [=y])

4.11.2. 用户空间

在用户空间，vpu 主要功能是把摄像头或者其他方式输入的 yuv422 格式的数据进行编码，转化为 jpeg 格式。在 development/source/jpg_api/jpeg_encode2.c 中提供相应的接口：

1.jpeg 初始化函数，用来初始化 vpu:

```
void* jz_jpeg_encode_init(int width,int height)
```

2. 反初始化:

```
void jz_jpeg_encode_deinit(void *handle)
```

3. 将 yuv422 格式的数据转换为 jpeg 格式的数据，并保存为 xxx.jpg 文件:

```
int yuv422_to_jpeg(void *handle,unsigned char *input_image, FILE *fp, int width, int height,  
int quality)
```

4.12. USB 模块

USB，是英文 Universal Serial Bus（通用串行总线）的缩写，而其中文简称为“通串线”，是一个外部总线标准，用于规范电脑与外部设备的连接和通讯。是应用在 PC 领域的接口技术。USB 接口支持设备的即插即用和热插拔功能。USB 是在 1994 年底由英特尔、康柏、IBM、Microsoft 等多家公司联合提出的。

4.12.1. 内核空间

资源相关文件路径：

- A) 板级资源定义路径:arch/mips/xburst/soc-x1000/common/platform.c
- B) 平台设备注册文件路径：
arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/ board_base.c
- C) GPIO 申请文件路径：

在 “ arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/misc.c ” 中 申 请 userusb 的 de_te_pin,id_pin,bus_pin

usb 控制器驱动的目录如下：

```
drivers/usb/dwc2/
|——dwc2
|——dwc2_jz.c      #usb 控制器实现
|——core.c         #usb core 层在控制器中的具体实现
|——gadget.c       #usb 一些工具在控制器的具体实现
|——rh.c
|——debugfs.c      #debugfs 虚拟文件系统的实现
|——ep0.c
|——host-ddma.c
```

在发布版本中会默认添加 usb 驱动，usb 分为 host 端和 device 端，如果想要自己改变 usb 的编译配置可以通过以下进行配置：

4.12.2. usb-host

usb-host 端又可以分为 usb-mass_storage 和 usb-camera，下面会议此介绍：

4.12.2.1. usb-mass_storage:

在电脑端输入 make menuconfig，然后配置：

```
Device Drives
->USB support
->DesignWare USB2 DRD Core Support
->Driver Mode
->Both host and device.
```

同时在本级目录下选上 USB Mass Storage support，然后配置：

```
->DeviceDrivers
```

-> SCSI device support

->SCSI device support

现在 usb-host-mass_storage 功能就配置成功。当有 u 盘插入时会有如下提示：

```
[ 22.092290] jz-dwc2 jz-dwc2: set vbus on(on) for host mode
[ 22.202174] USB connect
[ 22.902209] usb 1-1: new high speed USB device number 2 using dwc2
[ 23.328262] usb 1-1: New USB device found, idVendor=0930, idProduct=6545
[ 23.342090] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 23.356745] usb 1-1: Product: DataTraveler 2.0
[ 23.365858] usb 1-1: Manufacturer: Kingston
[ 23.374570] usb 1-1: SerialNumber: C86000BDBA09EF60CA285106
[ 23.412410] scsi0 : usb-storage 1-1:1.0
[ 24.475824] scsi 0:0:0:0: Direct-Access Kingston DataTraveler 2.0 PMAP PQ: 0 ANSI: 4
[ 25.751099] sd 0:0:0:0: [sda] 30497664 512-byte logical blocks: (15.6 GB/14.5 GiB)
[ 25.768695] sd 0:0:0:0: [sda] Write Protect is off
[ 25.779135] sd 0:0:0:0: [sda] No Caching mode page present
[ 25.790501] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 25.807171] sd 0:0:0:0: [sda] No Caching mode page present
[ 25.832513] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 25.879083] sda:sda1
[ 25.895075] sd 0:0:0:0: [sda] No Caching mode page present
[ 25.932372] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 25.964595] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

4.12.2.2. USB Camera

4.12.2.2.1. 驱动配置

再上述的基础上，配置：

-> Device Drivers

-> Multimedia support

-> Media USB Adapters

-> USB Video Class

就是可以使用 USB Camera。

4.12.2.2.2. grab 测试

grab 应用代码：packages/example/App/grab

将应用编译进文件系统，具体方式不再赘述，详细过程请参见《Manhattan_platform 编译系统使用指南》，整体编译后 grab 应用路径：/usr/bin/grab。

打开串口，开发板上电启动之后，执行以下命令使用 USB Camera 功能：

```
#grab -w 320 -h 240 -c 10 -r 5 -y
```

注：在当前目录下执行以下命令可查看具体参数说明：

```
#grab --help
```

串口打印如下则 USB Camera 照相成功：

```
width = 320, height = 240
input_yuv = 1
grab version 0.1.4
video /dev/video0
coc --1
{ pixelformat = 'YUYV', description = 'YUV 4:2:2 (YUYV)' }
coc --2
ddddddddddddCurrent data format information:
        width:320
        height:240
running.....
forever = 0,fcount=10
oooooooooooo 120 frame time = 32 429496733696ns/frame
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
yuv
savejpeg.c:get_pictureYUYV:293
[ 25.541770] dwc2 dwc2: Unlink after no-IRQ? Controller is probably using the wrong IRQ.
close fd 3
```

照相成功后会在当前路径下产生“p-x.jpg”照片文件，其中“x”为“0~9”。可在 pc 端输入以下命令查看图片：

下面以查看“p-0.jpg”为例：

```
$adb pull /p-0.jpg .
```

命令执行成功后即可在本地查看图片“p-0.jpg”是否正确。

4.12.3. usb-device

目前 gadget 功能使用 android gadget , android gadget 目前支持 mass_storge, adb, rndis 功能。

Device Drivers

->USB support

->USB Gadget support

->USB Gadget Drivers (Android Gadget)

下面分别说明上述三个功能如何开启:

4.12.3.1. adb & mass_storage

在文件系统中输入下面命令:

```
# cd /sys/class/android_usb/android0
# echo 0 > enable
# echo 18d1 > idVendor
# echo d002 > idProduct
# echo mass_storage,adb > functions
# echo 1 > enable
# /sbin/adbd &
```

现在 usb 支持了 adb 和 mass_storage 功能。具体验证方法:

(1)验证 adb:

在电脑端输入 adb shell 出现/sys/devices/virtual/android_usb/android0 # 说明 adb 开启成功。

(2)验证 mass_storage:

PC 端:

