龙芯 3A 处理器用户手册

中国科学院计算技术研究所

北京龙芯中科技术服务中心有限公司

中文版本： 版本 1.01，二零一零年十月

目录

插图

表格

第一章　概述

1.1　龙芯系列处理器介绍

龙芯处理器主要包括三个系列。龙芯 1 号处理器及其 IP 系列主要面向嵌入式应用，龙芯 2 号超标量处理器及其 IP 系列主要面向桌面应用，龙芯 3 号多核处理器系列主要面向服务器和高性能机应用。根据应用的需要，其中部分龙芯 2 号也可以面向部分高端嵌入式应用，部分低端龙芯 3 号也可以面向部分桌面应用。上述三个系列将并行地发展。龙芯 3 号多核系列处理器基于可伸缩的多核互连架构设计，在单个芯片上集成多个高性能处理器核以及大量的 2 级 Cache，还通过高速 I/O 接口实现多芯片的互连以组成更大规模的系统。龙芯 3 号采用的可伸缩互连结构如下图 1-1 所示。龙芯 3 号片内及多片系统均采用二维 mesh 互连结构，其中每个结点由 $8*8$ 的交叉开关组成，每个交叉开关连接四个处理器核以及四个二级 Cache，并与东（E）南（N）西（W）北（N）四个方向的其他结点互连。因此，$2*2$ 的 mesh 可以连接 16 个处理器核，$4*4$ 的 mesh 可以连接 64 个处理器核。

龙芯 3 号结点及二维互连结构，(a) 结点结构，(b) 2*2 的 mesh 网络连接 16 个处理器，(c) $4*4$ 的 mesh 网络连接 64 个处理器。图 1-1 龙芯 3 号系统结构 (FIXME)

龙芯 3 号的结点结构如下图 1-2 所示。每个结点有两级 AXI 交叉开关连接处理器、二级 Cache、内存控制器以及 IO 控制器。其中第一级 AXI 交叉开关（X1 Switch，简称 X1）连接处理器和二级 Cache。第二级交叉开关（X2 Switch，简称 X2）连接二级 Cache 和内存控制器。

图: 龙芯 3 号节点结构 (FIXME)

在每个结点中，最多 $8*8$ 的 X1 交叉开关通过四个主端口连接四个 GS464 处理器核（图中 P0、P1、P2、P3），通过四个从端口连接统一编址的四个 interleave 二级 Cache 块（图中 S0、S1、S2、S3），通过四对主/从端口连接东、南、西、北四个方向的其他结点或 IO 结点（图中 EM/ES、SM/SS、WM/WS、NM/NS）。X2 交叉开关通过四个主端口连接四个二级 Cache，至少一个从端口连接一个内存控制器，至少一个从端口连接一个交叉开关的配置模块（Xconf）用于配置本结点的 X1 和 X2 的地址窗口等。还可以根据需要连接更多的内存控制器和 IO 端口等。在龙芯 3 号的互连系统只是定义上层协议，会不对传输协议的实现做具体规定，因此，结点之间的互连即可以采用片上网络进行实现，也可以通过 I/O 控制链路实现多芯片的互连。在一个 4 结点 16 核系统为例，既可以通过 4 片 4 核芯片组成，也可以通过 2 片 8 核芯片，或基于一个单芯片 4 节点 16 核芯片组成。由于互连系统的物理实现对软件透明，上述 3 种配置的系统可以在相同的操作系统上运行。

1.2 龙芯 3A 简介

龙芯 3A 是龙芯 3 号多核处理器系列的第一款产品，是一款配置为单节点 4 核的处理器，采用 65nm 工艺制造，最高工作主频为 1GHz，主要技术特征如下：

- 片内集成 4 个 64 位的四发射超标量 GS464 高性能处理器核；
- 片内集成 4 MB 的分体共享二级 Cache(由 4 个体模块组成，每个容量为 1MB) ；
- 通过目录协议维护多核及 I/O DMA 访问的 Cache 一致性；
- 片内集成 2 个 64 位 400MHz 的 DDR2/3 控制器；
- 片内集成 2 个 16 位 800MHz 的 HyperTransport 控制器；
- 每个 16 位的 HT 端口可以拆分成两个 8 路的 HT 端口使用；
- 片内集成 1 个 32 位的 100MHz PCIX/66MHz PCI 接口；
- 片内集成 1 个 LPC、2 个 UART、1 个 SPI、16 路 GPIO 接口。

龙芯 3A 号芯片整体架构基于两级互连实现，结构如下图 1-3 所示。

图 1-3 龙芯 3A 芯片结构 (FIXME)

第一层互连采用 6×6 的交叉开关，用于连接四个 CPU（作为主设备）、四个二级 Cache 模块（作为从设备）、以及两个 IO 端口（每个使用一个主端口和一个从端口）。一级互连开关连接的每个 IO 端口连接一个 16 位的 HT 控制器，每个 16 位的 HT 端口还可以作为两个 8 位的 HT 端口使用。HT 控制器通过一个 DMA 控制器和一级互联开关相连，DMA 控制器负责 IO 的 DMA 控制并负责片间一致性的维护。龙芯 3 号的 DMA 控制器还可以通过配置实现预取和矩阵转置。第二级互连采用 5x4 的交叉开关，连接 4 个 2 级 Cache 模块（作为主设备），两个 DDR2 内存控制器、低速高速 I/O（包括 PCI、LPC、SPI 等）以及芯片内部的控制寄存器模块。上述两级互连开关都采用读写分离的数据通道，数据通道宽度为 128bit，工作在与处理器核相同的频率，用以提供高速的片上数据传输。基于龙芯 3 号可扩展互联架构，4 片四核龙芯 3A 可以通过 HT 端口连接构成 4 芯片 16 核的对称多处理器系统 (SMP) 结构。

第二章　GS464 处理器核

GS464 是四发射 64 位的高性能处理器核。该处理器核既可以作为单核面向高端嵌入式应用和桌面应用，也可以作为基本的处理器核构成片内多核系统面向服务器和高性能机应用。在龙芯 3A 中的多个 GS464 核通过以及二级 Cache 模块通过 AXI 互联网络形成一个分布式共享二级 Cache 的多核结构。GS464 的主要特点如下：

- MIPS64 兼容，增加 SIMD 型多媒体指令以及 X86 虚拟机指令；
- 四发射超标量结构，两个定点、两个浮点、一个访存部件；
- 每个浮点部件都支持全流水 64 位/双 32 位浮点乘加运算；
- 访存部件支持 128 位存储访问，虚地址和物理地址各为 48 位；
- 支持寄存器重命名、动态调度、转移预测等乱序执行技术；
- 64 项全相联 TLB，独立的 16 项指令 TLB，可变页大小；
- 一级指令 Cache 和数据 Cache 大小各为 64KB，4 路组相联；
- 支持 Non-blocking 访问及 Load-Speculation 等访存优化技术；
- 支持 Cache 一致性协议，可用于片内多核处理器；
- 指令 Cache 实现奇偶校验，数据 Cache 实现 ECC 校验；
- 支持标准的 EJTAG 调试标准，方便软硬件调试；
- 标准的 128 位 AXI 接口。

GS464 的结构如下图所示。GS464 更多的详细介绍请参考 GS464 用户手册以及 MIPS64 用户手册。

第三章　系统配置与控制

3.1　芯片工作模式

根据组成系统的结构，龙芯 3A 有两种工作模式：（1）单芯片模式：系统只集成 1 片龙芯 3A，构成一个对称多处理器系统（SMP）；（2）多芯片互连模式：系统中包含 2 片或 4 片龙芯 3A，通过龙芯 3A 的 HT 端口进行互连，构成一个非均匀访存多处理器系统（CC-NUMA）。

3.2　控制引脚说明

龙芯 3A 的控制引脚包括 DO_TEST 、ICCC_EN 、NODE_ID[1:0] 、CLKSEL[15:0]、PCI_CONFIG。见表 3.1。

3.3　Cache 一致性

龙芯 3A 由硬件维护处理器、以及通过 HT 端口接入的 I/O 之间的 Cache 一致性，但硬件不维护通过 PCI 等慢速总线接入到系统中的 I/O 设备的 Cache 一致性。在驱动程序开发时，对通过 PCI 接入的设备进行 DMA （Direct Memory Access）传输时，需要软件进行 Cache 一致性维护。Remark: 是不是 HT1 也需要 Cache 一致性维护?

3.4　系统节点级的物理地址空间分布

龙芯 3 号系列处理器的系统物理地址分布采用全局可访问的层次化寻址设计，以保证系统开发的扩展兼容。整个系统的物理地址宽度为 48 位。按照地址的高 4 位，整个地址空间被均匀分布到 16 个结点上，即每个结点分配 44 位地址空间。龙芯 3A 采用单节点 4 核配置，因此龙芯 3A 芯片集成的 DDR 内存控制器、HT 总线、PCI 总线的对应地址都包含在从 0x0 至 0xFFFF_FFFF_FFFF 的 44 位地址空间内，具体各设备地址分布请参见后续相关章节。

在每个节点的内部，位地址空间又进一步均匀分布给结点内连接的可能最 44 多 8 个设备。其中低 43 位地址由 4 个 2 级 Cache 模块所拥有，高 43 位地址则进一步按地址的 [43:42] 位分布给连接在 4 个方向端口的设备上：如表 **??**。根据芯片和系统结构配置的不同，如果某端口上没有连接从设备，则对应的地址空间是保留地址空间，不允许访问。

例如节点 0 的东端口设备的基地址为 0x0800_0000_0000, 节点 1 的东端口设备的基地址为 0x1800_0000_0000，依次类推。不同于方向端口的映射关系，龙芯 3A 可以根据实际应用的访问行为，来决定二级 Cache 的交叉寻址方式。节点内的 4 个 2 级 Cache 模块一共对应 43 位地址空间，

<table>
<tr><th>信号</th><th>上下拉</th><th>描述</th></tr>
<tr><td>DO_TEST</td><td>上拉</td><td>1: 功能模式; 0: 测试模式</td></tr>
<tr><td>ICCC_EN</td><td>下拉</td><td>1: 多芯片一致性互联模式（使用 HT0 互联）;
0: 单芯片模式</td></tr>
<tr><td>NODE_ID[1:0]</td><td></td><td>在多芯片一致性互连模式下表示处理器号</td></tr>
<tr><td>CLKSEL[15:0]</td><td></td><td>上电时钟控制
<table>
<tr><th>模块</th><th>信号</th><th>作用</th></tr>
<tr><td rowspan="4">HT</td><td>15</td><td>1: 采用内部参考电压
0: 采用外部参考电压</td></tr>
<tr><td>14</td><td>1: HT PLL 采用差分时钟输入
0: HT PLL 采用普通时钟输入</td></tr>
<tr><td>13:12</td><td>00: PHY 时钟为 1.6GHZ/1
01: PHY 时钟为 3.2GHZ/2
10: PHY 时钟为普通输入时钟
11: PHY 时钟为差分输入时钟</td></tr>
<tr><td>11:10</td><td>00: HT 控制器时钟 200MHz
01: HT 控制器时钟 400MHz
1x: HT 控制器时钟为普通输入时钟</td></tr>
<tr><td>MEM</td><td>9:5</td><td>11111: MEM 时钟直接采用 memclk
其它:
memclk*(clksel[8:5]+30)/(clksel[9]+3)</td></tr>
<tr><td>CORE</td><td>4:0</td><td>11111: CORE 时钟直接采用 sysclk
其它:
sysclk*(clksel[3:0]+30)/(clksel[4]+1)</td></tr>
</table></td></tr>
<tr><td>PCI_CONFIG[7:0]</td><td></td><td>IO 配置控制
<table>
<tr><th>信号</th><th>说明</th></tr>
<tr><td>PCI_CONFIG[7]</td><td>HT 控制信号引脚电压控制位 1</td></tr>
<tr><td>PCI_CONFIG[6:5]</td><td>PCIX 总线速度选择</td></tr>
<tr><td>PCI_CONFIG[4]</td><td>PCIX 总线模式选择</td></tr>
<tr><td>PCI_CONFIG[3]</td><td>PCI 主设备模式</td></tr>
<tr><td>PCI_CONFIG[2]</td><td>系统从 PCI 空间启动</td></tr>
<tr><td>PCI_CONFIG[1]</td><td>使用外部 PCI 仲裁</td></tr>
<tr><td>PCI_CONFIG[0]</td><td>HT 控制信号引脚电压控制位 0</td></tr>
</table>
具体设置
<table>
<tr><th>6</th><th>5</th><th>4</th><th>PCI 模式速度</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>PCI 33/66</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>PCI-X 66</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>PCI-X 100</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>PCI-X 133</td></tr>
</table>
<table>
<tr><th>7</th><th>0</th><th>HT 信号引脚电压 [1]</th></tr>
<tr><td>0</td><td>0</td><td>3.3v</td></tr>
<tr><td>0</td><td>1</td><td>2.5v</td></tr>
<tr><td>1</td><td>0</td><td>1.8v</td></tr>
<tr><td>1</td><td>1</td><td>保留</td></tr>
</table></td></tr>
</table>

表 3.1: 控制引脚说明

节点号	地址位 [47:44]	起始地址	结束地址
0	0x0	0x0000_0000_0000	0x0FFF_FFFF_FFFF
1	0x1	0x1000_0000_0000	0x1FFF_FFFF_FFFF
2	0x2	0x2000_0000_0000	0x2FFF_FFFF_FFFF
3	0x3	0x3000_0000_0000	0x3FFF_FFFF_FFFF
4	0x4	0x4000_0000_0000	0x4FFF_FFFF_FFFF
5	0x5	0x5000_0000_0000	0x5FFF_FFFF_FFFF
6	0x6	0x6000_0000_0000	0x6FFF_FFFF_FFFF
7	0x7	0x7000_0000_0000	0x7FFF_FFFF_FFFF
8	0x8	0x8000_0000_0000	0x8FFF_FFFF_FFFF
9	0x9	0x9000_0000_0000	0x9FFF_FFFF_FFFF
10	0xA	0xA000_0000_0000	0xAFFF_FFFF_FFFF
11	0xB	0xB000_0000_0000	0xBFFF_FFFF_FFFF
12	0xC	0xC000_0000_0000	0xCFFF_FFFF_FFFF
13	0xD	0xD000_0000_0000	0xDFFF_FFFF_FFFF
14	0xE	0xE000_0000_0000	0xEFFF_FFFF_FFFF
15	0xF	0xF000_0000_0000	0xFFFF_FFFF_FFFF

表 3.2: 节点级的系统全局地址分布

设备	地址 [43:41] 位	节点内起始地址	节点结束地址
2 级 Cache	0,1,2,3	0x000_0000_0000	0x7FF_FFFF_FFFF
东	4	0x800_0000_0000	0x9FF_FFFF_FFFF
南	5	0xA00_0000_0000	0xBFF_FFFF_FFFF
西	6	0xC00_0000_0000	0xDFF_FFFF_FFFF
北	7	0xE00_0000_0000	0xFFF_FFFF_FFFF

表 3.3: 节点内的地址分布

而每个 2 级模块所对应的地址空间根据地址位的某两位选择位确定，并可以通过软件进行动态配置修改。系统中设置了名为 SCID_SEL 的配置寄存器来确定地址选择位，如下表所示。在缺省情况下采用 [6:5] 地位散列的方式进行分布，即地址 [6:5] 两位决定对应的 2 级 Cache 编号。该寄存器地址 0x3FF00400。

3.5 地址路由分布与配置

Remark: 这段需要改写一下。 龙芯 3A 的路由主要通过系统的两级交叉开关实现。一级交叉开关可以对每个主端口接收到的请求进行路由配置，每个主端口都拥有 8 个地址窗口，可以完成 8 个地址窗口的目标路由选择。每个地址窗口由 BASE、MASK 和 MMAP 三个 64 位寄存器组成，BASE 以 K 字节对齐；MASK 采用类似网络掩码高位为 1 的格式；MMAP 的低三位表示对应目标从端口的编号，MMAP[4] 表示允许取指，MMAP [5] 表示允许块读，MMAP [7] 表示窗口使能。(Remark: merge with X2 MMAP.. same.)

窗口命中公式:

SCID_SEL	地址位选择	SCID_SEL	地址位选择
4'h0	6: 5	4'h8	23:22
4'h1	9: 8	4'h9	25:24
4'h2	11:10	4'ha	27:26
4'h3	13:12	4'hb	29:28
4'h4	15:14	4'hc	31:30
4'h5	17:16	4'hd	33:32
4'h6	19:18	4'he	35:34
4'h7	21:20	4'hf	37:36

表 3.4: 节点内地址地址选择位设置

```
IN_ADDR & MASK == BASE
```

由于龙芯 3 号缺省采用固定路由，在上电启动时，配置窗口都处于关闭状态，使用时需要系统软件对其进行使能配置。表 3.5 列出了一级交叉开关的 8 个主端口的窗口寄存器基地址。 每个端口

基地址	Core0	Core1	Core2	Core3
	0x3FF0_2000	0x3FF0_2100	0x3FF0_2200	0x3FF0_2300
基地址	East	South	West	North
	0x3FF0_2400	0x3FF0_2500	0x3FF0_2600	0x3FF0_2700

表 3.5: 一级交叉开关主端口窗口寄存器基地址

对应的地址窗口转换寄存器如下表所示。

在龙芯 3 号的二级 XBAR 中有 CPU 地址空间 (包括 HT 空间)、DDR2 地址空间、以及 PCI 地址空间共三个 IP 相关的地址空间。地址窗口是供 CPU 和 PCI-DMA 两个具有主功能的 IP 进行路由选择和地址转换而设置的。CPU 和 PCI-DMA 两者都拥有 8 个地址窗口，可以完成目标地址空间的选择以及从源地址空间到目标地址空间的转换。每个地址窗口由 BASE、MASK 和 MMAP 三个 64 位寄存器组成，BASE 以 K 字节对齐，MASK 采用类似网络掩码高位为 1 的格式，MMAP 的低三位是路由选择。在 2 级 XBAR 处，标号与所述模块的对应关系如下表示对应新地址空间的编号。

偏移	寄存器	偏移	寄存器	偏移	寄存器
0x00	WIN0_BASE	0x40	WIN0_MASK	0x80	WIN0_MMAP
0x08	WIN1_BASE	0x48	WIN1_MASK	0x88	WIN1_MMAP
0x10	WIN2_BASE	0x50	WIN2_MASK	0x90	WIN2_MMAP
0x18	WIN3_BASE	0x58	WIN3_MASK	0x98	WIN3_MMAP
0x20	WIN4_BASE	0x60	WIN4_MASK	0xA0	WIN4_MMAP
0x28	WIN5_BASE	0x68	WIN5_MASK	0xA8	WIN5_MMAP
0x30	WIN6_BASE	0x70	WIN6_MASK	0xB0	WIN6_MMAP
0x38	WIN7_BASE	0x78	WIN7_MASK	0xB8	WIN7_MMAP

表 3.6: 一级交叉开关地址窗口寄存器地址表

标号	缺省值
0	0 号 DDR2/3 控制器
1	1 号 DDR2/3 控制器
2	低速 I/O（PCI，LPC 等）
3	配置寄存器

表 3.7: 二级 XBAR 地址路由表

[7]	[5]	[4]
窗口使能	允许块读	允许取指

表 3.8: MMAP 字段对应的该空间访问属性

二级 XBAR 的地址配置与一级 XBAR 的地址配置相比增加了地址转换的功能 (Remark: this statement doesn't seem right...)。相比之下，一级 XBAR 的窗口配置不能对 Cache 一致性的请求进行地址转换，否则在二级 Cache 处的地址会与处理器一级 Cache 处的地址不一致，导致 Cache 一致性的维护错误。

窗口命中公式: (IN_ADDR & MASK) == BASE OUT_ADDR = (IN_ADDR & MASK) | 新地址换算公式: MMAP[63:10],10 h0

地址窗口转换寄存器如下表。

表 2-8 二级 XBAR 地址窗口转换寄存器表 (FIXME)

根据缺省的寄存器配置，芯片启动后，CPU 的 0x00000000 - 0x0fffffff 的地址区间（256M）映射到 DDR2 的 0x00000000 - 0x0fffffff 的地址区间，CPU 的 0x10000000 - 0x1fffffff 区间（256M）映射到 PCI 的 0x10000000 - 0x1fffffff 区间，PCIDMA 的 0x80000000 - 0x8fffffff 的地址区间（256M）映射到 DDR2 的 0x00000000 - 0x0fffffff 的地址区间。软件可以通过修改相应的配置寄存器实现新的地址空间路由和转换。此外，当出现由于 CPU 猜测执行引起对非法地址的读访问时，8 个地址窗口都不命中，由配置寄存器模块返回全 0 的数据给 CPU，以防止 CPU 死等。

3.6 芯片配置及采样寄存器

龙芯 3 号中的芯片配置寄存器 (Chip_config) 及芯片采样寄存器 (chip_sample) 提供了对芯片的配置进行读写的机制。Remark: What's the width of these registers? BTW, in both tables, some fields are missing, and more explanation is needed.

