3A 地址空间文档

guobamantou@126.com

本文档的编写目的：

帮助开发者在几小时内对 3A 的基本框架和地址空间问题有个总体的把握。如果我足够诚实，我将会说前四节就是 3A 用户手册极度阉割版。

本文档知识要求：

读者需要有 mips 相关功底，并且已经阅读过 3A 的用户手册。

除非你是牛叉的或者不可理解的人，或者两者兼而有之，否则都应先满足预备知识的要求。

如果你看 3A 手册已经很清楚了，请瞬间关闭本文档。

当然，对于漂亮女人永远存在例外，如果她允许，我会从头到尾给她介绍十遍，直到她说：我们一块去吃饭吧。

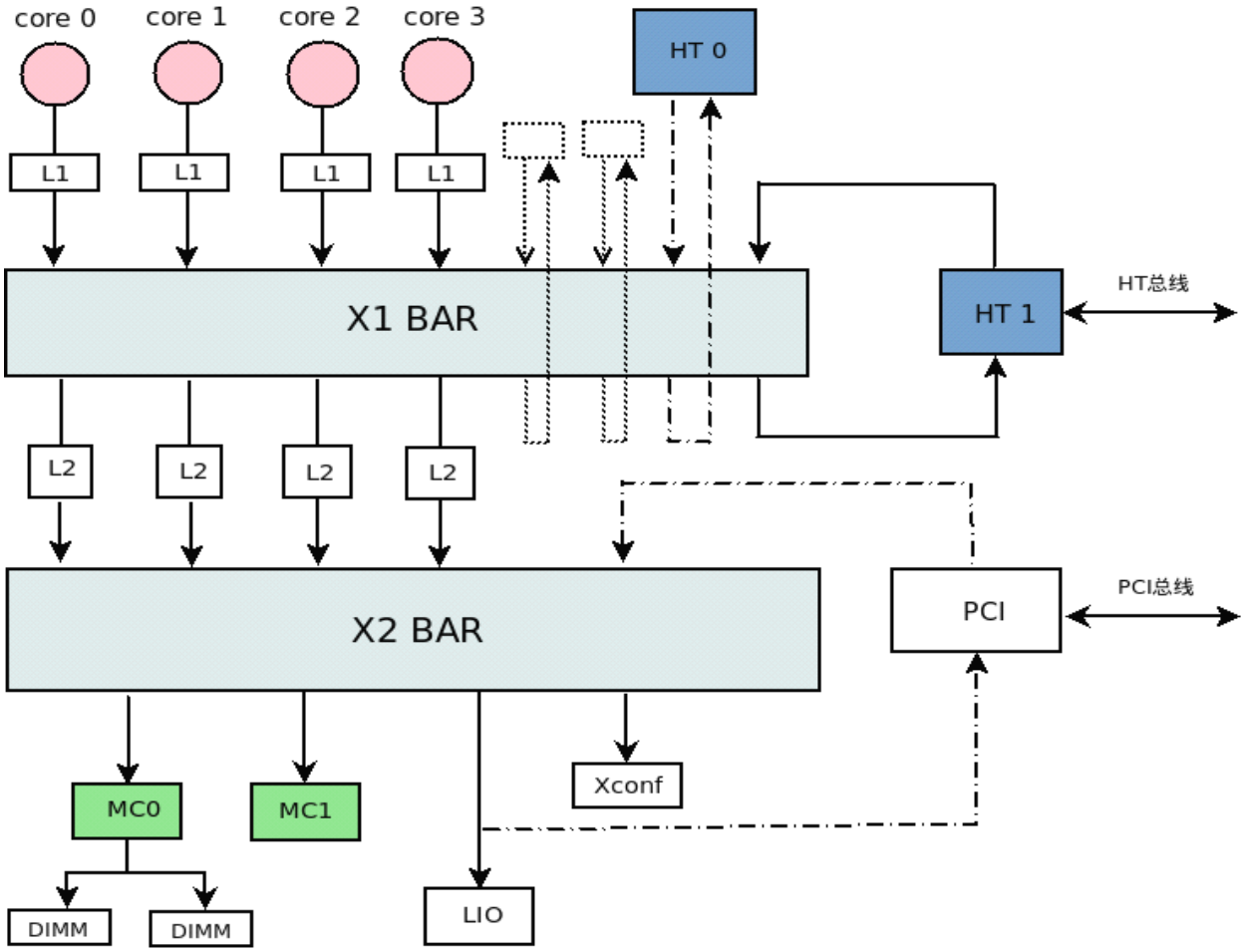
修订记录:

版本	修订内容	修订时间	修订人
V1.8	重写 6.3 一节的内容	2012-04-01	项宇
V1.7	1.增加路由到 HT 的 mmap 的描述 2.增加映射要覆盖全地址的观点	2012-3-6	项宇
V1.6	1.增加 X2 和 HT 配置寄存器描述 2.解释高端内存的物理地址选择	2011-10-22	项宇
V1.5	1. 明确 L2 的路由由 SCID_SEL 确定 2. 加上 3ff 地址路由的描述	2011-10-09	项宇
V1.4	修正 5.1 中的表格描述错误	2011-09-22	项宇
V1.3	增加 mem.h 的头文件修改	2011-6-27	项宇
V1.2	修改 HT mem 的新方案描述	2011-6-13	项宇
V1.1	1.修改 X2 默认路由描述 2.删除映射窗口和性能相关的描述	2011-6-3	项宇
V1.0	初始版本	2011-6-2	项宇

