顶层结构设计示意图

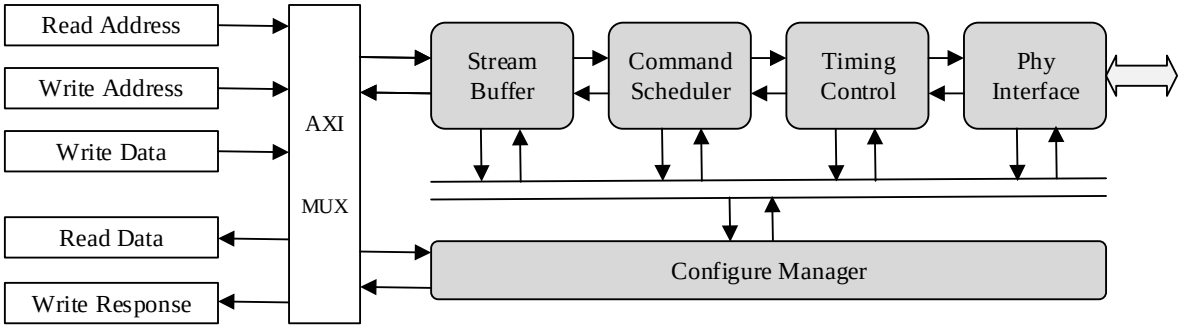


图 1 DDR Controller

1.参数配置

1.1. 配置控制器

参数列表及说明如下表：

（注：黑色字体为 0x1 版本，蓝色字体为 0x2 版本新增或与之前版本不一样，红色字体为 0x3 版本新增或与之前版本不一样）

	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
0x000	Dll_value_0(RD)/Dll_adj_cnt/dll_sync_disable/dll_close_disable		Dll_value_ck(RD)		Dll_init_done(RD)		Version(RD)	
0x008	Dll_value_4(RD)		Dll_value_3(RD)		Dll_value_2(RD)		Dll_value_1(RD)	
0x010	Dll_value_8(RD)		Dll_value_7(RD)		Dll_value_6(RD)		Dll_value_5(RD)	
0x018	Dll_ck_3	Dll_ck_2	Dll_ck_1	Dll_ck_0	Dll_increment	Dll_start_point	Dll_bypass	Init_start
0x020	Dq_oe_end_0	Dq_oe_begin_0	Dq_stop_edge_0	Dq_start_edge_0	Rddata_delay_0	Rddqs_lt_half_0	Wrdqs_lt_half_0	Wrdq_lt_half_0
0x028	Rd_oe_end_0	Rd_oe_begin_0	Rd_stop_edge_0	Rd_start_edge_0	Dqs_oe_end_0	Dqs_oe_begin_0	Dqs_stop_edge_0	Dqs_start_edge_0
0x030	Enzi_end_0	Enzi_begin_0	Wrclk_sel_0	Wrdq_clkdelay_0	Odt_oe_end_0	Odt_oe_begin_0	Odt_stop_edge_0	Odt_start_edge_0
0x038	Enzi_stop_0	Enzi_start_0	Dll_oe_shorten_0	Dll_rddqs_n_0	Dll_rddqs_p_0	Dll_wrdqs_0	Dll_wrdata_0	Dll_gate_0
0x040	Dq_oe_end_1	Dq_oe_begin_1	Dq_stop_edge_1	Dq_start_edge_1	Rddata_delay_1	Rddqs_lt_half_1	Wrdqs_lt_half_1	Wrdq_lt_half_1
0x048	Rd_oe_end_1	Rd_oe_begin_1	Rd_stop_edge_1	Rd_start_edge_1	Dqs_oe_end_1	Dqs_oe_begin_1	Dqs_stop_edge_1	Dqs_start_edge_1
0x050	Enzi_end_1	Enzi_begin_1	Wrclk_sel_1	Wrdq_clkdelay_1	Odt_oe_end_1	Odt_oe_begin_1	Odt_stop_edge_1	Odt_start_edge_1
0x058	Enzi_stop_1	Enzi_start_1	Dll_oe_shorten_1	Dll_rddqs_n_1	Dll_rddqs_p_1	Dll_wrdqs_1	Dll_wrdata_1	Dll_gate_1
0x060	Dq_oe_end_2	Dq_oe_begin_2	Dq_stop_edge_2	Dq_start_edge_2	Rddata_delay_2	Rddqs_lt_half_2	Wrdqs_lt_half_2	Wrdq_lt_half_2
0x068	Rd_oe_end_2	Rd_oe_begin_2	Rd_stop_edge_2	Rd_start_edge_2	Dqs_oe_end_2	Dqs_oe_begin_2	Dqs_stop_edge_2	Dqs_start_edge_2
0x070	Enzi_end_2	Enzi_begin_2	Wrclk_sel_2	Wrdq_clkdelay_2	Odt_oe_end_2	Odt_oe_begin_2	Odt_stop_edge_2	Odt_start_edge_2
0x078	Enzi_stop_2	Enzi_start_2	Dll_oe_shorten_2	Dll_rddqs_n_2	Dll_rddqs_p_2	Dll_wrdqs_2	Dll_wrdata_2	Dll_gate_2
0x080	Dq_oe_end_3	Dq_oe_begin_3	Dq_stop_edge_3	Dq_start_edge_3	Rddata_delay_3	Rddqs_lt_half_3	Wrdqs_lt_half_3	Wrdq_lt_half_3
0x088	Rd_oe_end_3	Rd_oe_begin_3	Rd_stop_edge_3	Rd_start_edge_3	Dqs_oe_end_3	Dqs_oe_begin_3	Dqs_stop_edge_3	Dqs_start_edge_3
0x090	Enzi_end_3	Enzi_begin_3	Wrclk_sel_3	Wrdq_clkdelay_3	Odt_oe_end_3	Odt_oe_begin_3	Odt_stop_edge_3	Odt_start_edge_3
0x098	Enzi_stop_3	Enzi_start_3	Dll_oe_shorten_3	Dll_rddqs_n_3	Dll_rddqs_p_3	Dll_wrdqs_3	Dll_wrdata_3	Dll_gate_3
0x0A0	Dq_oe_end_4	Dq_oe_begin_4	Dq_stop_edge_4	Dq_start_edge_4	Rddata_delay_4	Rddqs_lt_half_4	Wrdqs_lt_half_4	Wrdq_lt_half_4
0x0A8	Rd_oe_end_4	Rd_oe_begin_4	Rd_stop_edge_4	Rd_start_edge_4	Dqs_oe_end_4	Dqs_oe_begin_4	Dqs_stop_edge_4	Dqs_start_edge_4
0x0B0	Enzi_end_4	Enzi_begin_4	Wrclk_sel_4	Wrdq_clkdelay_4	Odt_oe_end_4	Odt_oe_begin_4	Odt_stop_edge_4	Odt_start_edge_4
0x0B8	Enzi_stop_4	Enzi_start_4	Dll_oe_shorten_4	Dll_rddqs_n_4	Dll_rddqs_p_4	Dll_wrdqs_4	Dll_wrdata_4	Dll_gate_4

