龙芯中科
LOONGSON TECHNOLOGY

2018 年龙芯技术培训主题三

龙芯固件

龙芯中科技术有限公司

安全应用事业部

2018 年 10 月

目　录

一、龙芯固件介绍

1.1 龙芯固件

龙芯 CPU 是基于 MIPS 架构设计的。目前所支持的固件有 uboot，PMON，昆仑固件、UEFI 等。uboot 主要是针对一些 soc 的芯片，如龙芯 1 号系列 CPU、龙芯 2 号系列 CPU；PMON 是龙芯 CPU 发展以来一直使用开发调试的固件，并且随着 CPU 的更新换代而更新，是目前龙芯 CPU 系列维护时间最久的一种固件，相对稳定而且比较完善。UEFI 是龙芯平台固件的新选择，未来的发展方向。目前龙芯研发团队正在研发基于 EDK2018 版本的 UFEI 固件，未来将支持目前龙芯的 2 号、3 号系列 CPU，并进行应用和推广。同时，昆仑固件已经研发出基于 UEFI 架构支持龙芯平台的固件，在广大客户群体中得到很好的反响和回馈。

本文以 PMON 为例为大家详细介绍龙芯固件的架构和板卡适配过程中常见问题及解决方法。

1.2 PMON 功能概述

PMON：MIPS 架构机器上使用的一种具有 BIOS 部分功能的开放源码软件。BIOS：基本输入输出系统，一组固化到主板的一个 ROM 芯片上的程序，它保存着计算机基本输入输出程序、系统设置信息、开机后自检程序、和系统自启动程序。PMON 作为龙芯产品的基本输入输出系统，是龙芯计算机系统的重要组成部分，负责计算机系统的开机自检、板级初始化、加载操作系统内核以及基本 I / O 功能。

PMON 作为 loader，支持从多种媒介中加载内核，包括：硬盘、U 盘、网络、光盘、Flash，甚至串口和 Ejtag 等。

调试时，通常使用网络进行加载， PMON 支持的网络加载方式的协议包括： http、tftp。

在 Release 时，通常需要将内核及操作系统固化到硬盘、Flash 等存储介质，此时通过环境变量或配置文件规定操作系统的加载方式和参数，PMON 通过解析这些配置项执行加载启动过程。

作为诊断功能时：PMON 提供了方便的调试选项，可以诊断内存故障、数据通路故障、

硬件资源分配故障等功能。

1.3 PMON 代码组织结构（静态）

PMON 代码组织结构如表 1-1 所示。

表 1-1 PMON 目录结构

目录	子目录	说明
Conf	conf/	公共部分配置文件目录
Lib	lib/libc	库文件目录
	lib/libc/arch/mips	
	lib/libz	
	lib/libz/arch/mips	
	lib/libm	
	lib/libm/arch/mips	
Doc		文档目录
examples	examples/	示例程序目录
Pmon	pmon/arch/mips	与 MIPS 相关的目录
	pmon/arch/mips/mm	与内存配置相关的目录
	pmon/cmds	各种命令目录
	pmon/cmds/cmd_main	main 菜单相关目录
	pmon/cmds/gzip	gzip 压缩解压相关目录
	pmon/cmds/lwdhcp	DHCP 相关目录
	pmon/cmds/test	测试程序
	pmon/common	公用的文件目录
	pmon/common/smbios	SMBIOS 相关目录
	pmon/dev	与设备相关的目录
	pmon/fs	文件系统目录
	pmon/fs/cramfs	cramfs 文件系统目录
	pmon/fs/cramfs/include	cramfs 文件系统头文件目录
	pmon/fs/yaffs2	yaffs2 文件系统目录
	pmon/loaders	loader 相关目录
	pmon/loaders/zmodem	Zmodem 协议目录
	pmon/netio	网络相关目录
Fb	fb/	显示相关目录
Sys	sys/arch/mips/include	MIPS 体系头文件目录
	sys/dev	各种设备目录
	sys/dev/ata	ATA 设备目录
	sys/dev/gmac	GMAC 目录
	sys/dev/ic	各种 IC 目录
	sys/dev/mii	网卡相关

	sys/dev/pci	PCI 目录
	sys/dev/usb	USB 目录
	sys/dev/fd	软驱相关
	sys/dev/nand	NAND FLASH 目录
	sys/kern	PMON 的内核
	sys/linux	Linux 相关头文件
	sys/net	网络核心代码
	sys/netinet	各种网络协议目录
	sys/scsi	SCSI 设备目录
	sys/sys/	头文件
	sys/vm	内存管理
Include	include	头文件
Tools	tools	各种工具目录
Targets		各种开发板相关目录
x86emu		模拟执行显卡 BIOS 中的 X86 指令代码目录

1.4 PMON 初始化流程(动态)

PMON 初始化流程是,板卡上电、CPU 自身初始化、串口、Cache、TLB、内存、北桥、南桥、外设、引导操作系统。

PMON 中使用到的处理器的组件:Cache、MC、TLB,但没有使用到异常机制及 MMU 机制,这两个组件是内核驱动运转的。

龙芯计算机系统基本软件部分包括 PMON,内核,Linux 操作系统。

操作系统：是软硬件资源的管理和调度者;是一个所有应用程序和中间件的调用库;是填坑者，让上面的路看起来是平坦的。

与操作系统内核的边界:BIOS 中未使用中断,但是对于 PCI 设备进行了中断号的分配;在内核下需要对中断进行使能及路由配置;处理器窗口、桥片等配置不会在内核中重做；PCI 资源等在内核中将会重新分配;设备驱动使用内核的机制、显示会重新初始化。

1.4.1 PMON 的启动流程概述

1、启动位置位于 CPU 的启动地址是 0xbfc00000，这个地址对应的物理地址是 0x1fc00000，北桥将这一地址影射到 Flash 的 0 地址上。

因此准确的说 PMON 是从 Flash 的 0 地址开始运行。编译的时候 start.o 正好是第

一个被链接的 obj 文件,因此 start.S 的第一条指令是 CPU 运行的第一个指令,位于_start。

2、C 代码的第一个入口函数是 initmips,位于源代码文件 tgt_machdep.c 中。程序最终到 main 函数中运行命令循环。

3、PMON 开始应该是 freebsd 移植过来的,系统调用,设备驱动是 Unix 风格的。

4、PMON 中 CPU 运行于 32 位模式下,屏蔽所有中断。PMON 完全靠查询来完成整个系统,技巧是 idle 函数中调用 scandevs 来扫描设备驱动程序。驱动程序中的中断也是通过被系统查询的时候不断调用来实现的。

1.4.2 PMON 的初始化芯片相关部分

PMON 中 Start.S 的执行流程和龙芯 3A 处理器中 TLB、 Cache、Xbar 的初始化过程分析：

在 Start.S 中主要是完成了主处理器核与其它从处理器核初始化先后顺序的安排，以及在进入 initmips 之前，各个处理器核需要完成的 TLB、 Cache、Xbar 初始化操作流程。

TLB 的作用就是实现虚拟地址到物理地址的转换。对 TLB 初始化就是完 TLB 的无效即可,在代码中主要有两个函数来完成,tlb_clear(无效)与 tlb_init(无效+miss)。初始化 TLB 的过程:设置页大小、设置 wried 使 TLB 表项全部为随机表项；然后用 INDEX 寄存器索引所有 TLB 表项，并将所有 TLB 表项全部清零。

Cache 的作用是将访问过的数据缓存在 Cache 中,以便下次访问时，可以直接从 Cache 中取，这样可以数量级的提高运算速度。龙芯 3A 处理器对 Cache 的初始化基本操作就是用 Cached 地址索引所有 Cache line，并将 Cache line 所对应的 Tag 置为 0。这个初始化操作通过调用 godson2_cache_init,和 scache_init_64 来对 Cache 的 Tag 域进行初始化。TLB 初始化与 Cache 初始化异同点：TLB 初始化和 Cache 的初始化都是将对应行无效化，区别是：TLB 由 Index 寄存器索引对应行，Cache 由地址索引对应行。

Xbar 的作用主要是完成地址空间的路由，通过配置一、二级交叉开关的各个窗口来实现。在 PMON 代码中 Xbar 的匹配代码在 loongson3_fixup.S，loongson3_HT_init.S 和 loongson3_ddr2_config.S 中。Xbar 完成了非法地址的路由，防止

系统死机，以及为了在 PMON 里面可以直接使用 32 位地址，不经 TLB 转换即可实现 HT 空间及 HT 设备的访问。二级交叉开关的初始化主要是针对内存控制器的映射和 PCI 从端口的映射。所以二级交叉开关的配置文件设置在“loongson3_ddr2_config.S”中，这样每个节点对自己内存初始化时，就可以配置对应本节点的内存控制器端口和 PCI 端口了。

1.4.3 复位后 CPU 的初始状态

龙芯 CPU 上电启动后处理器核状态:

1、小端模式(龙芯只支持小端模式);

2、特权等级处于内核态;

3、浮点寄存器处于 32 位数据模式(PMON 启动过程中应将浮点寄存器配置为 64 位模式);

4、中断处于关闭态;

5、非对齐访问会引发例外;

6、64 位地址空间未使能,用户态 64 位操作未使能(龙芯 1 号系列除外);

7、TLB 未初始化(PMON 启动过程中地址空间一直处于未映射段);

8、所有的 Cache 处于未初始化、未使能态(软件应该在初始化内存之前初始化并使能 Cache)。

龙芯系列 CPU 启动地址都固定为 0xbfc00000。PMON 支持从以下两种介质启动：直接寻址介质 NOR FLASH,间接寻址介质：NAND FLASH。但针对具体 CPU 型号或硬件设计，可能不会同时支持,具体按实际情况而定。

1、直接寻址介质：NOR FLASH

现有龙芯 CPU 家族都支持此启动方式,也是目前设计中最常见的方式。总线接口依据具体 CPU 型号有所差别，其支持总线类型有 SPI、LPC、LocalBus。此存储介质空间大小为 1MB。

2、间接寻址介质：NAND FLASH

某些型号的 CPU 支持从 NAND FLASH 或者其它存储介质启动，此时介质起始的 1K 大小会被加载到芯片内部的 RAM 中，映射到 0xBFC00000 或者其它位置。此 1K 大

小的程序/数据可以被处理器直接寻址,支持 32 位访问。

1.4.4 PMON 的上电启动过程

PMON 的上电启动过程如图 1.1 所示:

针对启动过程的注释如下：

1、PMON 基本运行环境建立，即保证处理器正常运行的最基本设置，如：关中断，配置异常向量；如果 CPU 为多核处，确定 PMON 启动核、从核自初始化 Cache、TLB、清 Mailbox（缓存寄存器）等。

2、如果串口在桥片上,则还需对桥片做初步配置。

3、处理器自身相关配置，如：修正频率、初始化Cache、CPU 内部互联配置、非法地址处理。

4、如果连接内存的 I2C 控制器在桥片上，则还需对桥片做初步配置。

5、处理器级 IO 相关的一些配置，如：IO 地址映射、桥片互联配置等。

6、内存运行的相关准备工作，如：内存初始化、拷贝代码到内存、设置堆栈、设置传参等。

7、初始化显示，以及 PCI 设备中断初始化。

8、构造提供给内核与系统的信息，如：内存布局、开发板类型等。

9、自启动系统时会查找 boot.cfg 或相关环境变量，根据查找结果来启动系统，如果找不到上述文件或变量，则会返回到命令行界面。

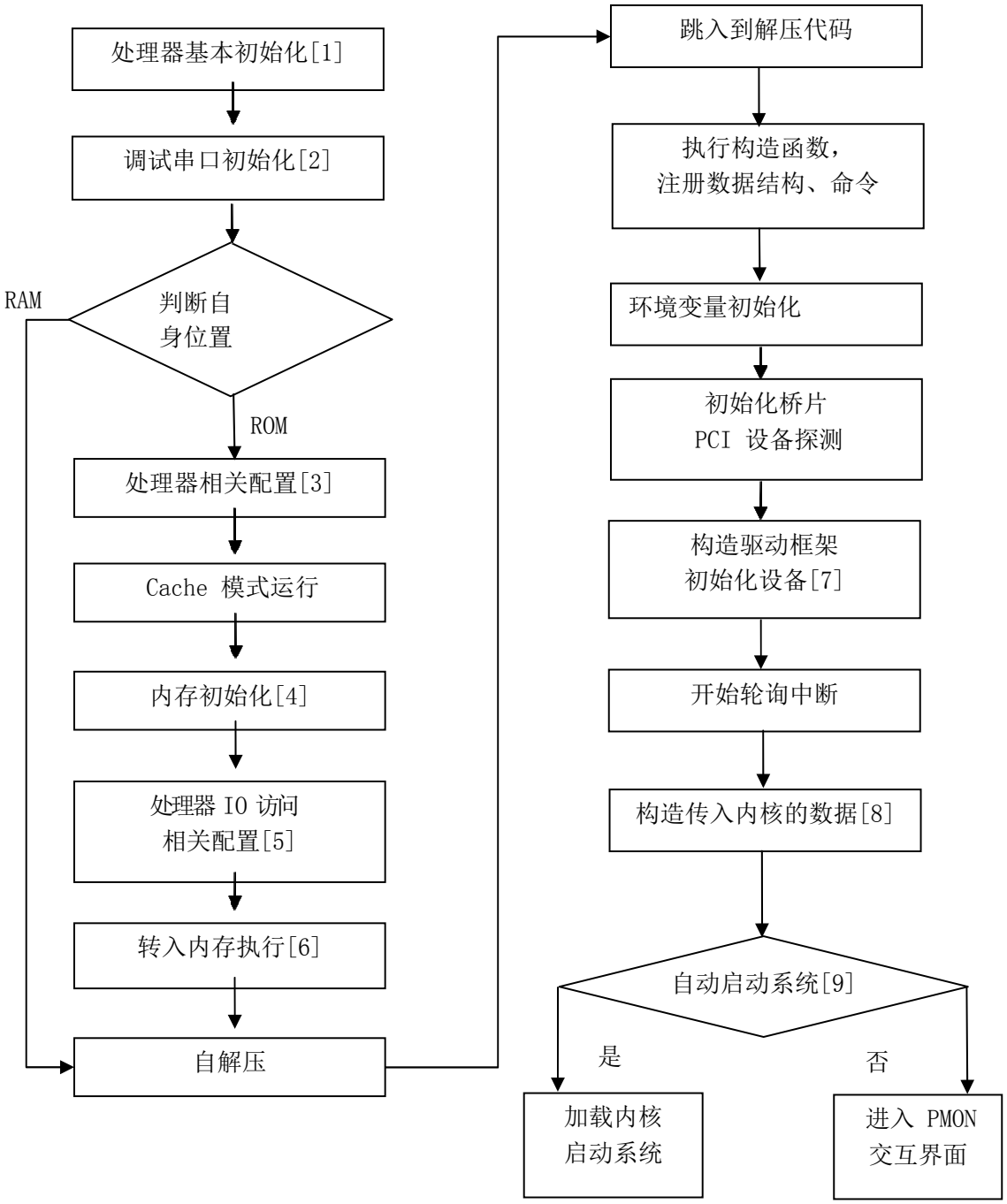
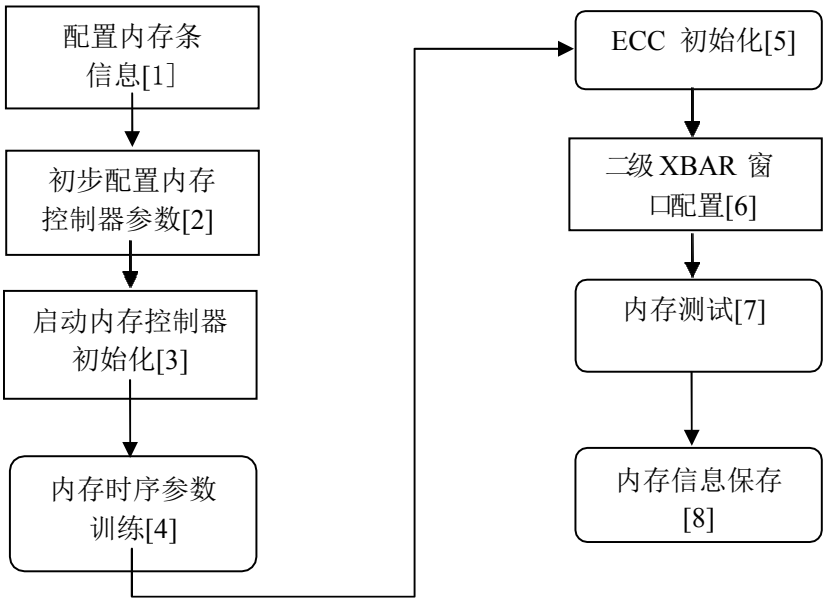


图 1.1PMON 上电启动过程

1.4.5 内存初始化

内存初始化的主要工作包括：配置内存控制器、配置二级 XBAR。目前 PMON 内存初

始化的过程如下图 1.2 所示：



注：1. 方角矩形框为必选步骤，圆角矩形框为可选步骤。
2. 内存时序参数训练，也称内存训练（代码中命名为 RB_level）。

图1.2　　PMON内存初始化流程图

1、配置内存条信息

该步骤的作用是获得需要的内存信息，包括：内存类型（DDR2 还是 DDR3）、内存条类型（UDIMM、RDIMM、是否带 ECC）、内存条的数据线宽度、内存颗粒的 Bank 个数/地址线的行数/列数、是否进行地址 mirror、与 CPU 的引脚连接方式（CS 的映射方式）、内存的容量等。配置方式分为两种（通过配置选项 AUTO_DDR_CONFIG 选择）：根据内存条 SPD 自动检测和人工配置。通过 SPD 自动检测时，需要提供内存条 SPD 所在的 I2C 总线地址。

2、初步配置内存控制器参数，该过程分为三步：

1）根据内存类型从 BIOS 数据段装载基本参数

根据内存类型（DDR3 或 DDR2，UDIMM（包括 SODIMM）或 RDIMM，以及是否进行了内存训练）从 BIOS 数据段的不同位置选择相应的基本参数。当尚未进行内存训练时，根据 DDR3/2 以及 UDIMM/RDIMM 共 4 种组合对应选择一个位置加载；当进行了内存训练时，直接从训练后保存的内存参数位置加载。

2）根据内存条信息重新配置部分参数

重新配置 1）中所述除了内存类型和内存条类型以外的其他内存条信息。如果加载的是训练后保存的参数，则跳过此步骤。

3）单独修改部分参数

调试时使用。

3、启动内存控制器初始化

向内存控制器的参数 param_start 写入值 1，使得硬件开始内存控制器的初始化过程。软件轮询初始化是否结束。

4、内存时序参数训练

调用 ARB_level 函数，对部分时序参数进行重新配置。

5、ECC 初始化

对于使能 ECC 的情况，进行 ECC 初始化。

6、 二级 XBAR 窗口配置

根据使能的内存控制器个数和每个控制器的内存容量来配置二级 XBAR。

7、内存测试

测试内存读写的正确性，同时可以测试二级 XBAR 窗口配置是否正确。供调试使用。

8、内存信息保存

保存内存频率、内存条的 ID 等相关信息，供智能的内存训练使用。内存训练可以根据内存频率、内存条是否发生过改变来决定是否需要重新训练。

配置内存控制器的过程为步骤 1-5。当系统中有多个内存控制器时,内存控制器需要分别进行配置(重复步骤 1-5)。

配置选项 ARB_LEVEL 决定是否包含步骤:4、8。宏定义 DEBUG_DDR 决定是否包含步骤 7。宏定义 DISABLE_DIMM_ECC 决定是否包含步骤 5。

如果使能内存训练,则配置内存控制器需要在 Cache 初始化并使能之后进行。

内存初始化的代码封装在 loongson*_ddr[2]_config.S 中,使用 64 位的寄存器 s1 进行参数传递,使用时需要正确设置输入参数(s1)。内存初始化完成后会设置寄存器 msize(寄存器 s2)的值,告诉后续代码系统的内存容量,每个节点对应 8 位,从低到高分别对应节点 0/1/2/3,单位为 512MB。

1.4.6 PMON 运行时的地址空间

PMON 将物理地址 0x0F000000~0x0FFFFFFF 之间的内存用作保留空间,操作系统内核不得使用该段内存;其中 0x0F400000~0x0F7FFFFF 用作运行时的堆空间,0x0F000000~0x0F3FFFFF 用作 PMON 的代码段与数据段。

表 1-2 PMON 低 256M 空间分配表

地址	描述
0x0F80 0000 - 0x0FFF FFFF	固件与内核接口参数地址，在操作系统启动后此空间要保留给固件使用， 其中 0x0FFFE000 是 SMBIOS 的基 地 址 ， 0x0FFF0000 - 0x0FFF0110， 用来保 存内存条信息，供智能的内存训练使用
0x0F40 0000 - 0x0F7F 0000	PMON 堆栈
0x0F00 0000 - 0x0F3F 0000	PMON 代码段和数据段
0x0020 0000 - 0x0EFF FFFF	内核和 RamDisk 加载
0x0000 0000 - 0x001F FFFF	保留

二、PMON 开发和调试

2.1 驱动框架和增减定制

PMON 来自于 FreeBSD，虽然是个 BIOS 但是兼具 linux 的设计风格，一个最大的特点就是把所有的设备、文件都抽象成文件进行操作。PMON 可分为文件系统层、中间层、和设备驱动层。文件系统层利用文件操作的方式进行用户交互及硬件设备管理和查询等功能；中间层衔接设备驱动层和文件系统层包含文件系统抽象和 TCP / IP 协议等组件；设备驱动层提供系统平台主要设备驱动的程序。