```
$ dd if=/dev/zero of=fat32.img bs=1k count=2048 #制作一个大小为 2M 的空文件
```

```
$ mkfs.vfat fat32.img #把这个文件格式化为 vfat 格式
```

开发板端:

```
# echo fat32.img > sys/devices/platform/jz-dwc2/dwc2/gadget/lun0/file
```

如果成功会在电脑端弹出 u 盘的盘符。

工程中为了方便用户的使用, 加入了 mass_storage demo, 用户可按照如下步骤进行验证:

1. Mass_storage demo 简介:

此功能主旨是为用户提供更加丰富的调试功能, 以便于用户开发。此功能基于 usb mass_storage, 同时结合开发板上的存储设备 (如: /dev/mmcblk0p1, /dev/ram0) 来形成。

2. 使用方法:

首先在开发板上准备一个存储设备分区, 如 sd 卡, nor 分区, emmc nand 分区等。

在开发板上执行如下命令:

```
# udc_mass_storage.sh <设备分区> <挂载目录>
```

以 mmc 为例, 命令为: udc_mass_storage.sh /dev/mmcblk0p1 /mnt/udc

就会在 PC 端弹出一个 u 盘, 此时 PC 端可以像操作普通 U 盘一样, 操作目标板挂载目

录所弹出的盘符。

3. 数据 copy

A. Pc 到目标板: 此种情况下与普通的 u 盘使用的方式是一样的, 将文件通过 pc 直接 copy 到目标板挂载目录即可。

B. 目标板到 pc: 此种情况下与普通的 u 盘使用的方式是一样的, 将目标板“挂载目录”中的文件通过 PC 端得盘符直接 copy 到 PC 上即可, 需要注意的是, 使用此种功能时, 如果需要 copy 的文件不在目标板的“挂载目录”下, 不能直接在目标板上 copy 所需文件到“挂载目录”下, 因为此时目标板的“挂载目录”为只读模式。用户需将 copy 文件准备到“挂载目录”下后, 才能使用此功能将所需文件 copy 到 PC 上。

4.12.3.2. rndis 功能

```
cd /sys/class/android_usb/android0
echo 0 > enable
echo 18d1 > idVendor
echo d002 > idProduct
echo rndis > functions
echo 1 > enable
```

配置完后同时在电脑端和设备端出现 usb0 网络端点, 可用 ifconfig -a 查看, 把这两个 usb0 配置成两个不同的 ip, 可用 ping 命令来检测是否成功。

4.12.3.3. USB Audio Gadget 功能

4.12.3.3.1. 驱动选项

Symbol: USB_GADGET [=y]

Type : tristate

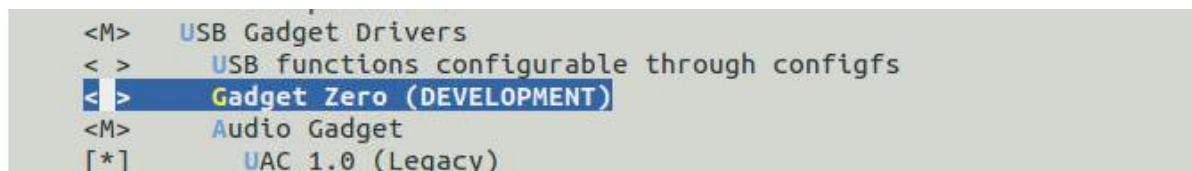
Prompt: USB Gadget Support

Location:

-> Device Drivers

-> USB support (USB_SUPPORT [=y])

以模块方式选择:



```
<M> USB Gadget Drivers
< > USB functions configurable through configs
< > Gadget Zero (DEVELOPMENT)
<M> Audio Gadget
[*] UAC 1.0 (Legacy)
```

4.12.3.3.2. 编译方式

\$ make uImage /*烧录新编译生成的 uImage */

\$ make modules /*以模块的方式编译 USB Audio*/

生成文件路径:

kernel/drivers/usb/gadget/

生成文件:

g_audio.ko

libcomposite.ko

将上述生成的.ko 文件 push 到开发板中:

```
$ adb push libcomposite.ko /
```

```
$ adb push g_audio.ko /
```

Note:

经过上述重选的驱动中没有支持 adb,不能正常使用 adb 命令,建议先将上述的 g_audio.ko libcomposite.ko 文件 push 到开发板中(在有 adb 支持的驱动条件下),之后重新烧录新编译的 ulmage.

4.12.3.3.3. 使用方式

开发板端:

```
# insmod libcomposite.ko
```

```
#insmod g_audio.ko
```

```
[ 40.929424] start set AIC register....
```

```
udhcpc: sending discover
```

```
[ 41.596439] g_audio gadget: play, Hardware params: access 3, format 2, channels 2, rate 48000
```

```
[ 41.605365] start set AIC register....
```

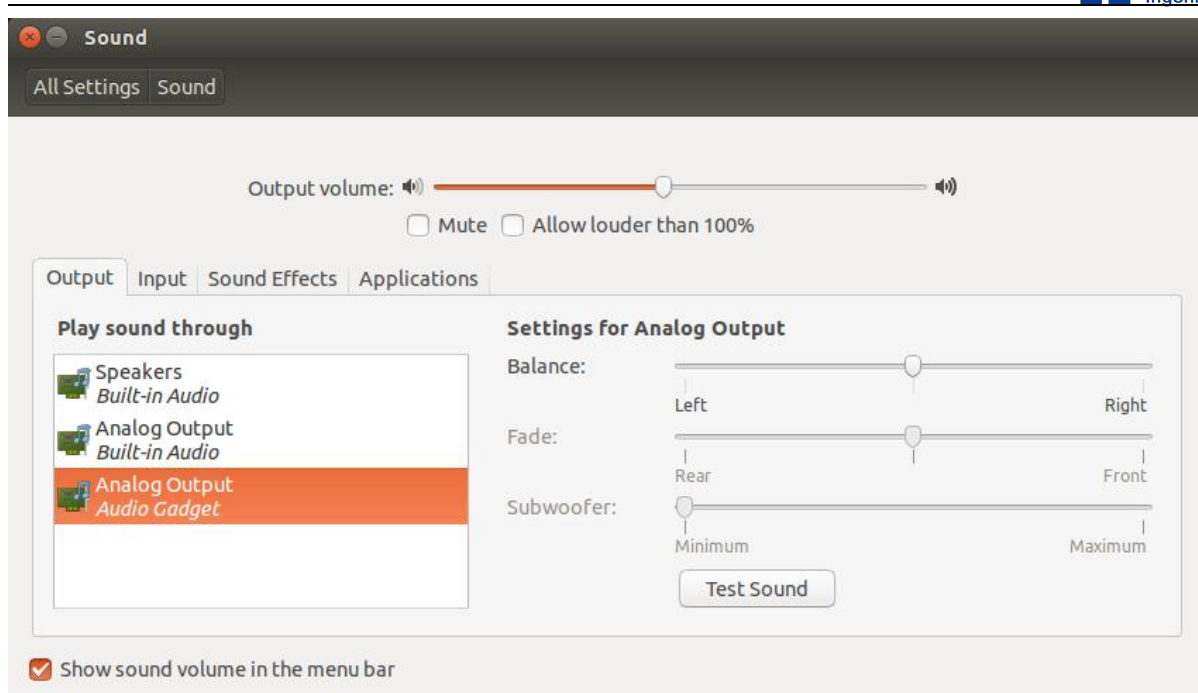
```
[ 41.611408] g_audio gadget: audio_buf_size 48000, req_buf_size 384, req_count 256
```