0x0C0	Dq_oe_end_5	Dq_oe_begin_5	Dq_stop_edge_5	Dq_start_edge_5	Rddata_delay_5	Rddqs_lt_half_5	Wrdqs_lt_half_5	Wrdq_lt_half_5
0x0C8	Rd_oe_end_5	Rd_oe_begin_5	Rd_stop_edge_5	Rd_start_edge_5	Dqs_oe_end_5	Dqs_oe_begin_5	Dqs_stop_edge_5	Dqs_start_edge_5
0x0D0	Enzi_end_5	Enzi_begin_5	Wrclk_sel_5	Wrdq_clkdelay_5	Odt_oe_end_5	Odt_oe_begin_5	Odt_stop_edge_5	Odt_start_edge_5
0x0D8	Enzi_stop_5	Enzi_start_5	Dll_oe_shorten_5	Dll_rddqs_n_5	Dll_rddqs_p_5	Dll_wrdqs_5	Dll_wrdata_5	Dll_gate_5
0x0E0	Dq_oe_end_6	Dq_oe_begin_6	Dq_stop_edge_6	Dq_start_edge_6	Rddata_delay_6	Rddqs_lt_half_6	Wrdqs_lt_half_6	Wrdq_lt_half_6
0x0E8	Rd_oe_end_6	Rd_oe_begin_6	Rd_stop_edge_6	Rd_start_edge_6	Dqs_oe_end_6	Dqs_oe_begin_6	Dqs_stop_edge_6	Dqs_start_edge_6
0x0F0	Enzi_end_6	Enzi_begin_6	Wrclk_sel_6	Wrdq_clkdelay_6	Odt_oe_end_6	Odt_oe_begin_6	Odt_stop_edge_6	Odt_start_edge_6
0x0F8	Enzi_stop_6	Enzi_start_6	Dll_oe_shorten_6	Dll_rddqs_n_6	Dll_rddqs_p_6	Dll_wrdqs_6	Dll_wrdata_6	Dll_gate_6
0x100	Dq_oe_end_7	Dq_oe_begin_7	Dq_stop_edge_7	Dq_start_edge_7	Rddata_delay_7	Rddqs_lt_half_7	Wrdqs_lt_half_7	Wrdq_lt_half_7
0x108	Rd_oe_end_7	Rd_oe_begin_7	Rd_stop_edge_7	Rd_start_edge_7	Dqs_oe_end_7	Dqs_oe_begin_7	Dqs_stop_edge_7	Dqs_start_edge_7
0x110	Enzi_end_7	Enzi_begin_7	Wrclk_sel_7	Wrdq_clkdelay_7	Odt_oe_end_7	Odt_oe_begin_7	Odt_stop_edge_7	Odt_start_edge_7
0x118	Enzi_stop_7	Enzi_start_7	Dll_oe_shorten_7	Dll_rddqs_n_7	Dll_rddqs_p_7	Dll_wrdqs_7	Dll_wrdata_7	Dll_gate_7
0x120	Dq_oe_end_8	Dq_oe_begin_8	Dq_stop_edge_8	Dq_start_edge_8	Rddata_delay_8	Rddqs_lt_half_8	Wrdqs_lt_half_8	Wrdq_lt_half_8
0x128	Rd_oe_end_8	Rd_oe_begin_8	Rd_stop_edge_8	Rd_start_edge_8	Dqs_oe_end_8	Dqs_oe_begin_8	Dqs_stop_edge_8	Dqs_start_edge_8
0x130	Enzi_end_8	Enzi_begin_8	Wrclk_sel_8	Wrdq_clkdelay_8	Odt_oe_end_8	Odt_oe_begin_8	Odt_stop_edge_8	Odt_start_edge_8
0x138	Enzi_stop_8	Enzi_start_8	Dll_oe_shorten_8	Dll_rddqs_n_8	Dll_rddqs_p_8	Dll_wrdqs_8	Dll_wrdata_8	Dll_gate_8
0x140	Pad_ocd_clk	Pad_ocd_ctl	Pad_ocd_dqs	Pad_ocd_dq	Pad_enzi		Pad_en_ctl	Pad_en_clk
