lizhiyong_beyond@163.com

Arm 20 15 + 5

1 什么是嵌入式

2 搭建嵌入式开发环境

3 arm 架构 arm 指令 AEE www.arm.com

4 裸板编程

1 GPIO

LED BUTTON

2 arm 内存映射

arm 内存和外设统一编址

3 CLOCK

12M---->533M

4 DDR

5 NAND

6 UART

7 异常

reset

unde

svc swi 系统调用

irq 中断

8 ADC

9 TS

9 LCD

10 ROTATOR

11 AC97 Wm9714

12 Dm9000(ETH)

13 ETH IP UDP TFTP

裸板 tftp 客户端

14 MMU MPU

15 RTC

16 PWM(TIMER)

5 测试

6 项目

用户态移植：俄罗斯方块 网络多播

自己写 bootloader

移植 uboot-2012-04

自己写数码相框

串口和以太网接口的转换(rj45)

下载课程资料

Mount 192.168.1.253:/home/hero/student/20130509 /mnt

Umount /mnt

一、什么是嵌入式

手机：图形界面 图形程序 QQ 微信

GUI QT(C++) Android(JAVA) GTK(C) MiniGUI

本地可执行程序 ls ps

文件系统

根文件系统 linux 启动后挂载的第一个文件系统

磁盘文件系统

VFS

ramfs

Proc /proc

Sysfs /sys

Tmpfs

/tmp

操作系统内核

Arm_linux wince andorid uclinux uc/os

Linux

2.4 readhat9

2.6 设备模型 支持内核抢占

3.4.4

3.4.24

Bootloader

Vivi uboot myboot

检测和初始化硬件 引导内核

硬件

手机　平板　开发板

V4　arm7　arm920T(v4T)

V5　arm9

V6 arm11

V7　arm-cotex-A5/7/8/9/15

Arm-cotex-R4/5/6

Arm-cotex-M0/3

Secure

SOC system on chip

S3C6410

Arm9　　S3C2440

Arm11　　S3C6410

Arm-cotexA9　OMAP4460　FS6Q

开发板：

SMDK6410

MINI6410

TY6410

OK6410

QT6410

UP6410

开发板接口简介

开发板的内存地址范围：0x50000000-0x58000000

arm-linux-source 开发包简介

zImage u-boot-movi.bin u-boot-nand.bin rootfs2010*

zImage3.4.24 u-boot-movi_for3.4.bin u-boot-nand_for3.4.bin rootfs2013*

二、搭建嵌入式开发环境

1.把 u-boot-movi.bin 写到 SD 卡

a)把 SD 卡插入 pc

b)Fdisk -l 查看 SD 卡对应的设备文件

c)./write_sd /dev/sdb ../images/u-boot-movi.bin

2.配置 minicom

a)minicom -s

| A - Serial Device:/dev/ttyUSB0　串口设备文件

| B - Lockfile Location:/var/lock

| E - Bps/Par/Bits:115200 8N1　数据协议

115200 波特率

8N1 每贞数据 8 个有效位一个停止位无校验位

| F - Hardware Flow Control : No

| G - Software Flow Control : No

b)运行 minicom

Minicom

c)重启开发板查看 uboot 输出信息

3.uboot 命令和环境变量简介

Saveenv 保存环境变量

Printenv 打印环境变量

Setenv 修改或者删除环境变量

4.修改开发板的环境变量

Set serverip 192.168.1.253

Save

在用开发板 ping 主机(和开发板链接的 PC)

Ping 192.168.1.253

……is alive 表示通

5.配置 tftp 服务器

a)vim /etc/xinetd.d/tftp

Disable = no

Server_args = -s /tftpboot

b)service xinetd restart

c)关闭防火墙 关闭 selinux

Redhat6 setenforce 0

d)测试

[root] cp <path>/smdk6410_lzy/image/* /tftpboot

[u-boot-sd]# tftp 50008000 zImage

[u-boot-sd]# bootm 50008000 如果启动成功 LCD 上

会出现小企鹅

6.配置 nfs 服务

[root]# vim /etc/exports

/nfsroot *(rw,sync,no_root_squash)

[root]# service nfs restart

[root]# mkdir /nfsroot

[root]# chmod 777 /nfsroot

7.给开发板制作网络根文件系统

[root]# tar -xvf <path>/smdk6410_lzy/rootfs/rootfs20101220.tar -C /nfsroot

打开 minicom 自动换行　ctrl + a　w

退出 minicom ctrl + a　q

[u-boot-sd]# set bootargs root=/dev/nfs nfsroot=192.168.1.253:/nfsroot ip=192.168.1.20 console=ttySAC0

[u-boot-sd]# sav

测试：

[u-boot-sd]# tftp 50008000 zImage

[u-boot-sd]# bootm 50008000 如果启动成功 LCD 上会出现小企鹅，过一会后 LCD 上出现图形界面，minicom 中出现控制终端(shell)

8.矫正触摸屏

先挂载文件系统，然后删除矫正文件

[root@uplooking /]# rm /etc/pointercal

重新启动开发板

[u-boot-sd]# tftp 50008000 zImage

[u-boot-sd]# bootm 50008000

启动成功后会出现矫正界面

9.安装交叉编译工具链

Gcc ---->x86

arm-linux-gcc/as/ld/objdump

mkdir /usr/local/arm/

tar -jxvf arm-linux-gcc.tar.bz2 -C /usr/local/arm/

vim ~/. bashrc vim /etc/profile vim /etc/bashrc

export PATH=/usr/local/arm/4.3.2/bin:$PATH

Cd

Source ~/.bashrc

Arm-linux-gcc -v

测试：

[root]# Cd /nfsroot

[root]# Touch hello.c

[root]# Arm-linux-gcc hello.c -o hello

[root@uplooking]# ./hello

三、把 uboot zImage rootfs 都写到开发板

在内核启动过程中会输出：

0x00000000-0x00040000："Bootloader"

/dev/mtdblock0

0x00040000-0x00400000："Kernel"

/dev/mtdblock1

0x00400000-0x05400000："Rootfs"

/dev/mtdblock2

0x05400000-0x10000000："File System"

/dev/mtdblock3

把 uboot 写到 nand

[u-boot-sd]# nand erase

[u-boot-sd]# nand scrub

y enter

[u-boot-sd]# tftp 50000000 u-boot-nand.bin

[u-boot-sd]# nand write 50000000 0 40000

关掉电源，切换为 nand 启动，上电，如果成功出现：

[u-boot-nand]#,这时候会有 ECC 错误， 解决办法：

[u-boot-nand]# set serverip 192.168.1.253

[u-boot-nand]# tftp 50000000 u-boot-nand.bin

[u-boot-nand]# nand erase 0 40000

[u-boot-nand]# nand write 50000000 0 40000

[u-boot-nand]# reset

把 linux 内核写到 nand

[u-boot-nand]# set serverip 192.168.1.253

[u-boot-nand]# sav

[u-boot-nand]# tftp 50000000 zImage

[u-boot-nand]# nand erase 40000 300000

[u-boot-nand]# nand write 50000000 40000 300000

测试：

[u-boot-nand]# nand read 50008000 40000 300000

[u-boot-nand]# bootm 50008000

启动成功会出现小企鹅

把跟文件系统写到 nand

[u-boot-nand]# set bootargs root=/dev/nfs nfsroot=192.168.,1.253:/nfsroot ip=192.168.1.20 console=ttySAC0,115200

[u-boot-nand]# sav

[u-boot-nand]# nand read 50008000 40000 300000

[u-boot-nand]# bootm 50008000

启动成功后：

[root@uplooking]# mount /dev/mtdblock2 /mnt

[root]# cp <path>/rootfs/rootfs20101220.tar /nfsroot

[root@uplooking]# tar -xvf rootfs20101220.tar -C /mnt

重启开发板

[u-boot-nand]# set bootargs root=/dev/mtdbloc2 consloe=ttySAC0,115200

[u-boot-nand]# sav

[u-boot-nand]# nand read 50008000 40000 300000

[u-boot-nand]# bootm 50008000

设置自动启动

[u- boot-nand]# set bootcmd “nand read 50008000 40000 300000;bootm 50008000”

[u-boot-nand]# sav

[u-boot-nand]# reset

四、编译 u-boot

编译 SD 卡启动的 uboot :

hero@Beyond:~/smdk6410_lzy/src$ tar -jxvf u-boot-1.1.6.tar.bz2

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ make smdk6410_config

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ vim include/configs/smdk6410.h

205 #define CFG_PROMPT "[zhangsan@sd]# "

188 #define CONFIG_SERVERIP 192.168.1.253

183 #define CONFIG_BOOTARGS "root=/dev/nfs nfsroot=192.168.1.253:/home/hero/nfsroot/rootfs_qt ip=192.168.1.20 console=ttySAC0,115200"

445 /* Boot configuration (define only one of next) */

446 //#define CONFIG_BOOT_NOR

447 //#define CONFIG_BOOT_NAND

448 #define CONFIG_BOOT_MOVINAND

449 //#define CONFIG_BOOT_ONENAND

450 //#define CONFIG_BOOT_ONENAND_IROM

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ vim Makefile

161 CROSS_COMPILE = arm-linux-

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ make

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ ./mkmovi

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ mkdir /home/hero/student/20130509/uboot

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ cp u-boot-movi.bin !$

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ cp System.map /home/hero/student/20130509/uboot/System_movi.map

编译完成后用 write_sd 把 u-boot-movi.bin 烧写到 SD 卡测试

uboot 源码→u-boot（ELF）→u-boot.bin(192K)

./mkmovi u-boot.bin→u-boot-movi.bin

uboot = u-boot.bin + u-boot.bin

Uboot256K = uboot(前 256K)

Uboot8K = uboot(前 8K)

u-boot-movi.bin = Uboot256K + Uboot8K

Write_sd u-boot-movi.bin→SD 卡

0[------|BL2256K|EVN16K|BL18K|1K]2G-1

编译 NAND 启动的 uboot（在 SD 卡启动的基础上修改）

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ vim include/configs/smdk6410.h

445 /* Boot configuration (define only one of next) */

446 //#define CONFIG_BOOT_NOR

447 #define CONFIG_BOOT_NAND

448 //#define CONFIG_BOOT_MOVINAND

449 //#define CONFIG_BOOT_ONENAND

450 //#define CONFIG_BOOT_ONENAND_IROM

205 #define CFG_PROMPT "[zhangsan@nand]# "

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ make

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ ./mknand

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ cp u-boot-nand.bin /home/hero/student/20130509/uboot/

hero@Beyond:~/smdk6410_lzy/src/u-boot-1.1.6_smdk6410$ cp System.map /home/hero/student/20130509/uboot/System_nand.map

./mknand:mv u-boot.bin u-boot-nand.bin

五、编译 linux 内核

编译 2.6.28 内核

hero@Beyond:~/smdk6410_lzy/src$ tar -jxvf linux-2.6.28.tar.bz2

hero@Beyond:~/smdk6410_lzy/src$ cd linux-2.6.28_smdk6410/

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ vim .cross_compile

arm-linux-

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ cp smdk6410_config .config

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ make menuconfig

Device Drivers →

Graphics support --->

<*> Support for frame buffer devices →

Select LCD Type (UT_LCD43C_D 480*272) --->

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ cd drivers/video/samsung/

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410/drivers/video/samsung$ mv s3cfb_UT_LCD43C_D\ .c s3cfb_UT_LCD43C_D.c

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410/drivers/video/samsung$ cd ../../../

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ make

hero@Beyond:~/smdk6410_lzy/src/linux-2.6.28_smdk6410$ cp arch/arm/boot/zImage /tftpboot

测试：

[u-boot-nand/sd]# tftp 50008000 zImage

[u-boot-nand/sd]# bootm 50008000

编译 3.4.24 内核

hero@Beyond:~/smdk6410_lzy/src$ tar -jxvf linux3.4.24_ok.tar.bz2

hero@Beyond:~/smdk6410_lzy/src$ cd linux3.4.24_ok/

hero@Beyond:~/smdk6410_lzy/src/linux3.4.24_ok$ make -j4

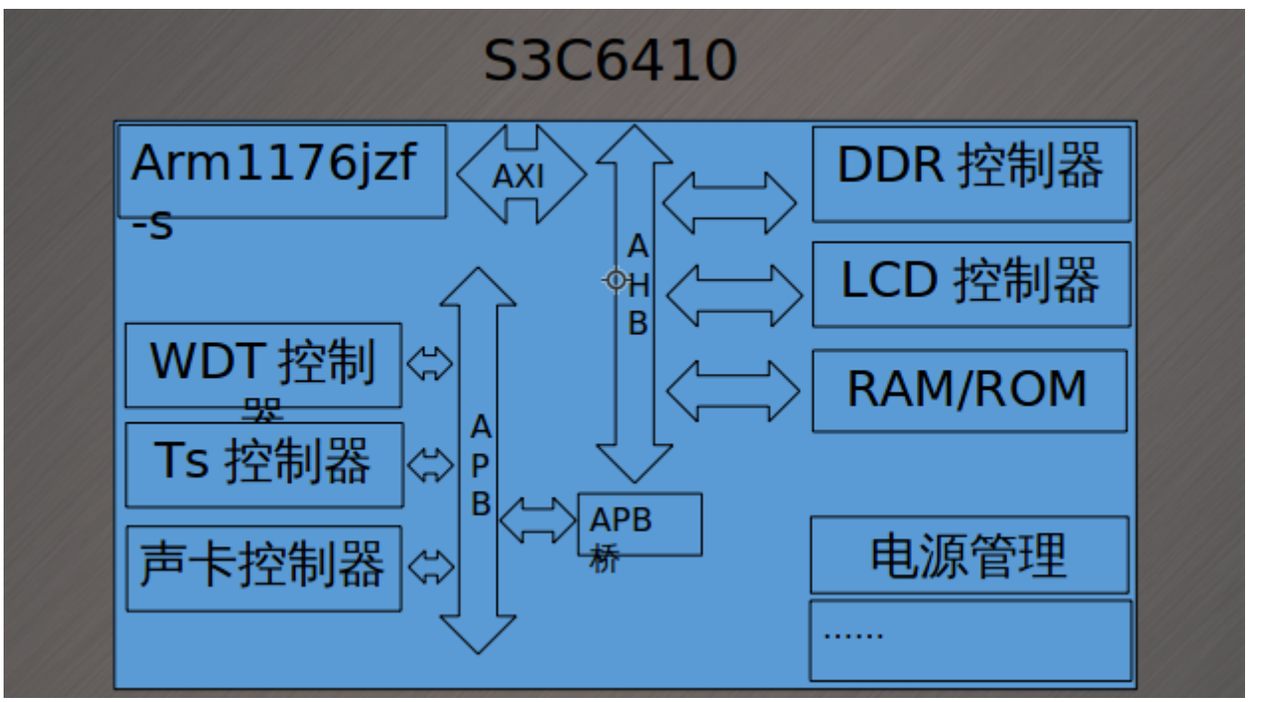
总结：

嵌入式 = hw + uboot + zImage + rootfs + filesystem

Arm→cpu

x86:cpu 南桥 北桥

SOC:cpu 控制器 内存 rom 电源管理模块



开发阶段：uboot-sd zImage-pc-tftp rootfs-nfs

发布：uboot-nand zImage-nand rootfs-nand

Uboot-sd zImage-sd rootfs-sd

Write_sd

[u-boot-sd]# nand xxx tftp

Write_nand

编译 uboot zImage

六、做根文件系统

脚本 profile rcS　命令 ls export　proc sys

Busybox

hero@Beyond:~/smdk6410_lzy/src$ tar -jxvf busybox-1.20.1.tar.bz2

hero@Beyond:~/smdk6410_lzy/src$ cd busybox-1.20.1

hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ vim Makefile

164 CROSS_COMPILE ?=arm-linux-

hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ make defconfig

hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ make menuconfig

Busybox Settings　-→

Build Options　-→

[*] Build BusyBox as a static binary (no shared libs)