```
[ 41.626167] g_audio gadget: Linux USB Audio Gadget, version: Feb 2, 2012
```

```
[ 41.633090] g_audio gadget: g_audio ready
```

PC 端:

选中 Audio Gadget 设备:



之后通过 PC 端默认的播放软件播放音乐，开发板放音成功。

4.12.4. Debug

Linux 下的 minicom 的功能与 Windows 下的超级终端功能相似,可以通过串口控制外部的硬件设备。适于在 linux 通过超级终端对嵌入式设备行管理.同样也可以使用 minicom 对外置 Modem 进行控制。开发过程中需注意的,是 USB 线尽量使用正规厂家生产的数据线(信号质量好),不要使用非正规厂家生产的数据线(信号质量差),否则有可能出现不能识别设备的现象。

下面以 Halley2 的串口设置说明进行举例(Linux 环境)。

执行命令:

```
$ sudo minicom -s -w
```

参数说明:

-s: 进入设置模式

-w: 支持换行

串口配置:

```
A - Serial Device      : /dev/ttyUSB0
B - Lockfile Location  : /var/lock
C - Callin Program     :
D - Callout Program    :
E - Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No

Change which setting?
```

4.13. 蓝牙-WIFI 模块

蓝牙、wifi 芯片使用的是 BCM43438，发布的版本默认提供蓝牙 wifi 配置，这里只说明测试方法。

4.13.1. 蓝牙

使用步骤详解：

1. 开发板执行：# bt_enable

```
# bt_enable
[ 34.997129] restore_pin_status is not defined
port --baudrate ----- 5
=====read from uart_fd==(null)=Done setting line discipline
Broadcom firmware initialized.
#
```

2. 开发板执行：# sdptool add OPUSH

```
#
# sdptool add OPUSH
OBEX Object Push service registered
#
```

3. 链接设备开启蓝牙，搜索 BlueZ，点击该设备进行配对

注： 配对成功，开发板有如下相应提示

```
#
# Confirmation request of 232480 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
#
```

4. 开发板执行：# obex_test -b local 9

```
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
>
```

5. 链接设备准备好要传送的文件

6. 开发板执行 s （准备传输文件）

链接设备要在规定时间内将文件分享到 BlueZ 蓝牙

```
# sdptool add OPUSH
OBEX Object Push service registered
# Confirmation request of 359585 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
```

注：如果分享文件超时会有如下提示：

```
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
> s
Server register error! (Bluetooth)
```

注：分享超时，可重新执行第 6 步即可

7. 开发板执行 s（开始传输）

```
# sdptool add OPUSH
OBEX Object Push service registered
# Confirmation request of 359585 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
> s
connect_server()
connect_server() Skipped header c0
Server request finished!
server_done() Command (00) has now finished
OBEX_HandleInput() returned 12
OBEX_HandleInput() returned 990
Unknown event 0b!
Made some progress...
OBEX_HandleInput() returned 1024
Made some progress...
OBEX_HandleInput() returned 954
```

执行以下命令修改蓝牙名称并重启蓝牙设备：

```
# hciconfig hci0 name <BLUZ_NAME>
# hciconfig hci0 reset
```

重启休眠注意事项：

1. 重启后，开发板不会保存蓝牙设置，要使用蓝牙，需要将按上述步骤重新设置一遍。
2. 休眠后重启后，要先杀死下图中的进程，再重新设置一遍

```
134 root brcm_patchram_plus --enable_hci --baudrate 3000000 --no2bytes --
```

注意：

在多次链接蓝牙时，或链接蓝牙错误时，会建立多个重复的上述进程。要将这些进程清理干净，

4.13.2. WIFI

Wifi 使用操作步骤

一、手机端

1. 在 Android（Android 4.0 及以上）手机上安装 com.broadcom.cooeedemo-v1.4.0.apk（或者 AirKissDebugger.apk 都可，以下以 BrcmCooee 应用为例）。
2. 打开手机上的 WI-FI 并连接到一个 AP 路由器。
3. 运行 BrcmCooee 应用。
4. 在 BrcmCooee 应用中输入 SSID 名称(第一个输入项)，AP 路由器的密钥(第二个输入项)（确保 BrcmCooee 应用中所连接的路由器和手机连接的是同一个）。

二、开发板端

1. 开发板执行：# airkiss

```
# airkiss
killall: udhpc: no process killed
killall: wpa_supplicant: no process killed
[ 900.361412] BCM:
[ 900.361412] Dongle Host Driver, version 1.141.66
[ 900.361412] Compiled in drivers/net/wireless/bcmdhd on Apr 8 2016 at 17:47:47
[ 900.376280] BCM:wifi_android_wifi_on in
[ 900.380068] BCM:wifi_platform_set_power = 1
[ 900.384920] BCM:===== WLAN placed in RESET ON =====
[ 900.612576] sdio_reset_comm():
[ 900.641229] BCM:F1 signature OK, socitype:0x1 chip:0xa9a6 rev:0x0 pkg:0x4
[ 900.649431] BCM:DHD: dongle ram size is set to 524288(orig 524288) at 0x0
[ 900.780699] BCM:dhd_write_vars: Download, Upload and compare of NVRAM succeeded.
[ 900.838309] BCM:dhd_bus_init: enable 0x06, ready 0x06 (waited 0us)
[ 900.846643] BCM:wifi_platform_get_mac_addr
[ 900.852523] BCM:dhd_get_concurrent_capabilities: Get P2P failed (error=-23)
[ 900.860231] BCM:Firmware up: op_mode=0x0001, MAC=00:90:4c:c5:12:38
[ 900.872060] BCM:dhd_preinit_ioctl pspretend_threshold for HostAPD failed -23
[ 900.885051] BCM:Firmware version = wl0: Jun 12 2015 17:06:46 version 7.10.323.47.cn2.x7 FWID 01-dcca911a es4.c3.n3.a2
[ 900.897361] BCM:dhd_wlfc_hostreorder_init(): successful bdcv2 tlv signaling, 64
Easy setup target library v3.2.0
```

当出现“Easy setup target library v3.2.0”时，表示开发板 wifi 准备成功，此时可以执行下一步操作。

2. 链接设备按提示点击手机端 BrcmCooee 应用的”Start”按钮发送 AP 的 ssid 与密钥
开发板提示：


