龙芯内存调试手册

修订历史

版本号	更新日期	更新人	更新内容
V1.0	2017-7-28	黄帅	初稿完成

适用范围

芯片型号：

该文档适用于龙芯 3A1500、3A2000、3A3000、3B1500、3B2000、3B3000、2J1500 以及 2K1000 几款芯片。

内存类型：

DDR2/DDR3、UDINN/RDIMM/SODIMM 以及按照 DIMM 方式走线的贴片颗粒

训练程序：

在 PMON 中训练程序文件为 loongson3C_ddr3_leveling.S。该训练程序仅支持 DDR3 类型的内存，由于 DDR2 类型的内存本身不支持 Leveling 操作，所以只能手动配置 DDR2 参数。另外，该训练程序主要针对 UDIMM/RDIMM 进行了适配，如果同一内存通道内不同颗粒之间的走线关系与 UDIMM/RDIMM 存在差别，那么可能需要修改训练程序。

参考文档：

芯片的用户手册中关于内存控制器的章节以及 DDR2/3 协议

1 内存系统简要介绍

1.1 内存基本原理

内存是现代计算机的重要组成部分。内存的读写速度对计算机的整体性能影响重大。为了提升处理器的访存性能，现代处理器采用了大量的结构优化方法。例如，在片内增加了多级的 Cache 层次，或更进一步将内存控制器与 CPU 集成在同一芯片内，以减小平均访存延迟。大部分处理器已经集成内存控制器。

现代内存基本都采用同步动态随机存储器（SDRAM）实现。SDRAM 芯片一般采用行列地址线复用技术，对 SDRAM 进行读写时，需要先发送行地址打开一行，再发送列地址读写需要访问的存储单元。为了提高访问的并发度，SDRAM 芯片一般包含多个 bank，这些 bank 可以并行操作。图 1-1 显示了一个 DDR2 SDRAM x8 芯片的内部结构图。可以看到，该 SDRAM 内部包含了 8 个 bank，每个 bank 对应一个存储阵列（Memory array）和一组感应放大器（Sense amplifiers），所有的 bank 共用读锁存（Read latch）和写 FIFO。对 SDRAM 进行写操作后，由于必须等到写数据从 IO 引脚传送到对应 bank 的感应放大器后，才能进行后续的预充电操作（针对相同 bank）或着读操作（针对所有 bank），因此写操作会给后续的其它操作带来较大的延迟，但连续的写操作却可以流水执行。为了降低写操作带来的开销，内存控制器往往将多个写操作聚集在一起连续发送，以分摊单个写操作的开销。

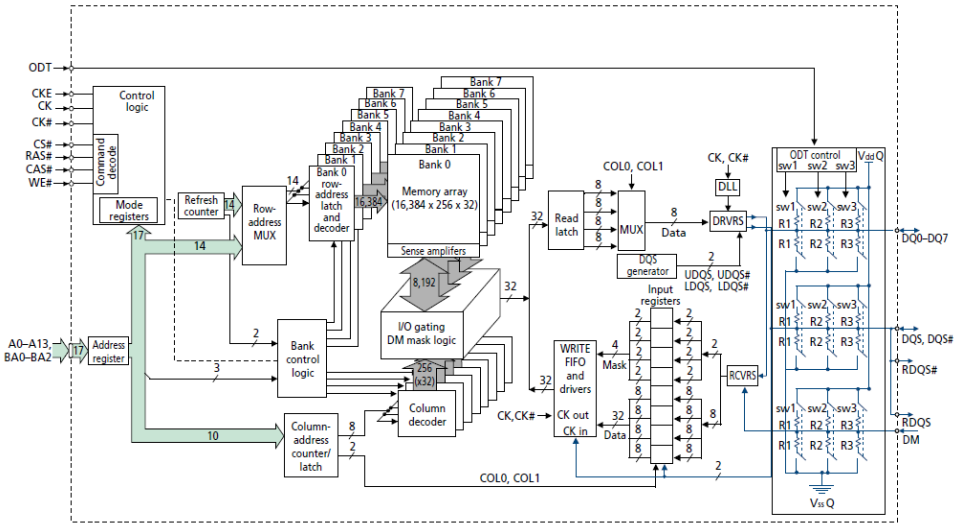


图 1-1 SDRAM 的功能结构图

存在两个影响 SDRAM 芯片读写速度的重要方面：行缓冲局部性（row buffer locality，RBL）和 bank 级并行度（bank level parallelism，BLP）。

行缓冲局部性

如图 1-1 所示，SDRAM 芯片的一行数据在从存储体中读出后，会保存在对应的一组感应放大器中，在访问其他行的数据前，如果访问该行的数据，可以直接从该感应放大器中读出，而不需要重新访问存储体内部，这组感应放大器也被称为行缓冲。命中行缓冲的数据可以大大降低 SDRAM 的访问延迟。当然，在行缓冲不命中而且发生行冲突的时候，就需要首先将行缓冲中的数据写回存储体，也就是 DRAM 中，再将下一行读出到行缓冲中进行访问。

由此，每一个读写请求都可能产生三种不同的状态，导致了访存延迟上的差异。这三种状态分别为：一、行命中，这种情况下，直接进行读写即可，延迟最短；二、行关闭，这种情况下，需要先将该行数据读入行缓冲，再进行读写；三、行冲突，这种情况下，需要先将行缓冲中的数据写回对应的行，再将新地址的数据读入行缓冲，再进行读写，延迟最长。

bank 级并行度

SDRAM 芯片包含的多个 bank 体是相互独立的，它们可以同时执行不同的操作，比如，对 bank 0 激活的同时，可以对 bank 1 发出预充电操作，因此，访问不同 bank 的多个操作可以并行执行。bank 级并行度可以降低冲突命令的等待时间，容忍单个 bank 访问的延迟。

利用内存的这两个特性，可以在内存控制器上对并发访问进行调度，尽可能降低读写访问的平均延迟，提高内存的有效带宽。内存控制器可以对十几甚至几十个访存请求进行调度，有效并发的访存请求数越多，可用于调度的空间就越大，可能得到的访存性能就更优。

1.2 内存接口

内存接口用于连接处理器和主存储器。

前面章节我们介绍了目前使用的主存储器——DRAM 芯片以及内存条、内存控制器的一些概念。内存控制器和内存芯片（或者说内存条）的接口就是内存总线。内存总线规范是由 JEDEC（Joint Electron Device Engineering Council）组织制定的。内存总线规范包含了一般总线的三个层级：机械层、电气层和协议层。

在机械层，JEDEC 规定了内存芯片的封装方式、封装大小和引脚排布，内存条生产厂家可以据此设计内存条 PCB 板，可以使用不同 DRAM 厂家的芯片。同时，JEDEC 也制订了内存条和计算机主板连接的规范，也就是内存插槽规范，规定了内存条的引脚个数、排布和内存条的长度、厚度、机械形式。这样不同厂家的内存条就可以在同一块主板上使用。图 1-2 是台式机使用的 DDR3 内存条和对应的内存插槽的图片。DDR3 内存条使用双列直插式设计，每列分布了 120 个引脚，共 240 个引脚。中间的缺口不是位于内存条的正中心，目的是为了防止将内存条反插。图 1-3 是台式机使用的 DDR2 内存条的图片。DDR3 内存条和

DDR2 内存条的长度大小相同，但内存条上的缺口位置是不同的，以防止 DDR2 和 DDR3 内存条之间的误插。

图 1-2 台式机的 DDR3 内存条和内存插槽

图 1-3 台式机脑的 DDR2 内存条

在电气层，JEDEC 组织规定了 DRAM 芯片的电气特性。例如，DDR2 内存使用 1.8V 电压，而 DDR3 内存使用 1.5V 电压。另外，规范还会规定输入电压高低电平的标准、信号斜率、时钟抖动的范围等信号电气特性。

在协议层，JEDEC 组织规定了 DRAM 芯片的操作时序。协议规定了 DRAM 芯片的上电和初始化过程，DRAM 工作的几种状态，状态之间的转换，以及低功耗控制等内容。比如，DRAM 初始化完成后，进入空闲态，通过激活（activate）命令进入打开某一行激活态，只有在激活态，才可以读写 DRAM 的数据，单纯的读写操作后，DRAM 仍会进入激活态，等待下一次读写。如果想要读写其他行，需要首先发送预充（precharge）命令将 DRAM 转回空闲态，然后再发送激活命令。这些命令不是在任意时刻都可以发送的，需要满足协议规定的时序要求。

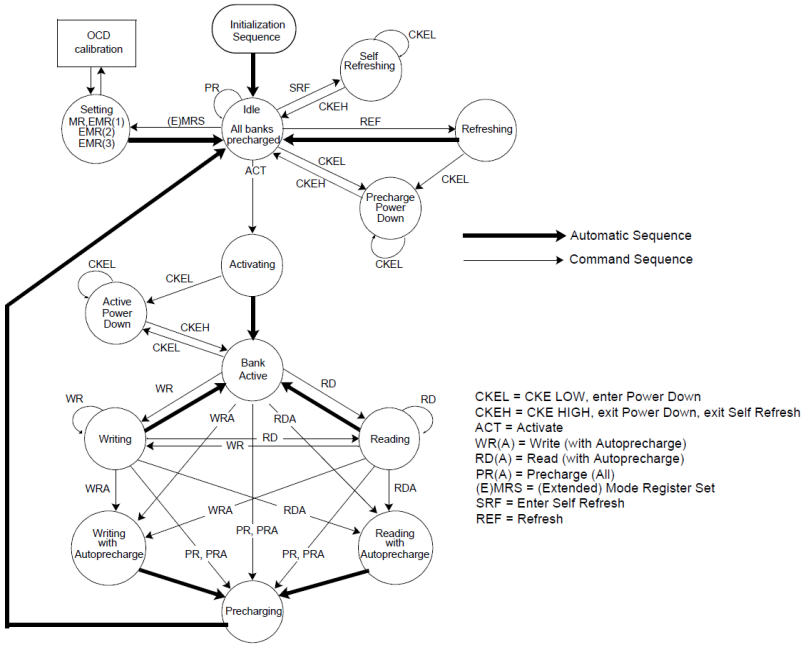


图 1-4 DDR2 内存各状态转换图

表 1-1 双面 DDR3 UDIMM 内存条的接口信号列表

引脚名称	描述	引脚名称	描述
A0-A15	SDRAM 地址线	SCL	EEPROM I^2C 总线时钟
BA0-BA3	SDRAM bank 地址	SDA	EEPROM I^2C 总线数据线
RAS#	SDRAM 行地址选通	SA0-SA2	EEPROM I^2C 从设备地址
CAS#	SDRAM 列地址选通	V_{DD}	SDRAM core 电源
WE#	SDRAM 写使能	$V_{DD}Q$	SDRAM IO 输出电源
S0#-S1#	SDRAM 片选信号	VrefDQ	SDRAM IO 参考电源
CKE0-CKE1	SDRAM 时钟使能信号	VrefCA	SDRAM 命令地址参考电源
ODT0-ODT1	SDRAM 终端匹配电阻控制信号	Vss	电源地信号
DQ0-DQ63	DIMM 内存数据线	V_{DDSPD}	EEPROM 电源
CB0-CB7	DIMM ECC 数据线	NC	空闲引脚
DQS0-DQS8	SDRAM 数据选通线（差分对的正沿）	TEST	测试引脚
DQS0-DQS8	SDRAM 数据选通线（差分对的负沿）	RESET#	复位引脚
DM0-DM8	SDRAM 数据 mask 线	EVENT#	温度传感器引脚（可选）
CK0-CK8	SDRAM 时钟信号线（差分对的正沿）	V_{TT}	SDRAM IO 终端匹配电阻电源
CK0-CK8	SDRAM 时钟信号线（差分对的负沿）	RSVD	保留

DDR3 内存条的接口信号见表 1-1。内存条将多个 DDR3 SDRAM 存储芯片简单地并列在一起，因此表中所列的信号主要是 DDR3 SDRAM 的信号。此外，表中还包含了一组 I2C 总线信号（SCL、SDA）和 I2C 地址信号（SA0-SA2）用来支持内存条的软件识别。内存条将自身的一些设计信息，包括 SDRAM 类型、SDRAM 的速度等级、数据宽度、容量以及机械尺寸标准等信息保存在一个 EEPROM 中，该 EEPROM 可以通过 I2C 总线访问，称为 SPD（Serial Present Detect）。计算机系统可以通过 I2C 总线来读取内存条的信息，从而自动匹配合适的控制参数并获取正确的系统内存容量。组装电脑时，可以选用不同容量、品牌的内存条，无需修改软件或主板，就是通过 SPD 的软件识别来实现的。值得一提的是，表中给出的信号是按照双面内存条带 ECC 功能列出来的，如果只有单面，或者不带 ECC 校验功能，只需将相应的引脚位置悬空。

DRAM 存储单元是按照 bank、行、列来组织的。因此对 DRAM 的寻址是通过 bank 地址、行地址和列地址来进行的。此外，计算机系统中可以将多组 DRAM 串接在一起，不同组之间通过片选（CS）信号来区分。在计算机系统中，程序的地址空间是线性的，处理器发出的内存访问地址也是线性的，由内存控制器负责将该地址转换为对应于 DRAM 的片选、bank、行、列地址。

DDR3 SDRAM 读操作时序如图 1-5 所示。图中命令（Command，简称 CMD）由 RAS_n，CAS_n 和 WE_n 三个信号组成。对于读操作，RAS_n=1，CAS_n＝0，WE_n＝1。该图中，Cas Latency (CL) = 5，Read Latency (RL) = 5，Burst Length = 8。控制器发出 READ 命令后，经过 CL 个时钟周期，SDRAM 开始驱动 DQS 和 DQ 总线输出数据。DQ 数据信号和 DQS 信号是边沿对齐的。在 DQS 的起始，DQ 传输数据之前，DQS 信号会有一个时钟周期长度的低电平，称为读前导（read preamble）。

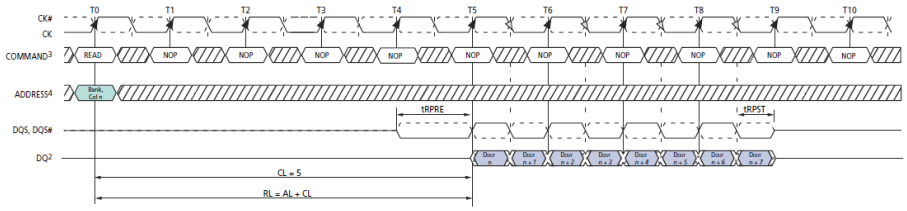


图 1-5 DDR3 SDRAM 读时序

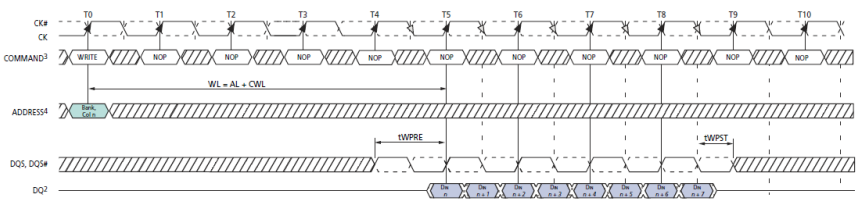


图 1-6 DDR3 SDRAM 写时序

DDR3 SDRAM 写操作的协议如图 1-6 所示。对于写操作，RAS_n=1，CAS_n＝0，WE_n＝0。与读操作不同，写操作需要 DM 来标识写操作的掩码，当 DM 为高时，对应时钟沿的数据并不写入 SDRAM，当 DM 为低时，对应时钟沿的数据才写入 SDRAM。DM 信号与 DQ 信号同步。在写操作时，DQS 信号和 DQ 信号是由控制器驱动的。同样在 DQS 的起始，DQ 数据开始传输之前，DQS 信号也存在一个写前导（write preamble）。DDR3 SDRAM 写前导为一个周期的时钟信号。