Linux Module Utilities　-→

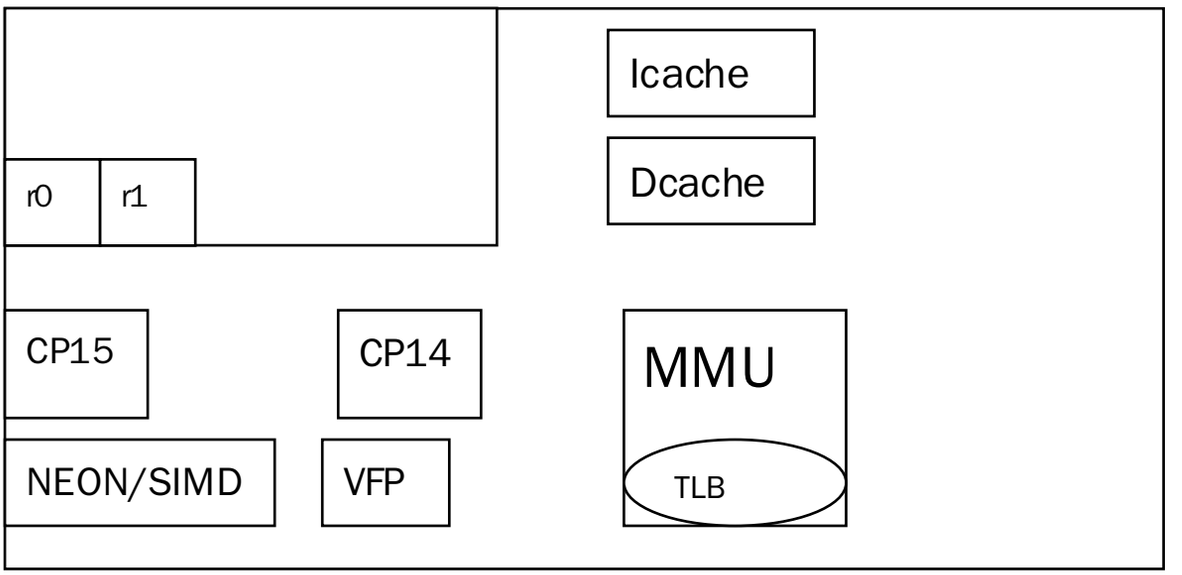
[*] modinfo

[ ] Simplified modutils

```
        [*]     insmod
        [*]     rmmod
        [*]     lsmod
        [ ]       Pretty output (NEW)
        [*]     modprobe
        [ ]       Blacklist support (NEW)
        [*]     depmod
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ make
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ make
install
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ cd /
hero@Beyond:/$ mv nfsroot nfsroot_bac
hero@Beyond:/$ chmod 777 nfsroot
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1/_install$
cp * /nfsroot -rf
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1/_install$
cd /nfsroot
hero@Beyond:/nfsroot$ mkdir dev
hero@Beyond:/nfsroott$ mkdir proc
hero@Beyond:/nfsroot$ mkdir sys
hero@Beyond:/nfsroot$ mkdir mnt
hero@Beyond:/nfsroot$ mkdir tmp
```

```
hero@Beyond:/nfsroot$ mkdir root
hero@Beyond:/nfsroot$ cp
/usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/lib/ . -rf
hero@Beyond:~/nfsroot/nfsroot_test$ cd
/home/hero/smdk6410_lzy/src/busybox-1.20.1/
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1$ cd
examples/
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1/example
s$ cd bootfloppy/
hero@Beyond:~/smdk6410_lzy/src/busybox-1.20.1/example
s/bootfloppy$ cp etc/ /nfroot -rf
hero@Beyond:/nfsroot$ cd /dev/
hero@Beyond:/nfsroot/dev$ mknod console c 5 1
hero@Beyond:/nfsroot/dev$ mknod tty2 c 4 2
hero@Beyond:/nfsroot$ vim etc/fstab
  proc            /proc     proc       defaults       0     0
  sysfs           /sys      sysfs      defaults       0     0
  tmpfs           /tmp      tmpfs      defaults       0     0
hero@Beyond:~/nfsroot/nfsroot_test$ vim etc/profile
  PS1="zhangsan@uplooking \w# "
  #利用 mdev 自动创建设备文件
  mdev -s
```

七、arm 体系结构



arm 架构 P43

User system svc

R13 Sp

R14 Lr

R15 Pc

R9 SB

CPSR 当前程序状态寄存器 if(a > b){} else{}

SPSR 每一个异常模式

qq--->user msg→net→kernel→qq

Cpsr irq-->IRQ-->CPSR

CPSR→ SPSR_irq

Swi/svc:user→svc

特权模式之间的切换：CPSR[4:0]

CPSR：N C Z V Q　J A I F T E M[4:0]　　arm 架构手册 P49

汇编文件　*.s

内嵌汇编(内联汇编)

在 C 语言里写汇编　*.c

语法

```
asm volatile (
        “xxxxxxxx\n”
        “xxxxxxxx\n”
);
Int   a ;
__asm__ __volatile__(
   “xxx”
 : 声明输出变量 “=r”(a)
 : 声明输入变量 “r”(a)
 : 保护使用过或影响过的寄存器，保护内存
);
```

裸汇编

位置相关 运行地址必须==链接地址

全局变量

ldr r0,=addr ldr r1，[r0]

位置无关 运行地址可以！=链接地址

链接地址 在程序的链接阶段指定,是整个程序的入口地址

立即数传递规则

指令编码(mov b 跳转范围)

八、GPIO

工程：driver.c driver.h main.c start.s Makefile ld.lds

调试代码的两种方式：

Tftp 50000000 arm.bin

Go 50000000

优点：可以利用 uboot 已经实现的函数，如 printf

System.map

Tftp 50000000 arm.bin

Nand erase 0 40000

Nand write 50000000 0 40000

切换为 nand 启动，上电

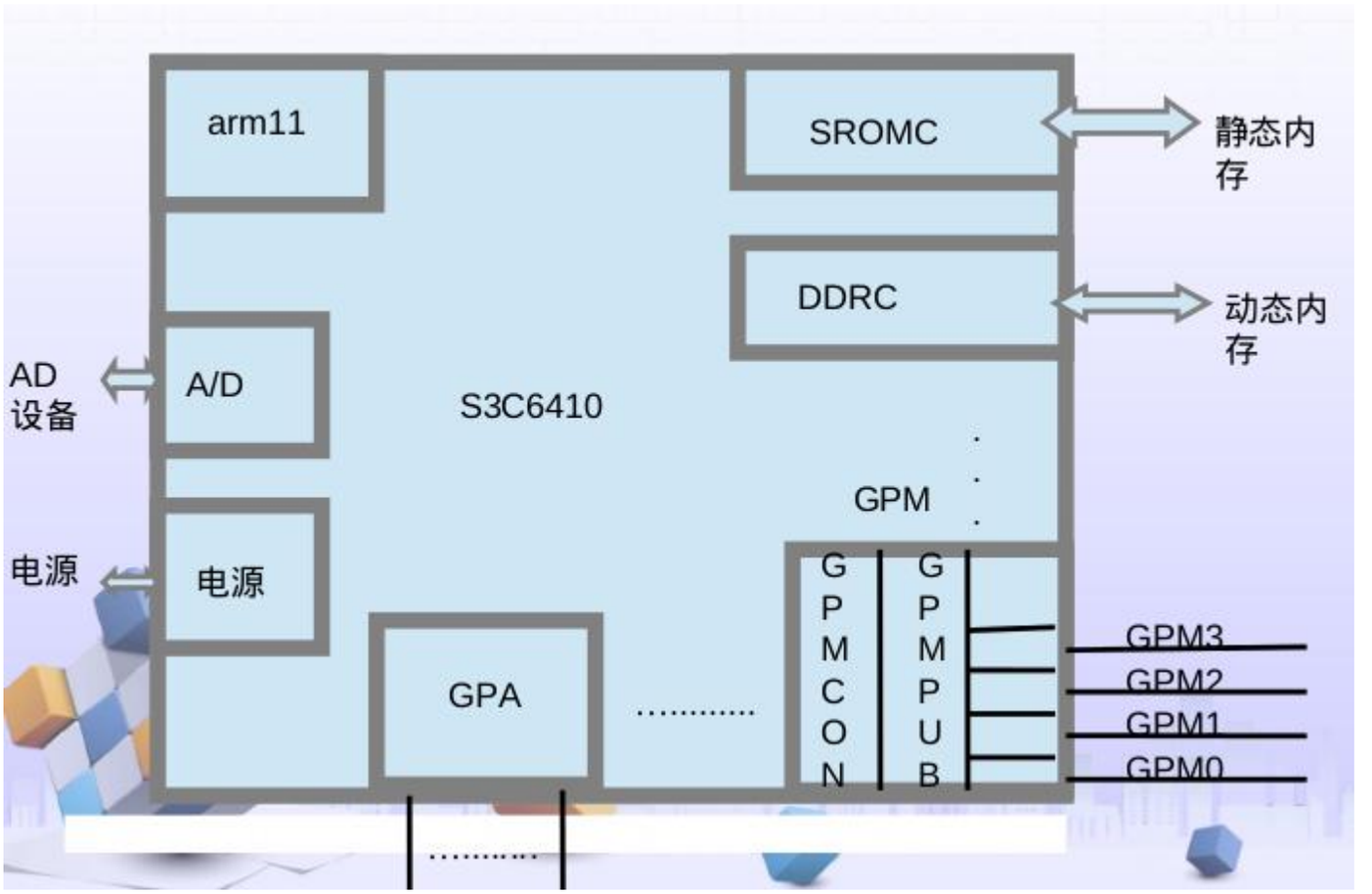
2146 arm-linux-gcc -c -o main.o main.c

2147 arm-linux-gcc -c -o driver.o driver.c

2148 arm-linux-as start.s -o start.o

2151 arm-linux-ld start.o driver.o main.o -o arm -Ttext 0x50000000

2161 arm-linux-objcopy -O binary arm arm.bin



九、arm 统一编址

X86 内存和外设是分别编址

Arm 内存外设统一编址

Arm9/11 通过cp15指定外设地址	访问外设
	访问内存

Arm-cotex-A9	访问内存和外设

查看 6410 手册 MMAP 章节

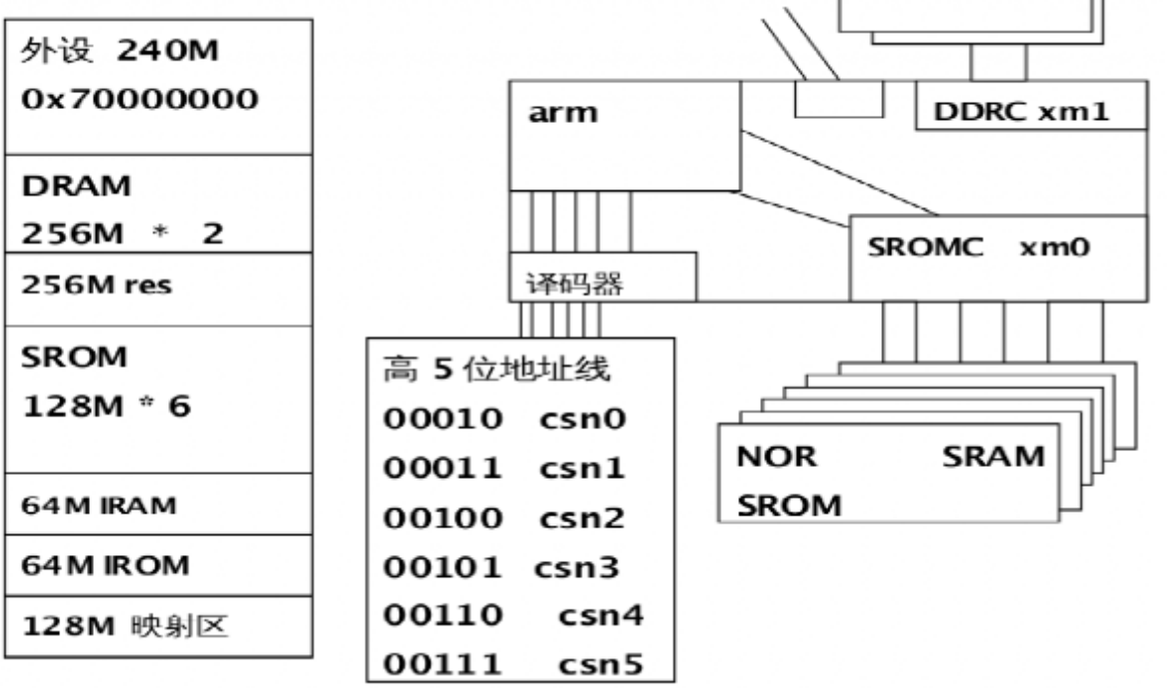
Table 2-2. Device Specific Address Space

Address		Size(MB)	Description	Note
0x0000_0000	0x07FF_FFFF	128MB	Booting Device Region by XOM Setting	Mirrored Region
0x0800_0000	0x0BFF_FFFF	64MB	Internal ROM	
0x0C00_0000	0x0FFF_FFFF	64MB	Stepping Stone (Boot Loader)	
0x1000_0000	0x17FF_FFFF	128MB	SROMC Bank0	
0x1800_0000	0x1FFF_FFFF	128MB	SROMC Bank 1	
0x2000_0000	0x27FF_FFFF	128MB	SROMC Bank 2	
0x2800_0000	0x2FFF_FFFF	128MB	SROMC Bank 3	
0x3000_0000	0x37FF_FFFF	128MB	SROMC Bank 4	
0x3800_0000	0x3FFF_FFFF	128MB	SROMC Bank 5	
0x4000_0000	0x47FF_FFFF	128MB	Reserved	
0x4800_0000	0x4FFF_FFFF	128MB		
0x5000_0000	0x5FFF_FFFF	256MB	DRAM Controller of the Memory Port1	
0x6000_0000	0x6FFF_FFFF	256MB		

内存：ddr　sram nor srom iram(8K) irom(32K)

