龙芯中科
LOONGSON TECHNOLOGY

LOONGSON

# 龙芯 2G 处理器用户手册

## 上册

### 多核处理器架构、寄存器描述与系统软件编程指南

*2012 年 3 月*

*中国科学院计算技术研究所*

*龙芯中科技术有限公司*

## 版权声明



## 免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

## 龙芯中科技术有限公司

Loongson Technology Corporation Limited
地址：北京市海淀区中关村科学院南路 10 号
No.10 Kexueyuan South Road, Zhongguancun Haidian District, Beijing
电话(Tel)：010-62546668
传真(Fax)：010-62600826

# 阅读指南

本手册分为两部分，第一部分（第 1 章~第 10 章）介绍龙芯 2G 多核处理器架构与寄存器描述，对芯片系统架构、主要模块的功能与配置、寄存器列表及位域进行详细说明；第二部分（第 11 章~第 16 章）是系统软件编程指南，对 BIOS 和操作系统开发过程中的常见问题进行专题介绍。

关于龙芯 2G 多核芯片所集成的 GS464 高性能处理器核的相关资料，请参阅《龙芯 GS464 处理器核用户手册》。

# 修订历史

| 文档更新记录 | 文档编号: | |
|---|---|---|
| | 文档名: | 龙芯 2G 处理器用户手册 |
| | 版本号 | V1.0 |
| | 创建人: | 研发中心 |
| | 创建日期 ： | 2012-03-18 |
| | | |

## 更新历史

| 序号. | 更新日期 | 更新人 | 版本号 | 更新内容 |
|---|---|---|---|---|
| 1 | 2012-3-18 | 研发中心 | V1.0 | 手册初版 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# 目 录

# 图目录

# 表目录

# 第一部分

## 多核处理器架构、寄存器描述

# 1 概述

龙芯2G是一个3-4核的处理器，采用65nm工艺制造，最高工作频率为1GHz，主要技术特征如下：

- 片内集成 3-4 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，每个体模块容量为 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 1 个 16 位 800MHz 的 HyperTransport 控制器；
- 片内集成 1 个 LPC、2 个 UART、1 个 SPI、16 路 GPIO 接口；

龙芯 2G 芯片整体架构基于两级互连实现，结构如图 1-1 所示。

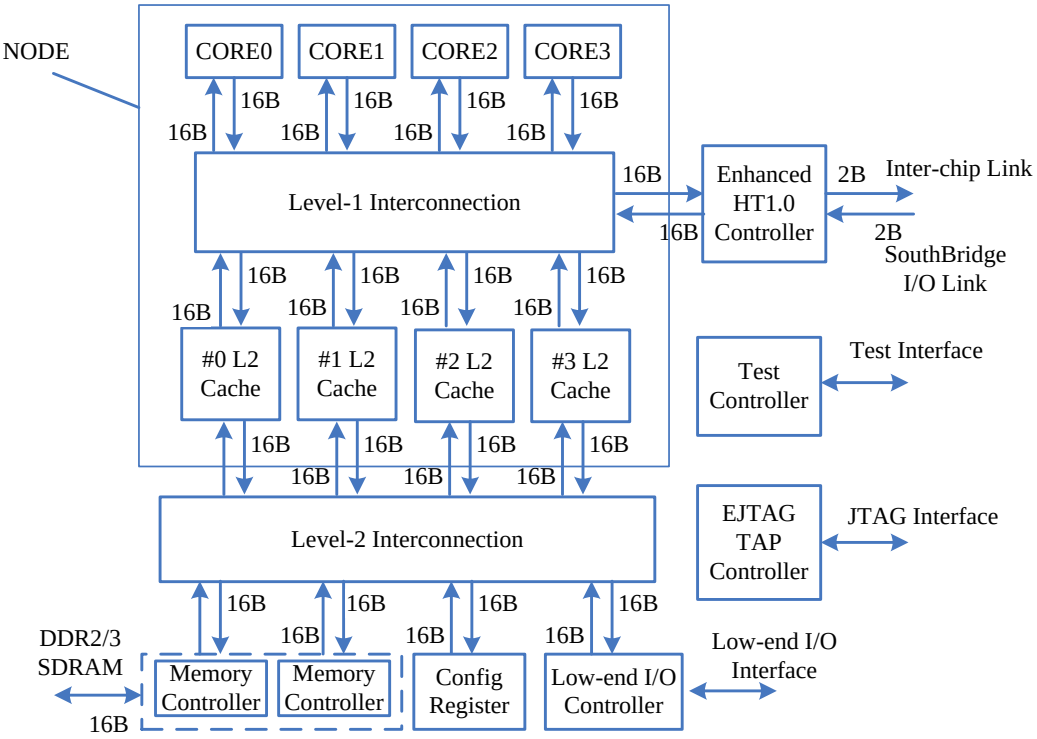



图 1-1 龙芯 2G 芯片结构

第一层互连采用 5x5 的交叉开关，用于连接最多四个 CPU（作为主设备）、四个二级 Cache 模块（作为从设备）、以及 IO 端口（每个端口使用一个 Master 和一个 Slave）。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。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。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 4x4 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 LPC、SPI、UART 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 128 位，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。

# 2 系统配置与控制

## 2.1 控制引脚说明

龙芯 2G 的控制引脚包括 DO_TEST、CLKSEL[15:0]、SYS_CONFIG。其中 CLKSEL[9:5]固定为 5’b11111，DDR 频率的修改需要通过软件对全芯片配置寄存器（0xbfe00180）进行配置。

表 2-1 控制引脚说明

<table>
<tr><th>信号</th><th>上下拉</th><th>作用</th></tr>
<tr><td>DO_TEST</td><td>上拉</td><td>1’b1 表示功能模式<br>1’b0 表示测试模式</td></tr>
<tr><td>CLKSEL[15:0]</td><td></td><td>上电时钟控制<br>HT 时钟控制
<table>
<tr><th>信号</th><th>作用</th></tr>
<tr><td>CLKSEL[15]</td><td>1’b1 表示采用内部参考电压<br>1’b0 表示采用外部参考电压</td></tr>
<tr><td>CLKSEL[14]</td><td>1’b1 表示 HT PLL 采用差分时钟输入<br>1’b0 表示 HT PLL 采用普通时钟输入</td></tr>
<tr><td>CLKSEL[13:12]</td><td>2’b00 表示 PHY 时钟为 1.6GHZ/1<br>2’b01 表示 PHY 时钟为 3.2GHZ/2<br>2’b10 表示 PHY 时钟为普通输入时钟<br>2’b11 表示 PHY 时钟为差分输入时钟</td></tr>
<tr><td>CLKSEL[11:10]</td><td>2’b00 表示 HT 控制器时钟 200MHz<br>2’b01 表示 HT 控制器时钟 400MHz<br>2’b1x 表示 HT 控制器时钟为普通输入时钟</td></tr>
</table>
MEM 时钟控制
<table>
<tr><th>信号</th><th>作用</th></tr>
<tr><td>CLKSEL[9:5]</td><td>内部信号，未引出，需要通过软件对实际频率进行设置。<br>5’b11111 表示 MEM 时钟直接采用 memclk，<br>其它情况下 MEM 时钟为 memclk*(clksel[8:5]+30)/(clksel[9]+3)</td></tr>
</table>
CORE 时钟控制</td></tr>
</table>

| 信号 | 作用 |
|---|---|
| CLKSEL[4:0] | 5'b11111 表示 CORE 时钟直接采用 sysclk<br>其它情况下 CORE 时钟为 sysclk*(clksel[3:0] 30)/(clksel[4]+1) |

SYS_CONFIG[7:0]

IO 配置控制
7 HT 控制信号引脚电压控制位 1*
0 HT 控制信号引脚电压控制位 0*
注*:

| 7 | 0 | HT 控制信号引脚电压，这些信号包括 HT_Mode，HT_Powerok，HT_Rstn，HT_Ldt_Stopn，HT_Ldt_Reqn。 |
|---|---|---|
| 0 | 0 | 1.8v |
| 0 | 1 | Reserved |
| 1 | 0 | 2.5v |
| 1 | 1 | 3.3v |

## 2.2 Cache 一致性

龙芯 2G 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。

## 2.3 系统节点级的物理地址空间分布

龙芯多核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 2G 采用单节点最多 4 核配置，因此龙芯 2G 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（含）至 0x1000_0000_0000（不含）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。

表 2-2 节点级的系统全局地址分布

| 节点号 | 地址[47:44]位 | 起始地址 | 结束地址 |
|---|---|---|---|
| 0 | 0 | 0x0000_0000_0000 | 0x1000_0000_0000 |
| 1 | 1 | 0x1000_0000_0000 | 0x2000_0000_0000 |
| 2 | 2 | 0x2000_0000_0000 | 0x3000_0000_0000 |
| 3 | 3 | 0x3000_0000_0000 | 0x4000_0000_0000 |
| 4 | 4 | 0x4000_0000_0000 | 0x5000_0000_0000 |

| | | | |
|---|---|---|---|
| 5 | 5 | 0x5000_0000_0000 | 0x6000_0000_0000 |
| 6 | 6 | 0x6000_0000_0000 | 0x7000_0000_0000 |
| 7 | 7 | 0x7000_0000_0000 | 0x8000_0000_0000 |
| 8 | 8 | 0x8000_0000_0000 | 0x9000_0000_0000 |
| 9 | 9 | 0x9000_0000_0000 | 0xa000_0000_0000 |
| 10 | 0xa | 0xa000_0000_0000 | 0xb000_0000_0000 |
| 11 | 0xb | 0xb000_0000_0000 | 0xc000_0000_0000 |
| 12 | 0xc | 0xc000_0000_0000 | 0xd000_0000_0000 |
| 13 | 0xd | 0xd000_0000_0000 | 0xe000_0000_0000 |
| 14 | 0xe | 0xe000_0000_0000 | 0xf000_0000_0000 |
| 15 | 0xf | 0xf000_0000_0000 | 0x1_0000_0000_0000 |

在每个节点的内部，44 位地址空间又进一步均匀分布给结点内连接的可能最多 8 个设备。其中低 43 位地址由 4 个 2 级 Cache 模块所拥有，[43:41] = 3'b111 是 HT 的地址空间，其它地址为非法空间，不允许软件访问。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。

表 2-3 节点内的地址分布

| 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 | 0x800_0000_0000 |
| 东 | 4 | 0x800_0000_0000 | 0xa00_0000_0000 |
| 南 | 5 | 0xa00_0000_0000 | 0xc00_0000_0000 |
| 西 | 6 | 0xc00_0000_0000 | 0xe00_0000_0000 |
| 北 | 7 | 0xe00_0000_0000 | 0x1000_0000_0000 |

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 2G 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，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，如表 2-4 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。

表 2-4 节点内的地址分布

| 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 |

## 2.4 地址路由分布与配置

龙芯 2G 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 Master 端口接收到的请求进行路由配置，每个 Master 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 Slave 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。

窗口命中公式: (IN_ADDR & MASK) == BASE

由于龙芯 2G 缺省采用固定路由，在上电启动时，配置窗口都处于未使能状态，所有访问使用默认地址路由，在软件需要使用时可以对其进行使能配置。

地址窗口转换寄存器如下表所示。

表 2-5 一级交叉开关地址窗口寄存器表

| 地址 | 寄存器 | 地址 | 寄存器 |
|---|---|---|---|
| 0x3ff0_2000 | CORE0_WIN0_BASE | 0x3ff0_2100 | CORE1_WIN0_BASE |
| 0x3ff0_2008 | CORE0_WIN1_BASE | 0x3ff0_2108 | CORE1_WIN1_BASE |
| 0x3ff0_2010 | CORE0_WIN2_BASE | 0x3ff0_2110 | CORE1_WIN2_BASE |
| 0x3ff0_2018 | CORE0_WIN3_BASE | 0x3ff0_2118 | CORE1_WIN3_BASE |
| 0x3ff0_2020 | CORE0_WIN4_BASE | 0x3ff0_2120 | CORE1_WIN4_BASE |
| 0x3ff0_2028 | CORE0_WIN5_BASE | 0x3ff0_2128 | CORE1_WIN5_BASE |
| 0x3ff0_2030 | CORE0_WIN6_BASE | 0x3ff0_2130 | CORE1_WIN6_BASE |
| 0x3ff0_2038 | CORE0_WIN7_BASE | 0x3ff0_2138 | CORE1_WIN7_BASE |

| | | | |
|---|---|---|---|
| 0x3ff0_2040 | CORE0_WIN0_MASK | 0x3ff0_2140 | CORE1_WIN0_MASK |
| 0x3ff0_2048 | CORE0_WIN1_MASK | 0x3ff0_2148 | CORE1_WIN1_MASK |
| 0x3ff0_2050 | CORE0_WIN2_MASK | 0x3ff0_2150 | CORE1_WIN2_MASK |
| 0x3ff0_2058 | CORE0_WIN3_MASK | 0x3ff0_2158 | CORE1_WIN3_MASK |
| 0x3ff0_2060 | CORE0_WIN4_MASK | 0x3ff0_2160 | CORE1_WIN4_MASK |
| 0x3ff0_2068 | CORE0_WIN5_MASK | 0x3ff0_2168 | CORE1_WIN5_MASK |
| 0x3ff0_2070 | CORE0_WIN6_MASK | 0x3ff0_2170 | CORE1_WIN6_MASK |
| 0x3ff0_2078 | CORE0_WIN7_MASK | 0x3ff0_2178 | CORE1_WIN7_MASK |
| 0x3ff0_2080 | CORE0_WIN0_MMAP | 0x3ff0_2180 | CORE1_WIN0_MMAP |
| 0x3ff0_2088 | CORE0_WIN1_MMAP | 0x3ff0_2188 | CORE1_WIN1_MMAP |
| 0x3ff0_2090 | CORE0_WIN2_MMAP | 0x3ff0_2190 | CORE1_WIN2_MMAP |
| 0x3ff0_2098 | CORE0_WIN3_MMAP | 0x3ff0_2198 | CORE1_WIN3_MMAP |
| 0x3ff0_20a0 | CORE0_WIN4_MMAP | 0x3ff0_21a0 | CORE1_WIN4_MMAP |
| 0x3ff0_20a8 | CORE0_WIN5_MMAP | 0x3ff0_21a8 | CORE1_WIN5_MMAP |
| 0x3ff0_20b0 | CORE0_WIN6_MMAP | 0x3ff0_21b0 | CORE1_WIN6_MMAP |
| 0x3ff0_20b8 | CORE0_WIN7_MMAP | 0x3ff0_21b8 | CORE1_WIN7_MMAP |
| | | | |
| 0x3ff0_2200 | CORE2_WIN0_BASE | 0x3ff0_2300 | CORE3_WIN0_BASE |
| 0x3ff0_2208 | CORE2_WIN1_BASE | 0x3ff0_2308 | CORE3_WIN1_BASE |
| 0x3ff0_2210 | CORE2_WIN2_BASE | 0x3ff0_2310 | CORE3_WIN2_BASE |
| 0x3ff0_2218 | CORE2_WIN3_BASE | 0x3ff0_2318 | CORE3_WIN3_BASE |
| 0x3ff0_2220 | CORE2_WIN4_BASE | 0x3ff0_2320 | CORE3_WIN4_BASE |
| 0x3ff0_2228 | CORE2_WIN5_BASE | 0x3ff0_2328 | CORE3_WIN5_BASE |
| 0x3ff0_2230 | CORE2_WIN6_BASE | 0x3ff0_2330 | CORE3_WIN6_BASE |
| 0x3ff0_2238 | CORE2_WIN7_BASE | 0x3ff0_2338 | CORE3_WIN7_BASE |
| 0x3ff0_2240 | CORE2_WIN0_MASK | 0x3ff0_2340 | CORE3_WIN0_MASK |
| 0x3ff0_2248 | CORE2_WIN1_MASK | 0x3ff0_2348 | CORE3_WIN1_MASK |
| 0x3ff0_2250 | CORE2_WIN2_MASK | 0x3ff0_2350 | CORE3_WIN2_MASK |
| 0x3ff0_2258 | CORE2_WIN3_MASK | 0x3ff0_2358 | CORE3_WIN3_MASK |
| 0x3ff0_2260 | CORE2_WIN4_MASK | 0x3ff0_2360 | CORE3_WIN4_MASK |
| 0x3ff0_2268 | CORE2_WIN5_MASK | 0x3ff0_2368 | CORE3_WIN5_MASK |
| 0x3ff0_2270 | CORE2_WIN6_MASK | 0x3ff0_2370 | CORE3_WIN6_MASK |
| 0x3ff0_2278 | CORE2_WIN7_MASK | 0x3ff0_2378 | CORE3_WIN7_MASK |
| 0x3ff0_2280 | CORE2_WIN0_MMAP | 0x3ff0_2380 | CORE3_WIN0_MMAP |

| | | | |
|---|---|---|---|
| 0x3ff0_2288 | CORE2_WIN1_MMAP | 0x3ff0_2388 | CORE3_WIN1_MMAP |
| 0x3ff0_2290 | CORE2_WIN2_MMAP | 0x3ff0_2390 | CORE3_WIN2_MMAP |
| 0x3ff0_2298 | CORE2_WIN3_MMAP | 0x3ff0_2398 | CORE3_WIN3_MMAP |
| 0x3ff0_22a0 | CORE2_WIN4_MMAP | 0x3ff0_22G0 | CORE3_WIN4_MMAP |
| 0x3ff0_22a8 | CORE2_WIN5_MMAP | 0x3ff0_22G8 | CORE3_WIN5_MMAP |
| 0x3ff0_22b0 | CORE2_WIN6_MMAP | 0x3ff0_23b0 | CORE3_WIN6_MMAP |
| 0x3ff0_22b8 | CORE2_WIN7_MMAP | 0x3ff0_23b8 | CORE3_WIN7_MMAP |
| | | | |
| | | 0x3ff0_2700 | HT_WIN0_BASE |
| | | 0x3ff0_2708 | HT_WIN1_BASE |
| | | 0x3ff0_2710 | HT_WIN2_BASE |
| | | 0x3ff0_2718 | HT_WIN3_BASE |
| | | 0x3ff0_2720 | HT_WIN4_BASE |
| | | 0x3ff0_2728 | HT_WIN5_BASE |
| | | 0x3ff0_2730 | HT_WIN6_BASE |
| | | 0x3ff0_2738 | HT_WIN7_BASE |
| | | 0x3ff0_2740 | HT_WIN0_MASK |
| | | 0x3ff0_2748 | HT_WIN1_MASK |
| | | 0x3ff0_2750 | HT_WIN2_MASK |
| | | 0x3ff0_2758 | HT_WIN3_MASK |
| | | 0x3ff0_2760 | HT_WIN4_MASK |
| | | 0x3ff0_2768 | HT_WIN5_MASK |
| | | 0x3ff0_2770 | HT_WIN6_MASK |
| | | 0x3ff0_2778 | HT_WIN7_MASK |
| | | 0x3ff0_2780 | HT_WIN0_MMAP |
| | | 0x3ff0_2788 | HT_WIN1_MMAP |
| | | 0x3ff0_2790 | HT_WIN2_MMAP |
| | | 0x3ff0_2798 | HT_WIN3_MMAP |
| | | 0x3ff0_27a0 | HT_WIN4_MMAP |
| | | 0x3ff0_27a8 | HT_WIN5_MMAP |
| | | 0x3ff0_27b0 | HT_WIN6_MMAP |
| | | 0x3ff0_27b8 | HT_WIN7_MMAP |

在龙芯 2G 的二级 XBAR 中有 DDR2 地址空间、以及低速 IO 地址空间及系统配置空间共三个相关的地址空间。地址窗口是供 CPU 或 HT 向下访问这几个

空间时进行路由选择和地址转换而设置的。共拥有 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（其中两个 DDR2 的标号为 0 与 1，低速 IO 空间编号为 2，配置寄存器模块连接在端口 3 上）。

表 2-6 2 级 XBAR 处，标号与所述模块的对应关系

| 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（LPC、SPI、UART 等） |
| 3 | 配置寄存器 |

如下表所示。MMAP[4]表示允许取指，MMAP[5]表示允许块读，MMAP[7]表示窗口使能。

表 2-7 MMAP 字段对应的该空间访问属性

| [4] | [5] | [7] |
|---|---|---|
| 允许取指 | 允许块读 | 窗口使能 |

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。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

新地址换算公式： OUT_ADDR = (IN_ADDR & ~MASK) | {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

| 地址 | 寄存器 | 描述 | 缺省值 |
|---|---|---|---|
| 3ff0 0000 | CPU_WIN0_BASE | 窗口 0 的基地址 | 0x0 |
| 3ff0 0008 | CPU_WIN1_BASE | 窗口 1 的基地址 | 0x1000_0000 |
| 3ff0 0010 | CPU_WIN2_BASE | 窗口 2 的基地址 | 0x0 |
| 3ff0 0018 | CPU_WIN3_BASE | 窗口 3 的基地址 | 0x0 |

| | | | |
|---|---|---|---|
| 3ff0 0020 | CPU_WIN4_BASE | 窗口 4 的基地址 | 0x0 |
| 3ff0 0028 | CPU_WIN5_BASE | 窗口 5 的基地址 | 0x0 |
| 3ff0 0030 | CPU_WIN6_BASE | 窗口 6 的基地址 | 0x0 |
| 3ff0 0038 | CPU_WIN7_BASE | 窗口 7 的基地址 | 0x0 |
| 3ff0 0040 | CPU_WIN0_MASK | 窗口 0 的掩码 | 0xffff_ffff_f000_0000 |
| 3ff0 0048 | CPU_WIN1_MASK | 窗口 1 的掩码 | 0xffff_ffff_f000_0000 |
| 3ff0 0050 | CPU_WIN2_MASK | 窗口 2 的掩码 | 0x0 |
| 3ff0 0058 | CPU_WIN3_MASK | 窗口 3 的掩码 | 0x0 |
| 3ff0 0060 | CPU_WIN4_MASK | 窗口 4 的掩码 | 0x0 |
| 3ff0 0068 | CPU_WIN5_MASK | 窗口 5 的掩码 | 0x0 |
| 3ff0 0070 | CPU_WIN6_MASK | 窗口 6 的掩码 | 0x0 |
| 3ff0 0078 | CPU_WIN7_MASK | 窗口 7 的掩码 | 0x0 |
| 3ff0 0080 | CPU_WIN0_MMAP | 窗口 0 的新基地址 | 0xf0 |
| 3ff0 0088 | CPU_WIN1_MMAP | 窗口 1 的新基地址 | 0x1000_00f2 |
| 3ff0 0090 | CPU_WIN2_MMAP | 窗口 2 的新基地址 | 0 |
| 3ff0 0098 | CPU_WIN3_MMAP | 窗口 3 的新基地址 | 0 |
| 3ff0 00a0 | CPU_WIN4_MMAP | 窗口 4 的新基地址 | 0x0 |
| 3ff0 00a8 | CPU_WIN5_MMAP | 窗口 5 的新基地址 | 0x0 |
| 3ff0 00b0 | CPU_WIN6_MMAP | 窗口 6 的新基地址 | 0 |
| 3ff0 00b8 | CPU_WIN7_MMAP | 窗口 7 的新基地址 | 0 |

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）映射到低速 IO 的 0x10000000 - 0x1fffffff 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。

表 2-9 二级 XBAR 缺省地址配置

| 基地址 | 高位 | 所有者 |
|---|---|---|
| 0x0000_0000_0000_0000 | 0x0000_0000_1000_0000 | 0 号 DDR 控制器 |
| 0x0000_0000_1000_0000 | 0x0000_0000_2000_0000 | 低速 I/O（LPC、SPI、UART 等） |

## 2.5 芯片配置及采样寄存器

龙芯 2G 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。

表 2-10 芯片配置寄存器（物理地址 0x1fe00180）

| 位域 | 字段名 | 访问 | 复位值 | 描述 |
|---|---|---|---|---|
| 2:0 | Freq_scale_ctrl | RW | 3’b111 | 处理器核分频 |
| 3 | DDR_Clksel_en | RW | 1’b0 | 是否使用软件配置 DDR 倍频 |
| 8 | Disable_ddr2_confspace | RW | 1’b0 | 是否禁用 DDR 配置空间 |
| 9 | DDR_buffer_cpu | RW | 1’b0 | 是否打开 DDR 读访问缓冲 |
| 12 | Core0_en | RW | 1’b1 | 是否启用处理器核 0 |
| 13 | Core1_en | RW | 1’b1 | 是否启用处理器核 1 |
| 14 | Core2_en | RW | 1’b1 | 是否启用处理器核 2 |
| 15 | Core3_en | RW | 1’b1 | 是否启用处理器核 3 |
| 16 | Mc0_en | RW | 1’b1 | 是否启用 DDR 控制器 0 |
| 17 | Mc1_en | RW | 1’b1 | 是否启用 DDR 控制器 1 |
| 18 | DDR_reset0 | RW | 1’b1 | 软件 reset DDR 控制器 0 |
| 19 | DDR_reset1 | RW | 1’b1 | 软件 reset DDR 控制器 1 |
| 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） |
| 31:29 | HT_freq_scale_ctrl0 | RW | 3’b111 | HT 控制器 0 分频 |
| | | | | |
| 其它 | | R | | 保留 |

表 2-11 芯片采样寄存器（物理地址 0x1fe00190）

| 位域 | 字段名 | 访问 | 复位值 | 描述 |
|---|---|---|---|---|
| 15:0 | Pad2v5_ctrl | RW | 16’h780 | 2v5pad 控制 |
| 31:16 | Pad3v3_ctrl | RW | 16’h780 | 3v3pad 控制 |
| 47:32 | Sys_clksel | R | | 板上倍频设置 |

| | | | | |
|---|---|---|---|---|
| 51:48 | Bad_ip_core | R | | 4 个处理器核是否坏 |
| 53:52 | Bad_ip_ddr | R | | 2 个 DDR 控制器是否坏 |
| 57 | Bad_ip_ht | R | | HT 控制器是否坏 |
| 102:96 | Thsens0_out | R | | 温度传感器 0 温度 |
| 103 | Thsens0_overflow | R | | 温度传感器 0 温度上溢（超过 128 度） |
| 110:104 | Thsens1_out | R | | 温度传感器 1 温度 |
| 111 | Thsens1_overflow | R | | 温度传感器 1 温度上溢（超过 128 度） |
| 其它 | | R | | 保留 |

# 3 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 2G 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 的结构如图 3-1 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。