前面讲过 SDRAM 的基本操作包括：激活（Activate）、读写（Read/Write）和预充（Precharge）。这些操作之间有一定的时序要求，当 SDRAM 接收到一个操作后，它需要在固定的时钟周期之后开始进行相应的动作，并且这些动作是需要经过一定的时间才能完成的。对于同一个 Bank，不同操作之间是有时间限制的。例如，对于 DDR3-1600 内存来说，激活操作后，需要经过 13.75ns 的时间，才可以发送读写操作。发送读操作后，需要经过至少 7.5ns 的时间才可以发送预充操作。预充操作发送后，需要经过 13.75ns 才可以发送下一个激活操作。因此，对 SDRAM 的读写存在较大的访问延迟。为了掩盖访问延迟，SDRAM 允许针对不同 bank 的操作并发执行。

对内存总线的控制是由内存控制器实现的。内存控制器负责管理内存条的初始化、读写、低功耗控制等操作。内存控制器接收处理器发出的读写命令，将其转化为内存芯片可以识别的 DRAM 操作，并负责处理时序相关问题，最终返回数据（对于读命令）或者返回一个响应（对于写命令）给处理器。对于处理器来说，它只需要发送读写命令给内存控制器就可以了，而不必关心内存的状态以及内存是如何被读写的。

2 龙芯内存控制器主要结构及功能

2.1 龙芯内存控制器概述

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

龙芯处理器支持最大 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 19 位的地址总线（即：16 位的行列地址总线和 3 位的逻辑 Bank 总线）。

龙芯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 16，列地址（CAS_n）数为 15，逻辑体选择（BANK_n）数为 3。

CPU 发送的内存请求物理地址可以根据控制器内部不同的配置进行多种不同的地址映射。

龙芯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
- ECC 功能可以对数据通路上的 1 位和 2 位错误进行检测，并能对对 1 位错误进行自动纠错
- 支持 133-800MHZ 工作频率

2.2 龙芯内存控制器结构

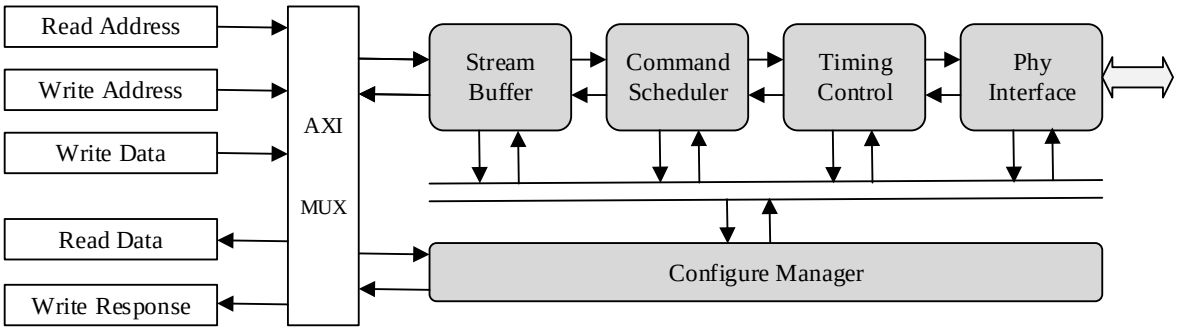


图 2-1 龙芯内存控制器顶层结构

龙芯内存控制器主要结构图如图 2-1 所示，主要模块包括:

streambuffer：用于内存数据预取，以降低内存访问延时、提升内存带宽；

command scheduler：基于优化的调度策略，对接收到的访存命令重排序，以降低访存冲突，提升内存带宽。

timing control：该模块主要按照 JEDEC 协议的要求处理内存接口的时序，以保证访问内存的命令不会违反协议。

PHY interface：该模块主要处理内存接口的信号。内存训练相关的控制、内存读写信号的时序处理以及输出驱动能力的调整、阻抗匹配这些功能都通过该模块实现。

configure manager：管理内存控制器所有的配置寄存器，也是硬件与软件的接口。

2.3 龙芯内存控制器 PHY 结构

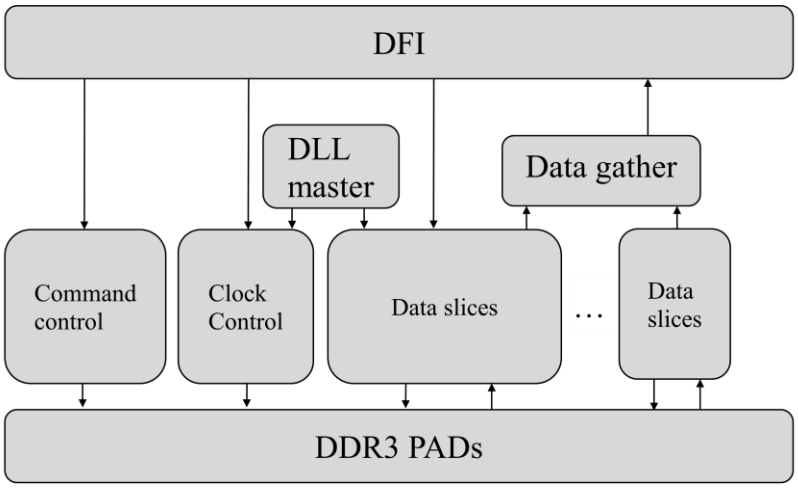


图 2-2 龙芯内存控制器 PHY 结构

PHY 主要结构图如图 **2-2** 所示，主要模块包括:

Dll master:DLL 主控模块，用于计算一个时钟周期内延时单元个数；

Command control:控制发给内存的命令、地址及控制信号，同时实现 2T/3T 功能；

Clock control:控制发送到内存的时钟信号，为了便于调试，实现了内存时钟信号的任意相位调整功能；

Dataslice: 数据控制模块，用于实现内存要求的 DQ-DQS 关系以及 PAD 输出使能的控制等功能。

Datagather:用于收集读返回数据并送给上层逻辑电路。

3 龙芯内存控制器主要参数及作用：

本章对龙芯内存调试过程中经常需要调整的参数进行详细介绍，其他参数请参考芯片对应的用户手册。

3.1 dll_init_done(0x2)

代表内部 DLL 锁定完成。用户可以通过该信号判断内部 DLL 是否已经锁定。由于该信号实时更新，所以在多次读该寄存器时会发现有时为 0 有时为 1，这种情况属正常现象，只要用户从该寄存器读到过 1 即认为锁定成功。

3.2 dll_value_ck(0x4)

代表内部 DLL 锁定值。内部 DLL 由延时单元组成，通过当前时钟周期以及 DLL 锁定值即可计算得到每级延时单元延时值。比如当前时钟周期为 2ns（500MHz），锁定值为 0x32，那么每级延时单元延时值=2ns/0x32=40ps。

3.3 init_start(0x18)

该信号置 1 则内存控制器开始初始化，置 0 复位 PHY 的部分寄存器，内存控制器需要重新初始化。

3.4 dll_bypass(0x19)

DLL 初始化 bypass 控制。当内存频率较低，以至于所有 DLL 延时单元延时值之和小于时钟周期时，DLL 会出现无法锁定的情况，这时需要设置该寄存器，以使内存控制器初始化可以继续进行。正确设置该位的方法是在 Init_start 有效一段时间后再设为 1。

3.5 dll_start_point(0x1a)

DLL 初始化的起始延迟单元个数。当发现 DLL 锁定值有问题时，可以调整该初始值，看问题是否解决。一般不需要调整该参数。

（注：对于 3B1500，如果发现 DLL 的锁定值 dll_value_ck 与初始值 dll_init_start 相同，需要把初始值稍微改小一点，这属于芯片问题）

3.6 dll_increment(0x1b)

每次 DLL 下溢时，起始延迟单元增加个数。该信号同 dll_start_point 一样，都是在 DLL 锁定值有问题时调整。一般不需要调整该参数。

3.7 dll_ck_*(0x1c-0x1f)

DDR 时钟相位调整。该参数主要决定 DDR 时钟相对于 DDR 命令/地址的相位关系。设置为 0 时,DDR 时钟边沿与 DDR 命令/地址边沿对齐，设置为 0x40 时，DDR 时钟处于 DDR 命令/地址窗口中间。

因为内存颗粒在时钟上升沿采集地址/命令，所以该参数的调整会导致 leveling 结果改变。而有时 leveling 结果不理想，调整该参数就会有效解决相应问题。所以该参数一般是调试内存时首选的修改对象。

3.8 dll_gate_*(0x38…)

内部读 gate 信号相对于 rd_oe_start_edge 对应时钟的相位关系。比如 rd_oe_start_edge 设置为 0x0 时，该参数设置内部读 gate 信号相对于 clk_dq 的上升沿的相位。所以在 gate leveling 完成后需要把 dll_gate 的值与对应的 dll_wrdq 的值相加才可以得到读返回 DQS 信号相对于 PHY 内部参考时钟的相位。该参数由内存训练程序在 gate leveling 完成后自动设置。

3.9 dll_wrdqs_*(0x39…)

DDR DQS 基于 PHY 内部参考时钟的相位调整。该寄存器由训练程序根据 DDR3 write leveling 的步骤进行自动配置，具体参考下文对 write leveling 的描述。一般情况下该寄存器不需要人为干预，除非用户在测量信号时发现训练之后 DDR CLK 信号与写 DQS 信号的上升沿没有对齐，那么可以通过下文中手动配置内存参数的方法尝试修改对应的 dll_wrdqs 的值。

3.10 dll_wrdata_*(0x3a…)

DDR DQ 相位调整，与 DDR DQS 一样，DDR DQ 的相位也是基于 PHY 内部参考时钟进行的调整。根据 DDR 协议对 write 操作的要求，如图 3-1 所示，最优情况下 DDR DQS 的上升下降沿处于 DDR DQ 信号的窗口正中间。根据该要求 dll_wrdata_*寄存器的设置需要比对应的 dll_wrdqs_*小 1/4 周期（DLL 非 bypass 情况下，1/4 周期为 0x20）。默认情况下，训练程序在对齐 DDR CLK 与 DQS 之后会自动把 dll_wrdqs_*的值减去 0x20 后写入 dll_wrdata_*。如果测量信

号时发现 DDR DQS-DQ 关系并未处于最优情况下，可以通过修改内存训练程序的宏定义 DLL_WRDQ_SUB 进行调整，参考 5.1.1 节。

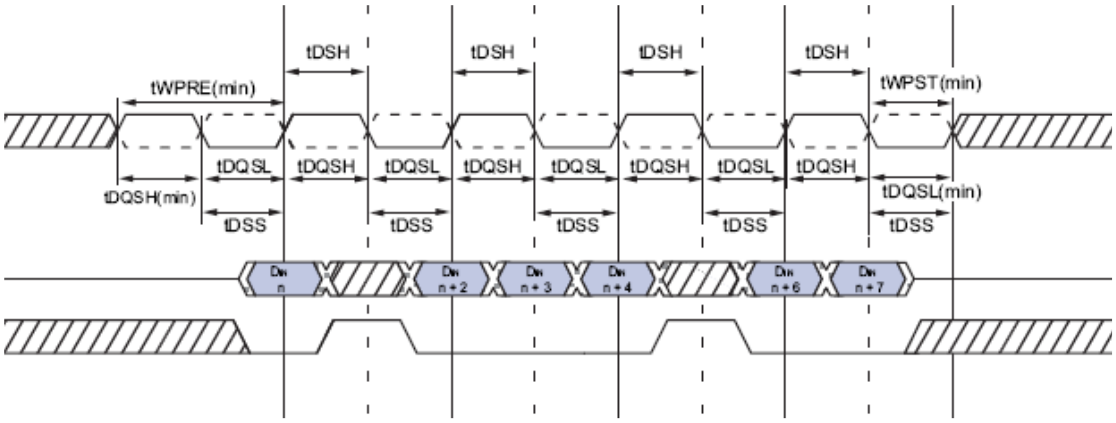


图 3-1 DDR DQS-DQ 关系图

3.11 dll_rddqs_p/n

DDR 协议规定内存返回的读 DQS 与读 DQ 是边沿对齐的关系。为了正确采到数据，需要把返回的读 DQS 向后推，最优的情况是把 DQS 的边沿置于 DQ 数据窗口的正中间。dll_rddqs_p/n 这组参数就是为了实现读 DQS 延时的功能。其中 dll_rddqs_p 用于延迟读 DQS 的上升沿，保证上升沿处于正沿数据的中间；dll_rddqs_n 用于延迟读 DQS 的下降沿，保证下降沿处于负沿数据的中间。通常情况下，这两个参数设置为 0x20，保证读 DQS 向后延迟 90°相位。

3.12 dq_oe_end/dq_oe_begin/dq_stop_edge/dq_start_edge(0x24-0x27,…)

该组参数组合使用，用于调整 DDR DQ 信号的输出使能。其中 dq_oe_end/begin 调整的粒度为 1 个周期，dq_stop/start_edge 调整的粒度为 1/4 个周期。如图 3-2 所示,

dq_stop/start_edge 为 0 时对应 clk_dq 的上升沿；

dq_stop/start_edge 为 1 时对应 clk_dqs 的上升沿；

dq_stop/start_edge 为 2 时对应 clk_dq 的下降沿；

dq_stop/start_edge 为 3 时对应 clk_dqs 的下降沿；

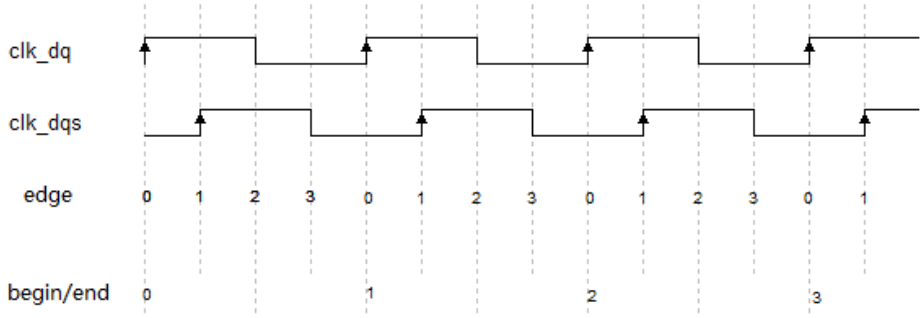


图 3-2 begin/end/edge 图示

在配置时需要保证 dq_oe_begin 与 dq_start_edge 组合得到的时钟边沿不可晚于 dq_oe_end 与 dq_stop_edge 组合得到的时钟边沿。根据 DDR 协议的定义以及龙芯内存控制器 PHY 的设计，DDR 写 DQ 的第一个有效数据从 clk_dq 的上升沿发出，所以理想情况下 dq_stop/start_edge 需要设置成 0。另外，由于写 DQ 信号的窗口大小为 burst_length/2，所以 dq_oe_begin/end 需要设置成相同的值。

但是由于芯片生产以及外部负载的不同，有可能导致内部输出使能信号偏离理想情况。这时就需要根据测量的内存信号对该组参数进行调整。

3.13 dqs_oe_end/dqs_oe_begin/dqs_stop_edge/dqs_start_edge(0x28-0x2b,…)

该组参数用于调整 DDR DQS 信号的输出使能信号，其配置方法同 dq_oe_* 对应参数，可参考。

因为 DDR 写 DQS 信号第一个有效沿为上升沿，从有效变为高阻状态时也对应 clk_dqs 的上升沿，所以理想情况下 dqs_stop/start_edge 需要设置成 0x1。另外，由于 preamble 和 postamble 的存在，写 DQS 信号的窗口大小为 burst_length/2+1，所以 dq_oe_end 需要设置成比 dq_oe_begin 大 0x1。

同样该组参数也可能需要根据内存信号测量结果进行调整。

3.14 rd_oe_end/rd_oe_begin/rd_stop_edge/rd_start_edge(0x2c-0x2f,…)

该组参数用于控制 PHY 内部读 DQS gate 信号的产生逻辑。内存控制器每发出一个对内存的读命令都会对应产生一个读使能信号，用于识别读 DQS 的返回窗口，该窗口通过 gate leveling 过程确定。一般情况下 rd_oe_begin/end 设置成相同值，rd_stop/start_edge 设置成相同值。

