Q/LS

Q/LS 0013-2014

龙 芯 中 科 技 术 有 限 公 司 企 业 标 准

龙芯 CPU 开发系统固件与内核接口规范

2014-04-01 发布　　2014-06-01 实施

龙 芯 中 科 技 术 有 限 公 司　　批 准

目　次

前　言

本规范是龙芯中科技术有限公司制定的企业规范，暂无国家相关行业通用规范可参考。

本规范涉及到固件与内核之间的传参、中断分配及地址空间划分等方面，主要介绍龙芯板卡固件与内核之间的接口定义，地址空间分配，中断在内核和固件之间的划分、中断号的分配及相关硬件布线约定，最后提到了 SMBIOS 中需要实现的类型约定。接口定义涉及到固件与内核间信息传递的数据结构定义，固件使用何种方式传递这些数据结构及内核如何解析等方面。

本规范的起草单位：龙芯中科技术有限公司、中电科技（北京）有限公司、江苏中科梦兰电子科技有限公司、中标软件有限公司。

本规范的主要起草人：孟小甫、司志莹、王焕东、成修治、陈华才、陈小春、何曌君、刘绍宗、苏孟豪、乔崇、夏启超。

本规范审核人：王剑，高翔，黄沛，邱吉，李文刚，王玉钱，袁俊卿。

本规范批准人：胡伟武。

龙芯 CPU 开发系统固件与内核接口规范

1 范围

本规范规定了龙芯 CPU 开发系统的地址空间、中断配置、固件与内核接口传参实现及 SMBIOS 实现约定的要求。本规范适用于龙芯 2 号和 3 号系列 CPU 开发系统。建议其它系统厂商遵循此规范开发相关产品。

本规范由通用规范和详细规范组成。正文及附录 A、B 为通用规范，通用规范描述一般性的约定。附录 C、附录 D 为详细规范，详细规范针对不同开发系统进行具体约定。当通用规范与详细规范不一致时，以详细规范为准。

2 术语与定义

本规范所用术语定义如下：

a) 固件：Firmware，写入 ROM、EPROM 等非易失存储器中的程序，负责控制和协调集成电路。

b) BIOS：基本输入输出系统，Basic Input Output System，一组固化到主板上一个 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。BIOS 与硬件系统集成在一起，也被称为固件，本规范中固件和 BIOS 不做区分。

c) UEFI：统一的可扩展固定接口，Unified Extensible Firmware Interface，是 Intel 为全新类型的 PC 固件的体系结构、接口和服务提出的建议标准。主要目的是提供在 OS 加载之前在所有平台上一致、正确指定的启动服务，被看做是有近 20 多年历史的 PC BIOS 的继任者。

d) PMON：MIPS 架构机器上使用的一种兼有 BIOS 和 boot loader 部分功能的开放源码软件。

e) SMBIOS(System Management BIOS)：是主板或系统制造者以标准格式显示产品管理信息所需遵循的统一规范。DMI (Desktop Management Interface)是帮助收集电脑系统信息的管理系统，DMI 信息的收集必须在严格遵照 SMBIOS 规范的前提下进行。SMBIOS 和 DMI 是由行业指导机构 Desktop Management Task Force (DMTF)起草的开放性的技术标准。

f) HT（HyperTransport）：是一种为主板上的集成电路互连而设计的端到端总线技术，目的是加快芯片间的数据传输速度。HT 通常指 CPU 到主板芯片（或北桥）之间的连接总线，即 HT 总线。类似于 Intel 平台中的前端总线(FSB)，HT 按技术规格分有 HT1.0、HT2.0、HT3.0、HT3.1。

g) PCI（Peripheral Component Interconnect）：是连接电子计算机主板和外部设备的总线标准，用于定义局部总线的标准。此标准允许在计算机内安装多达 10 个遵从 PCI 标准的扩展卡。

3 架构关系

龙芯 PC 产品的固件与内核接口在系统各软件之间所处的层次关系如图 1 所示：

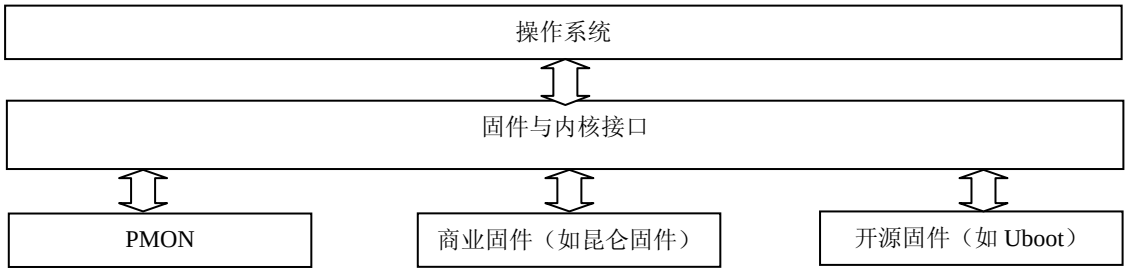

图 1 接口与内核和固件之间的关系

4 地址空间规范

4.1 芯片地址空间表

龙芯处理器可以通过芯片内的 PCI 接口连接外部桥片，也可以使用 HT 端口连接桥片。由于 HT 端口速度更快、位长更宽，现在多使用 HT1 端口连接桥片的模式。不同芯片的地址空间配置约定如各自的附录所示。

4.2 地址空间表

地址空间设计的规则做如下约定：

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，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～高端内存基址-0x1000_0001（缺省是 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-0x0000_0001 为保留空洞；