基地址	高位	所有者
0x0000_0000_0000_0000	0x0000_0000_1000_0000	0 号 DDR 控制器
0x1000_0000_0000_0000	0x0000_0000_2000_0000	低速 I/O（PCI 等）

表 3.9: 二级 XBAR 缺省地址配置

位域	字段名	访问	复位值	描述
其它	Reserved	R		保留
34:32	HT_freq_scale_ctrl0	RW	3'b111	HT 控制器 1 分频
31:29	HT_freq_scale_ctrl0	RW	3'b111	HT 控制器 0 分频
28:24	DDR_Clksel	RW	5'b11111	软件配置 DDR 时钟倍频关系（当 DDR_Clksel_en 为 1 时有效）
19	DDR_reset1	RW	1'b1	软件 reset DDR 控制器 1
18	DDR_reset0	RW	1'b1	软件 reset DDR 控制器 0
17	Mc1_en	RW	1'b1	是否启用 DDR 控制器 1
16	Mc0_en	RW	1'b1	是否启用 DDR 控制器 0
15	Core3_en	RW	1'b1	是否启用处理器核 3
14	Core2_en	RW	1'b1	是否启用处理器核 2
13	Core1_en	RW	1'b1	是否启用处理器核 1
12	Core0_en	RW	1'b1	是否启用处理器核 0
9	DDR_buffer_cpu	RW	1'b0	是否打开 DDR 读访问缓冲
8	Disable_ddr2_confspace	RW	1'b0	是否禁用 DDR 配置空间
3	DDR_Clksel_en	RW	1'b0	是否使用软件配置 DDR 倍频
2:0	Freq_scale_ctrl	RW	3'b111	处理器核分频

表 3.10: 芯片配置寄存器（物理地址 0x1fe00180）

位域	字段名	访问	复位值	描述
其它		R		保留
111	Thsens1_overflow	R		温度传感器 1 温度上溢（超过 128 度）
110:104	Thsens1_out	R		温度传感器 1 温度
103	Thsens0_overflow	R		温度传感器 0 温度上溢（超过 128 度）
102:96	Thsens0_out	R		温度传感器 0 温度
57:56	Bad_ip_ht	R		2 个 HT 控制器是否坏
53:52	Bad_ip_ddr	R		2 个 DDR 控制器是否坏
51:48	Bad_ip_core	R		4 个处理器核是否坏
47:32	Sys_clksel	R		板上倍频设置
31:16	Pad3v3_ctrl	RW	16'h780	3v3pad 控制
15:0	Pad2v5_ctrl	RW	16'h780	2v5pad 控制

表 3.11: 芯片采样寄存器（物理地址 0x1fe00190）

第四章　二级 Cache

二级 Cache 模块是与 GS464 处理器 IP 配套设计的模块。该模块既可以和 GS464 对接，使 GS464 成为包括二级 Cache 在内的处理器 IP；也可以通过 AXI 网络连接多个 GS464 以及多个二级 Cache 模块，形成片内多处理器的 CMP 结构。二级 Cache 模块的主要特征包括：

- 采用 128 位 AXI 接口；
- 8 项 Cache 访问队列；
- 关键字优先；
- 接收读失效请求到返回数据最快 8 拍；(Remark: 最慢多少拍？)
- 通过目录支持 Cache 一致性协议；
- 可用于片上多核结构，也可直接和单处理器 IP 对接；
- 软 IP 级可配置二级 Cache 的大小（512KB/1MB）；
- 采用四路组相联结构；
- 运行时可动态关闭；
- 支持 ECC 校验；
- 支持 DMA 一致性读写和预取读；
- 支持 16 种二级 Cache 散列方式；
- 支持按窗口锁二级 Cache；
- 保证读数据返回原子性。(Remark: 写数据保证原子性吗？)

二级 Cache 模块包括二级 Cache 管理模块 (scachemanage) 及二级 Cache 访问模块 (scacheaccess)。二级 Cache 管理模块负责处理器来自处理器和 DMA 的访问请求，而二级 Cache 的 TAG、目录和数据等信息存放在二级 Cache 访问模块中。为降低功耗，二级 Cache 的 TAG、目录和数据可以分开访问，二级 Cache 状态位、w 位与 TAG 一起存储，TAG 存放在 TAG RAM 中，目录存放在 DIR RAM 中，数据存放在 DATA RAM 中。失效请求访问二级 Cache，同时读出所有路的 TAG、目录和数据，并根据 TAG 来选出数据和目录。替换请求、重填请求和写回请求只操作一路的 TAG、目录和数据。

为提高一些特定计算任务的性能，二级 Cache 增加了锁机制。落在被锁区域中的二级 Cache 块会被锁住，因而不会被替换出二级 Cache （除非四路二级 Cache 中都是被锁住的块)。通过

名称	地址	位域	描述
slock0_valid	0x3FF00200	[63:63]	0 号锁窗口有效位
slock0_addr	0x3FF00200	[47:0]	0 号锁窗口锁地址
slock0_mask	0x3FF00240	[47:0]	0 号锁窗口掩码
slock1_valid	0x3FF00208	[63:63]	1 号锁窗口有效位
slock1_addr	0x3FF00208	[47:0]	1 号锁窗口锁地址
slock1_mask	0x3FF00248	[47:0]	1 号锁窗口掩码
slock2_valid	0x3FF00210	[63:63]	2 号锁窗口有效位
slock2_addr	0x3FF00210	[47:0]	2 号锁窗口锁地址
slock2_mask	0x3FF00250	[47:0]	2 号锁窗口掩码
slock3_valid	0x3FF00218	[63:63]	3 号锁窗口有效位
slock3_addr	0x3FF00218	[47:0]	3 号锁窗口锁地址
slock3_mask	0x3FF00258	[47:0]	3 号锁窗口掩码

表 4.1: 二级 Cache 锁窗口寄存器配置

confbus 可以对二级 Cache 模块内部的四组锁窗口寄存器进行动态配置，但必须保证四路二级 Cache 中一定有一路被锁住。每组窗口的大小可以根据 mask 进行调整，但不能超过整个二级 Cache 大小的 3/4。此外当二级 Cache 收到 DMA 写请求时，如果被写的区域在二级 Cache 中命中且被锁住，那么 DMA 写将直接写入到二级 Cache 而不是内存。

举例来说，一个地址 addr，在 slock0_valid 使能的情形下，如果满足如下条件

```
(addr & slock0_mask) == (slock0_addr & slock0_mask)
```

那么这个地址就被锁窗口 0 锁住了。

第五章　矩阵转置模块

龙芯 3A 中内置了两个独立于处理器核的矩阵转置模块，其基本功能是通过软件的配置，实现对存放在内存中矩阵进行从源矩阵到目标矩阵的行列转置功能。两个矩阵转置模块分别集成在龙芯 3A 的两个 HyperTransport 控制器内部 (Remark: HT0 还是 HT1，还是一个控制器一个？)，通过 1 级交叉开关实现对二级 Cache 及内存的读写。这两个转置模块的基地址为:

	转置模块 0	转置模块 1
基地址	0x3FF0_0600	0x3FF0_0700

表 5.1: 矩阵转置模块基地址

对每一个转置模块，由于转置前同一 Cache 行的元素顺序在转置后的矩阵中是分散的，为了提高读写效率，需要读入多行数据，使得这些数据可以在转置后的矩阵中以 Cache 行为单位进行写入，因此在矩阵转置模块中设置了一个大小为 32 行的缓存区，实现横向方式写入（从源矩阵读入到缓冲区），纵向读出（由缓冲区写入到目标矩阵）。

矩阵转置的工作过程为先读入 32 行源矩阵数据，再将该 32 行数据写入到目标矩阵，依次下去，直至完成整个矩阵的转置。矩阵转置模块还可以根据需要，仅进行预取源矩阵而不写目标矩阵，以此方式来实现对数据进行预取到 2 级 Cache 的操作。转置涉及的源矩阵可能是位于一个大矩阵中的一个小矩阵，因此，其矩阵地址可能不是完全连续，相邻行之间的地址会有间隔，需要实现更多的编程控制接口。下面表 5.2 和 5.3 说明了每个矩阵转置涉及到的编程接口。

Remark: 问题:

- 源矩阵和目标矩阵的地址是否可以（部分）重合？
- Cache 一致性是否由硬件保证？

地址偏移	名称	属性	说明
0x00	src_start_addr	RW	源矩阵起始地址
0x08	dst_start_addr	RW	目标矩阵起始地址
0x10	row	RW	源矩阵的一行元素个数
0x18	col	RW	源矩阵的一列元素个数
0x20	length	RW	源矩阵所在大矩阵的行跨度（字节）
0x28	width	RW	目标矩阵所在大矩阵的行跨度（字节）
0x30	trans_ctrl	RW	转置控制寄存器 (具体位域解释见表5.3)
0x38	trans_status	RO	转置状态寄存器 ([0]: 源矩阵读完毕, [1]: 目标矩阵写完毕)

表 5.2: 矩阵转置编程接口说明

字段	名称	说明
21:20		矩阵的元素大小: $2^{[21:20]}$ 字节
19:16	Awcache	写控制位。4'hF: 使用 cache 通路; 4'h0: uncache 通路; 其它值无意义
15:12	Awcmd	写控制位。Awcache 为 4'hF 时，必须设为 4'hB。否则无意义
11:8	Arcache	读控制位: 0xF 时，cache 通路;0x0，uncache 通路; 其它值无意义
7:4	Arcmd	读控制位。Arcache 为 4'hF 时，必须设为 4'hC; 否则无意义
3		目标矩阵写入完毕后，是否有效中断
2		源矩阵读取完毕后，是否有效中断
1		是否允许写目标矩阵。为 0 时，转置过程只预取源矩阵，但不写目标矩阵
0		使能位

表 5.3: trans_ctrl 寄存器位域解释 (其他位保留?)

第六章　处理器核间中断与通信

龙芯 3A 为每个处理器核都实现了 8 个核间中断寄存器（inter-processor interrupt，简称 IPI）以支持多核 BIOS 启动和操作系统运行时在处理器核之间进行中断和通信。龙芯 3A 的四个核间中断寄存器块的基地址分别为：

	Core0	Core1	Core2	Core3
IPI 基地址	0x3FF0_1000	0x3FF0_1100	0x3FF0_1200	0x3FF0_1300

表 6.1: 核间中断寄存器块的基地址

每个核间中断寄存器块 8 个寄存器的说明见表 6.2。其中四个缓存寄存器 (Remark: 这个名字好像不太好，就用邮箱寄存器如何？) IPI_Mailbox0-3 用于供启动时传递参数使用，按 64 或者 32 位的非缓存方式进行访问。

名称	偏移	权限	位宽	描述
IPI_Status	0x00	R	32	状态寄存器: 任何一位被置 1，且对应位使能情况下，处理器核 INT4 (??) 中断线被置位
IPI_Enable	0x04	RW	32	使能寄存器: 控制对应中断位是否有效
IPI_Set	0x08	W	32	置位寄存器: 在任何位写 1，对应的状态寄存器位被置 1
IPI_Clear	0x0C	W	32	清除寄存器: 在任何位写 1，对应的状态寄存器位被清 0
IPI_MailBox0	0x20	RW	64	缓存寄存器 0
IPI_MailBox1	0x28	RW	64	缓存寄存器 1
IPI_MailBox2	0x30	RW	64	缓存寄存器 2
IPI_MailBox3	0x38	RW	64	缓存寄存器 3

表 6.2: 处理器核间中断寄存器

表 6.2 列出的是单个龙芯 3A 芯片所组成的单节点多处理器系统的的核间中断相关寄存器列表。在采用多片龙芯 3A 互连构成多节点 CC-NUMA 系统时，每个芯片节点对应一个系统全局节点编号，节点内处理器核的 IPI 寄存器地址与其所在节点的基地址成固定偏移关系。例如，1 号节点 (其节点基地址为 0x1000_0000_0000) 的 2 号处理器 IPI_Enable 地址为

Core2_IPI_Enable: 0x1000_0000_0000 + 0x3FF0_1200 + 0x04 = 0x1000_3FF0_1204。

其他节点，其他处理器核，其他 IPI 寄存器依次类推。

第七章　I/O 中断

龙芯 3A 芯片支持 32 个中断源，以统一方式管理。图 reffig:interrupt 给出了这些中断对应的中断源，以及中断的路由的简单示意。

图 7-1 龙芯 3A 处理器中断路由示意图 (FIXME)

表 7.1 列出了这 32 个中断的配置寄存器。这些中断相关配置寄存器都是以位的形式对相应的中断线进行控制的：任意一个 IO 中断源可以被配置为是否使能、触发的方式、以及被路由的目标处理器核中断脚。

名称	地址	位宽	描述
Entry0-31	0x3FF0_1400-0x3FF0_141F	8	32 个 8 位中断路由寄存器
Intisr	0x3FF0_1420	32	中断状态寄存器
Inten	0x3FF0_1424	32	中断使能状态寄存器
Intenset	0x3FF0_1428	32	设置使能寄存器
Intenclr	0x3FF0_142c	32	清除使能寄存器
Intedge	0x3FF0_1438	32	触发方式寄存器
Core0_Intisr	0x3FF0_1440	32	路由给 Core0 的中断状态寄存器
Core1_Intisr	0x3FF0_1448	32	路由给 Core1 的中断状态寄存器
Core2_Intisr	0x3FF0_1450	32	路由给 Core2 的中断状态寄存器
Core3_Intisr	0x3FF0_1458	32	路由给 Core3 的中断状态寄存器

表 7.1: I/O 中断配置寄存器

使能（Enable）配置由三个寄存器控制：Inten、Intenset、和 Intenclr：每个中断与中断源的对应关系，和使能寄存器的属性配置见表 7.2。

- Inten 寄存器读取当前各中断使能的情况；
- Intenset 设置中断使能：Intenset 寄存器写 1 的位对应的中断被使能；
- Intenclr 清除中断使能：Intenclr 寄存器写 1 的位对应的中断被清除；
- Intedge 配置寄存器选择中断信号是否采用脉冲形式的（如 PCI_serr）；写 1 表示脉冲触发，写 0 表示电平触发。中断处理程序可以通过设置 Intenclr 的相应位来清除脉冲记录。

这 32 个 I/O 中断源每个都对应一个 8 位的路由配置（Entry）寄存器。龙芯 3A 中集成了 4 个处理器核。软件可以通过设置对应的路由配置寄存器，选择中断的目标处理器核，以及路由到处理器核中断 Int0 到 Int3 中的任意一个（即对应 CP0_Status 的 IP2 到 IP5 中的一个??）。路由寄存器采用向量的方式进行路由选择，其格式如图 7.1 所示。 例如 Entry2=0x48 表明 2 号中断被

位域	中断源	访问属性/缺省值			
		Intedge	Inten	Intenset	Intenclr
31:24	HT1_Int7-0	RW/0	R/0	RW/0	RW/0
23:16	HT0_Int7-0	RW/0	R/0	RW/0	RW/0
15	PCI_perr 和 serr	RW/0	R/0	RW/0	RW/0
14	保留	RW/0	R/0	RW/0	RW/0
13	Barrier	RW/0	R/0	RW/0	RW/0
12:11	MC1-0	RW/0	保留	保留	保留
10	LPC	R/1	R/0	RW/0	RW/0
9	MT_Int1	R/1	R/0	RW/0	RW/0
8	MT_Int0	R/0	R/0	RW/0	RW/0
7:4	PCI_Int3-0	R/0	R/0	RW/0	RW/0
3:0	SYS_Int3-0	RW/0	R/0	W/0	W/0

表 7.2: 中断使能控制寄存器属性

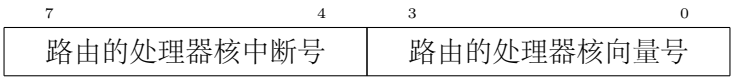


图 7.1: 中断路由寄存器位域图

路由到 3 号处理器的 INT2 上。(why??)

Remark: 问题:

- 同一中断可以同时 dispatch 到几个核吗？如果有冲突，硬件如何处理？
- MT_Int0 和 MT_Int1 的缺省值不同。有什么原因吗？
- I/O interrupt 和 IPI 中断的关系？

第八章　DDR2/3 SDRAM 控制器配置

第九章　HyperTransport 控制器

龙芯 3 号提供了两个 HyperTransport 控制器 (HT0 和 HT1)，用于连接外部设备及多芯片互联。这两个 HT 控制器实现了对 IO Cache 一致性的支持。在 Cache 一致性支持模式下，IO 设备的 DMA 访问对于 Cache 层透明，即不需要通过 Cache 指令，而是由硬件自动维护其一致性。同时，用户程序也可以通过对非缓存地址窗口设置（见 9.4.2.4 节），对某些特定内存段不提供 IO Cache 一致性支持，这有助于提高某些设备如显卡等的效率。当系统引脚 ICCC_EN 被置位时，HT0 控制器将会被用于多芯片互联，这时控制器硬件自动维护相联的 CPU 之间的 Cache 一致性 (即片间 Cache 一致性)。HT1 控制器只能用于连接 IO 外设。

龙芯 3 号的 HT 控制器最高支持双向 16 位宽度，最高运行频率为 800Mhz。在系统自动初始化建立连接后，HT 控制器总是工作在最低宽度、最低频率。用户可通过修改协议中的配置寄存器对需要的运行频率与宽度进行修改，并重新初始化。具体细节见 9.3.4 节。

龙芯 3 号 HyperTransport 控制器的主要特征如下：

- 两个 HyperTransport 控制器：HT0 和 HT1；
- HT0 控制器用于多片互联时可配置为片间 Cache 一致性模式；
- 外设 DMA 空间 Cache/Uncache 可配置；
- 最大支持 16 位宽度，同时每个 HT 控制器可配置为两个 8 位 HT 控制器；
- 支持频率：200/400/800Mhz；
- 总线控制信号（包括 PowerOK，Rstn，LDT_Stopn）方向可配置。

9.1　HT 协议支持

龙芯 3 号的 HyperTransport 总线支持 HT 1.03 协议中的大部分命令，并增加了支持多芯片互联的扩展一致性协议。在这两种模式（标准及扩展）下，HyperTransport 接收端可接收的命令，以及发送端可发送的命令分别列在表 9.1 和 9.2 中。注意，HT 总线的原子操作命令以及 EOI (End of Interrupt, 自动中断结束) 未被支持。

Post Write：HyperTransport 协议中，这种写访问不需要等待写完成响应，即在控制器向总线发出这个写访问之后就将对处理器进行写访问完成响应。

9.2　HT 信号引脚

HyperTransport 总线由传输总线信号和控制信号引脚等组成。这些控制信号包括 HT_8x2，HT_Mode，HT_PowerOK，HT_Rstn，HT_Ldt_Stopn，HT_Ldt_Reqn，它们的电压值由系统引脚

编码	通道	命令	标准模式	扩展（一致性）
000000	—	NOP	空包或流控	—
000001	NPC	Flush	无操作	—
x01xxx	NPC/PC	Write	[5]: 0 – Nonposted; 1 – Posted	[5]: 必为 1，Posted
			[2]: 0 – 字节; 1 – 双字	[2]: 0 – 字节; 1 – 双字
			[1:0]: 忽略	[1]: 忽略；[0]: 必为 1
01xxxx	NPC	Read	[3]: 忽略;	[3]: 忽略;
			[2]: 0 – 字节; 1 – 双字	[2]: 0 – 字节; 1 – 双字
			[1:0]: 忽略	[1]: 忽略; [0]: 必为 1
110000	R	RdResponse	读操作返回	—
110011	R	TgtDone	写操作返回	—
110100	PC	WrCoherent	—	写命令扩展
110101	PC	WrAddr	—	写地址扩展
111000	R	RespCoherent	—	读响应扩展
111001	NPC	RdCoherent	—	读命令扩展
111010	PC	Broadcast	无操作	—
111011	NPC	RdAddr	—	读地址扩展
111100	PC	Fence	保证序关系	—
111111	—	Sync/Error	Sync/Error	—

表 9.1: HyperTransport 接收端可接收的命令

编码	通道	命令	标准模式	扩展（一致性）
000000		NOP	空包或流控	–
x01x0x	NPC/PC	Write	[5]: 0 – Nonposted; 1 – Posted	[5]: 必为 1，Posted
			[2]: 0 – 字节; 1 – 双字	[2]: 0 – 字节; 1 – 双字
			[0]: 必为 0	[0]: 必为 1
010x0x	NPC	Read	[2]: 0 – 字节; 1 – 双字	[2]: 0 – 字节; 1 – 双字
			[0]: 忽略	[0]: 必为 1
110000	R	RdResponse	读操作返回	—
110011	R	TgtDone	写操作返回	—
110100	PC	WrCoherent	—	写命令扩展
110101	PC	WrAddr	—	写地址扩展
111000	R	RespCoherent	—	读响应扩展
111001	NPC	RdCoherent	—	读命令扩展
111011	NPC	RdAddr	—	读地址扩展
111111	—	Sync/Error	只会转发	–