3.15 odt_oe_end/odt_oe_begin/odt_stop_edge/odt_start_edge(0x30-0x33,…)

为了实现阻抗匹配，龙芯内存控制器的 PAD 包含了片上电阻，该电阻需要通过读 ODT 信号控制，只在读数据返回时打开。

读 ODT 打开窗口与读 DQS gate 窗口需要保持固定的时序关系。即读 ODT 需要比读 DQS gate 提前半个周期打开，并且需要比读 DQS gate 延迟半个周期关闭。比如在训练完成之后 rd_oe_end/rd_oe_begin/rd_stop_edge/rd_start_edge 的值为 0x02020000，那么 odt_oe_end/odt_oe_begin/odt_stop_edge/odt_start_edge 需要配置成 0x02010202。

3.16 rddata_delay_*(0x23)

这个参数控制读返回数据是否需要延迟一拍。当对性能要求高时可以配置为 0，但有可能导致系统不稳定，具体以实测为准。

3.17 pad_en_clk(0x140)

时钟引脚输出使能，设置为 1 后 DDR CLK 才会有输出。PMON 在初始化内存时会自动配置该寄存器。如果用户无法在 DDR CLK 引脚测量到时钟信号，需要检查该寄存器是否已经设置。

3.18 pad_en_ctl(0x141)

控制引脚输出使能，设置为 1 后 DDR 控制/地址/命令引脚才会有输出。PMON 在初始化内存时会自动配置该寄存器。如果用户无法在 DDR 引脚测量到对应信号，需要检查该寄存器是否已经设置。

3.19 pad_odt_se(0x149)

控制 CPU 端 ODT 值。读信号返回时，为了保证信号完整性，有时需要改变 CPU 端 ODT 的值。

3.20 pad_reset_po(0x156)

复位引脚的控制

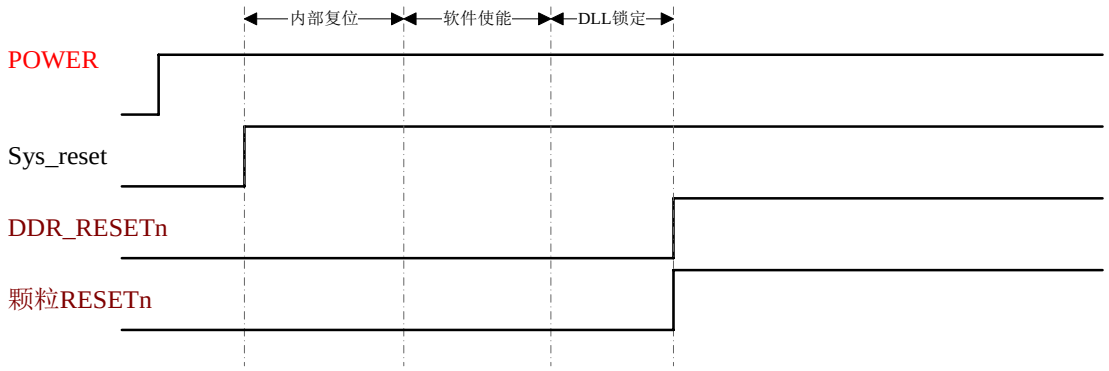
为了在 STR 等状态下更加简单地控制复位引脚，可以通过 reset_ctrl 寄存器进行特别的复位引脚（DDR_RESETn）控制，主要的控制模式有两种：

(1) 一般模式，reset_ctrl[1:0] == 2’b00。这种模式下，复位信号引脚的行为与一般的控制模式相兼容。主板上直接将 DDR_RESETn 与内存槽上的对应引脚相连。引脚的行为是：

- 未上电时：引脚状态为低；

- 上电时：引脚状态为低；
- 控制器开始初始化时，引脚状态为高；
- 正常工作时，引脚状态为高。

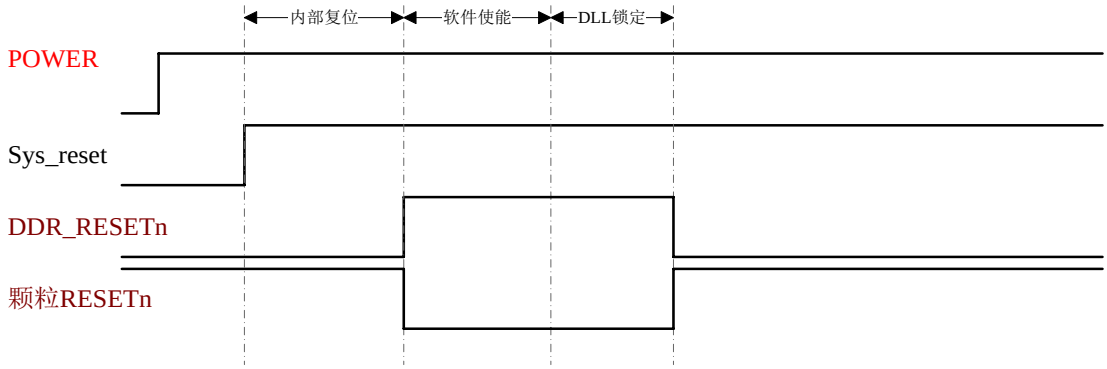
时序如下图所示：



(2) 反向模式，reset_ctrl[1:0] == 2'b10。这种模式下，复位信号引脚在进行内存实际控制的时候，有效电平与一般的控制模式相反。所以主板上需要将 DDR_RESETn 通过反向器与内存槽上的对应引脚相连。引脚的行为是：

- 未上电时：引脚状态为低；
- 上电时：引脚状态为低；
- 控制器开始配置时：引脚状态为高；
- 控制器开始初始化时：引脚状态为低；
- 正常工作时：引脚状态为低。

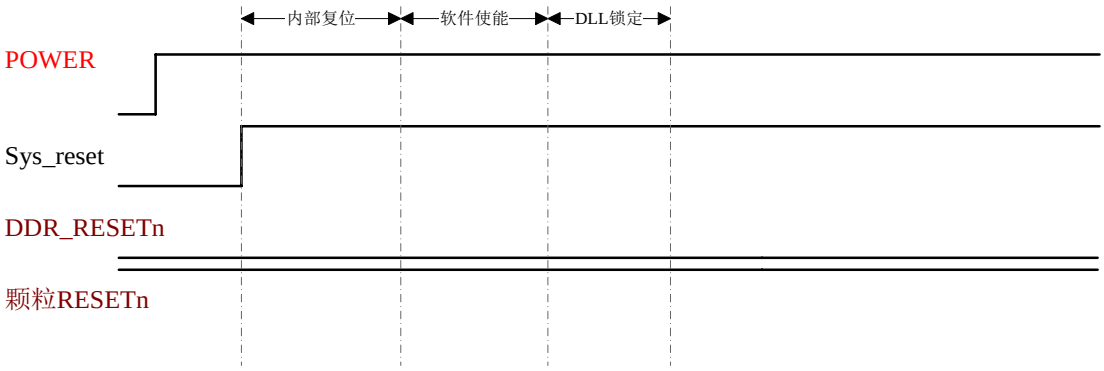
时序如下图所示：



(3) 复位禁止模式，pm_reset_ctrl[1:0] == 2'b01。这种模式下，复位信号引脚在整个内存工作期间，保持低电平。所以主板上需要将 DDR_RESETn 通过反向器与内存槽上的对应引脚相连。引脚的行为是：

- 始终为低；

时序如下图所示：



由后两种复位模式相配合，就可以直接在使用内存控制器的复位信号的情况下实现 STR 控制。当整个系统从关闭状态下启动时，使用（2）中的方法来使用内存条正常复位并开始工作。当系统从 STR 中恢复的时候，使用（3）中的方法来重新配置内存条，使得在不破坏内存条原有状态的条件上使其重新开始正常工作。

3.21 cmd_timing(0x161)

2T/3T 模式:该模式主要为了解决内存的命令、地址等信号驱动能力不够的问题，比如在一个内存通道同时使用了 4 个 CS 的情况可能需要设置内存控制器为该模式。

设置方法为：

- 将 cmd_timing(0x161)设置为 0x1(2T 模式)或者 0x2(3T 模式）
- 然后将 tRDDATA(0x1c0)和 tPHY_WRLAT（0x1d4)分别加一（2T 模式）或者加二（3T 模式）

另外，硬件工程师需要通过观测信号进一步确认是否存在命令、地址等信号驱动能力不够的问题，需要观测的信号为 CLK，CAS_N,RAS_N,WE_N,CS_N

3.22 rdfifo_valid(0x162)

该信号一般情况下设置成 1，在调试模式下才可以设置成 0。有时系统访问内存会出现卡死的情况，这有可能是因为某一个内存颗粒返回的数据个数不够导致。这种情况下可以设置 rdfifo_valid 为 0，由内存控制器在固定延迟之后返回数据，如果看到某个 Byte 的数据出错，那么基本上可以断定问题出在该 Byte 上。其中固定延迟由参数 tPHY_RDLAT(0x1d5)配置，一般设置成比较大的值，比如

0xc~0xf。

3.23 burst_length(0x16c)

该寄存器的值代表内存读写的突发长度，配置时需要保证与内存的配置寄存器 MR0[1:0]的值保持一致。

3.24 odt_*(0x170-0x177)

该组寄存器具体含义如下表所描述：

0x170	位域	读写	默认值	说明
Odt_wr_cs_map	63:48	读写	0x8421	对应 CS 发送写命令时，使能的 ODT 信号 Bit [15:12]：CS3 发写时对应 ODTx 是否有效，x=3..0 Bit [11: 8]：CS2 发写时对应 ODTx 是否有效，x=3..0 Bit [7: 4]：CS1 发写时对应 ODTx 是否有效，x=3..0 Bit [3: 0]：CS0 发写时对应 ODTx 是否有效，x=3..0
Odt_wr_length	43:40	读写	0x5	发送写命令时，ODT 信号有效时钟周期数减一后的值（也就是说 ODT 有效的时间长度等于 odt_wr_length 加 1）
Odt_wr_delay	35:32	读写	0x0	发送写命令时，ODT 信号与写命令的起始间隔
Odt_rd_cs_map	31:16	读写	0x1144	对应 CS 发送读命令时，使能的 ODT 信号 Bit [15:12]：CS3 发读时对应 ODTx 是否有效，x=3..0 Bit [11: 8]：CS2 发读时对应 ODTx 是否有效，x=3..0 Bit [7: 4]：CS1 发读时对应 ODTx 是否有效，x=3..0 Bit [3: 0]：CS0 发读时对应 ODTx 是否有效，x=3..0
Odt_rd_length	11:8	读写	0x5	发送读命令时，ODT 信号有效时钟周期数减一后的值（也就是说 ODT 有效的时间长度等于 odt_rd_length 加 1）
Odt_rd_delay	3:0	读写	0x1	发送读命令时，ODT 信号与读命令的起始间隔

以默认的 odt_wr_cs_map 设置为例：

bit[3:0]设置为 0x1，代表对 CS0 进行写操作时打开 CS0 的 ODT，其余 CS ODT

关闭；

bit[7:4]设置为0x2，代表对CS1进行写操作时打开CS1的ODT，其余CS ODT关闭；

bit[11:8]设置为 0x4，代表对 CS2 进行写操作时打开 CS2 的 ODT，其余 CS ODT 关闭；

bit[15:12]设置为 0x8，代表对 CS3 进行写操作时打开 CS3 的 ODT，其余 CS ODT 关闭；

具体配置需要根据内存信号测量结果来设置。

3.25 lvl_cs(0x183)

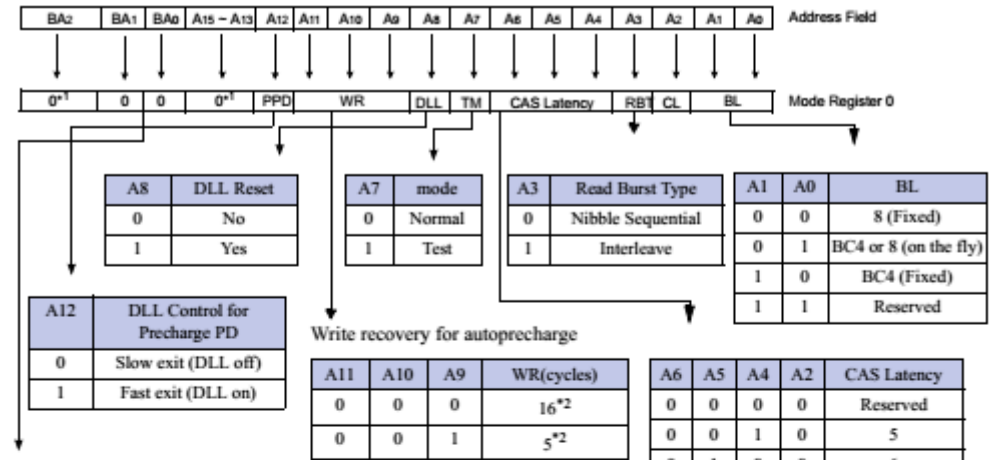
选择进行内存训练的 CS，同时只能有 1 位有效。一般由 PMON 自动配置，用户不需要关心。但在调试时可以通过修改该配置做些对比试验。

3.26 mr_*_cs_*(0x1a0-0x1b8)

该组寄存器分别对应 CS0/1/2/3 的 MR0/1/2/3 寄存器，在内存初始化时，由内存控制器把该组寄存器的值按照初始化序列分别写入对应的内存颗粒 MR 寄存器。

图 3-3 至图 3-6 分别给出了 DDR3 协议定义的 MR0/1/2/3 的寄存器供参考。

需要注意的是，DDR2 内存的配置寄存器名称为 MR，EMR(1)，EMR(2)，EMR(3)，这四个寄存器分别对应控制器的 MR0-3。



A8	DLL Reset
0	No
1	Yes

A7	mode
0	Normal
1	Test

A3	Read Burst Type
0	Nibble Sequential
1	Interleave

A1	A0	BL
0	0	8 (Fixed)
0	1	BC4 or 8 (on the fly)
1	0	BC4 (Fixed)
1	1	Reserved

A12	DLL Control for Precharge PD
0	Slow exit (DLL off)
1	Fast exit (DLL on)

Write recovery for autoprecharge

BA1	BA0	MR Select
0	0	MR0
0	1	MR1
1	0	MR2
1	1	MR3

A11	A10	A9	WR(cycles)
0	0	0	16*2
0	0	1	5*2
0	1	0	6*2
0	1	1	7*2
1	0	0	8*2
1	0	1	10*2
1	1	0	12*2
1	1	1	14*2

A6	A5	A4	A2	CAS Latency
0	0	0	0	Reserved
0	0	1	0	5
0	1	0	0	6
0	1	1	0	7
1	0	0	0	8
1	0	1	0	9
1	1	0	0	10
1	1	1	0	11 (Optional for DDR3-1600)
0	0	0	1	12
0	0	1	1	13
0	1	0	1	14
0	1	1	1	Reserved for 15
1	0	0	1	Reserved for 16
1	0	1	1	Reserved
1	1	0	1	Reserved
1	1	1	1	Reserved

*1: BA2 and A13~A15 are RFU and must be programmed to 0 during MRS.

*2: WR (write recovery for autoprecharge)min in clock cycles is calculated by dividing tWR(in ns) by tCK(in ns) and rounding up to the next integer: WRmin[cycles] = Roundup(tWR[ns] / tCK[ns]). The WR value in the mode register must be programmed to be equal or larger than WRmin. The programmed WR value is used with tRP to determine tDAL.

*3: The table only shows the encodings for a given Cas Latency. For actual supported Cas Latency, please refer to speedbin tables for each frequency

*4: The table only shows the encodings for Write Recovery. For actual Write recovery timing, please refer to AC timingtable.