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

地址空间划分如图 2 所示：

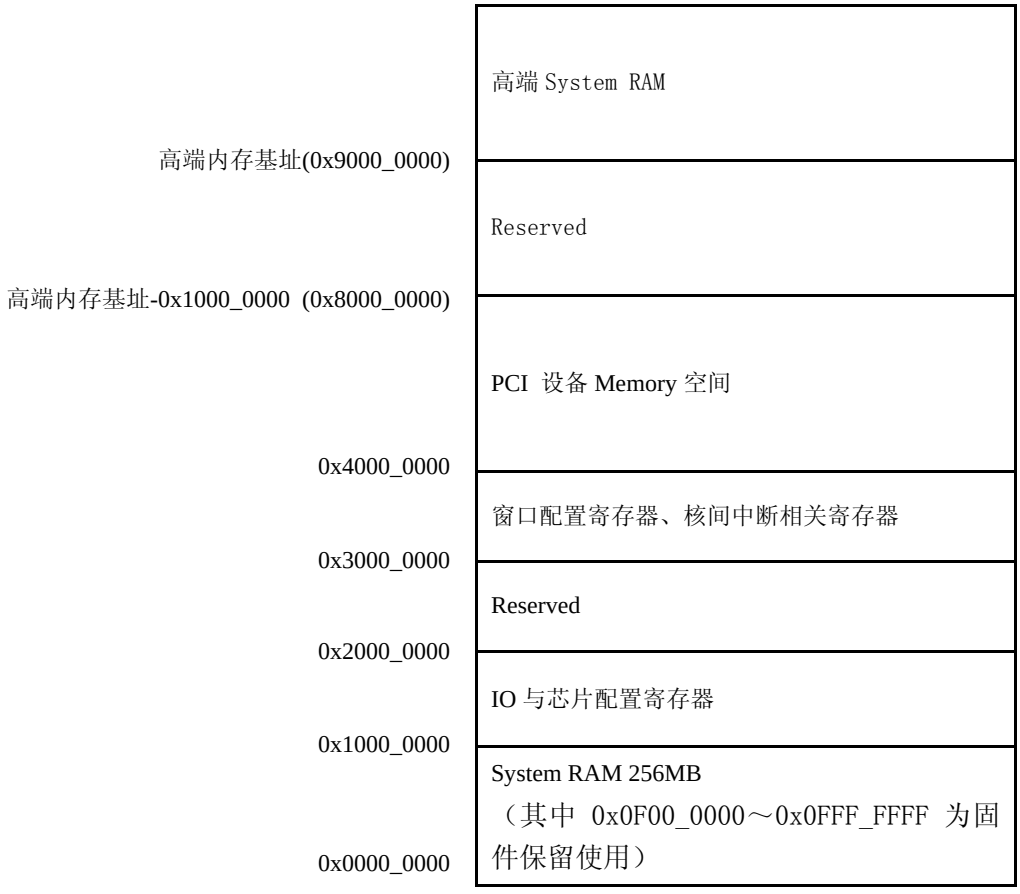

图 2 地址空间分布图

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 1：

表 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 2：

表 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 3：

表 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

5 中断配置规范

5.1 固件与内核关于中断的划分界面

5.1.1 固件与内核关于中断的分工

龙芯芯片内部集成了中断控制器用于处理自带串口、PCI 总线等中断，又可以通过 PCI、HT 接口所连接桥片上的中断控制器来处理外接设备中断。因此，龙芯芯片系统的中断控制较为复杂。中断配置规范的目的是划清固件和操作系统关于中断的配置范围，统一龙芯芯片操作系统的中断分配。为此，进行如下约定：

a) 桥片的中断路由由固件来做，其他的中断分配和路由由内核来做；
b) 固件负责完成桥片上的 PCI/PCIE 设备和插槽的中断号申请和路由，操作系统内核完成中断号的分配和芯片内部的中断路由，好处是固件只和具体的板卡相关，内核负责 CPU 相关的初始化（包括 CPU 核间中断），这样一个内核可以适配不同板卡的固件；
c) 固件完成 PCI 设备扫描和配置后，根据桥片手册、设备类型及硬件原理图，向每一个 PCI 设备或 PCI/PCIE 插槽上可能的设备的中断线寄存器写入申请的中断线号，同时需要对桥片上的中断控制器进行配置，保证它能路由到 HT 的 0-16 位中断向量的相应位置；
d) 操作系统内核在分配中断号时，根据中断线寄存器的值进行分配。此外，操作系统内核还负责对芯片内部的中断控制器进行设置；
e) 内核不再管 CPU，板卡相关中断的具体配置，所有信息均由固件传递过来，并通过 cpu_type 的类型选择不同的平台支持，这样可以做到一个内核二进制在不同平台和 CPU 上都适用；
f) 如果使用 MSI/MSI-X，由内核根据桥片的特性来分配中断，固件不再负责。

5.1.2 中断的处理方式

如果有 8259，其中断处理方式约定如下：

a) 每次从 HT 中断向量寄存器中读取一个中断位，然后进行中断处理，并且清除 HT 相应的中断向量位，这种方式使用了 8259 默认的中断优先级；
b) 固件需要将当前的设备中断号传递给内核，内核首先获知有哪些外设中断，并且给相应位使能。

其他芯片的中断如有约定参见各自的附录。

5.2 中断号码的分配及各类芯片内部中断路由约定

0-15 号中断：表 4 中所列的中断号需要保留：

表 4 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

15 号以后的中断号及芯片内部中断路由针对不同开发系统有不同约定，3A/B+780E 开发系统可参见附录 C，3A/B+2H 开发系统可参见附录 D。

5.3 与板级硬件设计有关的约定

开发系统必须使用桥片上的 LPC,不使用 CPU 的。

6 固件与内核接口传参规范

6.1 传参数据结构的约定

为便于同一个内核适配不同板卡、处理器，增强内核兼容性，龙芯平台规定由固件向内核提供 CPU 类型、中断号、总线信息、内存大小、pci-mem 地址、video_bios 地址、busclock 等信息。提供方式为：传递参数。该信息为内核必须获取信息，供内核解析结构体信息并启动运行。其中，同板卡不同芯片之间的区别可以通过 cpu_type,processor_id,PIC_type,ht_enable 以及 board_device 等结构联合描述清楚。

龙芯 2 号和 3 号开发系统采用附录 A 中的数据结构进行传参,其对应的头文件参见附录 B 所示,以后的扩展需要兼容此结构。

6.2 传参的实现

6.2.1 概述

本节的描述以 PMON 和 Linux 内核的实现为例来说明,但传参初始结构体指针 bp 规定放在 a2 寄存器中。

6.2.2 固件中实现

在固件中，每一个结构体针对一类型的功能或服务，具体定义在 boot_param.c 中，如 efi_memory_map 是针对系统的 memory 信息，efi_cpuinfo_loongson 处理 CPU 的相关信息等，为了给

UEFI 的模块开发预留空间，对每一类的服务或功能提供一个初始化函数，如：

```
Int init_boot_param(struct boot_params *bp)
{
        bp->video_bios_addr = 0xc3f00000;
        init_efi(&(bp->efi));
        return bp;
}
```

struct efi_memory_map_loongson *init_memory_map(){}中给每一项具体赋值。

6.2.3 参数的传递

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。

6.2.4 内核中实现

在内核中，内核的实现主要是对该结构的解析工作，目前是在 env.c 中直接解析。首先定义和固件完全一样的结构体,在 head.S 中把 a2 寄存器读出，赋值给 fw_arg2，依次读出固件传过的信息，并对相应的结构体变量赋值即可；如：

```
struct boot_params *bp;
bp = (struct boot_params)fw_arg2;
emap = bp + bp->memory_offset;
```