表 9.2: HyperTransport 发送端可发送的命令

PCI_Config[7], PCI_Config[0] 设置（见表 3.1)。表 9.3 给出了这些控制引脚信号的说明及其用途。

表 9.3: HyperTransport 总线控制信号

引脚信号	名称	描述
HT_8x2	总线宽度配置	1：将总线配置为两个独立的 8 位 HT 总线 0：将总线作为单个 16 位总线使用。
HT_Lo_mode	主设备模式	1：将 HT_Lo 设为主设备模式； 0：将 HT_Lo 设为从设备模式。
HT_Lo_PowerOK	总线 PowerOK	HT_Lo PowerOK 信号
HT_Lo_Rstn	总线 Rstn	HT_Lo 连接重置信号
HT_Lo_Ldt_Stopn	总线 Ldt_Stopn	HT_Lo 连接暂停信号
HT_Lo_Ldt_Reqn	总线 Ldt_Reqn	HT_Lo 连接请求信号
HT_Hi_mode	主设备模式	1：将 HT_Hi 设为主设备模式； 0：将 HT_Hi 设为从设备模式。(HT_8x2=0 时，无效)
HT_Hi_PowerOK	总线 PowerOK	HT_Hi PowerOK 信号 (HT_8x2=0 时，无效)
HT_Hi_Rstn	总线 Rstn	HT_Hi 连接重置信号 (HT_8x2=0 时，无效)
HT_Hi_Ldt_Stopn	总线 Ldt_Stopn	HT_Hi 连接暂停信号 (HT_8x2=0 时，无效)
HT_Hi_Ldt_Reqn	总线 Ldt_Reqn	HT_Hi 连接请求信号 (HT_8x2=0 时，无效)
HT_Rx_CLKp[1:0] HT_Rx_CLKn[1:0] HT_Tx_CLKp[1:0] HT_Tx_CLKn[1:0]	CLK[1:0]	时钟信号。HT_8x2 为 1 时，CLK[1] 由 HT_Hi 控制 CLK[0] 由 HT_Lo 控制 HT_8x2 为 0 时，CLK[1] 无效 CLK[0] 由 HT_Lo 控制
HT_Rx_CTLp[1:0] HT_Rx_CTLn[1:0] HT_Tx_CTLp[1:0] HT_Tx_CTLn[1:0]	CTL[1:0]	控制信号。HT_8x2 为 1 时，CTL[1] 由 HT_Hi 控制 CTL[0] 由 HT_Lo 控制 HT_8x2 为 0 时，CTL[1] 无效 CTL[0] 由 HT_Lo 控制
HT_Rx_CADp[15:0] HT_Rx_CADn[15:0] HT_Tx_CADp[15:0] HT_Tx_CADn[15:0]	CAD[15:0]	命令地址数据信号 HT_8x2 为 1 时，CAD[15:8] 由 HT_Hi 控制 CAD[7:0] 由 HT_Lo 控制 HT_8x2 为 0 时，CAD[15:0] 由 HT_Lo 控制

在这些信号中，HT_8x2、HT_Lo_mode、HT_Hi_mode 的设置又进一步决定了其他的 HT 信号的作用及意义。首先，HT_8x2 决定了 16 位的 HT 总线是否作为两个独立的 HT 控制器控制。

- 当 HT_8x2 = 1 时，16 位的 HT 总线作为两个独立的 HT 控制器控制，这时 HT 的 40 位地址空间按如下规则划分：

  HT_Lo：address[39] = 0；
  HT_Hi：address[39] = 1。

- 当 HT_8x2 = 0 时，整个 HT 总线将作为一个 16 位总线使用，由 HT_Lo 控制，合法的地址空间为 HT_Lo 的地址，即 address[39] =0；同时所有 HT_Hi 信号无效，包括 Hi_PowerOK、Hi_Rstn、Hi_Ldt_Stopn。当然，高位设备的主模式引脚 Hi_mode 亦无效。

HT_Lo_mode、HT_Hi_mode 为主设备模式引脚。如果该引脚置位，则对应的 HT 设备被设为主设备模式，反之为从设备模式。

- 在主设备模式下，总线控制信号等由该设备驱动：这些控制信号包括 PowerOK，Rstn，和 Ldt_Stopn （注意：不包括 Reqn）；
- 控制信号可以为双向驱动：即 HT 总线两端的设备都为主设备模式。
- 主设备模式引脚同时也决定了对应的功能指针寄存器 (见表 ??) 的 “Act As Slave” 位的初始值（取反）：(1) 当该位为 0 时，HyperTransport 总线上的包中的 Bridge 位为 1，否则为 0；(2) 同时，当 Act_As_Slave 位为 0 时，如果 HT 总线上的请求地址未命中控制器的接收窗口，将作为 P2P 请求重新发回总线；否则将作为错误请求做出响应。

需要注意的是，HyperTransport 总线两端的设备配置需要一一对应，如果没有被正确设置并正确驱动，将导致 HyperTransport 接口不能正常工作。

Remark: 是不是 HT1 无所谓主从模式？

9.3 HT 控制器配置寄存器模块

HT 控制器配置寄存器模块主要用于控制从 AXI SLAVE 端或是 HT RECEIVER 端到达的配置寄存器访问，外部中断处理，并保存了大量软件可见用于控制系统各种工作方式的配置寄存器。所有用于控制 HT 控制器各种行为的配置寄存器都在本模块中。本模块的访问地址在 AXI 端为 0xFD_FB00_0000 到 0xFD_FBFF_FFFF，表 9.4 列出了其所有软件可见寄存器。

Remark: 需要说明那些寄存器及其寄存器位是针对 HT0 的；同时说明如何分别设置 HT0 和 HT1。。。

表 9.4: 配置寄存器模块寄存器列表

偏移	名称	位宽	描述
0x3C	桥控制寄存器	32	总线重置控制
0x40	功能寄存器组	32	命令功能指针
0x44		32	连接控制
0x48		32	Revision ID，Link Freq，Link Error，Link Freq Cap
0x4C		32	特征功能寄存器
0x50	自定义寄存器	32	MISC
0x54		32	
0x58		32	
0x5C		32	
0x60	接收地址窗口寄存器	32	接收地址窗口 0 使能（外部访问）
0x64		32	接收地址窗口 0 基址（外部访问）
0x68		32	接收地址窗口 1 使能（外部访问）
0x6C		32	接收地址窗口 1 基址（外部访问）
0x70		32	接收地址窗口 2 使能（外部访问）
0x74		32	接收地址窗口 2 基址（外部访问）

未完待续

表 9.4: 配置寄存器模块寄存器列表（续）

偏移	名称	位宽	描述
0x78		32	
0x7C		32	
0x80	中断向量寄存器	256	中断向量寄存器
0xA0	中断使能寄存器	256	中断使能寄存器
0xC0	中断检测配置寄存器	32	Interrupt Capability
0xC4		32	DataPort
0xC8		64	IntrInfo
0xD0	POST 地址窗口寄存器	32	POST 地址窗口 0 使能（内部访问）
0xD4		32	POST 地址窗口 0 基址（内部访问）
0xD8		32	POST 地址窗口 1 使能（内部访问）
0xDC		32	POST 地址窗口 1 基址（内部访问）
0xE0	可预取地址窗口寄存器	32	可预取地址窗口 0 使能（内部访问）
0xE4		32	可预取地址窗口 0 基址（内部访问）
0xE8		32	可预取地址窗口 1 使能（内部访问）
0xEC		32	可预取地址窗口 1 基址（内部访问）
0xF0	非缓存地址窗口寄存器	32	非缓存地址窗口 0 使能（内部访问）
0xF4		32	非缓存地址窗口 0 基址（内部访问）
0xF8		32	非缓存地址窗口 1 使能（内部访问）
0xFC		32	非缓存地址窗口 1 基址（内部访问）

在以下的小节中，将给出 HT 控制器配置寄存器模块的桥控制寄存器 (bridge control)，功能寄存器组 (capability registers) 和自定义寄存器 (MISC) 的具体解释和说明。其他的寄存器，如中断向量寄存器、中断使能寄存器、中断检测配置寄存器将在 HT 中断支持 (9.5) 一节中解释；而接收地址窗口、POST 地址窗口、可预取地址窗口、非缓存地址窗口将在 HT 地址空间及窗口配置 (9.4) 中详述。

9.3.1 桥控制寄存器（Bridge Control）

寄存器名：总线重置控制；偏移：0x3C；复位值：0x0000_0000					
位域	位域名	位宽	复位值	访问	描述
31:23	—	4	0x0		保留
22	reset	12	0x0	RW	总线复位控制 (bus reset control) $0 \rightarrow 1$：HT_RSTn 置 0，总线复位 $1 \rightarrow 0$：HT_RSTn 置 1，总线热复位
21:0	—	5	0x0		保留

表 9.5: HT 桥控制（Bridge Control）寄存器组：总线重置控制

寄存器名：命令功能指针、功能 ID；偏移：0x40；复位值：0x2001_0008					
位域	位域名	位宽	复位值	访问	描述
31:29	HOST/Sec	3	0x1	R	命令格式为 HOST/Sec
28:27	—	2	0x0	R	保留
26	Act as Slave	1	—	RW	设备是否采用从模式, 初始值由对应的 HT_mode 引脚决定（见表 9.3）
25	—	1	0x0	—	保留
24	Host Hide	1	0x0	RW	是否禁止来自 HT 总线的寄存器访问
23	—	1	0x0	—	保留
22:18	Unit ID	5	0x0	RW	主模式：记录使用的 ID 个数 从模式：记录自身 Unit ID
17	Double Ended	1	0x0	R	是否采用双主模式
16	Warm Reset	1	0x1	R	桥控制寄存器中采用热复位方式
15:8	Capabilities Pointer	8	0xA0	R	下一个功能指针寄存器的偏移地址
7:0	Capability ID	8	0x08	R	功能 ID

表 9.6: 功能寄存器：命令功能指针、功能 ID

寄存器名：修改号、链接速度、错误；偏移：0x48；复位值：0x2001_0008					
位域	位域名	位宽	复位值	访问	描述
31:16	Link Freq Cap	16	0x0025	R	支持的 HT 总线频率
15:14	—	2	0x0	—	保留
13	Over Flow Error	1	0x0	R	HT 总线包溢出
12	Protocol Error	1	0x0	RW	协议错误：总线上收到不可识别的命令
11:8	Link Freq	4	0x0	RW	HT 总线工作频率
7:0	Revision ID	8	0x23	RW	版本号：1.03

表 9.7: 功能寄存器：修改号、链接速度、错误

9.3.2 功能寄存器组（Capability Registers）

200Mhz，400Mhz，800Mhz

写入此寄存器的值后将在下次热复位或是 HT Disconnect 之后生效 0000：200M 0010：400M 0101：800M

寄存器名：Feature Capability；偏移：0x4C					
位域	位域名	位宽	复位值	访问	描述
31:9	—	25	0x0	—	保留
8	Extended Register	1	0x0	R	没有扩展寄存器
7:4	—	3	0x0	—	保留
3	Extended CTL Time	1	0x0	R	扩展控制时间，不需要
2	CRC Test Mode	1	0x0	R	不支持 CRC 检验模式
1	LDT_Stopn	1	0x1	R	支持 LDT_Stopn 信号
0	Isochronous Mode	1	0x0	R	等时模式，不支持

表 9.8: 功能寄存器组：特征功能寄存器 (Feature Capability)

冷复位后的值为当前连接的最大宽度，写入此寄存器的值将会在下次热复位或是 HT Disconnect 之后生效 000：8 位方式 001：16 位方式

冷复位后的值为当前连接的最大宽度，写入此寄存器的值将会在下次热复位或是 HT Disconnect 之后生效

9.3.3 自定义寄存器（MISC）

将除了标准中断之外的其它中断重定向到哪个中断向量中（包括 SMI，NMI，INIT，INTA，INTB，INTC，INTD）总共 256 个中断向量，本寄存器表示的是中断向量的高 5 位

HT 内部中断向量如下：000：SMI 001：NMI 010：INIT 011：Reservered 100：INTA 101：INTB 110：INTC 111：INTD

0x0：最高优先级 0xF：最低优先级. 对于各个通道的优先级均采用根据时间变化提高的优先级策略，该组存器用于配置各个通道的初始优先级

9.3.4 HT 总线重初始化

HyperTransport 的初始化在每次复位完成后自动开始，冷启动后 HyperTransport 总线将自动工作在最低频率 (200Mhz) 与最小宽度 (8bit)，并尝试进行总线初始化握手。初始化是否完成的状态可以由寄存器“Init Complete”（见 9.6.2 节）读出。初始化完成后，总线的宽度可以由寄存器“Link Width Out”与“Link Width In”（见 9.6.2 节）读出。在初始化完成后，用户可以重新对寄存器“Link Width Out”，“Link Width In”以及“Link Freq”进行编程，同时需要对对方设备的相应寄存器也进行编程，编程完成后需要重新热复位总线或是通过 HT_Ldt_Stopn 信号对总线进行重新初始化操作，以使编程后的值在重新初始化后生效。在重新初始化后 HyperTransport 总线将工作在新的频率和宽度。需要注意的是，对 HT 总线两端的设备配置需要一一对应，否则将导致 HyperTransport 接口不能正常工作。

寄存器名：连接控制；偏移：0x44；复位值：0x0011_2000					
位域	位域名	位宽	复位值	访问	描述
31	—	1	0x0		保留
30:28	Link Width Out	3	0x0	R/W	发送端宽度
27	—	1	0x0		保留
26:24	Link Width In	3	0x0	R/W	接收端宽度
23	Dw Fc out	1	0x1	R	发送端不支持双字流控
22:20	Max Link Width out	3	0x0	R	HT 总线发送端最大宽度：16bits
19	Dw FC In	1	0x1	R	接收端不支持双字流控
18:16	Max Link Width In	3	0x0	R	HT 总线接收端最大宽度：16bits
15:14	—	2	0x1		保留
13	Ldt_Stopn Tristate En	1	0x0	R/W	当 HT 总线断开时，是否关闭 HT PHY。1：关闭 0：不关闭
12:10	—	3	0x0		保留
9	CRC Error (hi)	1	0x0	R/W	高 8 位发生 CRC 错
8	CRC Error (lo)	1	0x0	R/W	低 8 位发生 CRC 错
7	Trans off	1	0x0	R/W	HT PHY 关闭控制。运行 8 位协议时仅对低 8 位 PHY 有效
6	End of Chain	1	0x0	R	HT 总线末端
5	Init Complete	1	0x0	R	HT 总线初始化是否完成
4	Link Fail	1	0x0	R	指示连接失败
3:2	—	2	0x0		保留
1	CRC Flood Enable	1	0x0	R/W	发生 CRC 错误时，是否 flood HT 总线
0	Trans off (hi)	1	0x0	R/W	运行 8 位协议时，高 8 位 PHY 控制。1：关闭; 0：使能高 8 位 HT PHY

表 9.9: 功能寄存器：连接控制

寄存器名：MISC；偏移：0x50；复位值：0x2001_0008					
位域	位域名	位宽	复位值	访问	描述
31	—	1	0x0	–	保留
30	Ldt Stop Gen	1	0x0	R/W	断开总线连接：先置 0 再置 1
29	Ldt Req Gen	1	0x0	R/W	断开中唤醒总线: 先置 0 再置 1。向总线直接发出读写请求也可
28:24	Interrupt Index	5	0x0	RW	中断映射索引
23	Dword Write	1	0x1	RW	是否双字写。1：使用双字写；0：使用字节写（带 MASK）
22	Coherent Mode	1	0x0	R	片间 Cache 一致性模式：引脚 ICCC_EN 决定
21	Not Care Seqid	1	0x0	RW	是否不关心 HT 包的序号
20	Not AXI2Seqid	1	0x1	R	是否用不同 SeqID 转换总线命令。0：转换; 1：不转换，所有命令采用固定 ID 号
19:16	Fixed Seqid	4	0x0	RW	固定 ID 号
15:12	Priority NOP	4	0x4	RW	NOP 流控包优先级
11:8	Priority NPC	4	0x3	RW	Non Post 通道读写优先级
7:4	Priority RC	4	0x2	RW	Response 通道读写先级
3:0	Priority PC	4	0x1	RW	Post 通道读写优先级

表 9.10: 自定义寄存器

9.4 HT 地址空间及窗口配置

9.4.1 HyperTransport 空间

龙芯 3 号处理器中，默认的 4 个 HyperTransport 地址窗口如下： 在默认情况下（未对系统

基地址	结束地址	大小	定义
0x0C00_0000_0000	0x0CFF_FFFF_FFFF	1 Tbytes	HT0_LO 窗口
0x0D00_0000_0000	0x0DFF_FFFF_FFFF	1 Tbytes	HT0_HI 窗口
0x0E00_0000_0000	0x0EFF_FFFF_FFFF	1 Tbytes	HT1_LO 窗口
0x0F00_0000_0000	0x0FFF_FFFF_FFFF	1 Tbytes	HT1_HI 窗口

表 9.11: HyperTransport 默认地址窗口地址

地址窗口另行配置)，软件通过上述地址空间对各个 HyperTransport 接口进行访问，除此之外，软件可以通过对交叉开关上的地址窗口进行配置以用其它的地址空间对其访问（见 3.5 节）每个 HyperTransport。接口内部 40 位地址空间的具体地址窗口分布如下表 9.12所述。

9.4.2 HT 地址窗口

龙芯 3 号处理器 HyperTransport 接口中提供了多种丰富的地址窗口供用户使用，见表 9.13。这些接口窗口的设置都是通过对配置寄存器模块的相应寄存器实现的。

开始地址	结束地址	大小	定义
0x00_0000_0000	0xFC_FFFF_FFFF	1012 Gbytes	MEM 空间
0xFD_0000_0000	0xFD_F7FF_FFFF	3968 Mbytes	保留
0xFD_F800_0000	0xFD_F8FF_FFFF	16 Mbytes	中断
0xFD_F900_0000	0xFD_F90F_FFFF	1 Mbyte	PIC 中断响应
0xFD_F910_0000	0xFD_F91F_FFFF	1 Mbyte	系统信息
0xFD_F920_0000	0xFD_FAFF_FFFF	30 Mbytes	保留
0xFD_FB00_0000	0xFD_FBFF_FFFF	16 Mbytes	HT 控制器配置空间
0xFD_FC00_0000	0xFD_FDFF_FFFF	32 Mbytes	I/O 空间
0xFD_FE00_0000	0xFD_FFFF_FFFF	32 Mbytes	HT 总线配置空间
0xFE_0000_0000	0xFF_FFFF_FFFF	8 Gbytes	保留

表 9.12: HT 接口协议的地址空间

地址窗口	窗口数	总线	作用	备注
接收窗口 (见 XXX 节)	3	HT	是否接收 HT 总线的访问	落在这些地址窗口中的访问会被内部总线所接收并处理。对落在这些地址窗口外的访问，当 CPU 处于主桥模式时（即配置寄存器中 Act As Slave 位为 0），这些访问将会被以 P2P 方式重新发回到 HT 总线上；CPU 处于设备模式时，这些访问将会按照协议给出错误信息，并返回。
Post 窗口 (见 XXX 节)	2	内部总线	是否将内部总线对 HT 总线的写访问作为 Post Write	落在这些地址空间中的对外写访问将作为 Post Write。
可预取窗口 2 (见 XXX 节)	2	内部总线	判断是否接收内部的缓存访问，取指访问	当处理器核乱序执行时，会对总线发出一些猜测读访问或是取指访问，这种访问对于某些 IO 空间是错误的。默认情况下，这种访问 HT 控制器将直接返回而不对 HT 总线进行访问。通过这些窗口可以使能对 HT 总线的这类访问。
非缓存窗口 (见 XXX 节)	2	HT	判断是否使用非缓存模式访问 HT 总线	对龙芯 3 号处理器内部的 IO DMA 访问，硬件将维护其 IO 一致性信息。而通过这些窗口的配置，可以使命中这些窗口的访问以非缓存方式直接访问内存，而不通过硬件维护其 IO 一致性信息。

表 9.13: HyperTransport 地址窗口

下面，我们对每种地址窗口进行了具体的解释。以下几节中，所有地址窗口寄存器的 MASK 皆为网络掩码格式，即高位为 1，低位为 0。掩码中 0 的个数则表示了地址窗口的大小。

9.4.2.1 接收地址窗口配置寄存器

龙芯 3A 提供了 3 个接收地址窗口，用于外部访问。窗口的地址为 HT 总上接收的地址，落在本窗口内的 HT 地址将被发往 CPU，而其它地址的命令将作为 P2P (Peer-to-peer) 命令被转发回 HT 总线。每个接受窗口都由两个寄存器（使能和基地址）控制。表 9.14给它们的地址及每个寄存器的具体解释。 接受地址窗口的命中公式如下：