目　录

一、3A CPU 结构简图

先根据图下方的简单描述浏览下这个图，它是整个文档的灵魂。



上图和 3A 手册的有差别，因为我觉得这个图比较符合理解的要求。

图中粉红色（我承认，在色彩喜好上有女性倾向）的是 3A 的四个核。

图中深蓝色的是 HT 控制器模块，3A 有两个 HT 控制器模块。

图中浅蓝色的是地址路由模块，分为 X1 和 X2。

图中绿色的是内存控制器，3A 有两个内存控制器，这是双通道的前提。

图中有两种虚线，密虚线（全部是点）表示连线逻辑上存在，物理上不存在，松虚线（点线结合）表示物理上存在，但是现在没有使用。

二、3A CPU 内部地址资源

对于 X1 来说，发起请求的 master 设备只有 CPU core 和 HT 的 DMA。

3A 的 CPU 物理地址空间为 44 位，范围为 000_0000_0000-fff_ffff_ffff。

具体物理地址分布如下图所示：

目标设备	起始地址	结束地址
L2	000_0000_0000	7ff_ffff_ffff
X1 从端口 4	800_0000_0000	9ff_ffff_ffff
X1 从端口 5	a00_0000_0000	bff_ffff_ffff
HT0	c00_0000_0000	dff_ffff_ffff
HT1	e00_0000_0000	fff_ffff_ffff

其中 X1 的从端口 4，5 其实什么都没有，只有寄存器资源存在，这里不关心。HT0 在现在的两个项目（3A 笔记本，ITX）中都没有使用，也不关心。就 HT1 这个地址需要留意。在代码和映射中我们会遇到大量的以 e 打头的 44 位地址，就是这个空间。

一个 HT 控制器模块可以配置成两个 8 位的 HT 控制器，所以 HT 控制器模块的地址空间实际是再对半划分的，对于 HT1 控制器，就是：

起始地址	结束地址	空间大小	名称
e00_0000_0000	eff_ffff_ffff	1TB	HT1 LO 窗口
f00_0000_0000	fff_ffff_ffff	1TB	HT1 HI 窗口

实际为效率计，HT1 都作为 16 位 HT 控制器使用，所以地址空间 f 打头的 44 位地址实际也不用。

三、HT 内部地址资源

现在知道，实际我们访问的 HT1 地址是从 e00_0000_0000-eff_ffff_fff，大小为 1TB。内部地址空间为 40 位，下图很清楚：

基地址	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

从地址空间映射和路由的理解看，上述空间中比较重要的是 MEM 空间，I/O 空间，HT 总线配置空间，HT 控制器配置空间四个。HT 总线配置空间指的是往总线上发的，和 PCI 中的配置空间概念同，HT 控制器配置空间是 HT 控制器的配置空间，和总线无关。

四、地址路由配置

路由配置主要就是对 X1，X2 的配置，还有就是 HT 控制器内部的一小部分配置。

4.1 X1 的映射

先了解配置 X1 的寄存器资源，X1 有 8 个 master（4 个 CPU，2 个 HT，剩余保留），每个 master 有 8 个窗口，每个窗口由三个寄存器控制，这里篇幅计，只列出 master0 的配置如下表，寄存器都是 3ff020 打头，如下：（master1 的 3ff021 打头，以此类推）

master0 窗口号	BASE	MASK	MMAP
0	3ff0_2000	3ff0_2040	3ff0_2080
1	3ff0_2008	3ff0_2048	3ff0_2088
2	3ff0_2010	3ff0_2050	3ff0_2090
3	3ff0_2018	3ff0_2058	3ff0_2098

4	3ff0_2020	3ff0_2060	3ff0_20a0
5	3ff0_2028	3ff0_2068	3ff0_20a8
6	3ff0_2030	3ff0_2070	3ff0_20b0
7	3ff0_2038	3ff0_2078	3ff0_20b8

base，mask，mmap 都是 64 位寄存器。窗口命中计算方式为：

master 地址 & mask = base

命中后，mmap 会给出路由目标，并按照 mmap 的地址计算映射后的地址。

MMAP[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。从端口对应的内容可以本文第一节架构图中 X1 的引脚排列，作图时是按照这个路由目标号排列的。

3A 的 mask 寄存器没有高位为连续的 1 的要求，现在 X1 和 X2 都没有了这个限制。

实际中 X1 用到的目标路由就两个：L2 和 HT1。

考虑映射的时候永远不要忘了默认映射，X1 的默认路由为：

起始地址	结束地址	目标
0	bff_ffff_ffff	L2
c00_0000_0000	dff_ffff_ffff	HT0
e00_0000_0000	fff_ffff_ffff	HT1

这里说明下默认路由往 L2 发有四个选择，从端口 0 到 3。

路由到四个 L2 的哪个是由 SCID_SEL 寄存器的确定（对于路由到 L2 的哪个口不是由 mmap 的 bit[2:0]确定），实际就是按照地址的某两位进行分布。该寄存器地址为 3ff0_0400。具体见 3A 手册的表 2-4。

现在映射设置的 X1 路由中没有设置到 L2 的路由，全部使用默认映射。

HT 的映射的 mmap 该怎么写有个知识点。

HT 的地址是 40 位的，所以 mmap 的 bit0-bit39 之外的值在 HT 内部无意义。3A 的两个 HT 控制器都可以配成 2 个 8 位或者一个 16 位，由于某些问题的存在，现在都用成 8 位。于是就出现了 HT0 LO 和 HT0 HI 这样，这 LO 和 HI 的区别就是用 bit40 位来区别，所以 mmap 的时候 mmap 的 bit40 还要注意。现在比如 mmap 的值为 c00_0000_00f7，实际和 000_0000_00f7 效果是一样的，写成前者是为了易读性。