等等逐次赋值解析，则相应的信息均可被内核使用。

注意：内核与固件的结构体必须完全一致，任何类型改动，都会导致内核无法启动。

7 SMBIOS 规范的实现约定

SMBIOS 是主板或系统制造者以标准格式显示产品管理信息所需遵循的统一规范。 DMI (Desktop Management Interfacece, DMI)就是帮助收集电脑系统信息的管理系统，DMI 信息的收集必须在严格遵照 SMBIOS 规范的前提下进行。 SMBIOS 和 DMI 是由行业指导机构 Desktop Management Task Force (DMTF)起草的开放性的技术标准，其中，DMI 设计适用于任何的平台和操作系统。 DMI 充当了管理工具和系统层之间接口的角色。它建立了标准的可管理系统更加方便了电脑厂商和用户对系统的了解。DMI 的主要组成部分是 Management Information Format (MIF)数据库。这个数据库包括了所有有关电脑系统和配件的信息。通过 DMI，用户可以获取序列号、电脑厂商、串口信息以及其它系统配件信息。

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：

a) BIOS 信息(Type 0)
b) 系统信息 (Type 1)
c) 产品信息 (Type 2)
d) 处理器信息 (Type 4)
e) 物理存储阵列 (Type 16)
f) 存储设备(Type 17)

g) 温度传感器 (Type 28)

h) 表格结束指示 (Type 127)

其中:

Type0 BIOS 信息中的固件版本(BIOS Version)约定格式见表 5:

表 5 BIOS 信息约定格式说明及举例

说明 / 举例	厂商代号	连字符	固件名称	连字符	版本	连字符	发布日期
PMON3	Loongson	-	PMON	-	V3.0.1	-	20120808

Type2 的产品(Product)信息格式约定见表 6:

表 6 产品信息约定格式说明及举例

说明 / 举例	厂商代号	连字符	CPU 型号	连字符	桥片	连字符	CPU 路数	连字符	板卡版本号	连字符	自定义信息
2HSOC 开发板	Loongson	-	LS2H	-	SOC	-	1w	-	V1.01	-	DEV
3A780E 双路开发板	Loongson	-	LS3A	-	RS780E	-	2w	-	V1.03	-	DEV
龙梦 3A780E 双路带 BMC 控制的服务器板	Lemote	-	LS3A	-	RS780E	-	2wBMC	-	V1.00	-	server

注:CPU 路数的后缀”w”是小写以便和后面跟的诸如 BMC(Baseboard Management Controller,远程管理控制器)或 TCM(*Trust* Cryptography Module, 可信密码模块)区分。

8 兼容性约定

内核加载虚地址从 0x8020_0000 开始。

9 总结

本传参规范的提出和制定旨在规范龙芯的固件和内核接口，重点规定了地址空间、中断分配、传参数据结构、SMBIOS 实现约定等方面，使得内核具有更广泛的适应性及兼容性，能有效解决内核对具体板卡设备的依赖性，有利于龙芯系列产品基础软件的规范和统一。

附录 A　龙芯 CPU 开发系统传参数据结构

A.1 固件与内核接口定义

本节规定了固件在加载内核阶段的接口定义，定义了接口信息结构体、预留了 UEFI 可扩展信息。信息结构体定义如下：

```
#define u64 phys_t
#define u64 unsigned long long
#define u32 unsigned int
#define u16 unsigned short
#define u8  unsigned char
```

注：为避免固件 32 位，内核 64 位的问题，在代码实现时，建议把指针替换成 u64，并在使用处注明 u64 代表什么结构体或定义，在本规范中不给出直接替换，否则会混淆结构含义，且对 UEFI/内核或 PMON64/内核 64 不具备兼容性。

传参主要结构体，如表 5 所示：

表 A.1 传参结构一览表

结构体	描述
struct boot_params	启动参数结构体，必选
Srtuct efi	Efi 入口结构体，必选
Struct smbios_tables	SMBIOS 相关信息描述，必选
Struct loongson_param	其他结构体相对的偏移地址
Struct interface_info	接口信息，描述规范本身的信息
Struct loongson_efi_memory_map	龙芯内存地址空间信息，必选
Struct system_loongson	龙芯 system 的信息，如单路还是双路，是否用 numa 等
Struct efi_cpuinfo_loongson	与龙芯 CPU 相关的信息，如频率，型号等，必选
Struct irq_source_routing_table	中断信息，包含中断控制器，中断号等等
Struct board_device	板载设备的描述结构
Struct uart_device	串口相关的设备结构
Struct sensor_device	温度、风扇传感器相关的设备结构
struct loongson_special_attribute	龙芯特殊应用的预留接口

注：各板卡可根据需要选择必须实现的结构体，但如果选用，需按照本章约定的结构体进行传参，不能自行重新设计结构体，为保证兼容性，如果需要扩充本规范没有的结构体，可加在结构体 boot_params 的最后添加，并在各自的附录中给以说明。

A.2 boot_params

```
struct boot_params {
        struct screen_info screen_info;  /*info about screen，reserved*/
        struct sys_desc_table sys_desc_table; /* tables in bios,reserved*/
        struct efi efi; /*efi struct */
        struct efi_info efi_info; /*information of efi, reserved*/
};
```

固件最终是将 boot_params 的结构体指针 bp 传递给内核，这样有一个统一的接口结构体,screen_info,sys_desc_table，efi_info 等可以暂时保留。

必须定义：strcut efi efi。

表 A.2 boot_params 结构体描述

成员	意义
screen_info	屏幕信息，包括打印到屏幕上的坐标
sys_desc_table	与系统相关的描述符表
Efi	efi 结构
efi_info	保存与 efi 自身相关的信息

A.3 efi

```
struct efi {
        efi_system_table_t systab;       /* EFI system table */
        u64  mps;              /* MPS table */
        u64  acpi;            /* ACPI table  (IA64 ext 0.71) */
        u64 acpi20;            /* ACPI table  (ACPI 2.0) */
        struct smbios_tables smbios;            /* SM bios table */
        u64 sal_systab;         /* SAL system table */
        u64 boot_info;         /* boot info table */
        /*have non this struct for PMON
          efi_get_time_t *get_time;
          efi_set_time_t *set_time;
          efi_get_wakeup_time_t *get_wakeup_time;
          efi_set_wakeup_time_t *set_wakeup_time;
          efi_get_variable_t *get_variable;
          efi_get_next_variable_t *get_next_variable;
          efi_set_variable_t *set_variable;
          efi_get_next_high_mono_count_t *get_next_high_mono_count;
          efi_reset_system_t *reset_system;
          efi_set_virtual_address_map_t *set_virtual_address_map;
         */
};
```