外设：ddrc sromc 等等 device memory

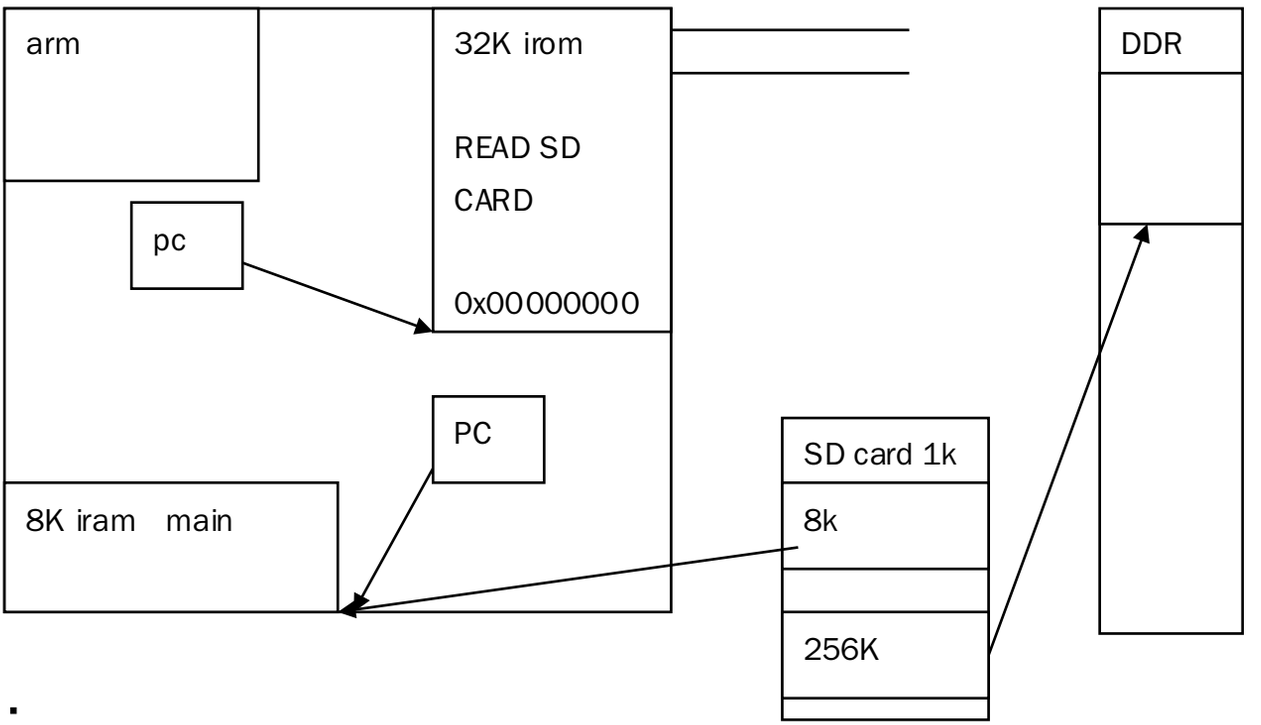
6410 手册第二章 P116



十、S3C6410 启动方式：

SD

Write_sd [xxxxxxx|BL2256K|ENV16K|BL18K||1K]



bl1:

1.初始化 sp

2.映射外设端口

3.关闭看门口

4.Clock 12M→533M

5.Ddr

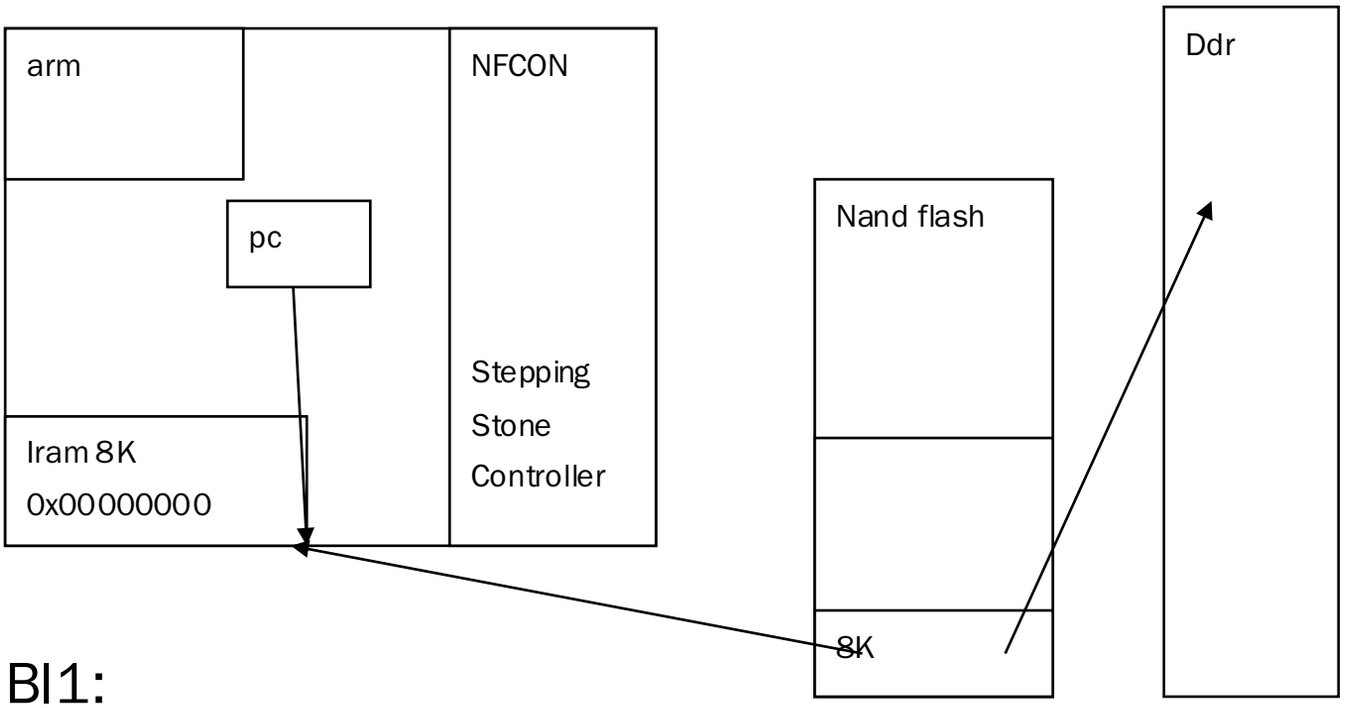
6.Movi_read(BL2256K→ddr(0x57e00000))

7.跳转 ldr pc, =main

8.Uart

9.Shell-------->cmd:bootm tftp nandxxx

NAND



Bl1:

1.初始化 sp

2.映射外设端口

3.关闭看门狗

4.Clock

5.Ddr

6.Nand

7.nand_read(256K---->ddr(0x57e00000))

8.Ldr pc, =main

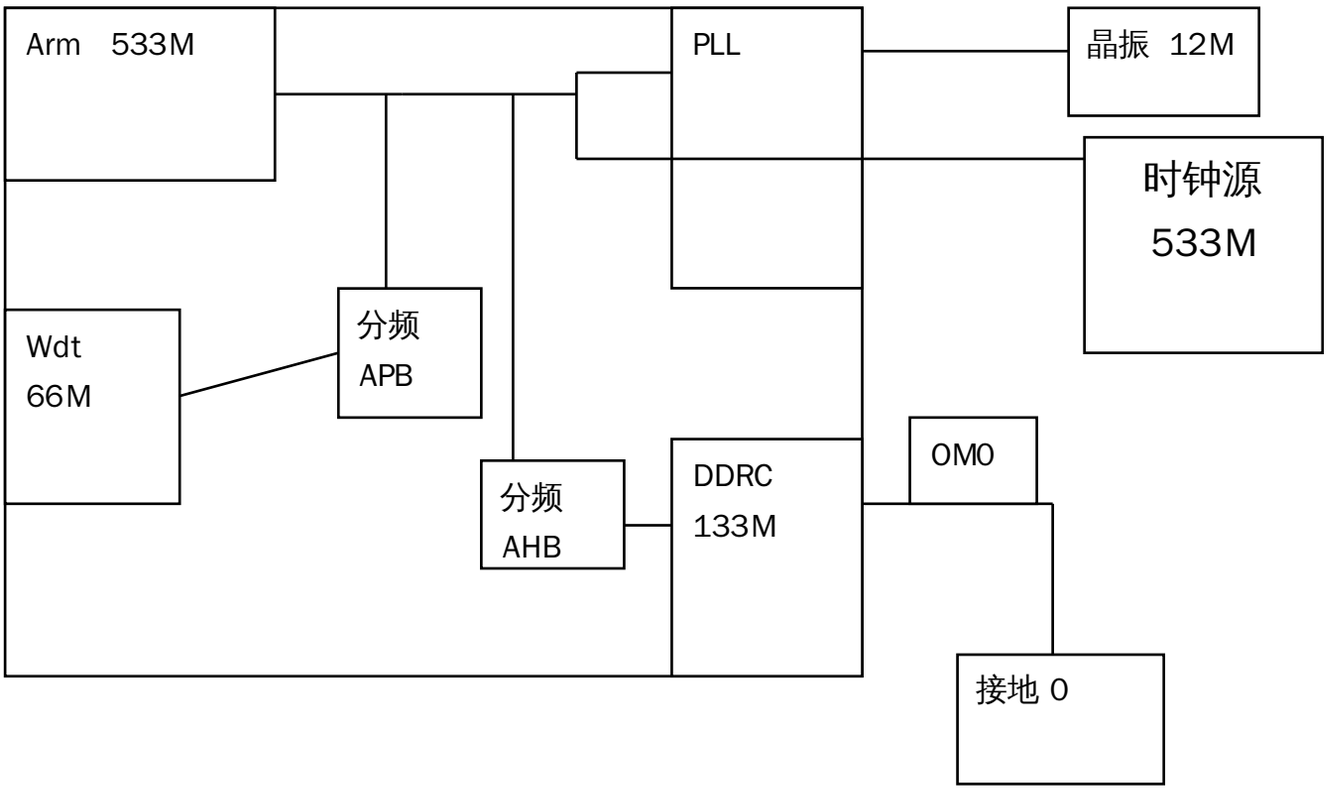
9.Shell---->cmd

十二、arm 裸板开发

Sp　映射外设端口　wdt

注意：如果 nand 启动，链接地址应该是 0

十三、clock



Armclk 533M

Pclk 66M

Hclk 133M

查看 6410 手册第三章 P125

6410 一共有 3 个 PLL

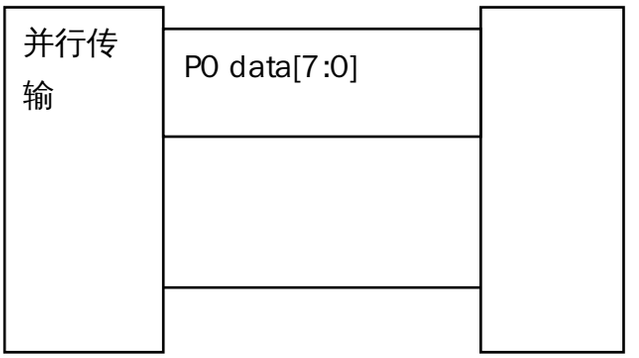
APLL arm

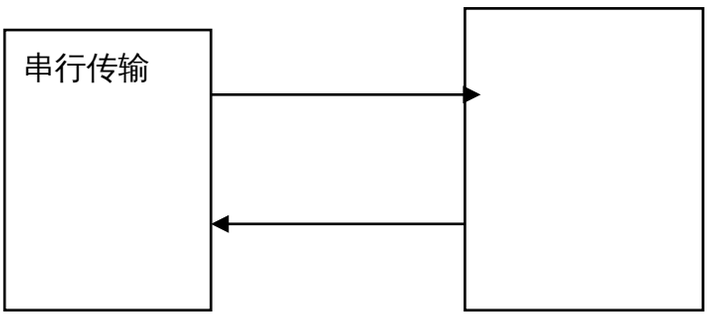
MPLL ahb apb

EPLL 其他 usb 2d 3d

十四、串口 UART

Minicom-→pc(uart)----development board(uart)





TTL/CMOS	RS232
3.3/5---->1	-3V — -12V--->1
0/0 ------->0	3V---------12V—→0

查看电路图：串口和 GPA0&GPA1 复用管脚

6410 包含四个串口 com0-3

115200 8N1

十五、如何在内核中寻找方法

mv linux3.4.24_ok llinux

cd llinux/

cd arch/

mv arm/ ../

rm * -rf

mv ../arm/ .

cd arm/

mv mach-s3c64xx/ ../

mv plat-samsung/ ../

rm mach-* -rf

rm plat-* -rf

mv ../mach-s3c64xx/ .

mv ../plat-samsung/ .

cd ..

cd ..

ctags -R .

Vim ~/.vimrc

Set tags=<tags_path>/tags

Vim -t xxxx

十六、DDR

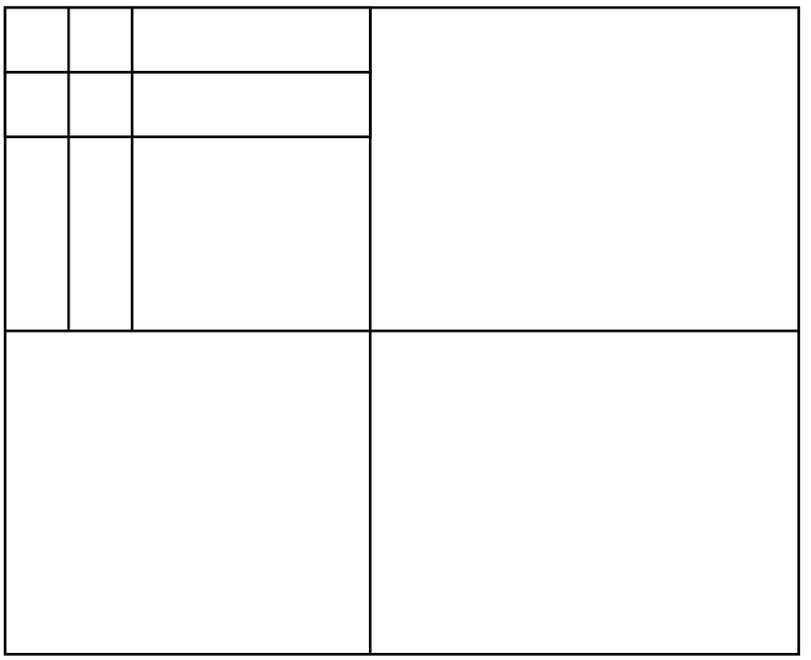
Dram Sram

Mobile DDR SDRAM

128M = 64M + 64M

K4X5xxxxxx 芯片 32MX16

用两块 16 位芯片按照位扩展的方式扩展为 32 位



5.4.3 DDR/MOBILE DDR SDRAM INITIALIZATION SEQUENCE

- Program mem_cmd in direct_cmd to '2'b10', which makes DRAM Controller issue 'NOP' memory command.
- Program mem_cmd in direct_cmd to '2'b00', which makes DRAM Controller issue 'Prechargeall' memory command.
- Program mem_cmd in direct_cmd to '2'b11', which makes DRAM Controller issue 'Autorefresh' memory command.
- Program mem_cmd in direct_cmd to '2'b11', which makes DRAM Controller issue 'Autorefresh' memory command.
- Program mem_cmd to '2'b10' in direct_cmd, which makes DRAM Controller issue 'MRS' memory command
  - Bank address for EMRS must be set.
- Program mem_cmd to '2'b10' in direct_cmd, which makes DRAM Controller issue 'MRS' memory command.