	窗口 0 使能	窗口 0 基址	窗口 1 使能	窗口 1 基址	窗口 2 使能	窗口 2 基址
地址偏移	0x60	0x64	0x68	0x6C	0x70	0x74

位域	位域名	位宽	复位值	访问	描述
寄存器名：HT 总线接收地址窗口使能；复位值：0x0000_0000					
31	ht_rx_image_en	1	0x0	R/W	接收地址窗口使能信号
30	ht_rx_image_trans_en	1	0x0	R/W	接收地址窗口映射使能信号
29:23	—	14	0x0	–	保留
15:0	ht_rx_image_trans[39:24]	16	0x0	R/W	接收地址窗口映射后地址的 [39:24]
HT 总线接收地址窗口基址；复位值：0x0000_0000					
31:16	ht_rx_image_base[39:24]	16	0x0	R/W	接收地址窗口地址基址的 [39:24]
15:0	ht_rx_image_mask[39:24]	16	0x0	R/W	接收地址窗口地址屏蔽的 [39:24]

表 9.14: HT 接收地址窗口寄存器

```
hit      = (BASE & MASK ) == ( ADDR & MASK )
addr_out = TRANS_EN ? TRANS | ADDR & ~MASK : ADDR
```

9.4.2.2 Post 地址窗口配置寄存器

龙芯 3A 提供了 2 个 POST 地址窗口。本窗口的地址是 AXI 总线上接收到的地址。落在本窗口的所有写访问将立即在 AXI B (??) 通道返回，并以 POST WRITE 的命令格式发给 HT 总线。而不在本窗口的写请求则以 NONPOST WRITE 的方式发送到 HT 总线，并等待 HT 总线响应后再返回 AXI 总线。每个 POST 窗口都由两个寄存器（使能和基地址）控制。列在表 **??**中。

表 **??**给出了它们的地址及每个窗口寄存器的具体解释。 POST 地址窗口的命中公式如下：

```
hit = (BASE & MASK) == (ADDR & MASK)
```

9.4.2.3 可预取地址窗口配置寄存器

龙芯 3A 提供了 2 个可预取地址窗口，用于内部访问。本窗口的地址是 AXI 总线上接收到的地址。落在本窗口的取指指令，CACHE 访问才会被发往 HT 总线，其它的取指或是 CACHE 访问将不会被发往 HT 总线，而是立即返回，如果是读命令，则会返回相应个数的无效读数据。每个可预取窗口都由两个寄存器（使能和基地址）控制。列在表 **??**中。表 9.16给出了它们的地址及每个窗口寄存器的具体位域解释。 可预取地址窗口的命中公式如下：

```
hit = (BASE & MASK) == (ADDR & MASK)
```

	窗口 0 使能	窗口 0 基址	窗口 1 使能	窗口 1 基址
地址偏移	0xD0	0xD4	0xD8	0xDC

位域	位域名	位宽	复位值	访问	描述
寄存器名：POST 地址窗口使能；复位值：0x0000_0000					
31	ht_post_en	1	0x0	R/W	Post 地址窗口使能信号
30:16	—	15	0x0	–	保留
15:0	ht_post_trans[39:24]	16	0x0	R/W	Post 地址窗口映射后地址的 [39:24]
寄存器名：POST 地址窗口基址；复位值：0x0000_0000					
31:16	ht_post_base[39:24]	16	0x0	R/W	Post 地址窗口地址基址的 [39:24]
15:0	ht_post_mask[39:24]	16	0x0	R/W	Post 地址窗口地址屏蔽的 [39:24]

表 9.15: HT 总线 Post 地址窗口寄存器

	窗口 0 使能	窗口 0 基址	窗口 1 使能	窗口 1 基址
地址偏移	0xE0	0xE4	0xE8	0xEC

位域	位域名	位宽	复位值	访问	描述
寄存器名：可预取地址窗口使能；复位值：0x0000_0000					
31	ht_prefetch_en	1	0x0	RW	可预取地址窗口使能信号
30:16	—	15	0x0	–	保留
15:0	ht_prefetch_trans[39:24]	16	0x0	RW	可预取地址窗口映射后地址的 [39:24]
可预取地址窗口基址；复位值：0x0000_0000					
31:16	ht_prefetch_base[39:24]	16	0x0	RW	可预取地址窗口地址基址的 [39:24]
15:0	ht_prefetch_mask[39:24]	16	0x0	RW	可预取地址窗口地址屏蔽的 [39:24]

表 9.16: HT 可预取地址窗口寄存器

9.4.2.4 非缓存地址窗口配置寄存器

龙芯 3A 提供了 2 个非缓存地址窗口，用于内部访问。本窗口的地址是 HT 总线上接收到的地址。落在本窗口地址的读写命令，将不会被送往二级 CACHE，也不会使得一级 CACHE 发生失效，而是被直接送至内存，或是其它的地址空间。这也就是说这个地址窗口中的读写命令将不会维持 IO 的 CACHE 一致性。这一窗口主要针对一些不会在 CACHE 中命中而可以提高存问效率的操作，如显存的访问等。每个 Uncache 窗口都由两个寄存器（使能和基地址）控制。它们的地址列在表 **??**中。表 9.17给除了每个窗口的寄存器的具体解释。 非缓存地址窗口的命中公式如下：

```
hit = (BASE & MASK) == (ADDR & MASK)
addr_out = TRANS_EN ? TRANS | ADDR & ~MASK : ADDR
```

9.4.3 HyperTransport 设备配置空间访问

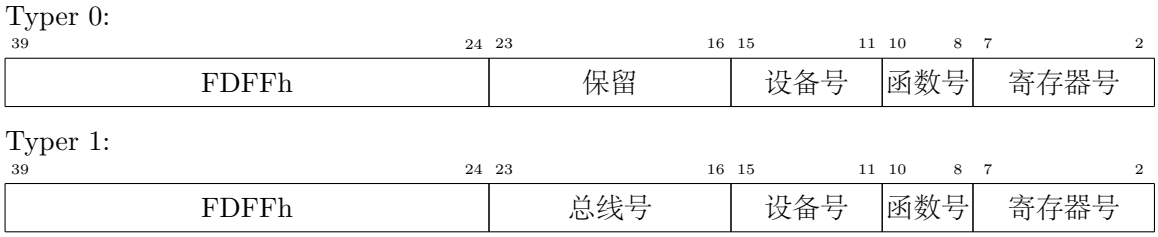
HyperTransport 接口软件层的协议与 PCI 协议基本一致，配置访问由于直接与底层协议相关，访问的方法可能略有不同。如表 9-5 中所示，配置访问空间位于地址 (0xFD_FE00_0000 ～ 0xFD_FFFF_FFFF)。对于 HT 协议中的配置访问，在龙芯 3 号中如图 9.1实现。

	窗口 0 使能	窗口 0 基址	窗口 1 使能	窗口 1 基址
地址偏移	0xF0	0xF4	0xF8	0xFC

位域	位域名	位宽	复位值	访问	描述
寄存器名：非缓存地址窗口使能；复位值：0x0000_0000					
31	ht_uncache_en	1	0x0	R/W	非缓存地址窗口使能信号
30:16	—	15	0x0	—	保留
15:0	ht_uncache_trans[39:24]	16	0x0	R/W	非缓存地址窗口映射后地址的 [39:24]
寄存器名：非缓存地址窗口基址；复位值：0x0000_0000					
31:16	ht_uncache_base[39:24]	16	0x0	R/W	非缓存地址窗口地址基址的 [39:24]
15:0	ht_uncache_mask[39:24]	16	0x0	R/W	非缓存地址窗口地址屏蔽的 [39:24]

表 9.17: HT 总线非缓存地址窗口寄存器

图 9.1: HT 协议的配置访问



9.5 HT 中断支持

HyperTransport 控制器提供了 256 个中断向量，可以支持 Fix，Arbitor 等类型的中断，但是，没有对硬件自动中断结束（EOI）提供支持。对于这两种 (??) 类型的中断，控制器在接收之后会自动写入中断寄存器中，并根据中断屏蔽寄存器的设置对系统中断控制器进行中断通知。具体的中断控制请见第 5 小节中的中断控制寄存器组。

另外，HT 控制器对 PIC 中断做了专门的支持，以加速这种类型的中断处理。一个典型的 PIC 中断由下述步骤完成：

- PIC 控制器向系统发送 PIC 中断请求；
- 系统向 PIC 控制器发送中断向量查询；
- PIC 控制器向系统发送中断向量号；
- 系统清除 PIC 控制器上的对应中断。

只有上述 4 步都完成后，PIC 控制器才会对系统发出下一个中断。对于龙芯 3 号 HyperTransport 控制器，将自动进行前 3 步的处理，并将 PIC 中断向量写入 256 个中断向量中的对应位置。而软件系统在处理了该中断之后，需要进行第 4 步处理，即向 PIC 控制器发出清中断，之后开始下一个中断的处理过程。

偏移	位域名	位宽	复位值	访问	描述
0x80	Interrupt_case[31:0]	32	0x0	R/W	中断向量寄存器 [31:0]，对应中断线 0 /HT HI 对应中断线 4
0x84	Interrupt_case[63:32]	32	0x0	R/W	中断向量寄存器 [63:32]，对应中断线 0 /HT HI 对应中断线 4
0x88	Interrupt_case[95:64]	32	0x0	R/W	中断向量寄存器 [95:64]，对应中断线 1 /HT HI 对应中断线 5
0x8C	Interrupt_case[127:96]	32	0x0	R/W	中断向量寄存器 [127:96]，对应中断线 1 /HT HI 对应中断线 5
0x90	Interrupt_case[159:128]	32	0x0	R/W	中断向量寄存器 [159:128]，对应中断线 2 /HT HI 对应中断线 6
0x94	Interrupt_case[191:160]	32	0x0	R/W	中断向量寄存器 [191:160]，对应中断线 2 /HT HI 对应中断线 6
0x98	Interrupt_case[223:192]	32	0x0	R/W	中断向量寄存器 [223:192]，对应中断线 3 /HT HI 对应中断线 7
0x9C	Interrupt_case[255:224]	32	0x0	R/W	中断向量寄存器 [255:224]，对应中断线 3 /HT HI 对应中断线 7

表 9.18: HT 总线中断向量寄存器

9.5.1 HT 中断向量寄存器

中断向量寄存器共有 256 位，其中除去 HT 总线上 Fixed 与 Arbitrated，PIC 中断直接映射到此 256 个中断向量之中，其它的中断，SMI，如 NMI，INIT，INTA，INTB，INTC，INTD 可以通过自定义寄存器（MISC）的 [28:24] 映射到任何一个 8 位中断向量上去，映射的顺序为 INTD，INTC，INTB，INTA，1’b0，INIT，NMI，SMI。此时中断向量对应值为 Interrupt Index, 内部向量 [2:0]。

9.5.2 中断使能寄存器

同样，HT 总线中断使能寄存器也有 256 位，与中断向量寄存器一一对应。置 1 为对应中断打开，置 0 则为中断屏蔽。

9.5.3 中断发现配置寄存器（Interrupt Discovery & Configuration）

9.6 HT 多处理器支持

龙芯 3 号处理器使用 HyperTransport 接口进行多处理器互联，并且可以硬件自动维护 4 个芯片之间的一致性请求。下面提供两种多处理器互联方法：

9.6.1 四片龙芯 3 号互联结构

四片 CPU 两两相联构成环状结构。每个 CPU 利用 HT0 的两个 8 位控制器与相邻两片相联，其中 HTx_LO 作为主设备，HTx_HI 作为从设备连接，由此而得到下图的互联结构：

偏移	位域名	位宽	复位值	访问	描述
0xA0	Interrupt_mask[31:0]	32	0x0	R/W	中断使能寄存器 [31:0]，对应中断线 0 /HT HI 对应中断线 4
0xA4	Interrupt_mask[63:32]	32	0x0	R/W	中断使能寄存器 [63:32]，对应中断线 0 /HT HI 对应中断线 4
0xA8	Interrupt_mask[95:64]	32	0x0	R/W	中断使能寄存器 [95:64]，对应中断线 1 /HT HI 对应中断线 5
0xAC	Interrupt_mask[127:96]	32	0x0	R/W	中断使能寄存器 [127:96]，对应中断线 1 /HT HI 对应中断线 5
0xB0	Interrupt_mask[159:128]	32	0x0	R/W	中断使能寄存器 [159:128]，对应中断线 2 /HT HI 对应中断线 6
0xB4	Interrupt_mask[191:160]	32	0x0	R/W	中断使能寄存器 [191:160]，对应中断线 2 /HT HI 对应中断线 6
0xB8	Interrupt_mask[223:192]	32	0x0	R/W	中断使能寄存器 [223:192]，对应中断线 3 /HT HI 对应中断线 7
0xBC	Interrupt_mask[255:224]	32	0x0	R/W	中断使能寄存器 [255:224]，对应中断线 3 /HT HI 对应中断线 7

表 9.19: HT 总线中断使能寄存器

位域	位域名	位宽	复位值	访问	描述
寄存器名：Interrupt Capability；偏移：0xC0；复位值：0x8000_0008					
31:24	Capabilities Pointer	8	0x80	R	Interrupt discovery configuration block
23:16	Index	8	0x0	R/W	读寄存器偏移地址
15:8	Capabilities Pointer	8	0x0	R	Capabilities Pointer
7:0	Capability ID	8	0x08	R	Hypertransport Capablity ID
寄存器名：Dataport；偏移：0xC4；复位值：0x0000_0000					
31:0	Dataport	32	0x0	R/W	当上一寄存器 Index 为 0x10 时，本寄存器读写结果为 0xA8 寄存器，否则为 0xAC
寄存器名：IntroInfo[31:0]；偏移：0xC8；复位值：0xF800_0000					
31:24	IntrInfo[31:24]	8	0xF8	R	保留
23:2	IntrInfo[23:2]	22	0x0	R/W	当发出 PIC 中断时，IntrInfo 的值用来表示中断向量
1:0	—	2	0x0	R	保留
寄存器名：IntroInfo[63:31]；偏移：0xCC；复位值：0x0000_0000					
31:0	—	32	0x0	R	保留

表 9.20: 中断发现配置寄存器（Interrupt Discovery & Configuration）

图 9-2 四片龙芯 3 号互联结构

龙芯 3 号互联路由龙芯 3 号互联路由采用简单 X-Y 路由方法。即，路由时，先 X 后 Y，以四片芯片为例，ID 号分别为 00，01，10，11。如果从 11 向 00 发出请求，则为 11 向 00 路由，首先走 X 方向，从 11 走到 10，再走 Y 方向，从 10 走到 00。而在请求的响应从 00 返回 11 时，路由首先走 X 方向，从 00 到 01，再走 Y 方向，从 01 到 11。可以看到，这是两个不同的路由线路。由于这个算法的特征，我们在构建两片芯片互联的时候，将采用不同的办法。

9.6.2 两片龙芯 3 号互联结构

由于固定路由算法的特性，我们在构建两片芯片互联时，有两种不同的方法。首先是采用 8 位 HT 总线互联。这种互联方式下，两个处理器之间只能采用 8 位 HT 互联。两个芯片号分别为 00 与 01，由路由算法，我们可以知道，两个芯片相互访问时都是通过与四片互联时一致的 8 位 HT 总线。如下所示：

图 9-3 两片龙芯 3 号 8 位互联结构 (FIXME)

但是，我们的 HT 总线最宽可以采用 16 位模式，由此最大化带宽的连接方式应该是采用 16 位互联结构。在龙芯三号中，只要把 HT0 控制器设置为 16 位模式，所有发到 HT0 控制器的命令都会被发往 HT0_LO，而不是以前的按照路由表分别发至 HT0_HI 或是 HT0_LO，这样，我们就可以在互联时使用 16 位总线。所以，我们只需要将 CPU0 与 CPU1 的 16 位模式正确配置并将高低位总线正确连接即可使用 16 位 HT 总线互联。而这种互联结构同时也可以使用 8 位的 HT 总线协议进行相互访问。所得到的互联结构如下：

图 9-4 两片龙芯 3 号 16 位互联结构 (FIXME)

第十章　低速 I/O 控制器配置

龙芯 3 号的慢速 I/O 控制器包括 PCI/PCIX 控制器、LPC 控制器、UART 控制器、SPI 控制器、GPIO 以及相关的配置寄存器等等。这些 I/O 控制器共享一个 AXI 端口，来自 CPU 的请求经过地址译码后发送到相应的设备。表 10.1 给出了这些低速设备的内部地址空间划分图 (Remark: 需要更低的地址空间也写出来吗？)。

地址名称	地址范围	大小
LPC Memory	0x1C00_0000 – 0x1DFF_FFFF	32 MByte
LPC Boot	0x1FC0_0000 – 0x1FCF_FFFF	1 MByte
PCI I/O 空间	0x1FD0_0000 – 0x1FDF_FFFF	1 MByte
PCI 控制器配置空间	0x1FE0_0000 – 0x1FE0_00FF	256 Byte
IO 控制器配置空间	0x1FE0_0100 – 0x1FE0_01DF	256 Byte
UART0 配置空间	0x1FE0_01E0 – 0x1FE0_01E7	8 Byte
UART1 配置空间	0x1FE0_01E8 – 0x1FE0_01EF	8 Byte
SPI 配置空间	0x1FE0_01F0 – 0x1FE0_01FF	16 Byte
LPC 配置空间	0x1FE0_0200 – 0x1FE0_02FF	256 Byte
PCI 配置空间	0x1FE8_0000 – 0x1FE8_FFFF	64 KByte
LPC I/O 空间	0x1FF0_0000 – 0x1FF0_FFFF	64 Kbyte
PCI MEM	其它	

表 10.1: 低速 I/O 设备地址空间

10.1　PCI/PCIX 控制器

龙芯 3 号的 PCI/PCIX 控制器的实现符合 PCIX 1.0b 和 PCI 2.3 规范。它的内部还内置了 PCI/PCIX 仲裁器，既可以作为主桥控制整个 PCI/PCIX 总线，也可以作为一个普通设备工作在 PCI/PCIX 总线上。

如表 10.1 显示，龙芯 3 号的 PCI/PCIX 控制器的配置空间起始于 0x1FE0_0000 开始的 256 字节。表 10.2 列出了所有的 PCIX 控制器配置寄存器，而表 10.3 则列出了所有和龙芯具体实现有关的寄存器（Implementation Specific Register, ISR）位域及其说明。

表 10.2: PCIX 控制器配置头

字节 3	字节 2	字节 1	字节 0	地址
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	CacheLine Size	0C
Base Address Register 0 (BAR0)				10
Base Address Register 1 (BAR1)				14
Base Address Register 2 (BAR2)				18
Base Address Register 3 (BAR3)				1C
Base Address Register 4 (BAR4)				20
Base Address Register 5 (BAR5)				24
				28
Subsystem ID		Subsystem Vendor ID		2C
				30
			Capabilities Pointer	34
				38
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	3C
Implementation Specific Register (ISR40)				40
Implementation Specific Register (ISR44)				44
Implementation Specific Register (ISR48)				48
Implementation Specific Register (ISR4C)				4C
Implementation Specific Register (ISR50)				50
Implementation Specific Register (ISR54)				54
Implementation Specific Register (ISR58)				58
				...
PCIX Command Register				E0
PCIX Status Register				E4

龙芯 3A 的 PCIX 控制器支持三个 64 位窗口。这些窗口的基址是由三对 PCIX 控制寄存器 {BAR1, BAR0}、{BAR3, BAR2}、{BAR5, BAR4} 决定的，而这三个窗口的大小、使能以及其它细节则是由做 I/O 控制器配置的三个寄存器 PCI_Hit0_Sel, PCI_Hit1_Sel, PCI_Hit2_Sel 控制的。它们的具体位域说明见表 10.4。

表 10.3: 龙芯 PCI 具体实现寄存器（ISR）

位域	位域名	访问	复位值	说明
REG_40 Remark: 用 ISR_40 如何？				
31	tar_read_io	读写	0	target 端收到对 IO 或者是不可预取区域的访问
30	tar_read_discard	读写	0	target 端的 delay 请求被丢弃