0x148	Pad_adj_code_dqs	Pad_code_dqs	Pad_adj_code_dq	Pad_code_dq		Pad_vref_internal	Pad_odt_se	Pad_modezi1v8
0x150		Pad_reset_po	Pad_adj_code_clk	Pad_code_lk	Pad_adj_code_cmd	Pad_code_cmd	Pad_adj_code_addr	Pad_code_addr
0x158		Pad_comp_code_o	Pad_comp_okn		Pad_comp_code_i	Pad_comp_mode	Pad_comp_tm	Pad_comp_pd
0x160	Rdfifo_empty(RD)		Overflow(RD)		Dram_init(RD)	Rdfifo_valid	Cmd_timming	Ddr3_mode
0x168	Ba_xor_row_offset	Addr_mirror	Cmd_delay	Burst_length	Bank/Cs_resync	Cs_zq	Cs_mrs	Cs_enable
0x170	Odt_wr_cs_map		Odt_wr_length	Odt_wr_delay	Odt_rd_cs_map		Odt_rd_length	Odt_rd_delay
0x178								
0x180	Lvl_resp_0(RD)	Lvl_done(RD)	Lvl_ready(RD)		Lvl_cs	tLVL_DELAY	Lvl_req(WR)	Lvl_mode
0x188	Lvl_resp_8(RD)	Lvl_resp_7(RD)	Lvl_resp_6(RD)	Lvl_resp_5(RD)	Lvl_resp_4(RD)	Lvl_resp_3(RD)	Lvl_resp_2(RD)	Lvl_resp_1(RD)
0x190	Cmd_a		Cmd_ba	Cmd_cmd	Cmd_cs	Status_cmd(RD)	Cmd_req(WR)	Command_mode
0x198			Status_sref(RD)	Srefresh_req	Pre_all_done(RD)	Pre_all_req(RD)	Mrs_done(RD)	Mrs_req(WR)
0x1A0	Mr_3_cs_0		Mr_2_cs_0		Mr_1_cs_0		Mr_0_cs_0	
0x1A8	Mr_3_cs_1		Mr_2_cs_1		Mr_1_cs_1		Mr_0_cs_1	
0x1B0	Mr_3_cs_2		Mr_2_cs_2		Mr_1_cs_2		Mr_0_cs_2	
0x1B8	Mr_3_cs_3		Mr_2_cs_3		Mr_1_cs_3		Mr_0_cs_3	
0x1C0	tRESET	tCKE	tXPR	tMOD	tZQCL	tZQ_CMD	tWLDQSEN	tRDDATA
0x1C8	tFAW	tRRD	tRCD	tRP	tREF	tRFC	tZQCS	tZQperiod
0x1D0	tODTL	tXSRD	tPHY_RDLAT	tPHY_WRLAT	tRAS_max			tRAS_min
0x1D8	tXPDLL	tXP	tWR	tRTP	tRL	tWL	tCCD	tWTR
0x1E0	tW2R_diffCS	tW2W_diffCS	tR2P_sameBA	tW2P_sameBA	tR2R_sameBA	tR2W_sameBA	tW2R_sameBA	tW2W_sameBA
0x1E8	tR2R_diffCS	tR2W_diffCS	tR2P_sameCS	tW2P_sameCS	tR2R_sameCS	tR2W_sameCS	tW2R_sameCS	tW2W_sameCS
0x1F0	Power_up	Age_step	tCPDED	Cs_map	Bs_config	Nc	Pr_r2w	Placement_en
0x1F8	Hw_pd_3	Hw_pd_2	Hw_pd_1	Hw_pd_0	Credit_16	Credit_32	Credit_64	Selection_en
0x200	Cmdq_age_16		Cmdq_age_32		Cmdq_age_64		tCKESR	tRDPDEN
0x208	Wfifo_age		Rfifo_age		Power_stat3	Power_stat2	Power_stat1	Power_stat0
0x210	Active_age		Cs_place_0	Addr_win_0	Cs_diff_0	Row_diff_0	Ba_diff_0	Col_diff_0