配置内存控制器一般步骤：

1.配置端口（GPIO 或者 其他复用）

2.让控制器进入配置模式

3.配置属性（内存大小，内存位宽等）

4.配置时序（控制器和内存芯片之间的硬件通信协议）

5.发送 cmd 到内存芯片(设置突发长度等)

6.让控制器进入运行模式

7.检测是否运行成功

十七、NAND　K9F2G08

NAND 是什么（和 NOR 区别）

NOR 三总线

NAND 电路

引脚 功能

链接 有没有复用 GPOO　GPP3-7

NAND 内部结构(算大小， 页， 块)

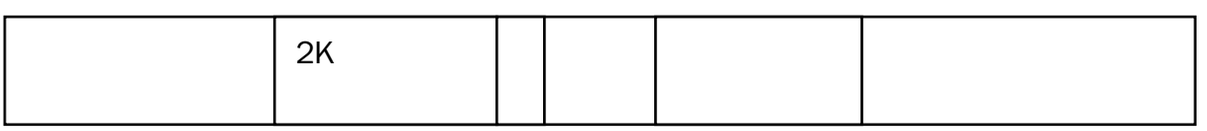
Device = 2K blocks

Block = 64 page

Page = 2K + 64 byte(ECC 坏块标记)

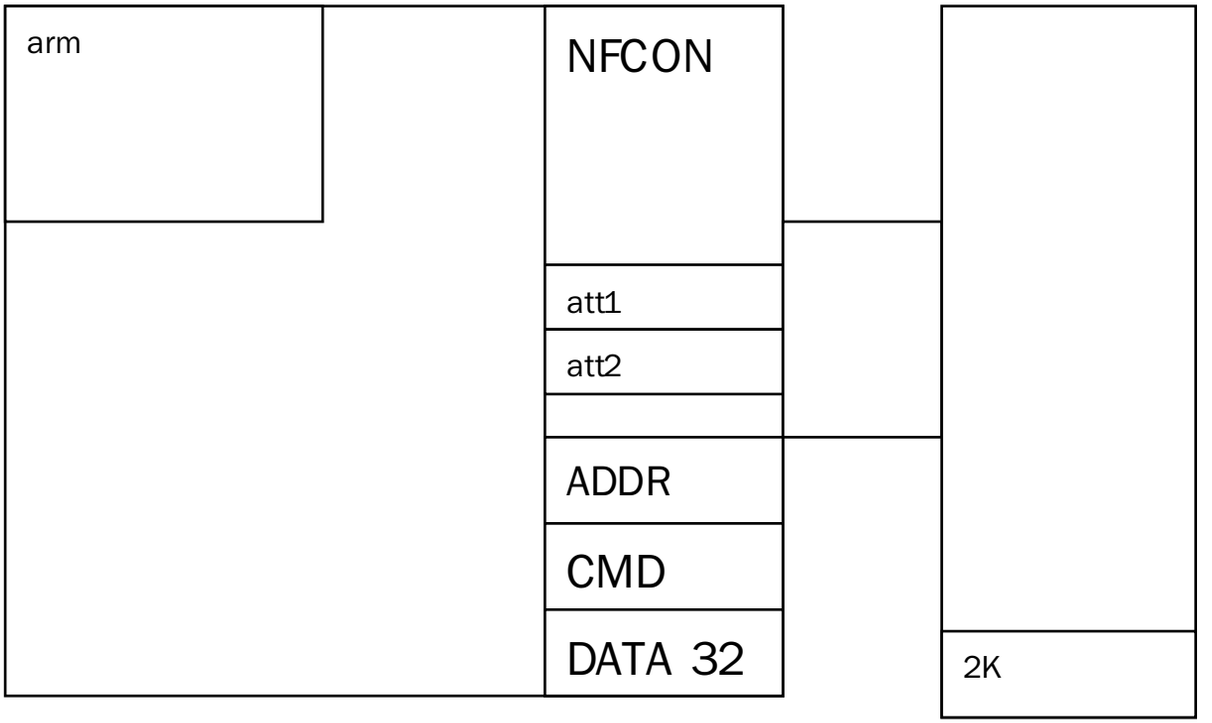
Erase 必须以块为单位 Set

Write 和 read 一般来说以页为单位（一次最大不大于一个页）



NAND 访问方式

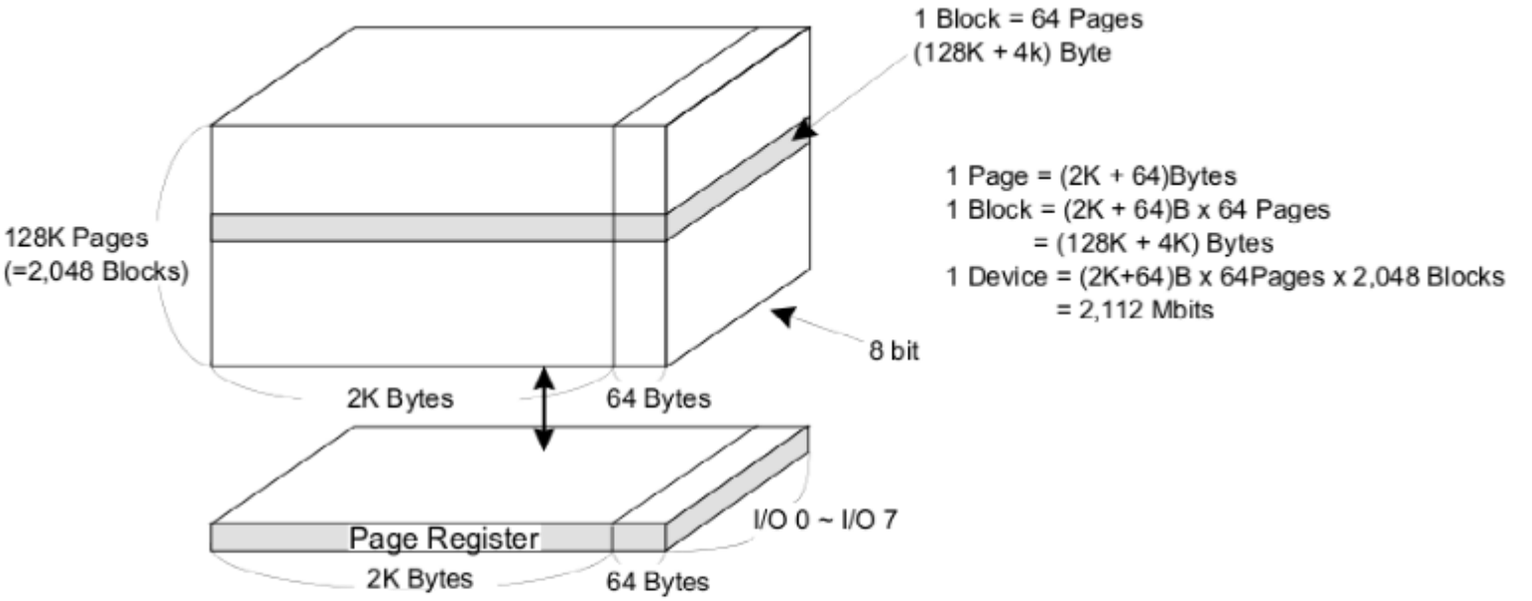
发送地址



28 [27-17][16-11][10-0]

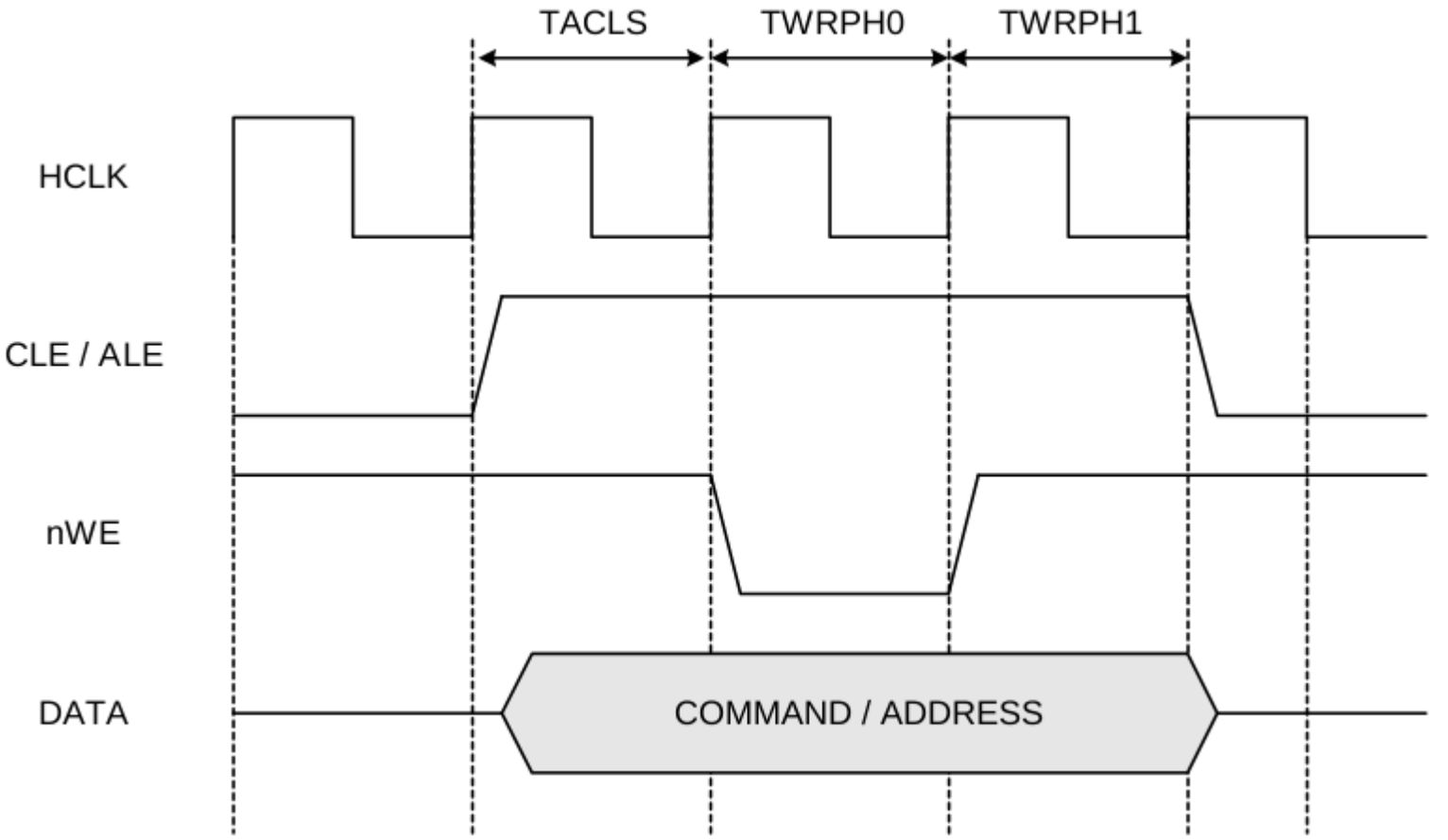
PRODUCT LIST

Part Number	Vcc Range	Organization	PKG Type
K9F2G08R0A-J	1.65 ~ 1.95V	X8	FBGA
K9F2G08U0A-P	2.70 ~ 3.60V		TSOP1
K9F2G08U0A-I			52ULGA