```
ssid: JZ_MD
password: 1JZmdingenic2
/etc/wpa_supplicant.conf create successfully!
random: 0x67
/etc/airkiss_random.conf create successfully!
[ 1020.587477] BCM:CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
[ 1023.605034] BCM:CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
[ 1026.613089] BCM:CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
security: wpa2# killall: udhcpc: no process killed
killall: wpa_supplicant: no process killed
Successfully initialized wpa_supplicant
[ 1029.797524] BCM:CFG80211-ERROR) wl_cfg80211_connect : BCM:Connecting withff:ff:ff:ff:ff:ff channel (0) ssid "JZ_MD", len (5)

udhcpc (v1.22.1) started
Sending discover...
[ 1030.905184] BCM:wl_bss_connect_done succeeded with 8c:0c:90:d8:47:c8
[ 1030.985464] BCM:wl_bss_connect_done succeeded with 8c:0c:90:d8:47:c8
Sending discover...
Sending select for 10.10.50.35...
Lease of 10.10.50.35 obtained, lease time 86400
deleting routers
adding dns 192.168.1.2
```

当出现“adding dns 192.168.1.2”时，表示开发板接受 wifi 密钥完成，此时可以执行下一步操作。

注：ssid, password 为 wifi 的用户名和密码。其被保存在/etc/wpa_supplicant.conf 文件中

```
# cat /etc/wpa_supplicant.conf

network={
scan_ssid=1
ssid="JZ_MD"
psk="1JZmdingenic2"
priority=1
}
```

3. 开发板执行 ping www.baidu.com, 验证 wifi 是否连接成功

```
# ping www.baidu.com
PING www.baidu.com (220.181.111.188): 56 data bytes
64 bytes from 220.181.111.188: seq=0 ttl=54 time=241.266 ms
64 bytes from 220.181.111.188: seq=1 ttl=54 time=19.682 ms
64 bytes from 220.181.111.188: seq=2 ttl=54 time=19.649 ms
```

wifi 重启休眠共用等注意事项

1. 重启，休眠唤醒后开发板会根据/etc/wpa_supplicant.conf 自动连接 wifi。

执行 wifi 功能的进程:

```
108 root wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf -B
```

2. Wifi 和 Lan 不可同时使用

Wifi 链接时会配置 wlan0, Lan 链接时会配置 eth0, 两者造成冲突。

如在 wifi 链接成功后, 需要使用 LAN, 需要删除/etc/wpa_supplicant.conf (其会导致

wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf 不执行), 即可。

4.14. WIFI AP 模式

4.14.1. AP 模式使用方法

开发板启动后，执行以下命令进入 wifi 的 AP 模式：

```
# sta_wifi_ap.sh
```

出现如下的打印代表 wifi 的 AP 模式使能成功：

```
wlan0: interface state UNINITIALIZED->ENABLED
```

```
wlan0: AP-ENABLED
```

Wifi 的 AP 模式启动后，执行以下命令：

```
# ifconfig
```

命令执行完成后，发现 wlan0 网卡启动并被分配出一个 ip ， inet addr:192.168.1.1：

```
# ifconfig
lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            UP LOOPBACK RUNNING  MTU:65536  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0       Link encap:Ethernet  HWaddr 02:1A:11:F6:BC:71
            inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:88 errors:0 dropped:12 overruns:0 frame:0
            TX packets:40 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:33151 (32.3 KiB)  TX bytes:11936 (11.6 KiB)
```

4.14.2. AP 模式的验证

1.AP 模式相关配置文件描述：

配置文件路径：/etc/hostapd.conf

在 hostapd.conf 文件中存有 AP 模式相关的配置信息，包括 ssid ,password 等，可根据实际情况进行修改。

在默认的配置中 ssid 为“ingenic_test”，对应文件描述：ssid=ingenic_test;

password 为 “123456789”，对应文件描述为：wpa_passphrase=123456789。

2.验证方法:

(1) 打开手机中的 wifi，搜索“ingenic_test” 密码“123456789”，连接此 ssid。手机端连接成功后，开发板端会有如下打印，提示有设备连接成功，并分配给设备一个 ip 地址：

```
# wlan0: STA 10:bf:48:d0:10:29 IEEE 802.11: associated
[ 257.244026] BCM:CFG80211-ERROR) wl_cfg80211_get_key : BCM:Invalid algo (0x46)
wlan0: AP-STA-CONNECTED 10:bf:48:d0:10:29
wlan0: STA 10:bf:48:d0:10:29 WPA: pairwise key handshake completed (RSN)
Sending OFFER of 192.168.1.169
Sending ACK to 192.168.1.169
```

(2) 将手机连接到 PC 端，进入手机端 adb shell 命令行模式，通过 netcfg 命令查看此时手机端的 ip 为 192.168.1.169，此 ip 与开发板端提示信息吻合。

```
shell@grouper:/ $ netcfg
lo          UP                127.0.0.1/8      0x00000049 00:00:00:00:00:00
dummy0      DOWN           0.0.0.0/0        0x00000082 aa:57:71:64:be:b9
sit0        DOWN           0.0.0.0/0        0x00000080 00:00:00:00:00:00
ip6tnl0     DOWN           0.0.0.0/0        0x00000080 00:00:00:00:00:00
p2p0        UP                0.0.0.0/0        0x00001003 12:bf:48:d0:10:29
wlan0       UP                192.168.1.169/24 0x00001043 10:bf:48:d0:10:29
```

可以通过手机端与开发板端是否可以互相通信来进行验证，手机端执行 ping 命令：

```
shell@grouper:/ $ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=21.9 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=19.5 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=15.0 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=14.4 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=15.8 ms
^C
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 14.487/17.368/21.911/2.875 ms
```

开发板端执行 ping 命令：