未完待续

表 10.3: 龙芯 PCI 具体实现寄存器（ISR）（续）

位域	位域名	访问	复位值	说明
29	tar_resp_delay	读写	0	target 访问何时给出 delay/split 0: 超时后 1: 马上
28	tar_delay_retry	读写	0	target 访问重试策略 0: 根据内部逻辑（见 29 位）1: 马上重试
27	tar_read_abort_en	读写	0	若 target 对内部的读请求超时，是否让以 target-abort 回应
26:25	Reserved	–	0	–
24	tar_write_abort_en	读写	0	若 target 对内部的写请求超时，是否让以 target-abort 回应
23	tar_master_abort	读写	0	是否允许 master-abort
22:20	tar_subseq_timeout	读写	000	target 后续延迟超时 000: 8 周期其它: 不支持
19:16	tar_init_timeout	读写	0000	target 初始延迟超时 PCI 模式下 0: 16 周期 1-7: 禁用计数器 8-15: 8-15 周期 PCIX 模式下超时计数固定为 8 周期，此处配置影响最大的 delay 访问数 0: 8 delay 访问 8: 1 delay 访问 9: 2 delay 访问 10: 3 delay 访问 11: 4 delay 访问 12: 5 delay 访问 13: 6 delay 访问 14: 7 delay 访问 15: 8 delay 访问
15:4	tar_pref_boundary	读写	000h	可预取边界配置（以 16 字节为单位）FFF: 64KB 到 16byte FFE: 64KB 到 32byte FF8: 64KB 到 128byte
3	tar_pref_bound_en	读写	0	–
2	Reserved (??)	–	0	使用 tar_pref_boundary 的配置 0: 预取到设备边界 1: 使用 tar_pref_boundary
1	tar_splitw_ctrl	读写	0	target split 写控制 0: 阻挡除 Posted Memory Write 以外的访问 1: 阻挡所有访问，直至 split 完成
0	mas_lat_timeout	读写	0	禁用 mater 访问超时 0: 允许 master 访问超时 1: 不允许
REG_44				
31:0	Reserved	–	–	–
REG_48				
31:0	tar_pending_seq	读写	0	target 未处理完的请求号位向量对应位写 1 可清
REG_4C				
31:30	Reserved	-	-	
29	mas_write_defer	读写	0	允许后续的读越过前面未完成的写 (只对 PCI 有效)

未完待续

表 10.3: 龙芯 PCI 具体实现寄存器（ISR）（续）

位域	位域名	访问	复位值	说明
28	mas_read_defer	读写	0	允许后续的读写越过前面未完成的读 (只对 PCI 有效)
27	mas_io_defer_cnt	读写	0	在外的最大 IO 请求数 0: 由控制, 1: 1 (?)
26:24	mas_read_defer_cnt	读写	010	master 支持在外读的最大数 (只对 PCI 有效) 0: 8 1-7: 1-7 注：一个双地址周期访问占两项
23:16	err_seq_id	只读	00h	target/master 错误号
15	err_type	只读	0	target/master 出错的命令类型 0:
14	err_module	只读	0	出错的模块 0: target 1: master
13	system_error	读写	0	target/master 系统错（写 1 清）
12	data_parity_error	读写	0	target/master 数据奇偶错（写 1 清）
11	ctrl_parity_error	读写	0	target/master 地址奇偶错（写 1 清）
10:0	Reserved	-	-	
REG_50				
31:0	mas_pending_seq	读写	0	master 未处理完的请求号位向量对应位写 1 可清
REG_54				
31:0	mas_split_err	读写	0	split 返回出错的请求号位向量
REG_58				
31:30	Reserved	-	-	
29:28	tar_split_priority	读写	0	target split 返回优先级 0 最高, 3 最低
27:26	mas_req_priority	读写	0	master 对外的优先级 0 最高, 3 最低
25	Priority_en	读写	0	仲裁算法（在 master 的访问和 target 的 split 返回间做仲裁）0: 固定优先级 1: 轮转
24:18	保留	-	-	
17	mas_retry_aborted	读写	0	master 重试取消（写 1 清）
16	mas_trdy_timeout	读写	0	master TRDY 超时计数
15:8	mas_retry_value	读写	00h	master 重试次数 0: 无限重试 1-255: 1-255 次
7:0	mas_trdy_count	读写	00h	master TRDY 超时计数器 0: 禁用 1-255: 1-255 拍

FIXME: put more details about the PCI controller, copy some materials from the 2F document.

10.1.1 PCIX I/O 控制器配置

PCIX I/O 配置寄存器主要用于配置 PCI/PCIX 控制器的地址窗口、仲裁器以及 GPIO 控制器。表 10.4 列出了所有的 PCIX I/O 配置寄存器，而这些寄存器的详细说明则在表 10.5 中给出。

表 10.4: PCIX I/O 控制寄存器

地址偏移	寄存器名	说明
00	PonCfg	上电配置
04	GenCfg	常规配置
08	–	保留
0C	–	保留
10	PCIMap	PCI 映射
14	PCIX_Bridge_Cfg	PCIX 桥相关配置
18	PCIMap_Cfg	PCI 配置读写设备地址
1C	GPIO_Data	GPIO 数据
20	GPIO_EN	GPIO 方向
24	–	保留
28	–	保留
2C	–	保留
30	–	保留
34	–	保留
38	–	保留
3C	–	保留
40	Mem_Win_Base_L	可预取窗口基址低 32 位
44	Mem_Win_Base_H	可预取窗口基址高 32 位
48	Mem_Win_Mask_L	可预取窗口掩码低 32 位
4C	Mem_Win_Mask_H	可预取窗口掩码高 32 位
50	PCI_Hit0_Sel_L	PCI 窗口 0 控制低 32 位
54	PCI_Hit0_Sel_H	PCI 窗口 0 控制高 32 位
58	PCI_Hit1_Sel_L	PCI 窗口 1 控制低 32 位
5C	PCI_Hit1_Sel_H	PCI 窗口 1 控制高 32 位
60	PCI_Hit2_Sel_L	PCI 窗口 2 控制低 32 位
64	PCI_Hit2_Sel_H	PCI 窗口 2 控制高 32 位
68	PXArb_Config	PCIX 仲裁器配置
6C	PXArb_Status	PCIX 仲裁器状态

在发起配置空间读写前，应用程序应先配置好 PCIMap_Cfg 寄存器，告诉控制器欲发起的配置操作的类型和高 16 位地址线上的值。然后对 0x1fe80000 开始的 2K 空间进行读写即可访问对应设备的配置头。设备号由根据 PCIMap_Cfg[15:0] 从低到高优先编码得到。配置操作地址生成见图 10-1。

图 10-1 配置读写总线地址生成

表 10.5: IO 控制寄存器位域解释

位域	字段名	访问	复位值	说明
CR00: PonCfg				
31:24	保留	只读		
23:16	pon_pci_config	只读	pci_config	PCI_Config 引脚值
15:8	保留	只读	lio_ad[15:8]	
7:0	pcix_bus_dev	只读	lio_ad[7:0]	PCIX Agent 模式下 CPU 取指所使用的总线、设备号
CR04, CR08: 保留				
31:0	保留	只读	0	
CR10: PCIMap				
31:18	保留	只读	0	
17:12	trans_lo2	读写	0	PCI_Mem_Lo2 窗口映射地址高 6 位
11:6	trans_lo1	读写	0	PCI_Mem_Lo1 窗口映射地址高 6 位
5:0	trans_lo0	读写	0	PCI_Mem_Lo0 窗口映射地址高 6 位
CR14: PCIX_Bridge_Cfg				
31:18	保留	只读	0	
6	pcix_ro_en	读写	0	PCIX 桥是否允许写越过读
5:0	pcix_rgate	读写	6'h18	PCIX 模式下向 DDR2 发读取数门限
CR18: PCIMap_Cfg				
31:17	保留	只读	0	
16	conf_type	读写	0	配置读写的类型
15:0	dev_addr	读写	0	PCI 配置读写时 AD 线高 16 位
CR1C: GPIO_Data				
31:20	保留	只读	0	
19:16	gpio_in	读写	0	GPIO 输入数据
15:4	保留	只读	0	
3:0	gpio_out	读写	0	GPIO 输出数据
CR20: GPIO_EN				
31:4	保留	只读	0	
3:0	gpio_en	读写	F	高为输入，低输出
CR24,2C,30,34,38,3C: 保留				
31:0	保留	只读	0	保留
CR50,54/58,5C/60,64: PCI_Hit*_Sel				
63	burst_cap	读写	1	是否允许突发传送
62:12	bar_mask	读写	0	窗口大小掩码（高位 1，低位 0）
11:4	保留	只读	0	
3	pref_en	读写	0	预取使能
2:1	pci_img_size	读写	2'b11	00: 32 位；10: 64 位；其它: 无效
0	保留	只读	0	

未完待续

表 10.5: IO 控制寄存器位域解释（续）

位域	字段名	访问	复位值	说明
CR68:PXArb_Config				
31:13	保留	只读	0	
23:16	rude_dev	读写	0	强制优先级设备为 1 的位对应的 PCI 设备在得到总线后可以通过持续请求来占住总线
15:8	level	读写	8'h01	处于第一级的设备
7:6	park_delay	读写	2'b11	从没有设备请求总线开始到触发停靠默认设备行为的延迟 00: 0 周期 01: 8 周期 10: 32 周期 11: 128 周期
5:3	default_master	读写	0	总线停靠默认主设备号
2	default_mas_en	读写	1	总线停靠到默认主设备 0：停靠到最后一个主设备 1：停靠到默认主设备
1	disable_broken	读写	0	禁用损坏的主设备
0	device_en	读写	1	外部设备允许
CR6C: PXArb_Status				
31:11	保留	只读	0	
10:8	Last_master	只读	0	最后使用总线的主设备
7:0	broken_master	只读	0	损坏的主设备（改变禁用策略时清零）

PCI/PCIX 仲裁器实现了两级轮转仲裁、总线停靠和损坏主设备的隔离：其配置和状态由 PXArb_Config 和 PXArb_Status 寄存器控制。PCI/PCIX 总线请求与应答线分配见表 **??**。

请求与应答线	描述
0	内部集成 PCI/PCIX 控制器
7:1	外部请求 6 0

表 10.6: PCI/PCIX 总线请求与应答线分配

基于轮转的 PCIX 仲裁算法对设备提供了两个不同级别，其特点是第二级整体作为一个一级成员进行调度。当多个设备同时申请总线时，轮转完一次第一级设备，如果第二级整体被选中，那么第二级中优先级最高的设备将得到总线 (Remark: 如果优先级相同呢，随机或者不可能出现？)。仲裁器被设计成任何时候只要条件允许就可以切换：对于一些不符合协议的 PCI 设备，这样做可能会使之不正常。软件可以使用强制优先级可以让这些设备通过持续请求来占有总线。总线停靠是指当没有设备请求使用总线时是否选择其中一个给出允许信号。对于已经得到允许的设备而言，直接发起总线操作能够提高效率。龙芯 3 号的 PCI 仲裁器提供两种停靠模式：最后一个主设备和默认主设备。如果在特殊场合下没有设备能够停靠，则可以将仲裁器设置为停靠到默认 0 号主设备 (内部控制器)，且设置依靠延迟为 0。

位域	字段名	访问	复位值	说明
REG0				
REG0[31]	SIRQ_EN	读写	0	SIRQ 使能控制
REG0[23:16]	LPC_MEM_TRANS	读写	0	LPC Memory 空间地址转换控制
REG0[15:0]	LPC_SYNC_TIMEOUT	读写	0	LPC 访问超时计数器
REG1				
REG1[31]	LPC_MEM_IS_FWH	读写	0	LPC Memory 空间访问类型设置
REG1[17:0]	LPC_INT_EN	读写	0	LPC SIRQ 中断使能
REG2				
REG2[17:0]	LPC_INT_SRC	读写	0	LPC SIRQ 中断源指示
REG3				
REG3[17:0]	LPC_INT_CLEAR	写	0	LPC SIRQ 中断清除

表 10.7: LPC 配置寄存器含义

10.2 LPC 控制器

LPC (Low-Pin Count) 控制器具有以下特性:

- 符合 LPC1.1 规范;
- 支持 LPC 访问超时计数器;
- 支持 Memory Read、Memory write 访问类型;
- 支持 Firmware Memory Read、Firmware Memory Write 访问类型（单字节);
- 支持 I/O read、I/O write 访问类型;
- 支持 Memory 访问类型地址转换;
- 支持 Serizlized IRQ 规范，提供 17 个中断源。

如表 10.1 显示，有四个与 LPC 控制器相关的地址空间，它们分别是 LPC Boot，LPC Memory，LPC I/O 空间，和 LPC 配置空间；同时 LPC 配置空间的基地址为 0x1FE0_0200。LPC 控制器配置寄存器由 4 个 32 位寄存器组成。这些配置寄存器的含义见表 10.7。

与 LPC 相关的地址空间中，LPC Boot 是系统启动时处理器最先访问的地址空间。这个地址空间支持 LPC Memory 或 Firmware Memory 这两种不同访问类型。系统启动时发出哪种访问类型由 LPC_ROM_INTEL 引脚控制 (Remark: where is the definition of this signal?)。LPC_ROM_INTEL 引脚上拉时发出 LPC Firmware Memory 访问，下拉时发出 LPC Memory 访问类型。

LPC Memory 地址空间是系统用 Memory/Firmware Memory 访问的地址空间。LPC 控制器发出哪种类型的内存访问，由 LPC 控制器的配置寄存器 REG1 的 LPC_MEM_IS_FWH 位决定。处理器发往这个地址空间的地址可以进行地址转换。转换后的地址由 LPC 控制器的配置寄存器 REG0 的 LPC_MEM_TRANS 位设置。处理器发往 LPC I/O 地址空间的访问按照 LPC I/O 访问类型发往 LPC 总线，地址为地址空间的低 16 位。

I/O 地址	DLAB = 0 读	DLAB = 0 写	DLAB = 1 读	DLAB = 1 写
base	DAT	DAT	LSB	LSB
base+1	IER	IER	MSB	MSB
base+2	IIR	FCR	IIR	FCR
base+3	LCR	LCR	LCR	LCR
base+4	MCR	MCR	MCR	MCR
base+5	LSR	–	LSR	–
base+6	MSR	–	MSR	–
base+7	SCR	SCR	SCR	SCR

表 10.8: UART 转换表：端口 vs 寄存器

10.3 UART 控制器

龙芯 3 号的 UART 控制器具有以下特性

- 全双工异步数据接收/发送；
- 可编程的数据格式；
- 16 位可编程时钟计数器；
- 支持接收超时检测；
- 带仲裁的多中断系统；
- 仅工作在 FIFO 方式；
- 在寄存器与功能上兼容 NS16550A。

本模块有两个并行工作 UART 接口，它们的功能寄存器完全一样，只是访问基址不同。如表 10.1 显示，UART0 和 UART1 的寄存器物理地址基址分别为 0x1FE0_01E0，0x1FE0_01E8。每个 UART 控制器都有 10 个 8 位的控制寄存器，他们的详细信息列在表 10.9 中。

表 10.9: UART 控制寄存器说明

位域	位域名	位宽	访问	说明
数据寄存器（DAT）；地址偏移：0x00；复位值：0x00				
7:0	Tx FIFO	8	W	数据传输寄存器
中断使能寄存器（IER）；地址偏移：0x01；复位值：0x00				
7:4	Reserved	4	RW	保留
3	IME	1	RW	Modem 状态中断使能 ‘0’ – 关闭 ‘1’ – 打开
2	ILE	1	RW	接收器线路状态中断使能 ‘0’ – 关闭 ‘1’ – 打开
1	ITxE	1	RW	传输保存寄存器为空中断使能 ‘0’ – 关闭 ‘1’ – 打开
0	IRxE	1	RW	接收有效数据中断使能 ‘0’ – 关闭 ‘1’ – 打开

未完待续

表 10.9: UART 控制寄存器说明（续）

位域	位域名	位宽	访问	说明
中断标识寄存器（IIR）；地址偏移：0x02；复位值：0xC1				
7:4	Reserved	4	R	保留
3:1	II	3	R	中断源表示位，详见下表
0	INTp	1	R	中断表示位
FIFO 控制寄存器（FCR）；地址偏移：0x03；复位值：0xC0				
7:6	TL	2	W	接收 FIFO 提出中断申请的 trigger 值 “00” 1 字节 “01” 4 字节 “10” 8 字节 “11” 14 字节
5:3	Reserved	3	W	保留
2	Txset	1	W	“1” 清除发送 FIFO 的内容，复位其逻辑
1	Rxset	1	W	“1” 清除接收 FIFO 的内容，复位其逻辑
0	Reserved	1	W	保留
线路控制寄存器（LCR）；地址偏移：0x0；复位值：0x00				
7	dlab	1	RW	分频锁存器访问位 ‘1’ – 访问操作分频锁存器 ‘0’ – 访问操作正常寄存器
6	bcb	1	RW	打断控制位 ‘1’ – 此时串口的输出被置为 0(打断状态). ‘0’ – 正常操作
5	spb	1	RW	指定奇偶校验位 ‘0’ – 不用指定奇偶校验位 ‘1’ – 如果 LCR[4] 位是 1 则传输和检查奇偶校验位为 0。如果 LCR[4] 位是 0 则传输和检查奇偶校验位为 1。
4	eps	1	RW	奇偶校验位选择 ‘0’ – 在每个字符中有奇数个 1 (包括数据和奇偶校验位) ‘1’ – 在每个字符中有偶数个 1
3	pe	1	RW	奇偶校验位使能 ‘0’ – 没有奇偶校验位 ‘1’ – 在输出时生成奇偶校验位，输入则判断奇偶校验位
2	sb	1	RW	定义生成停止位的位数 ‘0’ – 1 个停止位 ‘1’ – 在 5 位字符长度时是 1.5 个停止位，其他长度是 2 个停止位
1:0	bec	2	RW	设定每个字符的位数 ‘00’ – 5 位 ‘01’ – 6 位 ‘10’ – 7 位 ‘11’ – 8 位
MODEM 控制寄存器（MCR）；地址偏移：0x0；复位值：0x00				
7:5	Reserved	3	W	保留
4	Loop	1	W	回环模式控制位 ‘0’ – 正常操作 ‘1’ – 回环模式。在在回环模式中，TXD 输出一直为 1，输出移位寄存器直接连到输入移位寄存器中。其他连接如下。DTR DSR RTS CTS Out1 RI Out2 DCD
3	OUT2	1	W	在回环模式中连到 DCD 输入
2	OUT1	1	W	在回环模式中连到 RI 输入
1	RTSC	1	W	RTS 信号控制位
0	DTRC	1	W	DTR 信号控制位

未完待续

表 10.9: UART 控制寄存器说明（续）

位域	位域名	位宽	访问	说明
线路状态寄存器（LSR）；地址偏移：0x0；复位值：0x00				
7	ERROR	1	R	错误表示位 ‘1’ – 至少有奇偶校验位错误，帧错误或打断中断的一个。‘0’ – 没有错误
6	TE	1	R	传输为空表示位 ‘1’ – 传输 FIFO 和传输移位寄存器都为空。给传输 FIFO 写数据时清零 ‘0’ – 有数据
5	TFE	1	R	传输 FIFO 位空表示位 ‘1’ – 当前传输 FIFO 为空，给传输 FIFO 写数据时清零 ‘0’ – 有数据
4	BI	1	R	打断中断表示位 ‘1’ – 接收到起始位＋数据＋奇偶位＋停止位都是 0，即有打断中断 ‘0’ – 没有打断
3	FE	1	R	帧错误表示位 ‘1’ – 接收的数据没有停止位 ‘0’ – 没有错误
2	PE	1	R	奇偶校验位错误表示位 ‘1’ – 当前接收数据有奇偶错误 ‘0’ – 没有奇偶错误
1	OE	1	R	数据溢出表示位 ‘1’ – 有数据溢出 ‘0’ – 无溢出
0	DR	1	R	接收数据有效表示位 ‘0’ – 在 FIFO 中无数据 ‘1’ – 在 FIFO 中有数据
MODEM 状态寄存器（MSR）；地址偏移：0x0；复位值：0x00				
7	CDCD	1	R	DCD 输入值的反，或者在回环模式中连到 Out2
6	CRI	1	R	RI 输入值的反，或者在回环模式中连到 OUT1
5	CDSR	1	R	DSR 输入值的反，或者在回环模式中连到 DTR
4	CCTS	1	R	CTS 输入值的反，或者在回环模式中连到 RTS
3	DDCD	1	R	DDCD 指示位
2	TERI	1	R	RI 边沿检测。RI 状态从低到高变化
1	DDSR	1	R	DDSR 指示位
0	DCTS	1	R	DCTS 指示位
分频锁存器 1；地址偏移：0x0；复位值：0x00				
7:0	LSB	8	RW	存放分频锁存器的低 8 位
分频锁存器 2；地址偏移：0x0；复位值：0x00				
7:0	MSB	8	RW	存放分频锁存器的高 8 位

对 LSR 寄存器进行读操作时，LSR[4:1] 和 LSR[7] 被清零，LSR[6:5] 在给传输 FIFO 写数据时清零，LSR[0] 则对接收 FIFO 进行判断。

表 10.10: 中断控制功能表

Bit3	Bit2	Bit1	优先级	中断类型	中断源	中断复位控制
0	1	1	1st	接收线路状态	奇偶、溢出或帧错误，或打断中断	读 LSR

表 10.10: 中断控制功能表

Bit3	Bit2	Bit1	优先级	中断类型	中断源	中断复位控制
0	1	0	2nd	接收到有效数据	FIFO 的字符个数达到 trigger 的水平	FIFO 的字符个数低于 trigger 的值
1	1	0	2nd	接收超时	在 FIFO 至少有一个字符，但在 4 个字符时间内没有任何操作，包括读和写操作	读接收 FIFO
0	0	1	3rd	传输保存寄存器为空	传输保存寄存器为空	写数据到 THR 或者多 IIR
0	0	0	4th	Modem 状态	CTS, DSR, RI or DCD	读 MSR

10.4 SPI 控制器

龙芯 3 号的 SPI (serial peripheral interface) 控制器具有以下特性：

- 全双工同步串口数据传输；
- 支持到 4 个的变长字节传输；
- 主模式支持；
- 模式故障产生错误标志并发出中断请求；
- 双缓冲接收器；
- 极性和相位可编程的串行时钟；
- 可在等待模式下对 SPI 进行控制。

SPI 控制器模块寄存器物理地址基址为 0x1FE001F0。

表 10.11: SPI 控制器模块寄存器

位域	位域名	位宽	访问	描述
控制寄存器（SPCR）0x00 0x10				
7	Spie	1	RW	中断输出使能信号高有效
6	spe	1	RW	系统工作使能信号高有效
5	–	1	RW	保留
4	mstr	1	RW	master 模式选择位，此位一直保持 1
3	cpol	1	RW	时钟极性位
2	cpha	1	RW	时钟相位位 1 则相位相反，为 0 则相同
1:0	spr	2	RW	与 SPER:spre 一起设定 sclk_o 分频

未完待续

SPER:spre	00	01	10	11	00	01	10	11	00	01	10	11
SPCR:spr	00	00	00	00	01	01	01	01	10	10	10	10
分频系数	2	4	16	32	8	64	128	256	512	1024	2048	4096

表 10.12: SPI 设备分频系数

表 10.11: SPI 控制器模块寄存器（续）

位域	位域名	位宽	访问	描述
状态寄存器（SPSR）0x01 0x05				
7	spif	1	RW	中断标志位 1 表示有中断申请，写 1 则清零
6	wcol	1	RW	写寄存器溢出标志位为 1 表示已经溢出, 写 1 则清零
5:4	–	2	RW	保留
3	wffull	1	RW	写寄存器满标志 1 表示已经满
2	wfempty	1	RW	写寄存器空标志 1 表示空
1	rffull	1	RW	读寄存器满标志 1 表示已经满
0	rfempty	1	RW	读寄存器空标志 1 表示空
数据寄存器（TxFIFO）0x02 0x00				
7:0	Tx FIFO	8	W	数据传输寄存器
外部寄存器（SPER）0x03 0x00				
7:6	icnt	2	RW	传输完多少个字节后送出中断申请信号。00: 1 字节; 01: 2 字节; 10: 3 字节; 11: 3 字节 (??)。
5:2	–	4	RW	保留
1:0	spre	2	RW	与 SPCR:spr 一起设定分频比率

第十一章　中断的配置及使用

material largely overlaps with that of chapter 7. Question: move below to there?