0x218	Fastpd_age		Cs_place_1	Addr_win_1	Cs_diff_1	Row_diff_1	Ba_diff_1	Col_diff_1
0x220	Slowpd_age		Cs_place_2	Addr_win_2	Cs_diff_2	Row_diff_2	Ba_diff_2	Col_diff_2
0x228	Selfref_age		Cs_place_3	Addr_win_3	Cs_diff_3	Row_diff_3	Ba_diff_3	Col_diff_3
0x230	Win_mask_0				Win_base_0			
0x238	Win_mask_1				Win_base_1			
0x240	Win_mask_2				Win_base_2			
0x248	Win_mask_3				Win_base_3			
0x250		Cmd_monitor	Axi_monitor		Ecc_code(RD)	Ecc_enable	Int_vector	Int_enable
0x258								
0x260	Ecc_addr(RD)							
0x268	Ecc_data(RD)							
0x270	Lpbk_ecc_mask(RD)	Prbs_init			Lpbk_error(RD)	Prbs_23	Lpbk_start	Lpbk_en
0x278	Lpbk_ecc(RD)		Lpbk_data_mask(RD)		Lpbk_correct(RD)		Lpbk_counter(RD)	
0x280	Lpbk_data_r(RD)							
0x288	Lpbk_data_f(RD)							
0x290	Axi0_bandwidth_w				Axi0_bandwidth_r			
0x298	Axi0_latency_w				Axi0_latency_r			
0x2A0	Axi1_bandwidth_w				Axi1_bandwidth_r			
0x2A8	Axi1_latency_w				Axi1_latency_r			
0x2B0	Axi2_bandwidth_w				Axi2_bandwidth_r			
0x2B8	Axi2_latency_w				Axi2_latency_r			
0x2C0	Axi3_bandwidth_w				Axi3_bandwidth_r			
0x2C8	Axi3_latency_w				Axi3_latency_r			
0x2D0	Axi4_bandwidth_w				Axi4_bandwidth_r			
0x2D8	Axi4_latency_w				Axi4_latency_r			
0x2E0	Cmdq0_bandwidth_w				Cmdq0_bandwidth_r			
0x2E8	Cmdq0_latency_w				Cmdq0_latency_r			
0x2F0	Cmdq1_bandwidth_w				Cmdq1_bandwidth_r			
0x2F8	Cmdq1_latency_w				Cmdq1_latency_r			
0x300	Cmdq2_bandwidth_w				Cmdq2_bandwidth_r			
0x308	Cmdq2_latency_w				Cmdq2_latency_r			
0x310	Cmdq3_bandwidth_w				Cmdq3_bandwidth_r			
0x318	Cmdq3_latency_w				Cmdq3_latency_r			
0x320	tRESYNC_length	tRESYNC_shift	tRESYNC_max	tRESYNC_min	Pre_predict		tXS	tREF_low
0x328								tRESYNC_delay
0x330	Stat_en	Rdbuffer_max	Retry	Wr_pkg_num	Rwq_rb	Stb_en	Addr_new	tRDQidle
0x338				Rd_fifo_depth	Retry_cnt			
0x340	tREFretention					Ref_num	tREF_IDLE	Ref_sch_en
0x348								
0x350	Lpbk_data_en							
0x358						Lpbk_ecc_mask_en	Lpbk_ecc_en	Lpbk_data_mask_en
0x360			Int_ecc_cnt_fatal	Int_ecc_cnt_error	Ecc_cnt_cs_3	Ecc_cnt_cs_2	Ecc_cnt_cs_1	Ecc_cnt_cs_0
0x368								