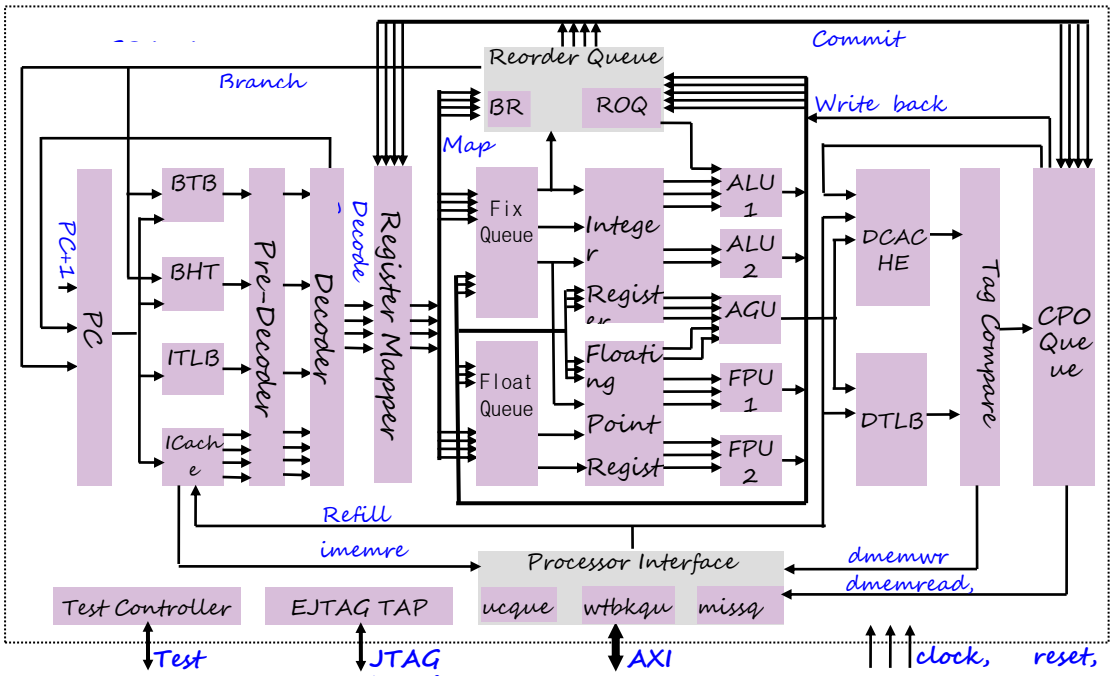



图 3-1 GS464 结构图

# 4 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 和多个二级 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：

- 通过目录支持 Cache 一致性协议；
- 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。

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）。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。

表 4-1 二级 Cache 锁窗口寄存器配置

| 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 |

举例来说，当一个地址 addr 使得 slock0_valid && ((addr & slock0_mask) == (slock0_addr & slock0_mask) )为 1 时，这个地址就被锁窗口 0 锁住了。

# 5 矩阵转置模块

龙芯 2G 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，对存放在内存中的矩阵进行从源矩阵到目标矩阵的行列转置功能。两个矩阵转置模块通过 1 级交叉开关实现对二级 Cache 及内存的读写。

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 5-1 到 5-4 说明了矩阵转置涉及到的编程接口。

表 5-1 矩阵转置编程接口说明

| 地址 | 名称 | 属性 | 说明 |
|---|---|---|---|
| 0x3ff00600 | src_start_addr | RW | 源矩阵起始地址 |
| 0x3ff00608 | dst_start_addr | RW | 目标矩阵起始地址 |
| 0x3ff00610 | row | RW | 源矩阵的一行元素个数 |
| 0x3ff00618 | col | RW | 源矩阵的一列元素个数 |
| 0x3ff00620 | length | RW | 源矩阵所在大矩阵的行跨度（字节） |
| 0x3ff00628 | width | RW | 目标矩阵所在大矩阵的行跨度（字节） |
| 0x3ff00630 | trans_ctrl | RW | 转置控制寄存器 |
| 0x3ff00638 | trans_status | RO | 转置状态寄存器 |

表 5-2 矩阵转置寄存器地址说明

| 地址 | 名称 |
|---|---|
| 0x3ff00600 | 0 号转置模块的 src_start_addr |
| 0x3ff00608 | 0 号转置模块的 dst_start_addr |
| 0x3ff00610 | 0 号转置模块的 row |
| 0x3ff00618 | 0 号转置模块的 col |
| 0x3ff00620 | 0 号转置模块的 length |
| 0x3ff00628 | 0 号转置模块的 width |
| 0x3ff00630 | 0 号转置模块的 trans_ctrl |
| 0x3ff00638 | 0 号转置模块的 trans_status |
| 0x3ff00700 | 1 号转置模块的 src_start_addr |
| 0x3ff00708 | 1 号转置模块的 dst_start_addr |
| 0x3ff00710 | 1 号转置模块的 src_row_stride |
| 0x3ff00718 | 1 号转置模块的 src_last_row_addr |
| 0x3ff00720 | 1 号转置模块的 length |
| 0x3ff00728 | 1 号转置模块的 width |
| 0x3ff00730 | 1 号转置模块的 trans_ctrl |
| 0x3ff00738 | 1 号转置模块的 trans_status |

表 5-3 trans_ctrl 寄存器的各位解释

| 字段 | 说明 |
|---|---|
| 0 | 使能位 |
| 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。 |
| 2 | 源矩阵读取完毕后，是否有效中断。 |
| 3 | 目标矩阵写入完毕后，是否有效中断， |
| 7..4 | Arcmd，读命令内部控制位。当 arcache 为 4’hf 时，必须设为 4’hc。当 arcache 为其它值时无意义。 |
| 11..8 | Arcache，读命令内部控制位。为 4’hf 时，使用 cache 通路，为 4’h0 时，使用 uncache 通路。其它值无意义。 |
| 15..12 | Awcmd，写命令内部控制位。当 awcache 为 4’hf 时，必须设为 4’hb。当 awcache 为其它值时无意义。 |
| 19..16 | Awcache，写命令内部控制位。为 4’hf 时，使用 cache 通路，为 4’h0 时，使用 uncache 通路。其它值无意义。 |

| | |
|---|---|
| 21..20 | 矩阵的元素大小，00 表示 1 个字节，01 表示 2 个字节，10 表示 4 个字节，11 表示 8 个字节 |

表 5-4 trans_status 寄存器的各位解释：

| 字段 | 说明 |
|---|---|
| 0 | 源矩阵读取完毕 |
| 1 | 目标矩阵写入完毕 |

# 6 处理器核间中断与通信

龙芯 2G 为每个处理器核都实现了 8 个核间中断寄存器（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，其说明和地址见表 6-1 到表 6-5。

表 6-1 处理器核间中断相关的寄存器及其功能描述

| 名称 | 读写权限 | 描述 |
|---|---|---|
| IPI_Status | R | 32 位状态寄存器，任何一位有被置 1 且对应位使能情况下，处理器核 INT4 中断线被置位。 |
| IPI_Enable | RW | 32 位使能寄存器，控制对应中断位是否有效 |
| IPI_Set | W | 32 位置位寄存器，往对应的位写 1，则对应的 STATUS 寄存器位被置 1 |
| IPI_Clear | W | 32 位清除寄存器，往对应的位写 1，则对应的 STATUS 寄存器位被清 0 |
| MailBox0 | RW | 缓存寄存器，供启动时传递参数使用，按 64 或者 32 位的 uncache 方式进行访问。 |
| MailBox01 | RW | 缓存寄存器，供启动时传递参数使用，按 64 或者 32 位的 uncache 方式进行访问。 |
| MailBox02 | RW | 缓存寄存器，供启动时传递参数使用，按 64 或者 32 位的 uncache 方式进行访问。 |
| MailBox03 | RW | 缓存寄存器，供启动时传递参数使用，按 64 或者 32 位的 uncache 方式进行访问。 |

在龙芯 2G 与处理器核间中断相关的寄存器及其功能描述如下：

表 6-2 0 号处理器核核间中断与通信寄存器列表

| 名称 | 地址 | 权限 | 描述 |
|---|---|---|---|
| Core0_IPI_Status | 0x3ff01000 | R | 0 号处理器核的 IPI_Status 寄存器 |
| Core0_IPI_Enalbe | 0x3ff01004 | RW | 0 号处理器核的 IPI_Enalbe 寄存器 |
| Core0_IPI_Set | 0x3ff01008 | W | 0 号处理器核的 IPI_Set 寄存器 |
| Core0 _IPI_Clear | 0x3ff0100c | W | 0 号处理器核的 IPI_Clear 寄存器 |
| Core0_MailBox0 | 0x3ff01020 | RW | 0 号处理器核的 IPI_MailBox0 寄存器 |
| Core0_ MailBox1 | 0x3ff01028 | RW | 0 号处理器核的 IPI_MailBox1 寄存器 |
| Core0_ MailBox2 | 0x3ff01030 | RW | 0 号处理器核的 IPI_MailBox2 寄存器 |
| Core0_ MailBox3 | 0x3ff01038 | RW | 0 号处理器核的 IPI_MailBox3 寄存器 |

表 6-3 1 号处理器核的核间中断与通信寄存器列表

| 名称 | 地址 | 权限 | 描述 |
|---|---|---|---|
| Core1_IPI_Status | 0x3ff01100 | R | 1 号处理器核的 IPI_Status 寄存器 |
| Core1_IPI_Enalbe | 0x3ff01104 | RW | 1 号处理器核的 IPI_Enalbe 寄存器 |

| | | | |
|---|---|---|---|
| Core1_IPI_Set | 0x3ff01108 | W | 1 号处理器核的 IPI_Set 寄存器 |
| Core1 _IPI_Clear | 0x3ff0110c | W | 1 号处理器核的 IPI_Clear 寄存器 |
| Core1_MailBox0 | 0x3ff01120 | R | 1 号处理器核的 IPI_MailBox0 寄存器 |
| Core1_ MailBox1 | 0x3ff01128 | RW | 1 号处理器核的 IPI_MailBox1 寄存器 |
| Core1_ MailBox2 | 0x3ff01130 | W | 1 号处理器核的 IPI_MailBox2 寄存器 |
| Core1_ MailBox3 | 0x3ff01138 | W | 1 号处理器核的 IPI_MailBox3 寄存器 |

表 6-4 2 号处理器核的核间中断与通信寄存器列表

| 名称 | 地址 | 权限 | 描述 |
|---|---|---|---|
| Core2_IPI_Status | 0x3ff01200 | R | 2 号处理器核的 IPI_Status 寄存器 |
| Core2_IPI_Enalbe | 0x3ff01204 | RW | 2 号处理器核的 IPI_Enalbe 寄存器 |
| Core2_IPI_Set | 0x3ff01208 | W | 2 号处理器核的 IPI_Set 寄存器 |
| Core2 _IPI_Clear | 0x3ff0120c | W | 2 号处理器核的 IPI_Clear 寄存器 |
| Core2_MailBox0 | 0x3ff01220 | R | 2 号处理器核的 IPI_MailBox0 寄存器 |
| Core2_ MailBox1 | 0x3ff01228 | RW | 2 号处理器核的 IPI_MailBox1 寄存器 |
| Core2_ MailBox2 | 0x3ff01230 | W | 2 号处理器核的 IPI_MailBox2 寄存器 |
| Core2_ MailBox3 | 0x3ff01238 | W | 2 号处理器核的 IPI_MailBox3 寄存器 |

表 6-5 3 号处理器核的核间中断与通信寄存器列表

| 名称 | 地址 | 权限 | 描述 |
|---|---|---|---|
| Core3_IPI_Status | 0x3ff01300 | R | 3 号处理器核的 IPI_Status 寄存器 |
| Core3_IPI_Enalbe | 0x3ff01304 | RW | 3 号处理器核的 IPI_Enalbe 寄存器 |
| Core3_IPI_Set | 0x3ff01308 | W | 3 号处理器核的 IPI_Set 寄存器 |
| Core3 _IPI_Clear | 0x3ff0130c | W | 3 号处理器核的 IPI_Clear 寄存器 |
| Core3_MailBox0 | 0x3ff01320 | R | 3 号处理器核的 IPI_MailBox0 寄存器 |
| Core3_ MailBox1 | 0x3ff01328 | RW | 3 号处理器核的 IPI_MailBox1 寄存器 |
| Core3_ MailBox2 | 0x3ff01330 | W | 3 号处理器核的 IPI_MailBox2 寄存器 |
| Core3_ MailBox3 | 0x3ff01338 | W | 3 号处理器核的 IPI_MailBox3 寄存器 |

上面列出的是单个龙芯 2G 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。

# 7 I/O 中断

龙芯 2G 芯片最多支持 32 个中断源，以统一方式进行管理，如图 7-1 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。

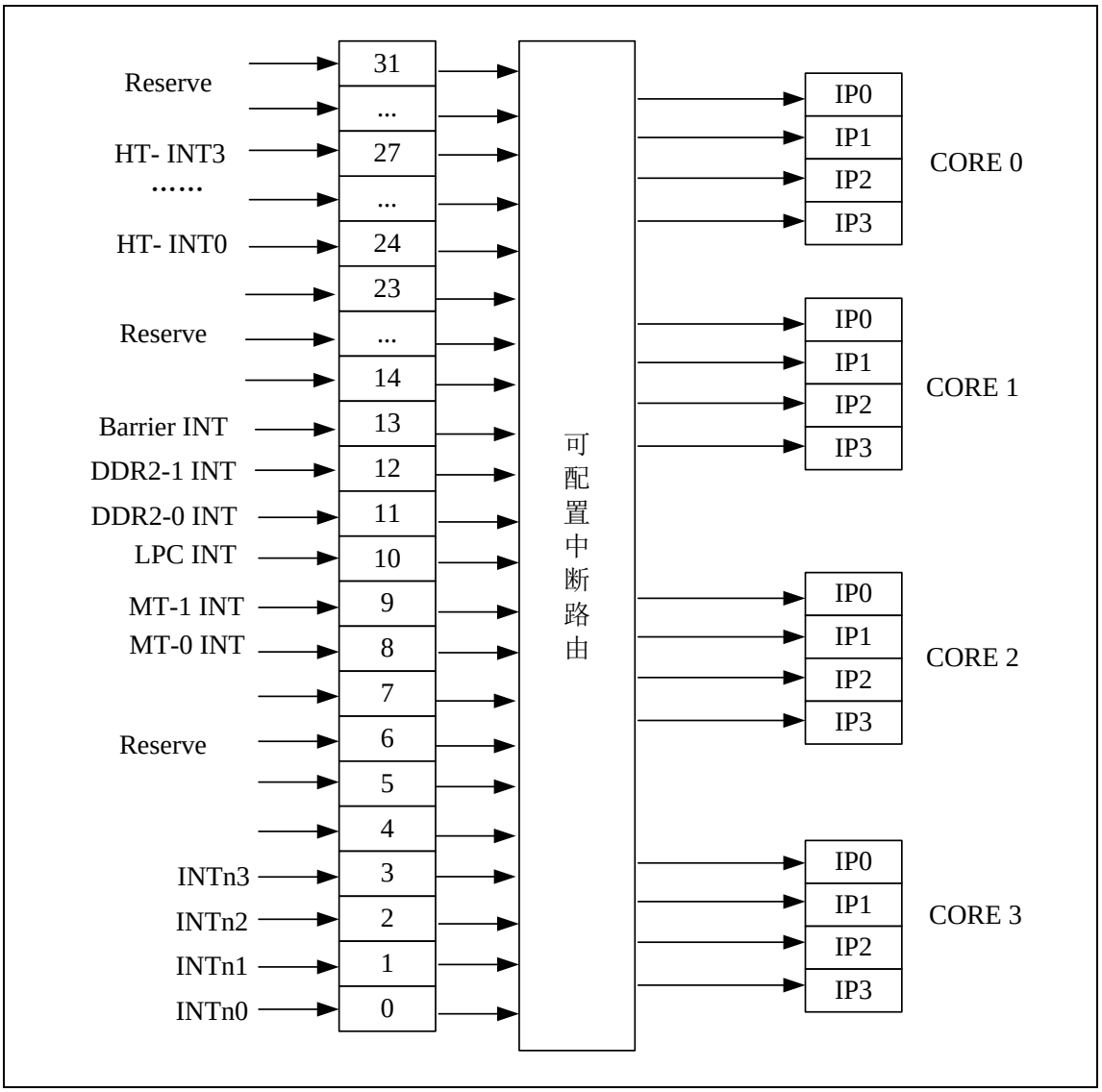



图 7-1 龙芯 2G 处理器中断路由示意图

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 7-1。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。

表 7-1 中断控制寄存器

| 位域 | 访问属性/缺省值 | | | | |
|---|---|---|---|---|---|
| | Intedge | Inten | Intenset | Intenclr | 中断源 |
| 3 : 0 | RW / 0 | R / 0 | W / 0 | W / 0 | Sys_int0-3 |
| 7 : 4 | RO / 0 | R / 0 | RW / 0 | RW / 0 | 保留 |
| 8 | RO / 0 | R / 0 | RW / 0 | RW / 0 | Matrix_int0 |
| 9 | RO / 1 | R / 0 | RW / 0 | RW / 0 | Matrix_int1 |
| 10 | RO / 1 | R / 0 | RW / 0 | RW / 0 | Lpc |
| 12 : 11 | RW / 0 | 保留 | 保留 | 保留 | Mc0-1 |
| 13 | RW / 0 | R / 0 | RW / 0 | RW / 0 | Barrier |
| 14 | RW / 0 | R / 0 | RW / 0 | RW / 0 | 保留 |
| 15 | RW / 0 | R / 0 | RW / 0 | RW / 0 | 保留 |
| 23 : 16 | RW / 0 | R / 0 | RW / 0 | RW / 0 | 保留 |
| 27 : 24 | RW / 0 | R / 0 | RW / 0 | RW / 0 | HT int0-3 |
| 31 : 28 | RW / 0 | R / 0 | RW / 0 | RW / 0 | 保留 |

表 7-2 IO 控制寄存器地址

| 名称 | 地址偏移 | 描述 |
|---|---|---|
| Intisr | 0x3ff01420 | 32 位中断状态寄存器 |
| Inten | 0x3ff01424 | 32 位中断使能状态寄存器 |
| Intenset | 0x3ff01428 | 32 位设置使能寄存器 |
| Intenclr | 0x3ff0142c | 32 位清除使能寄存器 |
| Intedge | 0x3ff01438 | 32 位触发方式寄存器 |
| CORE0_INTISR | 0x3ff01440 | 路由给 CORE0 的 32 位中断状态 |
| CORE1_INTISR | 0x3ff01448 | 路由给 CORE1 的 32 位中断状态 |
| CORE2_INTISR | 0x3ff01450 | 路由给 CORE2 的 32 位中断状态 |
| CORE3_INTISR | 0x3ff01458 | 路由给 CORE3 的 32 位中断状态 |

在龙芯 2G 中集成了 3~4 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。32 个 I/O 中断源中每一个都对应一个 8 位的路由控制器，其格式和地址如下表 7-3 和 7-4 所示。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。

表 7-3 中断路由寄存器的说明

| 位域 | 说　明 |
|---|---|
| 3:0 | 路由的处理器核向量号 |

| | |
|---|---|
| 7:4 | 路由的处理器核中断引脚向量号 |

表 7-4 中断路由寄存器地址

| 名称 | 地址偏移 | 描述 | 名称 | 地址偏移 | 描述 |
|---|---|---|---|---|---|
| Entry0 | 0x3ff01400 | Sys_int0 | Entry16 | 0x3ff01410 | 保留 |
| Entry1 | 0x3ff01402 | Sys_int1 | Entry17 | 0x3ff01411 | 保留 |
| Entry2 | 0x3ff01403 | Sys_int2 | Entry18 | 0x3ff01412 | 保留 |
| Entry3 | 0x3ff01404 | Sys_int3 | Entry19 | 0x3ff01413 | 保留 |
| Entry4 | 0x3ff01405 | 保留 | Entry20 | 0x3ff01414 | 保留 |
| Entry5 | 0x3ff01406 | 保留 | Entry21 | 0x3ff01415 | 保留 |
| Entry6 | 0x3ff01407 | 保留 | Entry22 | 0x3ff01416 | 保留 |
| Entry7 | 0x3ff01408 | 保留 | Entry23 | 0x3ff01417 | 保留 |
| Entry8 | 0x3ff01409 | Matrix int0 | Entry24 | 0x3ff01418 | HT-int0 |
| Entry9 | 0x3ff0140a | Matrix int1 | Entry25 | 0x3ff01419 | HT-int1 |
| Entry10 | 0x3ff0140b | Lpc int | Entry26 | 0x3ff0141a | HT-int2 |
| Entry11 | 0x3ff0140c | Mc0 | Entry27 | 0x3ff0141b | HT-int3 |
| Entry12 | 0x3ff0140d | Mc1 | Entry28 | 0x3ff0141c | 保留 |
| Entry13 | 0x3ff0140e | Barrier | Entry29 | 0x3ff0141d | 保留 |
| Entry14 | 0x3ff0140f | 保留 | Entry30 | 0x3ff0141e | 保留 |
| Entry15 | 0x3ff0140f | Pci_perr/serr | Entry31 | 0x3ff0141f | 保留 |

# 8 DDR2/3 SDRAM 控制器配置

龙芯 2G 处理器内部集成 DDR2/3 内存控制器，其设计遵守 DDR2/3 SDRAM 的行业标准（JESD79-2B 和 JESD79-3）。在龙芯 2G 处理器中，所实现的所有内存读/写操作都遵守 JESD79-2B 及 JESD79-3 的规定。

## 8.1 DDR2/3 SDRAM 控制器功能概述

龙芯 2G 处理器支持最多 4 个 CS（由 4 个 DDR2 SDRAM 片选信号实现，即两个双面内存条），一共含有 18 位的地址总线（即：15 位的行列地址总线和 3 位的逻辑 Bank 总线）。

龙芯 2G 处理器在具体选择使用不同内存芯片类型时，可以调整 DDR2/3 控制器参数设置进行支持。其中，支持的最大片选（CS_n）数为 4，行地址（RAS_n）数为 15，列地址（CAS_n）数为 14，逻辑体选择（BANK_n）数为 3。最大支持的地址空间为 128GB（$2^{37}$）。

CPU 发送的内存请求物理地址将按照如图 8-1 所示的方法进行行列地址转换：

以 4GB 地址空间为例，按照下面的配置：

片选 = 4　　Bank 数 = 8

行地址数 = 12　列地址数 = 12


图 8-1 DDR2 SDRAM 行列地址与 CPU 物理地址的转换

龙芯 2G 处理器所集成的内存控制电路只接受来自处理器或者外部设备的内存读/写请求，在所有的内存读/写操作中，内存控制电路处于从设备状态（Slave State）。

龙芯 2G 处理器中内存控制器具有如下特征：

- 接口上命令、读写数据全流水操作
- 内存命令合并、排序提高整体带宽
- 配置寄存器读写端口，可以修改内存设备的基本参数
- 内建动态延迟补偿电路（DCC），用于数据的可靠发送和接收
- 支持 133-400MHZ 工作频率

## 8.2 DDR2/3 SDRAM 读操作协议

DDR2/3 SDRAM 读操作的协议如图 8-2 所示。在图中命令（Command，简称 CMD）由 RAS_n，CAS_n 和 WE_n，共三个信号组成。对于读操作，RAS_n=1，CAS_n＝0，WE_n＝1。

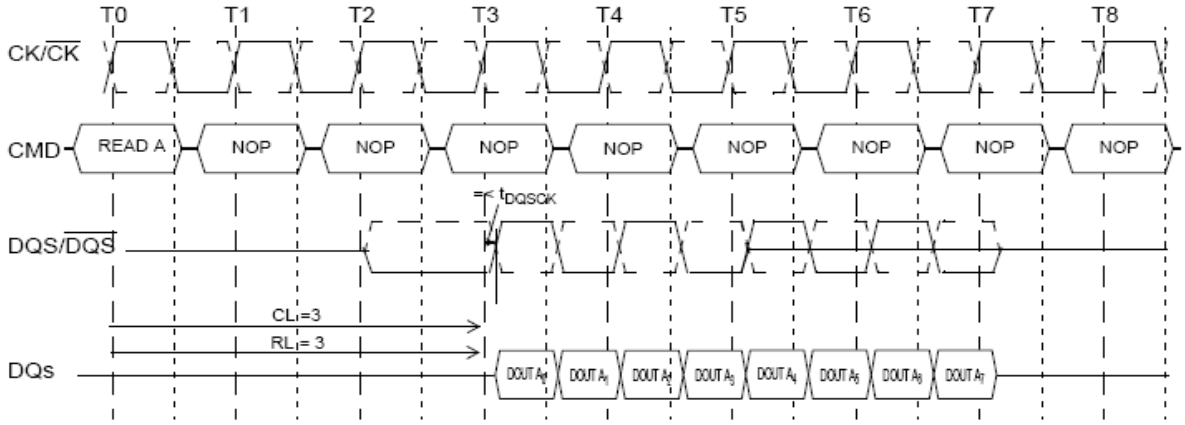



图 8-2 DDR2 SDRAM 读操作协议

图 8-2 中，Cas Latency (CL) = 3，Read Latency (RL) = 3，Burst Length = 8。

## 8.3 DDR2/3 SDRAM 写操作协议

DDR2/3 SDRAM 写操作的协议如图 8-3 所示。在图中命令 CMD 是由 RAS_n，CAS_n 和 WE_n，共三个信号组成的。对于写操作，RAS_n=1，CAS_n＝0，WE_n＝0。另外，与读操作不同，写操作需要 DQM 来标识写操作的掩码，即需要写入的字节数。DQM 与图 8-3 中 DQS 信号同步。

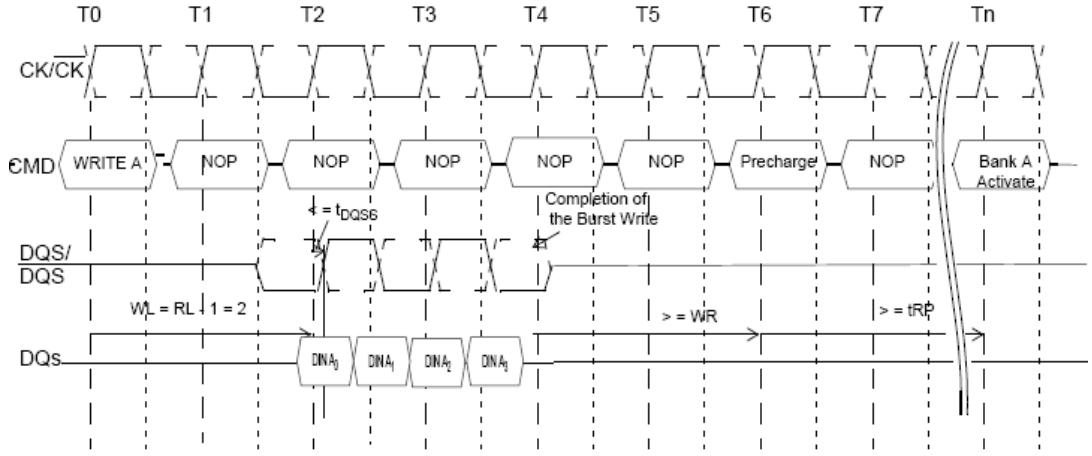



图 8-3 DDR2 SDRAM 写操作协议

图 8-3 中，Cas Latency (CL) = 3，Write Latency (WL) = Read Latency (RL) – 1 = 2，Burst Length = 4。

## 8.4 DDR2/3 SDRAM 参数配置格式

由于系统中可能使用不同类型的 DDR2/3 SDRAM，因此，在系统上电复位以后，需要对 DDR2/3 SDRAM 进行配置。在 JESD79-2B 和 JESD79-3 中规定了详细的配置操作和配置过程，在没有完成 DDR2/3 的内存初始化操作之前，DDR2/3 不可用。内存初始化操作执行顺序如下：