```
# ping 192.168.1.169
PING 192.168.1.169 (192.168.1.169): 56 data bytes
64 bytes from 192.168.1.169: seq=0 ttl=64 time=6.554 ms
64 bytes from 192.168.1.169: seq=1 ttl=64 time=120.414 ms
64 bytes from 192.168.1.169: seq=2 ttl=64 time=122.669 ms
64 bytes from 192.168.1.169: seq=3 ttl=64 time=122.067 ms
64 bytes from 192.168.1.169: seq=4 ttl=64 time=9.508 ms
64 bytes from 192.168.1.169: seq=5 ttl=64 time=7.888 ms
^C
--- 192.168.1.169 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 6.554/64.850/122.669 ms
```

开发板与手机端可以互相 ping 通，代表两者通信成功，AP 模式可正常使用，至此 AP 模式验证完成。

4.14.3. AP 模式切换 wifi 模式

执行以下命令将 ap 模式切换回 wifi 模式：

```
# ap_wifi_sta.sh
```

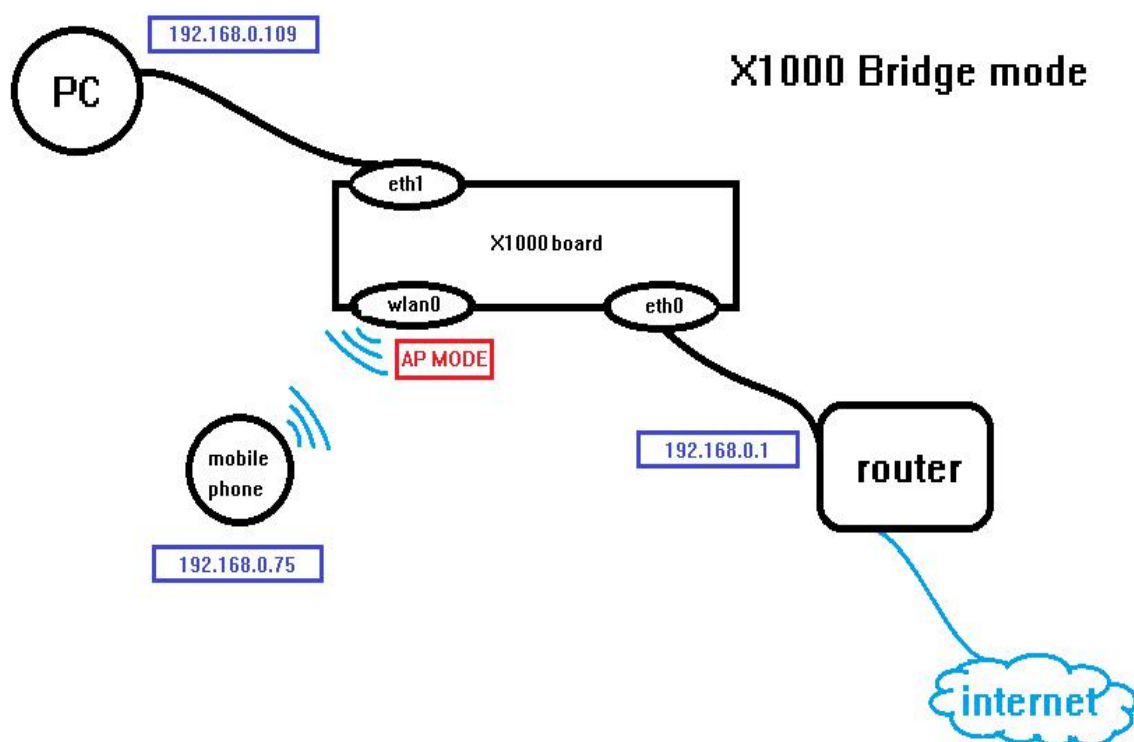
4.15. Network 工作模式

关于 x1000 的网络工作模式，下面以桥接模式，Nat 模式进行分开描述说明。

4.15.1. 桥接模式

4.15.1.1. 原理及说明

一 桥接模式原理



二 网络接口说明

在 X1000 开发板上, 我们有 3 个网络接口:

eth0: 连接到路由器, 路由器提供 DHCP 服务;

eth1: 此处连接 USB 网卡, 连接到 PC 终端;

wlan0: wifi 网卡, wlan0 配置成 AP 模式;

4.15.1.2. 内核配置

\$ make menuconfig

```

Symbol: BRIDGE [=y]
| Type : tristate
| Prompt: 802.1d Ethernet Bridging
| Location:
|   -> Networking support (NET [=y])
|   -> Networking options
| Defined at net/bridge/Kconfig:5
| Depends on: NET [=y] && (IPV6 [=n] || IPV6 [=n]=n)
| Selects: LLC [=n] && STP [=n]

```

4.15.1.3. 设置桥接

以下操作均在开发板端运行。

一 设置 wlan0 为 AP 模式

```
# sta_wifi_ap.sh
```

清除所有网络接口 ip:

```
# ifconfig eth0 0.0.0.0
# ifconfig eth1 0.0.0.0
# ifconfig wlan0 0.0.0.0
```

二 创建桥接 br0

```
# brctl addbr br0
```

三 添加所有网络接口为桥接:

```
# brctl addif br0 eth0
# brctl addif br0 eth1
# brctl addif br0 wlan0
```

四 使能桥接

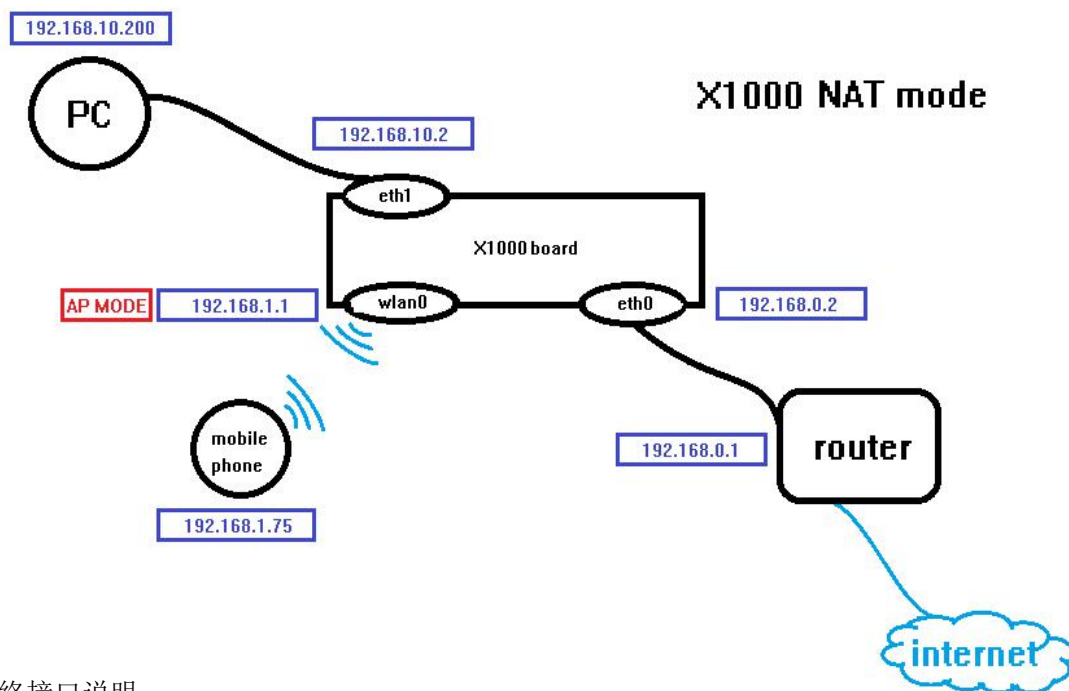
```
# ifconfig br0 up
```

桥接配置完成后，客户端（PC 和手机）都可以访问路由器。若路由器连接互联网，则客户端也可以访问互联网。

4.15.2. NAT 模式

4.15.2.1. 原理及说明

一 NAT 模式原理



二 网络接口说明

在 X1000 开发板上, 我们有 3 个网络接口:

eth0: 连接到路由器, 路由器提供 DHCP 服务;

eth1: 此处连接 USB 网卡, 连接到 PC 终端;

wlan0: wifi 网卡, wlan0 配置成 AP 模式;

4.15.2.2. 内核配置