0x370	Prior_age3			Prior_age2			Prior_age1			Prior_age_0		
0x378												Row_hit_place
0x380	Zq_cnt_1						Zq_cnt_0					
0x388	Zq_cnt_3						Zq_cnt_2					

1.2. 详细说明列表

（注：该表中的默认值均为 DDR3_1600 匹配参数）

0x000	位域	读写	默认值	说明
Dll_close_disable	57:57	读写	0x0	在版本 0x3 中： 不动态关闭 dll 功能
Dll_sync_disable	56:56	读写	0x0	在版本 0x3 中： 关闭 dll 的 sync 功能
Dll_value_0	56:48	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2 中： 为 Dll_adj_cnt 在版本 0x3 中： 为 Dll_adj_cnt 和 Dll_sync_disable
Dll_adj_cnt	55:48	读写	0x7	在版本 0x1 中： 为 Dll_value_0 在版本 0x2/0x3 中 调整时机控制信号 每过 dll_adj_cnt 周期后 dll 调整一个码
Dll_value_ck	40:32	只读	0x0	时钟组 DLL 锁定值
Dll_init_done	25:16	只读	0x0	控制器内部 DLL 锁定信号 [25:17]：恒为 0,无实际意义 [16:16]：DLL 锁定标志
Version	15:0	只读	0x3	控制器版本号
0x008				
Dll_value_4	56:48	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_3	40:32	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_2	24:16	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_1	8:0	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义

				在版本 0x2/0x3 中： 保留
0x010				
Dll_value_8	56:48	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_7	40:32	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_6	24:16	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
Dll_value_5	8:0	只读	0x0	在版本 0x1 中： 恒为 0,无实际意义 在版本 0x2/0x3 中： 保留
0x018				
Dll_ck_3	63:56	读写	0x12	时钟 3 延迟值 [63:63]：bypass 控制 [62:56]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_ck_2	55:48	读写	0x12	时钟 2 延迟值 [55:55]：bypass 控制 [54:48]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_ck_1	47:40	读写	0x12	时钟 1 延迟值 [47:47]：bypass 控制 [46:40]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_ck_0	39:32	读写	0x12	时钟 0 延迟值 [39:39]：bypass 控制 [38:32]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_increment	31:24	读写	0x4	每次 DLL 下溢时，起始延迟单元增加个数
Dll_start_point	23:16	读写	0x10	DLL 初始化的起始延迟单元个数
Dll_bypass	8:8	读写	0x0	DLL 初始化 bypass 控制。 该位只有当 Dll_init_done 一直无法锁定时需要设置，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。而且设置之前相应的 DLL 延迟的最高位（bypass 控制）也应该设置