该结构体中包含了与 efi 服务相关的很多信息，图 system_table, acpi,smbios,time 服务等，这些信息在当前 PMON 的基础上无法实现，是作为 UEFI 的传参结构体预留扩展预留的，目前主要使用的信息定义在 smbios 中。该结构体是标准的 UEFI 的结构定义。大部分保留，仅适用 smbios。

必须定义：strcut smbios_tables smbios;

表 A.3 efi 结构体描述

成员	意义
Systable	systable,需要在 UEFI 中实现；
Mps	电源管理相关信息；
Acpi	Acpi 的接口；
acpi20	Acpi 补充信息；
Smbios	描述与厂商相关的信息，如 CPU,memory,irq 等；

Sal_systable	SAL_Table 的信息；
boot_info	与启动相关的参数信息；
Get_time	获得当前的时间；
Set_time	设定当前的时间；
Get_wakeup_time	取得唤醒时间；
Set_wakeup_time	设定唤醒时间；
Get_variable	获得可变参数；
Get_next_variable	获得下一项可变参数；
Set_variable	设定变量；
Get_next_high_mono_count	
reset_system	调用重启，开关机函数；
Set_virtual_address_map	虚存空间地址图；

A.4 smbios_tables

```
struct smbios_tables {
        u16 vers;         /* version of smbios */
        u64 vga_bios;      /*vga_bios address */
        struct loongson_param lp;
};
```

Smbios_tables 可遵循 SMBIOS 规范，SMBIOS 是主板或系统制造者以标准格式显示产品管理信息所需遵循的统一规范，可以给出序列号、电脑厂商信息、固件、操作系统、主板、内存、扩展槽及扩展接口的详细。SMBIOS 给出版本信息，vga_bios 基地址，及龙芯平台相关的结构体信息。其中 vers 是标识本结构体的版本号，vga_bios 保存 vga_bios 的基地址，loongson_param 则是与龙芯体系相关的结构体；vga_bios 告诉内核 vga_bios 的基地址，在龙芯上使用 x86 模拟的显卡的处理，该地址必须也是龙芯板卡的特性之一,如果当前使用的是自带 vbios 的独立显卡，vga_bios 的值默认设定为 0，而不能是个未赋值的随机数。

必须定义：原则上，这结构体的每一项都必须定义，任何一个如果不定义或者定义出错，内核就无法启动。如 vga_bios，不定义图形界面启动不起来。

A.5 loongson_param

```
struct loongson_params{
        u64 memory_offset;/*efi_memory_map_loongson struct offset*/
        u64 cpu_offset;  /*efi_cpuinfo_loongson struct offset*/
        u64 system_offset;  /*system_info struct offset*/
        u64 irq_offset;  /*irq_source_routing_table struct offset*/
        u64 interface_offset;  /*interface_info struct offset*/
        u64 special_offset;  /*loongson_special_attribute struct offset*/
        u64 boarddev_table_offset;  /*board_device offset*/
};
```

该结构保存所有结构体相对 loong_param 结构体地址的偏移，memory_offset 是内存结构体的偏移，cpu_offset 是 CPU 结构体的偏移，system_offset 是系统结构体的偏移，irq_offset 是龙芯中断控制器相关的偏移，special_offset 是龙芯特殊特性结构体的偏移，board_offset 是龙芯板载设备结构体偏移。该结构体中只存放各个结构体的偏移，因此不能在填充其他变量。表 C.4 为和龙芯平台相

关的信息一览表。

表 A.4 loongson_params 结构体描述

成员	意义
Interface_info	接口信息，描述规范本身的信息；
efi_memory_map_loongson	龙芯内存地址空间信息；
system_info	龙芯 system 的信息，如单路还是双路，是否用 numa 等；
efi_cpuinfo_loongson	与龙芯 CPU 相关的信息，如频率，型号等；
irq_source_info	中断信息，包含中断控制器，中断号等等；
Special_attr	龙芯的其他特殊属性；
Boarddev_table	板载设备的描述结构；

A.6 board_devices

```
struct board_devices {
        u8 name[64]; /*hold the device name*/
        u32 num_resources; /*number of device_resource*/
        struct resource resource; /*for each device`s resource*/
        /*arch specific additions*/
        struct pdev_archdata archdata;
};
```

表 A.5 board_devices 结构体描述

成员	意义
Name	设备名称；
Num_resource	设备资源个数；
Resource	设备资源信息；
Data	设备相关的数据信息；

```
#define MAX_RESOUCR_NUMBER 128
struct resource {
        u64 start; /*resource start address*/
        u64 end; /*resource end address*/
        u8 name[32];
        u32 flags;
}resource[MAX_RESOUCR_NUMBER];
struct pdev_archdata {};
```

注：如果固件中实现了 SMbios 规范的 type2 board information，使用此结构体的 name 进行传递，传递板卡名称的约定见本规范第 7 节表 6。

A.7 interface_info

```
struct interface_info {
        u16 vers; /*version of the specificition*/
        u16 size;  /*size of this interface*/
        u8 flag;/*use or unuse*/
        u8 description[64]; /*description for each change*/
```

```
        u64 DoSuspend; /* NULL if not support */
}__attribute__((packed));
```

本结构提供接口的版本信息，vers 表示本规范的版本信息，可以用来却别不同版本下不同的功能，重大改动版本使用 v0，v1,v2 表示，小改动版本使用 v0.1,v1.1,,v1.2 等表示；flag 标识当前内核或固件有没有使用本规范。这个作为接口升级和变更标记，同时区别可以规范前后固件和内核的情况，使得内核的处理和使用有一定的兼容性。description[64]简要描述版本更迭重大变化。

必须定义：vers，flag。

表 A.6 interface_info 结构体描述

成员	意义
Vers	Interface_info 的版本信息，标识规范的版本更迭；
Size	结构的大小，所占内存大小；
Flag	标识当前有没有使用规范；
Description	对规范的简单描述，及每次规范升级的关键内容描述等。

注：如果固件中实现了 SMBIOS 规范的 type0 BIOS Information，使用此结构体的 description 进行传递，传递固件名称的约定见本规范第 7 节表 5。