(1) 系统复位，此时控制器内部所有寄存器内容将被清除为初始值。

(2) 系统解复位。

(3) 向配置寄存器地址发 64 位写指令，配置所有 180 个配置寄存器。此时如果写 CTRL_03，应将其中参数 START 设为 0。所有寄存器都必须正确配置才可以正常工作。

(4) 向配置寄存器 CTRL_03 中发 64 位写指令。此时应将参数 START 设为 1。结束后内存控制器将自动对内存发起初始化指令。

在龙芯 2 号处理器设计中，DDR2/3 SDRAM 的配置在系统主板初始化完成以后，需要使用内存之前，进行内存类型的配置。具体的配置操作是对物理地址 0x0000 0000 0FF0 0000 相对应的 180 个 64 位寄存器写入相应的配置参数。一个寄存器可能会包括多个、一个、部分参数的数据。这些配置寄存器及其包含的参数意义如下表（寄存器中未使用的位均为保留位），表中还给出了基于 DDR2 667 的一种寄存器配置方式，具体的配置可以根据实际情况再决定：

表 8-1 DDR2 SDRAM 配置参数寄存器格式

| 参数名称 | 位 | 缺省值 | 范围 | 描述 |
|---|---|---|---|---|
| CONF_CTL_00[63:0] | Offset: 0x00 | | DDR2 667：0x0000010000000101 | |
| CONCURRENTAP | 48:48 | 0x0 | 0x0-0x1 | 是否允许控制器对一个 bank 进行 auto precharge 时，对另外一个 bank 发出命令。注：部分内存条不支持 |
| BANK_SPLIT_EN | 40:40 | 0x0 | 0x0-0x1 | 是否允许命令队列重排序逻辑对 bank 进行拆分（split） |
| AUTO_REFRESH_MODE | 32:32 | 0x0 | 0x0-0x1 | 设置 auto-refresh 是在下一个 burst 还是下一个命令边界发出 |
| AREFRESH | 24:24 | 0x0 | 0x0-0x1 | 根据 auto_refresh_mode 参数的设置，向内存发起自动刷新命令（只写） |
| AP | 16:16 | 0x0 | 0x0-0x1 | 是否使能内存控制器自动刷新功能，置 1，表示内存访问为 CLOSE PAGE 方式。 |

| | | | | |
|---|---|---|---|---|
| ADDR_CMP_EN | 8:8 | 0x0 | 0x0-0x1 | 是否允许命令队列重排序逻辑对地址冲突进行检测 |
| CONF_CTL_01[63:0] | Offset: 0x10 | | DDR2 667：0x0000010100010000 | |
| FWC | 56:56 | 0x0 | 0x0-0x1 | 是否强制进行写检查，当这个参数设置后，内存控制器将用 xor_check_bits 参数指定的数与数据进行异或写入内存（只写） |
| FAST_WRITE | 48:48 | 0x0 | 0x0-0x1 | 是否允许控制器打开快速写功能。打开快速写功能后，控制器在未收到全部写数据后即向内存模块发出写命令。 |
| ENABLE_QUICK_SREFRESH | 40:40 | 0x0 | 0x0-0x1 | 是否使能快速自刷新。当这个参数使能后，内存的初始化未进行完就进入自刷新状态 |
| EIGHT_BANK_MODE | 32:32 | 0x0 | 0x0-0x1 | 指示内存模块是否有 8 个 bank |
| ECC_DISBALE_W_UC_ERR | 24:24 | 0x0 | 0x0-0x1 | 当检测到不可恢复的错误时，是否将 ECC 关闭 |
| DQS_N_EN | 16:16 | 0x0 | 0x0-0x1 | 设置 DQS 信号为单端还是差分信号。 |
| DLLLOCKREG | 0:0 | 0x0 | 0x0-0x1 | 指示 DLL 是否已锁定(只读)，只有在 DLL 锁定之后，对内存发起的读写操作才能有效到达内存，所以，可以用本位判断第一次写内存的时机。 |
| CONF_CTL_02[63:0] | Offset: 0x20 | | DDR2 667：0x0100010100000000 | |
| PRIORITY_EN | 56:56 | 0x0 | 0x0-0x1 | 是否使能命令队列重排序逻辑使用优先级 |
| POWER_DOWN | 48:48 | 0x0 | 0x0-0x1 | 当使能这个参数时，内存控制器将用 pre-charge 命令关闭内存模块的所有页面，使时钟使能信号为低，不发送收到的所有命令，直到这个参数重新设置为 0 |
| PLACEMENT_EN | 40:40 | 0x0 | 0x0-0x1 | 是否使能命令重排序逻辑 |
| ODT_ADD_TURN_CLK_EN | 32:32 | 0x0 | 0x0-0x1 | 在对不同片选的快速背对背读或者写命令中间是否插入一个 turn-around 时钟。通常情况下，插入一个这样的周期是对内存是需要的。 |
| NO_CMD_INIT | 24:24 | 0x0 | 0x0-0x1 | 在内存初始化过程中，是否禁止在内存模 |

| | | | | |
|---|---|---|---|---|
| | | | | 块的 tDLL 时间内发出其它命令 |
| INTRPTWRITENA | 16:16 | 0x0 | 0x0-0x1 | 是否允许用 autoprecharge 命令加上对同一 bank 的其它写命令打断前一个写命令 |
| INTRPTREADA | 8:8 | 0x0 | 0x0-0x1 | 是否允许用 autoprecharge 命令加上对同一 bank 的其它读命令打断前一个读命令 |
| INTRPTAPBURST | 0:0 | 0x0 | 0x0-0x1 | 是否允许对另一 bank 的其它命令打断当前的 auto-precharge 命令 |
| CONF_CTL_03[63:0] | Offset: 0x30 | | DDR2 667：0x0101010001000000 | |
| SWAP_PORT_RW_SAME_EN | 56:56 | 0x0 | 0x0-0x1 | 当 swap_en 使能时，该参数决定是否将同一端口上的类似命令进行交换 |
| SWAP_EN | 48:48 | 0x0 | 0x0-0x1 | 在使能命令队列重排序逻辑时，当高优先级命令到达时，是否将正在执行的命令与新命令交换 |
| START | 40:40 | 0x0 | 0x0-0x1 | 是否开始内存的初始化工作。需要在所有的参数配置完成之后，再设置该位，让内存进入初始化配置。在没有完成其它位的配置之前就配置该位，很可能导致内存访问错误。 |
| SREFRESH | 32:32 | 0x0 | 0x0-0x1 | 内存模块是否进入自刷新工作模式 |
| RW_SAME_EN | 24:24 | 0x0 | 0x0-0x1 | 在命令队列重排序逻辑中是否考虑对同一 bank 读写命令的重组 |
| REG_DIMM_EN | 16:16 | 0x0 | 0x0-0x1 | 是否使能 registered DIMM 内存模组 |
| REDUC | 8:8 | 0x0 | 0x0-0x1 | 是否只使用 32 位位宽的内存数据通道，通常情况下，不应设置该位 |
| PWRUP_SREFRESH_EXIT | 0:0 | 0x0 | 0x0-0x1 | 是用 self-refresh 命令而不是用正常的内存初始化命令来脱离下电模式 |
| CONF_CTL_04[63:0] | Offset: 0x40 | | DDR2 667：0x0102010100010101 | |
| RTT_0 | 57:56 | 0x0 | 0x0-0x3 | 使能内存模块的片上终端电阻。<br>00 –disable<br>其它–enable，电阻大小由 mrs_data 中的值决定 |
| CTRL_RAW | 49:48 | 0x0 | 0x0-0x3 | 设置 ECC 的检错和纠错模式<br>2’b00 – 不使用 ECC<br>2’b01 – 只报错，不纠错 |

| | | | | |
|---|---|---|---|---|
| | | | | 2'b10 – 没有使用 ECC 设备<br>2'b11 – 使用 ECC 报错纠错 |
| AXI0_W_PRIORITY | 41:40 | 0x0 | 0x0-0x3 | 设置 AXI0 端口写命令优先级 |
| AXI0_R_PRIORITY | 33:32 | 0x0 | 0x0-0x3 | 设置 AXI0 端口读命令优先级 |
| WRITE_MODEREG | 24:24 | 0x0 | 0x0-0x1 | 是否写内存模块的 EMRS 寄存器(只写)，每次写 1 时控制器就将配置参数中 emrs_data 与 mrs_data 发往内存。 |
| WRITEINTERP | 16:16 | 0x0 | 0x0-0x1 | 定义是否能用一个读命令取打断一个写突发 |
| TREF_ENABLE | 8:8 | 0x0 | 0x0-0x1 | 是否使能控制器内部的自动刷新功能，通常的情况下，应该将该位置 1 |
| TRAS_LOCKOUT | 0:0 | 0x0 | 0x0-0x1 | 是否在 tRAS 时间到期之前发出 auto-prechareg 命令 |
| CONF_CTL_05[63:0] | | Offset: 0x50 | | DDR2 667：0x0700000404050100 |
| Q_FULLNESS | 58:56 | 0x0 | 0x0-0x7 | 定义内存控制器命令队列中有多少命令时认为命令队列满 |
| PORT_DATA_ERROR_TYPE | 50:48 | 0x0 | 0x0-0x7 | 定义内存控制器端口上数据错误类型（只读）<br>位 0 – 突发数据个数大于 16<br>位 1 – 写数据交错<br>位 2 – ECC 2 位错 |
| OUT_OF_RANGE_TYPE | 42:40 | 0x0 | 0x0-0x7 | 定义发生越界访问时的错误类型（只读） |
| MAX_CS_REG | 34:32 | 0x4 | 0x0-0x4 | 定义控制器所用片选个数（只读） |
| COLUMN_SIZE | 26:24 | 0x0 | 0x0-0x7 | 设置实际列地址数和最大列地址数(14)之间的差值，应该根据具体的内存颗粒进行配置。<br>内存所用列地址数 = 14 - COLUMN_SIZE |
| CASLAT | 18:16 | 0x0 | 0x0-0x7 | 设置 CAS latency 值。应当根据具体的内存颗粒在不同的运行频率下进行配置。 |
| ADDR_PINS | 10:8 | 0x0 | 0x0-0x7 | 设置实际地址引脚数和最大地址数(15)之间的差值 |

| | | | | |
|---|---|---|---|---|
| | | | | 内存所用地址线数 = 15 – ADDR_PINS |
| CONF_CTL_06[63:0] | | Offset: 0x60 | | DDR2 667：0x0a04040603040003 |
| APREBIT | 59:56 | 0x0 | 0x0-0xf | 定义用哪位地址线向内存发出 autoprecharge 命令，一般为 bit 10。 |
| WRLAT | 50:48 | 0x0 | 0x0-0x7 | 写操作时写命令发出到接收到第一个数据的时间（按时钟周期数），同时决定何时使对应的 ODT 信号有效。<br>注：当 WRLAT = (CASLAT_LIN /2)时，会在不同 CS 读写之间加入一拍额外延迟。 |
| TWTR | 42:40 | 0x0 | 0x0-0x7 | 定义从写命令切换到读命令所需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TWR_INT | 34:32 | 0x0 | 0x0-0x7 | 定义内存模组的写恢复时间，需要根据具体内存颗粒及运行频率进行配置。 |
| TRTP | 26:24 | 0x0 | 0x0-0x7 | 定义内存模组的读命令到 precharge 周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TRRD | 18:16 | 0x0 | 0x0-0x7 | 定义到不同 bank 的 active 命令时间间隔，需要根据具体内存颗粒及运行频率进行配置。 |
| TCKE | 2:0 | 0x0 | 0x0-0x7 | 定义 CKE 信号最小脉宽 |
| CONF_CTL_07[63:0] | | Offset: 0x70 | | DDR2 667：0x0f0e0200000f0a0a |
| MAX_ROW_REG | 59:56 | 0xf | 0x0-0xf | 系统最大行地址个数（只读） |
| MAX_COL_REG | 51:48 | 0xe | 0x0-0xe | 系统最大列地址个数（只读） |
| INITAREF | 43:40 | 0x0 | 0x0-0xf | 定义系统初始化时所需要执行的 autorefresh 命令个数。DDR2 时设为 2，DDR3 时设为 0。 |
| CS_MAP | 19:16 | 0x0 | 0x0-0xf | 定义可用片选信号，本参数应当根据实际使用的片选个数进行正确的配置，不正确的配置将会导致错误的内存访问。该参数的四位分别对应于 CS0- CS3 |
| CASLAT_LIN | 3:0 | 0x0 | 0x0-0xf | 当板上走线延迟为 DDR2 时钟周期的 0.5~1.5 倍：CASLAT_LIN = CASLAT×2<br>小于 0.5 倍：CASLAT_LIN = CASLAT× |

| | | | | |
|---|---|---|---|---|
| | | | | 2-1<br>大于 1.5 倍：CASLAT_LIN = CASLAT×2+1<br>（以半个时钟周期为单位） |
| CONF_CTL_08[63:0] | | Offset: 0x80 | | DDR2 667：0x0804020108040201 |
| ODT_WR_MAP_CS3 | 59:56 | 0x0 | 0x0-0xf | 定义 CS3 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_WR_MAP_CS2 | 51:48 | 0x0 | 0x0-0xf | 定义 CS2 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_WR_MAP_CS1 | 43:40 | 0x0 | 0x0-0xf | 定义 CS1 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_WR_MAP_CS0 | 35:32 | 0x0 | 0x0-0xf | 定义 CS0 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_RD_MAP_CS3 | 27:24 | 0x0 | 0x0-0xf | 定义 CS3 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_RD_MAP_CS2 | 19:16 | 0x0 | 0x0-0xf | 定义 CS2 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_RD_MAP_CS1 | 11:8 | 0x0 | 0x0-0xf | 定义 CS1 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0- CS3 |
| ODT_RD_MAP_CS0 | 3:0 | 0x0 | 0x0-0xf | 定义 CS0 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要 |

| | | | | |
|---|---|---|---|---|
| | | | | 求。该参数的四位分别对应于 CS0- CS3 |
| CONF_CTL_09[63:0] | | Offset: 0x90 | | DDR2 667：0x0000070d00000000 |
| OCD_ADJUST_PUP_CS0 | 60:56 | 0x0 | 0x0-0x1f | 设置内存模组片选 0 OCD 上拉调整值。内存控制器将在初始化时根据这个参数的值向内存模组发出 OCD 调整命令 |
| OCD_ADJUST_PDN_CS0 | 52:48 | 0x0 | 0x0-0x1f | 设置内存模组片选 0 OCD 下拉调整值。内存控制器将在初始化时根据这个参数的值向内存模组发出 OCD 调整命令 |
| TRP | 43:40 | 0x0 | 0x0-0xf | 定义内存模组执行 pre-charge 所需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TDAL | 35:32 | 0x0 | 0x0-0xf | 当 auto-precharge 参数设置后，该参数定义了 auto-precharge 和 write recovery 时钟周期数。<br>TDAL = auto-precharge + write recovery<br>该参数仅在设置了 AP 之后才生效。 |
| PORT_CMD_ERROR_TYPE | 19:16 | 0x0 | 0x0-0xf | 端口上发生命令错误的类型（只读）<br>位 0 – 数据位宽过大<br>位 1 – 关键字优先操作地址未对齐<br>位 2 – 关键字优先操作字数不是 2 幂<br>位 3 – narrow transform 出错 |
| CONF_CTL_10[63:0] | | Offset: 0xa0 | | DDR2 667：0x0000003f3f140612 |
| COMMAND_AGE_COUNT | 37:32 | 0x0 | 0x0-0x3f | 定义命令队列重排序逻辑使用 aging 算法时每个命令的 aging 初始值 |
| AGE_COUNT | 29:24 | 0x0 | 0x0-0x3f | 定义命令队列重排序逻辑使用 aging 算法时每个命令的 aging 初始值 |
| TRC | 20:16 | 0x0 | 0x0-0x1f | 定义对内存模组同一 bank 的 active 命令之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TMRD | 12:8 | 0x0 | 0x0-0x1f | 定义配置内存模组模式寄存器需要的时钟周期数，通常为 2 个周期 |
| TFAW | 4:0 | 0x0 | 0x0-0x1f | 定义内存模组 tFAW 参数，8 个逻辑 bank 时使用 |
| | | | | |

| CONF_CTL_12[63:0] | Offset: 0xc0 | | DDR2 667：0x00002c0511000000 | |
|---|---|---|---|---|
| TRFC | 47:40 | 0x0 | 0x0-0xff | 定义内存模组刷新操作需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TRCD_INT | 39:32 | 0x0 | 0x0-0xff | 定义内存模组 RAS 到 CAS 之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TRAS_MIN | 31:24 | 0x0 | 0x0-0xff | 定义内存模组行地址有效命令的最小时钟周期数 |
| OUT_OF_RANGE_LENGTH | 23:16 | 0x0 | 0x0-0xff | 发生越界访问时的命令长度（只读） |
| ECC_U_SYND | 15:8 | 0x0 | 0x0-0xff | 发生 2bit 不可纠错误时的原因（只读） |
| ECC_C_SYND | 7:0 | 0x0 | 0x0-0xff | 发生 1bit 可纠错错误时的原因（只读） |
| | | | | |
| CONF_CTL_17[63:0] | Offset: 0x110 | | DDR2 667：0x0000000000000c2d | |
| TREF | 13:0 | 0x0 | 0x0-0x3ff | 定义内存模组两次刷新命令的时钟间隔，需要根据具体内存颗粒及运行频率进行配置。 |
| CONF_CTL_18[63:0] | Offset: 0x120 | | DDR2 667：0x001c000000000000 | |
| AXI0_EN_LT_WIDTH_INSTR | 63:48 | 0x0000 | 0x0-0xffff | 定义 AXI0 端口是否接收小于 64 位位宽的内存访问 |
| CONF_CTL_19[63:0] | Offset: 0x130 | | DDR2 667：0x6d56000302000000 | |
| TRAS_MAX | 63:48 | 0x0000 | 0x0-0xffff | 定义内存模组行有效命令的最大时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| TPDEX | 47:32 | 0x0000 | 0x0-0xffff | 定义内存模组掉电退出命令的时钟周期数 |
| TDLL | 31:16 | 0x0000 | 0x0-0xffff | 定义内存模组 DLL 锁定需要的时钟周期数 |
| TCPD | 15:0 | 0x0000 | 0x0-0xffff | 定义内存模组时钟有效到 precharge 之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。 |
| CONF_CTL_20[63:0] | Offset: 0x140 | | DDR2 667：0x0000204002000030 | |
| XOR_CHECK_BITS | 63:48 | 0x0000 | 0x0-0xffff | 当 fwc 参数设定时，下次写操作的 check bit 将会与该参数进行异或后写入内存 |

| | | | | |
|---|---|---|---|---|
| VERSION | 47:32 | 0x2040 | 0x2040 | 定义内存控制器版本号（只读） |
| TXSR | 31:16 | 0x0000 | 0x0-0xffff | 定义内存模组自刷新退出需要的时钟周期数 |
| TXSNR | 15:0 | 0x0000 | 0x0-0xffff | 定义内存模组 tXSNR 参数 |
| CONF_CTL_21[63:0] | Offset: 0x150 | | DDR2 667：0x0000000000000000 | |
| ECC_C_ADDR[36:8] | 60:32 | 0x0 | 0x0-0x1fffffff | 记录发生 1bit ECC 错误时的地址信息（只读） |
| ECC_C_ADDR[7:0] | 31:24 | 0x0000 | 0x0-0x1fffffff | 记录发生 1bit ECC 错误时的地址信息（只读） |
| TINIT | 23:0 | 0x0000 | 0x0-0xfffff | 定义内存模组初始化需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。一般为 200us。 |
| CONF_CTL_22[63:0] | Offset: 0x160 | | DDR2 667：0x0000000000000000 | |
| ECC_U_ADDR[36:32] | 36:32 | 0x0 | 0x0-0x1fffffff | 记录发生 2bit ECC 错误时的地址信息（只读） |
| ECC_U_ADDR[31:0] | 31:0 | 0x0 | 0x0-0x1fffffff | 记录发生 2bit ECC 错误时的地址信息（只读） |
| CONF_CTL_23[63:0] | Offset: 0x170 | | DDR2 667：0x0000000000000000 | |
| OUT_OF_RANGE_ADDR[36:32] | 36:32 | 0x0 | 0x0-0x1fffffff | 记录发生越界访问时的地址信息（只读） |
| OUT_OF_RANGE_ADDR[31:0] | 31:0 | 0x0 | 0x0-0x1fffffff | 记录发生越界访问时的地址信息（只读） |
| CONF_CTL_24[63:0] | Offset: 0x180 | | DDR2 667：0x0000000000000000 | |
| PORT_CMD_ERROR_ADDR[36:32] | 36:32 | 0x0 | 0x0-0x1fffffff | 记录端口发生命令错误时的地址信息（只读） |
| PORT_CMD_ERROR_ADDR[31:0] | 31:0 | 0x0 | 0x0-0x1fffffff | 记录端口发生命令错误时的地址信息（只读） |
| CONF_CTL_25[63:0] | Offset: 0x190 | | DDR2 667：0x0000000000000000 | |
| ECC_C_DATA[63:32] | 63:32 | 0x0 | 0x0-0x1fffffff | 记录发生 1bit ECC 错误时的数据信息（只读） |
| ECC_C_DATA[31:0] | 31:0 | 0x0 | 0x0-0x1fffffff | 记录发生 1bit ECC 错误时的数据信息（只读） |
| CONF_CTL_26[63:0] | Offset: 0x1a0 | | DDR2 667：0x0000000000000000 | |
| ECC_U_DATA[63:3 | 63:32 | 0x0 | 0x0-0x1fffff | 记录发生 2bit ECC 错误时的数据信息（只 |

| | | | | |
|---|---|---|---|---|
| 2] | | | ffff | 读） |
| ECC_U_DATA[31:0] | 31:0 | 0x0 | 0x0-0x1ffffffff | 记录发生 2bit ECC 错误时的数据信息（只读） |
| CONF_CTL_27[63:0] | Offset: 0x1b0 | | DDR2 667：0x0000000000000000 | |
| CKE_DELAY | 2:0 | 0x0 | 0x0-0x7 | CKE 有效延迟。<br>注：用于控制内部 srefresh_enter 命令的响应时间，对于龙芯 2 号无效。 |
| | | | | |
| CONF_CTL_29[63:0] | Offset: 0x1d0 | | DDR2 667：0x0103070400000101 | |
| TDFI_PHY_WRLAT_BASE | 59:56 | 0x0 | 0x0-0xf | 设置 DDR PHY 中写数据需加入的延迟。对于龙芯 2 号这个值应为 2 |
| TDFI_PHY_WRLAT | 51:48 | 0x0 | 0x0-0xf | 用于显示实际从写命令发出到写数据发出间隔的周期数（只读） |
| TDFI_PHY_RDLAT | 44:40 | 0x0 | 0x0-0xf | 设置读命令发出到读数据返回间隔的周期数 |
| TDFI_CTRLUPD_MIN | 35:32 | 0x4 | 0x0-0xf | 保存 DFI Tctrlup_min 时间参数（只读） |
| DRAM_CLK_DISABLE | 19:16 | 0x0 | 0x0-0xf | 设置是否输出 DRAM 时钟信号，每位对应一个片选信号。0：输出时钟信号；1：禁止输出时钟信号。 |
| ODT_ALT_EN | 8:8 | 0x0 | 0x0-0x1 | 是否支持 CAS＝3 时的 ODT 信号。<br>注：对于龙芯 2 号，无效 |
| DRIVE_DQ_DQS | 0:0 | 0x0 | 0x0-0x1 | 设置当控制器空闲时是否驱动数据总线 |
| CONF_CTL_30[63:0] | Offset: 0x1e0 | | DDR2 667：0x0c2d0c2d0c2d0205 | |
| TDFI_PHYUPD_TYPE0 | 61:48 | 0x0000 | 0x0-0x3fff | 这个值等于 TREF（只读） |
| TDFI_PHYUPD_RESP | 45:32 | 0x0000 | 0x0-0x3fff | 这个值等于 TREF（只读） |
| TDFI_CTRLUPD_MAX | 29:16 | 0x0000 | 0x0-0x3fff | 这个值等于 TREF（只读） |
| TDFI_RDDATA_EN_BASE | 12:8 | 0x00 | 0x0-0x1f | DDR PHY 内部读命令发出到读返回的基本时间。对于龙芯 2 号这个值为 2 |
| TDFI_RDDATA_EN | 4:0 | 0x00 | 0x0-0x1f | 用于显示从读命令发出到读数据返回的实际周期数 |

| CONF_CTL_31[63:0] | Offset: 0x1f0 | | DDR2 667：0x0020008000000000 | |
|---|---|---|---|---|
| DLL_CTRL_REG_0_0 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 0 数据组（DQ7-DQ0）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| DFT_CTRL_REG | 7:0 | 0x00 | 0x0-0xff | 测试使能信号，0x0 为正常工作模式 |
| CONF_CTL_32[63:0] | Offset: 0x200 | | DDR2 667：0x0020008000200080 | |
| DLL_CTRL_REG_0_2 | 63:32 | 0x000000 | 0x0-0xffffffff | 第 2 数据组（DQ23-DQ16）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| DLL_CTRL_REG_0_1 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 1 数据组（DQ15-DQ8）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| CONF_CTL_33[63:0] | Offset: 0x210 | | DDR2 667：0x0020008000200080 | |
| DLL_CTRL_REG_0_4 | 63:32 | 0x000000 | 0x0-0xffffffff | 第 4 数据组（DQ39-DQ32）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 |

| | | | | |
|---|---|---|---|---|
| | | | | DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| DLL_CTRL_REG_0_3 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 3 数据组（DQ31-DQ24）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| CONF_CTL_34[63:0] | Offset: 0x220 | | DDR2 667：0x0020008000200080 | |
| DLL_CTRL_REG_0_6 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 6 数据组（DQ55-DQ48）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8'h80 |
| DLL_CTRL_REG_0_5 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 5 数据组（DQ47-DQ40）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8'h20 |

| | | | | |
|---|---|---|---|---|
| | | | | 7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8’h80 |
| CONF_CTL_35[63:0] | | Offset: 0x230 | | DDR2 667：0x0020008000200080 |
| DLL_CTRL_REG_0_8 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 8 数据组（DQ71-DQ64）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8’h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8’h80 |
| DLL_CTRL_REG_0_7 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 7 数据组（DQ63-DQ56）DLL 控制信号<br>24： 控制内部 DLL 的使能信号，为 0 时 DLL 有效<br>23:16： 控制写数据（DQ）与 DQS 之间的相位关系，每个数值表示为 (1/精度) * 360。在龙芯 2 号中，这个值一般为 1/4，即 8’h20<br>7:0： 控制内部 DLL 的精度，在龙芯 2 号中，这个值一般为 8’h80 |
| CONF_CTL_36[63:0] | | Offset: 0x240 | | DDR2 667：0x00001e0000001e00 |
| DLL_CTRL_REG_1_1 | 63:32 | 0x0000 | 0x0-0xffffffff | 第 1 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| DLL_CTRL_REG_1_0 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 0 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| CONF_CTL_37[63:0] | | Offset: 0x250 | | DDR2 667：0x00001e0000001e00 |
| DLL_CTRL_REG_1_3 | 63:32 | 0x0000 | 0x0-0xffffffff | 第 3 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。 |