4.2 X2 的映射

不同于 X1 每个 master 端口有一组配置寄存器，X2 的配置寄存器只分为两组：CPU 组和 PCI 组。在寄存器资源上，CPU 组有 4 个 master，寄存器也有四组，实际只要设置第一组即可，其余不用改动。

CPU 组寄存器负责处理全部来自 L2 访问请求（不 cache 的地址也过 L2），PCI 组的处理 PCI 作为 master 时候的路由请求（实际现在没有使用）。

下面是 X2 配置用到的寄存器的地址：

CPU 组 窗口号	BASE	MASK	MMAP
0	3ff0_0000	3ff0_0040	3ff0_0080
1	3ff0_0008	3ff0_0048	3ff0_0088
2	3ff0_0010	3ff0_0050	3ff0_0090
3	3ff0_0018	3ff0_0058	3ff0_0098
4	3ff0_0020	3ff0_0060	3ff0_00a0
5	3ff0_0028	3ff0_0068	3ff0_00a8
6	3ff0_0030	3ff0_0070	3ff0_00b0
7	3ff0_0038	3ff0_0078	3ff0_00b8

PCI 组由于都不用，这里不介绍。

先看看 X2 的从端口上都有啥。

从端口 0 是 MC0，从端口 1 是 MC1，从端口 2 是大杂烩 IO，从端口 3 是 Xconf。

两个 MC 实际上没有任何区别，都可以配置为 DDR2 或者 DDR3，如果只有一个 MC 起作用，很明显所有对于内存的访问都应该路由到那个 MC，如果两个 MC 都起作用，也就是每个 MC 上都接着内存，这时如果符合一定的要求，可以配置成双通道以提高同时访问的带宽。具体见“5.2 X2 的映射”一节中关于双通道的描述。

X2 的从端口 2 出来的访问目标是各种 CPU 内置的慢速设备(包括 PCI)，具体见 3A 手册 142 页表格。大致上，这个从端口接收的地址范围从 1c00_0000 到 1fff_ffff，其中用的到的现在就是 1fc0_0000 开始的 1MB 和串口 0 地址 1fe0_01e0-1fe0_01e7 两个。

Xconf 就是 X1 和 X2 的窗口配置模块，地址都是 3ff 打头。

这里又必须提示默认路由的作用。X2 的前两个窗口默认是有默认值的，win0 会把

1fc0_0000 开始的 1MB 映射到从端口 2，win1 会把 1000_0000-1fff_ffff 映射到从端口 2。

如果都没有匹配上，X2 默认路由都将其发往从端口 3，这里明显的例子就是 3ff 打头的一些寄存器，X1 的默认路由将其路由到 L2，X2 中没有任何能匹配的窗口，于是都发往端口 3，就是 Xconf，可见 Xconf 实际上内容不只是 X1 和 X2 的配置。

4.3 HT 的映射

这里说的 HT 映射是指 HT 控制器内部的，HT 连到 X1 后的路由在 X1 中配置。

HT 控制器内部有四个窗口：接收地址窗口、POST 窗口、可预取窗口、UNCACHE 窗口。这些窗口的地址都在从 0xfd_fb00_0000--0xfd_fb00_0100 的范围内。

简单描述下这几个窗口，理解的时候注意数据流的方向。

接收地址窗口：这个范围内的地址访问将被从 HT 发往 CPU，实际就是 DMA 的内存地址范围。

POST 窗口：落在这个窗口中的 HT 命令发到 HT 总线后立即返回，否则等到 HT 命令响应后返回。

可预取窗口：落在这个空间内的取指指令或者 cache 访问才会发往 HT。实际从 HT 取指听起来都很奇怪。

UNCACHE 窗口：窗口之内的访问均不使用 cache，而是直接访存。

可预取窗口和 UNCACHE 窗口很明显的区别就是可预取窗口管理发往 HT 的 cache 访问，而 UNCACHE 窗口管理来自 HT 的 cache 请求。

现在的设计中，接收窗口被设置为：

1. 把 00_8000_0000 到 00_8fff_ffff 的地址转化为 00_0000_0000-00_0fff_ffff，这个窗口大小为 256MB 。

2. 把 00_0000_0000 到 7f_ffff_ffff 的地址转化为 00_0000_0000 到 7f_ffff_ffff，这个窗口大小为 512GB。

实际上第一个规则不起作用，因为在现在的映射中，HT 设备不应该有 8 打头的地址。

在现在的代码中，除了接收地址窗口，其余三个均未启用。这表明：所有 HT 命令都需要等到响应后返回，不允许从 HT 的取指和对 HT 空间的 cache 访问。至于 UNCACHE 窗口不使能，应该表明所有 DMA 操作都会经过 cache。

这里特地尝试了把显存的空间设置到 POST 窗口，但是并没有测试出性能的变化。

五、3A 笔记本的地址映射举例(北京配置)

北京时间是标准，北京的配置就权当参考吧。

现在地址映射都是在 pmon 下完成，X1 配置代码在文件 loongson3_HT_init.S 和 loongson3_fixup.S，X2 配置代码在文件 loongson3_ddr2_config.S。

5.1 X1 的映射

X1 的 8 个 master 分为 4 个 CPU 的，2 个 HT 的，和剩余两个保留。实际现在除了保留的没有使用寄存器为全 0 之外，其余 6 个的配置是一样的。当然这不是必须的。

窗口号	BASE	MMAP	MASK
0	000_1800_0000	efd_fc00_00f7	ffff_ffff_fc00_0000
1	e00_0000_0000	e00_0000_00f7	ffff_ff00_0000_0000
2	0	0	0
3	000_1000_0000	e00_1000_00f7	ffff_ffff_f800_0000
4	000_1e00_0000	e00_0000_00f7	ffff_ffff_ff00_0000
5	c00_0000_0000	c00_0000_00f7	c00_0000_0000
6	2000_0000_0000	2000_0000_00f7	2000_0000_0000
7	1000_0000_0000	1000_0000_00f7	3000_0000_0000