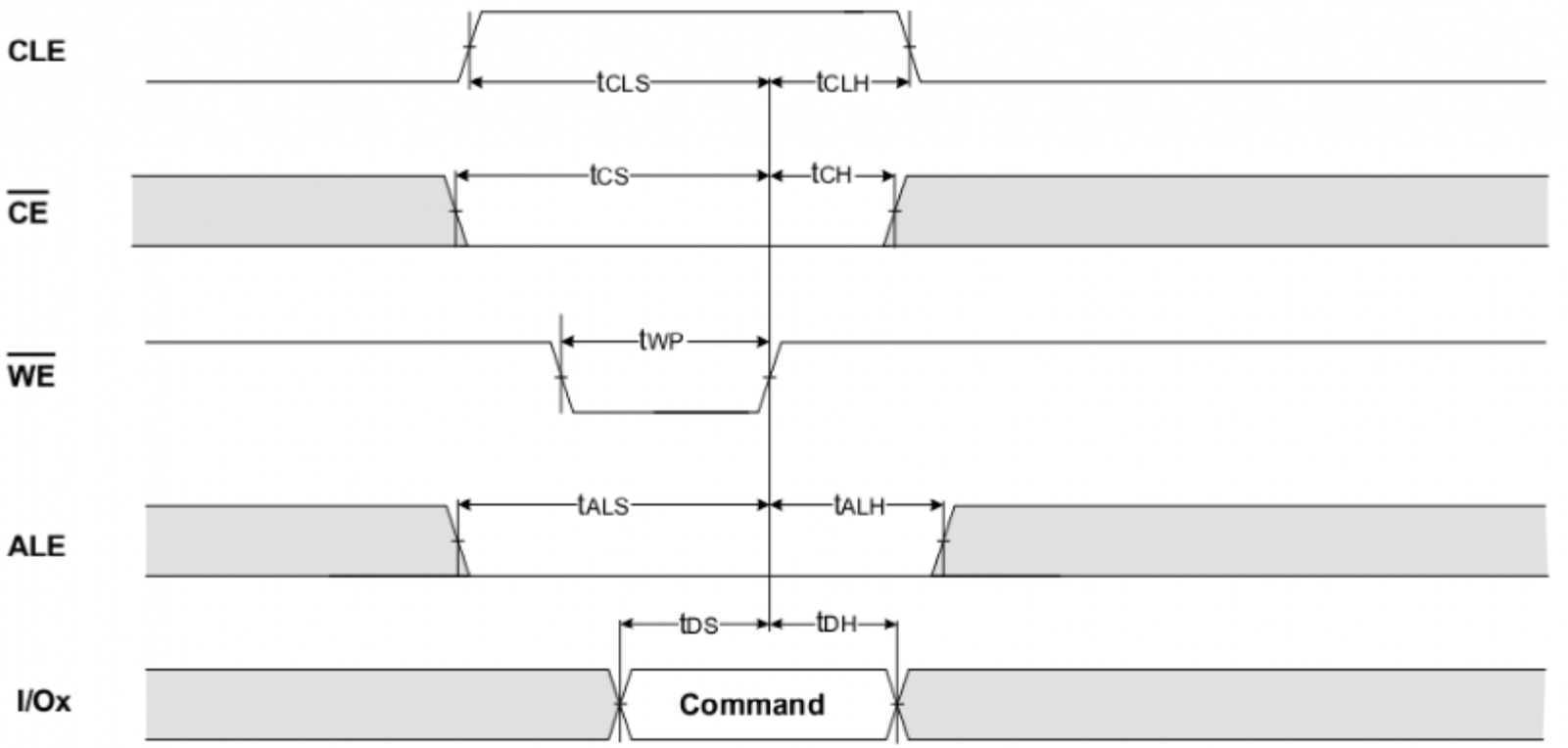


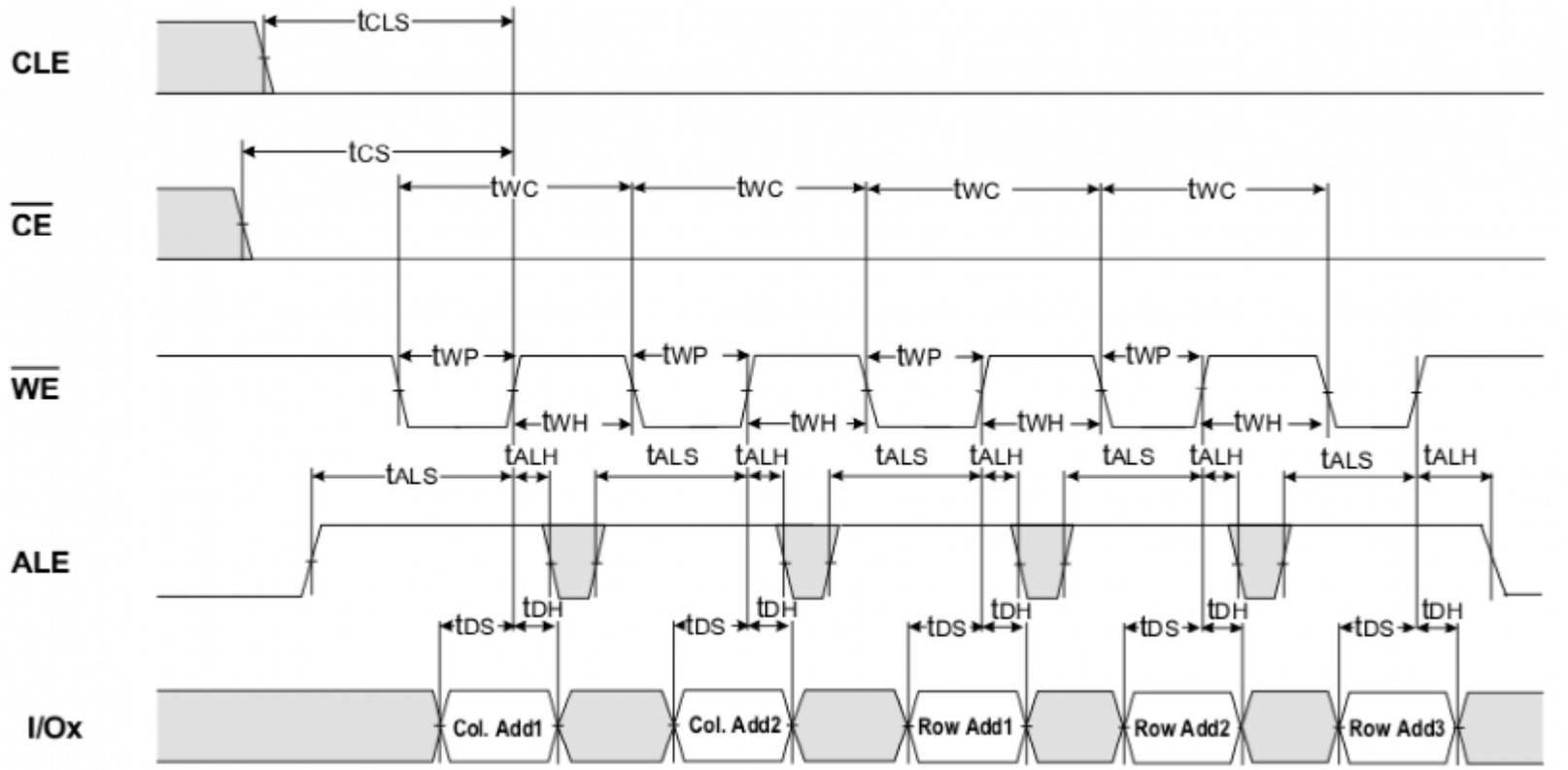
	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7	
1st Cycle	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	**Column Address**
2nd Cycle	A_8	A_9	A_{10}	A_{11}	*L	*L	*L	*L	**Column Address**
3rd Cycle	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	**Row Address**
4th Cycle	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	**Row Address**
5th Cycle	A_{28}	*L	*L	*L	*L	*L	*L	*L	**Row Address**

	MP0_CS_CFG						
	[5]	[4]	[3]	[2]	[1]	[0]	
Xm0CSn[0]	-	-	-	-	-	-	SROMC CS0
Xm0CSn[1]	-	-	-	-	-	-	SROMC CS1
Xm0CSn[2]	-	-	-	-	1	-	SROMC CS2
	-	-	-	-	0	-	OneNANDC CS0
	-	-	-	-	0	-	NFCON CS0
Xm0CSn[3]	-	-	1	-	-	-	SROMC CS3
	-	-	0	-	-	-	OneNANDC CS1
	-	-	0	-	-	-	NFCON CS1
Xm0CSn[4]	-	0	-	-	-	-	SROMC CS4
	-	1	-	-	-	-	CFCON CS0
Xm0CSn[5]	0	-	-	-	-	-	SROMC CS5
	1	-	-	-	-	-	CFCON CS1



Command Latch Cycle





AC Timing Characteristics for Command / Address / Data Input

Parameter	Symbol	Min		
		1.8V	3.3V	1
CLE Setup Time	t_{CLS}(1)	21	12	
CLE Hold Time	t_{CLH}	5	5	
$\overline{CE}$ Setup Time	t_{CS}(1)	25	20	
$\overline{CE}$ Hold Time	t_{CH}	5	5	
$\overline{WE}$ Pulse Width	t_{WP}	21	12	
ALE Setup Time	t_{ALS}(1)	21	12	
ALE Hold Time	t_{ALH}	5	5	
Data Setup Time	t_{DS}(1)	20	12	
Data Hold Time	t_{DH}	5	5	
Write Cycle Time	t_{WC}	42	25	
$\overline{WE}$ High Hold Time	t_{WH}	15	10	
Address to Data Loading Time	t_{ADL}(2)	100	100	

Nand_init

Nand_read

Nand_write

Nand_erase

Nand_id

十八、异常

arm 处理器的工作模式

Modes						
	Privileged modes					
		Exception modes				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

异常类型

Table A2-4 Exception processing modes

Exception type	Mode	VE[a]	Normal address	High vector address
Reset	Supervisor		0x00000000	0xFFFF0000
Undefined instructions	Undefined		0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor		0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort		0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort		0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0	0x00000018	0xFFFF0018
		1	IMPLEMENTATION DEFINED	
FIQ (fast interrupt)	FIQ	0	0x0000001C	0xFFFF001C
		1	IMPLEMENTATION DEFINED	

a. VE = vectored interrupt enable (CP15 control); RAZ when not implemented.

各种异常详解：

reset

```
R14_svc   = UNPREDICTABLE value
SPSR_svc  = UNPREDICTABLE value
CPSR[4:0] = 0b10011              /* Enter Supervisor mode */
CPSR[5]   = 0                    /* Execute in ARM state */
CPSR[6]   = 1                    /* Disable fast interrupts */
CPSR[7]   = 1                    /* Disable normal interrupts */
CPSR[8]   = 1                    /* Disable Imprecise Aborts (v6 only) */
CPSR[9]   = CP15_reg1_EEbit      /* Endianness on exception entry */
if high vectors configured then
    PC    = 0xFFFF0000
else
    PC    = 0x00000000
```

Unde

```
R14_und   = address of next instruction after the Undefined instruction
SPSR_und  = CPSR
CPSR[4:0] = 0b11011              /* Enter Undefined Instruction mode */
CPSR[5]   = 0                    /* Execute in ARM state */
                                 /* CPSR[6] is unchanged */
CPSR[7]   = 1                    /* Disable normal interrupts */
                                 /* CPSR[8] is unchanged */
CPSR[9]   = CP15_reg1_EEbit      /* Endianness on exception entry */
if high vectors configured then
    PC    = 0xFFFF0004
else
    PC    = 0x00000004
```

To return after emulating the Undefined instruction use:

```
MOVS PC,R14
```

svc/swi

```
R14_svc   = address of next instruction after the SWI instruction
SPSR_svc  = CPSR
CPSR[4:0] = 0b10011              /* Enter Supervisor mode */
CPSR[5]   = 0                    /* Execute in ARM state */
                                 /* CPSR[6] is unchanged */
CPSR[7]   = 1                    /* Disable normal interrupts */
                                 /* CPSR[8] is unchanged */
CPSR[9]   = CP15_reg1_EEbit      /* Endianness on exception entry */
if high vectors configured then
    PC    = 0xFFFF0008
else
    PC    = 0x00000008
```

To return after performing the SWI operation, use the following instruction to resto
(from R14_svc) and CPSR (from SPSR_svc) and return to the instruction following

```
MOVS PC,R14
```

Irq

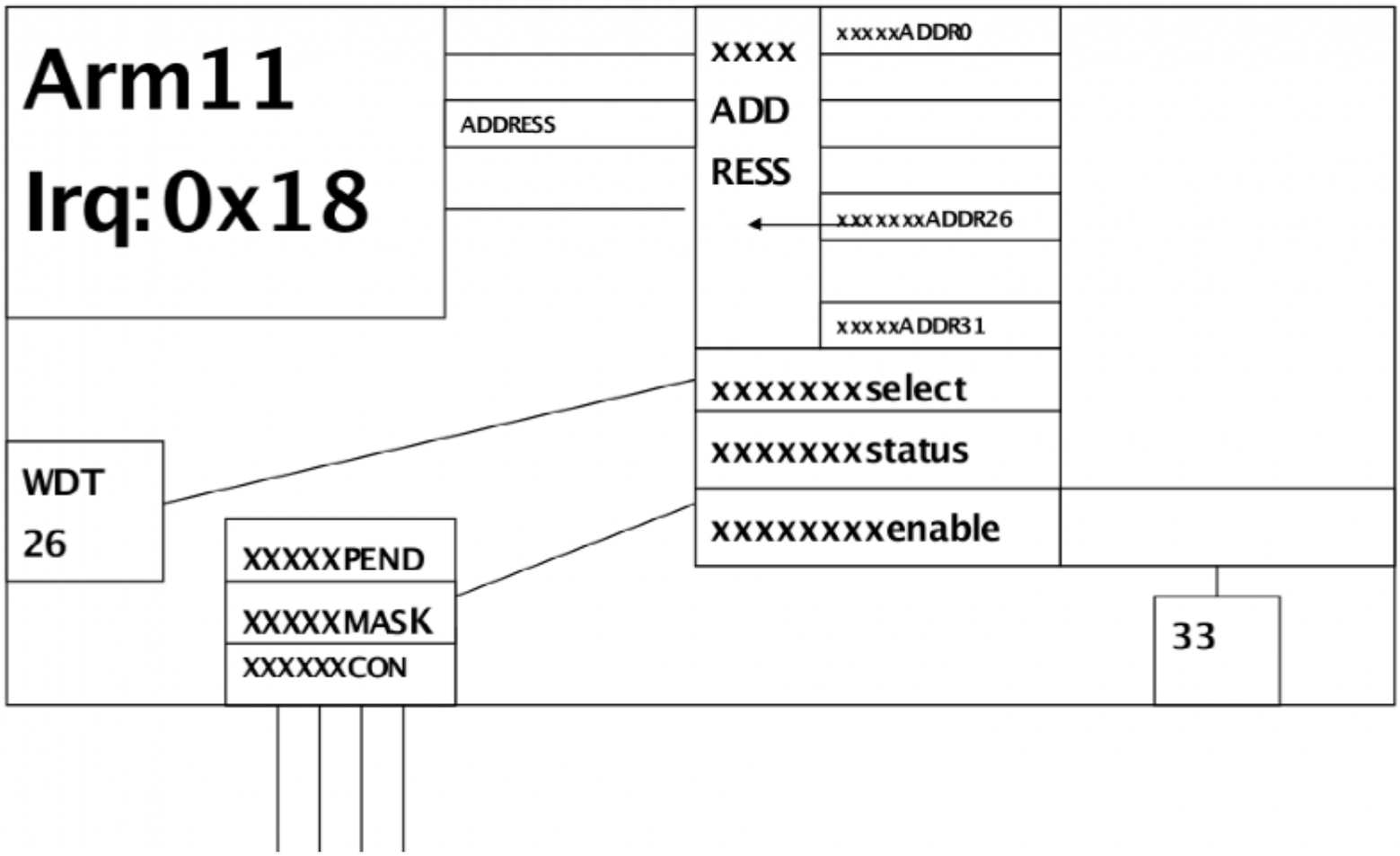
```
R14_irq   = address of next instruction to be executed + 4
SPSR_irq  = CPSR
CPSR[4:0] = 0b10010              /* Enter IRQ mode */
CPSR[5]   = 0                    /* Execute in ARM state */
                                 /* CPSR[6] is unchanged */
CPSR[7]   = 1                    /* Disable normal interrupts */
CPSR[8]   = 1                    /* Disable Imprecise Data Aborts (v6 only) */
CPSR[9]   = CP15_reg1_EEbit      /* Endianness on exception entry */
if VE==0 then
  if high vectors configured then
      PC    = 0xFFFF0018
  else
      PC    = 0x00000018
else
  PC = IMPLEMENTATION DEFINED    /* see page A2-26 */
```

To return after servicing the interrupt, use:

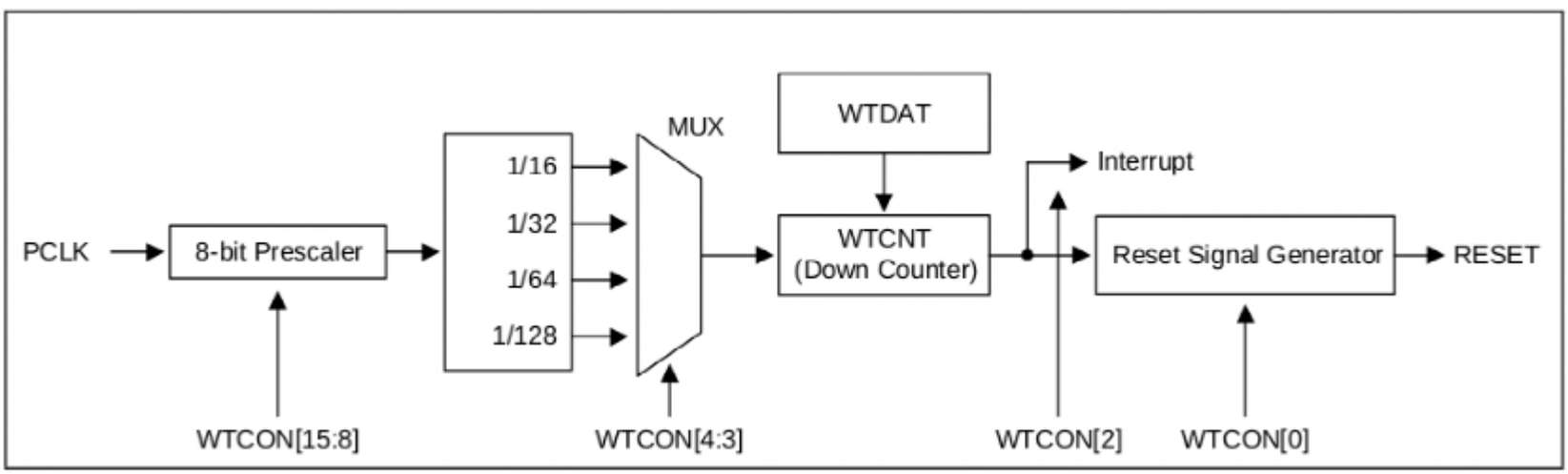
```
SUBS PC,R14,#4
```

6410 中断控制器

中断源 6410 手册 P410



十九、WDT



二十、外部中断

6410 外部的设备不能直接和中断控制器链接

所以外部的设备想产生中断必须要借助于中断源(在 6410 内部)