Init_start	0:0	读写	0x0	控制器初始化开始。 只有当其它的所有相关参数设置好了之后才可以将该位置位，使控制器进行初始化，并向内存发起初始化。只有这个操作完成后内存空间才可以被访问，否则内存空间不可被外部访问。
0x020				
Dq_oe_end_0	58:56	读写	0x2	在版本 0x1 中： 第 0 组数据输出有效时期的结束时间，不可小于 Dq_oe_begin_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Dq_oe_begin_0	50:48	读写	0x2	在版本 0x1 中： 第 0 组数据输出有效时期的开始时间，不可大于 Dq_oe_end_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Dq_stop_edge_0	41:40	读写	0x0	第 0 组数据输出有效时期的结束相位，其与 Dq_oe_end_0 组合得到的时钟边沿不可早于 Dq_start_edge_0 与 Dq_oe_begin_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Dq_start_edge_0	33:32	读写	0x0	第 0 组数据输出有效时期的开始相位，其与 Dq_oe_begin_0 组合得到的时钟边沿不可晚于 Dq_stop_edge_0 与 Dq_oe_end_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Rddata_delay_0	24:24	读写	0x1	读返回数据在 FIFO 中延迟一周期输出
Rddqs_lt_half_0	16:16	读写	0x0	当读返回 DQS 信号（延时后）相比内部时钟的延迟小于半周期时需要设为 1
Wrdqs_lt_half_0	8:8	读写	0x0	当 Dll_wrdqs_0 的设置小于 0x40 时需要设为 1
Wrdq_lt_half_0	0:0	读写	0x0	当 Dll_wrdata_0 的设置小于 0x40 时需要设为 1
0x028				
Rd_oe_end_0	58:56	读写	0x1	在版本 0x1 中： 第 0 组数据读采样有效时期的结束时间，不可小于 Rd_oe_begin_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Rd_oe_begin_0	50:48	读写	0x1	在版本 0x1 中： 第 0 组数据读采样有效时期的开始时间，不可大于 Rd_oe_end_0

				在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Rd_stop_edge_0	41:40	读写	0x0	第 0 组数据读采样有效时期的结束相位，其与 Rd_oe_end_0 组合得到的时钟边沿不可早于 Rd_start_edge_0 与 Rd_oe_begin_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Rd_start_edge_0	33:32	读写	0x0	第 0 组数据读采样有效时期的开始相位，其与 Rd_oe_begin_0 组合得到的时钟边沿不可晚于 Rd_stop_edge_0 与 Rd_oe_end_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Dqs_oe_end_0	26:24	读写	0x2	在版本 0x1 中： 第 0 组数据写 DQS 有效时期的结束时间，不可小于 Dqs_oe_begin_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Dqs_oe_begin_0	18:16	读写	0x1	在版本 0x1 中： 第 0 组数据写 DQS 有效时期的开始时间，不可大于 Dqs_oe_end_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Dqs_stop_edge_0	9:8	读写	0x1	第 0 组数据写 DQS 有效时期的结束相位，其与 Dqs_oe_end_0 组合得到的、时钟边沿不可早于 Dqs_start_edge_0 与 Dqs_oe_begin_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Dqs_start_edge_0	1:0	读写	0x1	第 0 组数据写 DQS 有效时期的开始相位，其与 Dqs_oe_begin_0 组合得到的时钟边沿不可晚于 Dqs_stop_edge_0 与 Dqs_oe_end_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
0x030				
Enzi_end_0	57:56	读写	0x1	在版本 0x1 中： 保留 在版本 0x2/0x3 中：

				第 0 组数据读 DQ/DQS 有效时期的结束时间，不可小于 Enzi_begin_0。当 pad_enzi 为 1 时该参数不起作用。
Enzi_begin_0	49:48	读写	0x0	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 第 0 组数据读 DQ/DQS 有效时期的开始时间，不可大于 Enzi_end_0。当 pad_enzi 为 1 时该参数不起作用。
Wrclk_sel_0	40；40	读写	0x1	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 第 0 组数据写 DQ 的驱动时钟选择 1-选择驱动时钟为 clk_wrdqs， 0-选择驱动时钟为 clk_wrdq
Wrdq_clk_delay_0	28:28	读写	0x0	第 0 组数据写 DQ 延迟控制信号 在 Wrdq_lt_half_0 = 0 的时候将本组数据延迟增加一拍
Odt_oe_end_0	26:24	读写	0x2	在版本 0x1 中： 第 0 组数据读 ODT（控制器内部）有效时期的结束时间，不可小于 Odt_oe_begin_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Odt_oe_begin_0	18:16	读写	0x1	在版本 0x1 中： 第 0 组数据读 ODT（控制器内部）有效时期的开始时间，不可大于 Odt_oe_end_0 在版本 0x2/0x3 中： 该信号位宽为 2，仅[1:0]有效，意义同上
Odt_stop_edge_0	9:8	读写	0x0	第 0 组数据读 ODT（控制器内部）有效时期的结束相位，其与 Odt_oe_end_0 组合得到的时钟边沿不可早于 Odt_start_edge_0 与 Odt_oe_begin_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Odt_start_edge_0	1:0	读写	0x0	第 0 组数据读 ODT（控制器内部）有效时期的开始相位，其与 Odt_oe_begin_0 组合得到的时钟边沿不可晚于 Odt_stop_edge_0 与 Odt_oe_end_0 组合得到的时钟边沿 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
0x038				
Enzi_stop_0	57:56	读写	0x1	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 第 0 组数据读 DQ/DQS 有效时期的结束相位，其与