| | | | | |
|---|---|---|---|---|
| | | | | 5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| DLL_CTRL_REG_1_2 | 31:0 | 0x000000 | 0x0-0xffffffff | 第 2 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| CONF_CTL_38[63:0] | Offset: 0x260 | | DDR2 667：0x00001e0000001e00 | |
| DLL_CTRL_REG_1_5 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 5 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| DLL_CTRL_REG_1_4 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 4 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| CONF_CTL_39[63:0] | Offset: 0x270 | | DDR2 667：0x00001e0000001e00 | |
| DLL_CTRL_REG_1_7 | 63:32 | 0x0000 | 0x0-0xffffffff | 第 7 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0. |
| DLL_CTRL_REG_1_6 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 6 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| CONF_CTL_40[63:0] | Offset: 0x280 | | DDR2 667：0x0000000000001e00 | |
| DLL_OBS_REG_0_0 | 33:32 | 0x0 | 0x0-0x3 | 测试模式下的第 0 数据组 DLL 输出（只读） |
| DLL_CTRL_REG_1_8 | 31:0 | 0x000000 | 0x0-0xffffffff | 第 8 数据组 DLL 控制信号<br>15:8：读数据返回时，DQSn 的相位延迟。<br>5:0： DLL 测试控制信号，正常情况下为 8’h0 |
| CONF_CTL_41[63:0] | Offset: 0x290 | | DDR2 667：0x0000000000000000 | |
| DLL_OBS_REG_0_2 | 33:32 | 0x0 | 0x0-0x3 | 测试模式下的第 2 数据组 DLL 输出（只读） |

| | | | | |
|---|---|---|---|---|
| DLL_OBS_REG_0_1 | 1:0 | 0x0 | 0x0-0x3 | 测试模式下的第 1 数据组 DLL 输出（只读） |
| CONF_CTL_42[63:0] | Offset: 0x2a0 | | DDR2 667：0x0x0000000000000000 | |
| DLL_OBS_REG_0_4 | 33:32 | 0x0 | 0x0-0x3 | 测试模式下的第 4 数据组 DLL 输出（只读） |
| DLL_OBS_REG_0_3 | 1:0 | 0x0 | 0x0-0x3 | 测试模式下的第 3 数据组 DLL 输出（只读） |
| CONF_CTL_43[63:0] | Offset: 0x2b0 | | DDR2 667：0x0x0000000000000000 | |
| DLL_OBS_REG_0_6 | 33:32 | 0x0 | 0x0-0x3 | 测试模式下的第 6 数据组 DLL 输出（只读） |
| DLL_OBS_REG_0_5 | 1:0 | 0x0 | 0x0-0x3 | 测试模式下的第 5 数据组 DLL 输出（只读） |
| CONF_CTL_44[63:0] | Offset: 0x2c0 | | DDR2 667：0x0000000000000000 | |
| DLL_OBS_REG_0_8 | 33:32 | 0x0 | 0x0-0x3 | 测试模式下的第 8 数据组 DLL 输出（只读） |
| DLL_OBS_REG_0_7 | 1:0 | 0x0 | 0x0-0x3 | 测试模式下的第 7 数据组 DLL 输出（只读） |
| CONF_CTL_45[63:0] | Offset: 0x2d0 | | DDR2 667：0xf30029470000019d | |
| PHY_CTRL_REG_0_0 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 0 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17：写数据/DQS 延迟是否增加半周期延迟 |

| | | | | |
|---|---|---|---|---|
| | | | | 16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0： 写数据有效的结束时间 |
| PAD_CTRL_REG_0 | 25:0 | 0x0000 | 0x0-0x3ffffff | 引脚控制信号<br>25:22：对应 COMPZCP_dig<br>21:18：对应 COMPZCN_dig<br>17：对应引脚的 TQ1v8<br>16：对应内部反馈引脚的使能信号，低有效<br>15：对应内部反馈引脚的输出使能信号，低有效<br>14：对应数据选通引脚的输出使能信号，低有效<br>13：对应数据屏蔽引脚的输出使能信号，低有效<br>12：对应数据引脚的输出使能信号，低有效<br>11：对应引脚的 USEPAD<br>0：使用内部参考电压；<br>1：使用外部参考电压。<br>8：对应时钟引脚{1,3,5}的使能信号，高有效<br>7：对应时钟引脚{0,2,4}的使能信号，高有效<br>6：对应地址引脚的使能信号，低有效<br>5：对应引脚的 PROGB1v8<br>4：对应引脚的 PROGA1v8<br>用于控制引脚驱动能力<br>3：对应引脚的 ODTB<br>2：对应引脚的 ODTA |

<table>
<tr><td></td><td></td><td></td><td></td><td>用于控制引脚 ODT 阻值大小<br>ODTA ODTB DDRII DDRIII<br>1 0 150 120<br>1 1 75 60<br>0 0 Disable Disable<br>1：对应引脚的 MODEZI1v8<br>对于龙芯 2 号应该设为 0。<br>0：对应引脚的 DDR1v8<br>0：对应 DDRII 的 1.8v 模式<br>1：对应 DDRIII 的 1.5v 模式</td></tr>
<tr><td colspan="5">CONF_CTL_46[63:0] Offset: 0x2e0 DDR2 667：0xf3002947f3002947</td></tr>
<tr><td>PHY_CTRL_REG_0_2</td><td>63:32</td><td>0x00000000</td><td>0x0-0xffffffff</td><td>第 2 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17： 写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间</td></tr>
</table>

| | | | | |
|---|---|---|---|---|
| | | | | 2:0：写数据有效的结束时间 |
| PHY_CTRL_REG_0_1 | 31:0 | 0x00000000 | 0x0-0xffffffff | 第 1 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17：写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| CONF_CTL_47[63:0] | Offset: 0x2f0 | | DDR2 667：0xf3002947f3002947 | |
| PHY_CTRL_REG_0_4 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 4 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数 |

| | | | | |
|---|---|---|---|---|
| | | | | 据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17： 写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| PHY_CTRL_REG_0_3 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 3 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17： 写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开， |

| | | | | |
|---|---|---|---|---|
| | | | | 提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| CONF_CTL_48[63:0] | | Offset: 0x300 | | DDR2 667：0xf3002947f3002947 |
| PHY_CTRL_REG_0_6 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 6 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17：写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| PHY_CTRL_REG_0_5 | 31:0 | 0x00000000 | 0x0-0xffffffff | 第 5 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固 |

| | | | | |
|---|---|---|---|---|
| | | | | 定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17：写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| CONF_CTL_49[63:0] | Offset: 0x310 | | DDR2 667：0xf3002947f3002947 | |
| PHY_CTRL_REG_0_8 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 8 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步） |

| | | | | |
|---|---|---|---|---|
| | | | | 17： 写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| PHY_CTRL_REG_0_7 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 7 数据组时延控制.<br>28： 是否对读 DQS 使用去毛刺电路，指 gate 信号是否通过 PAD_feedback 延迟<br>27：使用读 FIFO 有效信号自动控制读数据返回采样（1），还是使用 26:24 中的固定时间采样（0）<br>26:24： 读数据返回采样完成时机，从内部时钟域采样的延迟。<br>21： 在 Read Leveling 模式下，采样数据总线的电平高低<br>20： 数据有效控制信号的电平，龙芯 2 号中为 0<br>19： 是否将写数据延迟再增加一周期<br>18： 读 DQS 采样是否提前 1/4 周期（与 clk_wr 同步）<br>17：写数据/DQS 延迟是否增加半周期延迟<br>16： CAS 延迟是否为半周期<br>15:12：写 DQS 有效的起始时间，对于 DDR3 应该比 DDR2 提前一个周期打开，提供颗粒要求的 Preamble DQS<br>11:8： 写 DQS 有效的结束时间<br>6:4：写数据有效的起始时间<br>2:0：写数据有效的结束时间 |
| CONF_CTL_50[63:0] | Offset: 0x320 | | DDR2 667：0x07c0000007c00000 | |

| | | | | |
|---|---|---|---|---|
| PHY_CTRL_REG_1_1 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 1 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| PHY_CTRL_REG_1_0 | 31:0 | 0x00000000 | 0x0-0xffffffff | 第 0 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效， |

| | | | | |
|---|---|---|---|---|
| | | | | 用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| CONF_CTL_51[63:0] | | Offset: 0x330 | | DDR2 667：0x07c0000007c00000 |
| PHY_CTRL_REG_1_3 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 3 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| PHY_CTRL_REG_1_2 | 31:0 | 0x00000000 | 0x0-0xffffffff | 第 2 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1） |

| | | | | |
|---|---|---|---|---|
| | | | | 21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| CONF_CTL_52[63:0] | | Offset: 0x340 | | DDR2 667：0x07c0000007c00000 |
| PHY_CTRL_REG_1_5 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 5 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| PHY_CTRL_REG_1_4 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 4 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1 |

| | | | | |
|---|---|---|---|---|
| | | | | 22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| CONF_CTL_53[63:0] | Offset: 0x350 | | DDR2 667：0x07c0000007c00000 | |
| PHY_CTRL_REG_1_7 | 63:32 | 0x00000000 | 0x0-0xffffffff | 第 7 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期<br>27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| PHY_CTRL_REG_1_6 | 31:0 | 0x00000000 | 0x0-0xffffffff | 第 6 数据组中 PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期 |

| | | | | |
|---|---|---|---|---|
| | | | | 27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| CONF_CTL_54[63:0] | | Offset: 0x360 | | DDR2 667：0x0800c00507c00000 |
| PHY_CTRL_REG_2 | 63:32 | 0x00000 | 0x0-0xffffffff | 读写数据延迟控制<br>27：选择读数据缓冲类型，默认为 0<br>26：用于清除读返回缓冲区的数据，正常为 0<br>25：高速引脚使能，为 1 时，所有信号通过引脚向外传输的延迟减小 1 周期<br>16:13：设置读数据有效时机，从 FIFO 中收集数据返回控制器的延迟。如果从引脚到 FIFO 的延迟增加，这个值也必须增加<br>8：设置 DQS 信号输出是否为 DDR3 模式，DDR3 模式下，写 DQS 的 Preamble 将含有一个脉冲<br>5：测试模式信号，正常为 0<br>4：测试模式信号，正常为 0 |
| PHY_CTRL_REG_1_8 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 8 数据组中的终端电阻控制<br>PAD 的终端电阻控制，发起读操作时，才会启用<br>31:28：终端电阻关闭时机控制，每个值表示半周期 |

| | | | | |
|---|---|---|---|---|
| | | | | 27:24：终端电阻开启时机控制，从发送读命令后 4 拍开始计算<br>23：终端电阻的有效电平控制，对于龙芯 2 号应为 1<br>22：终端电阻的使能信号，为 1 时，使用动态方式控制终端电阻的使能；为 0 时，可以通过第 23 位 PAD 上的终端电阻永远有效（置 0）或永远无效（置 1）<br>21：测试用信号，正常应为 0<br>20:16：测试用信号，正常应为 0<br>14:12：测试用信号，正常应为 0<br>11:8：读采样延时 1，其中只能 1 位有效，用于控制读 DQS 采样窗口关闭时机<br>7:0：读采样延时 0，其中只能 1 位有效，用于控制读 DQS 采样窗口打开时机 |
| CONF_CTL_55[63:0] | Offset: 0x370 | | DDR2 667：0x0000000000000000 | |
| PHY_OBS_REG_0_1 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 1 数据组测试用观测信号（只读） |
| PHY_OBS_REG_0_0 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 0 数据组测试用观测信号（只读） |
| CONF_CTL_56[63:0] | Offset: 0x380 | | DDR2 667：0x0000000000000000 | |
| PHY_OBS_REG_0_3 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 3 数据组测试用观测信号（只读） |
| PHY_OBS_REG_0_2 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 2 数据组测试用观测信号（只读） |
| CONF_CTL_57[63:0] | Offset: 0x390 | | DDR2 667：0x0000000000000000 | |
| PHY_OBS_REG_0_5 | 63:32 | 0x000000 | 0x0-0xffffffff | 第 5 数据组测试用观测信号（只读） |
| PHY_OBS_REG_0_4 | 31:0 | 0x0000 | 0x0-0xffffffff | 第 4 数据组测试用观测信号（只读） |
| CONF_CTL_58[63:0] | Offset: 0x2G0 | | DDR2 667：0x0000000000000000 | |
| PHY_OBS_REG_0_7 | 63:32 | 0x00000 | 0x0-0xffffffff | 第 7 数据组测试用观测信号（只读） |
| PHY_OBS_REG_0_6 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 6 数据组测试用观测信号（只读） |

| | | | | |
|---|---|---|---|---|
| CONF_CTL_59[63:0] | Offset: 0x3b0 | | DDR2 667：0x0000000000000000 | |
| PHY_OBS_REG_0_8 | 31:0 | 0x00000 | 0x0-0xffffffff | 第 8 数据组测试用观测信号（只读） |
| | | | | |
| CONF_CTL_114[63:0] | Offset: 0x720 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_REQ | 56 | 0x0 | 0x0-0x1 | 用户请求读选通采样训练功能.（只写） |
| RDLVL_GATE_PREAMBLE_CHECK_EN | 48 | 0x0 | 0x0-0x1 | 开启读选通采样训练时的前导采样检查 |
| RDLVL_GATE_EN | 40 | 0x0 | 0x0-0x1 | 使能 Read Leveling 时读选通采样训练，在初始化完成后会向颗粒发送命令，进行读 DQS 采样窗口的训练 |
| RDLVL_EN | 32 | 0x0 | 0x0-0x1 | 使能 Read Leveling 功能 |
| RDLVL_BEGIN_DELAY_EN | 24 | 0x0 | 0x0-0x1 | 使能 Read Leveling 寻找数据采样点功能 |
| SWLVL_OP_DONE | 8 | 0x0 | 0x0-0x1 | 用于指示软件 Leveling 是否完成（只读） |
| CONF_CTL_115[63:0] | Offset: 0x730 | | DDR2 667：0x0000000000000000 | |
| RDLVL_OFFSET_DIR_7 | 56 | 0x0 | 0x0-0x1 | 第 7 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_6 | 48 | 0x0 | 0x0-0x1 | 第 6 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_5 | 40 | 0x0 | 0x0-0x1 | 第 5 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_4 | 32 | 0x0 | 0x0-0x1 | 第 4 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_3 | 24 | 0x0 | 0x0-0x1 | 第 3 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_2 | 16 | 0x0 | 0x0-0x1 | 第 2 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 |

| | | | | |
|---|---|---|---|---|
| | | | | rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_1 | 8 | 0x0 | 0x0-0x1 | 第 1 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| RDLVL_OFFSET_DIR_0 | 0 | 0x0 | 0x0-0x1 | 第 0 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| CONF_CTL_116[63:0] | Offset: 0x740 | | DDR2 667：0x0100000000000000 | |
| AXI1_PORT_ORDERING | 57:56 | 0x0 | 0x0-0x3 | 内部端口 1 是否可乱序执行，对于龙芯 2 号无效 |
| AXI0_PORT_ORDERING | 49:48 | 0x0 | 0x0-0x3 | 内部端口 0 是否可乱序执行 |
| WRLVL_REQ | 40 | 0x0 | 0x0-0x1 | 用户请求开始 Write Leveling 训练功能.（只写） |
| WRLVL_INTERVAL_CT_EN | 32 | 0x0 | 0x0-0x1 | 使能 Write Leveling 时间间隔功能 |
| WEIGHTED_ROUND_ROBIN_WEIGHT_SHARING | 24 | 0x0 | 0x0-0x1 | Per-port pair shared arbitration for WRR |
| WEIGHTED_ROUND_ROBIN_LATENCY_ CONTROL | 16 | 0x0 | 0x0-0x1 | Free-running or limited WRR latency counters. |
| RDLVL_REQ | 8 | 0x0 | 0x0-0x1 | 用户请求开始 Read Leveling 训练功能.（只写） |
| RDLVL_OFFSET_DIR_8 | 0 | 0x0 | 0x0-0x1 | 第 8 数据组 Read Leveling 时中点的调整方向。为 0 时，中点计算为减去 rdlvl_offset_delay，为 1 则加。 |
| CONF_CTL_117[63:0] | Offset: 0x750 | | DDR2 667：0x0100000101020101 | |
| WRLVL_CS | 57:56 | 0x0 | 0x0-0x3 | 指示当前 Write Leveling 操作的片选信号 |
| SW_LEVELING_MODE | 49:48 | 0x0 | 0x0-0x3 | 定义软件 Leveling 操作的模式 |
| RDLVL_CS | 41:40 | 0x0 | 0x0-0x3 | 指示当前 Read Leveling 操作的片选信号 |
| AXI2_W_PRIORITY | 33:32 | 0x0 | 0x0-0x3 | 内部端口 2 的写操作优先级，对于龙芯 2 号无效 |
| AXI2_R_PRIORITY | 25:24 | 0x0 | 0x0-0x3 | 内部端口 2 的读操作优先级，对于龙芯 2 |

| | | | | |
|---|---|---|---|---|
| | | | | 号无效 |
| AXI2_PORT_ORDERING | 17:16 | 0x0 | 0x0-0x3 | 内部端口 2 是否可乱序执行，对于龙芯 2 号无效 |
| AXI1_W_PRIORITY | 9:8 | 0x0 | 0x0-0x3 | 内部端口 1 的写操作优先级，对于龙芯 2 号无效 |
| AXI1_R_PRIORITY | 1:0 | 0x0 | 0x0-0x3 | 内部端口 1 的读操作优先级，对于龙芯 2 号无效 |
| CONF_CTL_118[63:0] | | Offset: 0x760 | | DDR2 667：0x0303030000020002 |
| AXI0_PRIORITY2_RELATIVE_PRIORITY | 59:56 | 0x0 | 0x0-0xf | 内部端口 0 优先级 2 的命令的相对优先级 |
| AXI0_PRIORITY1_RELATIVE_PRIORITY | 51:48 | 0x0 | 0x0-0xf | 内部端口 0 优先级 1 的命令的相对优先级 |
| AXI0_PRIORITY0_RELATIVE_PRIORITY | 43:40 | 0x0 | 0x0-0xf | 内部端口 0 优先级 0 的命令的相对优先级 |
| ADDRESS_MIRRORING | 35:32 | 0x0 | 0x0-0xf | 指示哪个片选支持 Address mirroring 功能 |
| TDFI_DRAM_CLK_DISABLE | 26:24 | 0x0 | 0x0-0x7 | 从内部时钟关闭到外部时钟关闭的延迟设置 |
| BSTLEN | 18:16 | 0x0 | 0x0-0x7 | 设置控制器上向内存模块发送的 Burst 长度值 |
| ZQ_REQ | 9:8 | 0x0 | 0x0-0x3 | 用户请求开始 ZQ 调整功能 |
| ZQ_ON_SREF_EXIT | 1:0 | 0x0 | 0x0-0x3 | 定义在退出自刷新模式时 ZQ 调整功能的模式 |
| CONF_CTL_119[63:0] | | Offset: 0x770 | | DDR2 667：0x0101010202020203 |
| AXI2_PRIORITY2_RELATIVE_PRIORITY | 59:56 | 0x0 | 0x0-0xf | 内部端口 2 优先级 2 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI2_PRIORITY1_RELATIVE_PRIORITY | 51:48 | 0x0 | 0x0-0xf | 内部端口 2 优先级 1 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI2_PRIORITY0_RELATIVE_PRIORI | 43:40 | 0x0 | 0x0-0xf | 内部端口 2 优先级 0 的命令的相对优先级，对于龙芯 2 号无效 |

| | | | | |
|---|---|---|---|---|
| TY | | | | |
| AXI1_PRIORITY3_RELATIVE_PRIORITY | 35:32 | 0x0 | 0x0-0xf | 内部端口 1 优先级 3 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI1_PRIORITY2_RELATIVE_PRIORITY | 27:24 | 0x0 | 0x0-0xf | 内部端口 1 优先级 2 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI1_PRIORITY1_RELATIVE_PRIORITY | 19:16 | 0x0 | 0x0-0xf | 内部端口 1 优先级 1 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI1_PRIORITY0_RELATIVE_PRIORITY | 11:8 | 0x0 | 0x0-0xf | 内部端口 1 优先级 0 的命令的相对优先级，对于龙芯 2 号无效 |
| AXI0_PRIORITY3_RELATIVE_PRIORITY | 3:0 | 0x0 | 0x0-0xf | 内部端口 0 优先级 3 的命令的相对优先级 |
| CONF_CTL_120[63:0] | | Offset: 0x780 | | DDR2 667：0x0102020400040c01 |
| TDFI_DRAM_CLK_ENABLE | 59:56 | 0x0 | 0x0-0xf | 从内部时钟有效到输出时钟有效的延迟 |
| TDFI_CTRL_DELAY | 51:48 | 0x0 | 0x0-0xf | 从时钟有效到输出命令之间的延迟 |
| RDLVL_GATE_DQ_ZERO_COUNT | 43:40 | 0x0 | 0x0-0xf | 设置读选通采样训练时，表求由 1 到 0 的 0 的个数 |
| RDLVL_DQ_ZERO_COUNT | 35:32 | 0x0 | 0x0-0xf | 设置读 Read Leveling 时，表求由 1 到 0 的 0 的个数 |
| LOWPOWER_REFRESH_ENABLE | 27:24 | 0x0 | 0x0-0xf | 使能低功耗模式下的刷新功能 |
| DRAM_CLASS | 19:16 | 0x0 | 0x0-0xf | 定义控制器外接内存类型<br>110：DDR3<br>100：DDR2 |
| BURST_ON_FLY_BIT | 11:8 | 0x0 | 0x0-0xf | 对 DRAM 发出的模式配置中的 burst-on-fly 位 |
| AXI2_PRIORITY3_RELATIVE_PRIORITY | 3:0 | 0x0 | 0x0-0xf | 内部端口 2 优先级 3 的命令的相对优先级，对于龙芯 2 号无效 |

| CONF_CTL_121[63:0] | Offset: 0x790 | | DDR2 667：0x281900000f000303 | |
|---|---|---|---|---|
| WLMRD | 61:56 | 0x00 | 0x0-0x3f | 从对 DRAM 发送模式配置到 Write Leveling 的延迟 |
| WLDQSEN | 53:48 | 0x00 | 0x0-0x3f | 从对 DRAM 发送模式配置到 Write Leveling 的的选通数据采样延迟 |
| LOWPOWER_CONTROL | 44:40 | 0x00 | 0x0-0x1f | 低功耗模式使能<br>Bit 4: power down<br>Bit 3: power down external<br>Bit 2: self refresh<br>Bit 1: external<br>Bit 0: internal |
| LOWPOWER_AUTO_ENABLE | 36:32 | 0x00 | 0x0-0x1f | 使能当控制器内闲时自动进入低功耗模式<br>控制位与 LOWERPOWER_CONTROL 相同 |
| ZQCS_CHIP | 27:24 | 0x0 | 0x0-0xf | 定义下次 ZQ 时的有效片选 |
| WRR_PARAM_VALUE_ERR | 19:16 | 0x0 | 0x0-0xf | Errors/warnings related to the WRR parameters. （只读） |
| TDFI_WRLVL_DLL | 15:8 | 0x00 | 0x0-0xff | 读操作到 Write Leveling 更新延迟线数目的最小周期 |
| TDFI_RDLVL_DLL | 7:0 | 0x00 | 0x0-0xff | 读操作到 Read Leveling 更新延迟线数目的最小周期 |
| CONF_CTL_122[63:0] | Offset: 0x7a0 | | DDR2 667：0x0000000000000000 | |
| SWLVL_RESP_6 | 63:56 | 0x00 | 0x0-0xff | 第 6 数据组的 Leveling 响应 |
| SWLVL_RESP_5 | 55:48 | 0x00 | 0x0-0xff | 第 5 数据组的 Leveling 响应 |
| SWLVL_RESP_4 | 47:40 | 0x00 | 0x0-0xff | 第 4 数据组的 Leveling 响应 |
| SWLVL_RESP_3 | 39:32 | 0x00 | 0x0-0xff | 第 3 数据组的 Leveling 响应 |
| SWLVL_RESP_2 | 31:24 | 0x00 | 0x0-0xff | 第 2 数据组的 Leveling 响应 |
| SWLVL_RESP_1 | 23:16 | 0x00 | 0x0-0xff | 第 1 数据组的 Leveling 响应 |
| SWLVL_RESP_0 | 15:8 | 0x00 | 0x0-0xff | 第 0 数据组的 Leveling 响应 |
| CONF_CTL_123[63:0] | Offset: 0x7b0 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | 63:16 | | | |
| SWLVL_RESP_8 | 15:8 | 0x00 | 0x0-0xff | 第 8 数据组的 Leveling 响应 |

| | | | | |
|---|---|---|---|---|
| SWLVL_RESP_7 | 7:0 | 0x00 | 0x0-0xff | 第 7 数据组的 Leveling 响应 |
| CONF_CTL_124[63:0] | Offset: 0x7c0 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_125[63:0] | Offset: 0x7d0 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_CLK_ADJUST_3 | 63:56 | 0x00 | 0x0-0xff | 第 3 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_2 | 55:48 | 0x00 | 0x0-0xff | 第 2 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_1 | 47:40 | 0x00 | 0x0-0xff | 第 1 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_0 | 39:32 | 0x00 | 0x0-0xff | 第 0 数据组中，读采样训练的起始值 |
| CONF_CTL_126[63:0] | Offset: 0x7e0 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_CLK_ADJUST_8 | 39:32 | 0x00 | 0x0-0xff | 第 8 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_7 | 31:24 | 0x00 | 0x0-0xff | 第 7 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_6 | 23:16 | 0x00 | 0x0-0xff | 第 6 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_5 | 15:8 | 0x00 | 0x0-0xff | 第 5 数据组中，读采样训练的起始值 |
| RDLVL_GATE_CLK_ADJUST_4 | 7:0 | 0x00 | 0x0-0xff | 第 4 数据组中，读采样训练的起始值 |
| CONF_CTL_127[63:0] | Offset: 0x7f0 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_128[63:0] | Offset: 0x800 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_129[63:0] | Offset: 0x810 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_130[63:0] | Offset: 0x820 | | DDR2 667：0x0420000c20400000 | |
| TDFI_WRLVL_RESPLAT | 63:56 | 0x00 | 0x0-0xff | Write Leveling 选通到响应有效的周期数 |
| TDFI_RDLVL_RESPLAT | 39:32 | 0x00 | 0x0-0xff | Read Leveling 选通到响应有效的周期数 |