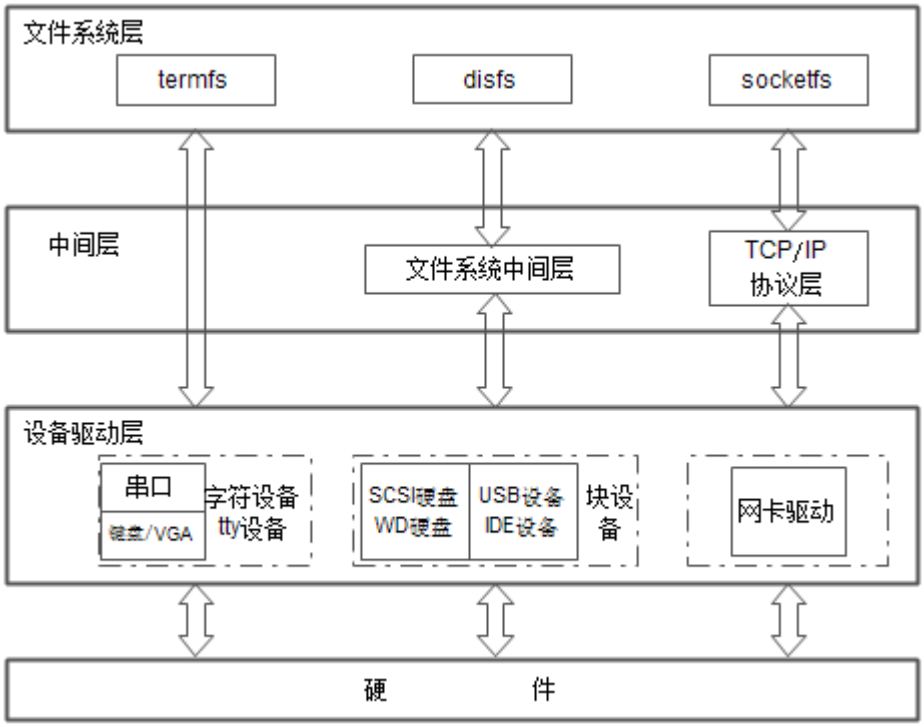


图2.1 PMON的驱动框架结构

2.1.1 PMON 设备驱动模型

1、PMON 设备组织成树型结构，这棵树由配置文件(conf/files, Targets/ Bonito3a780e /conf/Bonito.3a780e)来描述，叶子节点是设备，非叶子节点是总线或设备控制器，儿子挂载在父亲总线上，对于一个叶子节点（设备），可以有几个到达它的路径，但是每个路径提供的接口是一致的。每个设备都是挂载到父总线上的，为了避免先有鸡还是先有蛋的问题，树根是个虚拟的 root 总线（mainbus），cpu 就是挂载到其上的。总线和设备名后一般有个数字后缀，因为一个设备可有多个功能（例如 PCI 可有 8 个功能）。每个设备可提供一个定位器（locator），它是跟这个设备相关的一组参数，例如，可以是设备的基地址，IRQ 等参数组成了定位器，这在父亲（总线）的规范文件 files.*里，他们在自动配置（autoconfiguration）过程中使用。

pmonconfig 针对平台配置文件产生了设备树。在启动时系统通过自动配置过程使用这个设备树来初始化系统，系统使用深度优先方法搜索设备树来初始化。对于每个设备和总线，定义了 match()和 attach()两个方法。在系统初始化过程中，每个总线调用他们孩子（设备）的 match()，并提供给它一些配置文件描述的参数，match()使用提供的参数来探测所给信息对于孩子（设备）是否匹配，如果匹配，返回一个非零值。在孩子（设备）匹配后，父亲（总线）调用设备的 attach()，这里孩子设备初始化自己，如果孩子还是

总线的话，那么他采用他父亲一样的方法，即调用孩子的 match()。每个设备都有个通用的 cfdata struct 结构(Targets/Bonito3a780e/compile/Bonito/ioconf.c)，包含了定位器参数，在初始化过程中用于 match(),attach()。使用设备名作参数在 cfdriver 列表里搜寻，然后返回匹配的 cfdriver，使用 driver 作为参数用 driver_cd 在 cfdriver 的 cfattach 列表域搜寻，返回匹配的 cfattach。总线所用的 cfdata 结构由 ioconf.c 在自动配置期间从配置文件（conf/files, Targets/ Bonito3a780e /conf/Bonito.3a780e）产生。

在 file.*文件里，每个总线都定义了 cfdata 定位器域。例如 ata{[channel= -1], [drive = -1]}其中 channel，drive 是定位器名，是 cfdata 结构里 loc 数组的偏移。配置文件分与板卡无关的和板卡相关的，有 conf/files,conf/GENERIC_ALL, Targets/Bonito3a780e/conf/Bonito.3a780e。Targets/Bonito3a780e/conf/files.Bonito3a780e 要根据配置文件来增减相应文件包含。自动配置系统根据配置文件来决定包含哪些源文件。一般结构如下：

device <设备/总线名>

attach <设备名> at <总线名>

file 指出相应文件的路径

例如:

device pcibr{} : pcibus

attach pcibr at mainbus

file pmon/dev/pcibr.c pcibr needs-flag

need-flag 指示 config 要创建相的 pcibr.h 文件。

2、自动配置(autoconf)系统

Targets/Bonito3a780e/compile/Bonito/ioconf.c/cfdata,sys/kern/subr_autoconf.c 它们构成了 autoconf 系统。

pmonconfig 读取配置文件(Targets/Bonito3a780e/conf/)来确定包含那些驱动和功能，之后会在 Targets/Bonito3a780e/compile/下创建 Bonito 目录,包含针对各种驱动的*.c 和*.h 文件。然后就可以 make 了。

3、ioconf.c 和 cfdata

编译目录(Targets/Bonito3a780e/compile/Bonito)下的 ioconf.c 是自动配置过程访问的中心点，其中的一个结构体数组 cfdata 描述了要编译的驱动，例如对于配置文件里的:

```
mainbus0 at root
localbus0 at mainbus0
pcibr0 at mainbus0
pci* at pcibr?
ohci0 at pci? dev ? function ?
usb0 at usbbus ?
pciide* at pci ? dev ? function ? flags 0x0000
wd* at pciide? channel ? drive ? flags 0x0000
ide_cd* at pciide? channel ? drive ? flags 0x0001
```

在 ioconf.c 里会产生

```
/* attachment driver unit state loc flags parents nm ivstubs
/* 0: mainbus0 at root */
{&mainbus_ca, &mainbus_cd, 0, NORM, loc, 0, pv+ 1, 0, 0,0},
/* 1: pcibr0 at mainbus0 */
{&pcibr_ca, &pcibr_cd, 0, NORM, loc, 0, pv+ 6, 0, 0,0},
/* 2: usb0 at ohci0 */
{&usb_ca, &usb_cd, 0, NORM, loc, 0, pv+ 8, 0, 0,0},
/* 3: localbus0 at mainbus0 */
{&localbus_ca, &localbus_cd, 0, NORM, loc, 0, pv+ 6, 0, 0,0},
/* 4: pci* at pcibr0 bus -1 */
{&pci_ca, &pci_cd, 0, STAR, loc+ 1, 0, pv+ 4, 1, 0,0},
/* 6: pciide* at pci* dev -1 function -1 */
{&pciide_ca, &pciide_cd, 0, STAR, loc+ 0, 0, pv+ 0, 3, 0,0},
/* 7: ohci0 at pci* dev -1 function -1 */
{&ohci_ca, &ohci_cd, 0, NORM, loc+ 0, 0, pv+ 0, 3, 0,0},
/* 8: wd* at pciide* channel -1 drive -1 */
{&wd_ca, &wd_cd, 0, STAR, loc+ 0, 0, pv+ 2, 6, 0,0},
/* 9: ide_cd* at pciide* channel -1 drive -1 */
{&ide_cd_ca, &ide_cd_cd, 0, STAR, loc+ 0, 0x1, pv+ 2, 6, 0,0},
```

其中 struct cfdata 数据结构的定义在"sys/sys/device.h"文件中:

```
/*
* Configuration data (i.e., data placed in ioconf.c).
*/
struct cfdata {
        struct cfattach *cf_attach;   /* config attachment */
        struct cfdriver *cf_driver;   /* config driver */
        short    cf_unit;                    /* unit number */
        short    cf_fstate;                  /* finding state (below) */
        int          *cf_loc;                /* locators (machine dependent) */
        int          cf_flags;               /* flags from config */
        short    *cf_parents;            /* potential parents */
        int    cf_locnames;              /* start of names */
        void (**cf_ivstubs)            /* config-generated vectors, if any */
                                    __P((void));
        short    cf_starunit1;      /* 1st usable unit number by STAR */
};
extern struct cfdata cfdata[];
#define FSTATE_NOTFOUND 0 /* has not been found */
#define FSTATE_FOUND 1    /* has been found */
#define FSTATE_STAR 2 /* duplicable */
#define FSTATE_DNOTFOUND 3 /* has not been found, and is disabled */
#define FSTATE_DSTAR 4     /* duplicable, and is disabled */
```

我们可以很明显的看到设备是以树型组织的，mainbus 是 root 的孩子，pci 桥是 mainbus 的孩子，各个 pci 功能是 pci 桥的孩子，如此这般下去。ioconf 里的这个结构是由 config 来产生的，不需要你来做。

驱动不直接访问硬件，而是通过抽象的与总线无关的函数来访问，这样就隐藏了硬件细节，抽象处理和访问由父亲（总线）来提供，在驱动和内核子系统间的接口和依赖有 attributes 来描述。

在启动时通过 ioconf.c 里的 cfdata 表来走访整个设备树，对于每个设备都要实现它的接口函数，设备才能正确驱动。

4、直接配置和间接配置。

1）直接配置

总线适配器提供了所有当前可用孩子设备硬件的列表。通过读取 PCI 配置空间，

总线驱动可以找到当前可用的硬件设备并加载驱动。

2）间接配置

这时，总线驱动必须实现 config_search()函数，它走访 cfdata 里所有可能的孩子设备驱动。它调用所有可能孩子设备的 foo_match()函数。所以 config_search()只在初始化时调用一次。

5、设备类型与操作

设备类型与相应操作如下：

D_DISK: open, close, read, write, ioctl

D_TAPE: open, close, read, write, ioctl

D_TTY: open, close, read, write, ioctl, stop, tty, poll

cdevsw,bdevsw 两个数据结构表示字符设备，块设备。

具体结构在 sys/sys/conf.h。

2.1.2 PMON 设备驱动配置与加载过程

通过 pmoncfg 命令来自动生成 Makefile 和h 文件 pmoncfg，代码位 tools/pmoncfg.

pmoncfg 命令格式如下：

```
pmoncfg configfile
```

执行这个命令，pmoncfg 读 configfile 从里面查找包含下面关键字的行

```
machine target arch sysarch
```

然后包含如下配置文件 ：

```
Targets/<target>/conf/files.<target>
Targets/<sysarch>/conf/files.<sysarch>
conf/files
```

在这 3 个配置文件中还可能通过 include 命令包含其他配置文件。在配置文件中包含设备总线描述，例如以 3a780 为例，设备树的构成如图所示。

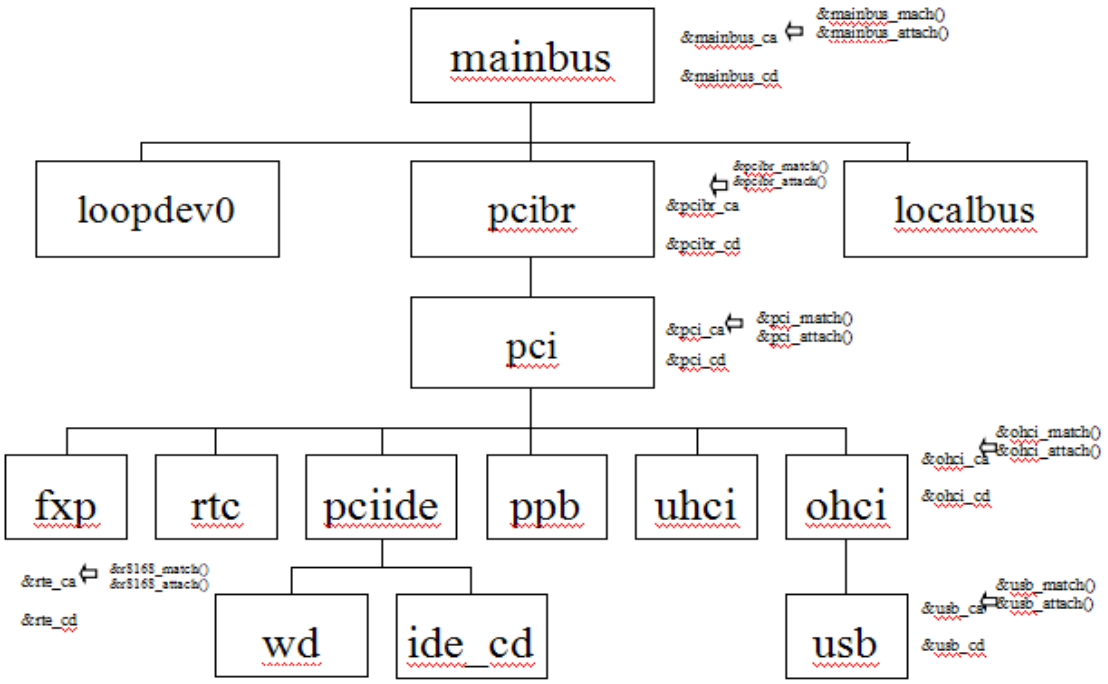


图 2.2　LS3A780E 平台一个典型的设备树结构

代码在 Targets/Bonito3a780e/conf/Bonito.3a780e 文件中。代码描述如下：

```
mainbus0 at root
localbus0 at mainbus0
loopdev0 at mainbus0
pcibr0 at mainbus0
pci* at pcibr? ppb* at pci? dev ? function ? # PCI-PCI bridges pci* at
ppb? bus ?
fxp0 at pci? dev ? function ? # Intel 82559 Device inphy*
at mii? phy ? # Intel 82555 PHYs
rte* at pci? dev ? function ?
uhci* at pci? dev ? function ?
ohci* at pci? dev ? function ?
usb* at usbbus ?
pciide* at pci ? dev ? function ? flags 0x0000 wd* at
pciide? channel ? drive ? flags 0x0000
pseudo-device loop 1 # network loopback
ide_cd* at pciide? channel ? drive ? flags 0x0001
```

在 make cfg 调用 pmoncfg 后生成 ioconf.c 文件,该文件位于 Targets/<主板型号

>/compile/目录下。

```
short pv[14] = {
1, 10, -1, 12, 11, -1, 6, -1, 9, -1, 7, -1, 0, -1,
};
struct cfdata cfdata[] = {
{&mainbus_ca,          &mainbus_cd,       0,      NORM,    loc,              0,     pv+ 2,   0,   0,
0},
{&pcibr_ca,            &pcibr_cd,         0,      NORM,    loc,              0,     pv+12,   0,
0,   0},
{&usb_ca,              &usb_cd,           0,      STAR,    loc,              0,     pv+ 3,
0,   0,   0},
{&loopdev_ca,          &loopdev_cd,       0,      NORM,   loc,              0,     pv+12,   0,   0,
0},
{&localbus_ca,         &localbus_cd,      0,      NORM,    loc,              0,     pv+12,   0,   0,
0},
{&inphy_ca,            &inphy_cd,         0,      STAR,    loc+ 1,           0,     pv+10,   1,
0,   0},
{&pci_ca,              &pci_cd,           0,      STAR,    loc+ 1,           0,     pv+ 0,
3,   0,   0},
{&fxp_ca,              &fxp_cd,           0,      NORM,    loc+ 0,           0,     pv+
6,    7,   0,    0},
{&rte_ca,              &rte_cd,           0,      STAR,    loc+ 0,           0,
pv+ 6,    7,   0,   0},
{&pciide_ca,           &pciide_cd,        0,      STAR,    loc+ 0,           0,     pv+ 6,
7,   0,   0},
{&ppb_ca,              &ppb_cd,           0,      STAR,    loc+ 0,           0,
pv+ 6,   7,   0,   0},
{&uhci_ca,             &uhci_cd,          0,      STAR,    loc+ 0,           0,
pv+ 6, 7, 0, 0},
{&ohci_ca,             &ohci_cd,          0,      STAR,    loc+ 0,           0,
pv+ 6, 7, 0, 0},
{&wd_ca,               &wd_cd,            0,      STAR,    loc+ 0,           0,
pv+ 8, 10, 0, 0},
{&ide_cd_ca,           &ide_cd_cd,        0,      STAR,    loc+ 0,           0x1,   pv+ 8,
10, 0, 0},
{0},
{0},
```

```
{0},
{0},
{0},
{0},
{0},
{0},
{(struct cfattach *)-1}
};
```

在生成的文件中,cfdata 是一个设备树,pv 数组定义一个设备的父设备。每个节点的父设备都是一个数组,在设备的 cfdata 结构中定义数组的开始。 为了简化,PMON 将所有设备的设备的父设备数组放在一个数值 pv 中.其中-1 表示数组的结束。

以上述龙芯 3A780E 中的 cfdata[1]为例子,cfdata[1].pv == pv+12，pv[12]==0,说明其父设备为 cfdata[0]。设备自动初始化也是利用树进行递归的初始化,其代码实现在 configure 函数中,从 mainbus 开始依次初始化各级设备。其中的 cfdata[1]有两个关键数据结构 pcibr_ca、pcibr_cd。为了方便具体,我们以 USB 设备为例,对于 USB 设备而言是 usb_ca 和 usb_cd,其具体定义在 usb_storage.c 中,usb_storage.c 定义的驱动结构如下:

```
struct cfattach usb_ca = {
        .ca_devsize = sizeof(struct device),
        .ca_match = usb_match,
        .ca_attach = usb_attach,
};
struct cfdriver usb_cd = {
        .cd_devs = NULL,
        .cd_name = "usb",
        .cd_class = DV_DISK,
};
```

cfattach,cfdriver 是每个驱动必须定义的结构体,分别指向函数列表和设备表述。

configure 函数在每个平台的 tgt_machdep.c 中的 tgt_devconfig 函数中调用。

在 configure 函数中,config_rootfound 首先调用 config_rootsearch 找到设备并返回 cf 结构,然后调用 config_attach 分配设备结构,并挂接设备。

config_attach 除了调用设备驱动的 attach 函数外，还调用 TAILQ_INSERT_TAIL(&alldevs, dev, dv_list)将设备挂在设备列表里面。 可以通过 devls 命令列出。

PMON 初始化时 PCI 设备驱动遍历挂载的过程中,不断调用的函数及其功能描述如下:

- config_search(): 找到当前设备的子设备,并返回所找到的子设备的 cfdata 结构体。
- mapply(): 执行当前设备的 match 函数。设备的 match():判断是否符合当前设备的特性,如果符合则返回 1，否则返回 0。
- config_attach():给找到的子设备分配 device 结构体，插入到 alldev 链表，子设备如果是桥，就继续往下寻找设备，如果是设备，就对设备进行初始化和中断注册。
- 设备的 attach()： 设备进行初始化和中断注册。PMON 代码从 BSD 代码移植过来，设备驱动用文件来访问，普通设置用 open() 访问，网络设备用 socket()来访问。通过 open()访问 PMON 下的设备时，格式如表 2-1。

表 2-1 open()访问 PMON 文件例子

格式	说明
/dev/fs/ext2@wd0/boot/vmlinux	ext2/ext3 wd0 硬盘下 boot/vmlinux 文件
/dev/fs/fat@usb0/boot/vmlinux	fat usb0 U 盘下 boot/vmlinux 文件
/dev/disk/wd0	整个硬盘 wd0
/dev/tty1	tty1 设备
tftp://10.0.0.1/vmlinux	tftp 10.0.0.1 上的 vmlinux 文件

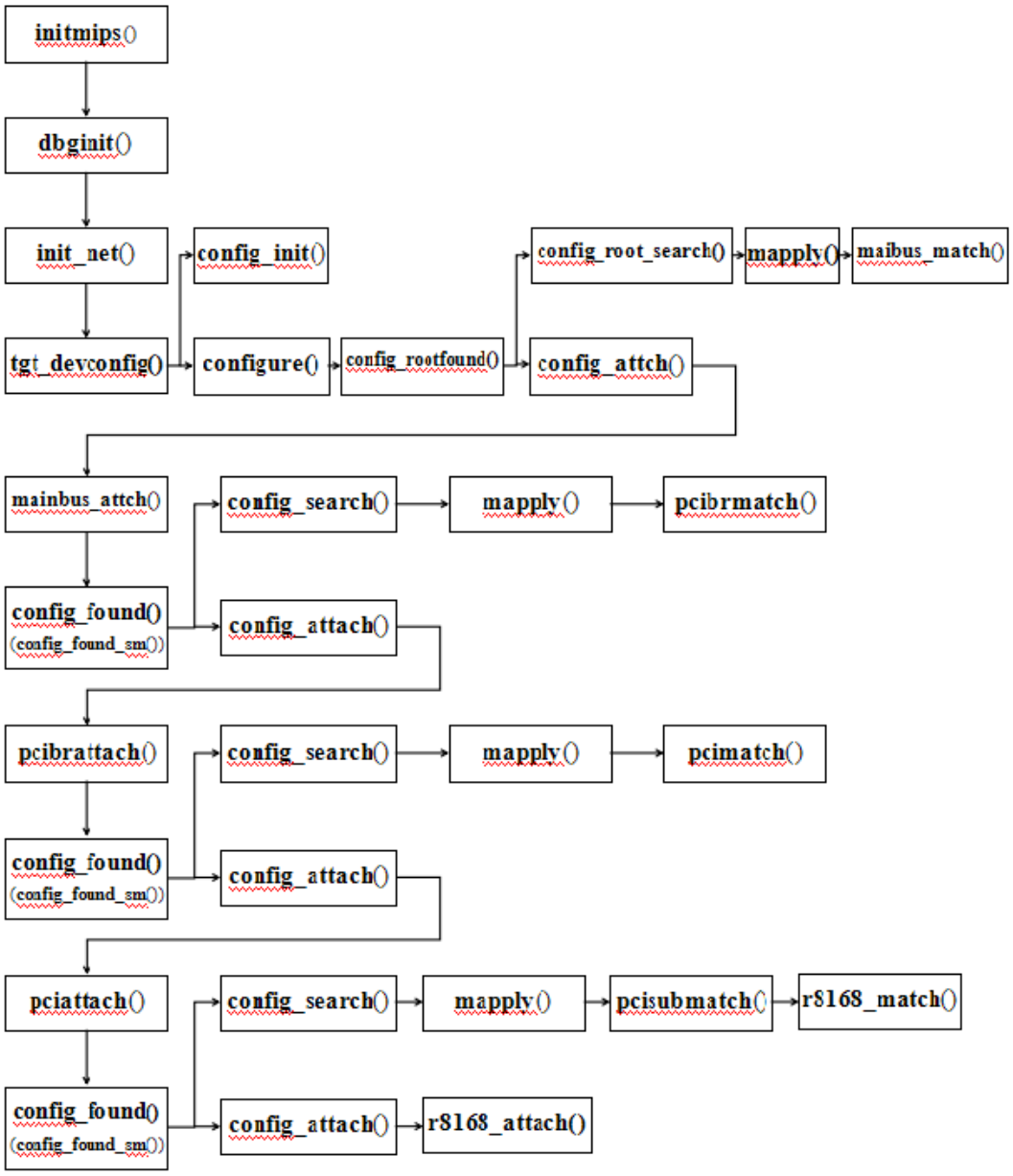


图2.3　　LS3A780E平台一个典型的设备树驱动加流程图

2.1.3　添加 82551 网卡驱动

PMON 中的网卡驱动程序遵循 UNIX PCI 设备驱动程序的构架,主要包含:初始化,数据的接收和发送两部分。但由于 PMON 中断处理机制的特殊性,在 PMON 中对中断是不做处理的,硬件产生中断后,只记录下中断标志状态,然后就立即返回,没有所谓的中断处理程序,所以 PMON 采用轮询的机制来代替中断,这使得网卡驱动的设计时,处理数据包收发时需要做相应的处理,PMON 下网卡中断的处理过程如图所示:

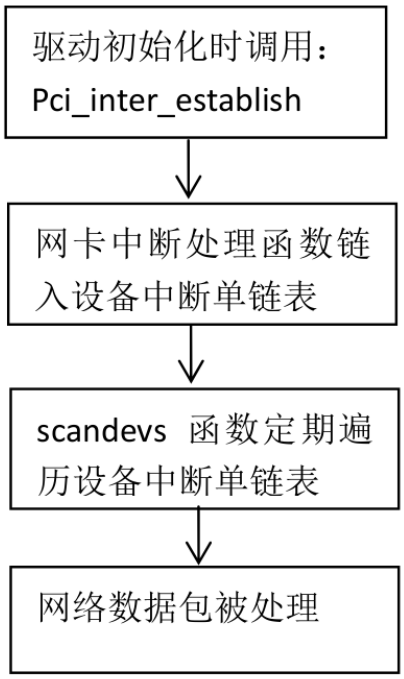


图 2.4 PMON 中一网卡设备的注册流程图

在 PMON 中,对于 PCI 设备，有一个重要的数据结构 struct cfdata cfdata，这个数组是根据具体平台的配置文件生成的，以下是配置文件的相关部分：

```
Mainbus0 at root
Localbus0 at mainbus0
Pcibr0 at mainbus0
Pci* at pcibr?
Fxp0 at pci? dev? Function?
rtk0 at pci? dev? Function?
Ohci* at pci? dev? Function?
Usb* at usbbus?
Pciide* at pci? dev? Function?
Wd* at pci? dev? Function?
Ide_cd* at pci? dev? Function?
```

这个部分描述了设备之间的链接关系，cfdata 是这个关系的数组表示。另外还有一个 PV 数组定义一个设备的父设备，每个节点的父设备都是一个数组，在设备 cfdata 结构中定义数组的开始。

上述配置文件中的 fxp0 表示的就是网卡设备。因为 fxp0 是 PCI 子设备，因此在

查找 PCI 子设备时,其 4 个子设备：fxp0，rtkO，ohci，pci-ide 的驱动也会在此时加载。查找设备有一个函数：config_found，其会调用 config_serch，config_search 从静态设备树 cfdata 中查找当前设备的子设备，然后对设备调用 mapply 函数，进行设备的匹配，如果设备存在，则会调用该设备的 ca_attach 函数来加载设备的驱动程序。这里找到网卡设备之后会调用 fxp_attach 函数。在设计的网卡驱动中，实现了协议栈将怎样通过网卡驱动来讲网络包发送出去，而网卡收到网络数据包之后怎样通过协议栈来做后续处理。 在 PMON 中，网卡中断通过查询来实现。fxp_attach 函数会调用 pci_intr_establish 将中断程序注册到查询列表 poll_list 上。网口要将数据包发送出去，就必须提供一个网络接口，以提供给发送函数。在这里，将网络数据包放在网络接口的 ifp→if_snd 队列中，然后启动 if_start 来开始发送。在网卡发送完一个包后，检查发送队列，如果有剩余则继续发送，直到发送完毕。然后通过函数 e1000_poll 来检查是否收到包，并进行收包处理。先进行缓冲区的清除，使能接收队列，查询是否有数据可接收，有则开始接收。无则继续查询并发出信息。

2.2 编译与开发环境

2.2.1 编译器的版本

PMON 的交叉编译工具链有两种，一种是 Gcc-2.95，另一种是 Gcc-4.4.0。2014 年6 月17 日之前的PMON 代码使用的工具链来自基于 Gcc-2.95 和Binutils-2.11 的工具链。该工具链在编译 C 代码和汇编代码时偶尔会将立即数加载指令序列（lui/addiu 或者 lui/daddiu）编错（变为 daddiu/addiu），从而导致生成的 PMON 无法正常启动。此后开始使用 Gcc-4.4.0 交叉工具链。

Gcc-4.4.0 与 Gcc-2.95 的主要区别：

1、 Gcc-2.95 对 64bit 下的 O32 ABI 支持与 Gcc-4.4.0 不一致，Gcc-2.95 直接支持 64bit 的整形变量（这与 O32 ABI 不一致），即对于如下的`int func(unsigned long long A0， unsigned long long A1)， Gcc-2.95 使用寄存器 a0 和 a1 传参；而 Gcc-4.4.0 则使用寄存器 a0，a1，a2，a3（分别对应 A0 的低 32 位与高 32 位，A1 的低 32bit 和高 32bit）与 O32ABI 一致；

2、 Gcc-4.4.0 使用更加严格的语法检查，包括但不限于：

a) 函数定义与函数原型声明不一致；

b) 函数的多重定义；

c) 条件编译中#ifndef 使用常数而非宏判断条件编译。

2.2.2 编译与开发步骤

PMON 的编译：跳转 PMON 源码对应板卡的编译目录下，如 zloader.3a780e 执行以下命令进行编译：

make cfg； make tgt=rom；

在线更新 PMON 方法：

可通过网络加载方式在线更新 PMON（前提：板卡可正常启动 PMON，板卡上有可使用的网络接口）。

具体步骤：板卡上电后启动到 PMON 的命令模式下，配置板卡网络 IP 与固件所在服务器的 IP 为同一网段 IP 地址，输入以下命令更新 PMON 即可：

load -r -f bfc00000 tftp://ip/gzrom.bin

2.3 调试方法与步骤

1、启动阶段：

2、上电取址：利用示波器或逻辑分析仪抓取取指信号。

3、串口初始化之前：借助于 Ejtag 扫描PC 与反汇编文件一起定位、GPIO 点灯或利用示波器或逻辑分析仪。

4、汇编语言级（架构级）：宏输出。

5、C 语言级（板级）：printf。

6、PMON 前期的初始化主要是打开通路的公共操作，没有对设备作具体的操作，通常可以采用读、写、比对的方式进行通路测试，如南桥蜂鸣器。

7、PMON 启动之后：利用调试命令 d\m\pcicfg\pciscan 等。

例子：判断当前接的串口是哪个串口?

对于单路系统，可以在 PMON 进入 Shell 后，分别输入：

```
m1 0xbfe001e0 0x41 // CPU0 UART0
m1 0xbfe001e8 0x41 // CPU0 UART1
m1 0xb80003f8 0x41 // SuperIO UART0
m1 0xb80002f8 0x41 // SuperIO UART1
```

对于双路系统,可以在 PMON 进入 Shell 后,分别输入：

```
pcs -3 //使用 64 位的 CPU 虚拟地址
m1 0x900010001fe001e0 0x41 // CPU1 UART0
m1 0x900010001fe001e8 0x41 // CPU1 UART1
```

观察哪一行输入后，会在终端打印输出 ASCII 码‘A’，即可判断当前是使用的哪个串口，进而在启动进入内核时传入正确的内核启动参数。

2.4 加速启动的办法

1、删减启动阶段部分打印信息：启动阶段输出的关于内存参数的打印信息， 关于窗口配置及HT 状态的打印信息以及PCI 设备总线枚举过程的调试信息。

2、测试内存及早期处理器版本 Bug 的代码。

3、显示 VBIOS 的模拟初始化，可采用正向初始化的方法。

4、PCI 设备驱动的加载过程，尽量去掉 PMON 中用不到的设备驱动。

5、BIOS 的图形用户界面的等待进入时间。

6、从硬盘加载内核的过程，尽量采用 DMA 的方式加载。

三、PMON 各组成部分的来源

PMON 整体上还是Unix 风格的：所有设备都被看成文件。

内存管理器：K & R Edition 2

网络协议栈：BSD4.3/net 设备

驱动程序：OpenBSD

显示初始化：int10(从 freex86 移植过来的）

四、3A780E 案例分析

4.1 使用环境变量作为配置项支持动态配置

1、早期：3A6U 板卡支持多种显示模式，如：VGA+DVI、DVI+DVI、VGA+LVDS 等3 种显示模式，每种显示模式对应一个二进行固件，当需要改变成另外一种显示模式时，需要重新刷固件，并重新启动。

2、现在：通过设置配置项，不用重复编译和烧写固件，支持动态配置切换显示模式和分辨率，使得三个二进制文件统一成一个，大大方便了客户的使用。

4.2 显示在模拟过程卡住问题

PMON 启动过程出现卡死，当出现以下现象时为显示卡死在模拟过程中。造成显示卡死问题的因素有哪些?

```
memorysize=8f10a3c8,base=b000000,sysMem=0,vram=8f6fdae0
in setup_int_vect!done!Found discrete graphics device: vendor=0x1002,
device=0x6822
Rom base addr: d0040000
VGA bios found
rom size is 64k
PCI data structure at offset 210
lock vga
starting bios emu...
ax=0,bx=0,cx=0,dx=0
```

4.2.1 IO 空间译码没有使能导致模拟卡住问题

PMON 中会检测有没有 PCIE 独立显卡，如果没有，则使能北桥集成显卡所在桥的 vga IO 译码使能位；如果有，则使能北桥 gfx 即0：2：0 的vga IO 译码使能位。目前发布的 PMON 源码只有这两种选择，如果 pcie switch 后端接有显卡设备，就需要自己打开显卡通路上显卡的父桥的 vga IO 译码使能位。

代码位置：在 x86emu/int10/generic.c 的vga_bios_init 的函数有如下代码：

```
val = _pci_conf_read(vga_bridge, 0x3c);
val |= 1 << 19;
_pci_conf_write(vga_bridge, 0x3c, val);
```

具体可以参考集成显卡的配置。

4.2.2 IO 空间分配导致模拟卡住问题

1、基础常识：

我们知道在 PMON 中，对于PCI 设备的资源（IO space/Mem space）分配采用由大到小，拓扑逆序的算法进行分配的。

在X86 的实模式下，内存与 IO 采用独立编址的方式进行寻址，而且 IO 空间总的大小为 2^16 bytes，即64KB，即IN/OUT 这类IO 端口寄存器访问指令的内存直接寻址空间是在 64KB 以内。

此外，video bios 做为BIOS 的一部分直接运行，其功能是初始化显卡的， 其内容是 X86 的汇编指令的二进制程序，即对显卡内部的端口寄存器进行初始化，那么此类指令必然是 IN/OUT 这类IO 端口寄存器访问指令。

2、猜想：

结合上述 3 点，我们得出的结论是：要保证显卡的端口寄存器的地址分配在 IO 空间的开始 64KB 范围以内，才可以保证显卡的正常初始化。

3、证明：

通过插入 3 个以上的 IO 卡（不能启动）及插入 2 个以内的 IO 卡（可以正常启动）的调试输出信息进行比较，可以看出，当显示的 IO BAR 的地址分配在 0x10000（64KB）以上的空间时，显示模拟肯定会卡死，而当显示的 IO BAR 的地址分配在 0x10000(64KB)以内的空间时，显示模拟可以正常通过，可以点亮显示器。

4、机理：

什么情况下，会把显卡的 IO BAR 空间分配到 64KB 以上：

1）接PCIE SWITCH 的时候，单个设备需要较大的 IO 空间。

2）通过 PCI 桥，扩展很多的 PCI 设备的时候，每个桥设备需要的 IO 空间至少分配 4KB,集显是最后分配 IO 空间，很容易算出接多少个 PCI 桥设备的时候，显示必定出问题。

5、结论：

我们现在的做法对 PMON 中关于PCIIO 资源分配的算法进行了修改具体思想是：

a) 无论扩展多少 IO 设备，优先保证独立显示设备的 IOBAR 第一个被分配。

b) 无论扩展多少 IO 设备，优先保证集成显示设备的 IOBAR 第二个被分配。

c) 其它 PCI 设备的 IO 资源分配的起始地址从 0x4000+0x2000 以后开始分配（预留 2*4KB 的空间给显示）。

通过大量的实验验证，这种修改方法有效地解决了扩展较多的设备时，PMON 卡死在显卡模拟处，导致 PMON 无法启动的问题。

具体的修改措施如下:

```
diff --git a/Targets/Bonito3a8780e/pci/pci_machdep.c
b/Targets/Bonito3a8780e/pci/pci_machdep.c
index 8c807cb..0db761a 100644
--- a/Targets/Bonito3a8780e/pci/pci_machdep.c
+++ b/Targets/Bonito3a8780e/pci/pci_machdep.c

@@ -118,7 +118,7 @@ _pci_hwinit(initialise, iot, memt)

#ifdef LS3_HT /* whd */
            pb->minpcimemaddr = BONITO_PCILO0_BASE;
            pb->nextpcimemaddr = BONITO_PCILO0_BASE+BONITO_PCILO_SIZE;
      -     pb->minpciioaddr = PCI_IO_SPACE_BASE+0x0004000;
      +    pb->minpciioaddr = PCI_IO_SPACE_BASE+0x0006000;
            pb->nextpciioaddr = PCI_IO_SPACE_BASE+ BONITO_PCIIO_SIZE;
            pb->pci_mem_base = BONITO_PCILO_BASE_VA;
            pb->pci_io_base = BONITO_PCIIO_BASE_VA;
diff --git a/sys/dev/pci/pciconf.c b/sys/dev/pci/pciconf.c
index 6794b6f..047d8cf 100644
--- a/sys/dev/pci/pciconf.c
+++ b/sys/dev/pci/pciconf.c
@@ -128,7 +128,7 @@ struct pci_bus *_pci_bushead3;
#endif
   struct pci_intline_routing *_pci_inthead;
   struct pci_device *vga_dev = NULL,*pcie_dev = NULL;
-
+vm_size_t vgaiobasetmp = PCI_IO_SPACE_BASE+0x0004000;
static void
print_bdf (int bus, int device, int function)
@@ -661,7 +661,19 @@ _pci_allocate_io(dev, size)
dev->bridge.secbus->nextpciioaddr = address;
#else
```

```
/* allocate upwards, then round to size boundary */
-     address=(dev->bridge.secbus->minpciioaddr+size-1)& ~(size - 1);
+     if ((_pci_make_tag(0,1,0) == dev->bridge.iospace->device->pa.pa_tag)
+
+
)
{
+     address = vgaiobasetmp;
+     vgaiobasetmp += 0x1000;
+     return (address);
+     }
+     else
+     {
+     /* allocate upwards, then round to size boundary */
+     address=(dev->bridge.secbus->minpciioaddr+size-1)& ~(size - 1);
+
}
+
address1 = address+size;
```

4.3 北桥上面接 PCIE Switch(如:8648,8619)常见问题

4.3.1 由于北桥 pcie 端口训练失败导致后面接的设备运行不正常

使用 pcie swtich 之后有时会出现北桥 pcie 端口训练失败的问题。这时对这个北桥 pcie 端口后面的设备，进行读写就会出现错误。以 8860 显卡为例，现象如下：

```
PCI bus 3 slot 0/0: vendor/product: 0x1002/0x6822 (display, VGA, interface: 0x00,
revision: 0x00)
PCI bus 3 slot 0/0: reg 0x10 = 0xc000000c    //正常时应为 reg 0x10 = 0xf000000c
PCI bus 3 slot 0/0: reg 0x18 = 0xfffc0004
PCI bus 3 slot 0/0: reg 0x20 = 0xffffff01
PCI bus 3 slot 0/0: reg 0x30 = 0xfffe0000
```

由此还会导致 PMON 卡死，现象如下：

```
Keyboard succesfully initialized.
on!
ON2000 MIPS Initializing. Standby...
MDDDDDDDS Initializing. Standby...
SDDDMDDMizing. Standby...
```

解决方法：增大北桥 pcie 训练的延时。

以 3A9780E 为例，代码修改如下：

```
diff --git a/Targets/Bonito3a9780e/pci/rs780_cmn.c
b/Targets/Bonito3a9780e/pci/rs780_cmn.c
index f440a7e..5578912 100644
--- a/Targets/Bonito3a9780e/pci/rs780_cmn.c
+++ b/Targets/Bonito3a9780e/pci/rs780_cmn.c
@@ -510,6 +510,8 @@ void set_pcie_reset()
u8 PcieTrainPort(device_t nb_dev, device_t dev, u32 port)
{
        u16 count = 5000;
        + u16 repeat_count = 15;
        + u16 repeat_bef = 1;
        u32 lc_state, reg;
        u8 current, res = 0;
        u32 gfx_gpp_sb_sel;
@@ -606,8 +608,30 @@ u8 PcieTrainPort(device_t nb_dev, device_t dev, u32 port)
}
                        break;
        default:      /* reset pcie */
 - res = 0;
- count = 0; /* break loop */
+ if(repeat_bef)
 + {
 + repeat_bef--;
 + delay(2000000);
 + count++;
+ }
+ else
+ {
+ printk_info("repeat_count = %d\n",repeat_count);
+ repeat_bef++;
```

```
+ if(repeat_count--)
+ {
+                                   set_pcie_reset();
+                                   delay(1);
+                                   set_pcie_dereset();
+                                   count++;
+                                   }
+                                   else
+                                   {
+                                   res = 0;
+                                   count = 0; /* break loop */
+                          }
+               }
            break;
            }
}
```

4.3.2 在 PCI 扫描处卡死 PMON 跑飞了

```
PCI bus 0 slot 20/3: vendor/product: 0x1002/0x439d (bridge, ISA, interface: 0x00,revision: 0x00)
PCI bus 0 slot 20/4: vendor/product: 0x1002/0x4384 (bridge, PCI, interface: 0x00,revision: 0x00)
on!
PMON2000 MIPS Initializing. Standby...
MDDDDDDDS Initializing. Standby...
SDDDMDDMizing. Standby...
SSDDD.Rndby...
```

出现这种现象一般是在 pcie switch 后端桥的个数超过 7 个时出现，当然不同板卡在 PCI 扫描处卡死的位置以及后端桥的个数可能不同。如果关掉北桥自身的一些桥或者将北桥的GPPSB 配成X4,gpp 配成X2,gfx 配成X16 都会对这种现象有所影响。造成这种现象的虽然 PCI 总线个数最大是 256，但是通常桥的个数不会太多，所以 PMON 中对桥的个数做了限制，默认是 16 个。如果多于 16 个桥设备就会导致内存溢出，从而导致写覆盖，最终导致 PMON 跑飞。

解决方法：在 pci_machdep.c 中有一个_pci_bus 的结构体数组变量，将这个数组的长度更改为需要的值即可解决这种问题。

3A780E: Targets/Bonito3a780e/pci/pci_machdep.c

4.4 RS780E 显存参数调试

关于 PMON 下显存调试一类是现有显存颗粒的基础上调整时钟频率，一类是因板卡更换显存颗粒型号，需要整体更改显存寄存器配置参数。前者可根据北桥手册修改相关寄存器，后者在前者的基础上需要借助 AMD 提供的工具 780CIM， 生成其相关寄存器的数据，此部分无手册可依。现对 780CIM 的使用方法做出详细说明。

780CIM 运行环境：dos

780CIM 相关文件：TOOLS/sptbllgen.exe 执行程序

4.4.1 780CIM 环境设置

1. 找一台 Windows 系统环境的 PC 机。
2. 将 780CIM 程序文件夹拷贝到本地磁盘。具体目录自由设置。

说明：780CIM 的可执行程序存放在 TOOLS 目录中，显存配置文件存放在 DOC 中，注意显存配置文件的格式为 conf.spprj。

780CIM 工具目录内容见示例图 4.1:

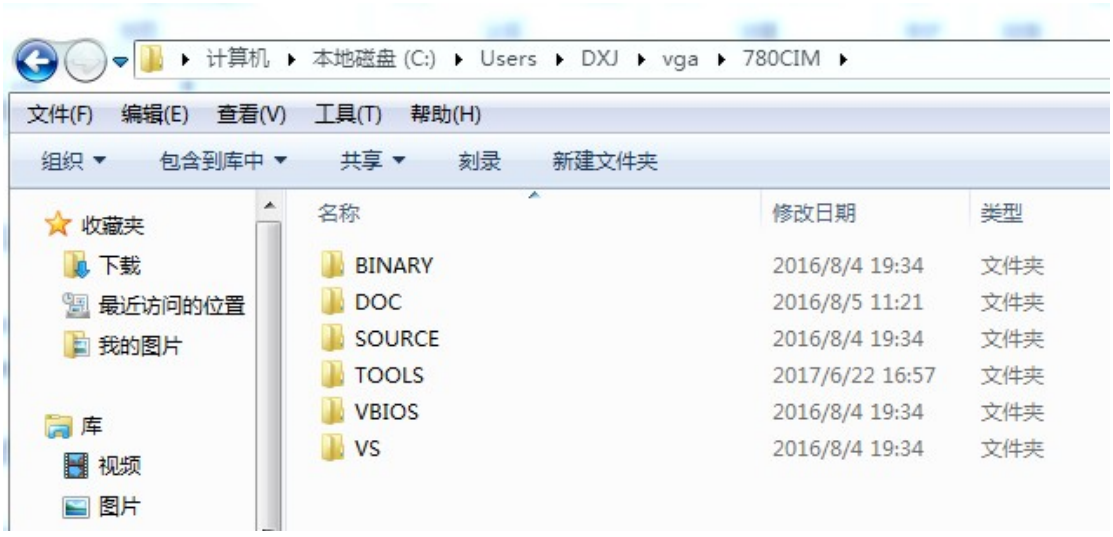


图 4.1 780CIM 在 Windows 系统机器中位置及 780CIM 目录内容

4.4.2 创建显存配置文件

在 780CIM 的 DOC 目录下新建一个显存配置文件，根据配置文件说明按照显存颗

粒配置手册进行配置。

4.4.3 配置文件格式及说明

文件 conf.spprj：

```
//Memory Model
MemoryVendor="HYB18T51216182F_2S"
//标注显存颗粒型号
//Memory clock support map
//MemoryClockMap[0] - 200Mhz
//MemoryClockMap[1] - 266Mhz
//MemoryClockMap[2] - 333Mhz
//MemoryClockMap[3] - 400Mhz
//MemoryClockMap[4] - 533Mhz
//MemoryClockMap[5] - 667Mhz
MemoryClockMap=0x3F     //显存频率设置—400MHZ

   //Memory type

   //MemoryType= 0 - "DDR2"

   //MemoryType= 1 – "DDR3"
   MemoryType = 0          //设置显存颗粒类型—DDR2
   //-------------------------------------------------------------

   //Memory chip organization

   // Memory          Memory chip       Bank addr      Row addr      Column addr      Num of          FB
   Size
   // chip  Size      organization          bits           bits          bits                  chip
   //MemoryOrganization = 0       256 Mbit      16M x 16       2        13            9          1          32
   MByte
   //MemoryOrganization = 1       256 Mbit      32M x  8       2        13           10          2          64
   Mbyte
   //MemoryOrganization = 2       512 Mbit      32M x 16       2        13           10          1          64
   Mbyte
   //MemoryOrganization = 3       512 Mbit      32M x 16       3        12           10 9        1       64 Mbyte
   //MemoryOrganization = 4       512 Mbit      64M x  8       2        14           10 6        1      128 Mbyte
```