这里 win2 未使用，win5 由于 HT0 没有连实际也没用，win6，win7 超出单片 3A 的地址范围，实际也不用。

win1 实际和默认路由相同（默认路由见”4.1 X1 的映射“一节），这里应该是无意为之，特意问过王焕东，使用默认路由和手动设置路由没有性能差异。

这些映射的路由目标都是 HT1，可见所有到 L2 的路由都由默认路由规则确定。

结合 X1 的设置映射和默认映射，X1 的全映射表如下：

物理地址范围	转化后地址范围	起作用映射	路由目标
0 - 000_0fff_ffff	0- 000_0fff_ffff	默认映射	L2
000_1000_0000 - 000_17ff_ffff	e00_1000_0000- e00_17ff_ffff	X1.win3	HT1 MEM
000_1800_0000 - 000_19ff_ffff	efd_fc00_0000 - efd_fdff_ffff	X1.win0	HT1 IO

000_1a00_0000 - 000_1bff_ffff	efd_fe00_0000 - efd_ffff_ffff	X1.win0	HT1 总线配置空间
000_1c00_0000 - 000_1dff_ffff	000_1c00_0000 - 000_1dff_ffff	默认映射	L2
000_1e00_0000 - 000_1eff_ffff	e00_0000_0000 - e00_00ff_ffff	X1.win4	HT1 MEM
000_1f00_0000 - bff_ffff_ffff	000_1f00_0000 - bff_ffff_ffff	默认映射	L2
c00_0000_0000 - dff_ffff_ffff	c00_0000_0000 - dff_ffff_ffff	X1.win5	HT1
e00_0000_0000 - eff_ffff_ffff	e00_0000_0000 - eff_ffff_ffff	X1.win1	HT1
f00_0000_0000 - fff_ffff_ffff	f00_0000_0000 - fff_ffff_ffff	X1.win5	HT1

上表按照 master 端的地址排序，可以看到 X1.win5 的设置有点让人抓狂。

对于 HT 来说所有的映射都通过 X1 表达，所以只看 X1 就可以得出访问 HT 的地址。

可以这么说，使用默认路由访问 HT 就够了，为什么要设置这么多的窗口来增加访问 HT 的入口呢？主要是 32 位兼容的需要。比如在 pmon（32 位编译）下，c 代码中无法使用 64 位地址，这样要使用默认路由的确没有简单的办法，所以必须要添加 32 位的映射来满足 32 访问的需求。

对于 pmon 下的 HT 访问，必须的需求有 HT IO(X1.win0)，HT MEM(X1.win3)，HT 配置空间(X1.win0)。

至于 X1.win4，这个是否有必要我还不太清楚，虽然有一两处地方用了，稍加修改应该可以取消。

这里有一个问题就是用哪些 32 位地址访问 HT 的资源？

以 HT IO 访问为例，原先 PCI IO 访问的地址是 1fdx_xxxx，那现在 HT IO 范围应该如何确定？1800_0000 是否合适，如果不合适，为什么？是否有更合适的地址呢？

这些在第六节的修改地址映射一节中再讨论。

5.2 X2 的映射

前面已经看到，X1 中没有一个手动设置的映射与 L2 相关，所以到 L2 的路由都使用 X1 的默认映射。

不妨先列出能映射到 L2 的地址：

物理地址范围	转化后地址范围	起作用映射	空间大小

0 - 000_0fff_ffff	0- 000_0fff_ffff	默认映射	256MB
000_1c00_0000 - 000_1dff_ffff	000_1c00_0000 - 000_1dff_ffff	默认映射	32MB
000_1f00_0000 - bff_ffff_ffff	000_1f00_0000 - bff_ffff_ffff	默认映射	约 768GB

能到 X2 的就是上表这三段有空洞的空间。前面已经提到，具体往 4 个 L2 中的哪个路由 SCID_SEL 寄存器确定，实际使用默认散列就行（这个效率的评估模型非常复杂）。

实际我们在 X2 的映射配置中只关心内存的配置，其余都是按照默认来。

假设现在系统上的内存为 1GB，由于 MIPS 架构的要求，分为低 256MB 和高 768MB 两部分。低 256MB 的物理地址（到 X1 时的地址）为 0-0x0fff_ffff，高 768M 的地址具体是多少，如何配置 X2 要看地址映射架构设计者的想法。

这里有两个基本的原则：

1. 物理地址和其它不冲突。

2. 无论如何保证内存都被使用。

第一个原则的不冲突就是比如当前 1fc0_0000（知道点 mips 的都知道这个地址吧）这样的物理地址已经被占用了，内存的物理地址总不能和这些地址重合吧。实际龙芯上 1 打头的很多空间是有特殊用途的，所以 0x1000_0000--0x1fff_ffff 这 256MB 就不要考虑了。现在龙芯 3 号系列中 3ff 打头的很多地址也是有特殊用途的，比如前面 X1，X2 的配置寄存器就是。所以，0x3000_0000 开始的 256MB 还是不要动的好。这些因素是确定的，不确定因素是 PCI（对于 3A 系统实际是 HT 外的 PCIE）的物理空间。原来 2F 的时候惯例是使用 0x4000_0000--0x7fff_ffff。所以个人建议 0x1000_0000--0x8000_0000 这个空间不要作为内存的物理地址，免得冲突。