| | | | | |
|---|---|---|---|---|
| REFRESH_PER_ZQ | 23:16 | 0x00 | 0x0-0xff | 自动 ZQCS 命令之间的刷新命令数目 |
| CONF_CTL_131[63:0] | Offset: 0x830 | | DDR2 667：0x0000000000000c0a | |
| TMOD | 15:8 | 0x00 | 0x0-0xff | DRAM 模式配置后需空闲的周期数 |
| CONF_CTL_132[63:0] | Offset: 0x840 | | DDR2 667：0x0000640064000000 | |
| AXI1_PRIORITY_RELAX | 49:40 | 0x000 | 0x0-0x3ff | 内部端口 1 上触发优先控制放松的计数器值，对于龙芯 2 号无效 |
| AXI0_PRIORITY_RELAX [9:8] | 33:32 | 0x0 | 0x0-0x3 | 内部端口 0 上触发优先控制放松的计数器值 |
| AXI0_PRIORITY_RELAX [7:0] | 31:24 | 0x00 | 0x0-0xff | 内部端口 0 上触发优先控制放松的计数器值 |
| CONF_CTL_133[63:0] | Offset: 0x850 | | DDR2 667：0x0000000000000064 | |
| OUT_OF_RANGE_SOURCE_ID | 57:48 | 0x000 | 0x0-0x3ff | 访问地址溢出请求的 ID 号（只读） |
| ECC_U_ID | 41:32 | 0x000 | 0x0-0x3ff | 访问出现 2 位错请求的 ID 号（只读） |
| ECC_C_ID | 25:16 | 0x000 | 0x0-0x3ff | 访问出现 1 位错请求的 ID 号（只读） |
| AXI2_PRIORITY_RELAX | 9:0 | 0x000 | 0x0-0x3ff | 内部端口 2 上触发优先控制放松的计数器值，对于龙芯 2 号无效 |
| CONF_CTL_134[63:0] | Offset: 0x860 | | DDR2 667：0x0000004000000000 | |
| ZQCS | 43:32 | 0x000 | 0x0-0xfff | ZQCS 命令需要的周期数 |
| PORT_DATA_ERROR_ID | 25:16 | 0x000 | 0x0-0x3ff | 内部端口数据错请求的 ID 号（只读） |
| PORT_CMD_ERROR_ID | 9:0 | 0x000 | 0x0-0x3ff | 内部端口命令错请求的 ID 号（只读） |
| CONF_CTL_135[63:0] | Offset: 0x870 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_136[63:0] | Offset: 0x880 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_137[63:0] | Offset: 0x890 | | DDR2 667：0x0000000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_138[63:0] | Offset: 0x8a0 | | DDR2 667：0x00000000001c001c | |
| LOWPOWER_INTERNAL_CNT | 63:48 | 0x0000 | 0x0-0xffff | Counts idle cycles to self-refresh with memory and controller clk gating. |

| | | | | |
|---|---|---|---|---|
| LOWPOWER_EXTERNAL_CNT | 47:32 | 0x0000 | 0x0-0xffff | Counts idle cycles to self-refresh with memory clock gating. |
| AXI2_EN_SIZE_LT_WIDTH_INSTR | 31:16 | 0x0000 | 0x0-0xffff | 使能内部端口 2 上的各种窄访问，对于龙芯 2 号无效 |
| AXI1_EN_SIZE_LT_WIDTH_INSTR | 15:0 | 0x0000 | 0x0-0xffff | 使能内部端口 1 上的各种窄访问，对于龙芯 2 号无效 |
| CONF_CTL_139[63:0] | Offset: 0x8b0 | | DDR2 667：0x0000000000000000 | |
| LOWPOWER_POWER_DOWN_CNT | 15:0 | 0x0000 | 0x0-0xffff | 进入 Power Down 模式前的空闲周期数 |
| LOWPOWER_REFRESH_HOLD | 31:16 | 0x0000 | 0x0-0xffff | 在时钟门控模式下，内存控制器 re-lock DLL 前的空闲周期数 |
| LOWPOWER_SELF_REFRESH_CNT | 47:32 | 0x0000 | 0x0-0xffff | 进入内存自刷新模式前的空闲周期数 |
| CONF_CTL_140[63:0] | Offset: 0x8c0 | | DDR2 667：0x0004000000000000 | |
| OBSOLETE | | | | |
| CONF_CTL_141[63:0] | Offset: 0x8d0 | | DDR2 667：0x00000000c8000000 | |
| CKE_INACTIVE[31:8] | 55:32 | 0x0000000 | 0x0-0xffffff | 从输出 DDR_RESET 有效到 CKE 有效的时间间隔高位 |
| CKE_INACTIVE [7:0] | 31:24 | 0x00 | 0x0-0xff | 从输出 DDR_RESET 有效到 CKE 有效的时间间隔低位 |
| WRLVL_STATUS | 17:0 | 0x00000 | 0x0-0x3ffff | 最近一次 Write Leveling 操作状态（只读） |
| CONF_CTL_142[63:0] | Offset: 0x8e0 | | DDR2 667：0x0000000000000050 | |
| TRST_PWRON | 31:0 | 0x0000000 | 0x0-0xffffff | 从 start 有效 500 拍后到 DDR_RESET 有效之间的延迟 |
| CONF_CTL_143[63:0] | Offset: 0x8f0 | | DDR2 667：0x0000000020202080 | |
| DLL_CTRL_REG_2 [32] | 32:32 | 0x0 | 0x0-0x1 | 输出时钟 DLL 使能信号，高有效 |
| DLL_CTRL_REG_2 [31:0] | 31:0 | 0x0000000 | 0x0-0xffffff | 输出时钟 DLL 控制<br>31:24：输出时钟 DLL 上 CLK4 与 CLK5 的延迟<br>23:16：输出时钟 DLL 上 CLK2 与 CLK3 的延迟<br>15:8：输出时钟 DLL 上 CLK0 与 CLK1 的延迟 |

| | | | | |
|---|---|---|---|---|
| | | | | 7:0：输出时钟 DLL 上精度值 |
| CONF_CTL_144[63:0] | | Offset: 0x900 | | DDR2 667：0x0000000000000000 |
| RDLVL_ERROR_STATUS[37:32] | 37:32 | 0x00 | 0x0-0x3f | 指示 RDLVL 发生错误时的状态 |
| RDLVL_ERROR_STATUS[31:0] | 31:0 | 0x00000000 | 0x0-0xffffffff | 指示 RDLVL 发生错误时的状态 |
| CONF_CTL_145[63:0] | | Offset: 0x910 | | DDR2 667：0x0000000000000000 |
| RDLVL_GATE_RESP_MASK[63:32] | 63:32 | 0x00000000 | 0x0-0xffffffff | 采样训练中读返回屏蔽 |
| RDLVL_GATE_RESP_MASK[31:0] | 31:0 | 0x00000000 | 0x0-0xffffffff | 采样训练中读返回屏蔽 |
| CONF_CTL_146[63:0] | | Offset: 0x920 | | DDR2 667：0x0000000000000000 |
| RDLVL_GATE_RESP_MASK[71:64] | 7:0 | 0x00 | 0x0-0xff | 采样训练中读返回屏蔽 |
| CONF_CTL_147[63:0] | | Offset: 0x930 | | DDR2 667：0x0000000000000000 |
| RDLVL_RESP_MASK[63:32] | 63:32 | 0x00000000 | 0x0-0xffffffff | Read Leveling 中读返回屏蔽 |
| RDLVL_RESP_MASK[31:0] | 31:0 | 0x00000000 | 0x0-0xffffffff | Read Leveling 中读返回屏蔽 |
| CONF_CTL_148[63:0] | | Offset: 0x940 | | DDR2 667：0x0301010000050500 |
| TDFI_RDLVL_EN | 59:56 | 0x0 | 0x0-0xf | Read Leveling 使能到 Read Leveling 读之间的最小周期数 |
| W2R_SAMECS_DLY | 50:48 | 0x0 | 0x0-0x7 | 对同一个片选信号，写到读之间的附加延迟 |
| W2R_DIFFCS_DLY | 42:40 | 0x0 | 0x0-0x7 | 对不同片选信号，写到读之间的附加延迟 |
| LVL_STATUS | 34:32 | 0x0 | 0x0-0x7 | Write Leveling，Read Leveling 与采样训练请求的状态，用于 LVL_REQ 中断（只读） |
| RDLVL_EDGE | 24 | 0x0 | 0x0-0x1 | Read Leveling 操作中，指明 DQS 上升沿有效或下降沿有效 |
| CKSRX | 19:16 | 0x0 | 0x0-0x0 | 退出自刷新模式的时钟周期延迟 |
| CKSRE | 11:8 | 0x0 | 0x0-0x0 | 进入自刷新模式的时钟周期延迟 |
| RDLVL_RESP_MA | 7:0 | 0x00 | 0x0-0xff | Read Leveling 中读返回屏蔽 |

| | | | | |
|---|---|---|---|---|
| SK[71:64] | | | | |
| CONF_CTL_149[63:0] | | Offset: 0x950 | | DDR2 667：0x0000000000000a03 |
| INT_MASK[17:0] | 41:24 | 0x00 | 0x0-0x3ffff | 中断屏蔽 |
| TXPDLL | 23:8 | 0x0000 | 0x0-0xffff | DRAM TXPDLL parameter in cycles. |
| TDFI_WRLVL_EN | 3:0 | 0x0 | 0x0-0xf | Write Leveling 使能到 Write Leveling 读操作最小周期数 |
| CONF_CTL_150[63:0] | | Offset: 0x960 | | DDR2 667：0x0604000000000000 |
| RDLAT_ADJ | 60:56 | 0x00 | 0x0-0x1f | PHY 读延迟周期 |
| WRLAT_ADJ | 51:48 | 0x0 | 0x0-0xf | PHY 写延迟周期 |
| SWLVL_START | 40 | 0x0 | 0x0-0x1 | 软件 Leveling 模式下开始操作（只写） |
| SWLVL_LOAD | 32 | 0x0 | 0x0-0x1 | 软件 Leveling 模式下装入操作（只写） |
| SWLVL_EXIT | 24 | 0x0 | 0x0-0x1 | 软件 Leveling 模式下退出操作（只写） |
| INT_STATUS | 18:0 | 0x00 | 0x0-0x7ffff | 中断状态（只读） |
| CONF_CTL_151[63:0] | | Offset: 0x970 | | DDR2 667：0x000000000003e805 |
| CONCURRENTAP_WR_ONLY | 56 | 0x0 | 0x0-0x1 | 写操作之后读操作之前是否通过等待写恢复时间来阻止并发的 auto-precharge 操作 |
| CKE_STATUS | 48 | 0x0 | 0x0-0x1 | 指示 CKE_STATUS（只读） |
| INT_ACK [16:0] | 40:24 | 0x00 | 0x0-0x1ffff | 中断清除（只写） |
| DLL_RST_DELAY | 23:8 | 0x0000 | 0x0-0xffff | DLL 复位最小周期数 |
| DLL_RST_ADJ_DLY | 7:0 | 0x00 | 0x0-0xff | 配置 DLL 精度到 DLL 复位结束的最小周期数 |
| CONF_CTL_152[63:0] | | Offset: 0x980 | | DDR2 667：0x0001010001000101 |
| ZQ_IN_PROGRESS | 56 | 0x0 | 0x0-0x1 | 指示 ZQ 操作正在进行（只读） |
| ZQCS_ROTATE | 48 | 0x0 | 0x0-0x1 | 使能 ZQCS（short ZQ）轮流校正。当该位为 0 时，每次 ZQCS 请求命令会对系统中所有的片选进行校正，当该位置为 1 时，系统在每个 ZQCS 命令到来时只对一个片选进行校正，系统会轮流校正所有的片选。ZQCS 和 REFRESH_PER_ZQ 参数的设置应该与该位一致 |
| WRLVL_REG_EN | 40 | 0x0 | 0x0-0x1 | 使能写 wrlvl_delay 寄存器 |
| WRLVL_EN | 32 | 0x0 | 0x0-0x1 | 使能控制器的 Write Leveling 功能 |

| | | | | |
|---|---|---|---|---|
| RESYNC_DLL_PER_AREF_EN | 24 | 0x0 | 0x0-0x1 | 使能在每个刷新命令之后 DLL 自动同步 |
| RESYNC_DLL | 16 | 0x0 | 0x0-0x1 | 发起一个 DLL 同步命令（只写） |
| RDLVL_REG_EN | 8 | 0x0 | 0x0-0x1 | 使能写 rdlvl_delay 寄存器 |
| RDLVL_GATE_REG_EN | 0 | 0x0 | 0x0-0x1 | 使能写 rdlvl_gate_delay 寄存器 |
| CONF_CTL_153[63:0] | Offset: 0x990 | | DDR2 667：0x0101020202010100 | |
| W2W_SAMECS_DLY | 58:56 | 0x0 | 0x0-0x7 | 对同一个片选的写命令到写命令的附加延时时钟周期数 |
| W2W_DIFFCS_DLY | 50:48 | 0x0 | 0x0-0x7 | 对不同片选的写命令到写命令的附加延时时钟周期数 |
| TBST_INT_INTERVAL | 42:40 | 0x0 | 0x0-0x7 | DRAM burst 中断间隔周期数 |
| R2W_SAMECS_DLY | 34:32 | 0x0 | 0x0-0x7 | 对同一个片选的读命令到写命令的附加延时时钟周期数 |
| R2W_DIFFCS_DLY | 26:24 | 0x0 | 0x0-0x7 | 对不同片选的读命令到写命令的附加延时时钟周期数 |
| R2R_SAMECS_DLY | 18:16 | 0x0 | 0x0-0x7 | 对同一个片选的读命令到读命令的附加延时时钟周期数 |
| R2R_DIFFCS_DLY | 10:8 | 0x0 | 0x0-0x7 | 对不同片选的读命令到读命令的附加延时时钟周期数 |
| AXI_ALIGNED_STROBE_DISABLE | 2:0 | 0x0 | 0x0-0x7 | 当 AXI 端口的事务具有以下特征之一时：一、事务的起始地址和结束地址按用户字对齐；二、事务长度为一个用户字（128 位），禁止 AXI Strobe，每位对应一个 AXI 端口。<br>当设为 0 时，写操作会按照读-修改-写的顺序执行；<br>当设为 1 时，写操作作为一个标准的写操作（非读-修改-写的顺序） |
| CONF_CTL_154[63:0] | Offset: 0x9a0 | | DDR2 667：0x0707040200060100 | |
| TDFI_WRLVL_LOAD | 63:56 | 0x0 | 0x0-0xff | 写 Leveling 延时数有效到第一个写 Leveling Load 命令的最小时钟周期数 |
| TDFI_RDLVL_LOAD | 55:48 | 0x0 | 0x0-0xff | 读 Leveling 延时数有效到第一个读 Leveling Load 命令的最小时钟周期数 |

| | | | | |
|---|---|---|---|---|
| TCKESR | 44:40 | 0x0 | 0x0-0x1f | 自刷新进入到退出时 CKE 保持为低电平的最小时钟周期数 |
| TCCD | 36:32 | 0x0 | 0x0-0x1f | CAS#到 CAS#命令的延时 |
| ADD_ODT_CLK_DIFFTYPE_DIFFCS | 28:24 | 0x0 | 0x0-0x1f | 为了满足 ODT 时序，对不同片选的不同命令之间插入的附加时钟周期数 |
| TRP_AB | 19:16 | 0x0 | 0x0-0xf | 对所有 bank 的 trp 时间 |
| ADD_ODT_CLK_SAMETYPE_DIFFCS | 11:8 | 0x0 | 0x0-0xf | 为了满足 ODT 时序，对不同片选的同类型命令之间插入的附加时钟周期数 |
| ADD_ODT_CLK_DIFFTYPE_SAMECS | 3:0 | 0x0 | 0x0-0xf | 为了满足 ODT 时序，对同一片选的不同命令之间插入的附加时钟周期数 |
| CONF_CTL_155[63:0] | | Offset: 0x9b0 | | DDR2 667：0x0200010000000000 |
| ZQINIT | 59:48 | 0x0 | 0x0-0xfff | DRAM 初始化过程中 ZQ 命令需要的时钟周期数 |
| ZQCL | 43:32 | 0x0 | 0x0-0xfff | 正常的 ZQCL 命令需要的时钟周期数，它应等于 ZQINIT 的一半 |
| TDFI_WRLVL_WW | 25:16 | 0x0 | 0x0-0x3ff | 连续两次写 leveling 命令之间的最小时钟周期数 |
| TDFI_RDLVL_RR | 9:0 | 0x0 | 0x0-0x3ff | 连续两次读 leveling 命令之间的最小时钟周期数 |
| CONF_CTL_156[63:0] | | Offset: 0x9c0 | | DDR2 667：0x0a52000000000000 |
| MR0_DATA_0 | 62:48 | 0x0 | 0x0-0x7fff | 对应片选 0 的模式寄存器 0 配置值 |
| TDFI_PHYUPD_TYPE3 | 45:32 | 0x0 | 0x0-0x3fff | 保存 DFI Tphyupd_type3 参数（只读） |
| TDFI_PHYUPD_TYPE2 | 29:16 | 0x0 | 0x0-0x3fff | 保存 DFI Tphyupd_type2 参数（只读） |
| TDFI_PHYUPD_TYPE1 | 13:0 | 0x0 | 0x0-0x3fff | 保存 DFI Tphyupd_type1 参数（只读） |
| CONF_CTL_157[63:0] | | Offset: 0x9d0 | | DDR2 667：0x00440a520a520a52 |
| MR1_DATA_0 | 62:48 | 0x0 | 0x0-0x7fff | 对应片选 0 的模式寄存器 1 配置值 |
| MR0_DATA_3 | 46:32 | 0x0 | 0x0-0x7fff | 对应片选 3 的模式寄存器 0 配置值 |
| MR0_DATA_2 | 30:16 | 0x0 | 0x0-0x7fff | 对应片选 2 的模式寄存器 0 配置值 |
| MR0_DATA_1 | 14:0 | 0x0 | 0x0-0x7fff | 对应片选 1 的模式寄存器 0 配置值 |

| | | | | |
|---|---|---|---|---|
| CONF_CTL_158[63:0] | Offset: 0x9e0 | | DDR2 667：0x0000004400440044 | |
| MR2_DATA_0 | 62:48 | 0x0 | 0x0-0x7fff | 对应片选 0 的模式寄存器 2 配置值 |
| MR1_DATA_3 | 46:32 | 0x0 | 0x0-0x7fff | 对应片选 3 的模式寄存器 1 配置值 |
| MR1_DATA_2 | 30:16 | 0x0 | 0x0-0x7fff | 对应片选 2 的模式寄存器 1 配置值 |
| MR1_DATA_1 | 14:0 | 0x0 | 0x0-0x7fff | 对应片选 1 的模式寄存器 1 配置值 |
| CONF_CTL_159[63:0] | Offset: 0x9f0 | | DDR2 667：0x0000000000000000 | |
| MR3_DATA_0 | 62:48 | 0x0 | 0x0-0x7fff | 对应片选 0 的模式寄存器 3 配置值 |
| MR2_DATA_3 | 46:32 | 0x0 | 0x0-0x7fff | 对应片选 3 的模式寄存器 2 配置值 |
| MR2_DATA_2 | 30:16 | 0x0 | 0x0-0x7fff | 对应片选 2 的模式寄存器 2 配置值 |
| MR2_DATA_1 | 14:0 | 0x0 | 0x0-0x7fff | 对应片选 1 的模式寄存器 2 配置值 |
| CONF_CTL_160[63:0] | Offset: 0xa00 | | DDR2 667：0x00ff000000000000 | |
| DFI_WRLVL_MAX_DELAY | 63:48 | 0x0 | 0x0-0xffff | Hareware Write leveling 会使用的延迟线的最大级数 |
| MR3_DATA_3 | 46:32 | 0x0 | 0x0-0x7fff | 对应片选 3 的模式寄存器 3 配置值 |
| MR3_DATA_2 | 30:16 | 0x0 | 0x0-0x7fff | 对应片选 2 的模式寄存器 3 配置值 |
| MR3_DATA_1 | 14:0 | 0x0 | 0x0-0x7fff | 对应片选 1 的模式寄存器 3 配置值 |
| CONF_CTL_161[63:0] | Offset: 0xa10 | | DDR2 667：0x0000000000000000 | |
| RDLVL_BEGIN_DELAY_3 | 63:48 | 0x0 | 0x0-0xffff | 第 3 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DELAY_2 | 47:32 | 0x0 | 0x0-0xffff | 第 2 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DELAY_1 | 31:16 | 0x0 | 0x0-0xffff | 第 1 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DELAY_0 | 15:0 | 0x0 | 0x0-0xffff | 第 0 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| CONF_CTL_162[63:0] | Offset: 0xa20 | | DDR2 667：0x0000000000000000 | |
| RDLVL_BEGIN_DELAY_7 | 63:48 | 0x0 | 0x0-0xffff | 第 7 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DELAY_6 | 47:32 | 0x0 | 0x0-0xffff | 第 6 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DELAY_5 | 31:16 | 0x0 | 0x0-0xffff | 第 5 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| RDLVL_BEGIN_DE | 15:0 | 0x0 | 0x0-0xffff | 第 4 数据组中，Read Leveling 时从第一 |

| | | | | |
|---|---|---|---|---|
| LAY_4 | | | | 个 1 到 0 的延迟单元数目 |
| CONF_CTL_163[63:0] | | Offset: 0xa30 | | DDR2 667：0x0000000000000000 |
| RDLVL_DELAY_2 | 63:48 | 0x0 | 0x0-0xffff | 第 2 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_1 | 47:32 | 0x0 | 0x0-0xffff | 第 1 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_0 | 31:16 | 0x0 | 0x0-0xffff | 第 0 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_BEGIN_DELAY_8 | 15:0 | 0x0 | 0x0-0xffff | 第 8 数据组中，Read Leveling 时从第一个 1 到 0 的延迟单元数目 |
| CONF_CTL_164[63:0] | | Offset: 0xa40 | | DDR2 667：0x0000000000000000 |
| RDLVL _DELAY_6 | 63:48 | 0x0 | 0x0-0xffff | 第 6 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_5 | 47:32 | 0x0 | 0x0-0xffff | 第 5 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_4 | 31:16 | 0x0 | 0x0-0xffff | 第 4 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_3 | 15:0 | 0x0 | 0x0-0xffff | 第 3 数据组中，Read Leveling 使用的延迟单元数目 |
| CONF_CTL_165[63:0] | | Offset: 0xa50 | | DDR2 667：0x0000000000000000 |
| RDLVL_END_DELAY_1 | 63:48 | 0x0 | 0x0-0xffff | 第 1 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_END_DELAY_0 | 47:32 | 0x0 | 0x0-0xffff | 第 0 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_DELAY_8 | 31:16 | 0x0 | 0x0-0xffff | 第 8 数据组中，Read Leveling 使用的延迟单元数目 |
| RDLVL_DELAY_7 | 15:0 | 0x0 | 0x0-0xffff | 第 7 数据组中，Read Leveling 使用的延迟单元数目 |
| CONF_CTL_166[63:0] | | Offset: 0xa60 | | DDR2 667：0x0000000000000000 |
| RDLVL_END_DELAY_5 | 63:48 | 0x0 | 0x0-0xffff | 第 5 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_END_DELAY_4 | 47:32 | 0x0 | 0x0-0xffff | 第 4 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_END_DEL | 31:16 | 0x0 | 0x0-0xffff | 第 3 数据组中，Read Leveling 时从第一 |

| | | | | |
|---|---|---|---|---|
| AY_3 | | | | 个 0 到 1 的延迟单元数目 |
| RDLVL_END_DELAY_2 | 15:0 | 0x0 | 0x0-0xffff | 第 2 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| CONF_CTL_167[63:0] | Offset: 0xa70 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_DELAY_0 | 63:48 | 0x0 | 0x0-0xffff | 第 0 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_END_DELAY_8 | 47:32 | 0x0 | 0x0-0xffff | 第 8 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_END_DELAY_7 | 31:16 | 0x0 | 0x0-0xffff | 第 7 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| RDLVL_END_DELAY_6 | 15:0 | 0x0 | 0x0-0xffff | 第 6 数据组中，Read Leveling 时从第一个 0 到 1 的延迟单元数目 |
| CONF_CTL_168[63:0] | Offset: 0xa80 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_DELAY_4 | 63:48 | 0x0 | 0x0-0xffff | 第 4 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_3 | 47:32 | 0x0 | 0x0-0xffff | 第 3 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_2 | 31:16 | 0x0 | 0x0-0xffff | 第 2 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_1 | 15:0 | 0x0 | 0x0-0xffff | 第 1 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| CONF_CTL_169[63:0] | Offset: 0xa90 | | DDR2 667：0x0000000000000000 | |
| RDLVL_GATE_DELAY_8 | 63:48 | 0x0 | 0x0-0xffff | 第 8 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_7 | 47:32 | 0x0 | 0x0-0xffff | 第 7 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_6 | 31:16 | 0x0 | 0x0-0xffff | 第 6 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| RDLVL_GATE_DELAY_5 | 15:0 | 0x0 | 0x0-0xffff | 第 5 数据组中，采样时机到选通信号上升沿的延迟单元个数 |
| CONF_CTL_170[63:0] | Offset: 0xaa0 | | DDR2 667：0x0000ffff00000010 | |
| RDLVL_MIDPOINT_DELAY_0 | 63:48 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_0 和 rdlvl_end_delay_0 的中间值，否则，等于 rdlvl_delay_0(只读) |

| | | | | |
|---|---|---|---|---|
| RDLVL_MAX_DELAY | 47:32 | 0x0 | 0x0-0xffff | Read Leveling 延迟线的最大数目 |
| RDLVL_GATE_REFRESH_INTERVAL | 31:16 | 0x0 | 0x0-0xffff | 两次自动 Gate Training 之间的最大刷新命令数（应设为 0） |
| RDLVL_GATE_MAX_DELAY | 15:0 | 0x0 | 0x0-0xffff | 采样延迟线的最大数目 |
| CONF_CTL_171[63:0] | | Offset: 0xab0 | | DDR2 667：0x0000000000000000 |
| RDLVL_MIDPOINT_DELAY_4 | 63:48 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_4 和 rdlvl_end_delay_4 的中间值，否则，等于 rdlvl_delay_4(只读) |
| RDLVL_MIDPOINT_DELAY_3 | 47:32 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_3 和 rdlvl_end_delay_3 的中间值，否则，等于 rdlvl_delay_3(只读) |
| RDLVL_MIDPOINT_DELAY_2 | 31:16 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_2 和 rdlvl_end_delay_2 的中间值，否则，等于 rdlvl_delay_2(只读) |
| RDLVL_MIDPOINT_DELAY_1 | 15:0 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_1 和 rdlvl_end_delay_1 的中间值，否则，等于 rdlvl_delay_1(只读) |
| CONF_CTL_172[63:0] | | Offset: 0xac0 | | DDR2 667：0x0000000000000000 |
| RDLVL_MIDPOINT_DELAY_8 | 63:48 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_8 和 rdlvl_end_delay_8 的中间值，否则，等于 rdlvl_delay_8(只读) |
| RDLVL_MIDPOINT_DELAY_7 | 47:32 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_7 和 rdlvl_end_delay_7 的中间值，否则，等于 rdlvl_delay_7(只读) |
| RDLVL_MIDPOINT_DELAY_6 | 31:16 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_6 和 rdlvl_end_delay_6 的中间值，否则，等于 rdlvl_delay_6(只读) |