图 3-3 MR0 寄存器描述

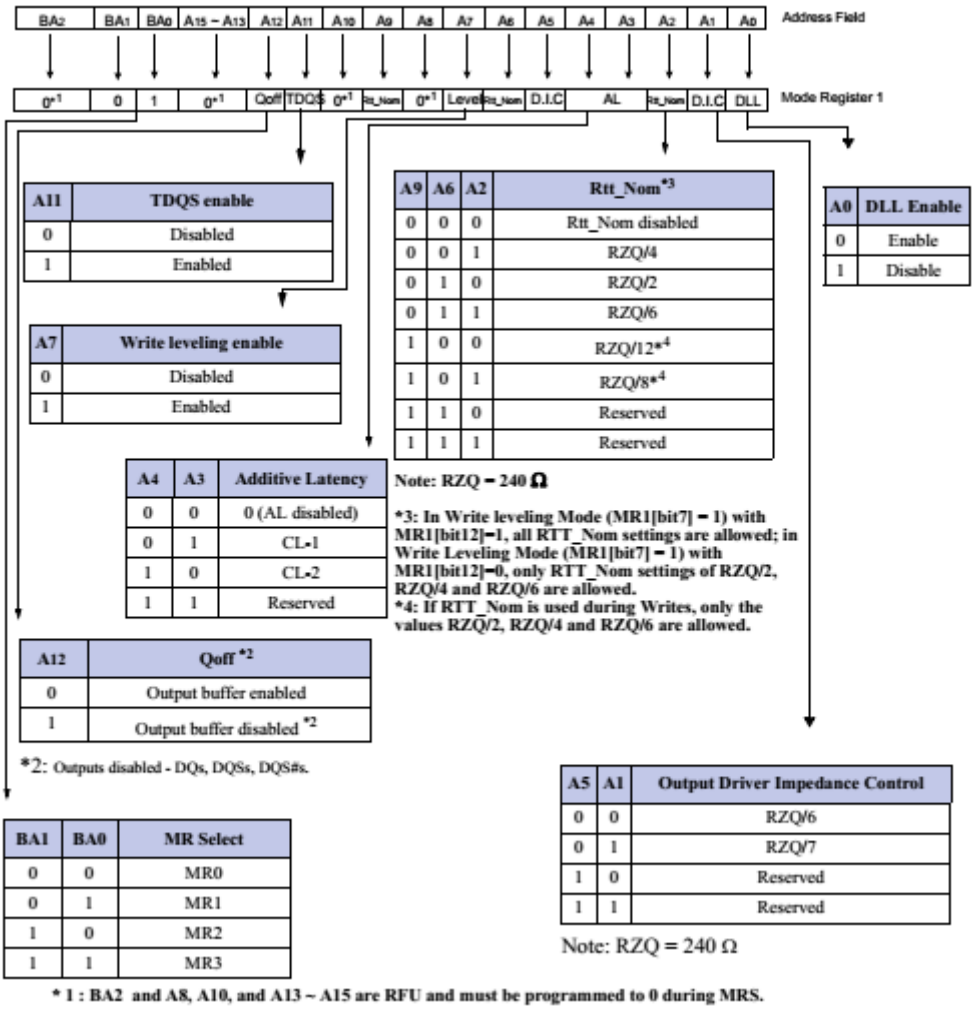
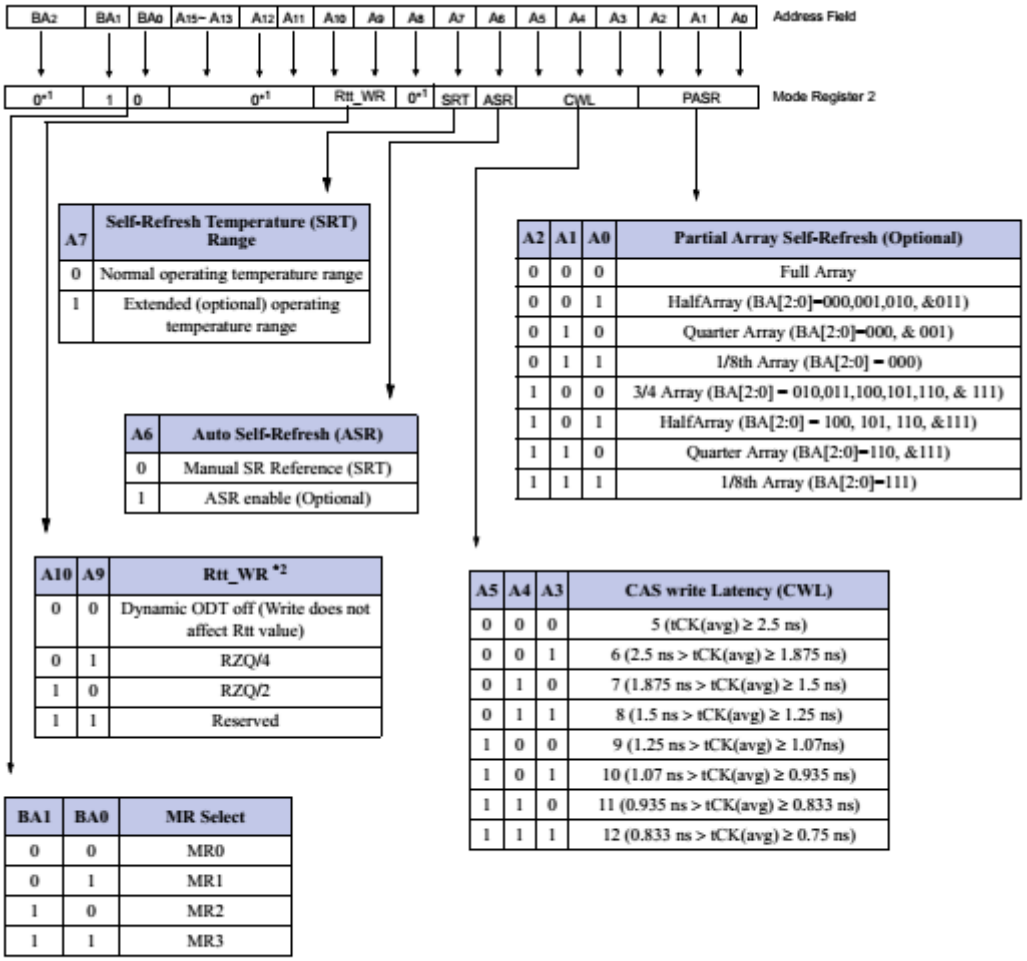


图 3-4 MR1 寄存器描述



A7	Self-Refresh Temperature (SRT) Range
0	Normal operating temperature range
1	Extended (optional) operating temperature range

A6	Auto Self-Refresh (ASR)
0	Manual SR Reference (SRT)
1	ASR enable (Optional)

A10	A9	Rtt_WR *2
0	0	Dynamic ODT off (Write does not affect Rtt value)
0	1	RZQ/4
1	0	RZQ/2
1	1	Reserved

BA1	BA0	MR Select
0	0	MR0
0	1	MR1
1	0	MR2
1	1	MR3

A2	A1	A0	Partial Array Self-Refresh (Optional)
0	0	0	Full Array
0	0	1	HalfArray (BA[2:0]=000,001,010, &011)
0	1	0	Quarter Array (BA[2:0]=000, & 001)
0	1	1	1/8th Array (BA[2:0] = 000)
1	0	0	3/4 Array (BA[2:0] = 010,011,100,101,110, & 111)
1	0	1	HalfArray (BA[2:0] = 100, 101, 110, &111)
1	1	0	Quarter Array (BA[2:0]=110, &111)
1	1	1	1/8th Array (BA[2:0]=111)

A5	A4	A3	CAS write Latency (CWL)
0	0	0	5 (tCK(avg) ≥ 2.5 ns)
0	0	1	6 (2.5 ns > tCK(avg) ≥ 1.875 ns)
0	1	0	7 (1.875 ns > tCK(avg) ≥ 1.5 ns)
0	1	1	8 (1.5 ns > tCK(avg) ≥ 1.25 ns)
1	0	0	9 (1.25 ns > tCK(avg) ≥ 1.07ns)
1	0	1	10 (1.07 ns > tCK(avg) ≥ 0.935 ns)
1	1	0	11 (0.935 ns > tCK(avg) ≥ 0.833 ns)
1	1	1	12 (0.833 ns > tCK(avg) ≥ 0.75 ns)

*** 1 : BA2, A5, A8, A11 ~ A15 are RFU and must be programmed to 0 during MRS.**
*** 2 : The Rtt_WR value can be applied during writes even when Rtt_Nom is disabled. During write leveling, Dynamic ODT is not available.**

图 3-5 MR2 寄存器描述

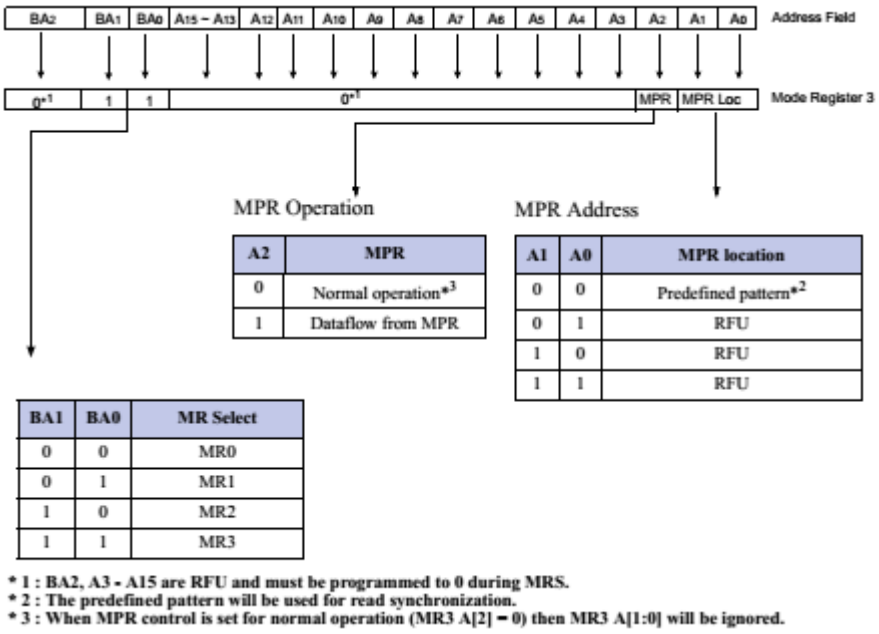


图 3-6 MR3 寄存器描述

3.27 tRDDATA(0x1c0)

从发送读命令到发送读数据有效命令的时间间隔。当 read latency 需要改变时，该寄存器的值需要做相应的调整。比如 PMON 中默认配置的 read latency 为 0xa，当需要把 read latency 缩短两个周期，即 0x8 时，需要同时调整以下几个寄存器：

- 把 tRDDATA 减小 2
- 设置 tRL(0x1db)为 0x8
- 设置 MR0 的 CAS Latency 为 0x8（参考图 3-3）

另外，该参数与 rd_oe_begin/end 共同影响了内部读 gate 信号的延时周期数。当 tRDDATA 增加 1 时，rd_oe_begin/end 需要减小 1。

3.28 REF(0x1cb,0x320)、tRFC(0x1ca)

这两个寄存器分别代表内存的刷新频率及刷新时间。需要注意的时，刷新频率(tREF)只与温度有关，而刷新时间(tRFC)只与内存容量有关，如图 3-7 所示。因为这两个寄存器都以时钟周期为单位，所以在内存频率变化时需要重新计算这两个寄存器的值，计算方法为：

以 2Gb 颗粒为例，工作频率 533MHz，

tREF(0x1cb) = 7.8us/(1/533MHz)/256

tREF(0x320) = 7.8us/(1/533MHz)/16，取低 4 位

tRFC(0x1ca) = 160ns/(1/533MHz)

Table 61 — Refresh parameters by device density

Parameter	Symbol		512Mb	1Gb	2Gb	4Gb	8Gb	Units	Notes
REF command to ACT or REF command time	tRFC		90	110	160	260	350	ns	
Average periodic refresh interval	tREFI	0 °C ≤ T_{CASE} ≤ 85 °C	7.8	7.8	7.8	7.8	7.8	µs	
		85 °C < T_{CASE} ≤ 95 °C	3.9	3.9	3.9	3.9	3.9	µs	1

NOTE 1. Users should refer to the DRAM supplier data sheet and/or the DIMM SPD to determine if DDR3 SDRAM devices support the following options or requirements referred to in this material.

图 3-7 刷新参数

3.29 tPHY_WRLAT(0x1d4)

从发送写命令到发送写数据的时间间隔。当 write latency 需要改变时，该寄存器的值需要做相应的调整。比如 PMON 中默认配置的 write latency 为 0x8，当需要把 write latency 缩短一个周期，即 0x7 时，需要同时调整以下几个寄存器：

a) 把 tPHY_WRLAT 减小 1

b).设置 tWL(0x1da)为 0x7

c).设置 MR2 的 CAS write Latency 为 0x7（参考图 3-5）

3.30 t*_diff*（0x1e0-0x1e8）

配置 read、write 及 precharge 命令之间的附加间隔时间。该组寄存器的值越大，访存请求的间隔越大，访存压力越小。在调试内存时，可以通过把改组寄存器的值增加来判断内存训练的结果是否正确。如果调试内存训练相关的参数总是不能解决问题，把该组寄存器的值加大后问题消失，那么基本可以排除内存训练导致的问题，需要分析是否是因为接口时序不满足或者硬件上的问题。这种方法可以用于所有与时序相关的参数的调整，时序参数的变化对内存带宽的影响可以参考 3.32 小节。

3.31 cs/row/ba/col_diff_*(0x210-0x228)

该组寄存器需要根据内存条的属性来配置。比如一个内存通道上插了 1 根双面 2GB(2RANKx1Gbx8)大小的内存条，这种内存条的 CS 个数为 2，行地址数为 14，列地址数为 10，BANK 数为 3，那么需要配置：

cs_diff_0 = 1；row_diff_0 = 2；ba_diff_0 = 0；col_diff_0 = 6

3.32 时序参数对内存带宽的影响：

值增大会降低内存带宽的参数：

tZQCS(0x1c9)、tRFC(0x1ca)、tRP(0x1cc)、tRCD(0x1cd)、tRRD(0x1ce)、tFAW(0x1cf)、tWTR(0x1d8)、tCCD(0x1d9)、tRTP(0x1dc)、tWR(0x1dd)、t*_diff*(0x1e0-0x1e8)。

值增大会提高内存带宽的参数：

tREF(0x1cb)

4 PMON 中内存控制器相关内容

4.1 内存相关宏定义

以下宏定义除 AUTO_DDR_CONFIG 外，其余都在 start.S 中定义。

- DEBUG_DDR:打开 PMON 下内存测试，如果怀疑内存有问题，可以在 PMON 下对内存进行压力测试，这样可以把一般的问题排除。如果该测试通过，那么起内核就不会有太大问题。
- DEBUG_DDR_PARAM:打开 PMON 下内存参数输入。有时调试内存时需要调整很多参数，打开这个宏定义就可以通过串口即时修改内存参数，省去了重复烧写 PMON 的过程。
- DISABLE_DDR_A15:对于需要用到地址线 A15 的颗粒（比如 512Mbx8），如果主板上没有连接 A15 地址线，那么可以通过 define DISABLE_DDR_A15 来避免使用 A15 地址线，但同时程序会自动把内存容量减半。如果用户发现板上插的单面 4GB 内存条，但是系统下却只显示有 2GB 内存容量，可以先检查是不是定义了这个宏。
- DISABLE_HARD_LEVELING:在有些情况下，为了排除内存训练程序带来的问题，需要手动配置内存参数并绕过内存训练程序，定义该宏定义就可以实现这个功能。比如有时系统重启无法正常开机，但是在能开机的情况下系统稳定运行并可以通过内存测试(stressapptset)，这种情况就可以通过把正常开机情况下的内存参数更新到 loongson_mc2_param.S 文件中（需注意 init_start 寄存器配置为 0）并打开 DISABLE_HARD_LEVELING 来确定有时无法开机的问题是否由内存训练程序导致。如果在写死内存参数的情况下，系统重启问题消失，那么基本上可以确定内存训练程序有问题；反之需要查找其他问题。
- DISABLE_DIMM_ECC：配置是否使能 ECC 功能。当内存条支持 ECC 功能时，可以通过该宏定义打开内存控制器的 ECC 功能。使能 ECC 功能后，程序会自动配置 ecc_enable 寄存器，训练程序也会自动训练 ECC 颗粒。配置完内存参数后，程序也会自动把整个内存地址空间写一遍，用于初始化 ECC 码。
- DDR_DLL_BYPASS：该宏定义会设置内部 dll slave 进入 bypass 模式。
- AUTO_DDR_CONFIG：位于 PMON 的 conf/Bonito.*配置文件中，用于使能自动配置 s1 参数的功能。如果不定义该宏定义，需要手动输入 s1 参数，当内存形态为非正常内存条时，比如内存条没有 SPD 或者内存通道由贴片颗粒组成时需要采用这种方法。