//MemoryOrganization = 5 512 Mbit 64M x 8 3 13 10 11(B) 2 128 Mbyte

//MemoryOrganization = 6 1024 Mbit 64M x 16 3 13 10 11(B) 1 128 Mbyte

//MemoryOrganization = 7 1024 Mbit 128M x 8 3 14 10 7/12(C) 2 256 Mbyte

//MemoryOrganization = 8 2048 Mbit 128M x 16 3 14 10 7/12(C) 1 256 MByte

MemoryOrganization = 2 //根据显存颗粒手册设置显存的 siz，行列地址线配置，选择对应序号 ， 此处设置 size=512Mb，row=13 col=10

//Memory Spec Parameter

//以下为显存颗粒的对应的各个延时参数，按照显存颗粒的 datasheet 最小值设置即可

cl = 6

tRCD=15

tRP=15

tRTP=7.5

tWR=15

tRRD=10

tRAS=45

tRC=60

tRFC=105

TREFI=7.8

tWTR=7.5

tCCD=4

tCKE=3

tXP= 2

tMRD=2

tFAW=0

```
tRPALL=0
//DDR2
tXARDS=8
tAXPD=8
//DDR3
tXPDLL=24
tZQCL=256
tZQCS=64
tZQCI=128
```

4.4.4 AMD 工具 780CIM 使用方法

打开dos 环境：在Windows 桌面点击开始菜单，在弹出的窗口底端搜索程序和文件栏输入 cmd，回车后即可打开 dos 环境。

进入780CIM程序目录，命令为：

cd .\vga\780CIM\TOOLS

示例图如图 4.2 所示：

图 4.2 DOS 环境下切换 780CIM 目录

运行显存配置可执行程序，命令格式：sptblgen.exe [option] config.spprj

option:/asm -assembly output

/c -C output

/bin -binary output

运行 780CIM 可执行程序后结果如示例图 4.3 所示：

图 4.3 780CIM 显存配置参数结果图

根据生成出显存配置修改相应寄存器，显存配置文件：Targets/Bonitoboard/pci/780_gfx.c

4.5 3A6U 计算机模块认不出 82574 网卡

1、故障现象

（1）PMON 命令行 devls 看不到网卡设备名称，pciscan 扫描可以识别到设备。

（2）查看 PMON 打印信息，在驱动加载处报 checksum error 的错误。

（3）通过 PMON 命令行读出网卡上 nvm 的内容，与写入的值进行比较，MAC 地址是随机生成的，checksum 的值是计算所得，其他的内容和数组一样。

（4）nvm 以双字节为单位求和得出结果，低 16 位不为 0xBABA。

2、机理分析

（1）由于 nvm 以双字节为单位求和，得出结果的低 16 位不为0xBABA，根据规则这种情况应该认为 nvm 内容是无效的，不能用这些数据初始化网卡。

（2）当NVM checksum is invalid，网卡的驱动在加载过程中异常退出，不会注册成功设备节点，所以最后就是 devls 看不到网卡设备。但是由于硬件本身没有问题，pciscan 是可以扫描到设备的。

3、措施

（1）在网卡驱动使用数据之前检查 checksum，如果错误就进行修改，然后再检查一次，如果正确就继续执行，否则跳出驱动。

（2）如果进行修改，在环境变量里记录修改的次数和最后一次修改的时间，这样可以抓取 NVRAM 值异常的情况。

4.6 3A2H 计算机模块 PMON 下修改分辨率

首先需要了解的是 PMON 阶段的显示都是在 framebuffer 模式下。下文以 3A20002H 为例，3A2H、3A30002H 同样适用。

第一步、根据实际需求配置色深，2H 显示控制器目前支持 RGB444、RGB555、RGB565、RGB888 四种色深，修改配置文件

Targets/Bonito3a82h/conf/Bonito.3a2h（一般默认 RGB565，不需要修改）

option CONFIG_VIDEO_16BPP

第二步、配置文件 bonito.3a2h 中默认 framebuffer 分辨率为 X800x600，该宏作用于文件 cfb_console.c，直接设定了 X Y 分辨率，修改为你想要的分辨率， 如：3840*1200

option X3840x1200

第三步、除了设置分辨率还需要设置该分辨率的屏幕尺寸、特定刷新率下的像素时钟等，查看文件 Targets/Bonito3a82h/dev/dc.c 中 vgamode[]中是否有你需要的分辨率，如果没有可以通过 gtf 命令计算出来进行添加。

```
usage: gtf x y refresh [-v|--verbose] [-f|--fbmode] [-x|--xorgmode]
x : the desired horizontal resolution (required)
```

```
y : the desired vertical resolution (required)
refresh : the desired refresh rate (required)
-v|--verbose : enable verbose printouts (traces each step of the computation)
-f|--fbmode : output an fbset(8)-style mode description
-x|--xorgmode : output an Xorg-style mode description (this is the default if no mode description is requested)
>>$ gtf 3840 1200 60
# 3840x1200 @ 60.00 Hz (GTF) hsync: 74.52 kHz; pclk: 386.31 MHz
Modeline "3840x1200_60.00" 386.31 3840 4096 4512 5184 1200 1201 1204 1242 -HSync +Vsync
```

Linux 下如果无法通过 EDID 信息读取可手动生成一组分辨率信息。

drivers/video/ls2h_fb.c 文件结构体填充：

```
static struct fb_var_screeninfo ls2h_fb_double_1920_1200 __initdata = {
.xres = 3840,
.yres = 1200,
.xres_virtual = 3840,
.yres_virtual = 1200,
.xoffset = 0,
.yoffset = 0,
.bits_per_pixel = 32,
.red = { 11, 5 ,0},
.green = { 5, 6, 0 },
.blue = { 0, 5, 0 },
.activate = FB_ACTIVATE_NOW,
.height = -1,
.width = -1,
.pixclock = 1000000/386.31,
.left_margin = 5184 - 4512,
.right_margin = 4096 - 3840,
.upper_margin = 38,
.lower_margin = 1,
.hsync_len = 4512 - 4096,
.vsync_len = 3,
.sync = 3,
.vmode = FB_VMODE_NONINTERLACED,
};
```

第四步、文件 fb/cfb_console.c 文件中 console_buffer 数组,和屏幕的字符打印相关。该区域大小的申请根据分辨率和字符大小计算得到。

```
#elif defined(X1024x768)
char console_buffer[2][49][129]={32}; //128*48->1024x768
```

一个字符大小是 8*16*2B,那么 1024*768 分辨率为 128*48 个字符,每个像素 2Byte。

4.7 CPU 输入时钟需要与 LPC 时钟同源同相

1、 故障现象：

某单位自研板卡上，通过 LPC 总线接口连接的 SuperIO 芯片中的串口偶发不正常 ，PS/2 基本不可用。

查看初始化代码是否写入寄存器，d1 0xbff003f88

1 有时可以写入，返回 43 c1 xx

2 有时无法写入，返回全 0

3 配置被写入时，向数据端口寄存器写入 ASCII 有回显

2、机理分析：

某单位自研板上没有时钟 Buffer，分别用两个晶振分别提供时钟，无法保证 CPU 输入时钟与LPC 时钟同源同相，导致LPC 总线接口下的设备工作不正常。

3、措施：

在板卡上进行飞线，使 CPU 输入时钟需要与 LPC 时钟同源同相后，PS/2 可用，串口工作正常。

板卡硬件设计时参考龙芯开发板的设计，用时钟 Buffer 芯片来做同步，保证 CPU 输入时钟需要与 LPC 时钟同源同相 。

五、LS3A-7A 代码简介

芯片 ＋7A 桥片单双路板卡的初始化代码和部分独有设备初始化代码。集成到 pmon-loongson3 主分支,便于 pmon 下共用代码升级方便且管理方便。

1) pmon 的管理基于 git。
2) pmon 的源码发布地址: http://cgit.loongnix.org
3) 下载源码：

git clone git://cgit.loongnix.org/pmon/pmon-loongson3.git

5.1 PMON 编译

PMON 的编译环境是交叉编译，即在 X86 机器上进行编译，编译出 MIPS 版本的 PMON 二进制。系统要求为 X86 Linux 操作系统。Ubuntu， Debian，Fedora，CentOS 都可以使用。系统中需要先安装一些开发包 flex，bison，xutils-dev。 Fedora 系统使用 yum install 命令安装，Ubuntu、Debian 系统使用 apt-get install 命令安装。

(1) 安装编译器

在 X86 Linux 机器上，下载编译器 gcc-4.4.0，保存在任意位置。

编译器下载地址：

http://ftp.loongnix.org/toolchain/gcc/release/CROSS_COMPILE/gcc-4.4.0-pmon.tgz

```
# mkdir  -p  /usr/local/comp/mips-elf/
# tar  -zxvf  gcc-4.4.0-pmon.tgz  -C  /usr/local/comp/mips-elf/
```

(2) 设置环境变量：

```
$ vi  ~/.bash_profile
```

在文件末尾添加下面三行配置：

```
export  LD_LIBRARY_PATH=/usr/local/comp/mips-elf/gcc-4.4.0-pmon/lib:
export  CROSS_COMPILE=mipsel-linux-
export  PATH=/usr/local/comp/mips-elf/gcc-4.4.0-pmon/bin/:$PATH
```

(3) 执行下列命令进行编译：

```
cd  zloder.3a3000_7a/
make  cfg
make  tgt=rom
```

(4) 编译成功后，生成 PMON 的二进制文件 gzrom.bin。

(5) 在机器上更新 PMON

龙芯 PMON 支持在线更新功能，即在本机上启动 PMON，通过命令行更新上面编译出来的二进制。在线更新命令如下，其中 URL 指向 PMON 二进制所在的位置，龙

芯支持通过 U 盘和网络两种形式。

PMON> load -rf 0xbfc00000 URL

附加说明：

如果在编译时提示“缺少 pmoncfg 文件”，通过以下步骤解决，再次编译，应该就可以正常通过了。

$ cd tools/pmoncfg

$ make 此步骤将生成 pmoncfg 文件

cp pmoncfg /usr/bin （以 root 身份执行此命令）

5.2 注意事项

(1) 因为 3A3000+7A 的单双路支持代码为同一分支，默认代码为支持单路，如使用双路板卡需修改：

Targets/Bonito3a3000_7a/conf/Bonito.3a3000_7a 文件，

```
#option          MULTI_CHIP #for 3a3000_2w_7a
option             BOOTCORE_ID=0
#option          RESERVED_COREMASK=0xff00 #for 3a3000_2w_7a
option             RESERVED_COREMASK=0xfff0 #for 3a3000_7a
option             SHUTDOWN_MASK=0x0000
option             LOONGSON_3ASINGLE #for 3a3000_7a
#option          LOONGSON_3ASERVER #for 3a3000_2w_7a
```

将其中注释为 for 3a3000_7a 的选项关闭，打开注释为 for 3a3000_2w_7a 的选项。

(2) 7A 板卡支持过程中，对公用代码的修改很大一部分为添加 7A 设备的支持，主要添加 pci 扫描设备加载的支持和部分 bug 修正，

a) 调试过程中发现 SATA 驱动的代码地址处理存在异常且不支持多个设备，对其进行了修正并添加多个设备的支持；

b) USB 调试过程中对代码编写异常的问题进行了修正；

c) 修正了 pci 扫描函数的代码异常问题，当发现 7A 内部的 PCIE 桥时，下级总线只能扫描 0 号设备，不再继续扫描其它设备；

d) 添加独立显卡的支持，修改公用文件的部分因代码本身存在 bug，部分是添加支持的过程中需要进行设置。

修改集成显示的分辨率不再对 conf 文件进行修改，需修改

Targets/Bonito3a3000_7a/dev/dc.c 文件中的宏定义的值：

FB_XSIZE

FB_YSIZE

文件./fb/cfb_console.c 中变量的值:

ScreenLineLength;

ScreenDepth;

ScreenHeight ;

其中 FB_XSIZE=ScreenLineLength/((ScreenDepth+1)/8)

FB_YSIZE=ScreenHeight;

例如 设置 800*600 分辨率：

#define FB_XSIZE 800

#define FB_YSIZE 600

ScreenLineLength = 1600;

ScreenDepth = 16;

ScreenHeight = 600

5.3 LS7A 初始化相关代码说明

5.3.1 启动初始化代码

在 start.S 中需要对 7A 进行简单初始化。按照代码的次序包括：

(1) 地址映射。相关文件：loongson3_ht1_32addr_trans.S

将桥片的 IO 空间、配置访问空间、部分 MEM 访问空间映射到 32 位地址段，通过配置 3A 的一级地址映射窗口实现。

(2) 配置 7A 的 MISC 设备块和 confbus 的地址。

(3) 建立 3A 和 7A 的 HT 链路连接。相关文件：ls3a7a_setup_ht_link.S

配置 HT 的频率、数据宽度、协议版本。这一步分为两个步骤：

a）写入配置参数。（config_ht_link）

b）复位总线使得配置参数起作用。(reset_ht_link)

(4) 配置 3A 和 7A 的 HT 窗口。相关文件：ls3a7a_ht_init.S

分别配置 3A 和 7A 的 HT 控制器的接收、发送窗口等。

(5) 7A 的初始化，相关文件：ls7a_init.S、ls7a_config.S

包括：7A 的相关配置、PLL 频率配置、中断和 HPET 的固定地址配置、PCIE、SATA、USB 的 PHY 初始化、显存初始化。

5.3.2 调试相关

start.S 中包含了一个简单的回环测试来验证 HT 通路的稳定性。

相关文件：ls7a_dbg.S

以上步骤 1-3 一般放在 cache 初始化之后，内存初始化之前进行，以便当 HT 通路出现连接异常时通过看门狗复位来快速重启。步骤 4、5 放在内存初始化之后。

5.3.3 相关宏定义说明

7A 相关的头文件包括：

（1） ls7a_config.h

板卡相关的配置文件。用来定义 3A 和 7A 连接的 HT 链路、以及板卡使用的 7A 设备功能。

禁用某个 PCIE 端口配置：默认为 0.

#define LS7A_PCIE_F0_DISABLE 0

#define LS7A_PCIE_F1_DISABLE 0

#define LS7A_PCIE_H_DISABLE 0

#define LS7A_PCIE_G0_DISABLE 0

#define LS7A_PCIE_G1_DISABLE 0

LS7A_SATA*_DISABLE：禁用某个 SATA 端口。默认为 0.

LS7A_USB*_DISABLE：禁用某个 USB 控制器。USB0 包含端口 0-2，USB1 包含端口 3-5. 默认为 0.

LS7A_LPC_DISABLE：禁用 LPC 接口。默认为 1.

LS7A_GMEM_CFG：是否初始化 7A 自带的显存。默认初始化。

CHECK_HT_PLL_MODE：检查 HT 的配置引脚是否使能软件配置 HT 频率。推荐板卡使能软件配置。如果板卡没有使能软件配置，PMON 在启动过程中会短暂延迟一会儿，并通过串口打印提示信息。

（2） ls7a_define.h　　7A 芯片相关的头文件。

（3） ht.h　　3A 和 7A 的 HT 相关头文件。

5.4 LS7A 问题汇总

5.4.1 PCIE 问题

5.4.1.1 集成的 PCIE 桥桥类型返回错误

桥片内集成的 PCIE 桥的 PCI 设备头的 subclass code 正确值应为 0x04(表示 PCI 类型的桥),但本桥片将其实现成了 0x00(表示 Host 类型的桥)。

解决方式:软件修复。当发现读配置头访问时,如果访问地址是 bus 0 的设备 9 到 20 且地址为 0x8 时,直接返回 0x06040001,而不返回硬件读取的值。

5.4.1.2 访问 7A 桥片后不存在的设备问题

PCIE 协议规定时返回 0xffffffff 的相应，但是，实际返回的是 0 值，此问题影响到 7A PCIE 的下级总线，这会导致只使用 0xffffffff 判断设备不存在的代码，误认为有设备。

解决方法：软件修复。更改 7A 的读函数，如果读设备 ID 时返回为 0,代码更改为 0xffffffff。

5.4.1.3 接 PCIE switch 无法识别 switch 后面的设备

对于桥片内集成的 PCIE 桥,扫描下级总线时,当扫描非 0 号设备时,本应返回无效值,但本桥片会返回 0 号设备的配置头,造成 0 号设备被重复发现，又由于 PCIE 是一对一的，所以最初的代码只扫描 0 号设备，但是 7A 本身的这个问题不会影响到 7A PCIE 的下级总线，所以通过改进代码解决以上现象。

解决方式: 更新 PMON 代码到最新版本。最新代码解决方法遍历总线号非 0 的某个总线下设备号大于 0 的设备时，先读设备 0，设备 15，设备 31 的配置头，如果相等不等于-1 并且不等于 0，在这个总线上的 0/15/31 号设备不是相同的情况下，只有 7A 的 PCIE 下级总线才会出现这种情况，由于 PCIE 是一对一的，所以 7A PCIE 直连的设备不存在设备号大于 0 的情况，所以不理会，不满足这种情况就对这个设备进行

配置访问。这样做有个前提，就是这个总线上的 0/15/31 号设备不是相同的。

5.4.1.4 接商用的 I210 网卡内核出现大量未处理中断

7A PCIE 插槽后面接商用的 I210 网卡，起内核后出现大量未处理的中断 ，这是由于在 PMON 扫描过程中 PCIE 控制器会产生了一个 verdor msg 的中断状态。起内核后一直没有被处理。

解决办法：PMON 加载内核前清一次 PCIE 的中断状态。

参考代码如下：

表 5-1 清 PCIE 中断状态代码改动

```
9 --- a/pmon/common/main.c
10 +++ b/pmon/common/main.c
11 @@ -75,7 +75,7 @@ unsigned int show_menu;
12   #include "cmd_more.h"          /* Test if more command is selected */
13
14   #include "../cmds/bootparam.h"
15 -
16 +#include <dev/pci/pcivar.h>
17   void print_mem_freq(void);
18   extern int bios_available;
19   extern int cmd_main_mutex;
20 @@ -452,6 +452,25 @@ main()
21   #ifdef ARB_LEVEL
22       save_board_ddrparam(0);
23   #endif
24 +#if defined(LS7A)
25 +    pcitag_t tag;
26 +    pcireg_t id;
27 +    bus_addr_t membase;
28 +    unsigned long pcie_port_ctl_reg = 0;
29 +    int dev = 9;
30 +    for(dev = 9;dev <= 20;dev++)
31 +    {
32 +        tag = _pci_make_tag(0, dev, 0);
32 +        tag = _pci_make_tag(0, dev, 0);
33 +        id = _pci_conf_read(tag, PCI_ID_REG);
34 +        if (id == 0 || id == 0xffffffff) {
35 +            continue;
36 +        }
37 +        pci_mem_find(NULL,tag,0x10,&membase,NULL,NULL);
38 +        pcie_port_ctl_reg = membase | BONITO_PCILO_BASE_VA;
39 +        *(int *)(pcie_port_ctl_reg + 0x1c) = 0xffffffff;
40 +    }
```

```
41 +//clear 7a pcie interrupt
42 +#endif
```

5.4.1.5 PCIE 设备能正常识别但使用异常

解决办法：

检查 PCI 配置头空间的 PCIE MPS(Max Payload Size)、MRRS（Max Read Request Size）配置(具体寄存器可查看 PCIE 协议手册)。7A 的 pcie 控制器 payload size 最大只支持 256 字节，对于超出 256 字节 payload size 的设备需要作出修正，强制配置成 256 字节。因此，需要将设备的 MPS、MRRS 设置为不大于 1 的值。

PCIE 控制器的初始化流程可能有问题，更新 PMON 到最新版本。

5.4.1.6 PCIE 设备使用 MSI 中断异常

PCIE 设备使用 MSI 中断异常，比如丢中断 。

解决办法：有些 PCIE 设备只支持 32 位的 MSI 中断地址，不支持 64 位 MSI 中断地址。对于这些设备，内核需要使用 32 位的 MSI 中断地址，而不是 64 位的 MSI 中断地址。

5.4.1.7 PCIE 设备不能正常识别

解决方法：

- 检查硬件 PCIE_*_PRSNTn 的状态，正确标准为：有设备时，对应的 PRSNTn 引脚应为低电平；没有设备时，对应的 PRSNTn 引脚应为高电平。
- 更新 PMON 到最新版本。
- 尝试增大 PCIE 初始化过程中的延迟。

5.4.2 7A USB 控制器问题

USB 设备(Device 4 和 Device 5)的 OHCI(Function 0)和 EHCI(Function 1)控制器的 PCI 配置头的 Memory Space Enable 控制位弄反了。即:EHCI 的 Memory Space Enable 位控制 OHCI 的 Mem 空间使能,而 OHCI 的 Memory Space Enable 位控制 EHCI 的 Mem 空间使能。