| | | | | |
|---|---|---|---|---|
| RDLVL_MIDPOINT_DELAY_5 | 15:0 | 0x0 | 0x0-0xffff | 当 Hardware read leveling 模块使能时，等于 rdlvl_begin_delay_5 和 rdlvl_end_delay_5 的中间值，否则，等于 rdlvl_delay_5(只读) |
| CONF_CTL_173[63:0] | Offset: 0xad0 | | DDR2 667：0x0000000000000000 | |
| RDLVL_OFFSET_DELAY_3 | 63:48 | 0x0 | 0x0-0xffff | 第 3 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_2 | 47:32 | 0x0 | 0x0-0xffff | 第 2 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_1 | 31:16 | 0x0 | 0x0-0xffff | 第 1 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_0 | 15:0 | 0x0 | 0x0-0xffff | 第 0 数据组中，到 Read Leveling 中点的偏移 |
| CONF_CTL_174[63:0] | Offset: 0xae0 | | DDR2 667：0x0000000000000000 | |
| RDLVL_OFFSET_DELAY_7 | 63:48 | 0x0 | 0x0-0xffff | 第 7 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_6 | 47:32 | 0x0 | 0x0-0xffff | 第 6 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_5 | 31:16 | 0x0 | 0x0-0xffff | 第 5 数据组中，到 Read Leveling 中点的偏移 |
| RDLVL_OFFSET_DELAY_4 | 15:0 | 0x0 | 0x0-0xffff | 第 4 数据组中，到 Read Leveling 中点的偏移 |
| CONF_CTL_175[63:0] | Offset: 0xaf0 | | DDR2 667：0x0000000000000000 | |
| WRLVL_DELAY_1 | 63:48 | 0x0 | 0x0-0xffff | 第 1 数据组中，控制写 DQS 经 DLL 延迟数 |
| WRLVL_DELAY_0 | 47:32 | 0x0 | 0x0-0xffff | 第 0 数据组中，控制写 DQS 经 DLL 延迟数 |
| RDLVL_REFRESH_INTERVAL | 31:16 | 0x0 | 0x0-0xffff | 两次自动 Read Leveling 之间的最大刷新命令数（应设为 0） |
| RDLVL_OFFSET_DELAY_8 | 15:0 | 0x0 | 0x0-0xffff | 第 8 数据组中，到 Read Leveling 中点的偏移 |
| CONF_CTL_176[63:0] | Offset: 0xb00 | | DDR2 667：0x0000000000000000 | |
| WRLVL_DELAY_5 | 63:48 | 0x0 | 0x0-0xffff | 第 5 数据组中，控制写 DQS 经 DLL 延迟数 |
| WRLVL_DELAY_4 | 47:32 | 0x0 | 0x0-0xffff | 第 4 数据组中，控制写 DQS 经 DLL 延迟 |

| | | | | |
|---|---|---|---|---|
| | | | | 数 |
| WRLVL_DELAY_3 | 31:16 | 0x0 | 0x0-0xffff | 第 3 数据组中，控制写 DQS 经 DLL 延迟数 |
| WRLVL_DELAY_2 | 15:0 | 0x0 | 0x0-0xffff | 第 2 数据组中，控制写 DQS 经 DLL 延迟数 |
| CONF_CTL_177[63:0] | | Offset: 0xb10 | | DDR2 667：0x0000000000000000 |
| WRLVL_REFRESH_INTERVAL | 63:48 | 0x0 | 0x0-0xffff | 两次自动 Write Leveling 之间的最大刷新命令数（应设为 0） |
| WRLVL_DELAY_8 | 47:32 | 0x0 | 0x0-0xffff | 第 8 数据组中，控制写 DQS 经 DLL 延迟数 |
| WRLVL_DELAY_7 | 31:16 | 0x0 | 0x0-0xffff | 第 7 数据组中，控制写 DQS 经 DLL 延迟数 |
| WRLVL_DELAY_6 | 15:0 | 0x0 | 0x0-0xffff | 第 6 数据组中，控制写 DQS 经 DLL 延迟数 |
| CONF_CTL_178[63:0] | | Offset: 0xb20 | | DDR2 667：0x00000c2d00000c2d |
| TDFI_RDLVL_RESP | 63:32 | 0x0 | 0x0-0xffff | 保存 DFI Trdlvl_resp 时间参数 |
| TDFI_RDLVL_MAX | 31:0 | 0x0 | 0x0-0xffff | 保存 DFI Trdlvl_max 时间参数 |
| CONF_CTL_179[63:0] | | Offset: 0xb30 | | DDR2 667：0x00000c2d00000c2d |
| TDFI_WRLVL_RESP | 63:32 | 0x0 | 0x0-0xffff | 保存 DFI Twrlvl_resp 时间参数 |
| TDFI_WRLVL_MAX | 31:0 | 0x0 | 0x0-0xffff | 保存 DFI Twrlvl_max 时间参数 |

# 9 HyperTransport 控制器

龙芯 2G 中，HyperTransport 总线用于外部设备连接。用户程序可以自由选择是否支持 IO Cache 一致性（由地址窗口 Uncache 窗口设置，见 9.4.2 节）。在 Cache 一致性支持模式下，IO 设备对内 DMA 访问对于 Cache 层次透明，即不需要通过程序 Cache 指令维护一致性，而是由硬件自动维护其一致性。

HyperTransport 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，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，重新初始化，详细方法见 9.1 节。

龙芯 2 号 HyperTransport 控制器的主要特征如下：

- 支持 200/400/800Mhz
- 支持 8/16 位宽度
- 总线控制信号（包括 PowerOK，Rstn，LDT_Stopn）方向可配置
- 外设 DMA 空间 Cache/Uncache 可配置

## 9.1 HyperTransport 硬件设置及初始化

HyperTransport 总线由传输信号总线和控制信号引脚等组成，下表给出了 HyperTransport 总线相关的引脚及其作用。

表 9-1 HyperTransport 总线相关引脚信号

| 引脚 | 名称 | 描述 |
|---|---|---|
| {PCI_Config[7], PCI_Config[0]} | HT 周边信号电压控制 | 00：将 HyperTransport 周边信号作为 1.8v 信号，这些信号包括 HT_8x2，HT_Mode，HT_Powerok，HT_Rstn，HT_Ldt_Stopn，HT_Ldt_Reqn。<br>01：保留。<br>10：将 HyperTransport 周边信号作为 2.5v 信号。<br>11：将 HyperTransport 周边信号作为 3.3v 信号。 |
| HT_mode | 主设备模式 | 1：将 HT 设为主设备模式，这个模式下，总线控制信号等由 HT 驱动，这些控制信号包括 HT _Powerok，HT_Rstn，HT_Ldt_Stopn。这个模式下，这些控制信号也可以为双向驱动。同时这个引脚决定（取反）寄存器“Act as Slave”的初始值，这个寄存器为 0 时，HyperTransport 总线上的包中的 Bridge 位为 1，否则为 0。另外，这个寄存器为 0 时，如果 HyperTransport 总线上的请求地址没有在控制器的接收窗口命中时，将作为P2P请求重新发回总线，如果这个寄存器为 1 时，没有命中，则作为错误请求做出响应。<br>0：将 HT 设为从设备模式，这个模式下，总线控制信 |

| | | |
|---|---|---|
| | | 号等由对方设备驱动，这些控制信号包括 HT_Powerok，HT_Rstn，HT_Ldt_Stopn。这个模式下，这些控制信号由对方设备驱动，如果没有被正确驱动，则 HT 总线不能正确工作。 |
| HT_Powerok | 总线 Powerok | HyperTransport 总线 Powerok 信号，<br>HT_Mode 为 1 时，由 HT0_Lo 控制；<br>HT_Mode 为 0 时，由对方设备控制。 |
| HT_Rstn | 总线 Rstn | HyperTransport 总线 Rstn 信号，<br>HT0_Mode 为 1 时，由 HT0_Lo 控制；<br>HT0_Mode 为 0 时，由对方设备控制。 |
| HT_Ldt_Stopn | 总线 Ldt_Stopn | HyperTransport 总线 Ldt_Stopn 信号，<br>HT_Mode 为 1 时，由 HT0_Lo 控制；<br>HT_Mode 为 0 时，由对方设备控制。 |
| HT_Ldt_Reqn | 总线 Ldt_Reqn | HyperTransport 总线 Ldt_Reqn 信号， |
| HT_Rx_CLKp[1:0]<br>HT_Rx_CLKn[1:0]<br>HT_Tx_CLKp[1:0]<br>HT_Tx_CLKp[1:0] | CLK[1:0] | HyperTransport 总线 CLK 信号 |
| HT_Rx_CTLp[1:0]<br>HT_Rx_CTLn[1:0]<br>HT_Tx_CTLp[1:0]<br>HT_Tx_CTLn[1:0] | CTL[1:0] | HyperTransport 总线 CTL 信号<br>CTL[1]无效<br>CTL[0]由 HT0_Lo 控制 |
| HT_Rx_CADp[15:0]<br>HT_Rx_CADn[15:0]<br>HT_Tx_CADp[15:0]<br>HT_Tx_CADn[15:0] | CAD[15:0] | HyperTransport 总线 CAD 信号 |

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 9.5.2 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 9.5.2 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，HyperTransport 两端的设备的配置需要一一对应，否则将使得 HyperTransport 接口不能正常工作。

## 9.2 HyperTransport 协议支持

HyperTransport 总线支持 1.03 协议中的大部分命令，其接收端可接收的命

令如下表所示。需要注意的是，HyperTransport 总线的原子操作命令不被支持。

表 9-2 HyperTransport 接收端可接收的命令

| 编码 | 通道 | 命令 | 标准模式 |
|---|---|---|---|
| 000000 | - | NOP | 空包或流控 |
| 000001 | NPC | FLUSH | 无操作 |
| x01xxx | NPC<br>or<br>PC | Write | bit 5: 0 - Nonposted<br>1 - Posted<br>bit 2: 0 - Byte<br>1 - Doubleword<br>bit 1: Don't Care<br>bit 0: Don't Care |
| 01xxxx | NPC | Read | bit 3: Don't Care<br>bit 2: 0 - Byte<br>1 - Doubleword<br>bit 1: Don't Care<br>bit 0: Don't Care |
| 110000 | R | RdResponse | 读操作返回 |
| 110011 | R | TgtDone | 写操作返回 |
| 110100 | PC | WrCoherent | ---- |
| 110101 | PC | WrAddr | ---- |
| 111000 | R | RespCoherent | ---- |
| 111001 | NPC | RdCoherent | ---- |
| 111010 | PC | Broadcast | 无操作 |
| 111011 | NPC | RdAddr | ---- |
| 111100 | PC | FENCE | 保证序关系 |
| 111111 | - | Sync/Error | Sync/Error |

对于发送端，会向外发送的命令如下表所示。

表 9-3 两种模式下会向外发送的命令

| 编码 | 通道 | 命令 | 标准模式 |
|---|---|---|---|
| 000000 | - | NOP | 空包或流控 |
| x01x0x | NPC<br>or<br>PC | Write | bit 5: 0 - Nonposted<br>1 - Posted<br>bit 2: 0 - Byte<br>1 - Doubleword<br>bit 0: 必为 0 |
| 010x0x | NPC | Read | bit 2: 0 - Byte<br>1 - Doubleword<br>bit 0: Don't Care |
| 110000 | R | RdResponse | 读操作返回 |
| 110011 | R | TgtDone | 写操作返回 |
| 110100 | PC | WrCoherent | ---- |
| 110101 | PC | WrAddr | ---- |
| 111000 | R | RespCoherent | ---- |
| 111001 | NPC | RdCoherent | ---- |

| | | | |
|---|---|---|---|
| 111011 | NPC | RdAddr | ---- |
| 111111 | - | Sync/Error | 只会转发 |

## 9.3 HyperTransport 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 EOI 提供支持。对于 Fix/Arbitor 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，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：①PIC 控制器向系统发送 PIC 中断请求；②系统向 PIC 控制器发送中断向量查询；③PIC 控制器向系统发送中断向量号；④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。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。

## 9.4 HyperTransport 地址窗口

### 9.4.1 HyperTransport空间

龙芯 2G 处理器中，默认的 HyperTransport 地址窗口如下：

表 9-4 默认的 HyperTransport 地址窗口的地址

| 基地址 | 结束地址 | 大小 | 定义 |
|---|---|---|---|
| 0x0E00_0000_0000 | 0x0EFF_FFFF_FFFF | 1 Tbytes | HT1_LO 窗口 |

在默认情况下（未对系统地址窗口另行配置），软件通过上述地址空间对 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 2.5 节）。HyperTransport 接口内部 40 位地址空间的具体地址窗口分布如下表所述。

龙芯 2G 处理器 HyperTransport 接口协议的地址窗口分布如下：

表 9-5 龙芯 2G 处理器 HyperTransport 接口地址窗口分布

| 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 |

## 9.4.2 HyperTransport控制器内部窗口配置

龙芯 2G 处理器 HyperTransport 接口中提供了多种丰富的地址窗口供用户使用，包括如下几种：

表 9-6 龙芯 2 号处理器 HyperTransport 接口中提供的地址窗口

| 地址窗口 | 窗口数 | 接受总线 | 作用 | 备注 |
|---|---|---|---|---|
| 接收窗口（窗口配置见 9.5.4 节） | 3 | HyperTransport | 判断是否接收 HyperTransport 总线上发出的访问。 | 处于主桥模式时（即配置寄存器中 act_as_slave 为 0），只有落在这些地址窗口中的访问会被内部总线所响应，其它访问将会被认为是 P2P 访问重新发回到 HyperTransport 总线上；处于设备模式时（即配置寄存器中 act_as_slave 为 1），只有落在这些地址窗口中的访问会被内部总线所接收并处理，其它访问将会按照协议给出错误返回。 |
| Post 窗口（窗口配置见 9.5.8 节） | 2 | 内部总线 | 判断是否将内部总线对 HyperTransport 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。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。 |
| 可预取窗口（窗口配置 | 2 | 内部总线 | 判断是否接收内部的 Cache 访 | 当处理器核乱序执行时，会对总线发出一些猜测读访问 |

| | | | | |
|---|---|---|---|---|
| 见 9.5.9 节） | | | 问，取指访问。 | 或是取指访问，这种访问对于某些 IO 空间是错误的。在默认情况下，这种访问 HT 控制器将直接返回而不对 HyperTransport 总线进行访问。通过这些窗口可以使能对 HyperTransport 总线的这类访问。 |
| Uncache 窗口（窗口配置见 9.5.10 节） | 2 | HyperTransport | 判断是否将 HyperTransport 总线上的访问作为对内部的 Uncache 访问 | 龙芯 2 号处理器内部的 IO DMA 访问，在情况下将作为 Cache 方式访问经由二级 Cache 判断是命中，从而维护其 IO 一致性信息。而通过这些窗口的配置，可以使在这些窗口命中的访问以 Uncache 的方式直接访问内存，而不通过硬件维护其 IO 一致性信息。 |

## 9.5 配置寄存器

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。

首先，用于控制 HT 控制器各种行为的配置寄存器的访问与存储都在本模块中，本模块的访问偏移地址在 AXI 端为 0xFD_FB00_0000 到 0xFD_FBFF_FFFF。本模块中所有软件可见寄存器如下表所示：

表 9-7 本模块中所有软件可见寄存器

| 偏移地址 | 名称 | 描述 |
|---|---|---|
| 0x30 | | |
| 0x34 | | |
| 0x38 | | |
| 0x3c | Bridge Control | Bus Reset Control |
| 0x40 | Capability Registers | Command，Capabilities Pointer，Capability ID |
| 0x44 | | Link Config，Link Control |
| 0x48 | | Revision ID，Link Freq，Link Error，Link Freq Cap |
| 0x4c | | Feature Capability |
| 0x50 | 自定义寄存器 | MISC |
| 0x54 | | |
| 0x58 | | |
| 0x5c | | |
| 0x60 | 接收地址窗口 | HT 总线接收地址窗口 0 使能（外部访问） |

| | | |
|---|---|---|
| 0x64 | 配置寄存器 | HT 总线接收地址窗口 0 基址（外部访问） |
| 0x68 | | HT 总线接收地址窗口 1 使能（外部访问） |
| 0x6c | | HT 总线接收地址窗口 1 基址（外部访问） |
| 0x70 | | HT 总线接收地址窗口 2 使能（外部访问） |
| 0x74 | | HT 总线接收地址窗口 2 基址（外部访问） |
| 0x78 | | |
| 0x7c | | |
| 0x80 | 中断向量寄存器 | HT 总线中断向量寄存器[31:0] |
| 0x84 | | HT 总线中断向量寄存器[63:32] |
| 0x88 | | HT 总线中断向量寄存器[95:64] |
| 0x8c | | HT 总线中断向量寄存器[127:96] |
| 0x90 | | HT 总线中断向量寄存器[159:128] |
| 0x94 | | HT 总线中断向量寄存器[191:160] |
| 0x98 | | HT 总线中断向量寄存器[223:192] |
| 0x9C | | HT 总线中断向量寄存器[255:224] |
| 0xA0 | 中断使能寄存器 | HT 总线中断使能寄存器[31:0] |
| 0xA4 | | HT 总线中断使能寄存器[63:32] |
| 0xA8 | | HT 总线中断使能寄存器[95:64] |
| 0xAC | | HT 总线中断使能寄存器[127:96] |
| 0xB0 | | HT 总线中断使能寄存器[159:128] |
| 0xB4 | | HT 总线中断使能寄存器[191:160] |
| 0xB8 | | HT 总线中断使能寄存器[223:192] |
| 0xBC | | HT 总线中断使能寄存器[255:224] |
| 0xC0 | Interrupt Discovery & Configuration | Interrupt Capability |
| 0xC4 | | DataPort |
| 0xC8 | | IntrInfo[31:0] |
| 0xCC | | IntrInfo[63:32] |
| 0xD0 | POST 地址窗口配置寄存器 | HT 总线 POST 地址窗口 0 使能（内部访问） |
| 0xD4 | | HT 总线 POST 地址窗口 0 基址（内部访问） |
| 0xD8 | | HT 总线 POST 地址窗口 1 使能（内部访问） |
| 0xDC | | HT 总线 POST 地址窗口 1 基址（内部访问） |
| 0xE0 | 可预取地址窗口配置寄存器 | HT 总线可预取地址窗口 0 使能（内部访问） |
| 0xE4 | | HT 总线可预取地址窗口 0 基址（内部访问） |
| 0xE8 | | HT 总线可预取地址窗口 1 使能（内部访问） |
| 0xEC | | Ht 总线可预取地址窗口 1 基址（内部访问） |
| 0xF0 | Uncache 地址窗口配置寄存器 | HT 总线 Uncache 地址窗口 0 使能（内部访问） |
| 0xF4 | | HT 总线 Uncache 地址窗口 0 基址（内部访问） |
| 0xF8 | | HT 总线 Uncache 地址窗口 1 使能（内部访问） |
| 0xFC | | HT 总线 Uncache 地址窗口 1 基址（内部访问） |

每个寄存存器的具体含义如下节如示：

### 9.5.1 Bridge Control

偏移量： 0x3C
复位值： 0x00200000
名称： Bus Reset Control

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:23 | Reserved | 4 | 0x0 | | 保留 |
| 22 | Reset | 12 | 0x0 | R/W | 总线复位控制：<br>0->1：HT_RSTn 置 0，总线复位<br>1->0：HT_RSTn 置 1，总线解复位 |
| 21:0 | Reserved | 5 | 0x0 | | 保留 |

### 9.5.2 Capability Registers

偏移量： 0x40
复位值： 0x20010008
名称： Command, Capabilities Pointer, Capability ID

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:29 | HOST/Sec | 3 | 0x1 | R | Command 格式为 HOST/Sec |
| 28:27 | Reserved | 2 | 0x0 | R | 保留 |
| 26 | Act as Slave | 1 | 0x0<br>/0x1 | R/W | HOST/SLAVE 模式<br>初始值由引脚 HOSTMODE 决定<br>HOSTMODE 上拉：0<br>HOSTMODE 下拉：1 |
| 25 | Reserved | 1 | 0x0 | | 保留 |
| 24 | Host Hide | 1 | 0x0 | R/W | 是否禁止来自 HT 总线的寄存器访问 |
| 23 | Reserved | 1 | 0x0 | | 保留 |
| 22:18 | Unit ID | 5 | 0x0 | R/W | HOST 模式时：可用于记录使用 ID 个数<br>SLAVE 模式时：记录自身 Unit ID |
| 17 | Double Ended | 1 | 0x0 | R | 不采用双 HOST 模式 |
| 16 | Warm Reset | 1 | 0x1 | R | Bridge Control 中 reset 采用热复位方式 |
| 15:8 | Capabilities Pointer | 8 | 0xa0 | R | 下一个 Cap 寄存器偏移地址 |
| 7:0 | Capability ID | 8 | 0x08 | R | HyperTransport capability ID |

偏移量： 0x44
复位值： 0x00112000
名称： Link Config, Link Control

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | Reserved | 1 | 0x0 | | 保留 |

| | | | | | |
|---|---|---|---|---|---|
| 30:28 | Link Width Out | 3 | 0x0 | R/W | 发送端宽度<br>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<br>000：8 位方式<br>001：16 位方式 |
| 27 | Reserved | 1 | 0x0 | | 保留 |
| 26:24 | Link Width In | 3 | 0x0 | R/W | 接收端宽度<br>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 |
| 23 | Dw Fc out | 1 | 0x0 | R | 发送端不支持双字流控 |
| 22:20 | Max Link Width out | 3 | 0x1 | R | HT 总线发送端最大宽度：16bits |
| 19 | Dw Fc In | 1 | 0x0 | R | 接收端不支持双字流控 |
| 18:16 | Max Link Width In | 3 | 0x1 | R | HT 总线接收端最大宽度：16bits |
| 15:14 | Reserved | 2 | 0x0 | | 保留 |
| 13 | LDTSTOP# Tristate Enable | 1 | 0x1 | R/W | 当 HT 总线进入 HT Disconnect 状态时，是否关闭 HT PHY<br>1：关闭<br>0：不关闭 |
| 12:10 | Reserved | 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<br>处于 16 位总线工作方式时<br>1：关闭 高/低 8 位 HT PHY<br>0：使能 低 8 位 HT PHY，<br>高 8 位 HT PHY 由 bit 0 控制 |
| 6 | End of Chain | 0 | 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 | Reserved | 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 | 使用 16 位 HT 总线运行 8 位协议时，高 8 位 PHY 关闭控制<br>1：关闭 高 8 位 HT PHY<br>0：使能 高 8 位 HT PHY |

偏移量：0x48
复位值：0x80250023
名称：Revision ID，Link Freq，Link Error，Link Freq Cap

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|

| 位域 | 位域名称 | 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<br>200Mhz，400Mhz，800Mhz， |
| 15:14 | Reserved | 2 | 0x0 | | 保留 |
| 13 | Over Flow Error | 1 | 0x0 | R | HT 总线包溢出 |
| 12 | Protocol Error | 1 | 0x0 | R/W | 协议错误，<br>指 HT 总线上收到不可识别的命令 |
| 11:8 | Link Freq | 4 | 0x0 | R/W | HT 总线工作频率<br>写入此寄存器的值后将在下次热复位或是 HT Disconnect 之后生效<br>0000：200M<br>0010：400M<br>0101：800M |
| 7:0 | Revision ID | 8 | 0x23 | R/W | 版本号：1.03 |

偏移量：0x4C
复位值：0x00000002
名称：Feature Capability

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:9 | Reserved | 25 | 0x0 | | 保留 |
| 8 | Extended Register | 1 | 0x0 | R | 没有 |
| 7:4 | Reserved | 3 | 0x0 | | 保留 |
| 3 | Extended CTL Time | 1 | 0x0 | R | 不需要 |
| 2 | CRC Test Mode | 1 | 0x0 | R | 不支持 |
| 1 | LDTSTOP# | 1 | 0x1 | R | 支持 LDTSTOP# |
| 0 | Isochronous Mode | 1 | 0x0 | R | 不支持 |

### 9.5.3 自定义寄存器

偏移量：0x50
复位值：0x00904321
名称：MISC

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | Reserved | 1 | 0x0 | | 保留 |
| 30 | Ldt Stop Gen | 1 | 0x0 | R/W | 使总线进入 LDT DISCONNECT 模式<br>正确的方法是：0 -> 1 |
| 29 | Ldt Req Gen | 1 | 0x0 | R/W | 从 LDT DISCONNECT 中唤醒 HT 总线，设置 LDT_REQ_n<br>正确的方法是先置 0 再置 1：0 -> 1<br>除此之外，直接向总线发出读写请求也可以自动唤醒总线 |

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 28:24 | Interrupt Index | 5 | 0x0 | R/W | 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）<br>总共 256 个中断向量，本寄存器表示的是中断向量的高 5 位，内部中断向量如下：<br>000：SMI<br>001：NMI<br>010：INIT<br>011：Reserved<br>100：INTA<br>101：INTB<br>110：INTC<br>111：INTD |
| 23 | Dword Write | 1 | 0x1 | R/W | 对于 32/64/128/256 位的写访问，是否采用 Dword Write 命令格式<br>1：使用 Dword Write<br>0：使用 Byte Write（带 MASK） |
| 22 | Coherent Mode | 1 | 0x0 | R | 是否是处理器一致性模式<br>由引脚 ICCC_EN 决定 |
| 21 | Not Care Seqid | 1 | 0x0 | R/W | 是否不关心 HT 序关系 |
| 20 | Not Axi2Seqid | 1 | 0x1 | R | 是否把 Axi 总线上的命令转换成不同的 SeqID，如果不转换，则所有的读写命令都会采用 Fixed Seqid 中的固定 ID 号<br>1：不转换<br>0：转换 |
| 19:16 | Fixed Seqid | 4 | 0x0 | R/W | 当 Not Axi2Seqid 有效时，配置 HT 总线发出的 Seqid |
| 15:12 | Priority Nop | 4 | 0x4 | R/W | HT 总线 Nop 流控包优先级 |
| 11:8 | Priority NPC | 4 | 0x3 | R/W | Non Post 通道读写优先级 |
| 7:4 | Priority RC | 4 | 0x2 | R/W | Response 通道读写先级 |
| 3:0 | Priority PC | 4 | 0x1 | R/W | Post 通道读写优先级<br>0x0：最高优先级<br>0xF：最低优先级<br>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 |

## 9.5.4 接收地址窗口配置寄存器

本控制器中的地址窗口命中公式如下：

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

addr_out = TRANS_EN ? TRANS | ADDR & ~MASK : ADDR

值得一提的是，配置地址窗口寄存器时，MASK 高位应全为 1，低位应全为 0。MASK 中 0 的实际位数表示的就是地址窗口的大小。

本窗口的地址为HT总线上接收的地址。落在本窗口内的HT地址将被发往CPU内，其它地址的命令将作为P2P命令被转发回HT总线。

偏移量： 0x60
复位值： 0x00000000
名称： HT总线接收地址窗口0使能（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_rx_image0_en | 1 | 0x0 | R/W | HT总线接收地址窗口0，使能信号 |
| 30 | ht_rx_image0_trans_en | 1 | 0x0 | R/W | HT总线接收地址窗口0，映射使能信号 |
| 29:23 | Reserved | 14 | 0x0 | | 保留 |
| 15:0 | ht_rx_image0_trans[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 0，映射后地址的[39:24] |