第二个原则就是内存不能有浪费，虽然在 MIPS 上内存几乎天然就无法物理地址连续，但是在内存控制器上地址当然是应该连续的，否则就是浪费。还是 1GB 的例子，低 256MB 的物理地址是 0--0x0fff_ffff，高端地址我们假设是 0x9000_0000--0xbfff_ffff。X2 的配置就是要保证这两个断开的空间实际映射到物理连续的 1GB 内存介质上。

回忆下 X2 的四个从端口，两个 MC，一个 Xconf，还有个大杂烩 IO，默认往 Xconf。

上述地址中，1c00_0000 开始的 32M 和 1f00_0000 开始的 1M 都应往大杂烩 IO 发。

至于具体实现，单个 MC 起作用的时候很简单，就是保证物理地址不连续的空间在内存控制器上连续，所有对 memory 的请求全部发往这个 MC，X2 中配置的核心问题其实是双通道配置。

实际上，由于体系结构和 32 位兼容的考虑，内存地址必须分为高低两个部分，低端内存固定为 256MB，物理地址从 0_0fff_ffff，高端内存为剩余的部分。无论如何由于高低端物理地址铁定不连续，除非整个内存大小刚好是 512MB，否则前半部分每个 MC 都需要两个地址窗口。如果双通道的总内存容量大于 512MB，必然需要使用 X2 的 6 个地址窗口。

在北京的地址设计中，高端内存的物理地址是没有限定，具体请参见 3A 手册，这里以 2Gx2 的情形解释双通道和高低端内存的关系。

按照北京的设计，2Gx2 内存容量，低 256MB 地址为 0_0fff_ffff，高 3840MB 地址为 1_1000_0000-1_ffff_ffff。

这样，如果块大小为 8KB，要映射成 5.3 节最后两个示意图的那种方式，需要 6 个 X2 窗口配置，如下表：

窗口	BASE	MASK	MMAP
0	0	ffff_ffff_f000_2000	00f0
1	2000	ffff_ffff_f000_2000	00f1
2	0001_0000_0000	ffff_ffff_8000_2000	00f0
3	0001_0000_2000	ffff_ffff_8000_2000	00f1
4	0001_8000_0000	ffff_ffff_8000_2000	20f0
5	0001_8000_2000	ffff_ffff_8000_2000	20f1

这里窗口 0 完成低端内存到 MC0 的映射；

窗口 1 完成低端内存到 MC1 的映射；

窗口 2 完成前半部分内存到 MC0 的映射；

窗口 3 完成前半部分内存到 MC1 的映射；

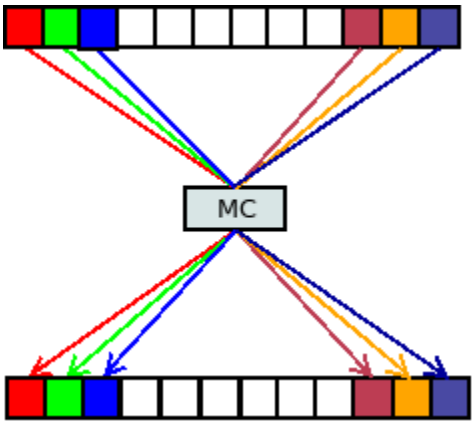
窗口 4 完成后半部分内存到 MC0 的映射；

窗口 5 完成后半部分内存到 MC1 的映射。

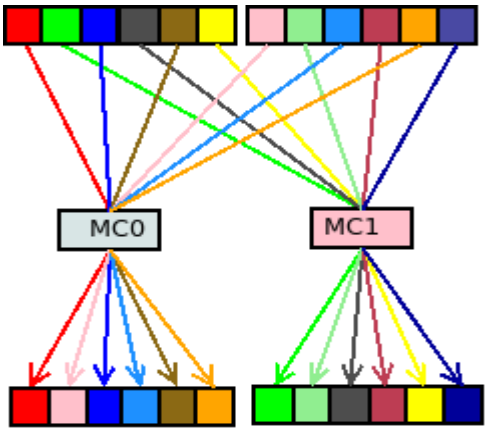
5.2 双通道的配置

双通道有个前提就是两个 MC 上的内存参数一致，比如容量，频率。实际上了解映射之后会明白，内存容量不同也可以双通道，只是配置比较麻烦。而且可能有副作用。

先来两个图对比下单通道和双通道（图中上方表示物理地址，下方表示内存）。



单通道示意图



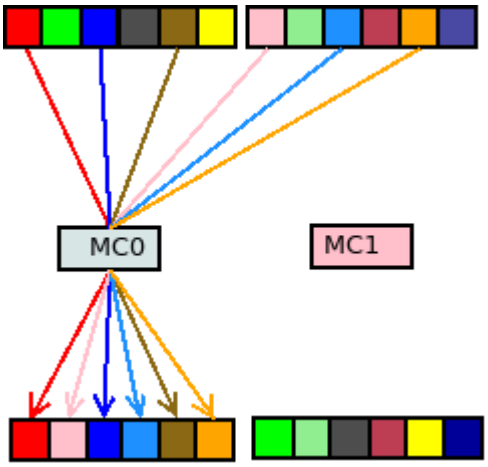
双通道示意图

单通道的物理地址到内存的映射非常直接，可以简单地理解为一一对应。就是物理地址最低部分位于内存起始处，物理地址最大部分位于内存末尾处。这个映射是线性的。

双通道因为有两个 MC，首先遇到的问题就是一个物理地址应该访问哪个 MC，哪个 MC 的哪处内存？这是一个映射策略的问题。

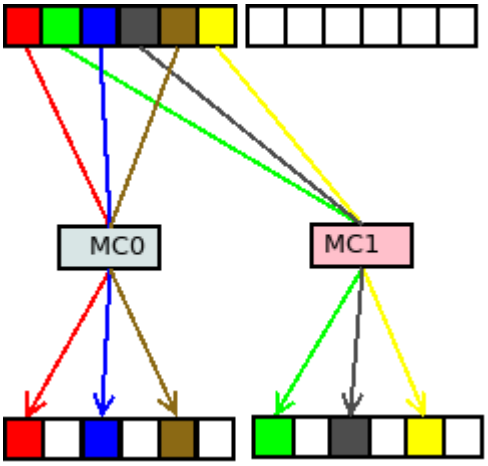
上面双通道示意图就是现在使用的映射策略。具体地说就是物理地址按照一定大小划分为很多块，前后两部分各按照交叉的顺序这些块映射到不同的 MC 中。比如图中物理地址共 12 块，映射到 MC0 的为 1、3、5、7、9、11，到 MC1 的为 2、4、6、8、10、12。