解决办法：软件修复。

5.4.3 GMAC DMA64 问题

在 64 位 DMA 模式下，GMAC 的发送描述符基地址的高 32 位寄存器(0x1094)只能读，不能写。

解决方法:软件修复。通过写如下地址{0x10a8[31:8], 0x1068[7:0]}，可以实现对高 32 位寄存器(0x1094)的写入。

5.4.4 电源相关问题

现象：PMON 启动卡死在读取 7A ID 处，打印 bridge CHIP ID check failed!跳过读取 7A ID 寄存器，PMON 又会随机卡死在访问 7A 的某个设备处，如网络 USB SATA 等。

解决方法：将 7A 的 HT PLL 提高到 1.25V。

5.5 中断

5.5.1 中断用法差异

LS7A 的中断控制器进行了部分重新设计。将中断使能/关闭（set/clr）寄存器修改为中断屏蔽寄存器（mask），不能直接关闭对应中断了，要先读回，然后屏蔽。中断清除寄存器（clr）仅用来清除边沿触发的中断（目前 7A 不需要，除非将 GPIO 设置为边沿触发），同时修正了 IRR，ISR 寄存器的功能。

5.5.2 MSI 中断

7A 的 MSI 中断可由中断控制器产生，也可由 PCIE 控制器直接发往 HT。因此，对于这两种中断要分别处理，内核为 7A 中断控制器的中断源分配 64-127 号中断，为 PCI-MSI 分配 0-55 号中断。

由于 3A-7A 系统默认采用 MSI 的方式，其中 PCIE 设备支持 MSI 中断，可以不经过 7A 上的中断控制器直接发送到 HT 中断向量上，而除 PCIE 设备之外的所有中断必须经过 7A 的中断控制器后，再发送到 HT 的中断向量上。

5.5.3 中断分配

由于桥片上 PCIE 设备的中断号在 0～55 之间动态分配，所以要保留此段中断号给 PCIE 设备使用，这部分的中断号分配在内核中完成，PMON 无需处理。

基于 7A 中断特点做以下约定：

表 5-2

中断名称	中断号	说明
LOONSON_PCIE_MIS_BASE	0	PCIE 设备的 MSI 中断
MIPS_CPU_IRQ_BASE	56	CPU 内部中断
LOONSON_UART_IRQ	MIPS_CPU_IRQ_BASE+2	CPU UART 中断
LOONSON_TIMER_IRQ	MIPS_CPU_IRQ_BASE+7	CPU TIMER 中断
LOONGSON_BRIDGE_IRQ_BASE	64	LS7A 桥片上设备的中断号起始地址
LOONGSON_BRIDGE_IRQ_RESERVE	128	LS7A 桥片上设备的保留中断号起始地址

内核中配置 LS7A 上设备通过 7A 的中断控制器发送中断到 HT 的 64～127 号中断向量上，对应设备的中断号为 64～127，具体对应关系如下：

表 5-3

中断名称	中断号	中断名称	中断号
RESERVED	64~71	pcie_h_lo	102
Uart[3:0]	72	pcie_h_hi	103
I2c[5:0]	73	pcie_g0_lo	104
RESERVED	74~75	pcie_g0_hi	105
gmac0_sbd	76	pcie_g1_lo	106
gmac0_pmt	77	pcie_g1_hi	107
gmac1_sbd	78	toy[0]	108
gmac1_pmt	79	toy[1]	109
sata[0]	80	toy[2]	110
sata[1]	81	acpi_int	111
sata[2]	82	usb_0_ehci	112
Lpc	83	usb_0_ohci	113
RESERVED	84~87	usb_1_ehci	114
pwm[0]	88	usb_1_ohci	115
pwm[1]	89	rtc[0]	116
pwm[2]	90	rtc[1]	117
pwm[3]	91	rtc[2]	118
Dc	92	hpet_int	119
Gpu	93	ac97_dma[0]	120

Gmem	94	ac97_dma[1]	121
Thsens	95	Ac97/hda	122
pcie_f0_0	96	gpio_hi	123
pcie_f0_1	97	gpio[0]	124
pcie_f0_2	98	gpio[1]	125
pcie_f0_3	99	gpio[2]	126
pcie_f1_0	100	gpio[3]	127
pcie_f1_1	101		

5.6 地址说明

5.6.1 地址概述

龙芯 7A1000 桥片的地址空间是 mips 架构统一地址空间的一部分。

（1） 从 CPU 的视角来看 ——即 CPU 可访问的设备地址空间，桥片的地址空间包括 3 部分：配置空间、PCI I/O 空间和 PCI MEM 空间。桥片的地址空间与 PCI 定义的地址空间形式相同。

1. 配置空间:该地址空间用来访问桥片内部设备(包括通过 PCIE 总线扩展的设备)的配置头,其地址组成符合 PCI 配置访问的地址组织形式;

2. I/O 空间:该地址空间用于访问 PCI 协议定义的 I/O 地址空间。在桥片中只有 PCIE 有这段地址空间,用于通过 I/O 类型的请求访问 PCIE 控制器的下游设备。

3. MEM 空间:除了以上两种地址空间之外的所有地址空间为 MEM 空间。

软件在访问时，需要通过处理器的一级 XBAR 将它们映射到 HT1 的地址空间段内或者直接添加 HT1 的地址空间偏移到该访问地址上。

注*:除了地址段 0x0f000000~0x0fffffff。这段地址不可用作桥片设备地址空间。

（2） 从 DMA 访问的视角来看——即桥片内部设备主动发起的访存地址空间，可使用的地址空间包括：处理器的内存空间和桥片的显存空间。

对于 4 个结点的系统,DMA 的地址空间必须位于结点地址空间的低 256GB 以内，这样桥片可以直接访问最多 4 个结点的内存。桥片内部可以发起 DMA 操作的设备包括：GPU、DC、PCIE、USB、SATA、GMAC、HDA 和 AC97。

以上两类地址(处理器的地址空间：即处理器的内存空间，处理器配置空间；桥片的地址空间：即桥片的配置访问空，I/O 空间，MEM 空间)采用统一编址的方式，都

位于同一地址空间内，且互不重合。

桥片内部设备的访问地址(PCI I/O 空间和 PCI MEM 空间)采用软件可配置设计，支持 PCI 体系架构的设备发现与管理。桥片内部的每个设备(设备块)都包含一个 PCI 配置头。软件通过访问配置头可以获得该设备的类型、支持的地址空间大小等信息,并通过配置该设备的 BAR 寄存器设置该设备的地址空间。该方式与 780E 一致。

5.6.2 地址空间划分

龙芯 3 号处理器+龙芯 7A 桥片地址空间划分示例：

表 5-4

	空间划分
7A PCI MEM 高地址空间	MEM_UP_LIMIT ~ 0xfc,ffff,ffff
内存高地址空间	0x8000,0000 ~ MEM_UP_LIMIT
7A PCI MEM 低地址空间	0x4000,0000 ~ 0x7fff,ffff
处理器配置空间	0x3000,0000 ~ 0x3fff,ffff
保留	0x2000,0000 ~ 0x2fff,ffff
处理器低速设备空间	0x1f00,0000 ~ 0x1fff,ffff
处理器 LPC MEM 空间	0x1c00,0000 ~ 0x1dff,ffff
7A PCI I/O 空间和配置空间	0x1800,0000 ~ 0x1bff,ffff
7A 固定地址设备空间	0x1000,0000 ~ 0x17ff,ffff
内存低地址空间	0x0000,0000 ~ 0x0fff,ffff

上表中，凡是 7A 桥片地址空间的地址都需要通过 L1 XBAR（处理器的一级交叉开关）进行地址映射。地址映射是指将桥片的 IO 空间、配置访问空间、部分 MEM 访问空间映射到 32 位地址段，原因是 PMON 的 C 代码只支持 32 位地址空间。这样实际上将原来需要使用 64 位地址才能访问的 HT IO 空间、HT 配置空间直接使用 32 位地址空间进行映射，使用映射之后的这些空间使用 32 位地址即可访问。

7A 桥片地址空间与 HT1 空间映射关系如下：

表 5-5

空间划分	32 位地址	X1 XBAR 映射的 64 位地址
7A PCI MEM 高地址空间	MEM_UP_LIMIT ~ 0xfc,ffff,ffff	0x90000e00,0000,0000+MEM_UP_LIMIT ~ 0x90000efc,ffff,ffff HT1 的 MEM 空间
7A PCI MEM 低地址空间	0x4000,0000 ~ 0x7fff,ffff	0x90000e00,4000,0000~ 0x90000e00,7fff,ffff

		HT1 的 MEM 空间
7A PCI I/O 空间	0x1800,0000 ~ 0x19ff,ffff	0x90000efd,fc00,0000 ~ 0x90000efd,fdff,ffff HT1 的 I/O 空间
7A PCI 配置空间	0x1a000000 ~ 0x1bff,ffff	0x90000efd,fe00,0000 ~ 0x90000efd,ffff,ffff HT1 的总线配置空间
7A 固定地址设备空间	0x1000,0000 ~ 0x17ff,ffff	0x90000e00,1000,0000 ~ 0x90000e00,17ff,ffff HT1 的 MEM 空间

7A 固定地址设备空间划分如下表，如: 中断控制器、HPET 控制器、confbus 配置寄存器和 MISC 低速设备块。

表 5-6 桥片固定地址设备地址空间

模块	地址空间	空间大小	访问类型
INT	0x1000,0000~0x1000,0fff	4K	BHW
HPET	0x1000,1000~0x1000,1fff	4K	BW
CONF reg	0x1001,0000~0x1001,ffff	64K	BHW
MISC 设备	0x1008,0000~0x100f,ffff	512K	BW

MISC 低速设备包括：UART、I2C、PWM、ACPI、RTC 和 GPIO。

MISC 低速设备块内部多个设备由地址位的 bit[18:16]进行区分,不同的设备只支持特定类型的访问。设备路由及支持的访问类型见下表。

表 5-7 MISC 低速设备地址路由及访问类型

Bit[18:16]	0	1	2	5	6
设备	UART	I2C	PWM	ACPI/RTC	GPIO
访问类型	B	B	W	W	B

因此 MISC 各低速设备在 MISC 基地址的基础上的偏移地址为：

#define UART_BASE_ADDR_OFFSET 0x00000

#define I2C_BASE_ADDR_OFFSET 0x10000

#define PWM_BASE_ADDR_OFFSET 0x20000
#define ACPI_BASE_ADDR_OFFSET 0x50000
#define GPIO_BASE_ADDR_OFFSET 0x60000

5.6.3 PCI 设备访问地址

桥片内部具有 DMA 功能的设备和一些其他设备包含了一个标准的 PCI 配置头。包含 PCI 配置头的设备包括：GPU、DC、PCIE、USB、SATA、GMAC、HDA/AC97、LPC 和 SPI。各设备对应的总线号，设备号，功能号见龙芯 7A 用户手册 3.2 节。

处理器可以通过 2 个地址空间来访问桥片的配置空间。 一个是 HT 定义的标准配置访问空间（0xFD,FE00,0000~0xFD,FFFF,FFFF），另一个是 HT 的保留空间（0xFE,0000,0000~0xFE,1FFF,FFFF）。推荐使用 HT 标准配置访问空间来进行 PCI 配置头访问。

通过 HT 标准配置访问空间访问的每个桥片设备的配置空间大小为 256 字节；通过保留地址空间访问的每个桥片设备的配置空间大小为 4K 字节。其中，地址的[39:24]决定配置头类型(0xFDFE 是 Type0,0xFDFF 是 Type1)；[23:16]表示总线号(Bus Number)；[15:11]表示设备号(Device Number)；[10:8]表示功能号(Function Number)；[7:0]表示偏移(offset)。

对桥片设备的访问主要通过 PCI MEM 空间来完成。软件可以在该地址段内任意分配桥片各个设备的访问地址。桥片的内部 PCI 设备包括：GPU/DC、GMEM、PCIE、USB、SATA、GMAC、HDA/AC97、LPC、SPI。所有这些设备可以通过 lspci 看到。这些设备的访问地址可以由软件动态分配。

一种分配方式如下：通过扫描 PCI 总线，读取各个设备(PCI 方式方位)的配置空间，获取各个设备使用的 MEM 空间和 I/O 空间大小，系统软件从 0x40000,0000~0x7fff,ffff 这个地址内分配合适大小的 MEM 空间，从 0x1800,0000~0x19ff,ffff 这个地址内分配合适大小的 I/O 空间(PCIE 设备)。

PCI 设备地址空间分配示例：

表 5-8

模块	地址空间	空间大小
GPU	0x5ff4,0000~0x5ff7,ffff	
DC	0x5ff8,0000~0x5ff8,ffff	64K

Graphic Memory (共享显存)	0x4000,0000~0x4fff,ffff	256M
PCIE I/O	0x1800,0000~0x19ff,ffff	32M
PCIE MEM	0x6000,0000~0x7fff,ffff	512M
LPC MEM	0x5c00,0000~0x5dff,ffff	32M
SPI MEM	0x5e00,0000~0x5eff,ffff	16M
USB-0/1-OHCI/EHCI	0x5fd0,0000~0x5fd1,ffff	32K*4
SATA0/1/2	0x5fe0,0000~0x5fe0,5fff	8K*3
GMAC 0/1	0x5fe1,0000~0x5fe1,ffff	32K*2
HDA	0x5fe2,0000~0x5fe2,ffff	64K
AC97	0x5fe3,0000~0x5fe3,ffff	64K
LPC REG	0x5ff1,0000~0x5ff1,0fff	4K
SPI REG	0x5ff1,1000~0x5ff1,1fff	4K
LPC I/O	0x5ff2,0000~0x5ff2,ffff	64K
LPC TPM	0x5ff3,0000~0x5ff3,ffff	64K

上述地址中，除了 PCIE MEM 和 Graphic Memory 外,其它设备的地址空间大小是固定不变的,软件可以改变地址空间的起始地址。PCIE MEM 地址空间的大小需要根据所接设备来决定。

当使用桥片内部集成的显存时,Graphic Memory 的大小根据所接显存的容量来决定(显存的容量最大为 256MB)。BIOS 需要通过访问桥片通用配置寄存器来修改 GPU 配置头的 BAR 寄存器 2/3 的 MASK 值来配置 Graphic Memory 的大小,然后软件再通过 PCI 扫描的方式来获得显存的大小。

5.7 GPIO

7A 的 GPIO 放在 MISC 设备块空间，仅支持 byte 访问，不支持其它类型的访问。同时，7A 新增了一种按字节访问单个 GPIO 的功能，推荐使用。

注意：桥片共 57 个 GPIO 引脚，除了 GPIO00 为专用 GPIO 引脚，其余 56 个与其它功能复用，复用关系查看 longson7A 用户手册附录 1：芯片引脚复用表。

配置举例：

要求：使用 GPIO1 功能，且输出为高电平。

管脚复用：VSB_GATEn（默认使用功能）与 GPIO1 复用。

首先使能 GPIO1 功能：VSB_GATEn 引脚和 GPIO01 的复用关系由电源管理模块

内部的寄存器(PMCON_RESUME)配置。ACPI 的 PMCON_SOC：SOC General PM Configuration Register 中偏移地址 0x04 寄存器的 bit13 配置该复用功能；将该寄存器的 bit13 配置为 0。（0：关闭 VSB_GATEn 功能，1：使能 VSB_GATEn 功能）

配置 GPIO1 为输出方向，输出为 1。使用推荐的按字节访问 GPIO 模块方法：配置 GPIO 输出使能寄存器 GPIO_OEN（0x801）为 1，配置 GPIO 输出值 GPIO_O（0x901)为 1。

5.8 龙芯 VBIOS

5.8.1 背景

为了解决长期困扰龙芯桥片的显示驱动碎片化的问题，在发布 7A 桥片软硬件规范的同时，增加了龙芯 VBIOS。

龙芯 VBIOS 的作用与 x86 PC 的 vbios 作用并不完全相同，并不具备初始化 UEFI 显示的功能，主要作用是向内核传递显示相关信息，用于内核显示驱动初始化。Vbios 可以由龙芯提供的 vbios 生成工具生成。

5.8.2 VBIOS 详细介绍

5.8.2.1 Vbios 生成工具需要配置的四个结构体

(1) **Vbios**： vbios 是 vbios 的基本信息，包括 vbios 遵循的版本号，厂商信息（最多 19 字节），crtc 数量、connector 数量、phy 数量。

(2) **Crtc（显示器控制器）**： 7A 中显示控制器最多有两个，分别对应硬件的 DVO0 和 DVO1，其中 crtc_id 的 0 和 1 分别对应 DVO0 和 DVO1，如果板卡中只引出一路 DVO 则只配置一个显示控制器结构体。其中 crtc_max_weight 和 crtc_max_height 是 crtc 默认支持的最大分辨率，默认均设置为 2048，该值取 phy 和 crtc 中较小的一个。

(3) **Connector**：connector 是获取显示器 EDID 的抽象层，一般是对应的获取显示器 EDID 的 i2c，每一个 connector 对应一个 crtc,其中的 crtc_id 表明该结构体对应的 dvo（0 对应 dvo0，1 对应 dvo1），i2c_id 对应的是内核中看到的 i2c 号，如果使用我们的参考设计，DVO0 GPIO I2C 对应 I2C 号为 6，DVO1 GPIO I2C 对应 I2C 号为 7。edid_method 对应的是 edid 的获取方式，默认通过 GPIO I2C 获取 EDID；另外还有 3

个选项是：

- 不具有 EDID 的获取能力；

设置为该选项，显示接口将认为一直连接有显示器，但分辨率最大只能1024x768。

- EDID 内置于 vbios 中（尚未实现）；
- 通过 phy 内置的 I2C 获取 EDID ；

如使用这一选项，phy 必须传递一个非透明型号。

(4) **phy**：因为龙芯 7A 的显示输出接口是 DVO，需要配合 PHY 芯片才能转换为 VGA、HDMI、DVI 等常见显示接口，phy 结构体记录的就是 phy 芯片的相关信息，每个显示控制器对应一个 PHY，其中最核心的是 phy 的型号,根据 phy 的特性，有三种常见状态：

a） 全透明 phy，这种 phy 不需要任何配置，唯一需要注意的是支持的最大分辨率是否比 CRTC 的最大支持分辨率小，

b） 只需要设置一次的 phy，这种 phy 只需要在 uefi 中设置配置一次（如不配置，UEFI 阶段屏幕无显示），在内核中可以认为是全透明 phy。

c） 每次切换分辨率都需要设置的 phy，这种 phy 需要传入一个特定型号，为了方便统一管理，需要板卡厂商，联系龙芯一起适配。自己不能私自修改 vbios 的 phy 型号，否则与龙芯的内核不兼容。

5.8.2.2 优先级

Vbios 有三种方式：1.固件（UEFI、PMON）传参。2.直接从 spi flash 中读取。3.使用内核内置的 default vbios。三者的优先级是依次递减，固件传参的 vbios 优先级最高。

举例说明:

以固件 UEFI 为例说明 vbios 的传递方法（非 SPI 读）。

1. 利用脚本将 Vbios 二进制转换为 C 数组:

./bin2c vbios.bin vbios.h

2. 为 vbios 数组命名（生成文件之后名称为空）：

打开 vbios.h，添加数组名称 Vbios[]

3. 将 vbios.h 添加到 uefi 目录下:

LoongsonPlatformPkg/Ls3a30007aPkg/Ls7a/Drivers/BootparamsDxe/

4. 修改目录下文件 BootparamsDxe.c，增加头文件:

#include “vbios.h”

5. 将 vbios 数组拷贝到地址 0x980000000F0000a0，然后将地址赋给vga_rom：

gBS->CopyMem((VOID *)DestVbios, (VOID *)Vbios, sizeof(Vbios));

smbios->vga_bios = (UINT64)DestVbios;

5.8.3 VBIOS Creator 工具使用说明

5.8.3.1 进入软件主界面

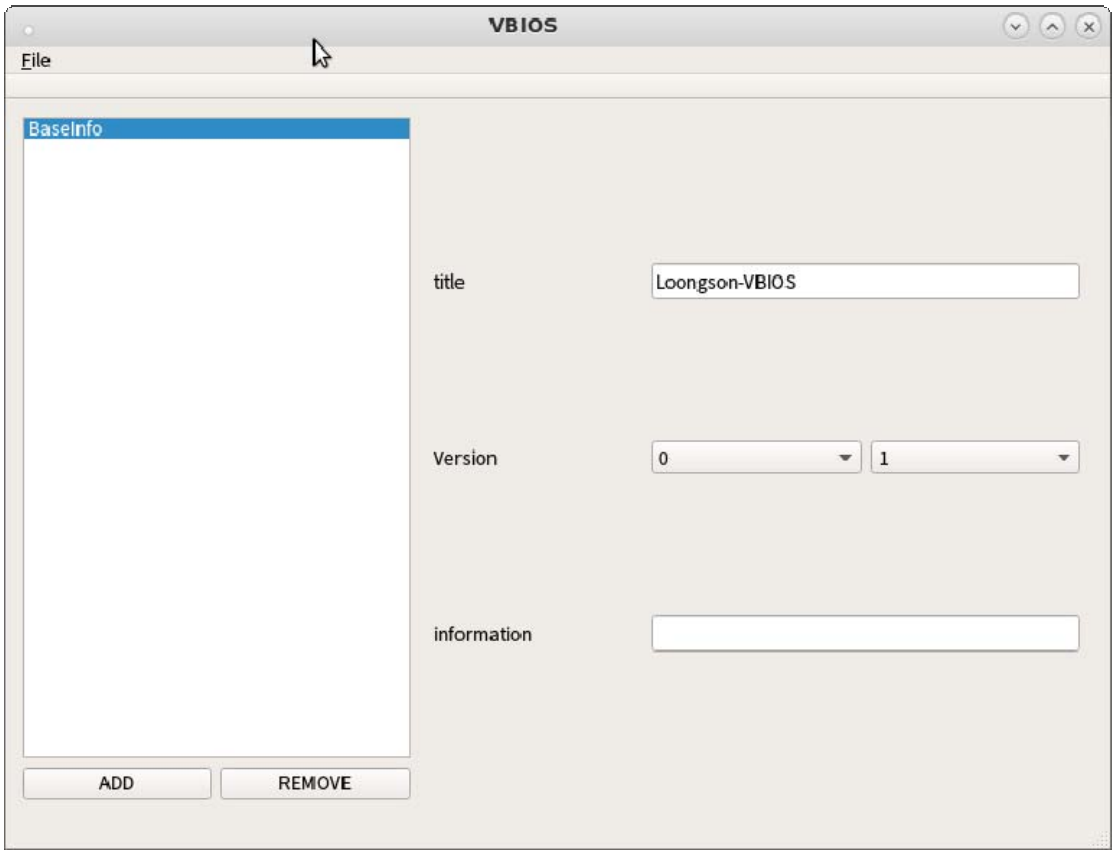


图 5-1 VBIOS Creator 主界面示例图

- “title”：设置基础信息，title 默认不可修改；
- “Version”：版本可以根据根据下拉菜单进行修改；
- “information”：可以任意填写不超过 19 个字符。