偏移量： 0x64
复位值： 0x00000000
名称： HT总线接收地址窗口0基址（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_rx_image0_base[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 0，地址基址的[39:24] |
| 15:0 | ht_rx_image0_mask[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 0，地址屏蔽的[39:24] |

偏移量： 0x68
复位值： 0x00000000
名称： HT总线接收地址窗口1使能（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_rx_image1_en | 1 | 0x0 | R/W | HT总线接收地址窗口1，使能信号 |
| 30 | ht_rx_image1_trans_en | 1 | 0x0 | R/W | HT总线接收地址窗口1，映射使能信号 |
| 29:23 | Reserved | 14 | 0x0 | | 保留 |
| 15:0 | ht_rx_image1_trans[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 1，映射后地址的[39:24] |

偏移量： 0x6c
复位值： 0x00000000
名称： HT总线接收地址窗口1基址（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_rx_image1_base[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 1，地址基址的[39:24] |

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 15:0 | ht_rx_image1_mask[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 1，地址屏蔽的[39:24] |

偏移量：0x70
复位值：0x00000000
名称：HT 总线接收地址窗口 2 使能（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_rx_image2_en | 1 | 0x0 | R/W | HT 总线接收地址窗口 2，使能信号 |
| 30 | ht_rx_image2_trans_en | 1 | 0x0 | R/W | HT 总线接收地址窗口 2，映射使能信号 |
| 29:23 | Reserved | 14 | 0x0 | | 保留 |
| 15:0 | ht_rx_image2_trans[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 2，转译后地址的[39:24] |

偏移量：0x74
复位值：0x00000000
名称：HT 总线接收地址窗口 2 基址（外部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_rx_image2_base[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 2，地址基址的[39:24] |
| 15:0 | ht_rx_image2_mask[39:24] | 16 | 0x0 | R/W | HT 总线接收地址窗口 2，地址屏蔽的[39:24] |

### 9.5.5 中断向量寄存器

中断向量寄存器共 256 个，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，如 SMI，NMI，INIT，INTA，INTB，INTC，INTD 可以通过寄存器 0x50 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]}。

偏移量：0x80
复位值：0x00000000
名称：HT 总线中断向量寄存器[31:0]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case[31:0] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[31:0]，对应中断线 0 |

偏移量：0x84
复位值：0x00000000

名称： HT 总线中断向量寄存器[63:32]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [63:32] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[63:32]，对应中断线 0 |

偏移量： 0x88
复位值： 0x00000000
名称： HT 总线中断向量寄存器[95:64]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [95:64] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[95:64]，对应中断线 1 |

偏移量： 0x8c
复位值： 0x00000000
名称： HT 总线中断向量寄存器[127:96]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [127:96] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[127:96]，对应中断线 1 |

偏移量： 0x90
复位值： 0x00000000
名称： HT 总线中断向量寄存器[159:128]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [159:128] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[159:128]，对应中断线 2 |

偏移量： 0x94
复位值： 0x00000000
名称： HT 总线中断向量寄存器[191:160]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [191:160] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[191:160]，对应中断线 2 |

偏移量： 0x98
复位值： 0x00000000
名称： HT 总线中断向量寄存器[223:192]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [223:192] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[223:192]，对应中断线 3 |

偏移量： 0x9c
复位值： 0x00000000
名称： HT 总线中断向量寄存器[255:224]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_case [255:224] | 32 | 0x0 | R/W | HT 总线中断向量寄存器[255:224]，对应中断线 3 |

### 9.5.6 中断使能寄存器

中断使能寄存器共 256 个，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。

偏移量： 0xa0
复位值： 0x00000000
名称： HT 总线中断使能寄存器[31:0]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [31:0] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[31:0]，对应中断线 0 |

偏移量： 0xa4
复位值： 0x00000000
名称： HT 总线中断使能寄存器[63:32]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [63:32] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[63:32]，对应中断线 0 |

偏移量： 0xa8
复位值： 0x00000000
名称： HT 总线中断使能寄存器[95:64]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [95:64] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[95:64]，对应中断线 1 |

偏移量： 0xac
复位值： 0x00000000
名称： HT 总线中断使能寄存器[127:96]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [127:96] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[127:96]，对应中断线 1 |

偏移量： 0xb0
复位值： 0x00000000
名称： HT 总线中断使能寄存器[159:128]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [159:128] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[159:128]，对应中断线 2 |

偏移量： 0xb4
复位值： 0x00000000

名称： HT 总线中断使能寄存器[191:160]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [191:160] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[191:160]，对应中断线 2 |

偏移量： 0xb8
复位值： 0x00000000
名称： HT 总线中断使能寄存器[223:192]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [223:192] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[223:192]，对应中断线 3 |

偏移量： 0xbc
复位值： 0x00000000
名称： HT 总线中断使能寄存器[255:224]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | Interrupt_mask [255:224] | 32 | 0x0 | R/W | HT 总线中断使能寄存器[255:224]，对应中断线 3 |

### 9.5.7 Interrupt Discovery & Configuration

偏移量： 0xc0
复位值： 0x80000008
名称： Interrupt Capability

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:24 | Capabilities Pointer | 8 | 0x80 | R | Interrupt discovery and 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 |

偏移量： 0xc4
复位值： 0x00000000
名称： Dataport

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 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 |

偏移量： 0xc8
复位值： 0xF8000000
名称： IntrInfo[31:0]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:24 | IntrInfo[31:24] | 32 | 0xF8 | R | 保留 |

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 23:2 | IntrInfo[23:2] | 22 | 0x0 | R/W | IntrInfo[23:2]，当发出 PIC 中断时，IntrInfo 的值用来表示中断向量 |
| 1:0 | Reserved | 2 | 0x0 | R | 保留 |

偏移量：0xcc
复位值：0x00000000
名称：IntrInfo[63:32]

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:0 | IntrInfo[63:32] | 32 | 0x0 | R | 保留 |

## 9.5.8 POST地址窗口配置寄存器

本控制器中的地址窗口命中公式如下：

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

值得一提的是，配置地址窗口寄存器时，MASK 高位应全为 1，低位应全为 0。MASK 中 0 的实际位数表示的就是地址窗口的大小。

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。

偏移量：0xd0
复位值：0x00000000
名称：HT 总线 POST 地址窗口 0 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_post0_en | 1 | 0x0 | R/W | HT 总线 POST 地址窗口 0，使能信号 |
| 30:23 | Reserved | 15 | 0x0 | | 保留 |
| 15:0 | ht_post0_trans[39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 0，转译后地址的[39:24] |

偏移量：0xd4
复位值：0x00000000
名称：HT 总线 POST 地址窗口 0 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_post0_base[39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 0，地址基址的[39:24] |
| 15:0 | ht_post0_mask[39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 0，地址屏蔽的[39:24] |

偏移量： 0xd8
复位值： 0x00000000
名称： HT 总线 POST 地址窗口 1 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_post1_en | 1 | 0x0 | R/W | HT 总线 POST 地址窗口 1，使能信号 |
| 30:23 | Reserved | 15 | 0x0 | | 保留 |
| 15:0 | ht_post1_trans [39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 1，转译后地址的 [39:24] |

偏移量： 0xdc
复位值： 0x00000000
名称： HT 总线 POST 地址窗口 1 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_post1_base [39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 1，地址基址的 [39:24] |
| 15:0 | ht_post1_mask [39:24] | 16 | 0x0 | R/W | HT 总线 POST 地址窗口 1，地址屏蔽的 [39:24] |

## 9.5.9 可预取地址窗口配置寄存器

本控制器中的地址窗口命中公式如下：

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

值得一提的是，配置地址窗口寄存器时，MASK 高位应全为 1，低位应全为 0。MASK 中 0 的实际位数表示的就是地址窗口的大小。

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。

偏移量： 0xe0
复位值： 0x00000000
名称： HT 总线可预取地址窗口 0 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_prefetch0_en | 1 | 0x0 | R/W | HT 总线可预取地址窗口 0，使能信号 |
| 30:23 | Reserved | 15 | 0x0 | | 保留 |
| 15:0 | ht_prefetch0_trans [39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 0，转译后地址的 [39:24] |

偏移量： 0xe4
复位值： 0x00000000
名称： HT 总线可预取地址窗口 0 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_prefetch0_base[39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 0，地址基址的[39:24]位地址 |
| 15:0 | ht_prefetch0_mask[39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 0，地址屏蔽的[39:24] |

偏移量：0xe8
复位值：0x00000000
名称：HT 总线可预取地址窗口 1 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_prefetch1_en | 1 | 0x0 | R/W | HT 总线可预取地址窗口 1，使能信号 |
| 30:23 | Reserved | 15 | 0x0 | | 保留 |
| 15:0 | ht_prefetch1_trans[39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 1，转译后地址的[39:24] |

偏移量：0xec
复位值：0x00000000
名称：HT 总线可预取地址窗口 1 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_prefetch1_base[39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 1，地址基址的[39:24] |
| 15:0 | ht_prefetch1_mask[39:24] | 16 | 0x0 | R/W | HT 总线可预取地址窗口 1，地址屏蔽的[39:24] |

## 9.5.10 UNCACHE地址窗口配置寄存器

本控制器中的地址窗口命中公式如下：

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

值得一提的是，配置地址窗口寄存器时，MASK 高位应全为 1，低位应全为 0。MASK 中 0 的实际位数表示的就是地址窗口的大小。

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。

偏移量：0xf0
复位值：0x00000000
名称：HT 总线 Uncache 地址窗口 0 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_uncache0_en | 1 | 0x0 | R/W | HT 总线 uncache 地址窗口 0，使能信号 |
| 30 | ht_uncache0_trans_en | 1 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，映射使能信号 |
| 29:23 | Reserved | 14 | 0x0 | | 保留 |
| 15:0 | ht_uncache0_trans[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 0，转译后地址的[39:24] |

偏移量：0xf4
复位值：0x00000000
名称：HT 总线 Uncache 地址窗口 0 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_uncache0_base[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 0，地址基址的[39:24] |
| 15:0 | ht_uncache0_mask[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 0，地址屏蔽的[39:24] |

偏移量：0xf8
复位值：0x00000000
名称：HT 总线 Uncache 地址窗口 1 使能（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31 | ht_uncache1_en | 1 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，使能信号 |
| 30 | ht_uncache1_trans_en | 1 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，映射使能信号 |
| 29:23 | Reserved | 14 | 0x0 | | 保留 |
| 15:0 | ht_uncache1_trans[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，转译后地址的[39:24] |

偏移量：0xfc
复位值：0x00000000
名称：HT 总线 Uncache 地址窗口 1 基址（内部访问）

| 位域 | 位域名称 | 位宽 | 复位值 | 访问 | 描述 |
|---|---|---|---|---|---|
| 31:16 | ht_uncache1_base[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，地址基址的[39:24] |
| 15:0 | ht_uncache1_mask[39:24] | 16 | 0x0 | R/W | HT 总线 uncache 地址窗口 1，地址屏蔽的[39:24] |

### 9.5.11 HyperTransport总线配置空间的访问方法

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_FFFFh。对于 PCI 协议中的配置访问，在龙芯 2 号中如下实现。

Type 0:

| 39 — 24 | 23 — 16 | 15 — 11 | 10 — 8 | 7 — 2 |
|---|---|---|---|---|
| FDFEh | Reserved | Device Number | Function Number | Register Number |

Type 1:

| 39 — 24 | 23 — 16 | 15 — 11 | 10 — 8 | 7 — 2 |
|---|---|---|---|---|
| FDFFh | Bus Number | Device Number | Function Number | Register Number |

图 9-1 龙芯 2 号中 HT 协议的配置访问

# 10 低速 IO 控制器配置

龙芯 2G 的低速 I/O 控制器包括 LPC 控制器、UART 控制器、SPI 控制器、GPIO 以及配置寄存器。这些 I/O 控制器共享一个 AXI 端口，CPU 的请求经过地址译码后发送到相应的设备。

## 10.1 LPC 控制器

LPC 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

LPC 控制器的地址空间分布见表 10-1：

表 10-1 LPC 控制器地址空间分布

| 地址名称 | 地址范围 | 大小 |
|---|---|---|
| LPC Boot | 0X1FC0_0000-0X1FD0_0000 | 1MByte |
| LPC Memory | 0X1C00_0000-0X1E00_0000 | 32MByte |
| LPC I/O | 0X1FF0_0000-0X1FF1_0000 | 64KByte |
| LPC Register | 0X1FE0_0200-0X1FE0_0300 | 256Byte |

LPC Boot 地址空间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 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。LPC_ROM_INTEL 引脚上拉时发出 LPC

Firmware Memory 访问，LPC_ROM_INTEL 引脚下拉时发出 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 Memory 访问，由 LPC 控制器的配置寄存器 LPC_MEM_IS_FWH 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 LPC_MEM_TRANS 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。地址为地址空间低 16 位。

LPC 控制器配置寄存器共有 3 个 32 位寄存器。配置寄存器的含义见表 10-2:

表 10-2 LPC 配置寄存器含义

| 位域 | 字段名 | 访问 | 复位值 | 说明 |
|---|---|---|---|---|
| REG0 | | | | |
| REG0[31:31] | SIRQ_EN | 读写 | 0 | SIRQ 使能控制 |
| REG0[23] | LPC_MEM_TRANS_EN | 读写 | 0 | LPC Memory 空间地址转换使能 |
| REG0[22:16] | 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[31:31] | LPC_MEM_IS_FWH | 读写 | 0 | LPC Memory 空间 Firmware Memory 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 |

## 10.2 UART 控制器

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，功能寄存器完全一样，只是访问基址不同。

UART0 寄存器物理地址基址为 0x1FE001E0。

UART1 寄存器物理地址基址为 0x1FE001E8。

### 10.2.1 数据寄存器（DAT）

中文名： 数据传输寄存器

寄存器位宽： [7：0]

偏移量： 0x00

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:0 | Tx FIFO | 8 | W | 数据传输寄存器 |

### 10.2.2 中断使能寄存器（IER）

中文名： 中断使能寄存器

寄存器位宽： [7：0]

偏移量： 0x01

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 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 |

### 10.2.3 中断标识寄存器（IIR）

中文名： 中断源寄存器

寄存器位宽： [7：0]

偏移量： 0x02

复位值： 0xc1

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 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 |

中断控制功能表

| Bit 3 | Bit 2 | Bit 1 | 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 |
| 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 |

### 10.2.4 FIFO控制寄存器（FCR）

中文名： FIFO 控制寄存器

寄存器位宽： [7：0]

偏移量： 0x02

复位值： 0xc0

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:6 | TL | 2 | W | 接收 FIFO 提出中断申请的 trigger 值<br>‘00’ – 1 字节 ‘01’ – 4 字节<br>‘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 |

### 10.2.5 线路控制寄存器（LCR）

中文名： 线路控制寄存器

寄存器位宽： [7：0]

偏移量： 0x03

复位值： 0x03

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7 | dlab | 1 | RW | 分频锁存器访问位<br>‘1’ – 访问操作分频锁存器<br>‘0’ – 访问操作正常寄存器 |
| 6 | bcb | 1 | RW | 打断控制位<br>‘1’ – 此时串口的输出被置为 0(打断状态).<br>‘0’ – 正常操作 |
| 5 | spb | 1 | RW | 指定奇偶校验位<br>‘0’ – 不用指定奇偶校验位<br>‘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<br>‘0’ – 在每个字符中有奇数个 1（包括数据和奇偶校验位）<br>‘1’ –在每个字符中有偶数个 1 |
| 3 | pe | 1 | RW | 奇偶校验位使能<br>‘0’ – 没有奇偶校验位<br>‘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<br>‘0’ – 1 个停止位<br>‘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<br>‘00’ – 5 位 ‘01’ – 6 位<br>‘10’ – 7 位 ‘11’ – 8 位 |

### 10.2.6 MODEM控制寄存器（MCR）

中文名： Modem 控制寄存器

寄存器位宽： [7：0]

偏移量： 0x04

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:5 | Reserved | 3 | W | 保留 |
| 4 | Loop | 1 | W | 回环模式控制位<br>‘0’ – 正常操作<br>‘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。<br>DTR ➔ DSR<br>RTS ➔ CTS<br>Out1 ➔ RI<br>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 |

### 10.2.7 线路状态寄存器（LSR）

中文名： 线路状态寄存器

寄存器位宽： [7：0]

偏移量： 0x05

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7 | ERROR | 1 | R | 错误表示位<br>‘1’ – 至少有奇偶校验位错误，帧错误或 |

| | | | | |
|---|---|---|---|---|
| | | | | 打断中断的一个。<br>‘0’ – 没有错误 |
| 6 | TE | 1 | R | 传输为空表示位<br>‘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<br>‘0’ – 有数据 |
| 5 | TFE | 1 | R | 传输 FIFO 位空表示位<br>‘1’ – 当前传输 FIFO 为空，给传输 FIFO 写数据时清零<br>‘0’ – 有数据 |
| 4 | BI | 1 | R | 打断中断表示位<br>‘1’ –接收到 起始位＋数据＋奇偶位＋停止位都是 0，即有打断中断<br>‘0’ – 没有打断 |
| 3 | FE | 1 | R | 帧错误表示位<br>‘1’ – 接收的数据没有停止位<br>‘0’ – 没有错误 |
| 2 | PE | 1 | R | 奇偶校验位错误表示位<br>‘1’ – 当前接收数据有奇偶错误<br>‘0’ – 没有奇偶错误 |
| 1 | OE | 1 | R | 数据溢出表示位<br>‘1’ – 有数据溢出<br>‘0’ – 无溢出 |
| 0 | DR | 1 | R | 接收数据有效表示位<br>‘0’ – 在 FIFO 中无数据<br>‘1’ – 在 FIFO 中有数据 |

对这个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。

### 10.2.8 MODEM状态寄存器 （MSR）

中文名： Modem 状态寄存器

寄存器位宽： [7：0]

偏移量： 0x06

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 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 |

### 10.2.9 分频锁存器

中文名： 分频锁存器 1

寄存器位宽： [7：0]

偏移量： 0x00

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:0 | LSB | 8 | RW | 存放分频锁存器的低 8 位 |

中文名： 分频锁存器 2

寄存器位宽： [7：0]

偏移量： 0x01

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:0 | MSB | 8 | RW | 存放分频锁存器的高 8 位 |

## 10.3 SPI 控制器

SPI 控制器具有以下特性：

- 全双工同步串口数据传输
- 支持 1 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。

### 10.3.1 控制寄存器（SPCR）

中文名： 控制寄存器

寄存器位宽： [7：0]

偏移量： 0x00

复位值： 0x10

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 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 | Reserved | 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 | sclk_o 分频设定，需要与 sper 的 spre 一起 |

| | | | | |
|---|---|---|---|---|
| | | | | 使用 |

### 10.3.2 状态寄存器（SPSR）

中文名： 状态寄存器

寄存器位宽： [7：0]

偏移量： 0x01

复位值： 0x05

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 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 | Reserved | 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 |

### 10.3.3 数据寄存器（TxFIFO）

中文名： 数据传输寄存器

寄存器位宽： [7：0]

偏移量： 0x02

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:0 | Tx FIFO | 8 | W | 数据传输寄存器 |

### 10.3.4 外部寄存器（SPER）

中文名： 外部寄存器

寄存器位宽： [7：0]

偏移量： 0x03

复位值： 0x00

| 位域 | 位域名称 | 位宽 | 访问 | 描述 |
|---|---|---|---|---|
| 7:6 | icnt | 2 | RW | 在传输完多少个字节后送出中断申请信号<br>00 – 1 字节　　01 - 2 字节<br>10 - 3 字节　　11 - 3 字节 |
| 5:2 | Reserved | 4 | RW | 保留 |
| 1:0 | spre | 2 | RW | 与 Spr 一起设定分频的比率 |

分频系数：

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

## 10.4 IO 控制器配置

配置寄存器主要用于配置芯片内部一些控制寄存器器及 GPIO 控制。表 10-6 列出了这些寄存器，表 10-7 给出寄存器的详细说明。这部分寄存器基地址为 0x1FE00100。

表 10-3 IO 控制寄存器

| 地　址 | 寄存器 | 说　明 |
|---|---|---|
| 00 | PonCfg | 上电配置 |
| 04 | GenCfg | 常规配置 |
| 08 | 保留 | |
| 0C | 保留 | |
| 10 | 保留 | |
| 14 | 保留 | |
| 18 | 保留 | |
| 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 | 保留 | |
| 44 | 保留 | |
| 48 | 保留 | |
| 4C | 保留 | |

| | | |
|---|---|---|
| 50 | 保留 | |
| 54 | 保留 | |
| 58 | 保留 | |
| 5C | 保留 | |
| 60 | 保留 | |
| 64 | 保留 | |
| 68 | 保留 | |
| 6C | 保留 | |
| 70 | 保留 | |
| 74 | 保留 | |
| 78 | 保留 | |
| 7C | 保留 | |
| 80 | Chip Config | 芯片配置寄存器 |
| 84 | | |
| 88 | | |
| 8C | | |
| 90 | Chip Sample | 芯片采样寄存器 |

表 10-4 寄存器详细描述

| 位域 | 字段名 | 访问 | 复位值 | 说明 |
|---|---|---|---|---|
| CR00: PonCfg | | | | |
| 15:0 | 保留 | 只读 | | |
| 15:8 | 保留 | 只读 | | |
| 23:16 | pon_sys_configi | 只读 | pci_configi | SYS_Configi 引脚值 |
| 31:24 | 保留 | 只读 | | |
| CR04: 保留 | | | | |
| 31:0 | 保留 | 只读 | 0 | |
| CR08: 保留 | | | | |

| | | | | |
|---|---|---|---|---|
| 31:0 | 保留 | 只读 | 0 | |
| CR10: 保留 | | | | |
| 31:0 | 保留 | 只读 | 0 | |
| CR1C: GPIO_Data | | | | |
| 3:0 | gpio_out | 读写 | 0 | GPIO 输出数据 |
| 15:4 | 保留 | 只读 | 0 | |
| 19:16 | gpio_in | 读写 | 0 | GPIO 输入数据 |
| 31:20 | 保留 | 只读 | 0 | |
| CR20: GPIO_EN | | | | |
| 3:0 | gpio_en | 读写 | F | 高为输入，低输出 |
| 31:4 | 保留 | 只读 | 0 | |
| CR3C: 保留 | | | | |
| 31:0 | 保留 | 只读 | 0 | 保留 |
| CR24,2C,30,34,38:保留 | | | | |
| CR80: Chip config | | | | |
| 2:0 | Freq_scale_ctrl | 读写 | 3’b111 | 处理器核分频 |
| 3 | DDR_Clksel_en | 读写 | 1’b0 | 是否使用软件配置 DDR 倍频 |
| 8 | Disable_ddr2_confspace | 读写 | 1’b0 | 是否禁用 DDR 配置空间 |
| 9 | DDR_buffer_cpu | 读写 | 1’b0 | 是否打开 DDR 读访问缓冲 |
| 12 | Core0_en | 读写 | 1’b1 | 是否启用处理器核 0 |
| 13 | Core1_en | 读写 | 1’b1 | 是否启用处理器核 1 |
| 14 | Core2_en | 读写 | 1’b1 | 是否启用处理器核 2 |
| 15 | Core3_en | 读写 | 1’b1 | 是否启用处理器核 3 |
| 16 | Mc0_en | 读写 | 1’b1 | 是否启用 DDR 控制器 0 |
| 17 | Mc1_en | 读写 | 1’b1 | 是否启用 DDR 控制器 1 |
| 18 | DDR_reset0 | 读写 | 1’b1 | 软件 reset DDR 控制器 0 |
| 19 | DDR_reset1 | 读写 | 1’b1 | 软件 reset DDR 控制器 1 |
| 28:24 | DDR_Clksel | 读写 | 5’b11111 | 软件配置 DDR 时钟倍频关系（当 DDR_Clksel_en 为 1 时有效） |
| 31:29 | HT_freq_scale_ctrl0 | 读写 | 3’b111 | HT 控制器分频 |
| | | | | |
| 其它 | | 只读 | | 保留 |
| CR90: Chip Sample | | | | |
| 15:0 | Pad2v5_ctrl | 读写 | 16’h780 | 2v5pad 控制 |
| 31:16 | Pad3v3_ctrl | 读写 | 16’h780 | 3v3pad 控制 |
| 47:32 | Sys_clksel | 只读 | | 板上倍频设置 |
| 51:48 | Bad_ip_core | 只读 | | 4 个处理器核是否坏 |
| 53:52 | Bad_ip_ddr | 只读 | | 2 个 DDR 控制器是否坏 |

| | | | | |
|---|---|---|---|---|
| 57 | Bad_ip_ht | 只读 | | HT 控制器是否坏 |
| 102:96 | Thsens0_out | 只读 | | 温度传感器 0 温度 |
| 103 | Thsens0_overflow | 只读 | | 温度传感器 0 温度上溢（超过 128 度） |
| 110:104 | Thsens1_out | 只读 | | 温度传感器 1 温度 |
| 111 | Thsens1_overflow | 只读 | | 温度传感器 1 温度上溢（超过 128 度） |
| 其它 | | 只读 | | 保留 |

# 第二部分

## 系统软件编程指南

# 11 中断的配置及使用

## 11.1 中断的流程

龙芯 2G 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。图 11-1 给出的就是 2G-690e 板卡的中断处理的流程图。