A.8 efi_memory_map_loongson

针对龙芯平台，定义了如下内存类型：

```
#define SYSTEM_RAM_LOW 1
#define SYSTEM_RAM_HIGH 2
#define MEM_RESERVED 3
#define PCI_IO 4
#define PCI_MEM 5
#define LOONGSON_CFG_REG 6
#define VIDEO_ROM 7
#define ADAPTER_ROM 8
#define ACPI_TABLE 9
#define SMBIOS_TABLE 10
#define MAX_MEMORY_TYPE 11
#define BOOT_MEM_MAP_MAX 128
struct efi_memory_map_loongson {
        u16 vers;/*version of efi_memory_map*/
        u32 nr_map; /*number of memory_maps*/
        u32 mem_freq;/*memory frequence*/
        struct mem_map{
                u32 node_id;/*recorde the node_id*/
                u32 mem_type;/*recorde the memory type*/
                u64 mem_start;/*memory map start address*/
                u32 mem_size; /*for each memory_map size,not the total size*/
        }map[BOOT_MEM_MAP_MAX];
}__attribute__((packed));
```

龙芯内存是统一编址的，整个内存是一张表，在本结构中，vers 表示当前 memory_map 的版本号，nr_map 是传递过来的内存表的个数，内核扫描传过来的全部的内存表，然后根据下面结构体中 node_id,mem_type 等两项判断传递过来的内存表是什么表，挂在哪个节点上。从而得到每个内存表的起始地址，大小等等信息,SMBIOS 使用此结构体向内存发布。

表 A.7 efi_memory_map_loongson 结构体描述

成员	意义
Vers	Efi_memory_map_loongson 的版本信息；
nr_map	内存表的个数，记录从固件传递过来的所有表的个数；
Mem_freq	内存的频率，ddr 频率
struct mem_map	表示每张表的信息的结构体；
	node_id :当前内存表挂在那个节点上； mem_type:当前内存表的类型； mem_start:当前表的起始地址； mem_size : 当前表的大小；

A.9 sytem_loongson

```
struct system_loongson {
        u16 vers;   //version of system_loongson
        u32 ccnuma_smp;//0:smp;1:numa;
        u32 sing_double_channel;//1:single; 2:double
        u32 nr_uarts;//number of uarts
        struct uart_device uarts[MAX_UARTS];
        u32 nr_sensors;//number of sensors
        struct sensor_device sensors[MAX_SENSORS];
        u8 has_ec;
        u8 ec_name[64];
        u64 ec_base_addr;
        u8 has_tcm;
        u8 tcm_name[64];
        u64 tcm_base_addr;
        u64 workarounds;
}__attribute__((packed));
```

这个结构体主要包含龙芯特性，如：系统是否用 ccnuma，当前是单路还是双路等等，也可以添加其他的特性。

ccnmua 用来判断，启动 numa-os 还是 smp；single_double 则选择单路还是双路；

在多核芯片中必须定义：ccnuma_smp，不定义影响性能

表 A.8 system_loongson 结构体描述

成员	意义
Vers	龙芯系统版本信息；
ccnuma_smp	0：代表 smp，1：代表 numa；
Sing_double_channel	1：代表单路，2：代表双路；
nr_uarts	串口数量
Uart_device uarts[MAX_UARTS]	串口设备数组

nr_sensors	传感器数量
sensor_device sensors[MAX_SENSORS]	传感器设备数组
Has_ec	是否存在 EC
ec_name[64]	用于创建 platform_device 的 EC 名字
ec_base_addr	EC 寄存器基地址
Has_tcm	是否存在 TCM
Tcm_name[64]	用于创建 platform_device 的 TCM 名字
Tcm_base_addr	TCM 寄存器基地址
Workarounds	根据有缺陷的机型做 workarounds，有厂家自定义,一般在 workarounds.h 里定义

A.10 uart_device

```
#define MAX_UARTS 64
struct uart_device {
        u32 iotype; /* see include/linux/serial_core.h */
        u32 uartclk;
        u32 int_offset;
        u64 uart_base;
}__attribute__((packed));
```

该结构体为描述串口相关信息 MAX_UARTS 为最大串口数量，本结构体是可选结构体，该结构体的访问通过 system_loongson 结构体寻址得到，而不是通过 smbios 结构体直接访问。

表 A.9 uart_device 结构体描述

成员	意义
Iotype	标示 IO 类型是 MMIO 还是 PortIO;
Uartclk	串口的时钟频率;
Int_offset	串口 IRQ 号的偏移量，MMIO 相对于 56，PortIO 是相对于 0;
Uart_base	串口寄存器的基地址。

A.11 sensor_device

```
#define MAX_SENSORS 64
#define SENSOR_TEMPER  0x00000001
#define SENSOR_VOLTAGE 0x00000002
#define SENSOR_FAN     0x00000004
struct sensor_device {
        u8 name[16];  /* a formal name */
        u8 label[64]; /* a flexible description */
        u32 type;       /* SENSOR_* */
        u32 id;         /* instance id of a sensor-class */
        u32 fan_policy; /* see arch/mips/include/asm/mach-loongson/loongson_hwmon.h */
        u32 fan_percent;/* only for constant speed policy */
        u64 base_addr;  /* base address of device registers */
}__attribute__((packed));
```

该结构体包括温度、电压传感器和风扇的描述，MAX_SENSORS 是最大传感器数量，本结构体为可选结构体。该结构体的访问通过 system_loongson 结构体寻址得到，而不是通过 smbios 结构体直接访问。

表 A.10 sensor_device 结构体描述

成员	意义
Name[16]	用于创建 platform_device 的名字;
label[64]	自定义的一个描述名;
Type	传感器类型;
Id	区分同等设备的不同实例;
fan_policy	三种策略,定速CONSTANT_SPEED_POLICY、变速STEP_SPEED_POLICY和内核辅助 KERNEL_HELPER（EC 控制）;
fan_percent	定速策略专用，用于风扇转速的百分比;
Base_addr	传感器寄存地址的基地址。

A.12 efi_cpuinfo_loongson

```
typedef enum loongson_cpu_type {
        Loongson_2F,Loongson_2E, Loongson_3A, Loongson_3B,Loongson_1A,Loongson_1B
};
struct efi_cpuinfo_loongson {
/*
 * Capability and feature descriptor structure for MIPS CPU
 */
       u16  vers; //version of efi_cpiinfog_loongson
       u32  processor_id;/* PRID, e.g. 6305, 6306 */
       u32 loongson_cpu_type  cputype;//3a-3b-2f-2e-1a-1b
       u32  total_node;   /* physical core number */
       u16  cpu_startup_core_id; /* Core id*/
       u16  reserved_cores_mask; /*Reserved Core mask*/
       u32  cpu_clock_freq; /*cpu_clock */
       u32 nr_cpu_loongson; /*number of cpus*/
}__attribute__((packed));
```