11.1　中断的流程

龙芯 3A 处理中断的流程，从外部中断请求到内核对中断的处理，其过程都是一样的。下图给出的就是 3A-690e 板卡的中断处理的流程图。

图 11-1 3A-690e 中断流程图 (FIXME)

11.2　中断路由及中断使能

龙芯 3A 芯片最多支持 32 个中断源，以统一方式进行管理，如图 reffig:IntDispatch (FIXME: use the one on chapter 7) 所示，任意一个 IO 中断源可以被配置为是否使能、触发的方式、以及被路由的目标处理器核中断脚。

11.2.1　中断路由

在龙芯 3A 中集成了 4 个处理器核，上述的 32 位中断源可以通过软件配置选择期望中断的目标处理器核。进一步，中断源可以选择路由到处理器核中断 INT0 到 INT3 中的任意一个，即对应 CP0_Status 的 IP2 到 IP5。也就是说上图所示 CORE0 CORE3 的 IP0 IP3 对应的就是 CP0_Status 的 IP2 IP5。个 I/O 中断 32 源中每一个都对应一个 8 位的路由控制器，表 ref(tab in ch7): just describ there is fine...路由寄存器采用向量的方式进行路由选择，如 0x48 标示路由到 3 号处理器的 INT2 上。中断路由寄存器的说明

为了便于理解，下面会给出 3A-690e 和 3A-KD60 两个板卡在中断路由上的配置情况：

a) 3A-690e 硬件连接为“CPU 串口 + HT1 接南北桥”。路由设置为：

```
/* Route the LPC interrupt to Core0 INT0 , 对应 CP0_Status 的 IP2*/
*(volatile unsigned char*)0x900000003ff0140a = 0x11;

/* Route the HT1 interrupt to Core0 INT1 , 对应 CP0_Status 的 IP3*/
*(volatile unsigned char*)0x900000003ff01418 = 0x21;

*(volatile unsigned char*)0x900000003ff01419 = 0x21;
*(volatile unsigned char*)0x900000003ff0141a = 0x21;
```

```
*(volatile unsigned char*)0x900000003ff0141b = 0x21;
*(volatile unsigned char*)0x900000003ff0141c = 0x21;
*(volatile unsigned char*)0x900000003ff0141d = 0x21;
*(volatile unsigned char*)0x900000003ff0141e = 0x21;
*(volatile unsigned char*)0x900000003ff0141f = 0x21;
```

b) 3A-KD60 硬件连接为“CPU 串口 + CPU 的 PCI + CPU”的 8259，路由设置为：

```
/* Route the LPC interrupt to Core0 INT0 , 对应 CP0_Status 的 IP2*/
*(volatile unsigned char*)0x900000003ff0140a = 0x11;

/* Route the HT1 interrupt to Core0 INT1 , 对应 CP0_Status 的 IP3*/
*(volatile unsigned char*)0x900000003ff01418 = 0x21;

*(volatile unsigned char*)0x900000003ff01419 = 0x21;
*(volatile unsigned char*)0x900000003ff0141a = 0x21;
*(volatile unsigned char*)0x900000003ff0141b = 0x21;
*(volatile unsigned char*)0x900000003ff0141c = 0x21;
*(volatile unsigned char*)0x900000003ff0141d = 0x21;
*(volatile unsigned char*)0x900000003ff0141e = 0x21;
*(volatile unsigned char*)0x900000003ff0141f = 0x21;
```

11.2.2 中断使能

中断相关配置寄存器都是以位的形式对相应的中断线进行控制，中断控制位连接及属性配置见表 11-3 (FIXME: quote the table in chapter 2)。中断使能（Enable）的配置有三个寄存器：Intenset、Intenclr 和 Inten。Intenset 设置中断使能，Intenset 寄存器写 1 的位对应的中断被使能。Intenclr 清除中断使能，Intenclr 寄存器写 1 的位对应的中断被清除。Inten 寄存器读取当前各中断使能的情况。脉冲形式的中断信号（如 PCI_SERR）由 Intedge 配置寄存器来选择，写 1 表示脉冲触发，写 0 表示电平触发。中断处理程序可以通过 Intenclr 的相应位来清除脉冲记录。

不仅需要使能 IO 的控制器，具体到所接的 IO，如果它有自己的中断控制器，也需要单独使能，如 LPC 中断控制器，HT 中断控制器，具体的寄存器配置可查看寄存器手册。下面列出了一些可能需要使能的中断控制器：

```
/* Enable the IO interrupt controller , () LPC10 and (HT16) ~31*/
t = *(volatile unsigned int*)0x900000003ff01428;

*(volatile unsigned int*)0x900000003ff01428 = t | (0xffff << 16) | (0x1 << 10);

/* Enable LPC interrupt controller*/
*(volatile unsigned int*)(0xffffffffbfe00200 + 0x00) = 0x80000000;

/* the 18-bit interrpt enable bit */
*(volatile unsigned int*)(0xffffffffbfe00200 + 0x04) = 0x0;
```

```
/* Enable HT , interruptonly used 7 interrupt vectors*/
*(volatile unsigned int*)0x90000EFDFB0000A0 = 0xffffff7f;
```

11.3 中断分发

中断发生时，硬件会设置 cause 寄存器的 Excode 域及相关的 IP 位。进而进入软件处理过程，软件通过查询 Excode 域来确定是哪一种类型的异常，来选择使用何种异常处理例程。如果是我们要讨论的硬件中断，就会进入平台相关的中断分发函数。中断分发函数再根据 cause 寄存器的 IP 位及 IM 位（中断屏蔽）来进行一级中断分发，然后再根据中断号的位域来进行二次分发，然后执行具体的 do_IRQ 中断操作的处理。

内核在启动阶段，会将每个异常向量与每个异常处理例程对应起来。异常共有 32 种，在这里我们只讨论 0 号异常也就是硬件中断。在 trap_init() 函数中，异常的通用入口地址设置为 0x80000180, 该地址保存了一个函数指针为 expect_vec3_generic，见内核 arch/mips/kernel/-genex.S。expect_vec3_generic() 函数会根据取得的 Excode 码，进入相应的例程函数。在 trap_init() 中异常代码 0 即硬件中断与 handle_int 例程相关，handle_int 见内核 arch/mips/kernel/genex.S。hanle_int 最终跳转到平台相关的 plat_irq_diapatch() 中断分发函数，进行中断的一级级分发。

以 3A-690e 为例，Cause 寄存器的 IP0 和 IP1 对应的是软中断，IP6 对应的是核间中断，IP7 对应的是时钟中断，IP2 对应的是 Cpu 的串口及 LPC 的中断，IP3 对应的是路由到 HT1 的中断。HT1 接北桥 690E，北桥再接南桥 SB600，南北桥的中断都由南桥上的 8259 控制，各个外设如 USB、等的中断路由到 8259 sata 控制器见 arch/mips/kernel/fixup_ev3a.c 中的 godson3a_smbus_fixup 函数，其中寄存器的定义见 AMD 南桥手册之《AMD SB600 Register Reference Manual.pdf》。pcibios_map_irq 函数是对 PCI 及 PCIE 槽位的扫描及中断号分配，文件 fixup_ev3a.c 中其他函数主要是对一些固定 PCI 设备的中断号进行分配，详细的中断分配及到 8259 路由的情况可以详见文件 fixup_ev3a.c 的代码及注释，寄存器使用查看 AMD 的南桥手册。

中断分发完后，运行 do_IRQ() 函数，跳转到对应的具体驱动程序执行。

第十二章　串口配置及使用

12.1　可选择的串口

串口作为一种通信接口，主要用于系统的调试。其工作的原理也就是配置好串口的波特率、数据位、停止位、校验位相关的寄存器，使得串口能按位的发送和接受字节。目前 3A 上可用的 UART 有两类：一类是 CPU 的 UART, 有 UART0 和 UART1，UART0 的串口寄存器的基址是 0xBFE001E0，UART1 的串口寄存器的基址是 0xBFE001E8, 波特率均为 115200；还有一类是 LPC 的 UART，其基址是 0xBFF003F8，波特率为 57600。在设置上，数据位均设置为 8 位，停止位为 1 位，无校验，无流控。

12.2　PMON 的串口配置

在 pmon 中，宏 USE_LPC_UART 是用来区分上述两类串口的。pmon 的设置主要涉及的文件有 start.S 和 tgt_machdep.c。下面以 CPU 的 UART0 为例，首先在 pmon 的 start.S 中有个初始化串口的函数 (寄存器的使用查看 UART 控制器部分):

```
LEAF(initserial)
        li  a0, GS3_UART_BASE
        li  t1, 128
        sb  t1, 3(a0)       // 访问分频寄存器
        li  t1, 0x12        // divider, highest possible baud rate
        sb  t1, 0(a0)       // 分频寄存器，存放分频寄存器的低位，计算公式为18
                            // 工作时钟 (波特率)，本例为 /*1633M/(115200*16)
        li  t1, 0x0         // divider, highest possible baud rate
        sb  t1, 1(a0)       // 分频寄存器，存放分频寄存器的高位28
        li  t1, 3
        sb  t1, 3(a0)       // 数据位为位8

        li  t1, 0
        sb  t1, 1(a0)       // 不使用中断
        li  t1, 71
        sb  t1, 2(a0)       // 设置控制寄存器FIFO
        jr  ra
        nop
END(initserial)
```

GS3_UART_BASE 也是在 start.S 文件中定义的，为 0xBFE001E0。在 tgt_machdep.c 中，结构体 ConfigEntry 也给出了 UART 的设置，函数 ns16550 也就是 UART 的发送和接收的处理函数。

12.3 Linux 内核的串口配置

在 Linux 内核中对串口的配置主要有三个文件：include/asm/serial.h，arch/mips/kernel/8250-platform.c，arch/mips/lemote/ev3a/dbg_io.c。这三个配置文件涉及到串口基址的设定，要根据具体所使用的串口的情况而定。内核中在 arch/mips/Kconfig 里也有一个宏定义 CONFIG_CPU_UART，用来选择是选用 LPC 串口还是 CPU 串口的。纵然选择了 CPU UART，但是具体是 UART0 还是 UART1 还是需要检查一下上述三个文件对串口基址的定义是否正确。在 arch/mips/lemote/ev3a/dbg_io.c 中主要是用于内核启动过程中，在中断还没有初始化阶段，为了内核调试的方便而使用的一种简单的串口打印方式，其中函数 prom_printf() 在内核调试阶段会经常被使用到，它会调用 putDebugChar() 将要输出的字符一个一个的打印到控制台上。include/asm/serial.h 为串口的驱动 driver/serial/8250.c 提供了一个宏定义，如使用 CPU 的 UART0，其定义是这样的：

```
#define STD_SERIAL_PORT_DEFNS
/* UART_CLK PORT_IRQ FLAGS */ \ \
{
  .baud_base = BASE_BAUD,
  .irq = 58,
  .flags = STD_COM_FLAGS,
  .iomem_base = (u8*)(0xFFFFFFFFBFE001E0),
  .io_type = SERIAL_IO_MEM
}
```

所使用的驱动是标准的 8250/16x50 系列的串口驱动。串口的中断号是 58 号，根据采用的是 CPU 的串口还是 LPC 的串口，中断的分发上也略有不同,irq.c 中关于串口中断的部分如下：

```
        ...
        else if (pending & CAUSEF\_IP2)
        { // For LPC
#ifdef CONFIG\_CPU\_UART
                do\_IRQ(58);
#else
                irq = *(volatile unsigned int*)(0xFFFFFFFFBFE00200 + 0x08);
                if((irq & 0x2))
                        do\_IRQ(1);
                if((irq & 0x1000))
                        do\_IRQ(12);
                if((irq & 0x10))
                        do\_IRQ(58);
#endif
```

```
}
```

如果是 CPU 的串口，直接处理 58 号中断，但是如果是 LPC 串口，需要读取 LPC 控制器的相关位域来判断该中断是键盘中断还是鼠标中断还是串口中断，串口中断号依然是 58 号。

第十三章　EJTAG 调试

13.1　EJTAG 介绍

EJTAG(Enhanced JTAG) 是 MIPS 根据 IEEE1149.1 协议的基本构造和功能扩展而定义的规范，是一个硬件/软件子系统，用于支持片上调试。EJTAG 规范通过定义一种新的调试模式，包括专用的指令、运行模式以及地址空间等，与原有的 MIPS 体系结构很好地融合在一起。调试模式的基本思想是采用例外处理的机制，使被调试的软件无法察觉调试器的存在。此外，处理器在调试模式下对地址空间进行了扩展，可以访问调试控制寄存器以及映射到内存区域的调试接口。下图给出了一个常见的 EJTAG 调试系统组成，包括:

- 调试主机 (Debug Host)：运行调试应用程序，控制 EJTAG 线缆
- 目标板 (Target Board)：包含被调试芯片的板卡，提供 EJTAG 接口

图 13-1 EJTAG 调试系统 (FIXME)

调试主机可以通过 EJTAG 线使目标板上的处理器进入调试模式，转向执行调试服务例程。调试服务的入口有两个，分别是位于 BIOS 的 0xbfc00480 和位于 EJTAG 调试内存接口的 0xff200200，由调试主机选择。

EJTAG 规范在处理器的调试模式地址空间中划出两块，映射到调试控制寄存器 (drseg) 和调试内存接口 (dmseg)。当处理器执行调试服务访问调试接口时，所发出的访存请求将阻塞在调试接口中。此时，调试主机能够检测到调试内存接口的访存请求，从而对其进行响应。简单的说，调试服务程序可访问位于调试主机的一段内存空间。

利用调试主机控制的调试用内存空间，调试服务几乎可以完成任意可以想象的功能，因为调试服务程序的代码本身就可以由调试主机提供 (目前龙芯 3A 和 2G 的样片中不能从 dmseg 取指，但可以执行访存指令)。龙芯 3A/2G 中还支持 EJTAG 规范的 PC 采样功能。

13.2　EJTAG 工具使用

13.2.1　环境准备

- EJTAG 线，并口转 14 针 EJTAG 口
- Linux 调试主机，带并口，有 root 权限
- PMON 中添加调试服务程序

13.2.2 PC 采样

在调试主机运行 jtag_1/4/16，将完成对处理器核的一次 PC 采样。其中程序后缀是指 EJTAG 接口中处理器核的数目，一个 2G 是 1，一个 3A 是 4，四个 3A 串一起是 16。PC 采样不需要调试服务程序。3A/2G 中 PC 采样的值不精确，存在抖动，用户需忽略采样结果的低 2 位。处理器核中真正的 PC 指针可能是所采样 PC 对应指令的程序流后方 0 4 条指令，即如果包含转移，则有真正的 PC 可能落在转移目标附近。如果不是有规律的循环或者软件控制的停时钟，PC 采样不应当静止不动。如果出现，则意味着处理器运行不正常。

13.2.3 读写内存

运行 pracc_1/4/16，在参数中指定执行调试服务的处理器核号、访问类型、访问地址、要写的值等。其工作原理是根据所要完成的任务，在调试用内存空间初始化一段参数区，与调试服务程序相配合，控制调试服务的执行。调试服务会首先将若干寄存器保存到调试主机，然后基于这些空出的寄存器运行，在读出相关的参数并执行，最后恢复先前保存过的寄存器。需注意的是，这一功能要求执行调试服务的处理器核还能执行指令。如果工作异常，则可能存在硬件故障。

13.2.4 执行说明

读 bfc00480 | 值的意义

```
cpu@ubuntu:~/ejtag$ ./pracc_4 0 0xffffffffbfc00480 d r
target = 0, addr = ffffffffbfc00480, dword read
breaking...
ctrl = 00049000
stat = 00008008
pracc write, size=3,  address = 000000000000000f, value = 0000000000000000 |t1
pracc write, size=3,  address = 0000000000000017, value = 0000000000000000 |t2
pracc write, size=3,  address = 000000000000001f, value = ffffffff80e1060 |t3
pracc write, size=3,  address = 0000000000000027, value = ffffffff8000b899 |t4
pracc write, size=3,  address = 000000000000002f, value = ffffffffffffffff |t5
pracc write, size=3,  address = 0000000000000037, value = ffffffff80110000 |t6
pracc write, size=3,  address = 000000000000003f, value = ffffffff800f7428 |t7
pracc write, size=3,  address = 0000000000000047, value = 00000000340000e0 |status
pracc write, size=3,  address = 000000000000004f, value = 0000000080034483 |config
pracc write, size=3,  address = 0000000000000057, value = 0000000040008000 |cause
pracc write, size=3,  address = 000000000000005f, value = 0000000000000033 |a0
pracc write, size=3,  address = 0000000000000067, value = ffffffff80120000 |a1
pracc write, size=3,  address = 000000000000006f, value = 0000000000000000 |a2
pracc read, size=3,  address = 0000000000000207, value = fff3ffffbfc00480
pracc write, size=3,  address = 0000000000000107, value = fff3ffffbfc00480
pracc write, size=3,  address = 0000000000000107, value = 0000000000000003
pracc write, size=3,  address = 0000000000000107, value = 00000000000001fc
pracc write, size=3,  address = 0000000000000107, value = 0000000000000001
```

```
pracc write, size=3,  address = 0000000000000107, value = ffffffffbfc00480
pracc write, size=3,  address = 000000000000020f, value = 3c08ff2040a8f800
return 3c08ff2040a8f800                                                    | 读返回值
pracc read, size=3,  address = 0000000000000217, value = 0000000000000000
pracc read, size=3,  address = 000000000000021f, value = 0000000000000000
pracc read, size=3,  address = 000000000000000f, value = 0000000000000000
pracc read, size=3,  address = 0000000000000017, value = 0000000000000000
pracc read, size=3,  address = 000000000000001f, value = ffffffff80e1060
pracc read, size=3,  address = 0000000000000027, value = ffffffff8000b899
pracc read, size=3,  address = 000000000000002f, value = ffffffffffffffff
pracc read, size=3,  address = 0000000000000037, value = ffffffff80110000
pracc read, size=3,  address = 000000000000003f, value = ffffffff800f7428
```