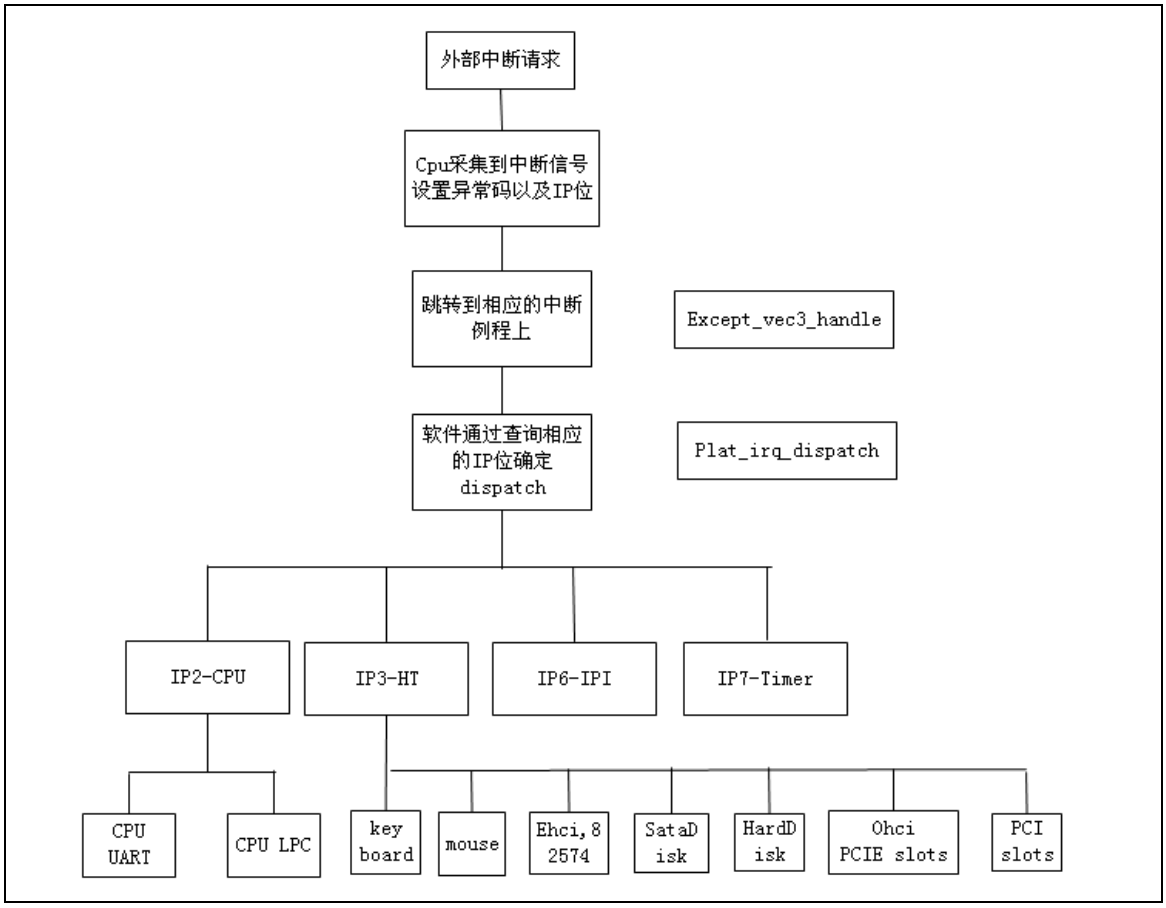



图 11-1 2G-690e 中断流程图

## 11.2 中断路由及中断使能

龙芯 2G 处理器支持最多 32 个中断源，其中部分被保留不作使用，以统一方式进行管理，如图 11-2 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。

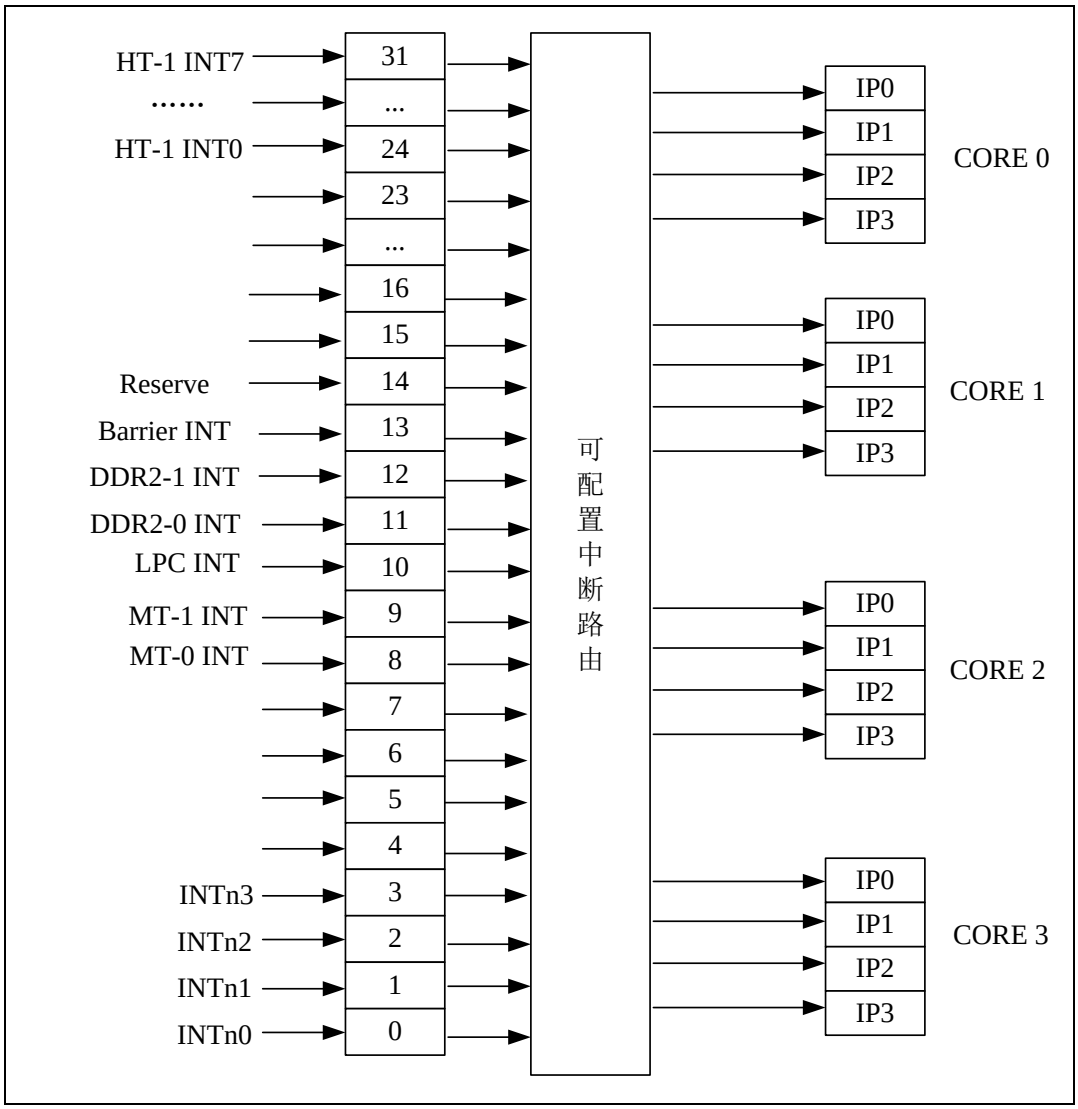



图 11-2 龙芯 2G 处理器中断路由示意图

## 11.2.1 中断路由

在龙芯 2G 中最多集成 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。也就是说图 11-2 所示 CORE0~CORE3 的 IP0~IP3 对应的就是 CP0_Status 的 IP2~IP5。32 个 I/O 中断源中每一个都对应一个 8 位的路由控制器，其格式和地址如下表 1 和表 2 所示。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。

表 11-1 中断路由寄存器的说明

| 位域 | 说　明 |
|---|---|

| | |
|---|---|
| 3:0 | 路由的处理器核向量（可以同时路由给多个处理器） |
| 7:4 | 路由的处理器核中断引脚向量号 |

表 11-2 中断路由寄存器地址

| 名称 | 地址偏移 | 描述 | 名称 | 地址偏移 | 描述 |
|---|---|---|---|---|---|
| Entry0 | 0x1400 | Sys_int0 | Entry16 | 0x1410 | 保留 |
| Entry1 | 0x1401 | Sys_int1 | Entry17 | 0x1411 | 保留 |
| Entry2 | 0x1402 | Sys_int2 | Entry18 | 0x1412 | 保留 |
| Entry3 | 0x1403 | Sys_int3 | Entry19 | 0x1413 | 保留 |
| Entry4 | 0x1404 | 保留 | Entry20 | 0x1414 | 保留 |
| Entry5 | 0x1405 | 保留 | Entry21 | 0x1415 | 保留 |
| Entry6 | 0x1406 | 保留 | Entry22 | 0x1416 | 保留 |
| Entry7 | 0x1407 | 保留 | Entry23 | 0x1417 | 保留 |
| Entry8 | 0x1408 | Matrix int0 | Entry24 | 0x1418 | HT1-int0 |
| Entry9 | 0x1409 | Matrix int1 | Entry25 | 0x1419 | HT1-int1 |
| Entry10 | 0x140a | Lpc int | Entry26 | 0x141a | HT1-int2 |
| Entry11 | 0x140b | Mc0 | Entry27 | 0x141b | HT1-int3 |
| Entry12 | 0x140c | Mc1 | Entry28 | 0x141c | 保留 |
| Entry13 | 0x140d | Barrier | Entry29 | 0x141d | 保留 |
| Entry14 | 0x140e | 保留 | Entry30 | 0x141e | 保留 |
| Entry15 | 0x140f | 保留 | Entry31 | 0x141f | 保留 |

为了便于理解，下面会给出 2G 板卡在中断路由上的配置情况：

硬件连接为“CPU 串口 + HT 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;
```

### 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。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。

表 11-3 中断控制位连接及属性配置

| 名称 | 地址偏移 | 描述 |
|---|---|---|
| Intisr | 0x1420 | 32 位中断状态寄存器 |
| Inten | 0x1424 | 32 位中断使能状态寄存器 |
| Intenset | 0x1428 | 32 位设置使能寄存器 |
| Intenclr | 0x142c | 32 位清除使能寄存器 |
| Intedge | 0x1438 | 32 位触发方式寄存器 |
| CORE0_INTISR | 0x1440 | 路由给 CORE0 的 32 位中断状态 |
| CORE1_INTISR | 0x1448 | 路由给 CORE1 的 32 位中断状态 |
| CORE2_INTISR | 0x1450 | 路由给 CORE2 的 32 位中断状态 |
| CORE3_INTISR | 0x1458 | 路由给 CORE3 的 32 位中断状态 |

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 , LPC (10)  and HT (16~31) */
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 interrupt, only 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 2G-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 HT 的中断。HT 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、sata 等的中断路由到 8259 控制器见 arch/mips/kernel/fixup_ev2G.c 中的 godson2G_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_ev2G.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_ev2G.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。

# 12 串口的配置及使用

## 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。

目前龙芯 2G 处理器上可用的 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 1，存放分频寄存器的低 8 位，计算公式为
                            //工作时钟 /（波特率*16），本例为 33M/(115200*16)
    li      t1,0x0          # divider, highest possible baud rate
    sb      t1,1(a0)        //分频寄存器 2，存放分频寄存器的高 8 位
    li      t1,3
    sb      t1,3(a0)        //数据位为 8 位
```

```
        li      t1,0
        sb      t1,1(a0)        //不使用中断
        li      t1,71
        sb      t1,2(a0)        //设置 FIFO 控制寄存器
        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/ev2G/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/ev2G/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。

# 13 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。图 13-1 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

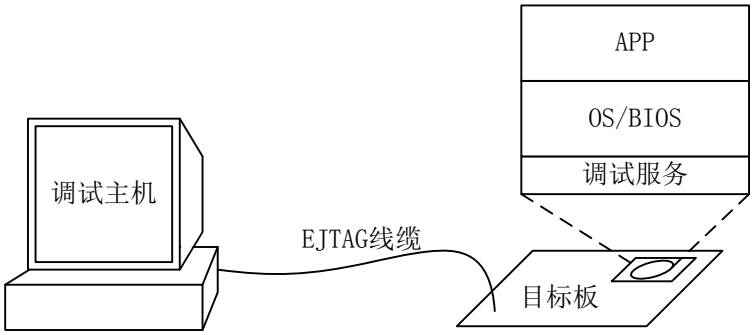



图 13-1 EJTAG 调试系统

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 2G 和 2G 的样片中不能从 dmseg 取指，但可以执行访存指令)。

龙芯 2G 处理器还支持 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_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，分别对应 jtag 线中串联 1 个和 4 个龙芯 2G 处理器。PC 采样不需要调试服务程序。

龙芯 2G 处理器中 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_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 = 0000000000000000f, 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 dmseg 指针
    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 cp0 寄存器
    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)|(((dlsb)
&0x1f)<<6)|(0x2))
#define dinsu(dest, src, dmsb, dlsb) \
    .word
((0x1f<<26)|((src&0x1f)<<21)|((dest&0x1f)<<16)|(((dmsb)&0x1f)<<11)|(((dlsb)&
0x1f)<<6)|(0x6))


    // 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
```

# 14 地址窗口配置转换

龙芯 2G 处理器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：

主端口地址 & MASK == BASE

映射后的从端口地址转换公式如下：

从端口地址 = 主端口地址 & (~(MASK)) | MMAP & MASK

需要注意的是对于一级交叉开关，MMAP[4]与 MMAP[5]必须为 1。而对于二级交叉开关，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。

另外，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。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：

0 号主端口：处理器核 0

1 号主端口：处理器核 1

2 号主端口：处理器核 2

3 号主端口：处理器核 3

7 号主端口：HyperTransport

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：

0 号从端口：二级 Cache 0

1 号从端口：二级 Cache 1

2 号从端口：二级 Cache 2

3 号从端口：二级 Cache 3

7 号从端口：HyperTransport

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。优先级次序如表 14-1：

表 14-1 【表头】

| 优先级 | 窗口号 | |
|---|---|---|
| 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 | 见表 14-2 |

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：

表 14-2 【表头】

| 起始地址 | 结束地址 | 目标 | 说明 |
|---|---|---|---|
| 0x0000_0000_0000 | 0x0BFF_FFFF_FFFF | 二级 Cache | 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 节。<br>例如，<br>当 SCID_SEL = 0 时，<br>0x000：路由至二级 Cache 0<br>0x020：路由至二级 Cache 1<br>0x040：路由至二级 Cache 2<br>0x060：路由至二级 Cache 3<br><br>当 SCID_SEL = 2 时，<br>0x000：路由至二级 Cache 0<br>0x400：路由至二级 Cache 1<br>0x800：路由至二级 Cache 2<br>0xc00：路由至二级 Cache 3 |
| 0x0E00_0000_0000 | 0x0FFF_FFFF_FFFF | HyperTransport | |

## 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

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 LPC 接口、UART 接口、SPI 接口、全系统寄存器空间
- 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_2FFF_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 2G 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 IO 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 IO 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 2G 处理器的 HyperTransport 地址空间如下：

| 基地址 | 结束地址 | 大小 | 定义 | 说明 |
|---|---|---|---|---|
| 0x0E00_0000_0000 | 0x0EFF_FFFF_FFFF | 1 Tbytes | HT 窗口 | |

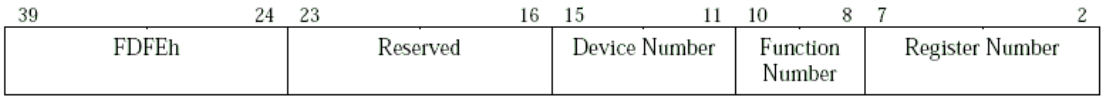
HyperTransport 控制器内拥有 40 位地址空间，按照 HyperTransport 协议，将这 40 位地址分为如下：

| 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 |

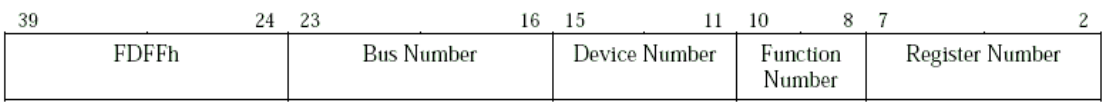
其中 MEM 空间、I/O 空间、HT 总线配置空间分别对应于传统 PCI 空间上的三种访问形式，分别是 PCI MEM 访问、PCI IO 访问与 PCI 配置访问。HT 控制器配置空间主要提供 HT 中断向量及内部窗口配置等功能。

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”以如下规则对地址进行直接读写访问。

Type 0:




Type 1:



### 14.6.2 外部设备对处理器内存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 &&
      (DMA 地址  & ht_rx_image_mask ) == ( ht_rx_image_base & ht_rx_image_mask )

Address_out = ht_rx_image_trans_en ?
                    ht_rx_image_trans | DMA 地址  & ~ht_rx_image_mask : DMA 地址
```

不同的地址窗口优先级依次下降。

### 14.6.3 低速设备地址窗口

低速设备空间包括了 LPC、UART、SPI 等设备，这一空间的内部地址划分如下表：

| 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 |
| IO 寄存器空间 | 0x1FE0_0100 – 0x1FE0_01DF | 256 Byte |
| UART 0 | 0x1FE0_01E0 – 0x1FE0_01E7 | 8 Byte |
| UART 1 | 0x1FE0_01E8 – 0x1FE0_01EF | 8 Byte |
| SPI | 0x1FE0_01F0 – 0x1FE0_01FF | 16 Byte |
| LPC Register | 0x1FE0_0200 – 0x1FE0_02FF | 256 Byte |
| LPC I/O | 0x1FF0_0000 – 0x1FF0_FFFF | 64 Kbyte |

## 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，设备使用 HT 接口连接。

### 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：

窗口 0，将 0x1800_0000 的地址转换为 0x0000_0EFD_FC00_0000，并路由至 HT 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 | |
| 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 |

窗口 1，将 0x1000_0000 的地址转换为 0x0000_0E00_1000_0000，并路由至 HT 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 |

窗口 2，将 0x1E00_0000 的地址转换为 0x0000_0E00_0000_0000，并路由至

HT 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 |

窗口 0 到窗口 2 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。

窗口 3 到窗口 7 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 2G 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。

除了这些异常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 = 2

| | 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 HT 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 |
| 窗口 3 | | | |
| 窗口 4 | | | |
| 窗口 5 | | | |
| 窗口 6 | | | |
| 窗口 7 | | | |

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。

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。

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。

*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 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 | HT MEM 空间 |

| | | | |
|---|---|---|---|
| 地址 2 | 0x0000_0000_1800_0000 | 0x0000_0000_19FF_FFFF | HT IO 空间 |
| 地址 3 | 0x0000_0000_1A00_0000 | 0x0000_0000_1BFF_FFFF | HT 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_1FE0_0100 | 0x0000_0000_1FE0_01DF | IO 寄存器空间 |
| 地址 7 | 0x0000_0000_1FE0_01E0 | 0x0000_0000_1FE0_01E7 | UART 0 |
| 地址 8 | 0x0000_0000_1FE0_01E8 | 0x0000_0000_1FE0_01EF | UART 1 |
| 地址 9 | 0x0000_0000_1FE0_01F0 | 0x0000_0000_1FE0_01FF | SPI |
| 地址 10 | 0x0000_0000_1FE0_0200 | 0x0000_0000_1FE0_02FF | LPC Register |
| 地址 11 | 0x0000_0000_1FF0_0000 | 0x0000_0000_1FF0_FFFF | LPC I/O |
| 地址 12 | 0x0000_0C00_0000_0000 | 0x0000_0FFF_FFFF_FFFF | HT 控制器，各种空间 |
| 地址 13 | 0x0000_1000_0000_0000 | 0x0000_3FFF_FFFF_FFFF | HT 控制器，猜测空间 |
| 地址 14 | 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 | | | |

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。

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。

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。

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。

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 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_1FD0_0000 | 0x0000_0000_1FDF_FFFF | PCI IO 空间 |
| 地址 5 | 0x0000_0000_1FE0_0000 | 0x0000_0000_1FE0_00FF | PCI 控制器配置空间 |
| 地址 6 | 0x0000_0000_1FE0_0100 | 0x0000_0000_1FE0_01DF | IO 寄存器空间 |
| 地址 7 | 0x0000_0000_1FE0_01E0 | 0x0000_0000_1FE0_01E7 | UART 0 |
| 地址 8 | 0x0000_0000_1FE0_01E8 | 0x0000_0000_1FE0_01EF | UART 1 |
| 地址 9 | 0x0000_0000_1FE0_01F0 | 0x0000_0000_1FE0_01FF | SPI |
| 地址 10 | 0x0000_0000_1FE0_0200 | 0x0000_0000_1FE0_02FF | LPC Register |
| 地址 11 | 0x0000_0000_1FF0_0000 | 0x0000_0000_1FF0_FFFF | LPC I/O |
| 地址 12 | 0x0000_0000_8000_0000 | 0x0000_0000_BFFF_FFFF | 内存控制器 0 |
| 地址 13 | 0x0000_0000_C000_0000 | 0x0000_0000_FFFF_FFFF | 内存控制器 1 |
| 地址 14 | 0x0000_0E00_0000_0000 | 0x0000_0FFF_FFFF_FFFF | HT 控制器，各种空间 |
| 地址 15 | 其它地址 | | 系统配置空间 |

# 15 系统内存空间分布设计

## 15.1 系统内存空间

龙芯 2G 处理器拥有两个内存控制器。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，可以使用下面的规则对内存地址空间进行设计。当然，根据系统设计者的需要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） 因此 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：

Base = Size + 0x1000_0000

Limit = Size + Size - 1

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

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 |

如果内存大小为 4GB，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_0001_1000_0000 | 0x0000_0001_FFFF_FFFF | 256MB – 4GB |

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。

| 说明 | | 窗口 | | |
|---|---|---|---|---|
| 用于使能对 BIOS 空间的访问 | | 0 | BASE | 0x00000000_1FC00000 |
| | | | MASK | 0xFFFFFFFF_FFF00000 |
| | | | MMAP | 0x00000000_1FC000F2 |
| 用于使能对低速 IO 空间的访问（仅允许非取指的 UNCACHE 访问通过） | | 1 | BASE | 0x00000000_10000000 |
| | | | MASK | 0xFFFFFFFF_F0000000 |
| | | | MMAP | 0x00000000_10000082 |
| 用于使能对芯片低 256M 空间的访问 | MC0 单通道 256MB 及以上 | 2 | BASE | 0x00000000_00000000 |
| | | | MASK | 0xFFFFFFFF_F0000000 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 3 | | |
| | 双通道 256MB x 2 及以上（以地址[10]做交错） | 2 | BASE | 0x00000000_00000000 |
| | | | MASK | 0xFFFFFFFF_F0000400 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 3 | BASE | 0x00000000_00000400 |
| | | | MASK | 0xFFFFFFFF_F0000400 |
| | | | MMAP | 0x00000000_000000F1 |
| 用于使能对内存高地址空间的访问 | MC0 单通道 256M | 4 | | |
| | | 5 | | |
| | | 6 | | |
| | | 7 | | |
| | MC0 单通道 512M | 4 | BASE | 0x00000000_20000000 |
| | | | MASK | 0xFFFFFFFF_F0000000 |
| | | | MMAP | 0x00000000_100000F0 |
| | | 5 | | |
| | | 6 | | |
| | | 7 | | |
| | MC0 单通道 1G | 4 | BASE | 0x00000000_40000000 |
| | | | MASK | 0xFFFFFFFF_C0000000 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 5 | | |
| | | 6 | | |

| | | | | |
|---|---|---|---|---|
| | | 7 | | |
| | MC0 单通道 2G | 4 | BASE | 0x00000000_80000000 |
| | | | MASK | 0xFFFFFFFF_80000000 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 5 | | |
| | | 6 | | |
| | | 7 | | |
| | 双通道 256M x 2（使用地址[10]交错） | 4 | BASE | 0x00000000_20000000 |
| | | | MASK | 0xFFFFFFFF_F0000400 |
| | | | MMAP | 0x00000000_000004F0 |
| | | 5 | BASE | 0x00000000_20000400 |
| | | | MASK | 0xFFFFFFFF_F0000400 |
| | | | MMAP | 0x00000000_000004F1 |
| | | 6 | | |
| | | 7 | | |
| | 双通道 512M x 2（使用地址[10]交错） | 4 | BASE | 0x00000000_40000000 |
| | | | MASK | 0xFFFFFFFF_E0000400 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 5 | BASE | 0x00000000_40000400 |
| | | | MASK | 0xFFFFFFFF_E0000400 |
| | | | MMAP | 0x00000000_000000F1 |
| | | 6 | BASE | 0x00000000_60000000 |
| | | | MASK | 0xFFFFFFFF_E0000400 |
| | | | MMAP | 0x00000000_000004F0 |
| | | 7 | BASE | 0x00000000_60000400 |
| | | | MASK | 0xFFFFFFFF_E0000400 |
| | | | MMAP | 0x00000000_000004F1 |
| | 双通道 1G x 2（使用地址[10]交错） | 4 | BASE | 0x00000000_80000000 |
| | | | MASK | 0xFFFFFFFF_C0000400 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 5 | BASE | 0x00000000_80000400 |
| | | | MASK | 0xFFFFFFFF_C0000400 |
| | | | MMAP | 0x00000000_000000F1 |
| | | 6 | BASE | 0x00000000_C0000000 |
| | | | MASK | 0xFFFFFFFF_C0000400 |
| | | | MMAP | 0x00000000_000004F0 |
| | | 7 | BASE | 0x00000000_C0000400 |
| | | | MASK | 0xFFFFFFFF_C0000400 |
| | | | MMAP | 0x00000000_000004F1 |
| | 双通道 2G x 2 | 4 | BASE | 0x00000001_00000000 |

| | | | | |
|---|---|---|---|---|
| | （使用地址[10]交错） | | MASK | 0xFFFFFFFF_80000400 |
| | | | MMAP | 0x00000000_000000F0 |
| | | 5 | BASE | 0x00000001_00000400 |
| | | | MASK | 0xFFFFFFFF_80000400 |
| | | | MMAP | 0x00000000_000000F1 |
| | | 6 | BASE | 0x00000001_80000000 |
| | | | MASK | 0xFFFFFFFF_80000400 |
| | | | MMAP | 0x00000000_000004F0 |
| | | 7 | BASE | 0x00000001_80000400 |
| | | | MASK | 0xFFFFFFFF_80000400 |
| | | | MMAP | 0x00000000_000004F1 |

## 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 2G 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，无需地址转换，直接使用即可。

因此，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_0000 | 0x0000_0000_0FFF_FFFF |
| | | DMA 空间 | 0x0000_0000_8000_0000 | 0x0000_0000_8FFF_FFFF |
| 地址 1 | 256MB – 2GB | 系统空间 | 0x0000_0000_9000_0000 | 0x0000_0000_FFFF_FFFF |
| | | DMA 空间 | 0x0000_0000_9000_0000 | 0x0000_0000_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”配置来实现。参见 1.6.2 节。使用两组地址窗口来完成这一转换。

| | 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 |

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。

# 16 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 2G-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。

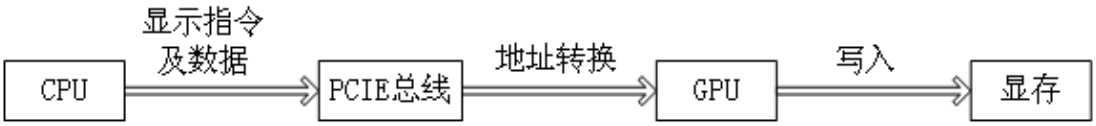



图 16-1 显卡处理图像显示的过程

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 2G-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    # entry_hi，需要映射的虚拟地址的起始地址
        li      a0, 0x3f000000
        bleu    msize, a0, 1f     //判断内存是 1G 还是 2G，如果是 1G，跳到 1 执行
        nop
        li      t4, 0x0000f000     //2G 情况，将 0xf0000000 转到 0xf0000000
        b   2f                     / /跳到 2 执行
        nop
1:
        li      t4, 0x00007000     //1G 情况，将 0xf0000000 转到 0x70000000
2:
        .set mips64
        dsll    t4, t4, 10
        .set mips3
        ori     t4, t4, 0x1f
        li      t5, (0x1000000>>6)              # 16M stride，一页为 16MB
        li      t6, 0x2000000                    # VPN2 32M stride
        .set mips64
1:
        dmtc0   t3, COP_0_TLB_HI            //填 TLB 表项
        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 16x16=256MB
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 0x1000M, 也就是 0x100000000, 如果显存是 128MB，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。