5.8.3.2 添加 phy

单击 ADD 按钮，选择 PHY 可以添加一个 phy，如下图：

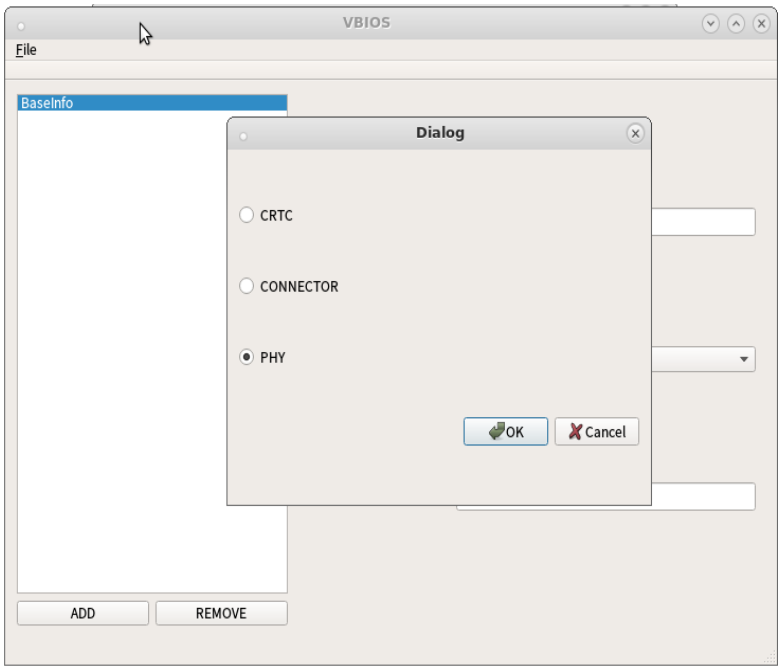


图 5-2 添加 PHY 示例图

左侧列表会增加 phy

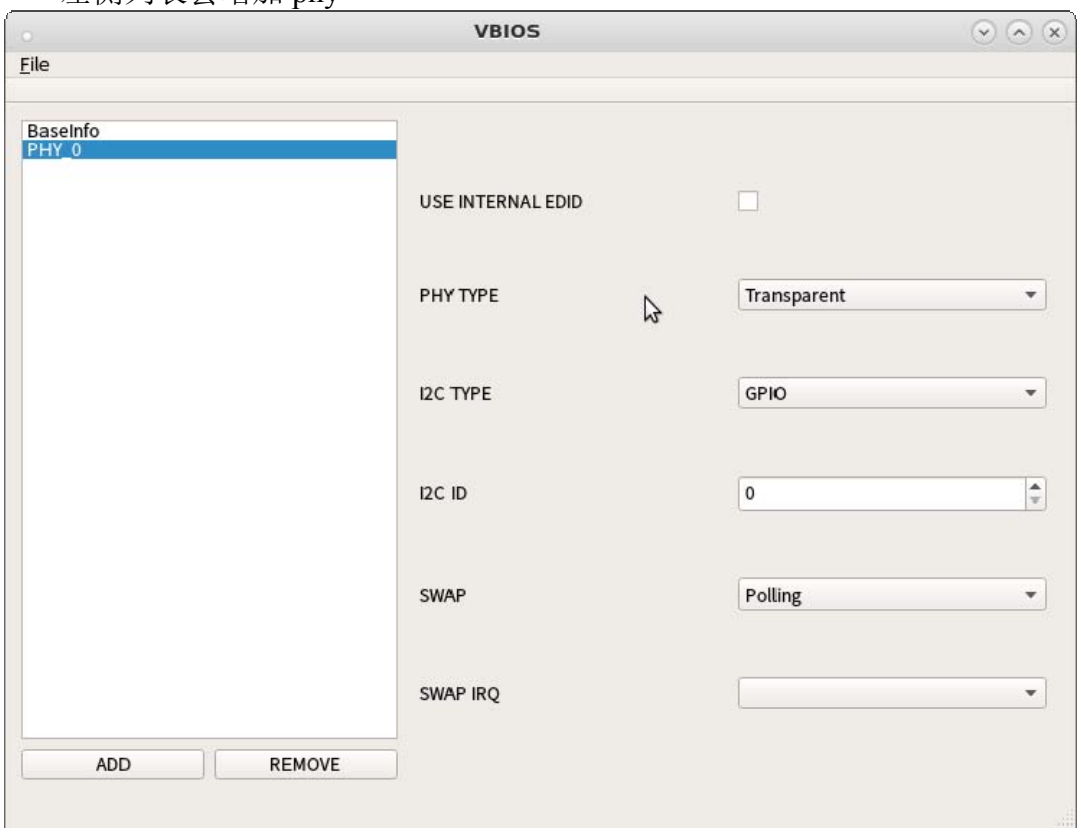


图 5-3 编辑 PHY 配置示例图

根据需求修改右侧编辑栏的对应值，即可完成一个 phy 的添加。

1 “PHY TYPE” ： DVO 输出 PHY 型号；

2 “I2C TYPE”：PHY 芯片与 I2C 控制器连接方式

a. “NULL”（无 I2C）、

b. “GPIO”（使用 7A 桥片为 DC 预留的 2 路 GPIO 模拟 I2C）、

c. “CPU”（其它 I2C，如 SOC 中的 I2C 控制器）、

d. “PHY”（保留位），7A 桥片默认选择“GPIO”；

3 “I2C ID”：I2C 的序号，“phy0”在 3A7A 开发板中因”Transparent“是全透明的 phy，所以此项可不选择；

4 “SWAP”：显示器热拔插检测方式，3A7A 开发板只支持”polling“（轮询）

5 “SWAP IRQ”：3A7A 开发板暂时不支持

在 3A7A 开发板中，有两路 PHY，需要再添加一个 phy:

a.“PHY TYPE” ： DVO 输出 PHY 型号；

b.“I2C TYPE”：PHY 芯片与 I2C 控制器连接方式，有：“NULL”（无 I2C）、“GPIO”（用 7A 桥片为 DC 预留的 2 路 GPIO 模拟 I2C）、“CPU”（其它 I2C，如 SOC 中的 I2C 控制器）、“PHY”（保留位），7A 桥片默认选择“GPIO”；

c.“I2C ID”：I2C 的序号，“phy0”在 3A7A 开发板中因“Transparent”是全透明的 phy，所以此项可不选择；

d.“SWAP”：显示器热拔插检测方式，3A7A 开发板只支持“polling”（轮询）

e.“SWAP IRQ”：3A7A 开发板暂时不支持

5.8.3.3 添加 connector

选择 CONNECTOR 可以添加一个 connector；如下图

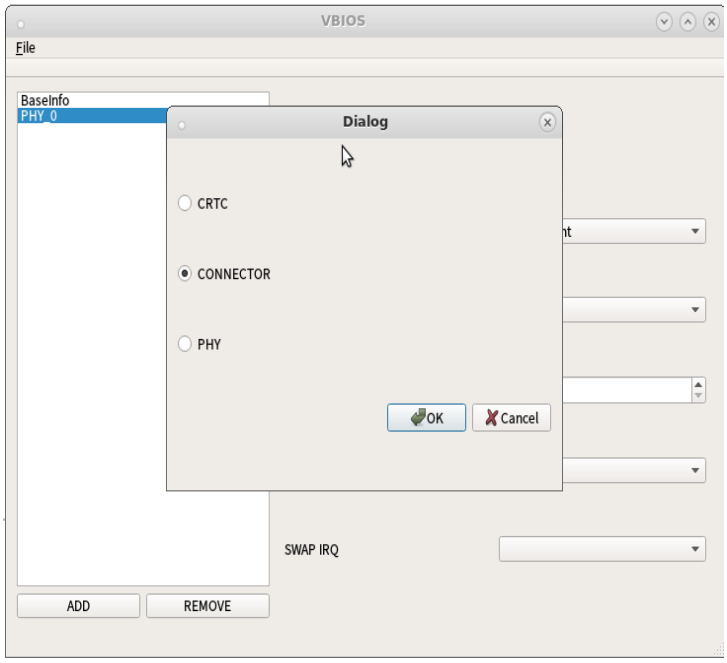


图　5-4　　添加 Connector 示例图

左侧列表会增加一个 connector

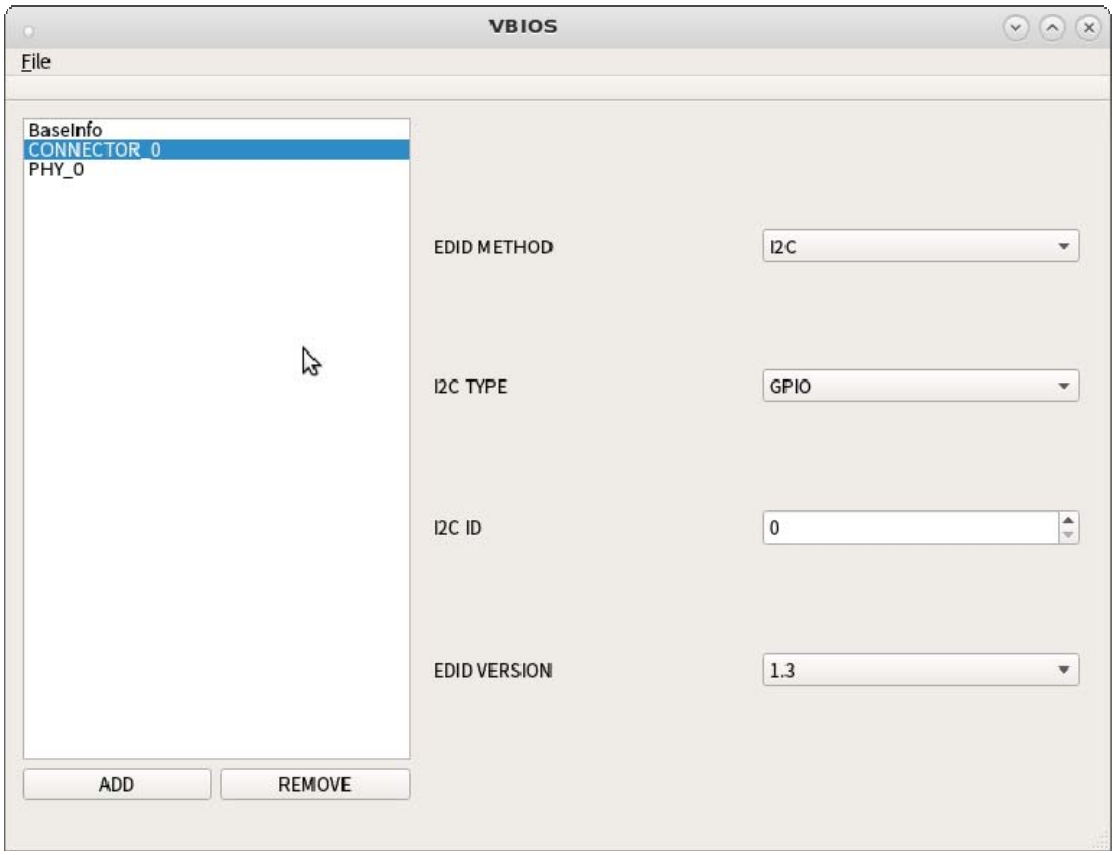


图　5-5　　编辑 Connector 配置示例图

1. “EDID METHOD”：　EDID 获取方式

a) “NULL”： 硬件上无法获取 EDID

b) “I2C”：使用“connector”中所配置的 I2C，3A7A 开发板默认选择

c) “VBIOS”：保留项

d) “PHY”：保留项

2. “I2C TYPE”：I2C 的连接方式

a) “NULL”（无 I2C）、

b) “GPIO”（使用 7A 桥片为 DC 预留的 2 路 GPIO 模拟 I2C）、3A7A 开发板默认选择

c) “CPU”（其它 I2C，如 SOC 中的 I2C 控制器）、

d) “PHY”（保留位），7A 桥片默认选择“GPIO”；

3. “I2C ID”：I2C 的序号，3A7A 开发板及配套内核中，“connetor 0”（与 DVO 0 连接的 I2C）默认使用“6”号 I2C ；

4. “EDID VERSION”：此项仅在“EDID METHOD”中选择“VBIOS”才有效；

3A7A 开发板配置后的 connetor 如下图所示：

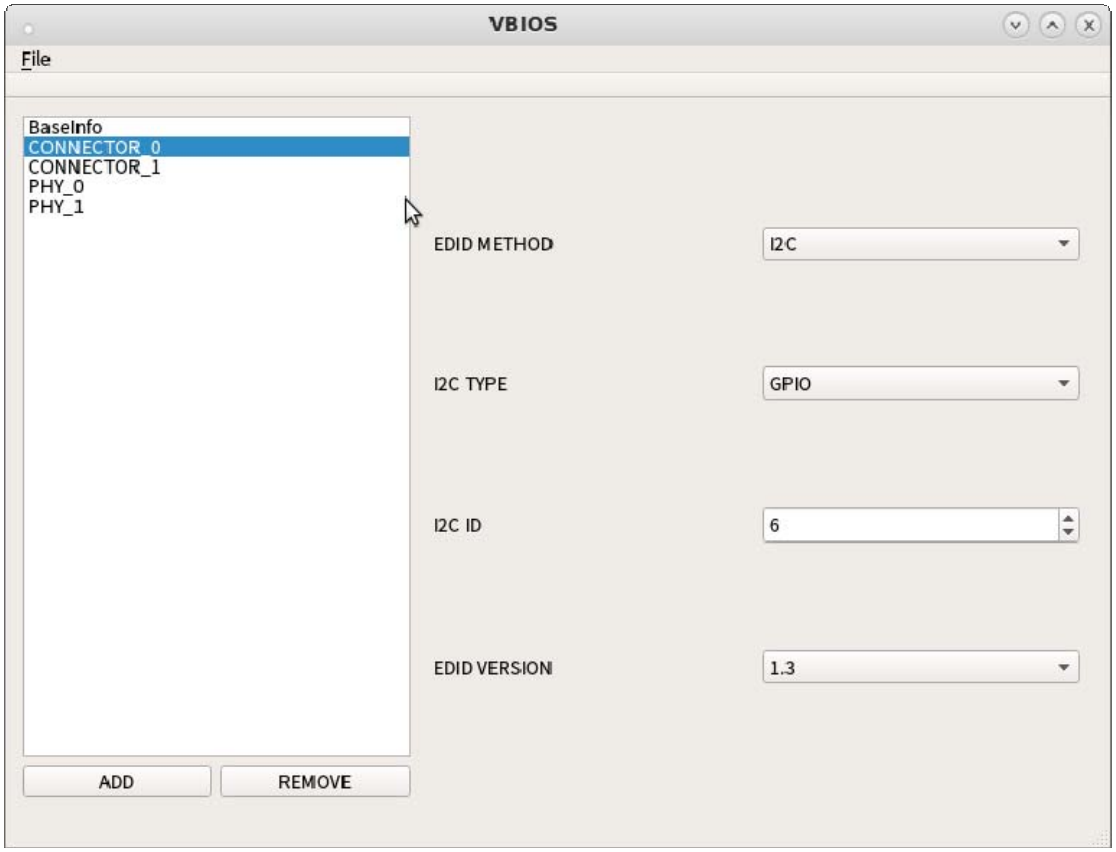


图 5-6 编辑 Connector 0 配置结果示例图

因 3A7A 开发板中连接 2 路 DVO，因还需要配置”connetor 1”:

1. “EDID METHOD”： EDID 获取方式
   a. “NULL”： 硬件上无法获取 EDID
   b. “I2C”：使用“connector”中所配置的 I2C，3A7A 开发板默认选择
   c. “VBIOS”：保留项
   d. “PHY”：保留项
2. “I2C TYPE”：I2C 的连接方式
   a. “NULL”（无 I2C）、
   b. “GPIO”（使用 7A 桥片为 DC 预留的 2 路 GPIO 模拟 I2C）、3A7A 开发板默认选择
   c. “CPU”（其它 I2C，如 SOC 中的 I2C 控制器）
   d. “PHY”（保留位），7A 桥片默认选择“GPIO”；
3. “I2C ID”：I2C 的序号，3A7A 开发板及配套内核中，“connetor 0”（与 DVO 0 连接的 I2C）默认使用“6”号 I2C， ；
4. “EDID VERSION”：此项仅在“EDID METHOD”中选择“VBIOS”才有效；

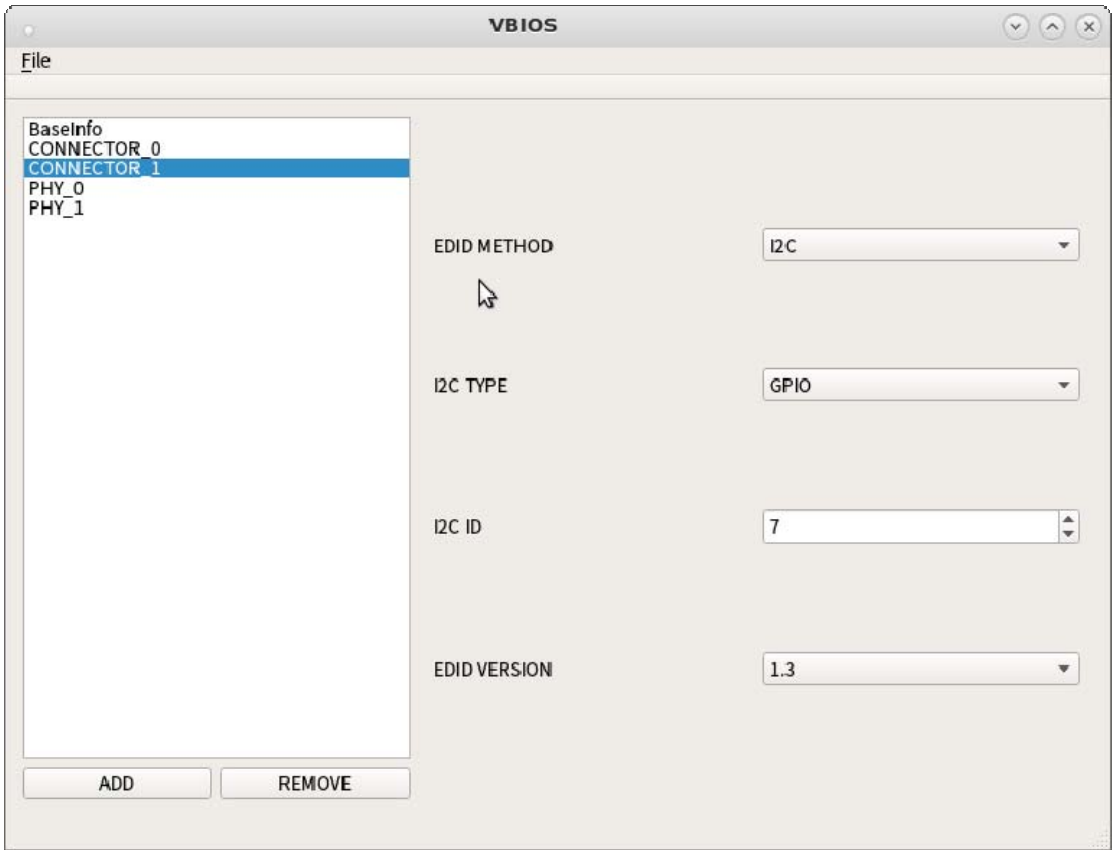


图 5-7 编辑 Connector 0 配置结果示例图

5.8.3.4 添加 crtc

点击 ADD 按钮，在弹出的对话框选择 CRTC 来添加一个 crtc，如下图：

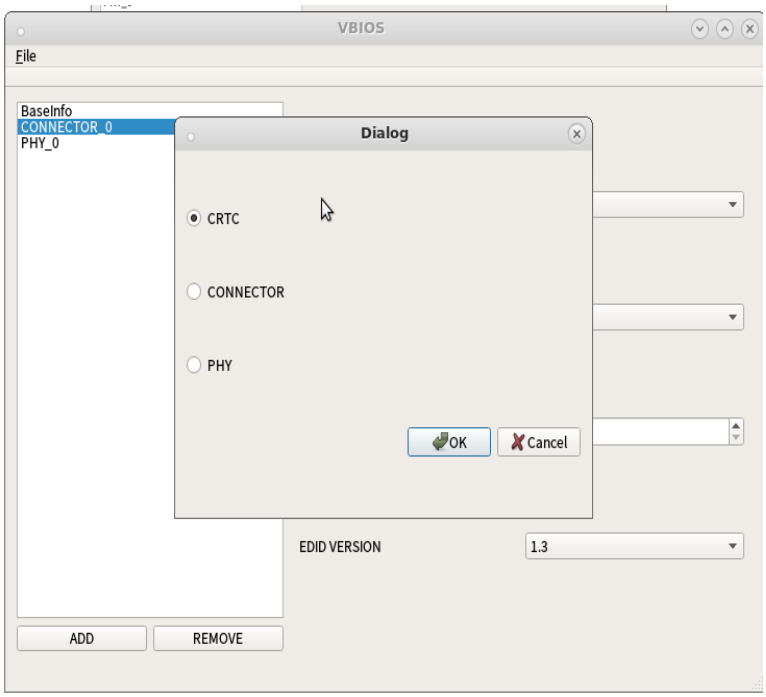


图 5-8 添加 crtc 示例图

左侧列表会出现一个 crtc

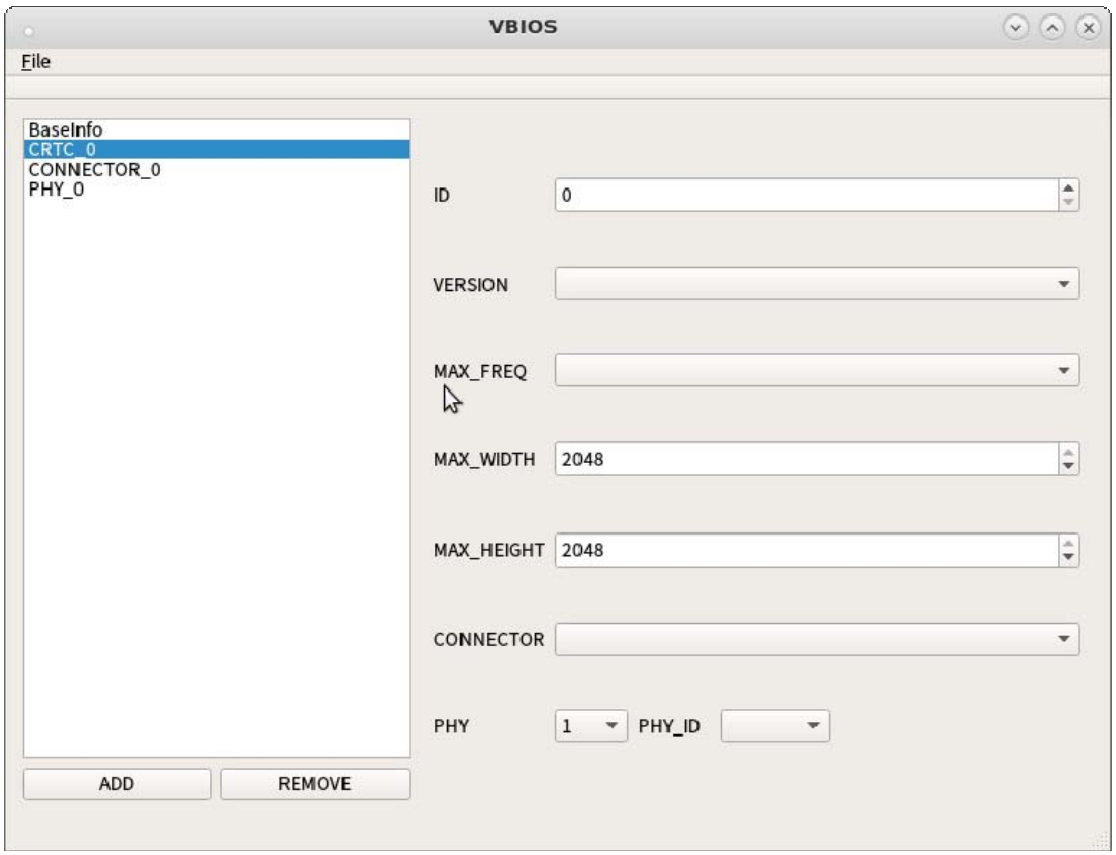


图 5-9 crtc 配置项示例图

1. “ID”：与“connetor”一一对应序号，如 connetor 0，则与 ctrc 0 对应
2. “VERSION”：7A 芯片中 DC 控制器型号，7A1000 中默认为“0”，暂不可选；