为了看清楚一个 MC 的请求如何往下映射，单看 MC0 映射示意图如下：

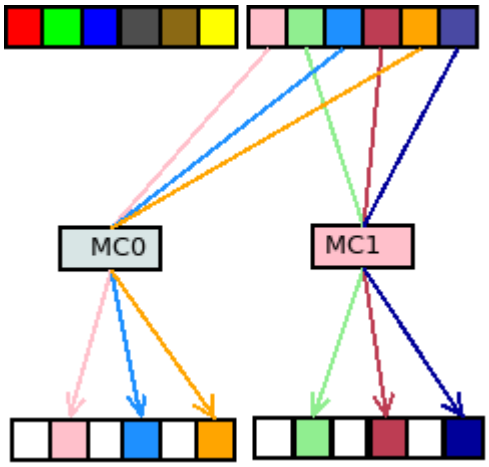


映射的单位为块，用颜色标注。

物理地址映射分为两部分，MC0 映射为映射前位于前半部的第 N 块，在映射后的内存也是第 N 块。后半部映射为映射前位于后半部的第 M 块，映射后的内存的第 M+1 块，MC1 的反之。这要求 X2 的映射也必须分为相等的前后两部分。前后两半地址映射示意图为：



前半部映射效果示意



后半部映射效果示意

如果能思考清楚这个映射的实现，会发现前后两部分需要使用不同的映射窗口。同时到 MC0 和 MC1 的窗口显然是不同的，所以完成上述映射需要 4 个地址映射配置窗口。

六、对北京地址映射机制的修改

6.1 修改映射的理由

世界上没有无缘无故的爱，也没有无缘无故的恨，更没有无缘无故的代码修改。如果非要给这个修改一个理由，那么我有两个理由。

一、32 位的 HT mem 空间太小

现实是 HT 的 mem 空间如果使用 64 位方式访问，的确没有问题，e 打头的地址空间根本用不完，地主家有的是余粮啊。可能是兼容 32 位支持的问题，内核下 HT 的 mem 空间使用的是 32 位的物理地址。北京的设置是 1000_0000-17ff_ffff，刚好 128MB。如果独立显存是 128MB 的，就一点没了。所以，地主家虽有余粮还要主动去挖掘。

二、内存高端地址不固定

这个不固定粗看上去没什么，但是会带来一个麻烦的问题，就是假如扩大 HT 的 mem 空间，那么物理地址范围该是多少。实际很容易和内存高端冲突。如果内存高端地址固定，剩下的就好确定了。

基于上述两个理由，修改映射。

6.2 修改映射的原则

映射有个必须注意的，就是映射完备性。考虑到 CPU 猜测执行的发生，只要是地址范围内的地址，都有可能被访问到，比如 3A 的物理地址是 44 位，那么就要保证这 44 位的地址空间中的任何一个地址访问都有映射（默认或者指定），这一点很重要，否则可能导致奇

怪的死机问题。修改地址映射后建议做一次全地址扫描，就是使用 cache 方式访问整个地址空间，这个时间可能会很长。

映射修改的几个原则。

首先是扩展性，就是在两三年内这个空间应该是没大问题的，比如 pcimem，如果只有 256M，可能就是不够的，1G 短期就够了。

还有就是考虑 32 位的支持，这个现在基本由于 pmon 的存在，都没什么问题。

最后就是映射要尽量方便，在修改的过程中明显是有一定灵活性的，同样的映射有不同的方式实现，简洁最好。

还有，大气，可以适当舍弃不规整的空间，比如 0x30000000 开始的 256MB 因为 3ff 开头是 cpu 用的，这 3 打头的空间就不要用了，显得小气，还容易出错。

6.3 动手

先捋捋思路，看物理地址的使用情况。

[0, 0x1000_000)　这个空间是 32 位支持必须的，低端 256MB 内存。

[0x1000_0000, 0x2000_0000) 部分 cpu 的内部寄存器是 1f 开头的。

[0x2000_0000, 0x3000_0000) free

[0x3000_0000, 0x4000_0000) 部分 cpu 的内部寄存器是 3ff 开头的。

Others　　free

最近半年的（今天是 2012-04-01）的调试中，发现很多设备超过 32 位地址的 DMA 会出现问题，现在我的观点是能用 32 位内的空间就用 32 位内的，不行才退而求其次。

由于对 pmon 的支持，1 打头的 256MB 留给 pmon 的 pcimem 和 pciio，这个不纠缠。

内核下 pci mem 应该放哪里呢，多大合适？

现在 mmio 占用最大的是显存，128M，剩余都小，所以 256M 够了，512M 短期不愁，1G 就是高瞻远瞩了。就高瞻远瞩一把吧，1G。

那物理地址放哪里呢？物理地址 4000_0000-ffff_ffff 的都可以。

有个潜在的问题，就是当用户空间使用 32 位系统时，8000_0000-ffff_ffff 这个空间无法直接使用 mmap，这会导致一系列的应用程序修改，为了兼容这些应用程序，pci 的 mem