424/187/127

127 外部中断分为 10 组 0-9

0-27 是第 0 组

第 0 组对应 4 个中断源 第 1-9 组对应同一个 53 号

0-3 中断源 0

4-11 中断源 1

12-19 中断源 32

20-27 中断源 33

CON 设置要检测的有效电平

MASK 屏蔽某个外部中断

PEND 标记是哪个外部中断发生 可以用来清中断

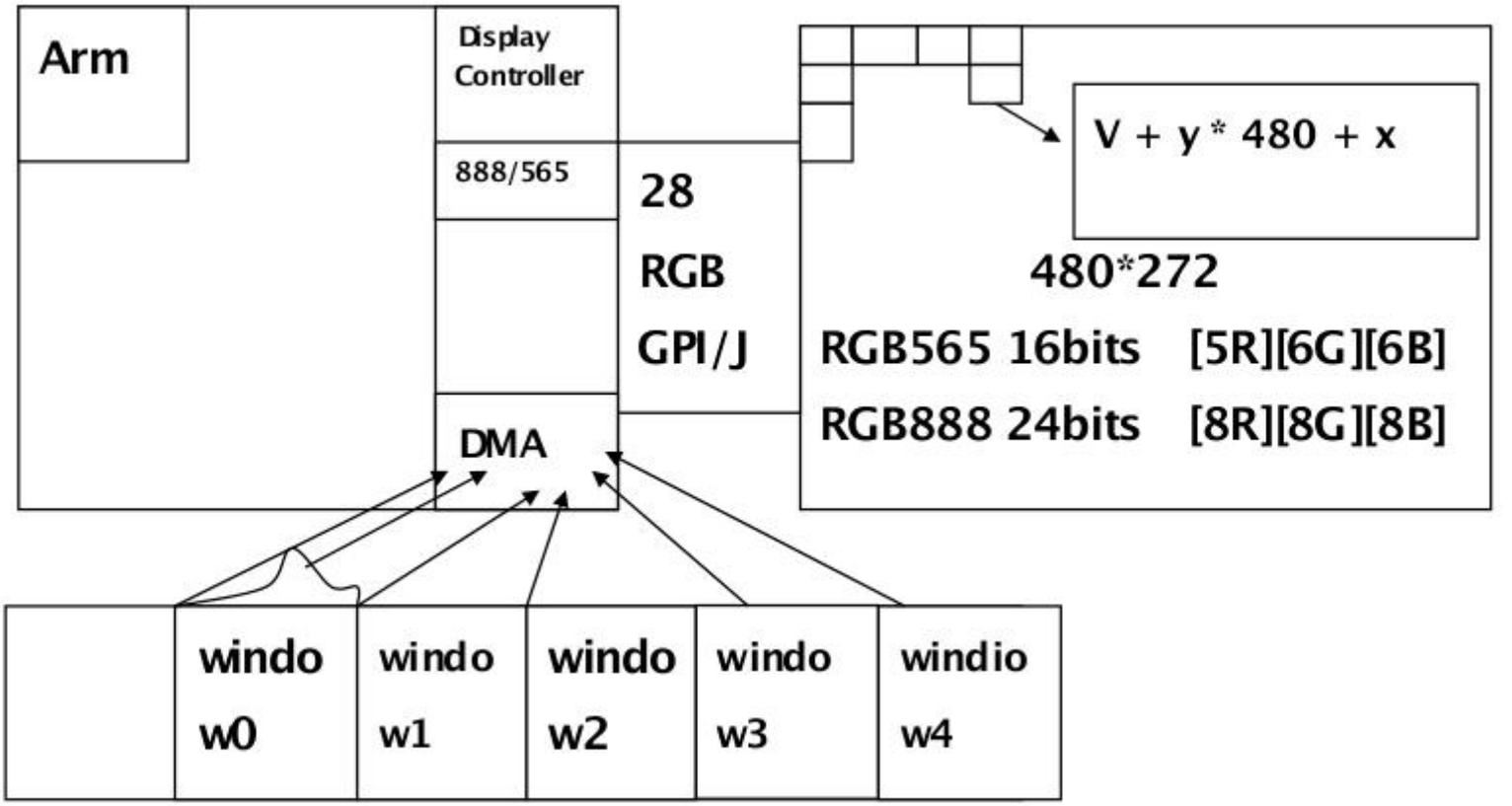
二十一、Framebuffer LCD

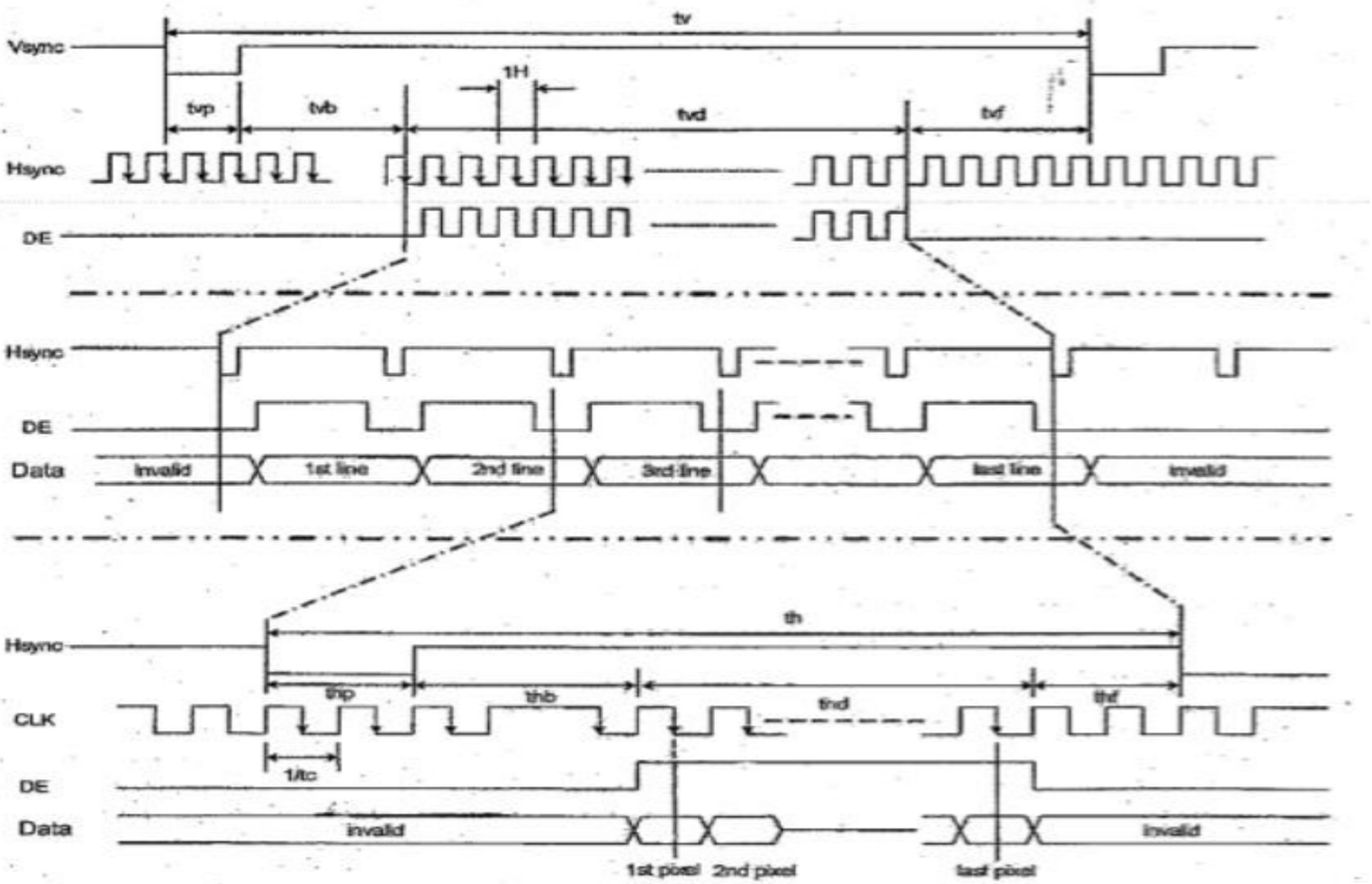
Fb 用户编程 /dev/fb0 fb1 fb2

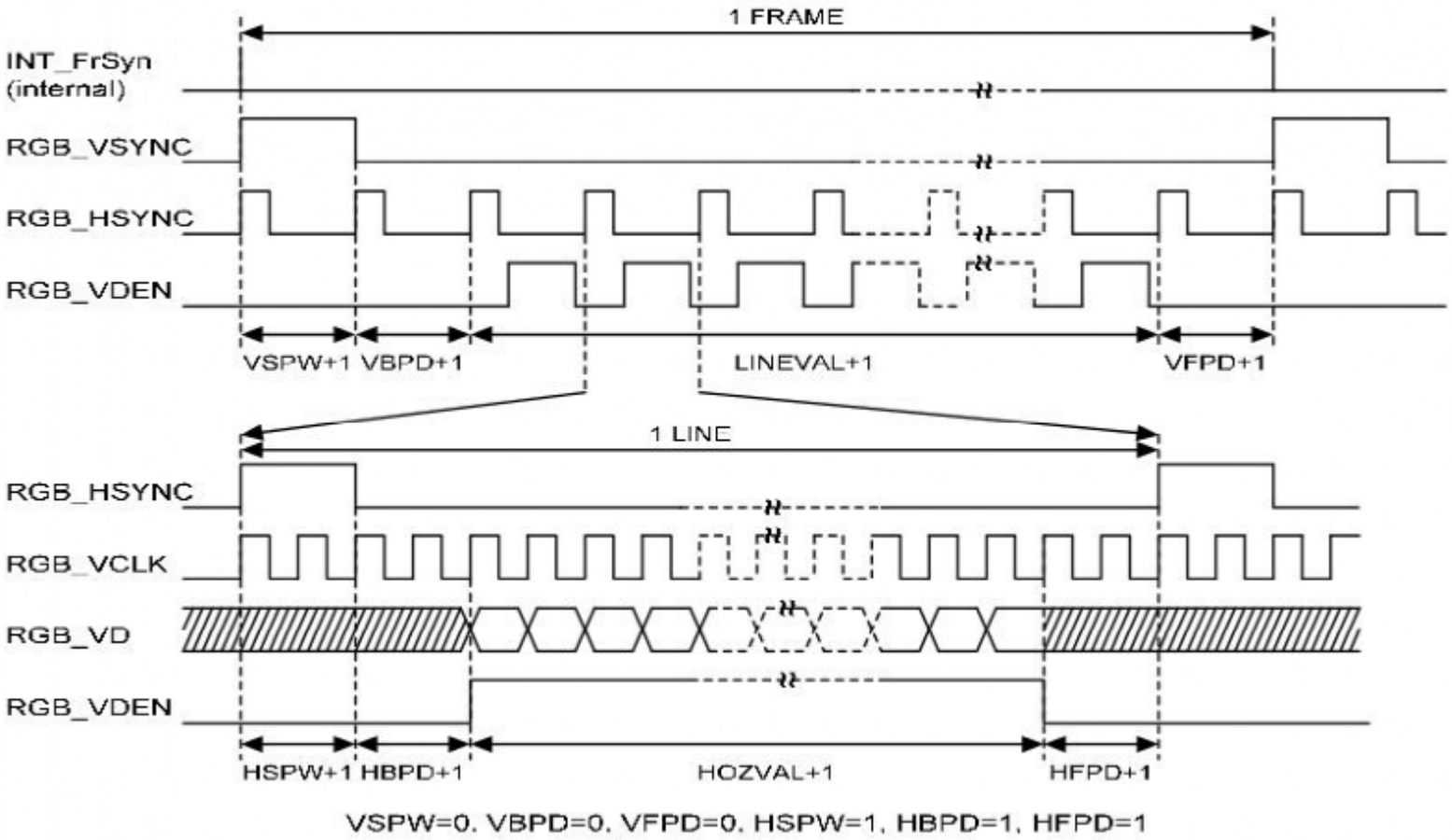
Open close mmap mmunmap

Cat /dev/fb0 >123.raw

Cat 123.raw >/dev/fb0


Arm
Display Controller
888/565
28
RGB
GPI/J
DMA
V + y * 480 + x
480*272
RGB565 16bits [5R][6G][6B]
RGB888 24bits [8R][8G][8B]
windo w0
windo w1
windo w2
windo w3
windio w4


tv
Vsync
1H
tvp
tvb
tvd
tvf
Hsync
DE
Hsync
DE
Data
invalid
1st line
2nd line
3rd line
last line
invalid
th
Hsync
thp
thb
thd
thf
CLK
1/tc
DE
Data
invalid
invalid
1st pixel
2nd pixel
last pixel



VIDOSD1C	Bit	Description
-	[24]	Reserved
ALPHA0_R	[23:20]	Red Alpha value(case AEN == 0)
ALPHA0_G	[19:16]	Green Alpha value(case AEN == 0)
ALPHA0_B	[15:12]	Blue Alpha value(case AEN == 0)
ALPHA1_R	[11:8]	Red Alpha value(case AEN == 1)
ALPHA1_G	[7:4]	Green Alpha value(case AEN == 1)
ALPHA1_B	[3:0]	Blue Alpha value(case AEN == 1)