4.2 start.S 中 s1 寄存器的设置

当自动探测内存的开关 AUTO_DDR_CONFIG 打开时需要确认 s1 中关于 i2c 的地址是否与硬件一致，其对应关系为：

s1[31:28] – MC1 SLOT1 的 i2C 地址

s1[27:24] – MC1 SLOT0 的 i2C 地址

s1[23:20] – MC0 SLOT1 的 i2C 地址

s1[19:16] – MC0 SLOT0 的 i2C 地址

给定 i2c 的地址后程序会根据读到的 SPD 信息更新 s1 的值，更新后 s1 的定义见下文。

当自动探测内存的开关 AUTO_DDR_CONFIG 关闭时需要根据具体的内存型号手动配置 s1 的值，需要配置的参数以及在 s1 中定义如下：

S1[31:30]—颗粒类型：2’b10 代表 DDR2，2’b11 代表 DDR3

S1[29] —是否包含 ECC 颗粒：1’b1 代表包含，1’b0 代表不包含

S1[28] —DIMM 类型：1’b1 代表 RDIMM，1’b0 代表 UDIMM

S1[27] —DIMM 数据宽度：1’b1 代表 32 位，1’b0 代表 64 位

S1[26:24]—行地址位数：设置为 16 与实际的行地址位数的差值

S1[23] —BANK 数量：1’b1 代表 8 个 BANK，1’b0 代表 4 个 BANK

S1[22] —ADDR MIRROR：1’b1 代表使能，1’b0 代表不使能

S1[21:20]—列地址位数：设置为 16 与实际的列地址位数的差值

S1[19:16]—CS_MAP：代表对应的 CS 是否连接内存条，分别为 CS3、CS2、CS1 和 CS0

S1[15] —颗粒的数据宽度：1’b1 代表 x16 颗粒，1’b0 代表 x8 颗粒

S1[14：8]—总的内存容量：该值以 512M 为单位

S1[7：4] —保留

S1[3：2]—内存通道选择：2’b00 代表使用两个通道，2’b01 代表仅使用 MC0，2’b10 代表仅使用 MC1

S1[1:0] – 节点号

这里经常出现的问题就是行列地址数错误，这可能会导致地址轮转。比如在

配置内存容量为 1G 时，可能由于行地址少一位导致从地址 0x0 读出写到地址 0x2000_0000 的数据。这种情况下需要确认 s1 的设置、内存控制器的设置（cs_diff_0、row_diff_0、ba_diff_0 和 col_diff_0）是否与硬件一致。

4.3 内存初始化流程：

loongson_mc2_param.S 文件中包含 DDR2、DDR3、UDIMM、RDIMM 的默认配置参数，程序通过内存条的不同类型来选择初始的内存参数。需要注意参数 init_start(0x18)初始值不能置为 1。

配置完初始参数后，程序会根据配置修改部分参数，主要包括：

1) 如果发现一个内存通道上插了两根 2Rank 的内存条，即每个 CS 上都有对应的内存 RANK，那么就会自动打开 2T 模式。
2) 自动配置内存 BANK 数为 8，同时把 bank_diff_*(0x211)设置为 0x0.
3) 对于版本号 3 以上的控制器，pm_bank(0x16b)参数作为 cs_resync 来使用。
4) 根据内存条的容量以及行列地址线个数自动配置 cs_diff_*（0x213）、row_diff_*（0x212）以及 col_diff_*（0x210）等参数。
5) 根据内存条在内存插槽上的位置来配置 cs_map（0x1f4）、cs_enable（0x168）、cs_mrs（0x169）、cs_zq(0x16a)以及 lvl_cs(0x183)。
6) 根据内存条类型（UDIMM 或者 RDIMM）配置 addr_mirror(0x16e)。对于双面的 UDIMM，需要设置第二面的 mirror 位为 1.
7) 根据内存条的 RANK 类型（单 RANK 还是双 RANK）以及内存条在内存插槽上的位置来配置 odt_wr_cs_map(0x176)和 odt_rd_cs_map(0x172).
8) 根据 DISABLE_DIMM_ECC 来配置 ecc_enable(0x252)。
9) 配置 init_start（0x18）为 0x1，开始内存初始化。
10) 软件读取 dram_init（0x163）的值，该寄存器值与 cs_enable(0x168)相同时代表所有 CS 的初始化序列完成，可以进入 leveling 过程。
11) 内存训练，包括 write leveling 以及 gate leveling。
12) 训练完成后进入正常工作模式，内存可以正常访问。

4.4 内存读写测试：

4.4.1 PMON 下简单读写测试

这个测试主要对内存物理地址 0x0~0x38 进行简单的读写测试，如果读写测试正常，PMON 打印信息如下：

```
 The uncache data is:
00000000:  5555555555555555
00000008:  aaaaaaaaaaaaaaaa
00000010:  3333333333333333
00000018:  cccccccccccccccc
00000020:  7777777777777777
00000028:  8888888888888888
00000030:  1111111111111111
00000038:  eeeeeeeeeeeeeeee
The cached  data is:
00000000:  5555555555555555
00000008:  aaaaaaaaaaaaaaaa
00000010:  3333333333333333
00000018:  cccccccccccccccc
00000020:  7777777777777777
00000028:  8888888888888888
00000030:  1111111111111111
00000038:  eeeeeeeeeeeeeeee
```

如果测试不正常，可能出现以下现象（仅以 uncache 读写为例）

```
 The uncache data is:
00000000:  3333333333333333
00000008:  cccccccccccccccc
```

00000010: 7777777777777777

00000018: 8888888888888888

00000020: 1111111111111111

00000028: eeeeeeeeeeeeeeee

00000030: 1111111111111111

00000038: eeeeeeeeeeeeeeee

其中地址 0x30 和 0x38 的数据无效，可能为任意数据。这种情况可能是因为读提前了一个周期或者写提前了一个周期导致。若怀疑是读提前了一个周期，需要将参数 tPHY_RDDATA（0x1c0）减一；而当怀疑写提前一个周期时，需要将 tPHY_WRLAT（0x1d4）加一来解决。

反之，当出现以下现象时，需要将参数 tPHY_RDDATA（0x1c0）加一或者将 tPHY_WRLAT（0x1d4）减一来解决。

The uncache data is:

00000000: ffefdfefafef5ac3

00000008: ffefdfefafef5ac3

00000010: 5555555555555555

00000018: aaaaaaaaaaaaaaaa

00000020: 3333333333333333

00000028: cccccccccccccccc

00000030: 7777777777777777

00000038: 8888888888888888

其中地址 0x0 和 0x8 的数据无效，可能为任意数据。而且可能在打印完 0x28 的数据之后就出现卡死的情况，关于卡死的解决方法请参考 9.2 小节。

4.4.2 PMON 下复杂读写测试

当以上测试通过之后可以在 PMON 中做进一步的较复杂的测试。该测试过程需要在 PMON 的 start.S 中将 DEBUG_DDR 宏定义打开，那么在 PMON 启动过程中会出现以下提示：

Do test?(0xf: skip): 16'h

输入 0xf 则跳过该测试，直接回车则出现

default s1 = 0x00100001__00000000

Change test param s1(0: skip)?: 16'h

这里需要输入测试的内存起始地址和大小，该 64 位数据的[43:0]代表内存起始地址,[55:48]代表内存大小(以 128M 为单位)，[61:60]代表测试所用处理器核，[63:62]代表节点号。一般来说可以通过设置 NODE 中使用的最大内存容量来设置内存大小以确保 s1 寄存器以及 cs_diff_0，row_diff_0，ba_diff_0 和 row_diff_0 等值设置正确。

如果在测试过程中发现问题则会出现如下提示信息：

addr 0x98000000801aa000 expected: 98000000801aa000 read: 9800000100fa4000 reread: 9800000100fa4000

addr 0x98000000801aa200 expected: 98000000801aa200 read: 9800000100fa4200 reread: 9800000100fa4200

addr 0x98000000801aa400 expected: 98000000801aa400 read: 9800000100fa4400 reread: 9800000100fa4400

如果 read 和 reread 的值不同，那么读操作的返回数据肯定是存在问题，一般来说需要先解决该问题。而如果 read 和 reread 的值相同，但是与 expected 的值不同，那么有可能是读操作的问题也可能是写操作的问题。具体解决方法如下：

如果怀疑是写操作的问题那么需要通过示波器观测内存颗粒端出错 BYTE 对应的 DQS/DQ 信号，以确保 DQS/DQ 信号的摆幅、slewrate、反射以及二者相位关系、写眼图、打开窗口是否正确。当信号完整性出现问题时，可能需要调整 CPU 的 PAD 驱动能力和内存颗粒端的阻抗值，这需要分别修改 0x140-0x158 和 0x1a0-0x1b8 地址范围内的寄存器，参数具体意义请参考芯片用户手册和 DDR2/3 协议。当发现 DQS/DQ 的相位关系有问题时，即 DQS 的上升下降沿没有在 DQ 的窗口中间，需要修改 loongson3C_ddr3_leveling.S 文件的 adjust wrdata 代码段。当发现 DQS/DQ 的打开窗口有问题时，需要修改 dq_oe_*和 dqs_oe_*等寄存器。

如果怀疑是读操作的问题，也需要在 CPU 端测量 DQS/DQ 的信号。在排除信号质量方面的问题外还有一些参数是需要手动调整的，一般来说需要手动调整的参数为：

a) pad_odt_se(0x149)用于调整CPU端阻抗

b) 出错BYTE所对应的dll_rddqs_p/n_*值，默认为0x20，可以尝试增大或者减小该值来反复测试读写

c) 还有一组参数可能引起内存训练出错从而导致内存不稳定或者出错

的问题，该组参数为rd_oe_end_*，rd_oe_begin_*，rd_stop_edge_*，rd_start_edge_*。当在训练前后该组寄存器发生变化时需要更改其初始值重新训练，比如训练前的初始值为0x03030202，训练后可能变为0x03ff0202，那么可以尝试把该组寄存器修改为0x03030101来看是否训练后其值仍为0x03030101。可以尝试各种组合，比如0x03030000,0x02020303,0x02020202,0x03030303，看哪一组可以通过内存测试。但是需要保证rd_oe_end_*与rd_oe_begin_*相等，同时rd_stop_edge_*和rd_start_edge_*相等。

5 内存训练

内存训练(Leveling)操作是在 DDR3 中，用于智能配置内存控制器读写操作中各种信号间相位关系的操作。通常它包括了 Write Leveling、Read Leveling 和 Gate Leveling。在龙芯内存控制器中，只实现了 Write Leveling 与 Gate Leveling，Read Leveling 没有实现，软件需要通过判断读写的正确性来实现 Read Leveling 所完成的功能。除了在 Leveling 过程中操作的 DQS 相位、GATE 相位之外，还可以根据这些最后确认的相位来计算出写 DQ 相位、读 DQ 相位的配置方法。

因为 DDR2 协议并未定义内存训练流程，而且 DDR2 频率较低，所以对于 DDR2 的内存并没有一个标准的训练方法，9.11 小节给出了龙芯内存控制器针对 DDR2 内存的配置流程。下面主要介绍 DDR3 相关的内容。

5.1 龙芯内存训练程序：

PMON 中训练文件为 loongson3C_ddr3_leveling.S。

5.1.1 write leveling

DDR3 协议定义了 write leveling 过程。如图 5-1 所示。其主要目的是为了对齐到达内存颗粒的 CLK 和 DQS 信号。

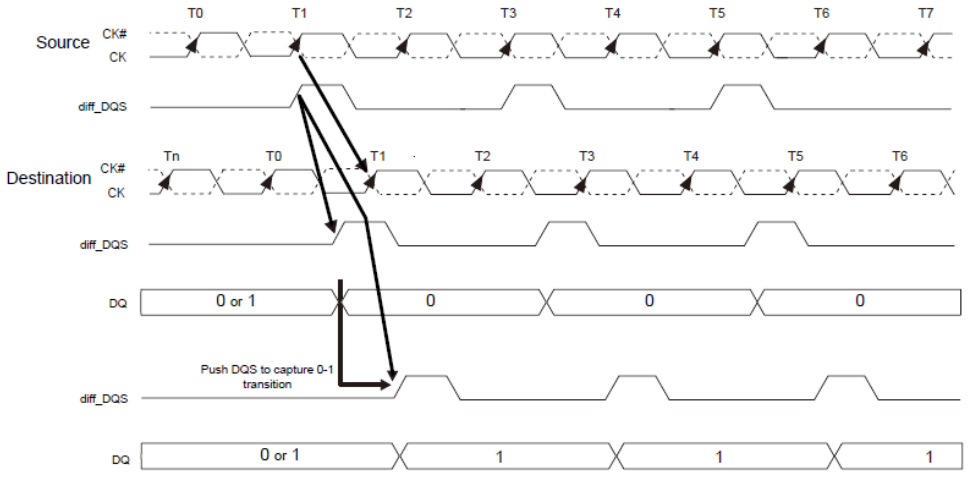


图 5-1 write leveling 示意图

Write leveling 的软件流程图如图 5-2 所示：

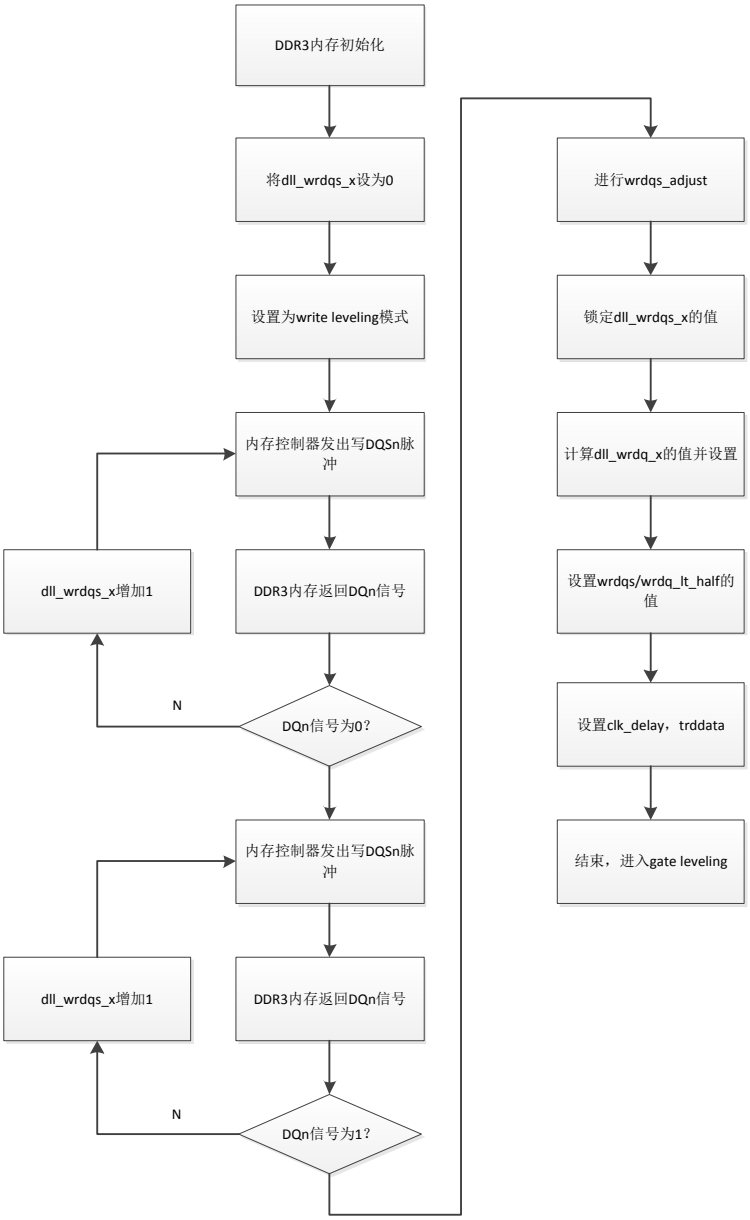


图 5-2 Write leveing 流程图

具体软件编程步骤如下：

1. 完成控制器初始化；
2. 将 hardware_pd3/2/1/0(0x1f8)设置为 4’b0，cs_resync、cs_zq(0x168)设置为 4’b0，ref_sch_en(0x340)设置为 1’b0；
3. 将 Dll_wrdqs_x（x = 0…8）设置为 0；
4. 设置 Lvl_mode（0x180）为 2’b01；
5. 采样 Lvl_ready（0x185）寄存器，如果为 1，表示可以开始 Write Leveling 请求；