Processor_id 是从 PRID 寄存器里读取的值,以 16 进制表示， cputype 传递是龙芯哪个型号的，total_node 保存当前 CPU 中包含几个节点，cpu_startup_core_id 保存启动核的 id，cpu_clock_freq 是当前 CPU 的频率，与 CPU 相关的信息在传递过程中要非常小心，如果弄错一点，内核绝对无法启动。

必须定义：cpu_type, cpu_clock_freq, nr_cpu_loongson,total_node,等。

表 A.11 efi_cpuinfo_loongson 结构体描述

成员	意义
Vers	CPU 结构的版本号;
processor_id	标识处理器的型号，具体的批次，如 3a3,3b2 等;
Cputype	CPU 的类型，如 1A,1B,2E,2F,3A,3B 等;
total_node	总结点数;
cpu_startup_core_id	CPU 启动核的 id 号,用其 10 进制值表示;
reserved_cores_mask	可工作的处理器核掩码，核序和字节的位序相同，即 0~15 位分别对应 0~15 号核的状态；0 表该核可用，1 表不可用;
Cpu_clock_freq	CPU 频率;

nr_cpu_loongson	CPU 物理核数;

A.13 irq_source_routing_table

该结构体描述龙芯上中断控制的信息。

```
struct irq_source_routing_table {
        u16 vers ; /*version of irq_source_routing_table*/
        u8 bus;
        u8 devfn;       /* Where the interrupt router lies */
        u16 endor, device,PIC_type; /* Vendor and device ID of interrupt router PIC */
        u64 ht_int_bit; //3a: 1<<24; 3b:1<<16
        u64 ht_enable;//all irqs used in HT PIC which from 8259 or other PICS
        u32 node_id;  //the PIC interrupter attach to the cpu_node
        u64 pci_mem_start_addr;
        u64 pci_mem_end_addr;
        u64 pci_io_start_addr;
        u64 pci_io_end_addr;
        u64 pci_config_addr;
        u32 dma_mask_bits;/*dma 的位数*/
}__attribute__((packed));
```

vers 是当前结构体版本号，bus 等表示中断控制器连接在哪个总线上，vendor,device 表示当前中断控制器的设备信息,PIC_type 告诉内核当前用的是什么类型的中断控制器，ht_int_bit 告诉内核当前的中断控制器是哪一位置 1 便使能 HT 中断控制器。ht_enable 表示板子上所有设备的中断号，node_id 表示中断控制器接在哪一个 CPU 节点上。pci_mem 相关信息传递当前 pci 的 mem,io 的起始，结束信息等，与 efi_memory_map 一起，就把规范了内存地址使用信息传递给内核，以后内核不再再次扫描分配，而直接使用规定好的内存段即可。Irq_info 传递了当前板卡上 PCI、PCIE、min-PCI 槽的中断，接的 bus 号，设备名称等，目前该 slots 暂时不用，目前的处理是把槽号当做 PCI 设备，其中断号等直接写入 PCI 设备的寄存器，内核扫描 PCI 设备即可获得。内核接收到该信息后，只处理固件传递过来的中断信息及设备中断，而不再做判断和扫描。

必须定义：

pci_mem_start_addr,pci_mem_end_addr,pci_io_start_addr,pci_io_end_addr

采用 HT 总线的必须定义:ht_int_bit,ht_enable,

表 A.12 irq_source_routing_table 结构体描述

成员	意义
Vers	本结构体版本号;
sys_int_bit	中断控制器哪一位使能;
bus,devfn	中断路由所在的 bus 号;
vendor, device,PIC_type	中断控制器的厂商号，设备号，及其本身的类型;
Ht_enable	Ht 中断控制器使能的中断位，i8259 也可用其标识;
Node_id	中断控制器接到哪一个 CPU 核上;
pci_mem_start_addr	Pci_mem 空间的起始地址;
pci_mem_end_addr	Pci_mem 空间的结束地址;
pci_io_start_addr	Pci_io 空间的起始地址;
pci_io_end_addr	Pci_io 空间的结束地址;

pci_config_start_addr	Pci_config 空间的起始地址；
dma_mask_bits	DMA 的位数：32 位或 64 位

A.14 loongson_special_attribute

```
struct loongson_special_attribute{
        u16 vers; /*version of this special*/
        u8 special_name[32]; /*special_atribute_name*/
        u32 loongson_special_type; /*tyoe of special device*/
        struct resource resource;/*special_vlaue resource*/
}__attribute__((packed));
```

该结构体为龙芯的特殊应用留下接口，定义该特殊应用的版本，名称，类型，地址及 data 的其他属性。本结构暂不实现。

表 A.13 loongson_special_attribute 结构体描述

成员	意义
Vers	本结构版本号；
special_name	特殊属性的名称；
loongson_special_type	特殊属性的类型；
Resource	特殊属性是否又含有自身的资源、属性之类；

本结构作为扩展可存放在龙芯平台上的一些特殊的属性，如之前 vga_bios 的基地址等。

A.15 小结

固件的接口结构体，能满足当前 PMON 基础上固件和内核的交互应用，预留的 UEFI 扩展，涵盖 UEFI 基本服务信息，详细、精确的扩展需要在使用 UEFI 的时候具体调试，定义。

固件接口中各模块的具体包含关系如下：

```
Struct boot_params{}->struct efi{}->struct smbios{}->
|->struct efi_memory_map;
|->struct efi_cpuinfo_longson;
|->struct irq_source_routing_table;
|->struct system_loongson;
|->struct uart_device;
|->struct sensor_device;
|->struct interface_info;
|->struct board_device;
```

所有的结构体均定义在 bootparam.h 中，所需要保证的是固件和内核中的 bootparam.h 完全一致，否则可能导致内核解析不正确，其中 uart_device 和 sensor_device 的访问是通过结构体 system_loongson，而不是通过结构 smbios 的偏移访问。