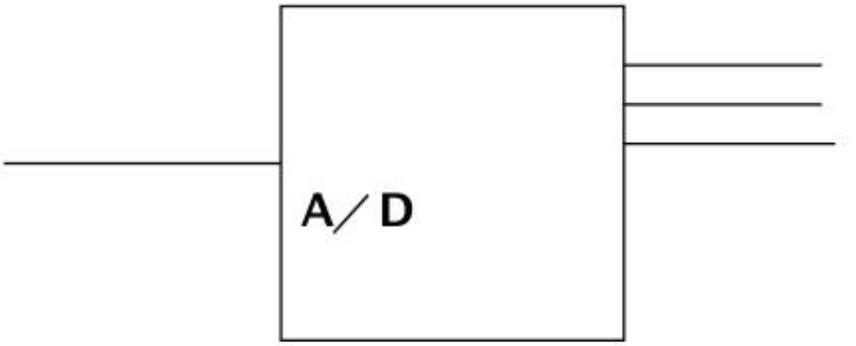
二十三、ADC

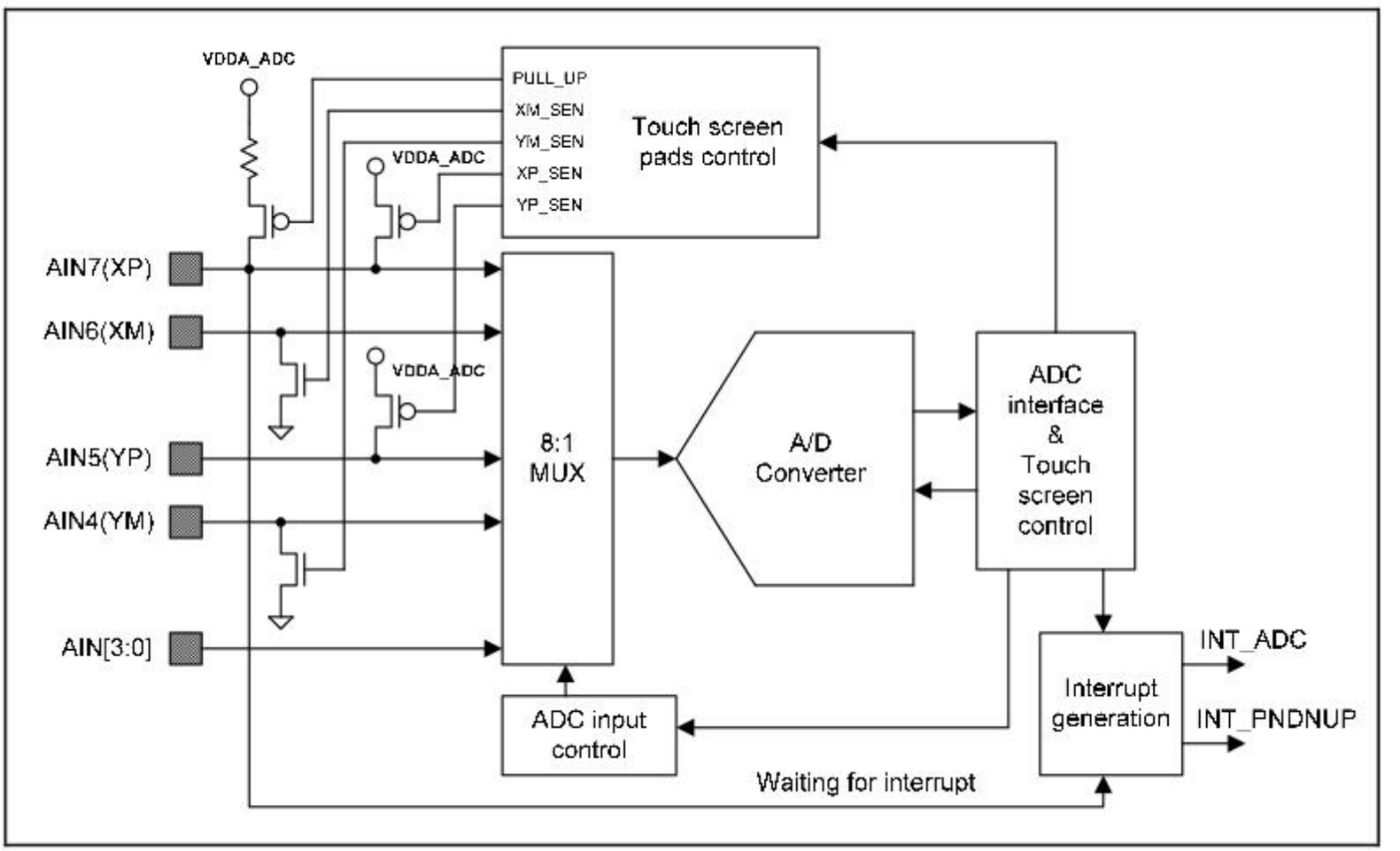
量化位、LSB

0　　0000

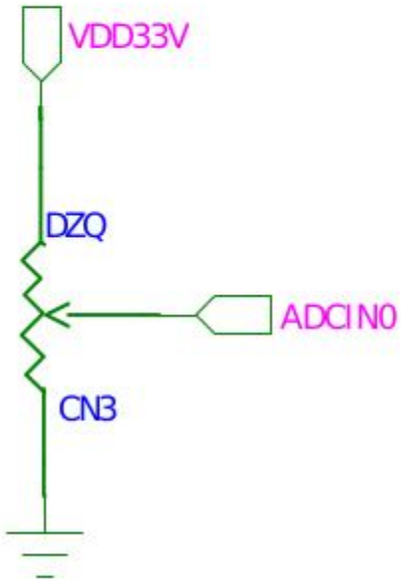
0.3　　0001

5V　　1111

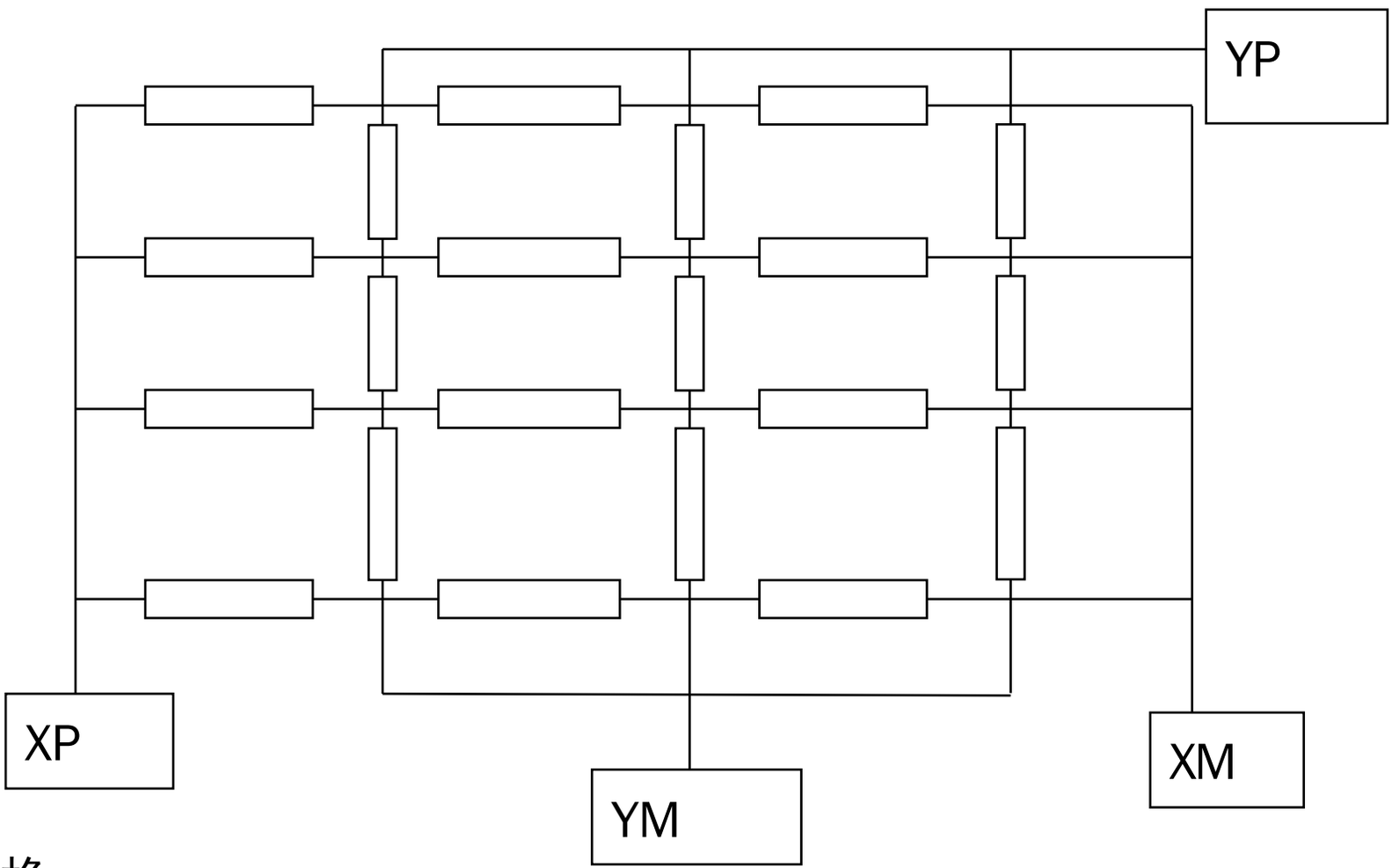




1. Normal Conversion Mode (AUTO_PST=0, XY_PST=0)



2. Separate X/Y position conversion Mode (AUTO_PST=0, XY_PST: control)



转换 x：

xp=高电平

Xm=低电平

ym=高阻态

yp--->x

转换 y:

Yp=高电平

ym=低电平

xm=高阻态

Xp--->y

3. Auto(Sequential) X/Y Position Conversion Mode (AUTO_PST=1, XY_PST=0)

4. Waiting for Interrupt Mode (ADCTSC=0xd3)

二十四、AC97

在用户态放音乐　oss　/dev/dsp　/dev/mixer

Alsa　API

音频解码 libmad

采样率 44100

量化位 16

声道数 2

硬件接口：

PCM　单声道　电话

IIS　双声道

AC97 多声道 带有控制信息

HD

AC97 1996 codec

1.硬件接口

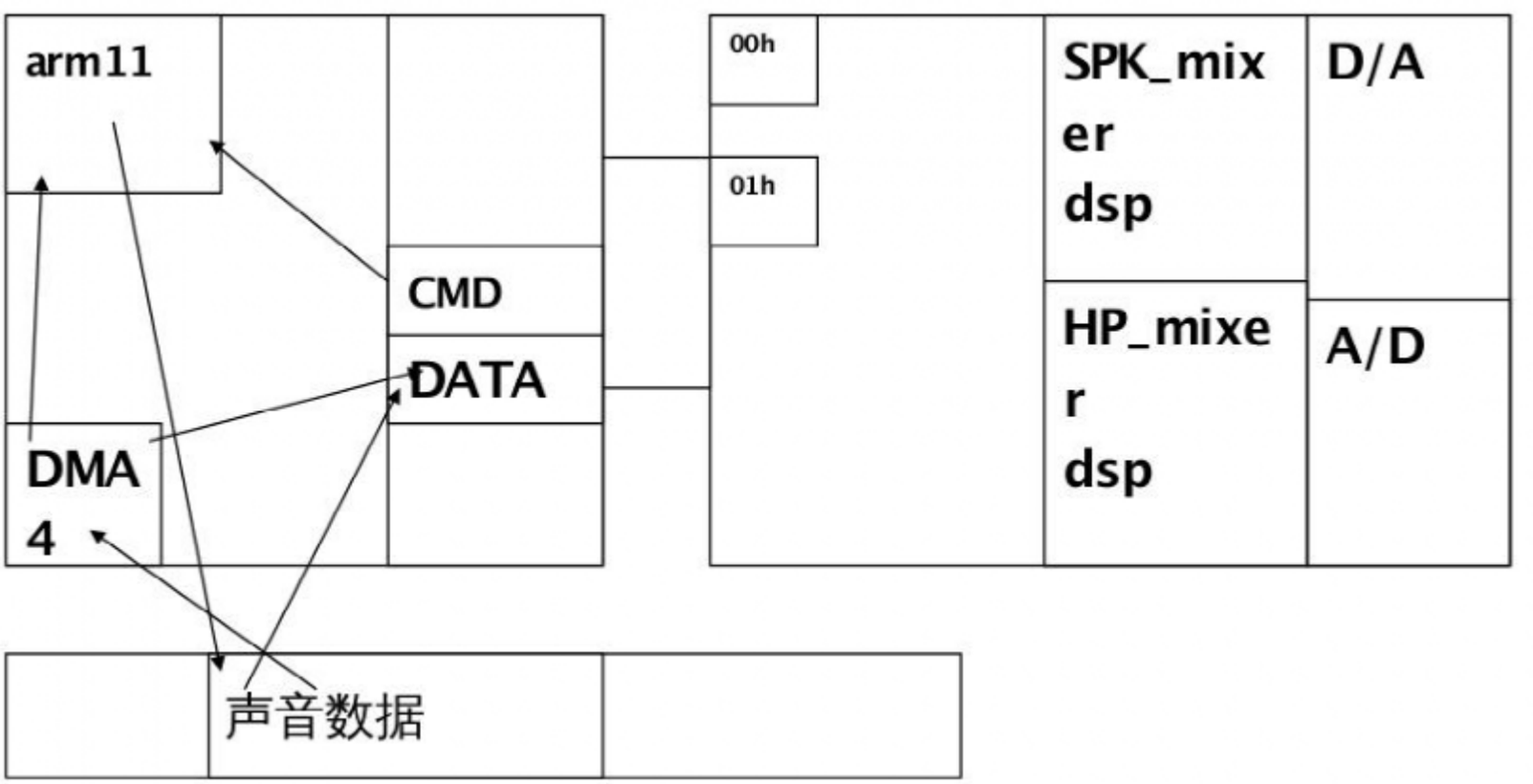
2.数据传输格式 frame 256bits

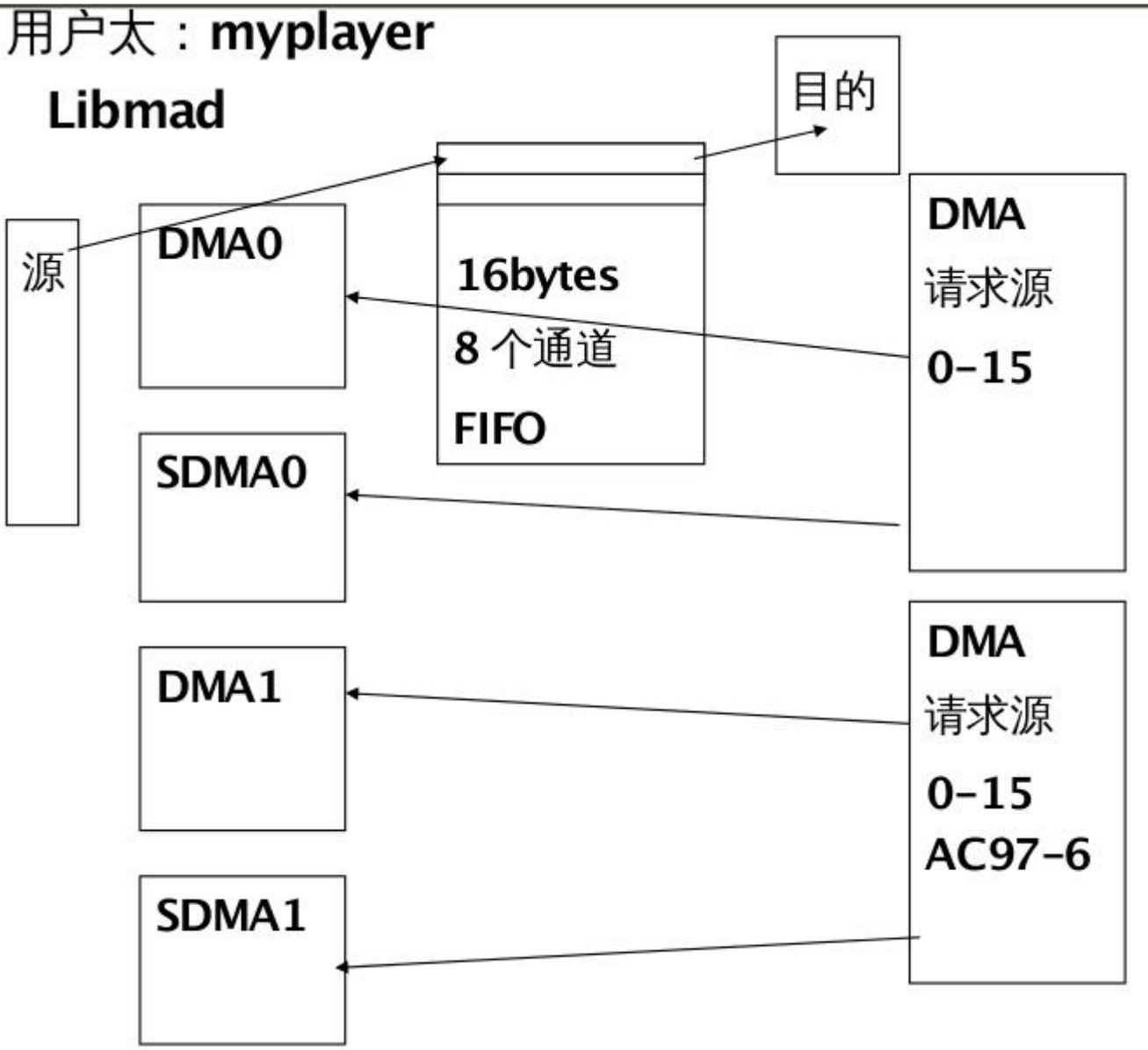
3.传输时序 AC-link

Wm9714 电路图

引脚

连接

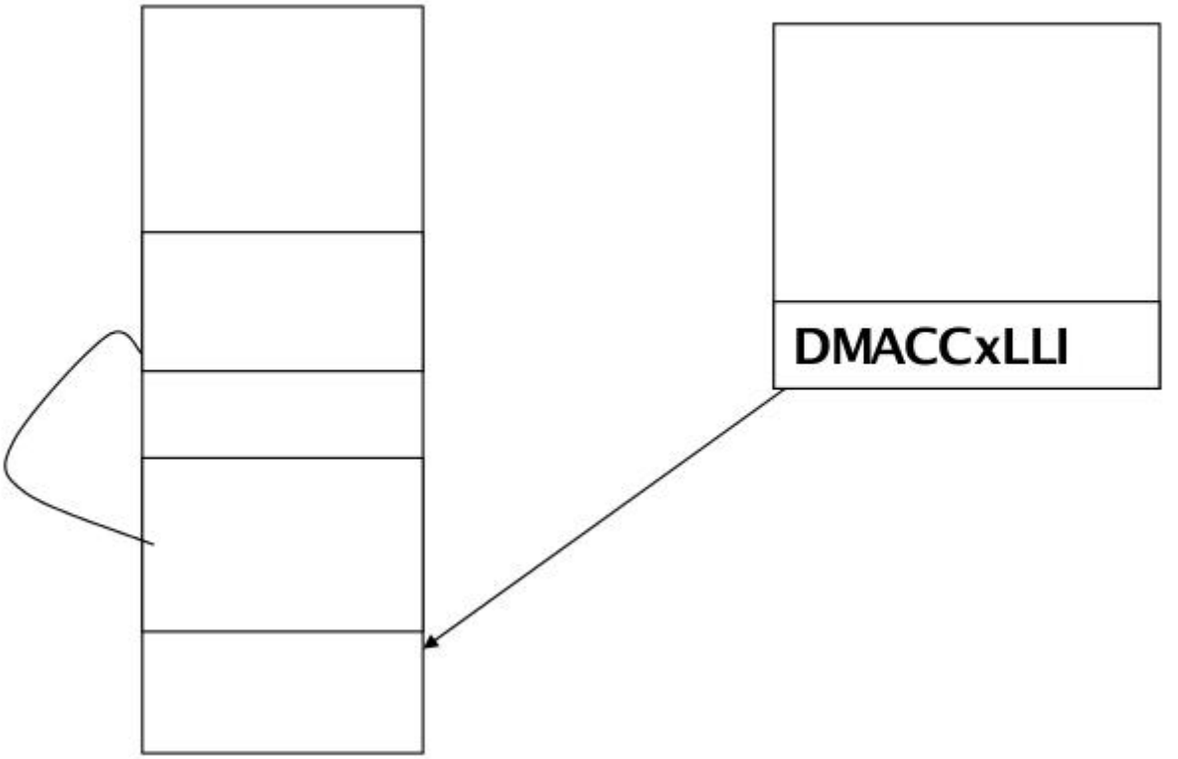




目的：会选择 D M A 控制器

会选择通道

DMA 的任务机制

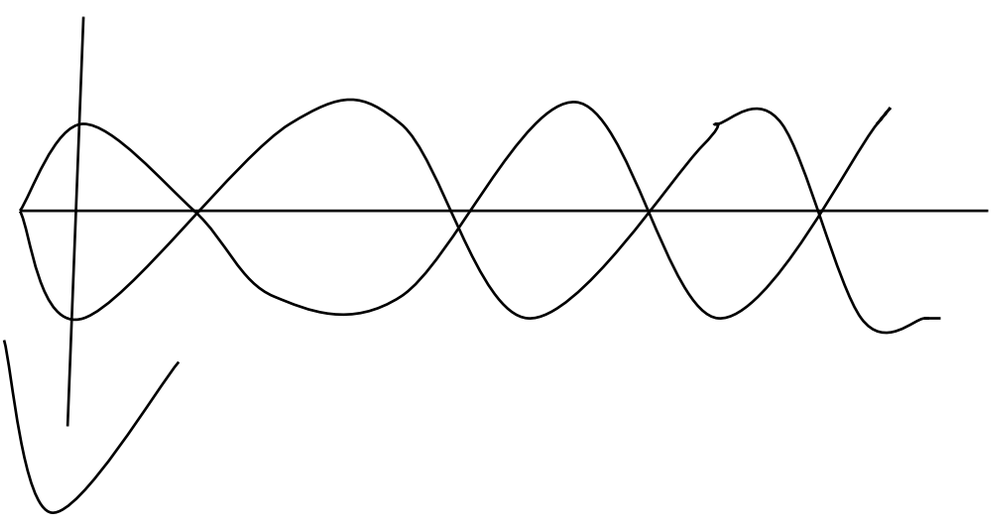


二十五、dm9000

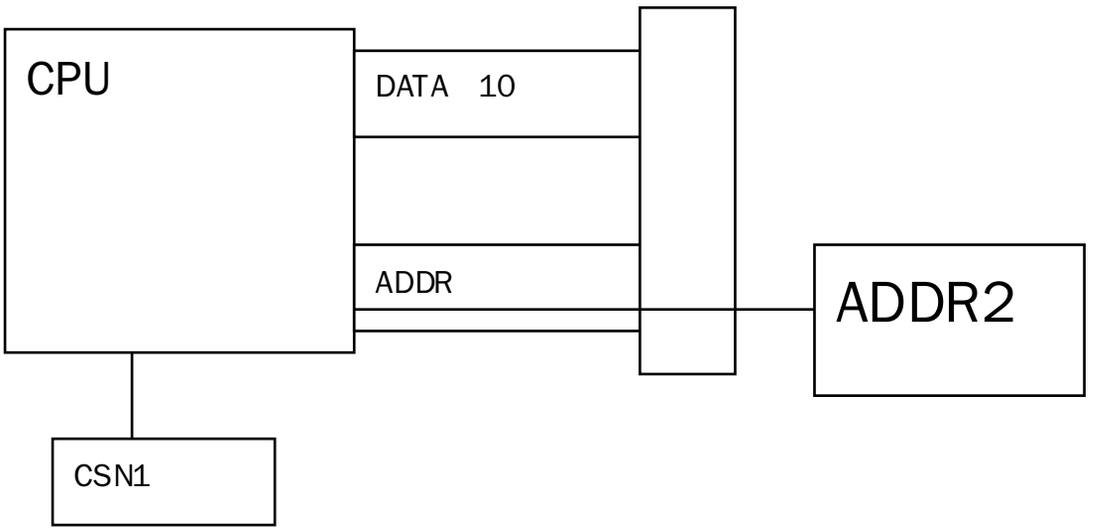
电路图

引脚

连接　SROMC---->static memory



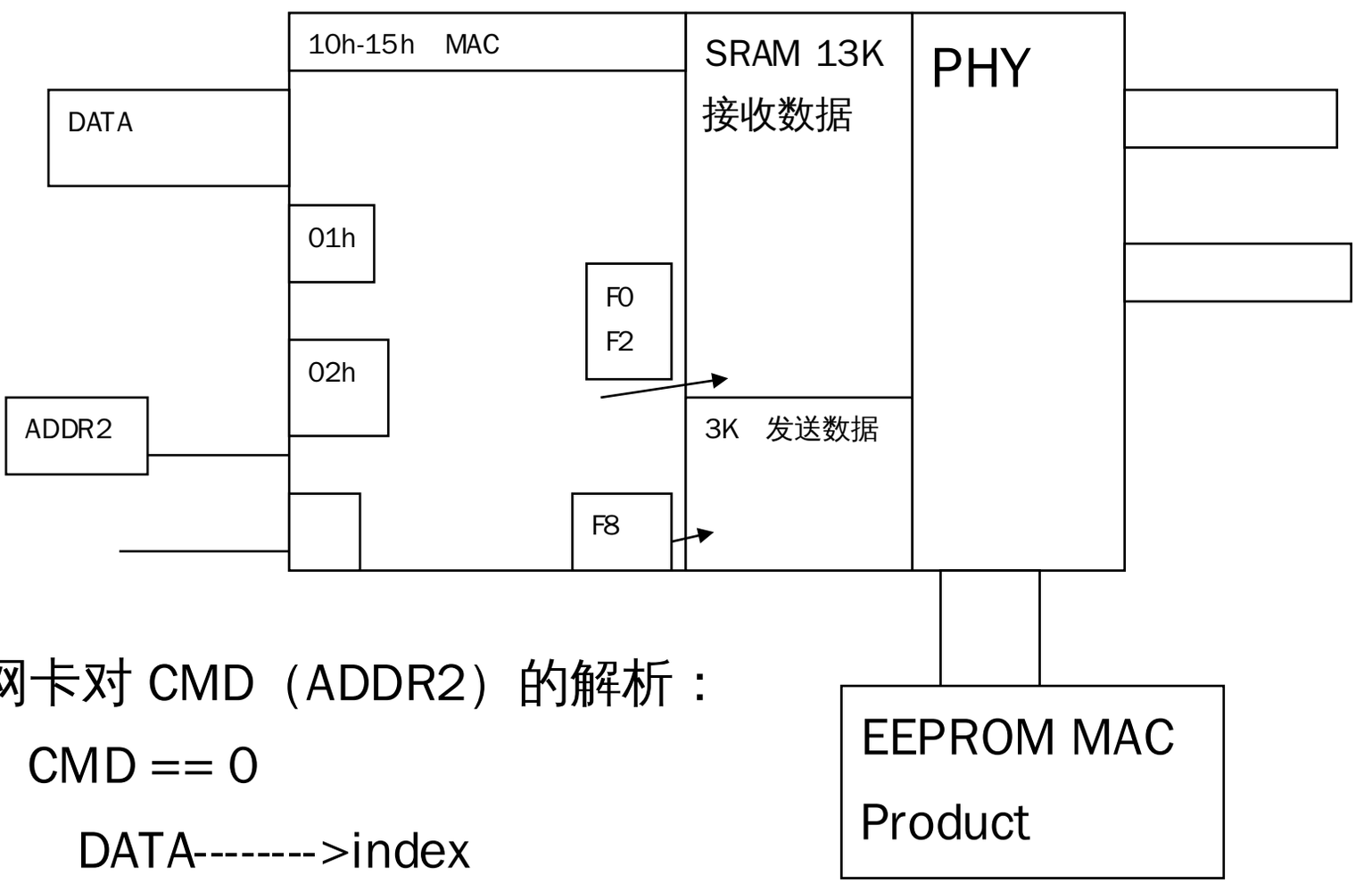
计算机原理



*(v u l *)0x18000000 = 10; ADDR2=0

*(v u l *)0x18000004 = 20; ADDR2=1

dm9000 网卡寻址原理



网卡对 CMD（ADDR2）的解析：

CMD == 0

DATA-------->index

*(v u l *)0x18020000 = 0xf0;

CMD==1

DATA-------->data

*(v u l *)0x18030004 = 20;

例如：给网卡的寄存器 0x01 寄存器写 0x10

*(v u l *)0x18020000 = 0x01;

*(v u l *)0x18020004 = 0x10;

利用 dm9000 收发包

作业：

1.利用一下资源搭建嵌入式开发环境

u-boot-movi_for3.4.bin

zImage3.4

rootfs20130115.tar.bz2

2.自己实现 write_sd

3.参考文档把 u-boot-movi.bin zImage rootfs2010* 写到 SD 卡

中

注意问题：第三步 4 之前要先格式化分区 #mkfs.ext3 /dev/sdb2 #mkfs.vfat /dev/sdb1

第五步 movi write kernel 50000000

4.把网络多播客户端移植到开发板

注意：使用 rootfs2010*文件系统，使用 zImage

播放器使用/usr/bin/madplay

5.把自己做的根文件系统烧写到 nand 中

6.自学 add adc sub sbc rsb rsc mul mla 指令

7.实现用汇编调用 C 语言函数(传递 6 个参数)

Int add(int a, int b, int c, int d, int e, int f)

R0 r1 r2 r3 sp(L) sp(H)

注意：sp 在使用之前是什么样，在使用之后还是什么样

8.运行裸板俄罗斯方框代码

进入源码目录

[root]# make

[root]# cp pic_sound/* /tftpboot -rf

[u-boot-sd]# tftp 54000000 11.bin

[u-boot-sd]# tftp 55000000 els.wav

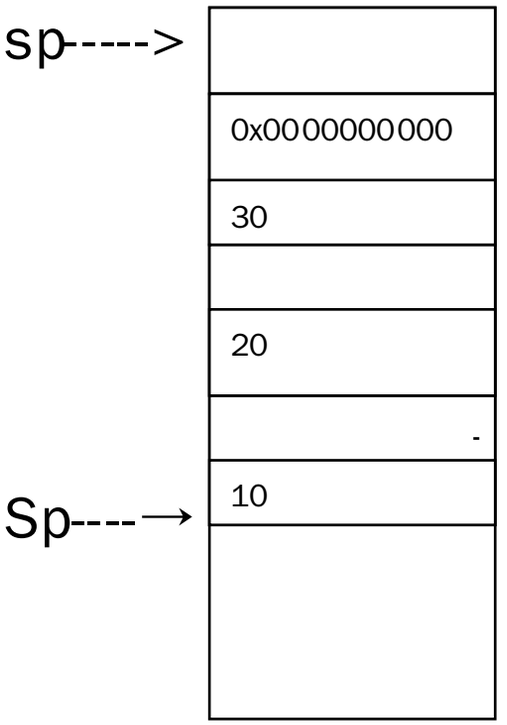
[u-boot-sd]# tftp 50000000 arm.bin

[u-boot-sd]# go 50000000

9.练习汇编中的循环

实现一个简单的链表查询

在 sp 里分配一段空间：直接把 sp 做减法



10.ldr str

Gpmcon 0x7f008820

Ldr r0, =0x7f008820

Ldr r1, [r0]

@Bic r1, r1, #0xff

@Bic r1, r1, #0xff00

Ldr r2, =0xffff

Bic r1, r1, r2

Orr r1, r1, #0x11

Orr r1, r1, #0x1100

Str r1, [r0]

用汇编实现 led 跑马灯

11.查看 6410 手册 31.6.11 章节，完成串口 bps 配置

最后把改程序改为裸板调试（需要初始化 clock）

12.把串口程序改为裸板调试，并且实现一个 shell，支持一下命令

[myuart@up]# led on n(n=0,1,2,3)

[myuart@up]# led off n(n=0,1,2,3)

[myuart@up]# ms addr val

*addr = val;

[myuart@up]# md addr num(word)

13.在 12 的基础上扩展

[myuart@up]# nand erase xxx xxx

[myuart@up]# nand write xxxx xxxx xxxx

[myuart@up]# nand read xxxx xxxx xxxx

14.完成看门狗程序

15.实现按键和看门狗定时换图片

16.实现 ts 换图片

17.把声卡代码改成裸板

18.UDP IP TFTP ETH

19.实现开发板发包，pc 机的 socket 用户态程序接受

buf[ETH|IP/ARP|UPD/TCP|”hello dm9000”]

eth_send(buf, len)

IP:0x800

ARP:0x806

UDP : 17

```
struct ethhdr {
        unsigned char     h_dest[ETH_ALEN];
        unsigned char     h_source[ETH_ALEN];
        unsigned short    h_proto;
};attribute(xxxxx)

struct iphdr {
        __u8       ihl:4,     5words
                   version:4; 4/6
        __u8       tos; 0
        __u16      tot_len; Ip 数据报长度
        __u16      id;0
        __u16      frag_off;0
        __u8       ttl;255
        __u8       protocol;17
```

```
        __u16     check;0
        __u32     saddr;192<<24 | 168 << 16 | 1 << 8 | 20
        __u32     daddr;
        };attribute(xxxxx)
    Check = cal_check(ip_head, ip_head_len);

struct udphdr {
        __be16    source; 1243
        __be16    dest; 5678
        __be16    len; UDP 数据报长度
        __sum16 check;0
  };attribute(xxxxx)

  short cal_sum(unsigned short *buf, int len)
  {
        unsigned int sum = 0;

        while(len > 1)
        {
                sum += *buf;
                buf++;
                len -= 2;
```

```
        }

        if(len == 1)
                sum += *(unsigned char *)buf;
        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum >> 16);

        return (~sum) & 0xffff;
}
```