写 bfc00480(其实写不进去)

```
cpu@ubuntu:/home/cpu/ejtag$ ./pracc_4 0 0xffffffffbfc00480 d w 0x0
target = 0, addr = ffffffffbfc00480, dword write with 0000000000000000
press <enter> to confirm..
breaking...
ctrl = 00049000
stat = 60008008
pracc write, size=3,  address = 000000000000000f, value = 0000000000000000
pracc write, size=3,  address = 0000000000000017, value = 0000000000000000
pracc write, size=3,  address = 000000000000001f, value = ffffffff80e1060
pracc write, size=3,  address = 0000000000000027, value = ffffffff8000b899
pracc write, size=3,  address = 000000000000002f, value = ffffffffffffffff
pracc write, size=3,  address = 0000000000000037, value = ffffffff80110000
pracc write, size=3,  address = 000000000000003f, value = ffffffff800f7428
pracc write, size=3,  address = 0000000000000047, value = 00000000340000e0
pracc write, size=3,  address = 000000000000004f, value = 0000000080034483
pracc write, size=3,  address = 0000000000000057, value = 0000000040008000
pracc write, size=3,  address = 000000000000005f, value = ffffffff8ec08c00
pracc write, size=3,  address = 0000000000000067, value = ffffffff8ec08400
pracc write, size=3,  address = 000000000000006f, value = 0000000000000000
pracc read, size=3,  address = 0000000000000207, value = 0000000000000000
pracc write, size=3,  address = 0000000000000107, value = 0000000000000000
pracc read, size=3,  address = 0000000000000217, value = fff3ffffbfc00480
pracc read, size=3,  address = 000000000000021f, value = 0000000000000000
pracc write, size=3,  address = 0000000000000107, value = fff3ffffbfc00480
pracc write, size=3,  address = 0000000000000107, value = 0000000000000003
pracc write, size=3,  address = 0000000000000107, value = 00000000000001fc
pracc write, size=3,  address = 0000000000000107, value = 0000000000000001
pracc write, size=3,  address = 0000000000000107, value = ffffffffbfc00480
```

```
pracc write, size=3,  address = 000000000000021f, value = 0000000000000000
pracc read, size=3,  address = 000000000000000f, value = 0000000000000000
pracc read, size=3,  address = 0000000000000017, value = 0000000000000000
pracc read, size=3,  address = 000000000000001f, value = ffffffff80e1060
pracc read, size=3,  address = 0000000000000027, value = ffffffff8000b899
pracc read, size=3,  address = 000000000000002f, value = ffffffffffffffff
pracc read, size=3,  address = 0000000000000037, value = ffffffff80110000
pracc read, size=3,  address = 000000000000003f, value = ffffffff800f7428
```

调试服务代码注解

```
# start.S
/* Debug exception */
        .align  7                               /* bfc00480 */
////////////////////////////////////////////
#define COP_0_DESAVE $31
    .set mips64
    // save context
    dmtc0   t0, COP_0_DESAVE     // 保存一个寄存器用于指针dmseg
    lui     t0, 0xff20  //
    sd      t1, 0x08(t0)         // 压栈
    sd      t2, 0x10(t0)
    sd      t3, 0x18(t0)
    sd      t4, 0x20(t0)
    sd      t5, 0x28(t0)
    sd      t6, 0x30(t0)
    sd      t7, 0x38(t0)
    dmfc0   t1, COP_0_STATUS_REG          // 输出若干寄存器cp0
    sd      t1, 0x40(t0)
    dmfc0   t1, COP_0_CONFIG
    sd      t1, 0x48(t0)
    dmfc0   t1, COP_0_CAUSE_REG
    sd      t1, 0x50(t0)
    sd      a0, 0x58(t0)         // 其它感兴趣的寄存器
    sd      a1, 0x60(t0)
    sd      a2, 0x68(t0)

#define _t1      9
#define _t2      10
#define _t3      11
#define _t4      12
#define dextu(dest, src, msbd, dlsb) \
    .word ((0x1f<<26)|((src&0x1f)<<21)|((dest&0x1f)<<16)|(( msbd)&0x1f)<<11)|(((dl
```

```
#define dinsu(dest, src, dmsb, dlsb) \
    .word ((0x1f<<26)|((src&0x1f)<<21)|((dest&0x1f)<<16)|(((dmsb)&0x1f)<<11)|(((dls

    // exec_main
    ld      t1, 0x200(t0)       // 读参数 addr/size/count
    beqz    t1, read_end
    sd      t1, 0x100(t0)       // debug...
    dextu   (_t2, _t1, 2-1, 48-32)
    sd      t2, 0x100(t0)       // debug...
    dextu   (_t3, _t1, 9-1, 50-32)
    sd      t3, 0x100(t0)       // debug...
    dextu   (_t4, _t1, 1-1, 47-32)
    sd      t4, 0x100(t0)       // debug...
    dsubu   t4, $0, t4          // sign bit extend
    dinsu   (_t1, _t4, 58-32, 48-32) // address back
    sd      t1, 0x100(t0)       // debug...
    // case t2 0,1,2,3  -> lb,lh,lw,ld
    beqzl   t2, 1f
    lb      t5, 0x0(t1)
    addiu   t2, t2, -1

    beqzl   t2, 1f
    lh      t5, 0x0(t1)
    addiu   t2, t2, -1

    beqzl   t2, 1f
    lw      t5, 0x0(t1)
    addiu   t2, t2, -1

    ld      t5, 0x0(t1)
1:
    sd      t5, 0x208(t0)       // 读返回值
read_end:
    // write
    ld      t1, 0x210(t0)       // 写参数 addr/size/count
    beqz    t1, write_end
    ld      t5, 0x218(t0)       写参数//
    sd      t1, 0x100(t0)       // debug...
    dextu   (_t2, _t1, 2-1, 48-32)
    sd      t2, 0x100(t0)       // debug...
    dextu   (_t3, _t1, 9-1, 50-32)
```

```
    sd      t3, 0x100(t0)       // debug...
    dextu   (_t4, _t1, 1-1, 47-32)
    sd      t4, 0x100(t0)       // debug...
    dsubu   t4, $0, t4          // sign bit extend
    dinsu   (_t1, _t4, 58-32, 48-32) // address back
    sd      t1, 0x100(t0)       // debug...
    // case t2 0,1,2,3  -> sb,sh,sw,sd
    beqzl   t2, 1f
    sb      t5, 0x0(t1)
    addiu   t2, t2, -1

    beqzl   t2, 1f
    sh      t5, 0x0(t1)
    addiu   t2, t2, -1

    beqzl   t2, 1f
    sw      t5, 0x0(t1)
    addiu   t2, t2, -1

    sd      t5, 0x0(t1)
1:
    sd      t5, 0x218(t0)       // 写响应
write_end:

    // restore context
    ld      t1, 0x08(t0)        // 退栈
    ld      t2, 0x10(t0)
    ld      t3, 0x18(t0)
    ld      t4, 0x20(t0)
    ld      t5, 0x28(t0)
    ld      t6, 0x30(t0)
    ld      t7, 0x38(t0)
    dmfc0   t0, COP_0_DESAVE
    deret       // 调试例外返回
```

13.2.5 在线 GDB 调试

由于目前样片存在的 bug，在线调试功能需要在常见的 MIPS 调试平台中作一定量的改动，并增加相应的调试服务程序。该功能尚未实现。

第十四章　地址窗口配置转换

龙芯 3A 采用两级交叉开关结构，两级交叉开关窗口可分别配置，用于控制将特定地址发往特定接收端进行处理。另外，HyperTransport 控制器内部也对芯片对内及对外可访问的地址窗口有所控制。

14.1　一二级交叉开关地址窗口配置方法

交叉开关上每个主端口各拥有 8 个地址窗口可供配置。每个地址窗口由 BASE、MASK 和 MMAP 三个 64 位寄存器组成，BASE 以 K 字节对齐，即每个地址窗口的所分空间最少为 1KB；MASK 为窗口掩码；MMAP 为窗口映射后的地址，同时 [2:0] 表示对应目标从端口的编号，MMAP[4] 表示允许取指，MMAP [5] 表示允许块读，MMAP [7] 表示窗口使能。窗口命中的判断如下：

$$\&MASK == BASE$$

映射后的从端口地址转换公式如下：

$$= \quad \&(\ (MASK))|MMAP\&MASK$$

注意:

- 对于一级交叉开关，MMAP[4] 与 MMAP[5] 必须为 1。而对于二级交叉开关，如果不允许 Cache 访问或取指访问的从端口可以将 MMAP[4] 或 MMAP[5] 设为 0。
- 如果使用一级交叉开关对二级 Cache 地址进行映射，映射后的地址，也即 “从端口地址” 必须与映射前地址，即 “主端口地址” 保持一致: 这是保持 Cache 一致性的需要。而映射到 HyperTransport 的地址及二级交叉开关上的配置不受这个约束。

14.2　一级交叉开关地址窗口

一级交叉开关拥有默认路由设置，这个设置不被地址窗口配置寄存器所显示，只有不在任意一个地址窗口命中的地址会被这个默认路由所解释。对于一级交叉开关的主端口，也即对外发出请求的主设备端，包括如下几个 (端口 4 和 5 被保留)：

主端口号	0	1	2	3	6
设备	处理器核 0	处理器核 1	处理器核 2	处理器核 3	HT 0

对于一级交叉开关的从端口，也即对外发出请求的从设备端，包括如下几个：

优先级	窗口
最高	配置窗口 0
	配置窗口 1
	配置窗口 2
	配置窗口 3
	配置窗口 4
	配置窗口 5
	配置窗口 6
	配置窗口 7
最低	系统默认窗口

表 14.1: 配置窗口路由优先级

起始地址	结束地址	目标
0x0000_0000_0000	0x0BFF_FFFF_FFFF	二级 Cache
0x0C00_0000_0000	0x0DFF_FFFF_FFFF	HyperTransport 0
0x0E00_0000_0000	0x0FFF_FFFF_FFFF	HyperTransport 1

表 14.2: 系统默认窗口

主端口号	0	1	2	3	6	7
设备	二级 Cache 0	二级 Cache 1	二级 Cache 2	二级 Cache 3	HT 0	HT 1

每个主端口的地址窗口配置相互独立，各拥有 8 个可配置窗口。配置窗口优先级依次下降，从配置窗口 0 开始，第一个命中的窗口对这个地址进行路由。优先级次序如下表 14.1： 其中，“系统默认窗口”只有在所有 8 个“配置窗口”都没有对某一地址命中的情况下才会生效。也就是说，在没有对“配置窗口”进行配置前，所有的读写请求都会按照“系统默认窗口”进行路由，有关“系统默认窗口”的说明如下表:

根据 scid_sel 的配置选择根据哪两位映射到不同的四个二级 Cache 中。详见用户手册 2.4 (FIXME) 节。例如，当 scid_sel = 0 时，0x000：路由至二级 Cache 0 0x020：路由至二级 Cache 1 0x040：路由至二级 Cache 2 0x060：路由至二级 Cache 3 当 scid_sel = 2 时，0x000：路由至二级 Cache 0 0x400：路由至二级 Cache 1 0x800：路由至二级 Cache 2 0xc00：路由至二级 Cache 3

14.3 一级交叉开关地址窗口配置时机

一级交叉开关地址窗口配置，对于需要路由至二级 Cache 请求的窗口配置要求是非常严格的，在配置前后需要保证二级 Cache 与一级 Cache 的数据一致性，也就是说，绝不允许在配置后出现如下情况： 配置前二级 Cache 中拥有一个备份，一级 Cache 中也拥有相应的备份，但在配置后有关这个备份地址的请求将被路由到其它的从端口上。 上述的这种情况最终将导致一二级 Cache 数据的错乱。因此，在配置各个窗口的最佳时机是在系统还没有进行 Cache 操作之前。其它的情况下如果需要对一级交叉开关进行配置都必须保证不会出现上述情况。需要注意的是，在进行 Cache 操作之后需改 scid_sel 的值本身也会带来这种不一致，因为地址空间与所映射的二级 Cache 号已经出现了改变。

14.4 二级交叉开关地址窗口

二级交叉开关同样拥有默认路由，所有不被任一地址窗口所命中的地址都会

被路由至从端口 3，也即系统配置寄存器空间上。对于二级交叉开关的主端口，也即对外发出请求的主设备端，包括如下几个：

- 0 号主端口：二级 Cache 0-3
- 1 号主端口：PCI 主端口

对于二级交叉开关的从端口，也即对外发出请求的从设备端，包括如下几个：

- 0 号从端口：内存控制器 0
- 1 号从端口：内存控制器 1
- 2 号从端口：低速设备接口，包括 PCI 从端口、LPC、UART、SPI 接口、PCI 寄存器空间
- 3 号从端口：系统配置寄存器

相对于一级交叉开关，二级交叉开关的地址窗口配置的限制条件会比较弱一些，主要由软件来保证重新配置地址窗口后的访问内容不会出现错误即可。比如，之前将系统地址的 0x0000_0000_0000 – 0x0000_0FFF_FFFF 映射到内存控制器 0 上的 0x0000_0000_0000 – 0x0000_0FFF_FFFF，并使用这个地址进行了一些读写操作，存储了一些有效数据。在对地址窗口进行配置之后，将系统地址的 0x0000_2000_0000 – 0x0000_2FFFF_FFFF 映射到内存控制器 0 的 0x0000_0000_0000 – 0x0000_0FFF_FFFF。此时对 0x0000_2000_0000 的访问会得到原来使用 0x0000_0000_0000 地址存入的值。

在这个过程中，需要注意的是二级 Cache 中的内容并没有根据地址窗口配置的改变而改变，也就是说，如果原来对 0x0000_0000_0000 的写访问采用 Cache 方式，那么很可能在地址窗口更新后对 0x0000_20000_0000 的访问会得到一个旧值。

14.5 对地址窗口配置的特别处理

由于龙芯 3A 处理器核会向外发生一些猜测访问，这些猜测访问可能会落在任意的地址空间。但是，并不是任何设备都允许被猜测访问，尤其是对于 PCI 设备来说，一个猜测的读访问很可能会造成一个“读清”寄存器数据的破坏，也有可能造成一些对非法地址的访问无法正常返回，从而发生处理器死机的情况。我们通过对一二级交叉开关的配置来防止这些情况的发生，将一些不可猜测访问的地址空间禁止掉。例如我们通过对二级交叉开关进行下面的设置来防止对 PCI 空间 0x1000_0000 的猜测访问，但同时允许对 0x1FC0_0000 的猜测访问。

	BASE	MASK	MMAP
窗口 0	0x0000_0000_1000_0000	0xFFFF_FFFF_F000_0000	0x0000_0000_1000_0082
窗口 1	0x0000_0000_1FC0_0000	0xFFFF_FFFF_FFF0_0000	0x0000_0000_1FC0_00F2

对于一级交叉开关来说，对二级 Cache 地址的映射同时也受到 scid_sel 的影响，这两者不可以冲突。如果需要将一个地址空间映射给二级 Cache 时，需要考虑二级 Cache 散列的影响。即将该地址空间映射到各个二级 Cache 上。因为每个地址窗口只可以对应一个从端口，那么对二级

Cache 的映射就需要通过四个地址窗口映射来完成。另外，因为地址窗口的最小单位是 1KB，所以如果此时对二级 Cache 空间进行配置就需要将 scid_sel 的值设为 2 以上，即使用 10 位以上的地址进行散列。如下表的例子，对一级交叉开关进行配置来将处理器核发出的所有地址访问映射至二级 Cache。

scid_sel = 2			
	BASE	MASK	MMAP
窗口 4	0x0000_0000_0000_0000	0x0000_0000_0000_0C00	0x0000_0000_0000_00F0
窗口 5	0x0000_0000_0000_0400	0x0000_0000_0000_0C00	0x0000_0000_0000_04F1
窗口 6	0x0000_0000_0000_0800	0x0000_0000_0000_0C00	0x0000_0000_0000_08F2
窗口 7	0x0000_0000_0000_0C00	0x0000_0000_0000_0C00	0x0000_0000_0000_0CF3

由此窗口可知，凡是没有在窗口 0-3 中命中的地址都将会在这 4 个窗口中命中，并根据 scid_sel 将整个地址空间散列至四个正确的二级 Cache 中。

14.6 HyperTransport 地址窗口

HyperTransport 控制器不仅可以向外发送读写访问，也可以实现外部设备对处理器内部内存的 DMA 访问。这两个访问的访问地址空间相互独立，下面分别介绍。

14.6.1 处理器核对外访问地址窗口

龙芯 3A 芯片共有 4 个 HyperTransport 控制器，分别为 HT0_LO、HT0_HI、HT1_LO、HT1_HI，其中，HT0_LO 与 HT0_HI 共用一个物理接口，HT1_LO 与 HT1_HI 共用一个物理接口，当芯片引脚 HTx_8x2 置为低时，只有 HTx_LO 对用户可见，而 HTx_HI 的地址空间无效。按照一级交叉开关的默认路由，各个控制器的地址空间如下:

基地址	结束地址	大小	定义	说明
0x0C00_0000_0000	0x0CFF_FFFF_FFFF	1 Tbytes	HT0_LO 窗口	
0x0D00_0000_0000	0x0DFF_FFFF_FFFF	1 Tbytes	HT0_HI 窗口	HT0_8x2 =1 时有效
0x0E00_0000_0000	0x0EFF_FFFF_FFFF	1 Tbytes	HT1_LO 窗口	
hline 0x0F00_0000_0000	0x0FFF_FFFF_FFFF	1 Tbytes	HT1_HI 窗口	HT1_8x2 =1 时有效

每个 HyperTransport 控制器内拥有 40 位地址空间，按照 HyperTransport 协议，表 9.12 列出了这 40 位地址的划分。其中 MEM 空间、I/O 空间、HT 总线配置空间分别对应于传统 PCI 空间上的三种访问形式，分别是 PCI MEM 访问、PCI IO 访问与 PCI 配置访问。HT 控制器配置空间主要提供 HT 中断向量及内部窗口配置等功能。HT 总线配置空间，按外部设备的“总线号”“设备号”“功能号”“寄存、、、器偏移”按图 9.1表示的规则对设备配置地址进行直接读写访问。

14.6.2 外部设备对处理器芯片内存 DMA 访问地址窗口

为了保护处理器芯片内的内存数据，HyperTransport 控制器为外部设备的 DMA 访问提供了一组窗口，即用户手册中 9.5.4 节的“接收地址窗口”，只有落在这组窗口中的 DMA 地址才会被

真正对芯片内的内存空间进行操作，否则会给发起该访问的外设做出错误应答。这组窗口由下面的几个参数决定，窗口命中规则如下：

```
hit = ht_rx_image_en &&
        (bus_addr & ht_rx_image_mask ) == (ht_rx_image_base & ht_rx_image_mask)
addr_out = ht_rx_image_trans_en ?
        ht_rx_image_trans | bus_addr & ~ht_rx_image_mask : bus_addr
```

不同的地址窗口优先级依次下降。

14.7 地址空间配置实例分析

下面针对 PMON 里面对两级交叉开关的配置分别予以说明。以下的实例中，HT 设备使用 HT1 接口连接，HT0 接口悬空不用。

14.7.1 一级交叉开关实例 1

一级交叉开关的一种配置如下：

	BASE	MASK	MMAP
窗口 0	0x0000_0000_1800_0000	0xFFFF_FFFF_FC00_0000	0x0000_0EFD_FC00_00F7
窗口 1	0x0000_0000_1000_0000	0xFFFF_FFFF_F800_0000	0x0000_0E00_1000_00F7
窗口 2	0x0000_0000_1E00_0000	0xFFFF_FFFF_FF00_0000	0x0000_0E00_0000_00F7
窗口 3	—	—	
窗口 4	0x0000_0C00_0000_0000	0xFFFF_FC00_0000_0000	0x0000_0C00_0000_00F7
窗口 5	—	—	
窗口 6	0x0000_1000_0000_0000	0x0000_1000_0000_0000	0x0000_1000_0000_00F7
窗口 7	0x0000_2000_0000_0000	0x0000_2000_0000_0000	0x0000_2000_0000_00F7

下面一一分析每个配置窗口的作用。

1. 窗口 0，将 0x1800_0000 的地址转换为 0x0000_0EFD_FC00_0000，并路由至 HT1 控制器。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。

	转换前地址	转换后地址	说明
地址 0	0x0000_0000_18xx_xxxx	0x0000_0EFD_FCxx_xxxx	HT IO 空间
地址 1	0x0000_0000_19xx_xxxx	0x0000_0EFD_FDxx_xxxx	HT IO 空间
地址 2	0x0000_0000_1Axx_xxxx	0x0000_0EFD_FExx_xxxx	HT 配置空间：Type 0
地址 3	0x0000_0000_1Bxx_xxxx	0x0000_0EFD_FFxx_xxxx	HT 配置空间：Type 1