```
[*] Networking support --->
    Networking options --->
        [*] Network packet filtering framework (Netfilter) --->
            Core Netfilter Configuration --->
                <*> Netfilter connection tracking support
                    [ ] Connection mark tracking support
                    [*] Supply CT list in procfs (OBSOLETE)
                    [ ] Connection tracking events
                    [ ] Connection tracking timeout
                    [ ] Connection tracking timestamping
            IP: Netfilter Configuration --->
```

```
<*> IPv4 connection tracking support (required for NAT)
[*] proc/sysctl compatibility with old connection tracking
<*> IPv4 NAT
<*> MASQUERADE target support
```

4.15.2.3. 设置 NAT

开发板端:

一 设置 eth0

```
# udhcpc -i eth0
```

二 设置 eth1

```
# ifconfig eth1 192.168.10.2 netmask 255.255.255.0
```

三 确保配置文件 “udhcpd.conf” router、 dns 信息正确

```
# vi /etc/udhcpd.conf
opt      router      192.168.1.1
opt      dns      'your DNS server'
```

四 设置 wlan0 为 AP 模式

```
# sta_wifi_ap.sh
```

五 开启路由转发

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

六 设置转发规则

```
# iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
```

七 检查 route 表

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default localhost 0.0.0.0 UG 0 0 0 eth0
192.168.0.0 * 255.255.255.0 U 0 0 0 eth0
192.168.10.0 * 255.255.255.0 U 0 0 0 eth1
192.168.1.0 * 255.255.255.0 U 0 0 0 wlan0
```

PC 端:

```
$ sudo route add default netmask 255.255.255.0 gateway 192.168.10.2
```

至此 NAT 模式配置完成。

4.16. TouchScreen

Halley2 默认的屏幕是液晶屏，若您想要添加触摸屏驱动支持，可参考以下流程，下面以在 halley2 开发板上添加触控 IC FT6236 为例。

4.16.1. 内核配置

编译选项如下：

```
$ make menuconfig
Symbol: TOUCHSCREEN_FT6X06 [=y]
| Type : tristate
| Prompt: FocalTech FT6X06/FT6236 TouchScreen driver
| Location:
|   -> Device Drivers
|       -> Input device support
|           -> Generic input layer (needed for keyboard, mouse, ...) (INPUT [=y])
|               -> Touchscreens (INPUT_TOUCHSCREEN [=y])
| Defined at drivers/input/touchscreen/Kconfig:29
```

4.16.2. 代码目录及描述

触摸相关的代码分布在以下目录及文件中：

[1] [arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/i2c_bus.c](#)

[2] [kernel/drivers/input/touchscreen/ft6x06_ts.c](#)

4.16.2.1. 注册触摸设备信息

[arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/i2c_bus.c](#)

其中主要相关的数据结构为：

1) struct **i2c_board_info** jz_i2c2_devs[]

.type	sensor 的名称
.addr	设备地址
.platform_data	自定义设备信息，如 struct ft6x06_platform_data

根据实际触摸设备连接的 I2C 通道，将信息填入对应的结构中，如这里的 jz_i2c2_devs。

2) struct **ft6x06_platform_data** ft6x06_tsc_pdata

.va_x_max	触摸屏宽
-----------	------

.va_y_max	触摸屏长
.irqflags	触摸 IC 的中断 PIN 属性，如触发电平
.irq	触摸 IC 的中断 PIN
.reset	触摸 IC 的复位 PIN
	irq 和 reset 使用的 GPIO 定义在 arch/mips/xburst/soc-x1000/chip-x1000/halley2/halley2_v10/board.h

4.16.2.2. 触摸屏驱动

[kernel/drivers/input/touchscreen/ft6x06_ts.c](#)

其中较主要的接口：

request_irq	注册 irq 的 handle
ft6x06_work_handler	触摸事件处理
ft6x06_report_value	上报触摸事件

具体实现请自行分析。

4.17. Button

4.17.1. Kernel 的配置

Symbol: INPUT_POLLDEV [=y]

Type : tristate

Prompt: Polled input device skeleton

Location:

-> Device Drivers

-> Input device support

-> Generic input layer (needed for keyboard, mouse, ...) (INPUT [=y])

Symbol: INPUT_EVDEV [=y]

Type : tristate

Prompt: Event interface

Location:

-> Device Drivers

-> Input device support

-> Generic input layer (needed for keyboard, mouse, ...) (INPUT [=y])

Symbol: KEYBOARD_GPIO [=y]

Type : tristate

Prompt: GPIO Buttons

Location:

-> Device Drivers

-> Input device support

-> Generic input layer (needed for keyboard, mouse, ...) (INPUT [=y])

-> Keyboards (INPUT_KEYBOARD [=y])

4.17.2. Button demo

我们在 Manhatton 工程中有关于 button 的测试 demo:

路径: packages/example/Sample/button

以 Halley2 开发板为例需要将捕获事件由 “event1 ” 改为”event 0”

路径: packages/example/Sample/button/button.c

Source :

```
#define BUTTON_EVENT "/dev/input/event1"
```

Target:

```
#define BUTTON_EVENT "/dev/input/event0"
```


修改完成后可通过在当前目录下执行”mma “进行单独编译，可执行文件”button “生成路径为”out/product/halley2/obj/packages/example/Sample/button/”

开发板端 button Demo 执行效果:

```
# ./button
===== type = 1 code = 116 value = 0
===== type = 0 code = 0 value = 0
===== type = 1 code = 116 value = 1
===== type = 0 code = 0 value = 0
```