附录 B　传参数据结构头文件 bootparam.h

B.1 说明

此头文件在内核和固件中须保持一致，其结构体的描述参见附录 A。

B.2 bootparam.h

```
#ifndef __ASM_MACH_LOONGSON_BOOT_PARAM_H_
#define __ASM_MACH_LOONGSON_BOOT_PARAM_H_

#define SYSTEM_RAM_LOW          1
#define SYSTEM_RAM_HIGH    2
#define MEM_RESERVED         3
#define PCI_IO                  4
#define PCI_MEM        5
#define LOONGSON_CFG_REG 6
#define VIDEO_ROM          7
#define ADAPTER_ROM           8
#define ACPI_TABLE         9
#define SMBIOS_TABLE           10
#define MAX_MEMORY_TYPE  11

#define LOONGSON3_BOOT_MEM_MAP_MAX 128
struct efi_memory_map_loongson {
        u16 vers; /* version of efi_memory_map */
        u32 nr_map;    /* number of memory_maps */
        u32 mem_freq;       /* memory frequence */
        struct mem_map {
                u32 node_id;   /* node_id which memory attached to */
                u32 mem_type;       /* system memory, pci memory, pci io, etc. */
                u64 mem_start;      /* memory map start address */
                u32 mem_size;       /* each memory_map size, not the total size */
        } map[LOONGSON3_BOOT_MEM_MAP_MAX];
} __packed;

enum loongson_cpu_type {
        Loongson_2E = 0,
        Loongson_2F = 1,
        Loongson_3A = 2,
        Loongson_3B = 3,
        Loongson_1A = 4,
        Loongson_1B = 5
};

```

```
38  /*
39   * Capability and feature descriptor structure for MIPS CPU
40   */
41  struct efi_cpuinfo_loongson {
42          u16 vers;         /* version of efi_cpuinfo_loongson */
43          u32 processor_id; /* PRID, e.g. 6305, 6306 */
44          u32 cputype;    /* Loongson_3A/3B, etc. */
45          u32 total_node;      /* num of total numa nodes */
46          u32 cpu_startup_core_id; /* Core id */
47          u16  reserved_cores_mask; /*Reserved CPU Core mask*/
48          u32 cpu_clock_freq; /* cpu_clock */
49          u32 nr_cpus;
50  } __packed;

51  #define MAX_UARTS 64
52  struct uart_device {
53          u32 iotype; /* see include/linux/serial_core.h */
54          u32 uartclk;
55          u32 int_offset;
56          u64 uart_base;
57  } __packed;
58
59  #define MAX_SENSORS 64
60  #define SENSOR_TEMPER    0x00000001
61  #define SENSOR_VOLTAGE 0x00000002
62  #define SENSOR_FAN          0x00000004
63  struct sensor_device {
64          char name[32];    /* a formal name */
65          char label[64]; /* a flexible description */
66          u32 type;              /* SENSOR_* */
67          u32 id;                /* instance id of a sensor-class */
68          u32 fan_policy; /* see arch/mips/include/asm/mach-loongson/loongson_hwmon.h */
69  u32 fan_percent;/* only for constant speed policy */
70          u64 base_addr;    /* base address of device registers */
71  } __packed;
72
73  struct system_loongson {
74          u16 vers;         /* version of system_loongson */
75          u32 ccnuma_smp; /* 0: no numa; 1: has numa */
76          u32 sing_double_channel; /* 1:single; 2:double */
77          u32 nr_uarts;
78          struct uart_device uarts[MAX_UARTS];
79          u32 nr_sensors;
80          struct sensor_device sensors[MAX_SENSORS];
```

```
81          char has_ec;
82          char ec_name[32];
83          u64 ec_base_addr;
84          char has_tcm;
85          char tcm_name[32];
86          u64 tcm_base_addr;
87          u64 workarounds; /* see workarounds.h */
88  } __packed;
89
90  struct irq_source_routing_table {
91          u16 vers;
92          u16 size;
93          u16 rtr_bus;
94          u16 rtr_devfn;
95          u32 vendor;
96          u32 device;
97          u32 PIC_type;      /* conform use HT or PCI to route to CPU-PIC */
98          u64 ht_int_bit; /* 3A: 1<<24; 3B: 1<<16 */
99          u64 ht_enable;    /* irqs used in this PIC */
100         u32 node_id;       /* node id: 0x0-0; 0x1-1; 0x10-2; 0x11-3 */
101         u64 pci_mem_start_addr;
102         u64 pci_mem_end_addr;
103         u64 pci_io_start_addr;
104         u64 pci_io_end_addr;
105         u64 pci_config_addr;
106         u32 dma_mask_bits;
107 } __packed;
108
109 struct interface_info {
110  u16 vers; /* version of the specificition */
111  u16 size;
112  u8    flag;
113  char description[64];
114 } __packed;
115
116 #define MAX_RESOURCE_NUMBER 128
117 struct resource_loongson {
118  u64 start; /* resource start address */
119  u64 end;      /* resource end address */
120         char name[64];
121         u32 flags;
122 };
123
124 struct archdev_data {};    /* arch specific additions */
```

```
125
126 struct board_devices {
127         char name[64];        /* hold the device name */
128         u32 num_resources; /* number of device_resource */
129         /* for each device's resource */
130         struct resource_loongson resource[MAX_RESOURCE_NUMBER];
131         /* arch specific additions */
132         struct archdev_data archdata;
133 };
134
135 struct loongson_special_attribute {
136         u16 vers;         /* version of this special */
137         char special_name[64]; /* special_atribute_name */
138         u32 loongson_special_type; /* type of special device */
139         /* for each device's resource */
140         struct resource_loongson resource[MAX_RESOURCE_NUMBER];
141 };
142
143 struct loongson_params {
144         u64 memory_offset;       /* efi_memory_map_loongson struct offset */
145         u64 cpu_offset;          /* efi_cpuinfo_loongson struct offset */
146         u64 system_offset;  /* system_loongson struct offset */
147         u64 irq_offset;       /* irq_source_routing_table struct offset */
148         u64 interface_offset;       /* interface_info struct offset */
149         u64 special_offset;  /* loongson_special_attribute struct offset */
150         u64 boarddev_table_offset;    /* board_devices offset */
151 };
152
153 struct smbios_tables {
154         u16 vers;         /* version of smbios */
155         u64 vga_bios; /* vga_bios address */
156         struct loongson_params lp;
157 };
158
159 struct efi_reset_system_t {
160         u64 ResetCold;
161         u64 ResetWarm;
162         u64 ResetType;
163         u64 Shutdown;
164         u64 DoSuspend; /* NULL if not support */
165 };
166
167 struct efi_loongson {
168         u64 mps; /* MPS table */
```

```
169         u64 acpi; /* ACPI table (IA64 ext 0.71) */
170         u64 acpi20;     /* ACPI table (ACPI 2.0) */
171         struct smbios_tables smbios;  /* SM BIOS table */
172         u64 sal_systab;       /* SAL system table */
173         u64 boot_info;/* boot info table */
174 };
175
176 struct boot_params {
177         struct efi_loongson efi;
178         struct efi_reset_system_t reset_system;
179 };
180
181 struct loongson_system_configuration {
182         u32 nr_cpus;
183         u32 nr_nodes;
184         int cores_per_node;
185         int cores_per_package;
186         enum loongson_cpu_type cputype;
187         u64 ht_control_base;
188         u64 pci_mem_start_addr;
189         u64 pci_mem_end_addr;
190         u64 pci_io_base;
191         u64 restart_addr;
192         u64 poweroff_addr;
193         u64 suspend_addr;
194         u64 vgabios_addr;
195         u32 dma_mask_bits;
196 };
197
198 extern struct efi_memory_map_loongson *loongson_memmap;
199 extern struct loongson_system_configuration loongson_sysconf;
200
201 extern u32 loongson_nr_uarts;
202 extern struct uart_device loongson_uarts[MAX_UARTS];
203 extern char loongson_ecname[32];
204 extern u32 loongson_nr_sensors;
205 extern struct sensor_device loongson_sensors[MAX_SENSORS];
206 #endif
```