6. 开始训练一个 Byte，设置 Lvl_req（0x181）为 1；
7. 采样 Lvl_done（0x186）寄存器，如果为 1，表示一次 Write Leveling 请求完成；
8. 采样对应 Byte 的 Lvl_resp_x（0x187-0x18f）寄存器，如果为 0 则跳至步骤 9。否则（采样 Lvl_resp_x 结果为 1），将对应的 Dll_wrdqs_x[6:0]增加 1，并重复执行步骤 6-8 直至采样 Lvl_resp_x 结果为 0；
9. 采样对应 Byte 的 Lvl_resp_x（0x187-0x18f）寄存器，如果为 0 则将对应的 Dll_wrdqs_x[6:0]增加 1，并重复执行步骤 6，7，9 直至采样 Lvl_resp_x 结果为 1；
10. 至此可能已经找到 CLK 的边沿，为了保证步骤 9 的采样为可靠结果，需要重复步骤 6，7，9 数次（该次数由 WR_FILTER_LENGTH 宏定义确定）。
11. 将 Dll_wrdqs_x[6:0]减去 WR_FILTER_LENGTH
12. 重复执行步骤 6-11，训练下一个 Byte。
13. 此时所有 Dll_wrdqs_x 的值设置正确；

至此 Write Leveling 操作结束。

14. 对每一个处在特定的区间的 Dll_wrdqs_x[6:0]进行微调。调整方式为：

    Dll_wrdqs_x[6:0]在[0x00,0x05)范围内时设置为 0x05；

    Dll_wrdqs_x[6:0]在[0x20,0x25)范围内时设置为 0x25；

    Dll_wrdqs_x[6:0]在[0x40,0x45)范围内时设置为 0x45；

    Dll_wrdqs_x[6:0]在[0x60,0x65)范围内时设置为 0x65；

    Dll_wrdqs_x[6:0]在(0x1a,0x1F]范围内时设置为 0x1a；

    Dll_wrdqs_x[6:0]在(0x3a,0x3F]范围内时设置为 0x3a；

    Dll_wrdqs_x[6:0]在(0x5a,0x5F]范围内时设置为 0x5a；

    Dll_wrdqs_x[6:0]在(0x7a,0x7F]范围内时设置为 0x7a；

    由宏定义 WRDQS_ADJUST_LOOP 决定是否执行该步骤。

15. 设置 Wrdqs_lt_halt_x，如果 Dll_wrdqs_x 的值小于 WRDQS_LTHF_STD，则将对应的 Wrdqs_lt_half_x 置为 1，否则置为 0；
16. 设置 Dll_wrdq_x，用 Dll_wrdqs_x[6:0]减去宏定义 DLL_WRDQ_SUB
17. 设置 Wrdq_lt_half_x，如果 Dll_wrdq_x 小于 WRDQ_LTHF_STD，则将对

应的 Wrdq_lt_half_x 置为 1，否则置为 0

18. 根据颗粒顺序(由 ORDER_OF_XDIMM 宏定义确定)逐个检测 Wrdq_lt_half_x 的值。如果所有的 Wrdq_lt_half_x 都为 1，则将 tPHY_WRLAT(0x1d4)和 tPHY_RDDATA(0x1c0)减 1；如果检测过程中出现了 Wrdq_lt_half_x 从 1 变为 0 的情况，则从第一个变为 0 的颗粒开始，所有后面颗粒对应的 Wrdq_clkdelay_x 设置为 1，然后将 tPHY_WRLAT(0x1d4) 和 tPHY_RDDATA(0x1c0) 减 1；如果所有 Wrdq_lt_half_x 都为 0，不做任何处理。对于 RDIMM 类型的内存条，需要对寄存器两端的内存颗粒分别进行处理。

19. 将 Lvl_mode（0x180）设置为 2’b00，退出 Write Leveling 模式；

在 write leveling 做完后，将 cs_zq(0x169)设置为与 cs_enable(0x168)一致，并且重新进行初始化操作：先将 init_start(0x018)置为 1’b0，再置为 1’b1，接着等待 dram_init(0x160)即完成了重新初始化的操作。

5.1.2 Gate leveling

gate Leveling 用于配置控制器内使能读 DQS 窗口的时机。软件流程图如图 5-3 所示：

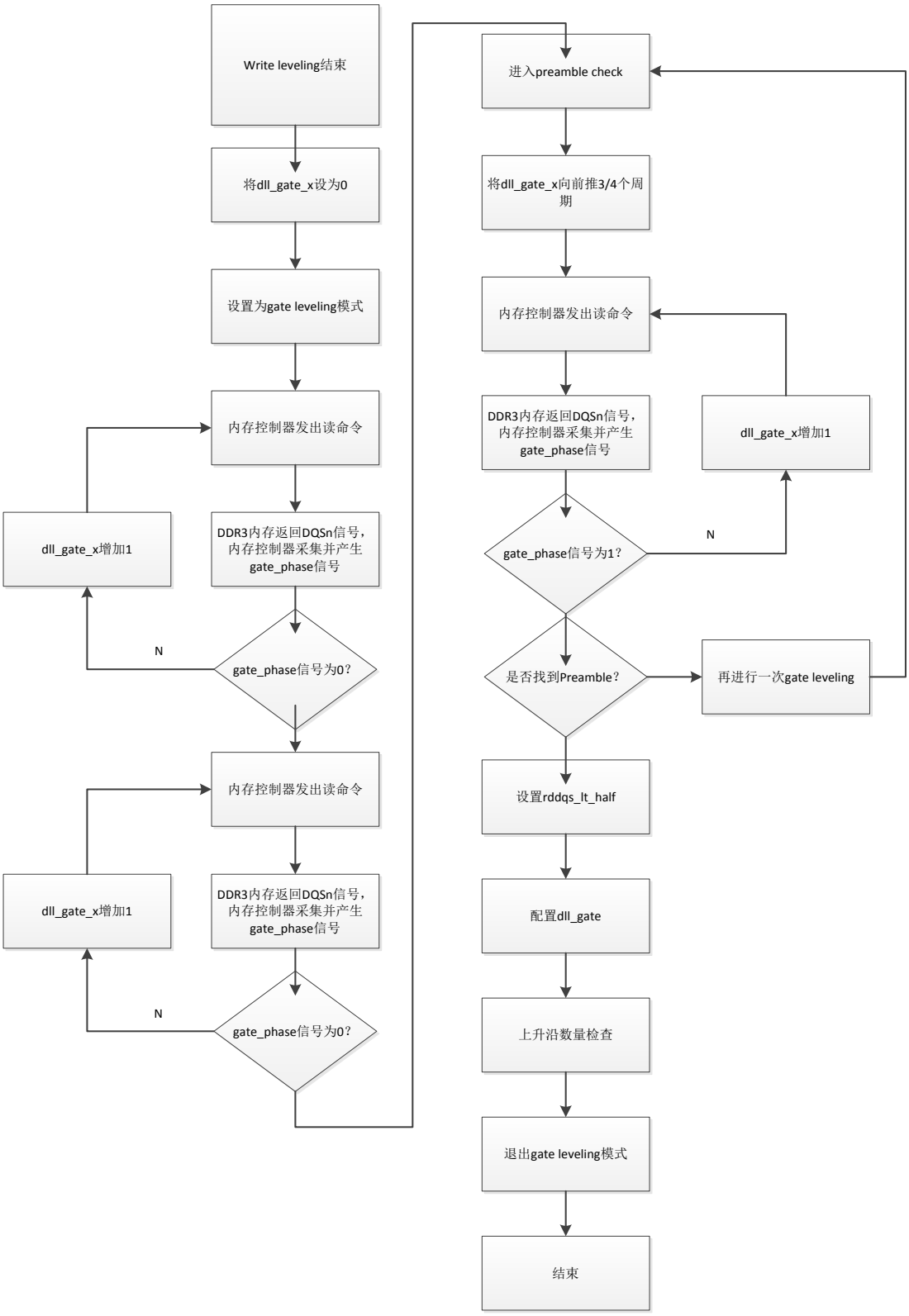


图 5-3 gate leveing 流程图

具体软件编程步骤如下：

1. 完成控制器初始化；

2. 完成 Write Leveling，参见上一小节内容；
3. 将 Cs_zq(0x168)设置为 4’b0；
4. 将 Dll_gate_x（x = 0…8）设置为 0；
5. 设置 Lvl_mode（0x180）为 2’b10；
6. 采样 Lvl_ready（0x180）寄存器，如果为 1，表示可以开始 Gate Leveling 请求；
7. 开始训练一个 Byte，设置 Lvl_req（0x180）为 1；
8. 采样 Lvl_done（0x180）寄存器，如果为 1，表示一次 Gate Leveling 请求完成；
9. 采样对应 Byte 的 Lvl_resp_x[0]（0x187-0x18f）寄存器，如果为 0 则跳至步骤 10。否则（采样 Lvl_resp_x[0]结果为 1），将对应的 Dll_gate_x[6:0]增加 1，并重复执行步骤 7-9 直至采样结果为 0；
10. 采样对应 Byte 的 Lvl_resp_x[0]（0x187-0x18f）寄存器，如果为 0 则将对应的 Dll_gate_x[6:0]增加 1，并重复执行步骤 7，8，10 直至采样结果为 1；
11. 至此可能已经找到读 DQS 的边沿，为了保证步骤 10 的采样为可靠结果，需要重复步骤 7，8，10 数次（该次数由 GATE_FILTER_LENGTH 宏定义确定）；
12. 将 Dll_gate_x[6:0]减去 GATE_FILTER_LENGTH；
13. 重复执行步骤 7-12，训练下一个 Byte；
14. 此时所有 Dll_gate_x 的值设置正确；
15. Preamble check。该步骤用于找到读 DQS 的起始位置，配置 Rd_oe_begin/end_x, tRDDATA(0x1c0)寄存器的值。具体过程说明如下：
    i. 把当前颗粒的 Dll_gate 减去 PREAMBLE_LENGTH_3AX
    ii. 流程与 gate leveling 相似，但是不找返回值为 0 的位置，直接找到返回值为 1 的位置
    iii. 若此时 Dll_gate 没有增加 PREAMBLE_LENGTH_3AX-5 次，则说明没有找到 Preamble，将该颗粒的 Rd_oe_begin/end 减 1，即向前推一个周期
    iv. 重新进行一次 gate_leveling，直到找到上升沿
    v. 重复步骤 i-iv 再次进行 Preamble Check。若 Dll_gate 增加了 PREAMBLE_LENGTH_3AX-5 次，则说明找到了 Preamble，继

续做下一个 Byte 的 Preamble Check。全做完后进行下一步

此时 Dll_gate_x[6:0]与 Dll_wrdata_x[6:0]的和实际上就是读 DQS 相对于 PHY 内部时钟的相位关系。下面根据 Leveling 的结果对各个参数进行调整。

16. 如果 Dll_gate_x[6:0]与 Dll_wrdata_x[6:0]的和大于 RDDQS_LTHF_STD1 或小于 RDDQS_LTHF_STD2 则设置 Rddqs_lt_half_x 为 1,否则设置为 0
17. 把每个颗粒的 Dll_gate_x 减 DLL_GATE_SUB
18. 调整完毕后，再分别进行两次 Lvl_req 操作，观察 Lvl_resp_x[7:5]与 Lvl_resp_x[4:2]的值变化，如果各增加为 Burst_length/2，则继续进行第 19 步操作；如果不为 4，可能需要对 Rd_oe_begin_x 进行加一或减一操作，如果大于 Burst_length/2，很可能需要对 Dll_gate_x 的值进行一些微调
19. 将 Lvl_mode（0x180）设置为 2'b00，退出 Gate Leveling 模式；
20. 完成了以上的 write leveling 和 gate leveling 后，将 hardware_pd3/2/1/0、cs_resync、cs_zq、ref_sch_en(0x340)这 7 组配置参数重新配置，以保证内存控制器的正常配置操作。

注： 在更改 dll_gate 值的时候，如果发生越界（增加时超过 0x80，减小时低于 0x00）则需要在结果的基础上与 0x7f 相与，同时更改 rd_oe_begin/end 的值（增加时溢出加 1，减小时减 1）。

注 2：在更改 rd_oe_begin/end 的值时，如果超过 0x3，则应把所有颗粒的 rd_oe_begin/end 值减小 1，同时将 tRDDATA 增加 1，由宏 RDOE_SUB_TRDDATA_ADD 控制。如果 rd_oe_begin/end 的值减小到 0x0，则应把所有颗粒的 rd_oe_begin/end 值增加 1，同时将 tRDDATA 减小 1，由宏 RDOE_ADD_TRDDATA_SUB 控制。如果发生两个颗粒的 rd_oe_begin/end 值相差超过 3，则会发生错误。

5.1.3 训练文件中宏定义说明

- ORDER_OF_UDIMM：内存条为 UDIMM 时，内存颗粒的排布顺序
- ORDER_OF_RDIMM：内存条为 RDIMM 时，内存颗粒的排布顺序
- WRDQS_LTHF_STD：wrdqs_lt_half 设置的参考基准值，dll_wrdqs 高于此值时 lt_half 设为 1，低于此值时设为 0
- WRDQ_LTHF_STD：wrdq_lt_half 设置的参考基准值，dll_wrdq 高于此值时

lt_half 设为 1，低于此值时设为 0

- RDDQS_LTHF_STD1/2：rddqs_lt_half 设置的参考基准值，算出的 rddqs 偏移高于 STD1 或低于 STD2 时设置为 1，否则设置为 0
- DLL_WRDQ_SUB：dll_wrdq 相对于 dll_wrdqs 减小的值
- DLL_GATE_SUB：dll_gate 相对于 rddqs 减小的值
- WR_FILTER_LENGTH：在 write leveling 找到 0 到 1 的跳变时，多校验的次数
- GATE_FILTER_LENGTH：在 gate leveling 找到 0 到 1 的跳变时，多校验的次数
- PREAMBLE_LENGTH_3AX：进行 preamble check 时，检验的 preamble 长度
- OFFSET_*：对应寄存器相对于(0x20,0x40...0x120)的地址偏移
- GET_NUMBER_OF_SLICES：获取颗粒数量（是否有 ECC）保存至 t0
- PRINT_THE_MC_PARAM：打印所有内存参数
- RDOE_SUB_TRDDATA_ADD：rd_oe 参数增加超过阈值（3）时进行所有颗粒 rd_oe 减少并增加 trddata 的操作
- RDOE_ADD_TRDDATA_SUB：rd_oe 参数减少超过阈值（0）时进行所有颗粒 rd_oe 增加并减少 trddata 的操作