空间还是用 4000_0000-7fff_ffff 吧，也就是 1GB 的空间。

0x2000_0000 开始的 512MB 本来蛮好的，就是因为 3ff 开始的 1M 被征用了，搞成鸡肋了。呜呼~~~~~

要使用这样的映射，需要修改 pmon 下的 X1 映射和内核 arch/mips/include/asm/mach-loongson/pci.h 中的常量设置，头文件按修改成如下即可：

#define LOONGSON_PCI_MEM_START 0x40000000ul
#define LOONGSON_PCI_MEM_END 0x7effffful

这里特意预留了最后 16MB 不用于常规的分配，因为有些乱七八糟的设备的地址需求不能通过 pci 的方式表达，但是偶尔确实需要，就留下这最后的 16MB，应该是够了。这部分空间的申请调用 request_mem_region 函数即可。

在用户态使用 mmap 系统调用的时候，会使用到 LOONGSON_MMIO_MEM_END 这个宏，所以在 arch/mips/include/asm/mach-loongson/mem.h 头文件定义中要修改 END 为 0x80000000 即可。

pmon 下的 X1 修改比较清楚，恰好原先 X1 的 win2 没有开启，直接配置为：

BASE	MASK	MMAP
4000_0000	ffff_ffff_4000_0000	e00_4000_00f7

这里 mmap 实际不使用 e00_0000_0000-e00_3fff_ffff，这样没有特别的原因，就是为了地址转换方便，这样映射使得物理地址和总线地址一致。鉴于 HT mem 空间太大，这点浪费就当成浪漫好了。

HT IO 现在的位置也挺蛋疼，但是在没有合适的替代方案时先不去动它。

剩下 0x8000_0000 开始的都给高端内存，超过 32 位的话就超过吧。

有个需要说明的是 0x8000_0000 开始的 256MB，配过双通道的就明白，还是不碰的好，如果不用双通道，这空间得拿下。

好了，就这么定了吧。

高端内存起始在 [9000_0000, 4G+memsize] ，至于高端的为什么是 0x9000_0000 而不是 0x8000_0000，和双通道配置有关。具体请先琢磨 5.3 一节对于双通道原理的描述和图示。

对于不同的内存大小，高端内存的地址范围具体如下表：

内存容量	高端起始地址	高端结束地址
512MB	9000_0000	9fff_ffff
1GB	9000_0000	bfff_ffff
2GB	9000_0000	ffff_ffff
4GB	9000_0000	1_7fff_ffff
8GB	9000_0000	2_7fff_ffff

内存要按照如上方式映射，需要修改 pmon 下 X2 的映射，还有内核中的 arch/mips/loongson/common/mem.c 中的 prom_init_memory 函数即可。

6.4 内存物理地址的 X2 映射

内存物理地址的映射要蛮小心的，所以特地废话下。

6.4.1 支持的搭配和内存的物理地址范围

单根(512M 到 4G 任选一根)

双根(512M 到 4G 任选两根)

512M+512M, 512M+1G, 512M+2G, 512M+4G，

1G+1G, 1G+2G, 1G+4G,

2G+2G, 2G+4G

4G+4G

对于两根大小不同的内存，不配置双通道。

假设 MC0 上为 xMB，MC1 上为 yMB（x 和 y 是 512, 1024, 2048, 4096 之一），则内存物理地址为[0, 256M) 和 [2G+256M, (x+y)M+2G)。

6.4.2 物理地址到物理内存的映射

X2 有 8 组地址窗口寄存器，一组用于到 1fc00000 的映射，一组用于剩余 1 打头的映射，弄两个是出于性能考虑，前者 cache，后者不 cache，所以现在能用的窗口还有 6 组。

下面的地址映射都有个特点就是[2G, 2G+256M)的映射和[0, 256M)重合。也就是 0x8000_0000 开始的 256M 不用的原因。

单通道单根配置

非常直接，大部分只要两个映射即可，低端 256M 一个，高端一个。如果物理地址超过 4G，要多一个映射。下面以内存在 MC0 为例，看下如何映射。

512MB 物理地址为[0, 256M)和 [0x9000_0000, 0xa000_0000)，映射配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
3	0x8000_0000	0xffff_ffff_e000_0000	0xf0	0x9000_0000-- 0xa000_0000

1GB 物理地址为[0, 256M), [0x9000_0000, 0xc000_0000)，映射配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
3	0x8000_0000	0xffff_ffff_c000_0000	0xf0	0x9000_0000-- 0xc000_0000

2GB 物理地址为[0, 256M)和 [0x9000_0000, 0x1_0000_0000)，映射配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
3	0x8000_0000	0xffff_ffff_8000_0000	0xf0	0x9000_0000-- 0x1_0000_0000

4GB 物理地址为[0, 256M), [0x9000_0000, 0x1_8000_0000)，映射配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
3	0x8000_0000	0xffff_ffff_8000_0000	0xf0	0x9000_0000-- 0x1_0000_0000
4	0x1_0000_0000	0xffff_ffff_8000_0000	0x800000f0	0x1_0000_0000-- 0x1_8000_0000

可以看到除了 4GB 的由于大于 2G，用到了 32 位之外的空间从而多了一个映射，其余都是 2 个，而且只有第二个映射的 mask 寄存器不同。

双通道双根配置

双通道的前两个映射是一样的，这里是用 8KB 作为一个块来间隔存取。

4GBx2 这个太特殊了，地址是低 256M 和 0x9000-0000 到 0x2_8000_0000，中间值为 0x1_8000_0000，按照前面的映射方式，必须只用两组就弄出大于 0x1_8000_0000 部分的映射，我想了很久，没找到一个既保持高端地址连续，使用低 32 位的 2G 空间，而且能用 6 个映射表达的。这个 4GBx2 的双通道暂缺，代码实现的时候先弄成两个各自映射，需要 4 组映射，分别映射[0, 256M)，[2G+256M，4G)，[4G，8G)， [8G，10G)。