附录 C　3A/B+780E 中断及地址空间约定

C.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。

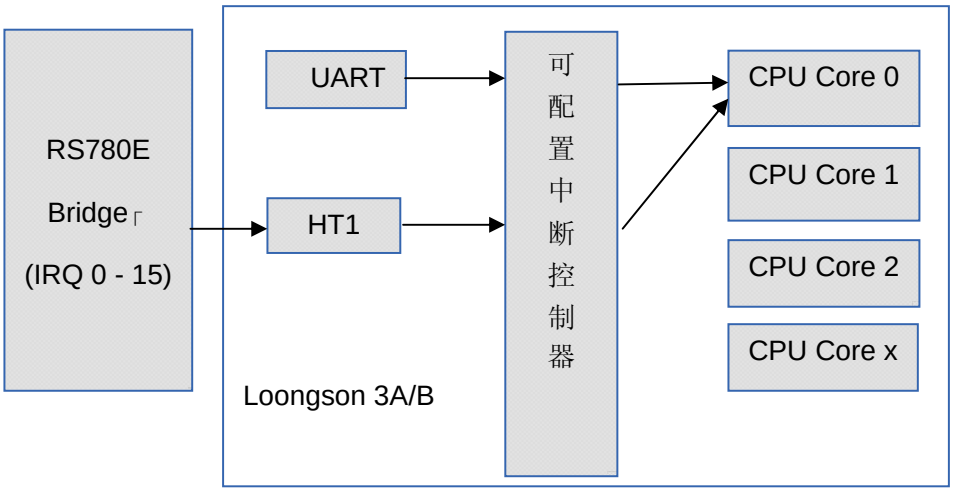

图 C.1 3A/B+780E 中断路由示意图

C.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 C.1 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。

C.2 3A/B+780E 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 C.2 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	

C.3 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 C.3 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
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

附录 D　3A/B+2H 中断及地址空间约定

D．1 3A/B+2H 中断系统描述

2H 南桥有 5 组中断控制寄存器，每组中断控制寄存器控制 32 个中断源，总共可以控制 160 个中断源。与 3A/B+780E 使用 HT 中断不同，3A/B+2H 使用 3A/B 的 INTn0 中断。3A/B 芯片的 INTN0 管脚与 2H 芯片的 SYS_INTN 管脚相连，将 2H 上的 160 个中断源连接到 3A/B 的 INTn0 中断。因此通过配置 3A/B 的中断路由寄存器可以把 INTn0 中断路由到 Core0 的 IP3。当 2H 南桥的中断触发时，3A/B Core0 的 Cause 寄存器 IP3 位就会被置 1，这时查询 2H 的中断状态寄存器就可以判断出中断来源，，其示意图如下。

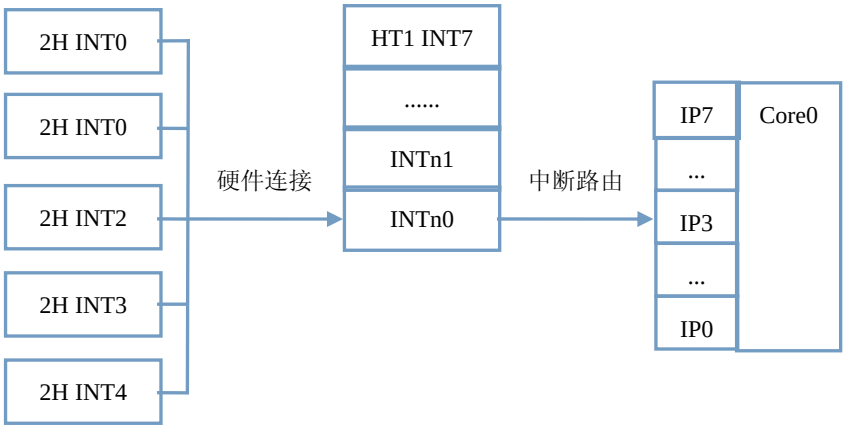

图 D.1 3A/B+2H 中断路由示意图

D.2 3A/B+2H 的中断号分配约定

0-15 号保留表 D.1 中的中断。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 D.1 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。

64-224 号： 分配给 2H 桥片。

D.3 3A/B+2H 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 D.2 3A/B+2H 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	直接连接中断设备时进行分配。若连接南桥则仅用于级联，不分配中断设备。
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	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	

D.4 3A/B+2H 芯片地址空间分布表

表 D.3 3A/B+2H 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 0 ～ 256M
地址 1	0x0000_0000_1000_0000	0x0000_0000_17FF_FFFF	HT1 MEM, LS2H PCIE MEM
地址 2	0x0000_0000_1800_0000	0x0000_0000_18FF_FFFF	HT1 MEM, LS2H PCIE IO
地址 3	0x0000_0000_1900_0000	0x0000_0000_1AFF_FFFF	保留
地址 4	0x0000_0000_1B00_0000	0x0000_0000_1BFF_FFFF	HT1 MEM, LS2H REGISTER
地址 5	0x0000_0000_1C00_0000	0x0000_0000_1DFF_FFFF	LPC MEM
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_4000_0000	0x0000_0000_7FFF_FFFF	HT1 MEM 空间
地址 17	0x0000_0C00_0000_0000	0x0000_0FFF_FFFF_FFFF	HT1 控制器，各种空间
地址 18	0x0000_1000_0000_0000	0x0000_3FFF_FFFF_FFFF	猜测空间
地址 19	0x0000_0000_8000_0000	0x0000_0000_FFFF_FFFF	保留
地址 20	0x0000_0001_0000_0000	0x0000_0001_0FFF_FFFF	保留
地址 21	0x0000_0001_1000_0000	0x0000_0001_FFFF_FFFF	内存 256M ～ 4G