6 内存测试常用程序：

6.1 PMON 测试程序：

PMON 命令行下可以使用 mt 或者 newmt 进行内存测试。

6.2 系统下 stressapptest

PMON 下的内存测试均通过后表示内存在较小访存压力下基本正常，接下来需要启动操作系统运行 stressapptest 进一步加大压力测试。若想在最大压力下测试内存，可以多运行几个 stressapptest 程序并将所有可用内存分配。如果测试出错，则会出现类似 PMON 下复杂读写测试的错误信息：

addr 0x98000000801aa000 expected: 98000000801aa000 read: 9800000100fa4000 reread: 9800000100fa4000

addr 0x98000000801aa200 expected: 98000000801aa200 read: 9800000100fa4200 reread: 9800000100fa4200

addr 0x98000000801aa400 expected: 98000000801aa400 read: 9800000100fa4400 reread: 9800000100fa4400

但是这里需要注意的是 read 和 reread 一般来说都一样，因为 reread 的地址会在 cache 内命中由 cache 直接返回数据，所以两次读到的数据一致并不能排除内存读操作仍然存在问题。如果内存出错，通过对比 expected 数据和 read 的数据可以确定出错的 BYTE，然后可以进一步调整相应的内存参数。

7 内存信号测量

7.1 测量点选择：

为了保证测量结果的可靠性，测量点需要选择在信号的终端。一般测量时选择单面内存条，在所测颗粒的背面的过孔选择所测信号。

时钟/地址/命令/控制信号都是单向传输的，测量时在内存条背面选测量点；

写 DQS-DQ 由内存控制器发给内存条，测量点也选在内存条背面过孔；

读 DQS-DQ 由内存条发给内存控制器，测量点需要在 CPU 端选择对应信号。

7.2 时钟信号 CLK

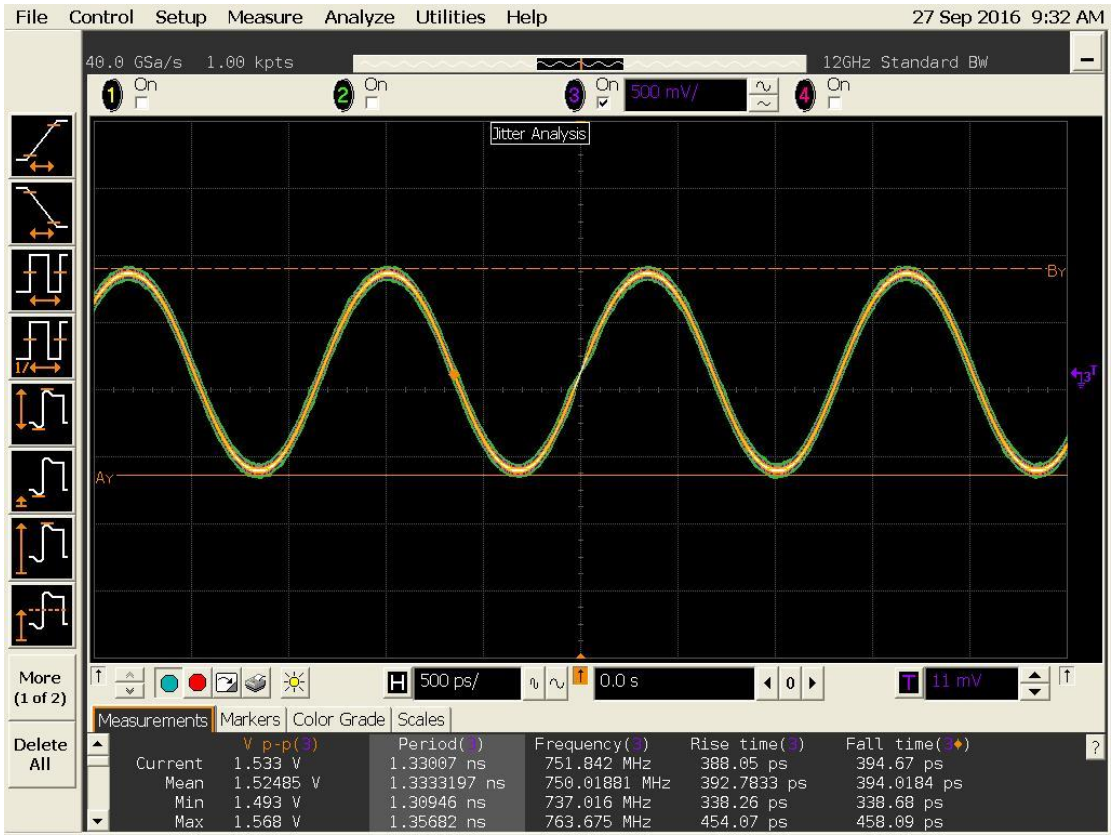


图 7-1 时钟信号测试图

图 7-1 为时钟信号的测试结果。对于时钟信号一般需要关注以下几点：

- 时钟信号是否存在反射；
- 时钟频率为多少；
- 时钟信号峰峰值(Vp-p)为多少；

- 时钟信号 Jitter
- 时钟信号占空比

对于图 7-1 的测试结果可以看出，时钟信号质量较高，不存在反射，时钟频率为 750MHz，峰峰值 1.53V，Jitter 47ps，占空比 48%/52%。

7.3 命令信号

一般情况下，命令/地址/控制信号质量类似，所以在测量时选择出现次数较高，更容易抓到的信号，比如 CS、CAS 信号。

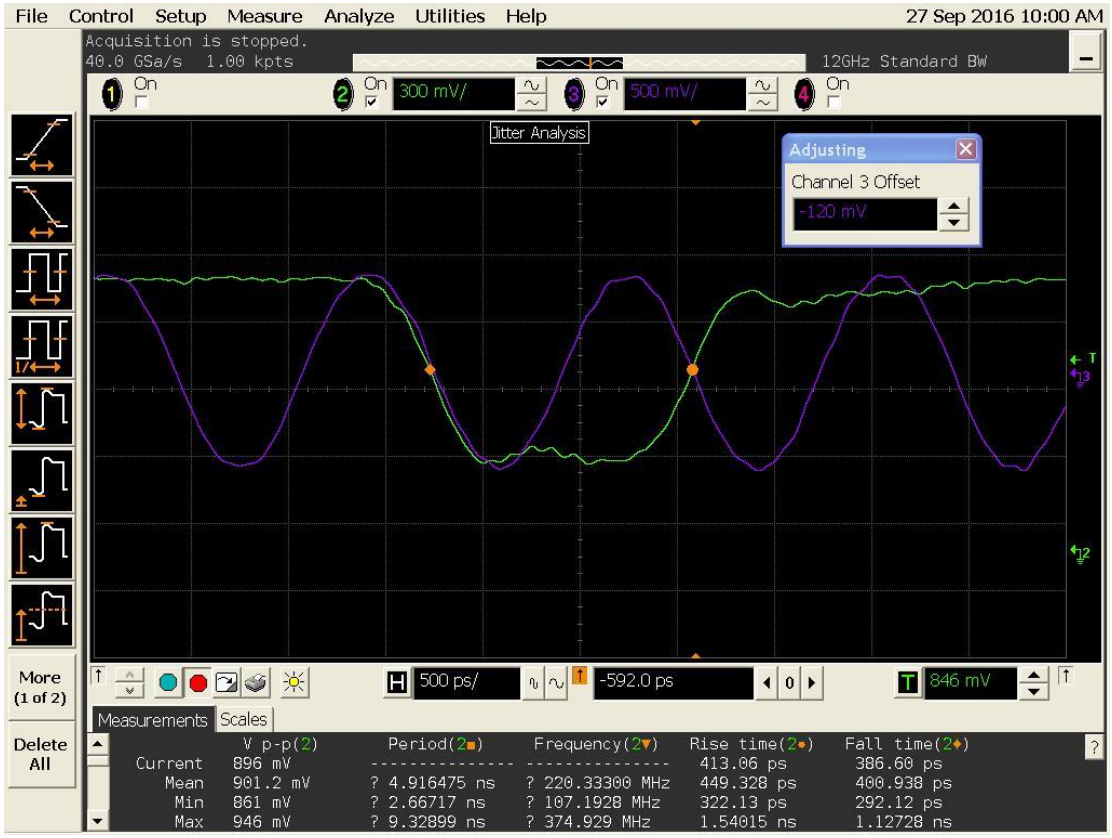


图 7-2 命令信号 CS 测试图

图 7-2 中绿色信号为 CS，紫色信号为时钟 CLK，长时间波形累计结果如下：

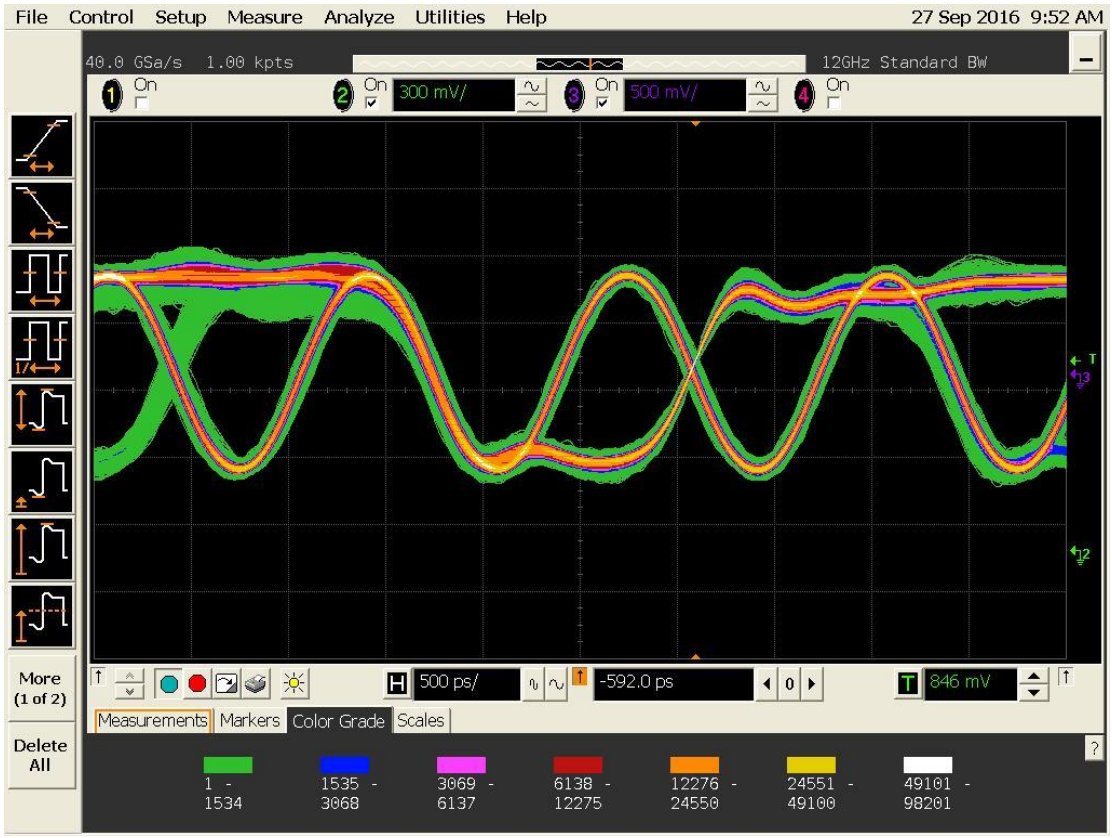


图 7-3 命令信号 CS 累积图

命令信号重要参数一般有：

- 命令信号是否存在反射；
- 命令信号最大最小值，摆幅；
- 命令信号与时钟信号的相位关系；

从图 7-2 和图 7-3 可以看出，电压最小值 288mV，最大值 1.357V，摆幅 1.069V，反射较小，而且时钟信号处于命令信号正中间，采样窗口较好。

7.4 写信号

写信号包含写 DQS 信号和写 DQ 信号，图 7-4 中上面的信号为写 DQS，下面信号为写 DQ。

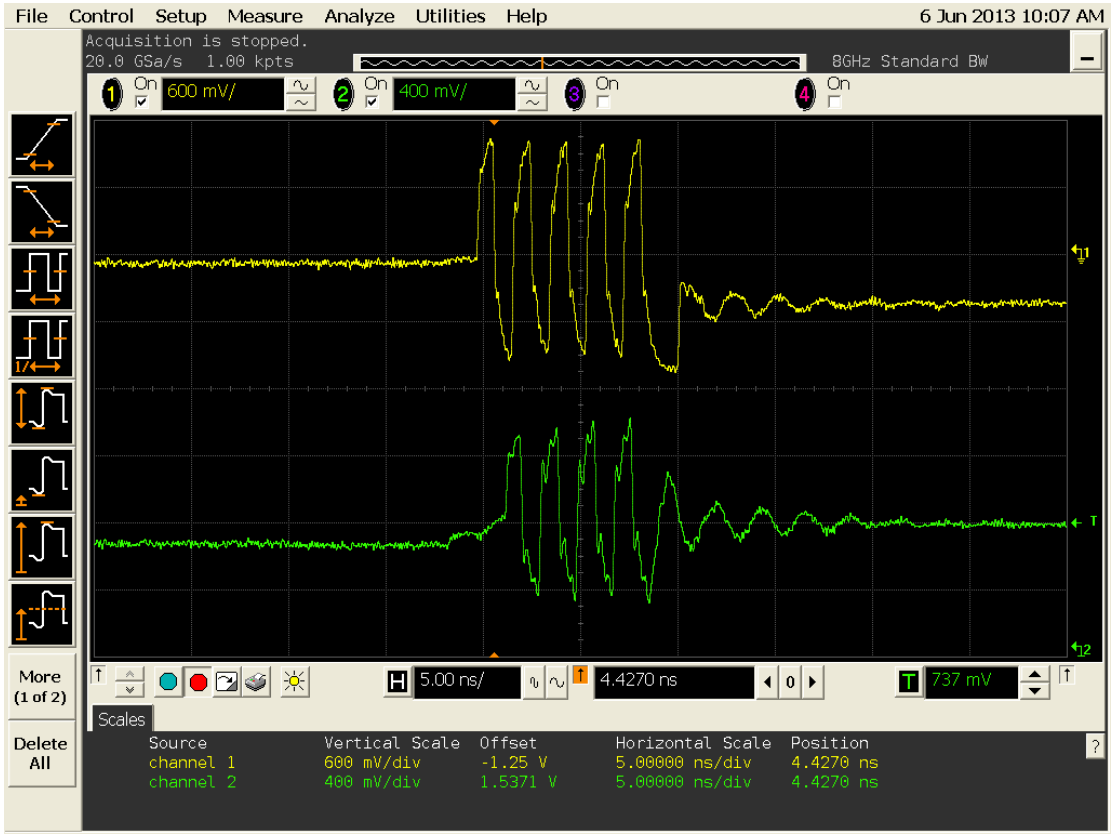


图 7-4 写信号

写信号需要关注以下几点：

■ 写信号是否存在反射；

图 7-4 的测量结果中写 DQS 和写 DQ 都存在一定的反射信号；

■ 写信号最大最小值；

需要注意写 DQS 为差分信号，所以其中间电平为 0；写 DQ 为单端信号，所以其中间电平为 750mV。

图 7-4 的测量结果中，写 DQS 最大值为 1V，最小值为-1V；写 DQ 最大值为 1.1V，最小值 250mV。

■ 写信号窗口是否设置正确；

写 DQS 信号为 5 个周期的时钟，且 preamble 为 3/4 个周期。写 DQS 窗口的开关均与 DQS 的上升沿对齐。

写 DQ 信号有效窗口为 4 个周期，其窗口打开时间比 DQS 的第一个上升沿晚 3/4 周期，窗口关闭时间比 DQS 的关闭时间早 1/4 周期。

所以图 7-4 的窗口设置正确。

需注意写 DQS 和写 DQ 结束后的震荡信号理想情况下应该不存在，但是实际情况下基本都会有，主要是由阻抗不匹配导致。

■ 写 DQS-DQ 相位关系是否正确

根据协议规定，写 DQS 信号上升下降沿需要处于 DQ 信号的中间，如图 7-4 所示，其信号相位关系正常。

7.5 写眼图

写眼图主要通过示波器根据 DQS 和 DQ 的相位关系自动抓取出写 DQS-DQ 对，通过累积 DQ 波形形成的。下图为实测的写眼图。图中的灰色梯形为模板，当 DQ 累积的波形与梯形重叠时代表时序违例。