512Mx2 的地址是[0, 256M)和[2G+256M, 3G)，配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_2000	0xf0	0-- 0x1000_0000
3	0x2000	0xffff_ffff_f000_2000	0xf1	0-- 0x1000_0000
4	0x8000_0000	0xffff_ffff_e000_2000	0xf0	0x9000_0000-- 0xa000_0000
5	0x8000_2000	0xffff_ffff_e000_2000	0xf1	0x9000_0000-- 0xa000_0000
6	0xa000_0000	0xffff_ffff_e000_2000	0x20f0	0xa00_0000-- 0xc000_0000
7	0xa000_2000	0xffff_ffff_e000_2000	0x20f1	0xa000_0000-- 0xc000_0000

1GBx2 的地址是[0, 256M)和[2G+256M, 4G)，配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_2000	0xf0	0-- 0x1000_0000
3	0x2000	0xffff_ffff_f000_2000	0xf1	0-- 0x1000_0000
4	0x8000_0000	0xffff_ffff_c000_2000	0xf0	0x9000_0000-- 0xc000_0000
5	0x8000_2000	0xffff_ffff_c000_2000	0xf1	0x9000_0000-- 0xc000_0000
6	0xc000_0000	0xffff_ffff_c000_2000	0x20f0	0xc000_0000-- 0x1_0000_0000
7	0xc000_2000	0xffff_ffff_c000_2000	0x20f1	0xc0000_0000-- 0x1_0000_0000

2GBx2 的地址是[0, 256M)和[2G+256M, 6G)，配置如下：

Win	Base	Mask	mmap	处理地址
2	0	0xffff_ffff_f000_2000	0xf0	0-- 0x1000_0000
3	0x2000	0xffff_ffff_f000_2000	0xf1	0-- 0x1000_0000
4	0x8000_0000	0xffff_ffff_8000_2000	0xf0	0x9000_0000-- 0x1_0000_0000
5	0x8000_2000	0xffff_ffff_8000_2000	0xf1	0x9000_0000-- 0x1_0000_0000
6	0x1_0000_0000	0xffff_ffff_8000_2000	0x20f0	0x1_0000_0000-- 0x1_8000_0000
7	0x1_0000_2000	0xffff_ffff_8000_2000	0x20f1	0x1_0000_0000-- 0x1_8000_0000

单通道双根配置

下面假设 MC0 上的内存小于等于 MC1 上的。

512M+1G 的地址是[0, 256M)和[2G+256M, 3G+512M)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_e000_0000	0xf0	0x9000_0000 0xa000_0000
1	4	0xa000_0000	0xffff_ffff_e000_0000	0xf1	0xa000_0000- 0xc000_0000
1	5	0xc000_0000	0xffff_ffff_e000_0000	0x200000f1	0xc000_0000- 0xe000_0000

512M+2G 的地址是[0, 256M)和[2G+256M, 4G+512M)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_e000_0000	0xf0	0x9000_0000 0xa000_0000
1	4	0xa000_0000	0xffff_ffff_e000_0000	0xf1	0xa000_0000- 0xc000_0000
1	5	0xc000_0000	0xffff_ffff_c000_0000	0x200000f1	0xc000_0000- 0x100000000
1	6	0x1_0000_0000	0xffff_ffff_e000_0000	0x600000f1	0x100000000- 0x120000000

512M+4G 的地址是[0, 256M)和[2G+256M, 6G+512M)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_e000_0000	0xf0	0x9000_0000 0xa000_0000
1	4	0xa000_0000	0xffff_ffff_e000_0000	0xf1	0xa000_0000- 0xc000_0000
1	5	0xc000_0000	0xffff_ffff_c000_0000	0x200000f1	0xc000_0000- 0x100000000
1	6	0x1_0000_0000	0xffff_ffff_8000_0000	0x600000f1	0x100000000- 0x180000000
1	7	0x1_8000_0000	0xffff_ffff_e000_0000	0xe00000f1	0x180000000- 0x1a0000000

1G+2G 的地址是[0, 256M)和[2G+256M, 5G)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_c000_0000	0xf0	0x9000_0000 0xc000_0000
1	4	0xc000_0000	0xffff_ffff_c000_0000	0xf1	0xc000_0000- 0x100000000
1	5	0x1_0000_0000	0xffff_ffff_c000_0000	0x400000f1	0x100000000- 0x140000000

1G+4G 的地址是[0, 256M)和[2G+256M, 7G)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_c000_0000	0xf0	0x9000_0000 0xc000_0000
1	4	0xc000_0000	0xffff_ffff_c000_0000	0xf1	0xc000_0000- 0x100000000
1	5	0x1_0000_0000	0xffff_ffff_8000_0000	0x400000f1	0x100000000- 0x180000000
1	6	0x1_8000_0000	0xffff_ffff_c000_0000	0xc00000f1	0x180000000- 0x1c0000000

2G+4G 的地址是[0, 256M)和[2G+256M, 8G)，映射为：

MC	Win	Base	Mask	mmap	处理地址
0	2	0	0xffff_ffff_f000_0000	0xf0	0-- 0x1000_0000
0	3	0x8000_0000	0xffff_ffff_8000_0000	0xf0	0x9000_0000 0x100000000
1	4	0x1_0000_0000	0xffff_ffff_0000_0000	0xf1	0x100000000- 0x200000000

好了，就是这些。