4.18. SECURITY 加密模块

AES-RSA 的驱动名称是 “jz-security”

(1) 设备节点: /dev/jz-security

使用如下的方式来操作驱动:

open(/dev/jz-security, ...) → ioctl(xxx...) → ioctl(xxx...) → ...→ ioctl(xxx) -->close(device)

(2) 驱动文件路径:

“kernel/drivers/misc/jz_security/”

4.18.1. 驱动配置

Device Drivers

-->Misc devices

-->JZ SECURITY Driver(AES && RSA)

4.18.2. IOCTL 命令定义

定义文件:kernel/drivers/misc/jz_security/jz_security.h

```
#define SECURITY_INTERNAL_CHANGE_KEY (0xffff0010)
/*设置 AES-KEY*/
#define SECURITY_INTERNAL_AES (0xffff0020)
/*AES 加密或解密*/
#define SECURITY_RSA (0xffff0030)
/*RSA 加密或解密*/
```

4.18.3. 驱动结构体描述

以下驱动结构体定义路径为“kernel/tools/security-test”。

(1) 定义 AES-KEY 结构如下：

文件路径：kernel/tools/security-test/sec_test.h

```
struct rsa_aes_packet {
    unsigned short oklen; //old security key len in words
    unsigned short nklen; // new security key len in words
    unsigned      int * okey; // old security key
    unsigned int * nkey; // new security key
};
```

也可以用数组来存储 AES-KEY，具体定义方式可见下例：

文件路径：kernel/tools/security-test/sec_test.c

```
unsigned int user_key[9] =
{
    0x40004, /* bit[31:16] = oldkey_len, bit[15:0] = newkey_len */
    0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c, /*old_key*/
    0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c, /*new_key*/
};
```

注：初始化 AES-KEY 时，需将“old AES-KEY”“new AES-KEY”设置成相同的初值。

(2) RSA 参数结构

文件路径：kernel/tools/security-test/sec_test.h

用户使用 RSA 加解密“AES-KEY”，描述 RSA 的结构体为“struct rsa_param”，具体结构如下：

```
struct rsa_param {
    unsigned int in_len;      //input data length (units:word)
    unsigned int key_len;     //private or public key length(units:word)
    unsigned int n_len;       //n length(units:word)
    unsigned int out_len;     //ouput length(units:word)
    unsigned int *input;      //input data buffer
    unsigned int *key;        /*Ku or KR*/ buffer
    unsigned int *n;          /*N*/buffer
    unsigned int *output;     //encrypted data or decrypted data buffer
    unsigned int mode;        //mode: 1,encrypt 0,decrypt
};
```

(3) CHANGE-KEY 结构体

```
struct change_key_param {
    int len;                  //rsa_enc_data len in bytes.
    int *rsa_enc_data;        //okey_len,nkey_len,okey,nkey
    int *n_ku_kr;             //NKU or NKR buffer, for rsa decrypt (62 words)
    int init_mode;            //init key 1;init key, 0:change key
};
```

其中 rsa_enc_data 是上述结构体“struct rsa_aes_packet”经过 rsa 加密后的数据，上述加密数据

的 NKU 或 NKR。

(4) AES 加解密参数

目前我们只支持 ECB 模式，AES-KEY 大小为 128-bit，支持每次四 word 的 AES 加密解密。

```
struct aes_param {
    unsigned int in_len;           //input data length (units:word)
    unsigned int key_len;         //private or public key length(units:word)
    unsigned int n_len;           //n length(units:word)
    unsigned int out_len;         //ouput length(units:word)
    unsigned int *input;          //input data buffer
    unsigned int *key;            /*Ku or KR*/ buffer
    unsigned int *n;              /*N*/buffer
    unsigned int *output;         //encrypted data or decrypted data buffer
    unsigned int mode;            //mode: 1,encrypt 0,decrypt
};
```

4.18.4. 编程指导

一：打开设备并初始化 AES 控制器。

二：使用正确格式的 AES-KEY，并进行 RSA 加密。

三：使用步骤 1 中的结果，设置 AES-KEY 到 CPU。

四：执行 AES 加密或解密。

若想改变 AES-KEY，请执行步骤二，步骤三，在步骤二中应注意“1”为初始 key 值，“0”为改变后的 key 值。

4.18.5. 用户 API

(1) rsa 加解密

```
int do_rsa(unsigned int fd,      //file node
    unsigned int orig_aes_len,  // original AES_KEY length
    unsigned int * rsa_key,     //KU or KR
    unsigned int * n,
    unsigned int * input,       // input_data
    unsigned int *output,       //encrypted or decrypted data
    unsigned int mode);         //mode: 0:encryption 1:decryption
```

(2) 设置 AES-KEY 或改变 AES-KEY

```
int setup_aes_key(int fd,       //file node
    unsigned int *key,          //encrypted AES-KEY by using RSA
    int len,                    // length of key
    unsigned int *nku_kr,       //NKU ok NKR(62 words)
    int init_mode);             //init_mode: 1:init code , 0:change code
```

(3) AES 加密解密

```
int do_aes(int fd,             //file node
    unsigned int *input,        //input data
```

unsigned int in_len, //input and output data length(unit:word)
unsigned int *output, //output data(encrypted data or decrypted data)
unsigned int mode); //mode :0,encryption 1:decryption

4.19. 语音唤醒

Voice trigger(语音休眠唤醒)主要是利用 linux 系统和君正处理器支持休眠唤醒等特点,使君正方案达到更好省电效果。

Voice trigger 的代码可划分为两部分:第一部分为语音识别测试的代码;第二部分为语音休眠唤醒的代码。

文件描述:

语音休眠唤醒固件代码: `drivers/char/voice_wakeup_v13/`

语音识别测试驱动代码: `drivers/char/jz_wakeup_v13.c`

提供给其他应用的 DMIC 驱动: `drivers/char/jz_dmic_v13.c`

测试用例代码: `tools/wakeup-test/wakup.c`

测试用例所需的语音比对文件: `tools/wakeup-test/ivModel_v21.irf`

linux 系统休眠后的入口文件: `arch/mips/xburst/soc-x1000/common/pm_p0.c`

4.19.1. Voice trigger 驱动配置方法

1.在“`drivers/char/voice_wakeup_v13/wakeup_module/`”目录下,首先执行 `make clean`,然后执行 `./mkmodule.sh`

配置编译选项:

Symbol: `WAKEUP_MODULE_V13` [=y]

Type : boolean

Prompt: Ingenic Voice Wakeup Module

Location:

-> Device Drivers

-> Character devices

Defined at `drivers/char/Kconfig:24`

Depends on: `SOC_X1000` [=y]

上述选项配置完成后执行以下命令进行编译:

```
$ make uImage
```

编译完成后会在“`kernel/arch/mips/boot/`”目录下产生 voice trigger 需要的 uImage。

4.19.2. 验证方法

将内核中 `tools/wakeup-test/wakeup` 目录下的 `wakeup` 及 `ivModel_v21.irf` 拷贝到文件系统。
在串口端输入以下命令 (后台运行):

```
# ./wakeup ivModel_v21.irf &
```

```
# ./wakeup ivModel_v21.irf &
# open file[ivModel_v21.irf], ok!
open file[/dev/jz-wakeup] ok![ 2019.532480] enable wakeup function

open file[/sys/class/jz-wakeup/jz-wakeup/wakeup] ok!
read don[ 2019.546428] module open open_cnt = 1
e!!!!
#### begin read !!!!!
```

出现上述打印后对着开发板上的音频模块音频输入“灵犀灵犀”，串口出现如下打印：

```
WKUPOK[ 2026.691348] sys wakeup ok!--wakeup_timer_handler():158, wakeup_pending:1
[ 2026.705626] [Voice Wakeup] wakeup by voice.
#####ret:9, wakeup_ok
sh: input: not found
#####read ok!, wakeup ok!
#### begin read !!!!!
```

之后使开发板进入休眠状态 `echo mem > /sys/power/state`,休眠之后通过音频“灵犀灵犀”唤醒系统。

4.20. 快速启动配置

1.Uboot :

配置选项: halley2_vmlinux_nor_spl_boot

编译命令:

```
$make halley2_vmlinux_nor_spl_boot
```

编译完成后在 uboot 当前目录下生成文件 “u-boot-with-spl.bin”

可以通过修改 loglevel 的打印级别来提升启动速度（详见 3.7 log 打印级别 loglevel），默认最高级别 “7”，可将其修改为最低级别 “0”：

```
#define BOOTARGS_COMMON "console=ttyS2,115200n8 mem=32M@0x0 loglevel=7 "
```

```
#define BOOTARGS_COMMON "console=ttyS2,115200n8 mem=32M@0x0 loglevel=0"
```

2.Kernel :

配置选项: halley2_nor_v10_fast_linux_defconfig

编译命令:

```
$make halley2_nor_v10_fast_linux_defconfig
```

```
$make uImage
```

编译完成后生成 “kernel/arch/mips/boot/vmlinux.bin”

3.文件系统类型: cramfs 类型的文件系统即可。

5. QEMU

本章节主要对如何使用 QEMU 协助应用程序和系统开发进行说明。

QEMU 有两种主要运作模式：

User mode 模拟模式，即是用户模式。QEMU 能启动那些为不同中央处理器编译的程序。

System mode 模拟模式，即是系统模式。QEMU 能模拟整个电脑系统，包括中央处理器及其他周边设备。它使得为跨平台编写的程序进行测试及除错工作变得容易。也能用来在一部主机上虚拟多部不同虚拟电脑。

5.1. User Mode

直接运行应用程序即使用了 QEMU 的 user 模式：

```
$/PATH/TO/INSTALL/bin/qemu-mipsel -cpu xburst1-x1000 -L /PATH/TO/LIBC  
/PATH/TO/THE/APP
```

说明：

1. 运行应用程序需要指明 libc 的路径，也可以直接指定到 gcc 的 libc，例如：
-L Project/prebuilts/toolchains/mips-gcc472-glibc216/mips-linux-gnu/libc
2. -cpu X1000 对应的 CPU 名字是 xburst1-x1000
3. /PATH/TO/THE/APP 是应用程序所在的路径。

Note:在 Manhatton 工程“prebuilts/mips-qemu”目录下是直接可以使用的 qemu 工具。

5.2. System Mode

使用 QEMU 启动 Linux 使用了 QEMU 中的 system 模式。

5.2.1. 启动脚本

5.2.1.1. JZ xburst1-x1000 CPU 启动脚本

```
#!/bin/bash
QEMU_PATH=/PATH/TO/INSTALL/bin                #QEMU 的安装路径
QEMU=qemu-system-mipselrootfs=/PATH/TO/ROOTFS    #要启动的文件系统
kernel=/PATH/TO/VMLINUX                        #要启动的 kernel
echo "----- Xburst1 X1000 is booting $1 rootfs-----"
    $QEMU_PATH/${QEMU} \
    -M phoenix -cpu xburst1-x1000 \
    -kernel ${kernel} \
    -append "console=ttyS1,57600n8 rdinit=/linuxrc root=/dev/ram0 rw
mem=256M@0x0mem=768M@0x30000000" \
    -initrd ${rootfs} \
    -serial /dev/tty -serial /dev/tty \
    -nographic \
    -net nic,model=rtl8139 \
    -net user,hostfwd=tcp::2030-:22,hostfwd=tcp::2089-:80
                                #定义的 ssh 端口号    #定义的 http 的端口号
```

说明:

1. X1000 使用的 kernel 为 ingenic-master-3.10.14 分支，phoenix_linux_qemu_defconfig 进行编译。
2. QEMU_PATH 中最后的 '/bin' 是必须的。
3. rootfs 需要指定 rootfs 文件。上例中使用的是 ramdisk (-initrd 以及 root=/dev/ram0) 承载文件系统，所以一般来说镜像文件应该是 xxx.cpio.gz；如果是使用 sd 卡 (-sd 以及 root=/dev/mmcblk0) 承载文件系统，一般来说镜像文件应该是 xxx.ext4。
4. X1000 对应的 CPU 名字是 xburst1-x1000，-M 名字是 phoenix。
5. 如果在一台主机上启动多个 QEMU，不同的 QEMU，最后一行的 ssh 端口号和 http 端口号不可相同。

5.2.1.2. 文件系统启动后网络配置

以上方式启动后是不可以使用 ssh 和其他机器进行交互的，需要进行如下配置(在 QEMU 中进行操作):

```
# mknod /dev/fuse c 10 229
```

```
# ifconfig eth0 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255  
# route add default gw 10.0.2.2 dev eth0
```

5.2.1.3. 免密码登录文件系统

以 ssh 端口号 2030，主机的 IP 是 192.168.1.2 为例(在主机中进行操作):

```
$ ssh-copy-id -p 2030 root@192.168.1.2
```

6. 烧录工具使用

烧录工具能够将 uboot, kernel 和文件系统的镜像烧录到 flash 中, 具体使用方法参考烧录工具文档《USBCloner-<Version>烧录工具快速上手指南》

注: 烧录工具及文档路径: *prebuilts/burnertools/*