3. “MAX FREQ”：7A 芯片中 DC 控制器支持的最大时钟频率，保留
4. “MAX WIDTH”：7A 芯片中 DC 控制器支持的最大分辨率宽度值，默认最大为 2048
5. “MAX HEIGHT”：7A 芯片中 DC 控制器支持的最大分辨率高度值，默认最大为 2048
6. “CONNETOR”： 与“DVO”对应的编号，如 DVO 0, 则选择“0”；DVO 1,则选择“1”；
7. “PHY”： 与单个 DVO 所连接的 phy 的数量，3A7A 开发板仅连接 1 个 phy，默认选择 1；
8. “ID”：本 CRTC 所连接的 PHY 序号，3A7A 开发板的 crtc0 与 phy0 连接，故 crtc0 中此项选择“0”， crtc1 中此项选择“1”。

3A7A 开发板 crtc0 配置如下图所示：

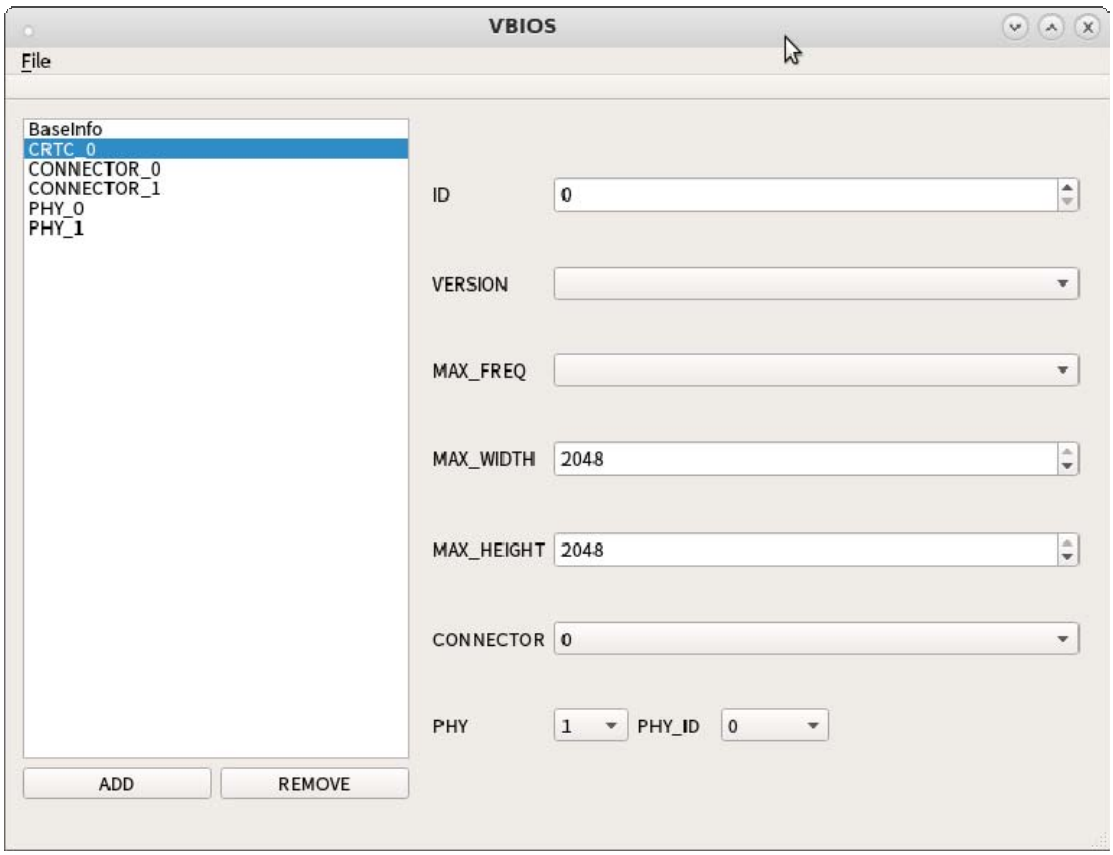


图 5-10 crtc_0 配置示例图

因 3A7A 开发板连接有 2 路显示，故还需要再配置一个 crtc：如下图所示

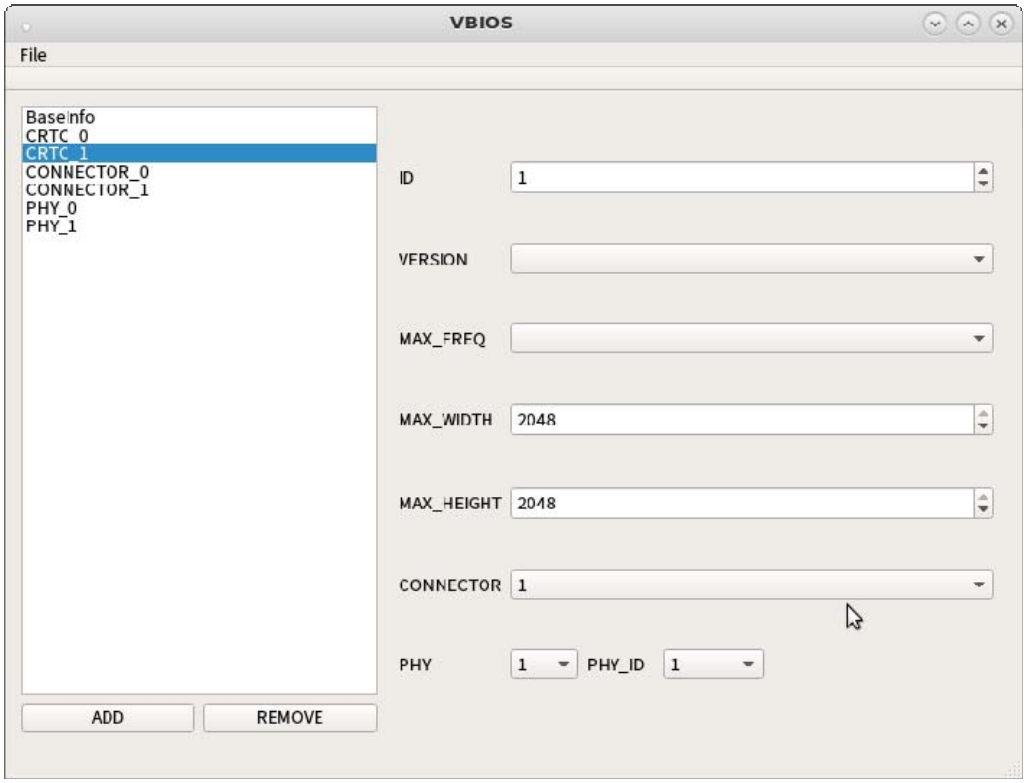


图 5-11 crtc_1 配置示例图

5.8.3.5 保存文件

单击 File 菜单，选择 save；或者使用快捷键 Ctrl+S。

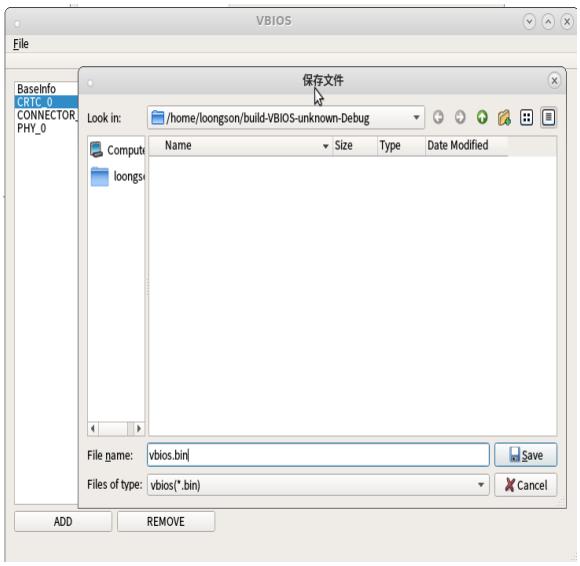


图 5-12 vbios 保存界面示例图

六、固件使用及注意事项

6.1 使用方法及操作说明

设置成自启动（PMON 自动加载内核，启动文件系统）：需要设置环境变量 $al、$append

以下结合实际，给出开发中 PMON 下常用的命令介绍

1、load --- 加载文件

load [-options] pathname

-o offset 将文件加载到内存 load_address + offset 处

-f addr 将文件加载到 flash 的地址 addr 处

pathname 要加载的文件路径或 URL 其

他选项 h load 查看

例子：

1）加载位于硬盘第一个分区的(PMON 只认第一个分区)内核:

load /dev/fs/ext2@wd0/boot/vmlinux ＃第一个分区是 ext2 文件系统 load

/dev/fs/ext2@wd0b/boot/vmlinux ＃加载第二个分区上的内核文件，wd0c,wd0d,为第三，四个分区。

load /dev/fs/ext2@usb0/path/to/kernel ＃从 U 盘或者移动硬盘加载 kernel，注意：如果根文件系统位于 U 盘或者移动硬盘的话，传给内核的参数应为：root=8:1 rootdelay=5。

load tftp://192.168.10.100/vmlinux ＃从远程 tftp 服务器上加载内核，先要配置好网络。

load /dev/fs/iso9660@cd0/boot/vmlinux ＃从光盘加载内核

load /dev/ram@address ＃从内存加载内核

load /dev/ram@address,size load

/dev/ram/logger

2）PMON 更新命令

load -r -f 0xbfc00000 tftp://SERVER_IP/gzrom.bin

load -r -f 0xbfc00000 /dev/fs/ext2@wd0/gzrom.bin

PS: 以上 wd0, cd0, ram 为 PMON 对设备的命名，可以用 devls 命令查看他发现

并支持的所有设备。

2、g ------ 执行程序

g [-st] [-b addr] [-e addr] [-- args...]

当 load 完一个文件后, 通过 g 命令告诉 PMON 开始执行刚刚载入的文件。

例子：

g # 从epc 寄存器指定的地址处开始执行

g -e addr # 从内存地址 addr 处开始执行

g -e addr -b add2 # 从内存地址 addr 处开始执行, 在addr2 处设置一个临时断点，在下一次执行挂起时删除该断点

g console=tty root=/dev/hda1 # 从epc 寄存器指定的地址处开始执行,并将参数 console=tty root=/dev/hda1 传递给程序(内核)

3、devls ------- 显示设备

devls [-a]

例子：

devls -a # 显示所有设备

4、ifaddr ------ 配置网卡

ifaddr ifname ipaddr [:ifparameters]

其中，第二个字段可以为 ipaddr:netmask:broadcast:gateway

例子：

ifaddr rtl0 192.168.10.99 # 设置网卡 rtl0 的 IP 地址

ifaddr rtl0 192.168.10.99:255.255.255.0::192.168.10.2 # 设置网卡 IP 地址和网关

5、 ping

ping 192.168.10.188 # 测试网络是否配置好

6、输出到串口／显示器（永久起效）

set novga 1/0

7 、输出到串口／显示器（临时起效）

setvga 0/1

内核作为PMON 的一个进程，由加载程序创建进程上下文，然后执行任务切换，一去不复返。

6.2 固件和内核的接口规范

本节主要描述与内核接口的两个头文件 smbios.h 及bootparam.h 及相关的说明。

6.2.1 固件与内核接口的约定

1、固件与内核接口的数据结构的头文件参见附录，结构的具体说明参见龙芯官网下载中心《龙芯 CPU 开发系统固件与内核接口详细规范 V2.0》。

头文件的命名约定为 bootparam.h，相应实现函数的 c 文件命名约定为 bootparam.c，以上两个文件放在 pmon/common 目录。

要求在固件中初始化好所有的结构体或服务，最终封装到 boot_params 的结构体中，将该结构体指针 bp 赋值给 a2 寄存器。在 main.c 中调用 boot_params 的指针，并将指针赋值给 a2 寄存器。具体实现是在 pmon/common/main.c 中的 initstack 函数中调用 md_setargs 实现，在 machdep.c 中该函数定义如下：

```
md_setargs(struct trapframe *tf, register_t a1, register_t a2,
                                register_t a3, register_t a4)
{
                if (tf == NULL)
                        tf = cpuinfotab[whatcpu];
                tf->a0 = a1;
                tf->a1 = a2;
                tf->a2 = a3;
                tf->a3 = a4;
}
```

2、SMBIOS

SMBIOS 要求实现目前建议必须实现的 SMBIOS 类别如下: 兼容 SMBIOS 2.3 规范版本， 必须实现包含以下 8 个数据表结构：

1） BIOS 信息(Type 0)

2） 系统信息 (Type 1)

3） 产品信息 (Type 2)

4） 处理器信息 (Type 4)

5） 物理存储阵列 (Type 16)

6） 存储设备(Type 17)

7） 温度传感器 (Tye 28)

8） 表格结束指示 (Type 127)

具体可参见《龙芯 CPU 开发系统固件与内核接口详细规范 V2.0》。

SMBIOS 要求放在 pmon/common 目录，SMBIOS 的结构入口地址定义为：

#define SMBIOS_PHYSICAL_ADDRESS 0x8fffe000

具体实现是在 pmon/common/smbios/smbios.h 中定义。

6.2.2 固件运行时服务的约定

如下函数约定在 PMON 中实现1、

init_reset_system()

概述：init_reset_system 用于复位整个平台，包括处理器、设备、以及重启系统。

函数原形 void init_reset_system(struct efi_reset_system_t *reset)

参数：reset

```
struct efi_reset_system_t {
u64 ResetCold;
u64 ResetWarm;
u64 ResetType;
u64 Shutdown;
u64 DoSuspend; /* NULL if not support */
};
```

表 6.1 efi_reset_system_t 结构含义说明

名称	描述
ResetCold	冷启动， 将全系统电路恢复到初始状态
ResetWarm	热启动，CPU 设置为初始状态，其他正常
ResetType	为后续预留
Shutdown	使系统进入类似 ACPI G2/S5(OS 不会保存和回复系统上下文，仅有很少设备如键盘，鼠标供电)或 G3(关机)状态
DoSuspend	为后续预留

返回的状态码: 该函数没有返回码。

目前在PMON 里必须实现 Shutdown 和ResetWarm 两个函数，对应的实现函数名称统一为poweroff_kernel 和reboot_kernel。

6.2.3 内核接口规范更改

6.2.3.1 地址分配的变动

随着龙芯平台的发展，之前的内核地址规范已经不能正常满足现在的需求。为能更好的适应未来的发展，针对《龙芯 CPU 开发系统固件与内核接口详细规范 V2.0》如下地址更改。

地址空间设计的规则约定如下：

a) 0x0000_0000～0x0FFF_FFFF 的低 256MB 空间为内存空间。其中 0x0F00_0000～0x0FFF_FFFF 为固件保留的 16M 地址空间，0x000 0000～0x001F FFFF 为兼容老版本固件保留的 2M 地址空间，操作系统内核不得使用；

b) 0x1000_0000～0x1FFF_FFFF 为 PCI 等 IO 设备空间及部分芯片配置寄存器空间；

c) 0x3000_0000～0x3FFF_FFFF 为窗口配置寄存器的空间范围；

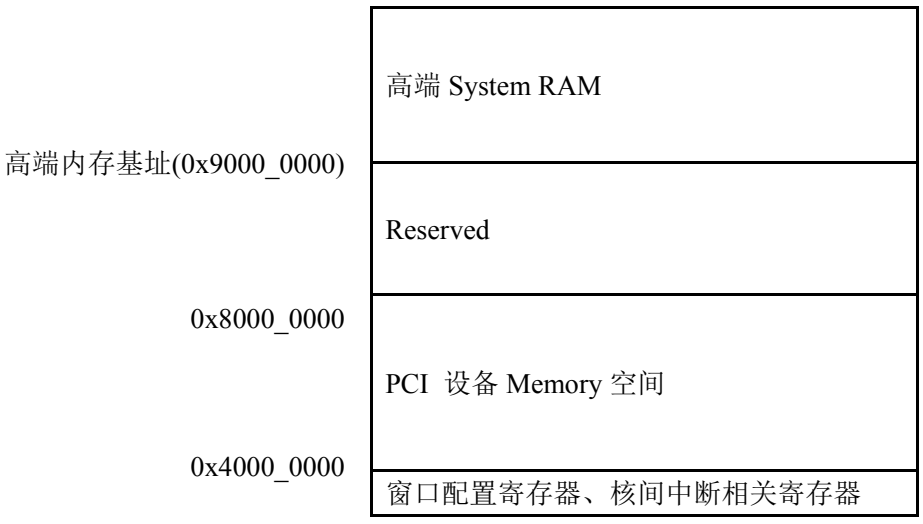
d) 0x4000_0000～0x7FFF_FFFF 为 PCI 设备 memory 空间范围；

e) 0x2000_0000～0x2FFF_FFFF 和 0x8000_0000～高端内存基址-1 为保留空洞；

f) 高端内存基址缺省值为 0x9000_0000，各系统如有特殊需要可从 emap 结构传递修改的值；

g) 内存地址信息必须由固件赋值后传参至内核，内核不可擅自修改。内存地址信息传参结构体描述详见附录 A。

地址空间划分如图 1 所示：



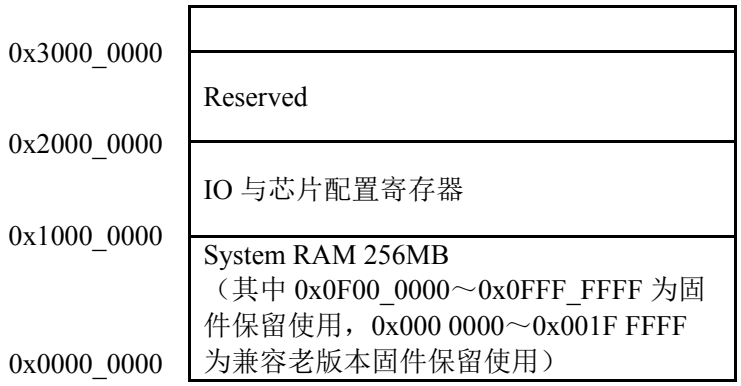


图 6.1 地址空间分布图

下面举例来说明高端内存的使用情况，

a) 如果内存大小为 1GB，则内存在系统中的地址空间表 6-1：

表 6-1 1GB 内存地址分布表

	起始地址	结束地址	物理内存
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	0 – 256MB
地址 1	0x0000_0000_9000_0000	0x0000_0000_BFFF_FFFF	256MB – 1GB

b) 如果内存大小为 2GB，则内存在系统中的地址空间如表 6-2：

表 6-2 2GB 内存地址分布表

	起始地址	结束地址	物理内存
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	0 – 256MB
地址 1	0x0000_0000_9000_0000	0x0000_0000_FFFF_FFFF	256MB – 2GB

c) 如果内存 4GB 以上，则内存在系统中的地址空间如表 6-3：

表 6-3 4GB 内存地址分布表

	起始地址	结束地址	物理内存
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	0 – 256MB
地址 1	0x0000_0000_9000_0000	0x0000_0001_7FFF_FFFF	256MB – 4GB

6.2.3.2 增加 3A/3B+7A 相关描述

- 3A/B+7A 中断系统描述

7A 桥片有 1 组中断控制寄存器，控制 64 个中断源。 3A/B+7A 板上，桥片上的中断则是通过 HT1 控制器进入到 Int Controller 的 HT1-2 和 HT1-3 脚，所有经过 7A 中断控制器的外设中断都是路由到 0 号核（即 CPU 0）的 IP3 ，而 7A 桥片上外接的 PCIE 设备默认采用了 MSI 中断，可以绕过 7A 的中断控制器直接发送到 HT1 的中断向量