2. 窗口 1，将 0x1000_0000 的地址转换为 0x0000_0E00_1000_0000，并路由至 HT1 控制器。这样实际上将原来需要使用 64 位地址才能访问的 HT MEM 空间的一部分直接使用 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。虽然没有映射全部的 HT MEM 空间，但在这里最多已经可以使用多达 128MB 的 HT MEM 空间了。

	转换前地址	转换后地址	说明
地址 0	0x0000_0000_1xxx_xxxx	0x0000_0E00_1xxx_xxxx	HT MEM 空间

3. 窗口 2，将 0x1E00_0000 的地址转换为 0x0000_0E00_0000_0000，并路由至 HT1 控制器。这样实际上将原来需要使用 64 位地址才能访问的 HT MEM 空间的最低 16MB 地址直接使用 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。之所以要映射这部分地址，是因为一些传统设备需要使用这部保留空间进行固定的译码，例如显卡设备等等。

	转换前地址	转换后地址	说明
地址 0	0x0000_0000_1Exx_xxxx	0x0000_0E00_1Exx_xxxx	HT MEM 空间低 16MB

前面这几个窗口的设置是为了在 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，以方便软件版本的兼容设计，在 linux 系统里面，因为可以直接使用 64 位地址进行访问，所以并不需要这些地址转换。但是需要注意的是，基于 linux 对 HT 的 MEM 空间的处理方式，所以依然需要 HT MEM 空间的转换来简化对 32 位地址寻址的外设的访问。

4. 剩余的窗口用于将没有响应设备的地址全部路由至 HT1 控制器，由 HT1 控制器进行响应。这些地址在正常的程序执行过程中并不会主动出现，但由于处理器核的猜测执行，任何地址的访问都有可能出现，如果得不到正确响应则有可能造成处理器死机。龙芯 3A 中的 HT 控制器可以正确识别并处理此类访问。因此需要将这些潜在的猜测访问全部路由至 HT 控制器。

5. 除了这些地址之外的其它地址都会根据默认路由方法路由至二级 Cache，再向二级交叉开关转发。

14.7.2 一级交叉开关实例 2

一级交叉开关的另一种配置如下 (scid_sel=1)：

	BASE	MASK	MMAP
窗口 0	0x0000_0000_1800_0000	0xFFFF_FFFF_FC00_0000	0x0000_0EFD_FC00_00F7
窗口 1	0x0000_0000_1000_0000	0xFFFF_FFFF_F800_0000	0x0000_0E00_1000_00F7
窗口 2	0x0000_0000_1E00_0000	0xFFFF_FFFF_FF00_0000	0x0000_0E00_0000_00F7
窗口 3	0x0000_0E00_0000_0000	0xFFFF_FE00_0000_0000	0x0000_0E00_0000_00F7
窗口 4	0x0000_0000_0000_0000	0x0000_0000_0000_0C00	0x0000_0000_0000_00F0
窗口 5	0x0000_0000_0000_0400	0x0000_0000_0000_0C00	0x0000_0000_0000_04F1
窗口 6	0x0000_0000_0000_0800	0x0000_0000_0000_0C00	0x0000_0000_0000_08F2
窗口 7	0x0000_0000_0000_0C00	0x0000_0000_0000_0C00	0x0000_0000_0000_0CF3

这种配置情况下，前 3 个窗口与前一种配置相同，这里不再赘述。窗口 3 将 0x0000_0E00_0000_0000 的地址全部路由至 HT1 控制器上，这本是默认的一种路由方式，在这里进行这个配置是因为在窗口 4-7 中，将所有的地址都路由至四个不同的二级 Cache 中，所以默认的路由将不再生效。窗口 4-7 的配置方式也在 1.5 节中已经解释过一遍，其中最重要的地方在于**路由至各个二级 *Cache* 的方式应该与 *scid_sel* 的配置一致**。这种配置的目的同第一种配置相同，都是为了防止一些没有设备响应的地址导致处理器死机。

14.7.3 二级交叉开关实例 1

二级交叉开关的一种配置如下。这种配置下只使用一个内存控制器。使用内存上的 256MB 空间。

	BASE	MASK	MMAP
窗口 0	0x0000_0000_1000_0000	0xFFFF_FFFF_F000_0000	0x0000_0000_1000_0082
窗口 1	0x0000_0000_1FC0_0000	0xFFFF_FFFF_FFF0_0000	0x0000_0000_1FC0_00F2
窗口 2	0x0000_0000_0000_0000	0xFFFF_FFFF_F000_0000	0x0000_0000_0000_00F0

其他的窗口（3-7）都没有打开。

1. 窗口 0 打开了低速设备空间的 uncache 且非取指的访问，这样就可以保证落在这个窗口的访问都是程序需要作出的访问。
2. 窗口 1 打开了低速设备空间中 BOOT 空间的所有类型访问，即包括 cache 访问和取指访问在内的正常访问，猜测访问可以对这个空间进行正常访问。
3. 窗口 2 打开了内存控制器 0 上的低 256MB 空间，允许所有类型的访问。
4. 除此之外的所有访问都将按照默认路由被路由至系统配置寄存器空间。

与 1.8.1 (FIXME) 中的一级交叉开关配置相结合，得到的全芯片地址空间如下:

	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	HT1 MEM 空间
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_0C00_0000_0000	0x0000_0FFF_FFFF_FFFF	HT1 控制器，各种空间
地址 16	0x0000_1000_0000_0000	0x0000_3FFF_FFFF_FFFF	HT1 控制器，猜测空间
地址 17	其它地址		系统配置空间

14.7.4 二级交叉开关实例 2

二级交叉开关的另一种配置如下。这种配置下使用两个内存控制器。每个内存控制器使用 1GB 的内存空间

	BASE	MASK	MMAP
窗口 0	0x0000_0000_1000_0000	0xFFFF_FFFF_F000_0000	0x0000_0000_1000_0082
窗口 1	0x0000_0000_1FC0_0000	0xFFFF_FFFF_FFF0_0000	0x0000_0000_1FC0_00F2
窗口 2	0x0000_0000_0000_0000	0xFFFF_FFFF_F000_0000	0x0000_0000_0000_00F0
窗口 3			
窗口 4	0x0000_0000_8000_0000	0xFFFF_FFFF_C000_0000	0x0000_0000_0000_00F0
窗口 5			
窗口 6	0x0000_0000_C000_0000	0xFFFF_FFFF_C000_0000	0x0000_0000_0000_00F1
窗口 7			

1. 窗口 0 打开了低速设备空间的 uncache 且非取指的访问，这样就可以保证落在这个窗口的访问都是程序需要作出的访问。

2. 窗口 1 打开了低速设备空间中 BOOT 空间的所有类型访问，即包括 cache 访问和取指访问在内的正常访问，猜测访问可以对这个空间进行正常访问。

3. 窗口 2 打开了内存控制器 0 上的低 256MB 空间，允许所有类型的访问。

4. 窗口 4 打开了内存控制器 0 上的所有 1GB 空间，系统使用 0x8000_0000 – 0xBFFF_FFFF 的地址进行访问，需要注意的是，系统 0x8000_0000 – 0x8FFF_FFFF 的空间与 0x0000_0000 – 0x0FFF_FFFF 的空间相重合，为了保证数据的正确性，系统软件必须保证仅使用其中一种地址对这部为进行访问，以 linux 为例，必须使用系统中 0x0000_0000 – 0x0FFF_FFFF 的地址，那么，对于这个空间，0x8000_0000 - 8FFF_FFFF 的访问就是被禁止的。

5. 窗口 6 打开了内存控制器 1 上的所有 1GB 空间，系统使用 0xC0000_0000 – 0xFFFF_FFFF 对这段内存进行访问。

与 1.8.2 (FIXME) 中的一级交叉开关配置相结合，得到的全芯片地址空间分布如下:

	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	HT1 MEM 空间
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_8000_0000	0x0000_0000_BFFF_FFFF	内存控制器 0
地址 16	0x0000_0000_C000_0000	0x0000_0000_FFFF_FFFF	内存控制器 1
地址 17	0x0000_0E00_0000_0000	0x0000_0FFF_FFFF_FFFF	HT1 控制器，各种空间
地址 18	其它地址		系统配置空间

第十五章　系统内存空间分布设计

15.1　系统内存空间

龙芯 3A 处理器内拥有两个内存控制器 (Memory controler, 或 MC)，如果能够将地址空间交错分布在两个内存控制器上，对于系统的平均访问延迟和平均访问带宽都会带来好处，但是受到交叉开关上配置窗口个数的限制，必须采取一定的规则对地址空间分布方法做出一定的设计。基于上述考虑，同时为了维护 linux 系统不同内存空间大小的不同需要，并保证内存空间分布的简单美观，推荐使用下面的规则对内存地址空间进行设计。当然，根据系统的需要，设计者也可以自定义内存空间分布规则。

1. 无论内存大小多大，都必须保证 0x0000_0000 – 0x0FFF_FFFF 的低 256MB 空间;
2. 为了给 IO 设备留出必要的直接访问地址空间，0x1000_0000 – 0x1FFF_FFFF 保留不用作空间地址空间;
3. 1GB 及以上内存空间的剩余部分按照的定义按照下面的公式:

$$\begin{aligned} Base &= Size + 0x1000\_0000 \\ Limit &= 2 * Size - 1, \end{aligned}$$

其中，Size 是所有内存的大小，Base 和 Limit 分别是这块空间的基地址和高地址。

举例说明，如果内存大小为 1GB，则内存在系统中的地址空间如下表:

	起始地址	结束地址	说明
地址 0	0x0000_0000_0000_0000	0x0000_0000_0FFF_FFFF	0 – 256MB
地址 1	0x0000_0000_5000_0000	0x0000_0000_7FFF_FFFF	256MB – 1GB

如果内存大小为 2GB，则内存在系统中的地址空间如下表:

	起始地址	结束地址	说明
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

其他以此类推。

除此之外，为了使两个内存控制器能够交错使用，我们按照以下方式配置二级交叉开关上的内存地址空间。

表 15.1:

说明		窗口号	寄存器设置
使能 BIOS 空间的访问		0	BASE 0x0000_0000_1FC0_0000 MASK 0xFFFF_FFFF_FFF0_0000 MMAP 0x0000_0000_1FC0_00F2
使能 PCI 空间的访问（仅允许非取指的 UNCACHE 访问通过）		1	BASE 0x0000_0000_1000_0000 MASK 0xFFFF_FFFF_F000_0000 MMAP 0x0000_0000_1000_0082
使能低 256M 空间的访问	MC0 单通道 256MB 及以上	2	BASE 0x0000_0000_0000_0000 MASK 0xFFFF_FFFF_F000_0000 MMAP 0x0000_0000_0000_00F0
	双通道 256MBx2 及以上（以地址 [10] 做交错）	2	BASE 0x0000_0000_0000_0000 MASK 0xFFFF_FFFF_F000_0400 MMAP 0x0000_0000_0000_00F0
		3	BASE 0x0000_0000_0000_0400 MASK 0xFFFF_FFFF_F000_0400 MMAP 0x0000_0000_0000_00F1
使能高地址空间的访问	MC0 单通道 512M	4	BASE 0x0000_0000_2000_0000 MASK 0xFFFF_FFFF_F000_0000 MMAP 0x0000_0000_1000_00F0
	MC0 单通道 1G	4	BASE 0x0000_0000_4000_0000 MASK 0xFFFF_FFFF_C000_0000 MMAP 0x0000_0000_0000_00F0
	MC0 单通道 2G	4	BASE 0x0000_0000_8000_0000 MASK 0xFFFF_FFFF_8000_0000 MMAP 0x0000_0000_0000_00F0
	双通道 256M x 2 （使用地址 [10] 交错）	4	BASE 0x0000_0000_2000_0000 MASK 0xFFFF_FFFF_F000_0400 MMAP 0x0000_0000_0000_04F0
		5	BASE 0x0000_0000_2000_0400 MASK 0xFFFF_FFFF_F000_0400 MMAP 0x0000_0000_0000_04F1
	双通道 512M x 2 （使用地址 [10] 交错）	4	BASE 0x0000_0000_4000_0000 MASK 0xFFFF_FFFF_E000_0400 MMAP 0x0000_0000_0000_00F0
		5	BASE 0x0000_0000_4000_0400 MASK 0xFFFF_FFFF_E000_0400 MMAP 0x0000_0000_0000_00F1
		6	BASE 0x0000_0000_6000_0000 MASK 0xFFFF_FFFF_E000_0400 MMAP 0x0000_0000_0000_04F0

表 15.1:

说明		窗口号	寄存器设置
		7	BASE 0x0000_0000_6000_0400 MASK 0xFFFF_FFFF_E000_0400 MMAP 0x0000_0000_0000_04F1
	双通道 1G x 2 （使用地址 [10] 交错）	4	BASE 0x0000_0000_8000_0000 MASK 0xFFFF_FFFF_C000_0400 MMAP 0x0000_0000_0000_00F0
		5	BASE 0x0000_0000_8000_0400 MASK 0xFFFF_FFFF_C000_0400 MMAP 0x0000_0000_0000_00F1
		6	BASE 0x0000_0000_C000_0000 MASK 0xFFFF_FFFF_C000_0400 MMAP 0x0000_0000_0000_04F0
		7	BASE 0x0000_0000_C000_0400 MASK 0xFFFF_FFFF_C000_0400 MMAP 0x0000_0000_0000_04F1
	双通道 2G x 2 （使用地址 [10] 交错）	4	BASE 0x0000_0001_0000_0000 MASK 0xFFFF_FFFF_8000_0400 MMAP 0x0000_0000_0000_00F0
		5	BASE 0x0000_0001_0000_0400 MASK 0xFFFF_FFFF_8000_0400 MMAP 0x0000_0000_0000_00F1
		6	BASE 0x0000_0001_8000_0000 MASK 0xFFFF_FFFF_8000_0400 MMAP 0x0000_0000_0000_04F0
		7	BASE 0x0000_0001_8000_0400 MASK 0xFFFF_FFFF_8000_0400 MMAP 0x0000_0000_0000_04F1

15.2 系统内存空间与外设 DMA 空间映射关系

经过这样的配置之后，我们再来介绍内存地址在 DMA 时的使用。传统的 PCI DMA 空间存在于 0x8000_0000 以上的地址，当设备进行 DMA 操作时，0x8000_0000 的访问被映射到 0x0000_0000 的空间，再与系统内存进行一一的映射。在龙芯 3A 中，为了解决大内存的使用 DMA 空间转换的问题，做出如下规定。

1. 系统内存空间 0x0000_0000 – 0x0FFF_FFFF 在作为 DMA 空间使用时，外设使用 0x8000_0000 – 0x8FFF_FFFF 进行访问；
2. 其它系统内存空间在作为 DMA 空间使用时，总线地址和内存地址一致，无需地址转换，直接使用即可。

以 2GB 内存空间为例，可以得到如下的地址转换表：

	说明		起始地址	结束地址
地址 0	0 – 256MB	系统空间	0x0000_0000_0000	0x0000_0FFF_FFFF
		DMA 空间	0x0000_8000_0000	0x0000_8FFF_FFFF
地址 1	256MB – 2GB	系统空间	0x0000_9000_0000	0x0000_FFFF_FFFF
		DMA 空间	0x0000_9000_0000	0x0000_FFFF_FFFF

使用 HyperTransport 接口时，上面的这种地址转换方法可以通过 HyperTransport 的接收地址窗口配置（FIXME: 参见 XXX 节），使用两组地址窗口来实现：

	说明	窗口使能寄存器	窗口基址寄存器
窗口 0	0 – 256MB	0xC000_0000	0x0080_FFF0
窗口 1	256MB – 2GB	0xC000_0080	0x0080_FF80

这这个设置中，窗口 0 将 0x8000_0000 – 0x8FFF_FFFF 的地址转换为 0x0000_0000 – 0x0FFF_FFFF 的访问。窗口 1 将 0x8000_0000 – 0xFFFF_FFFF 的地址转换为 0x8000_0000 – 0xFFFF_FFFF 的访问。由窗口命中的优先级规则，可以得知，0x8000_0000 – 0x8FFF_FFFF 的地址实际上只会被窗口 0 所映射，那么窗口 1 处理的地址实际上为 0x9000_0000 – 0xFFFF_FFFF。

15.3 系统内存空间的其它映射方法

前面两节介绍的系统内存空间映射方法仅是一种有效的参考方式，系统设计人员也可以根据自己的需要来重新定义内存映射规则。例如将内存空间全部集中在低地址，而将 IO 地址及系统配置空间映射到较高的空间。

第十六章　X 系统的内存分配

X 系统的内存分配问题，描述的就是显卡的显存分配问题。在 3A-690e 中，显卡是集成在北桥 690E 内部，是 PCIE 的一个设备。该显卡的显示核心是 ATI X1250，内部集成了 128M 的显存，也支持共享显存的方式，共享显存最大也可达 128M。显卡的显示过程是这样的：CPU 将有关作图的指令和数据通过 PCIE 总线传送给显卡。GPU 再根据 CPU 的要求，完成图像处理过程，并将最终图像数据保存在显存中。

图 16 1 显卡处理图像显示的过程

对于使用独立显存的情况来说，由于显存是在显卡内部，过程会变得比较简单，显卡可以直接将内容写入显存。而对于共享显存来说，过程会较为复杂一点，GPU 把要写入的显存地址告诉 CPU 之后，会有两种情况：一、显存地址是个 PCI 空间地址。CPU 会把内容再直接写到 PCIE 总线上，PCIE 再做一次地址转换，转换到实际的显存地址上，也就是物理地址上；二、显存地址就是内存地址。这时 CPU 会把要内容直接写入内存中的显存位置。显然，对于共享内存的方式来说，采用第二种方案，效率上会更高。

下面具体讲下共享显存的第二种方式在 PMON 中式如何实现的。为了扩展我们的 PCI 空间，我们使用的 TLB 映射。在 PMON 的 bonito.h 中，我们是这样定义的：

```
#define BONITO_PCILO_BASE                    0x10000000
#define BONITO_PCILO_BASE_VA                 0xD0000000
#define BONITO_PCIIO_BASE                    0x18000000
#define BONITO_PCIIO_BASE_VA                 0xB8000000
```

意思是把 256M 的 PCI 空间（0x10000000 0x20000000）分成的两个部分：0x10000000 0x17ffffff 为 mem 空间，0x18000000 0x20000000 为 IO 空间。而 mem 空间的虚拟地址 0xd0000000 到物理地址 0x10000000 是通过手动填充 TLB 来实现转换的。在 3A-690e 中，如果内存为 2G，显存的 PCI 地址其实就是 0x10000000, 对应的内存地址是 0xf8000000，如果内存为 1G，显存的 PCI 地址其实就是 0x10000000, 对应的内存地址是 0x78000000，这个也是通过 TLB 映射来实现的。TLB 映射的代码如下：

```
    li      t0, 15
    li      t3, 0xf0000000          # ，需要映射的虚拟地址的起始地址entry_hi
    li      a0, 0x3f000000
    bleu    msize, a0, 1f           # 判断内存是1还是，如果是，跳到执行G2G1G1
    nop
    li      t4, 0x0000f000          # 2情况，将转到G0xf00000000xf0000000
    b   2f                          # 跳到执行2
    nop
```

```
1:
    li      t4, 0x00007000          # 1情况，将转到G0xf00000000x70000000
2:
    .set mips64
    dsll    t4, t4, 10
    .set mips3
    ori     t4, t4, 0x1f
    li      t5, (0x1000000>>6)      # 16M ，一页为stride16M
    li      t6, 0x2000000           # VPN2 32M stride
    .set mips64
1:
    dmtc0   t3, COP_0_TLB_HI        # 填表项TLB
    daddu   t3, t3, t6
    dmtc0   t4, COP_0_TLB_LO0
    daddu   t4, t4, t5
    dmtc0   t4, COP_0_TLB_LO1
    daddu   t4, t4, t5
    .set mips3
    addiu   t1, t0, 16
    mtc0    t1, COP_0_TLB_INDEX     # 16MB page
    nop
    nop
    nop
    nop
    nop
    tlbwi
    bnez    t0, 1b                  # 共映射大小16*16=256M
    addiu   t0, t0, -1
```

这样，在显卡访问显存时，我们都可以使用 0xf8000000 的地址来作为显存的起始地址，这样它实际上使用的就是内存的高端部分了。这个是在 rs690_struct.c 中 ati_nb_cfg 的结构体中设置的，把 system_memory_tom_lo 设成 0x1000, 也就是 0x100000000, 如果显存是 128M，那么起始地址就是 0x1000M-128M=0xf8000000 的地址，而这个地址就是上面所说的显存的虚拟地址，根据 TLB 映射可以得到实际的显存物理地址。至此，显卡直接访问显存就是这么实现的。