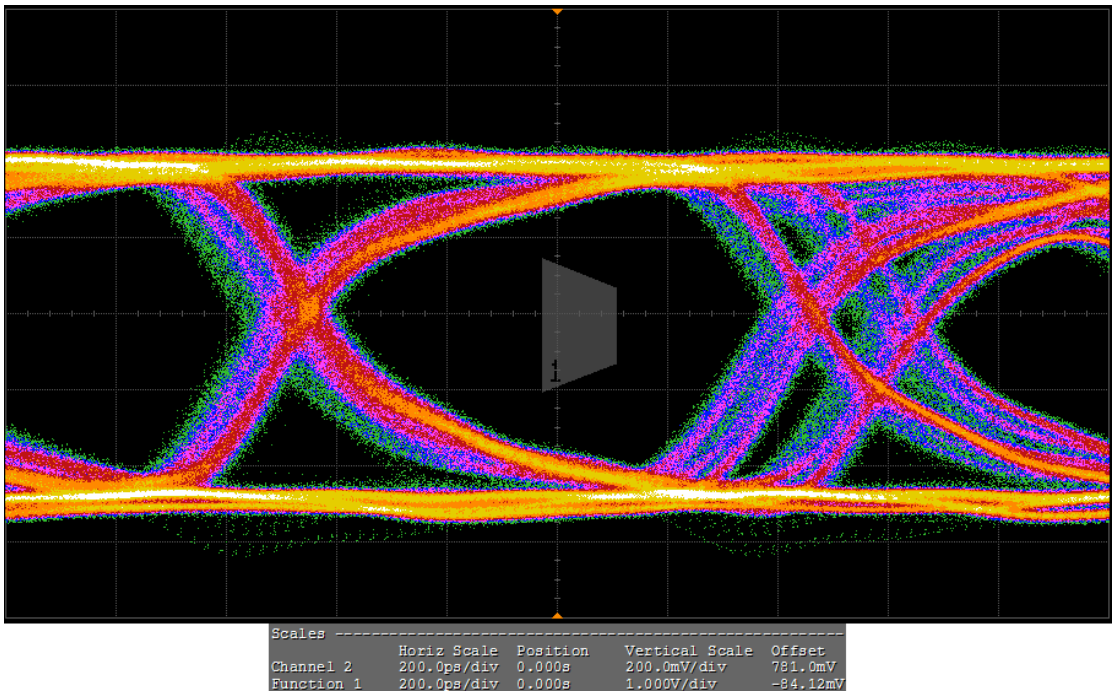


图 7-5 写眼图

7.6 读信号

图 7-6 读信号

读信号由内存条返回给内存控制器，主要包含读 DQS 信号和读 DQ 信号，上图中上面的信号为读 DQS，下面信号为读 DQ。读信号需要关注以下几点：

- 读信号是否存在反射；

  上图的测量结果中读 DQS 和读 DQ 反射较小；

- 读信号最大最小值；

  上图的测量结果中，读 DQS 最大值为 1V，最小值为-1V；读 DQ 最大值为 1.0V，最小值 250mV。

- 读 DQS-DQ 相位关系是否正确

  根据协议规定，读 DQS 信号上升下降沿需要与 DQ 信号边沿对齐，如上图所示，其信号相位关系正常。

7.7 读眼图

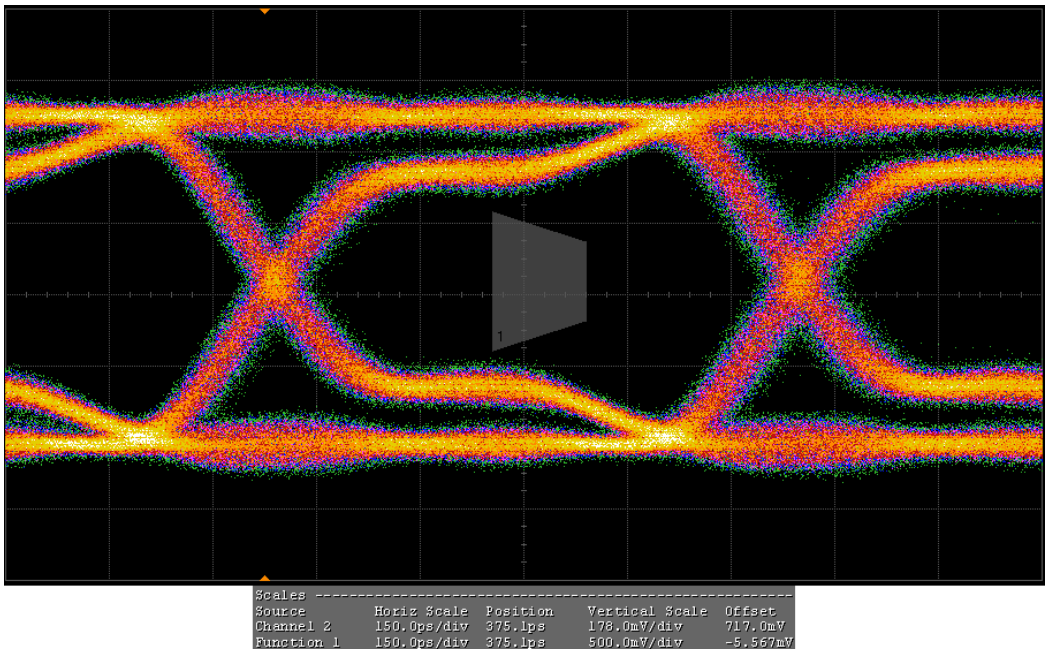


图 7-7 读眼图

与写眼图类似，读眼图也是通过示波器根据 DQS 和 DQ 的相位关系自动抓取出读 DQS-DQ 对，通过累积 DQ 波形形成的。图 7-7 为实测的读眼图。图中的灰色梯形为模板，当 DQ 累积的波形与梯形重叠时代表时序违例。

8 硬件设计注意事项

9 内存常见问题及解决方法总结：

9.1 内存调试最重要的参数--dll_ck_0-dll_ck_3(0x1c-0x1f)

因为内存训练的大部分参数的都与该参数有关系，所以当内存出现问题时首先需要调整的就是该组参数，一般调整范围为 0x10~0x70。该参数调整没有太严格的限制，只要是能保证训练之后的内存参数满足 9.5 节规律即可。

9.2 卡死情况

当在内存训练时出现卡死现象，卡死前的 PMON 串口打印信息为：

Start Hard Leveling...

这种情况一般是由内存金手指与插槽接触不良导致，需要把内存金手指擦一下然后重新插上即可。

如果训练完成后发生卡死现象，那么一定是读操作出了问题，可以通过以下方法解决和调试：

a) 调整dll_ck0-3(0x1c-0x1f)的值

b) 改变tPHY_RDDATA的值，增加或者减小1~2

c) 设置2T/3T模式，设置方法见3.21节

d) 设置rdfifo_valid(0x162)为0x0，这时内存肯定不会因为读操作卡死。参数tPHY_RDLAT控制读数据返回时间。

e) 如果尝试以上方法还无法解决卡死的现象，那么应该考虑硬件是否存在问题，比如电压、电流等，这需要硬件工程师进一步评估决定。

9.3 DLL_BYPASS 模式

内部 DLL 都有 BYPASS 控制。

对于 DLL master 来说，dll_bypass（0x19）控制 master 的锁定 bypass 功能，当 dll_bypass 设置为 1 时，即使 DLL master 没有锁定，内存控制器初始化仍可以继续进行。

对于其他 DLL（包括时钟 DLL，wrdqs DLL，wrdq DLL，dll_rddqs_p/n，dll_gate），每个 DLL 都有对应的 dll_*控制参数，该参数位宽为 8 位：

[7:7]：bypass 使能

[6:0]：当 bypass = 0 时，表示 n/128 个时钟周期。相对时间，与时钟周期相关。所以在时钟周期变化之后，不需要重新设置。

当 bypass = 1 时，表示 n 个延迟单元。绝对时间，与时钟周期无关。所以在时钟周期变化之后，为了保持相位关系，需要重新计算该值。

对于 3B1500，2J1500 和 3A1500 芯片，有可能需要定义 DLL_BYPASS，该参数在 PMON 的 start.S 中定义。

9.4 DLL 锁定问题

该问题只存在于 3B5 芯片，当 DLL 锁定之后的值 dll_value_ck(0x4)与所设置的初始值 dll_start_point(0x1a)的值相同时则认为 DLL 锁定有问题，这时需要重新设置 dll_start_point 为稍小于 dll_value_ck 的值。

9.5 训练后内存参数分析

下图为 2K1000 芯片内存频率为 500MHz 情况下 UDIMM(左)和 RDIMM(右)的训练结果。

```
00000020:  0202000001000001
00000028:  0202000002010101
00000030:  0000000003020202
00000038:  000000202004f2f65
00000040:  0201000201000001
00000048:  0202000002010100
00000050:  0000000004030202
00000058:  0000002020583860
00000060:  0201000201010000
00000068:  0202000002010100
00000070:  0000000103020202
00000078:  000000202070505d
00000080:  0201000201010000
00000088:  0202000002010100
00000090:  0000000104030202
00000098:  000000202076565d
000000a0:  0201000201010100
000000a8:  0202000002010100
000000b0:  0000000104030202
000000b8:  0000002020006060
000000c0:  0201000201010100
000000c8:  0202000002010100
000000d0:  0000000104030202
000000d8:  000000202011714d
000000e0:  0201000201010100
000000e8:  0202000002010100
000000f0:  0000000103020202
000000f8:  000000202018785e
00000100:  0201000201010101
00000108:  0202000002010100
00000110:  0000000104030202
00000118:  0000002020280844
00000120:  0201000201000000
00000128:  0303000002010100
00000130:  0000000003020202
00000138:  000000202007f6000
```

```
00000020:  0201000201010000
00000028:  0202000002010100
00000030:  0000000103020202
00000038:  0000002020684868
00000040:  0201000201010000
00000048:  0202000002010100
00000050:  0000000103020202
00000058:  0000002020684860
00000060:  0201000201000001
00000068:  0202000002010100
00000070:  0000000003020202
00000078:  0000002020583860
00000080:  0201000201000001
00000088:  0202000002010100
00000090:  0000000003020202
00000098:  000000202004f2f5e
000000a0:  0201000201000101
000000a8:  0202000002010100
000000b0:  0000000003020202
000000b8:  0000002020381860
000000c0:  0201000201000001
000000c8:  0202000002010100
000000d0:  0000000003020202
000000d8:  0000002020583844
000000e0:  0201000201000001
000000e8:  0202000002010100
000000f0:  0000000003020202
000000f8:  0000002020583860
00000100:  0201000201010000
00000108:  0202000002010100
00000110:  0000000103020202
00000118:  000000202006d4d53
00000120:  0201000201000000
00000128:  0303000002010100
00000130:  0000000003020202
00000138:  000000202007f6000
```

图 9-1 UDIMM(左)和 RDIMM(右)的训练结果

通过这个结果可以看出：

- dll_wrdqs/dll_wrdata 的值变化趋势：

对于 UDIMM，dll_wrdqs_0 至 dll_wrdqs_8(如果未使能 ECC 则为 dll_wrdqs_7)的值依次增大(当在非 DLL_BYPASS 模式下，该参数仅有 7 位有效位，在达到 0x7f 后回转到 0x0)。因为 dll_wrdata_*与对应的 dll_wrdqs_*的差值固定，所以 dll_wrdata_0 至 dll_wrdata_8 的值也依次增大。这体现了 DDR3 的 FLY-BY 的走线结构

对于 RDIMM，参数分为两组，其中一组 dll_wrdqs_8、dll_wrdqs_3、dll_wrdqs_2、dll_wrdqs_1、dll_wrdqs_0 的值依次增加，另外一组 dll_wrdqs_4、dll_wrdqs_5、dll_wrdqs_6、dll_wrdqs_7 的值依次增加。dll_wrdata_*同理。

- dll_gate 的值：

对于不同颗粒，其反映了 DQS、DQ 走线长度，其值越大则对应的 Byte 的 DQS/DQ 线越长。一般来说不同 BYTE 之间的 DQS/DQ 线等长，所以 dll_gate_* 的值应该大致相等，差别不会太大。

9.6 稳定性-性能矛盾

有时内存不稳定的问题可以通过牺牲内存性能的方式来解决。当内存的压力变小后，一些因为供电或者信号质量导致的问题可能就变的不明显。如果通过这种方式使内存变的稳定，那么基本上可以排除训练程序所带来的问题。当稳定性问题解决后，可以再逐步把部分参数还原来确定引起问题的参数，与此同时可以提升一些内存性能。有关性能相关的参数主要在 0x1a0-0x1e8 地址范围内。其中 0x1a0-0x1d8 地址范围内的参数需要根据颗粒的手册来确认，而 0x1e0-0x1e8 地址范围内的参数为内存控制器定义，数值越大，内存性能越低。

9.7 合理电压范围

硬件工程师需确认内存相关电压和 CPU 相关电压在合理范围内，且电压纹波较小。其中比较重要的几个内存的电压指标为：

对于 DDR3 内存： Vref -- 0.75V Vtt -- 0.75V VDD -- 1.5V

对于 DDR2 内存： Vref -- 0.9V Vtt -- 0.9V VDD -- 1.8V

9.8 32/16 位通道模式

当怀疑地址、命令线或者高位数据线有问题时可以通过配置内存控制器为 32/16 位通道模式来确定自己的推测。对于共用地址、命令线的颗粒来说（比如 UDIMM/RDIMM）,如果 32/16 位通道模式下内存工作稳定，则可以排除地址、命令线的问题。这两种模式的配置方式请参考芯片用户手册。

9.9 使用 ECC 功能排除内存问题

系统中出现的任何问题都有可能是内存导致。比如长时间测试时 stressapptest 报错，固定出现在某一个 bit，调整参数以及更换内存条等措施都没有解决问题，那问题也有可能不在内存上。芯片满负荷工作时的电压波动等因素也有可能导致内部数字电路逻辑出错。

对于支持内存 ECC 功能的芯片来说（目前只有 2K1000 芯片不支持），就可以使用带 ECC 颗粒的内存条来排除内存的问题。首先需要使能 ECC 功能（ecc_enable 0x252），然后在 PMON 或者系统报错时检查 ecc 状态相关的寄存器（比如

int_vector 0x251，ecc_code 0x253），如果这些寄存器为 0，那么表示从未发生过内存错误，基本就可以排除内存的问题。

为了公平对比起见，最好使用带 ECC 颗粒的内存条在关闭 ECC 功能时复现之前出现的问题。

9.10 双路开发板调试建议

建议在内存条较多的情况下分步测试，比如对于 3B1500 的双路板最多可以插 8 根内存条。而系统下的测试出错并不能确定是哪根内存条出了错，这种情况下就可以采取分步测试的方法，先保证 NODE0 的内存稳定的前提下再去测试 NODE1 的内存。

当然在 PMON 下也可以参考 4.4.2 节中介绍的方法，通过设置地址[63:62]选择节点号来测试不同节点的内存，这种方法对问题的初步定位有帮助。

9.11 DDR2 内存参数配置方法

1. 首先设置 dll_ck，如果时钟与命令/地址线在板上走线长度一致，那么设置 dll_ck 为 0x40 可以保证时钟的上升沿在命令/地址的窗口中间。

2. 如果时钟与 DQS/DQ 板上走线长度一致，那么设置 dll_dqs_*与 dll_ck 为相同的值，dll_wrdata_*比对应的 dll_dqs_*小 0x20。

3. 根据 dll_dqs_*和 dll_wrdata_*设置对应的 wrdqs_lt_half_*和 dll_wrdata_*，设置方法参考芯片用户手册。

4. 根据 2.1 节的测试结果和方法调整 tPHY_RDDATA 和 tPHY_WRLAT

5. 一般来说简单读写测试通过就可以确定基本参数，可以根据进一步的内存压力测试微调 dll 的相关参数。