HT1-0 和 HT1-1 上,并最终通过轮转机制发送到 cpu0~cpu3 上。当 7A 桥片的中断触发时，3A/B Core0 的 Cause 寄存器 IP3 位就会被置 1，这时查询 HT 的中断状态寄存器就可以判断出中断来源，如图 6-2。

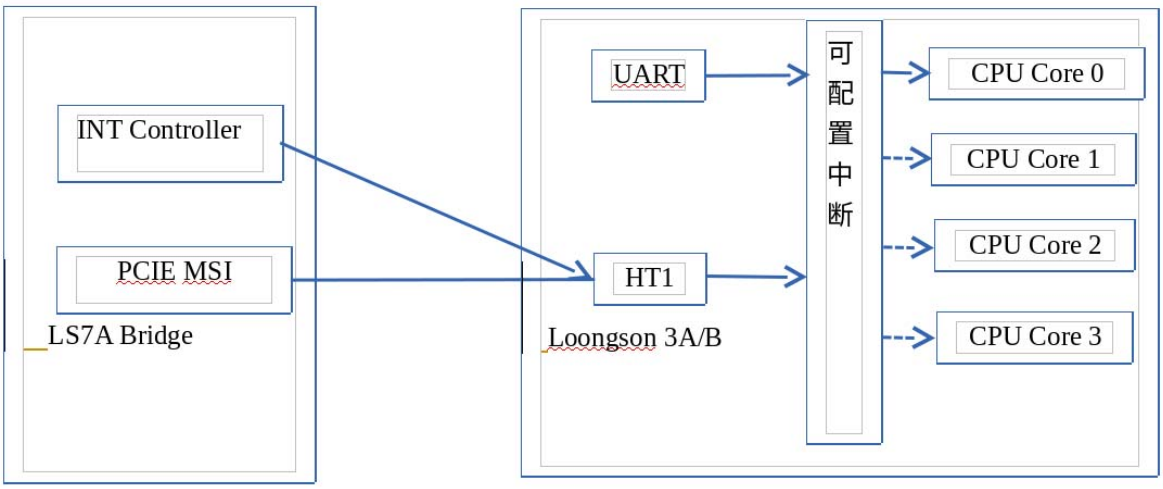


图 6-2 3A/3B+7A 中断路由示意图

- 3A/B+7A 的中断号分配约定

0-55 号保留给桥片上的 PCIE 设备发送 MSI 中断。56-63 号中断号保留给 cpu 内部中断源，如表所示。

表 6-4 56-63 号中断说明

中断号	中断源	说明
56		
57		
58	uart	cpu 中的 uart 中断
59	timer	timer 中断
60		
61		
62		
63		

64-128 号： 分配给 7A 桥片。

- 3A/B+7A 的中断路由约定

IP0~IP7 用于标识发生了哪些中断。IP0 和 IP1 是软件中断位，不提供对外的中断引脚。IP2~IP5 随着硬件输入引脚上的信号而变化，标识哪些设备发生了中断。IP6 用

于处理器核间中断，负责多核处理器的通信。IP7 一方面用于 MIPS 内部的定时器中断，另一方面用于性能计数器中断。具体描述如下：

IP0：软件中断（Linux 内核暂未使用）

IP1：软件中断（Linux 内核暂未使用）

IP2：CPU 内部 LPC 总线和 UART 设备

IP3：南桥设备中断

IP4：传统 Bonito 南桥中断

IP5：保留（以后扩展）

IP6：多核处理器核间中断

IP7：定时器和性能计数器中断

下表是根据上述约定给出的处理器中断路由配置寄存器的配置表，在此不考虑中断负载均衡，芯片内部所有的中断源都路由到第一个处理器核上。

表 6-5 3A/B+7A 芯片内部中断路由说明

中断源	中断号	路由至	路由 entry 值	说明
Sys_int0	16/级联	0 号核 IP3	0x21	
Sys_int1	17/级联	0 号核 IP3	0x21	
Sys_int2	18/级联	0 号核 IP3	0x21	
Sys_int3	19/级联	0 号核 IP3	0x21	
Pci _int0	20	0 号核 IP5	0x81	中断号与具体槽位相关，中断号对应 INTA 的号码
Pci _int1	21	0 号核 IP5	0x81	
Pci _int2	22	0 号核 IP5	0x81	
Pci _int3	23	0 号核 IP5	0x81	
MT0	24	0 号核 IP5	0x81	
MT1	25	0 号核 IP5	0x81	
LPC/UART	26	0 号核 IP2	0x11	
DRR INT0	27	0 号核 IP5	0x81	
DRR INT1	28	0 号核 IP5	0x81	
Barrier	29	0 号核 IP5	0x81	
保留	-	0 号核 IP5	0x81	
PCI-perr&serr	31	0 号核 IP5	0x81	
HT0 INT0	级联	0 号核 IP3	0x21	0 号 HT 对应的中断位。其中连接 HT 南桥时，HT0-INT0 与 HT0-INT1 实际作为级联通过，不直接对应中断设备。
HT0 INT1	级联	0 号核 IP3	0x21	
HT0 INT2	级联	0 号核 IP3	0x21	
HT0 INT3	级联	0 号核 IP3	0x21	
HT0 INT4	级联	0 号核 IP3	0x21	
HT0 INT5	级联	0 号核 IP3	0x21	
HT0 INT6	级联	0 号核 IP3	0x21	

HT0 INT7	级联	0 号核 IP3	0x21	
HT1 INT0	级联	0 号核 IP3	0x2F	1 号 HT 对应的中断位
HT1 INT1	级联	0 号核 IP3	0x2F	
HT1 INT2	级联	0 号核 IP3	0x21	
HT1 INT3	级联	0 号核 IP3	0x21	
HT1 INT4	级联	0 号核 IP3	0x21	
HT1 INT5	级联	0 号核 IP3	0x21	
HT1 INT6	级联	0 号核 IP3	0x21	
HT1 INT7	级联	0 号核 IP3	0x21	

- 3A/B+7A 的地址空间分布

表 6-6 3A/B+7A 芯片地址空间分布表

	起始地址	结束地址	说明
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	内存控制器 0
地址 1	0x0000_0000_1000_0000	0x0000_0000_17FF_FFFF	保留
地址 2	0x0000_0000_1800_0000	0x000 0_0000_1801_FFFF	32 位模式下 7A LPC 的 IO 空间
地址 3	0x0000_0000_1802_0000	0x0000_0000_19FF_FFFF	32 位模式下 7A PCI 的 IO 空间
地址 4	0x0000_0000_1A00_0000	0x0000_0000_1BFF_FFFF	32 位模式下 7A PCI 的配置空间
地址 5	0x0000_0000_1C00_0000	0x0000_0000_1DFF_FFFF	LPC Memory
地址 6	0x0000_0000_1FC0_0000	0x0000_0000_1FCF_FFFF	LPC Boot
地址 7	0x0000_0000_1FD0_0000	0x0000_0000_1FDF_FFFF	PCI IO 空间
地址 8	0x0000_0000_1FE0_0000	0x0000_0000_1FE0_00FF	PCI 控制器配置空间
地址 9	0x0000_0000_1FE0_0100	0x0000_0000_1FE0_01DF	IO 寄存器空间
地址 10	0x0000_0000_1FE0_01E0	0x0000_0000_1FE0_01E7	UART 0
地址 11	0x0000_0000_1FE0_01E8	0x0000_0000_1FE0_01EF	UART 1
地址 12	0x0000_0000_1FE0_01F0	0x0000_0000_1FE0_01FF	SPI
地址 13	0x0000_0000_1FE0_0200	0x0000_0000_1FE0_02FF	LPC Register
地址 14	0x0000_0000_1FE8_0000	0x0000_0000_1FE8_FFFF	PCI 配置空间
地址 15	0x0000_0000_1FF0_0000	0x0000_0000_1FF0_FFFF	LPC I/O
地址 16	0x0000_0000_2000_0000	0x0000_0000_3FFF_FFFF	7A GPU 访问空间 （因 7A GPU 只支持低 2G 内存直接访问能力，且此区间内存（0～0x10000000）无法满足应用需求，需单独开辟一段地址供其使用，地址空间的 0x20000000～0x40000000-1 通过 HT 转换映射内存的高 512M）
地址 17	0x0000_0000_4000_0000	0x0000_0000_7FFF_FFFF	HT1 MEM 空间
地址 18	0x0000_0000_8000_0000	0x0000_0000_8FFF_FFFF	保留
地址 19	0x0000_0000_9000_0000	0x0000_0001_7FFF_FFFF	以内存大小 4G 为例
地址 20	0x0000_0E00_1000_0000	0x0000_0E00_1000_0FFF	7A 中断控制器空间
地址 21	0x0000_0E00_1000_1000	0x0000_0E00_1000_1FFF	7A 中 HPET 寄存器空间
地址 22	0x0000_0E00_1000_2000	0x0000_0E00_1000_2FFF	7A 中 LPC 控制寄存器空间
地址 23	0x0000_0E00_1001_0000	0x0000_0E00_1001_FFFF	7A 中 confbus 空间
地址 24	0x0000_0E00_1008_0000	0x0000_0E00_100F_FFFF	7A 中 misc 设备寄存器空间
地址 25	0x0000_0E00_1200_0000	0x0000_0E00_13FF_FFFF	7A 中 LPC MEM 空间

地址 26	0x0000_0E00_4000_0000	0x0000_0E00_7FFF_FFFF	7A 中 PCI MEM 空间
地址 27	0x0000_0EFD_FC00_0000	0x0000_0EFD_FC01_FFFF	32 位模式下 7A LPC 的 IO 空间
地址 28	0x0000_0EFD_FC02_0000	0x0000_0EFD_FDFF_FFFF	32 位模式下 7A PCI 的 IO 空间
地址 29	0x0000_0EFD_FE00_0000	0x0000_0EFD_FF00_0000	32 位模式下 7A PCI 的配置访问空间
地址 30	0x0000_0C00_0000_0000	0x0000_0FFF_FFFF_FFFF	HT1 控制器，各种空间
地址 31	0x0000_1000_0000_0000	0x0000_3FFF_FFFF_FFFF	猜测空间
地址 32	其它地址		系统配置空间

6.2.3.3 修改 3A/3B+780E 中断及地址空间约定

1. 3A/B+780E 中断系统描述

3A/B+780E 板上，所有的外设中断都是路由到 0 号核（即 CPU 0），其中片上的 UART 控制器的中断直接进入 Int Controller 的 UART/LPC 脚，最后路由到 CPU 0 的 IP3，而片外的 780E 桥片上的中断则是通过 HT1 控制器进入到 Int Controller 的 HT1-0 脚，最后路由到 CPU 0 的 IP2，其示意图如下。

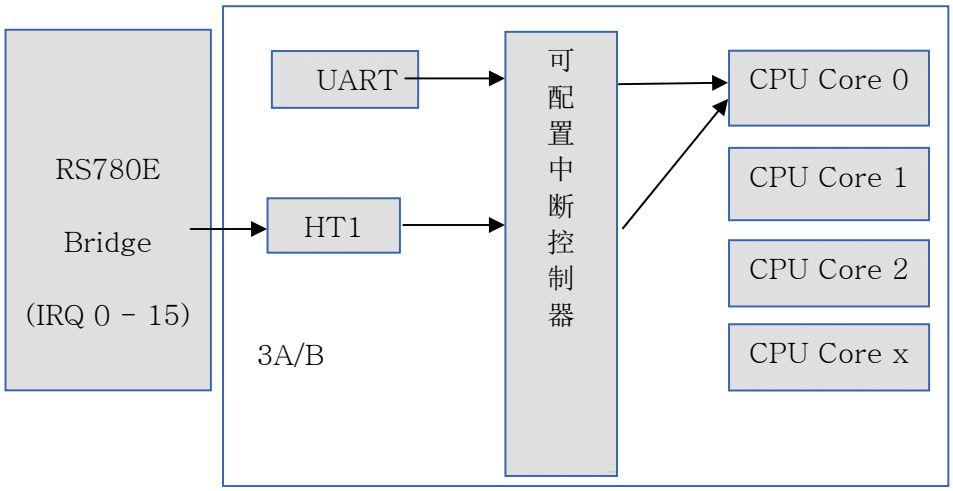


图 6-3 3A/B+780E 中断路由示意

2. 3A/B+780E 中断号分配约定

0-15 号保留分配给连接诸如 8259 的传统设备的中断。原则上，龙芯芯片 HT 总线接口和芯片 PCI 总线接口不会同时使用，因此，8259 中断控制寄存器只可能唯一出现在 HT 或者 PCI 总线上。由于历史原因，8259 中断控制寄存器也不是所有的中断号都可以使用，如表 C.1 所示的中断号就被保留。此外，剩下的中断号有些会被桥片保留，具体的需要查询桥片的中断控制器相关说明。

表 6-7 0-15 号中断说明

中断号	中断源	说明
0	HPET	HPET 高精度定时器
1	I8042	XT-PIC 键盘
2	级联	XT-PIC
3		
4		
5		
6		
7	SCI	系统控制中断（用于笔记本电脑 Fn 功能键）
8	RTC	XT-PIC 实时时钟
9		
10		
11		
12	I8042	XT-PIC 鼠标
13		
14	ide0	XT-PIC 硬盘
15	Ide1	XT-PIC 硬盘

16-47 号对应分配给芯片内部的中断控制器，其中级联部分不直接连接设备，不需要进行中断号分配。

64-319 号：使用 780E 桥片的约定如下，64-319 号分配给芯片内部的 HT 中断控制器。但由于 HT 套片上的 8259 控制器通过 HT 的 0-15 位中断向量连接 CPU，64-79 号中断实际作为对应 8259 的 16 个中断源，不对应中断设备。

3. 3A/B+780E 的中断路由约定

IP0~IP7 用于标识发生了哪些中断。详情参见 3A/3B+7A 的中断路由约定部分的定义。

下表是根据上述约定给出的处理器中断路由配置寄存器的配置表，在此不考虑中断负载均衡，芯片内部所有的中断源都路由到第一个处理器核上。

表 6-8 3A/B+780E 芯片内部中断路由说明

中断源	中断号	路由至	路由 entry 值	说明
Sys_int0	16/级联	0 号核 IP4	0x41	直接连接中断设备时进行分配。若连接 PCI 南桥则仅用于级联，不分配中断设备。
Sys_int1	17/级联	0 号核 IP4	0x41	
Sys_int2	18/级联	0 号核 IP4	0x41	
Sys_int3	19/级联	0 号核 IP4	0x41	
Pci _int0	20	0 号核 IP5	0x81	中断号与具体槽位相关，中断号对应 INTA 的号码
Pci _int1	21	0 号核 IP5	0x81	
Pci _int2	22	0 号核 IP5	0x81	
Pci _int3	23	0 号核 IP5	0x81	
MT0	24	0 号核 0IP5	0x81	

MT1	25	0 号核 IP5	0x81	
LPC/UART	26	0 号核 IP2	0x11	
DRR INT0	27	0 号核 IP5	0x81	
DRR INT1	28	0 号核 IP5	0x81	
Barrier	29	0 号核 IP5	0x81	
保留	-	0 号核 IP5	0x81	
PCI-perr&serr	31	0 号核 IP5	0x81	
HT0 INT0	级联	0 号核 IP3	0x21	0 号 HT 对应的中断位。其中连接 HT 南桥时，HT0-INT0 与 HT0-INT1 实际作为级联通过，不直接对应中断设备。
HT0 INT1	级联	0 号核 IP3	0x21	
HT0 INT2	级联	0 号核 IP3	0x21	
HT0 INT3	级联	0 号核 IP3	0x21	
HT0 INT4	级联	0 号核 IP3	0x21	
HT0 INT5	级联	0 号核 IP3	0x21	
HT0 INT6	级联	0 号核 IP3	0x21	
HT0 INT7	级联	0 号核 IP3	0x21	
HT1 INT0	级联	0 号核 IP3	0x21	1 号 HT 对应的中断位
HT1 INT1	级联	0 号核 IP3	0x21	
HT1 INT2	级联	0 号核 IP3	0x21	
HT1 INT3	级联	0 号核 IP3	0x21	
HT1 INT4	级联	0 号核 IP3	0x21	
HT1 INT5	级联	0 号核 IP3	0x21	
HT1 INT6	级联	0 号核 IP3	0x21	
HT1 INT7	级联	0 号核 IP3	0x21	

4. 3A/B+780E 芯片地址空间

使用这种模式，为了兼容之前的 32 位的固件代码，需要配置一级交叉开关，把 44 位的访问 HT 空间的地址转换成 32 位地址，芯片的地址空间如下表所示：

表 6-9 3A/B+780E 芯片地址空间分布表

	起始地址	结束地址	说明
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	内存控制器 0
地址 1	0x0000_0000_1000_0000	0x0000_0000_17FF_FFFF	保留
地址 2	0x0000_0000_1800_0000	0x0000_0000_19FF_FFFF	HT1 IO 空间
地址 3	0x0000_0000_1A00_0000	0x0000_0000_1BFF_FFFF	HT1 配置空间
地址 4	0x0000_0000_1C00_0000	0x0000_0000_1DFF_FFFF	LPC Memory
地址 5	0x0000_0000_1FC0_0000	0x0000_0000_1FCF_FFFF	LPC Boot
地址 6	0x0000_0000_1FD0_0000	0x0000_0000_1FDF_FFFF	PCI IO 空间
地址 7	0x0000_0000_1FE0_0000	0x0000_0000_1FE0_00FF	PCI 控制器配置空间
地址 8	0x0000_0000_1FE0_0100	0x0000_0000_1FE0_01DF	IO 寄存器空间
地址 9	0x0000_0000_1FE0_01E0	0x0000_0000_1FE0_01E7	UART 0
地址 10	0x0000_0000_1FE0_01E8	0x0000_0000_1FE0_01EF	UART 1
地址 11	0x0000_0000_1FE0_01F0	0x0000_0000_1FE0_01FF	SPI
地址 12	0x0000_0000_1FE0_0200	0x0000_0000_1FE0_02FF	LPC Register
地址 13	0x0000_0000_1FE8_0000	0x0000_0000_1FE8_FFFF	PCI 配置空间
地址 14	0x0000_0000_1FF0_0000	0x0000_0000_1FF0_FFFF	LPC I/O
地址 15	0x0000_0000_4000_0000	0x0000_0000_7FFF_FFFF	HT1 MEM 空间

地址 16	0x0000_0000_8000_0000	0x0000_0000_8FFF_FFFF	保留
地址 17	0x0000_0000_9000_0000	0x0000_0001_7FFF_FFFF	以内存大小 4G 为例,其中 0x0000_0000_FE00_0000 – 0x0000_0000_FFFF_FFFF 为 780E 桥片用做 MSI 中断等地址空间, 必须保留
地址 18	0x0000_0C00_0000_0000	0x0000_0FFF_FFFF_FFFF	HT1 控制器，各种空间
地址 19	0x0000_1000_0000_0000	0x0000_3FFF_FFFF_FFFF	猜测空间
地址 20	其它地址		系统配置空间

版权声明

龙芯中科技术有限公司版权所有。

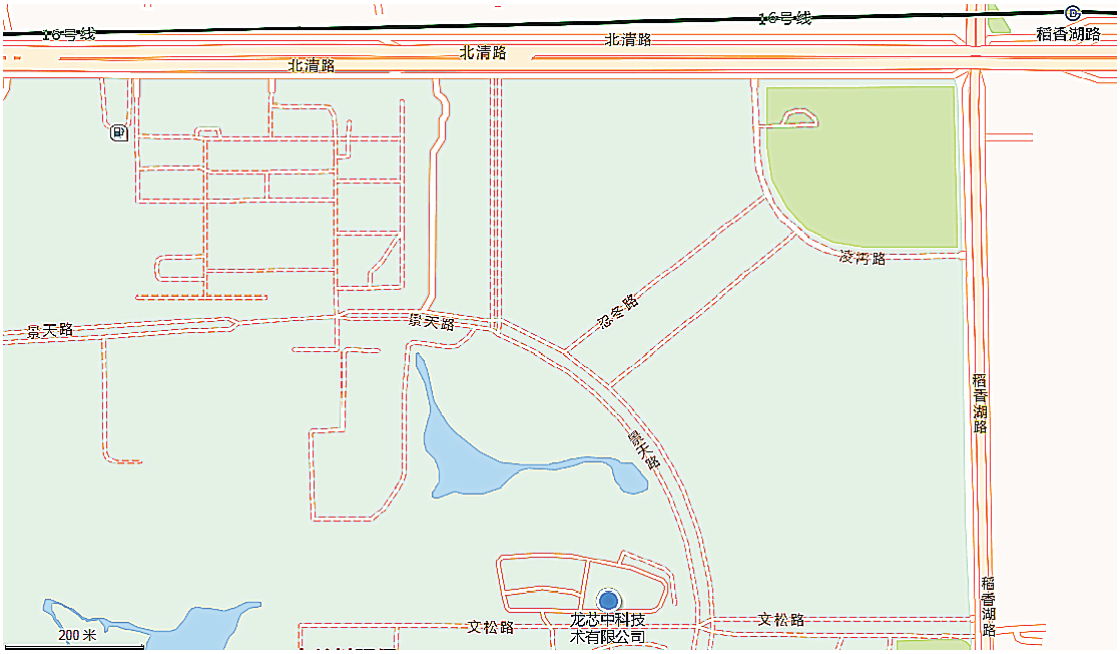
LOONGSON是龙芯中科技术有限公司的注册商标。本文中所涉及的其他商标或产品名称均为各自拥有者的商标或产品名称。

本文中的信息若有更改，恕不另行通知。虽然已尽力确保本文的完整性和准确性，但龙芯中科技术有限公司对本文的内容不作任何保证。龙芯中科技术有限公司对本文中包含的错误或遗漏，或者因使用本文引发的任何损失概不负责。

未经龙芯中科技术有限公司许可，任何个人和组织均不得以任何手段与形式对本文进行复制或传播。

龙芯中科技术有限公司

附　　龙芯中科技术有限公司在地图上的位置



龙芯中科技术有限公司　　安全应用事业部

地址：北京市海淀区温泉镇中关村环保科技示范园龙芯产业园 2 号楼

邮编：100095　　公司传真：010-62600826

联系电话：010-62546668 转 1801

联系邮箱：humingchang@loongson.cn

公司网址：http://www.loongson.cn

龙芯社区：http://www.loongnix.org

下载网址：http://ftp.loongnix.org/