				Enzi_end_0 组合得到的时钟边沿不可早于 Enzi_start_0 与 Enzi_begin_0 组合得到的时钟边沿。当 pad_enzi 为 1 时该参数不起作用。 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Enzi_start_0	49:48	读写	0x2	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 第 0 组数据读 DQ/DQS 有效时期的开始相位，其与 Enzi_end_0 组合得到的时钟边沿不可早于 Enzi_start_0 与 Enzi_begin_0 组合得到的时钟边沿。当 pad_enzi 为 1 时该参数不起作用。 0 – 比为 1 时提前 1/4 周期 1 – 对应于 wrdqs_0（第 0 组写 DQS）的上升沿 2 – 比为 1 时推后 1/4 周期 3 – 比为 1 时推后 1/2 周期
Dll_oe_shorten_0	46:40	读写	0x0	在版本 0x1/0x2 中： 保留 在版本 0x3 中： Bit0：dq_oe_shorten，允许缩短第 0 组数据写 dq 打开窗口 Bit1：dqs_oe_shorten，允许缩短第 0 组数据写 dqs 打开窗口 Bit2：rd_oe_shorten，允许缩短第 0 组数据读 gate 打开窗口 Bit3：odt_oe_shorten，允许缩短第 0 组数据 CPU 端 odt 打开窗口 Bit4：enzi_shorten，允许缩短第 0 组数据 pad 接收端打开窗口
Dll_rddqs_n_0	39:32	读写	0x20	读 DQSn 采样延迟值 [39:39]：bypass 控制 [38:32]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_rddqs_p_0	31:24	读写	0x20	读 DQSp 采样延迟值 [31:31]：bypass 控制 [30:24]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_wrdqs_0	23:16	读写	0x7F	写 DQS 延迟值 [23:23]：bypass 控制 [22:16]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_wrdata_0	15:8	读写	0x60	写数据延迟值（应该比 DQS 提前 1/4 周期）

				[15:15]：bypass 控制 [14:8]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
Dll_gate_0	7:0	读写	0x0	读 DQS 采样有效时期控制延迟值 [7:7]：bypass 控制 [6:0]：当 bypass = 0 时，表示 n/128 个时钟周期 当 bypass = 1 时，表示 n 个延迟单元
0x040				
0x138				
0x140				
Pad_ocd_clk	58:56	读写	0x0	时钟引脚输出阻抗控制 000 – 40 欧姆 001 – 30 欧姆 010 – 24 欧姆 011 – 20 欧姆 100 – 15 欧姆
Pad_ocd_ctl	50:48	读写	0x0	控制引脚输出阻抗控制 000 – 40 欧姆 001 – 30 欧姆 010 – 24 欧姆 011 – 20 欧姆 100 – 15 欧姆
Pad_ocd_dqs	40:40	读写	0x0	DQS 引脚输出阻抗控制 0 – 34 欧姆 1 – 40 欧姆
Pad_ocd_dq	32:32	读写	0x0	DQ 引脚输出阻抗控制 0 – 34 欧姆 1 – 40 欧姆
Pad_enzi	24:16	读写	0x0	分别对应 9 个数据组的引脚输入使能 1 – 使能 0 – 高阻
Pad_en_ctl	8:8	读写	0x1	控制引脚输出使能 1 – 使能 0 – 高阻
Pad_en_clk	7:0	读写	0x0	时钟引脚输出使能 1 – 使能 0 – 高阻
0x148				
Pad_adj_code_dqs	63:56	读写	0x0	设置当 Pad_code_dqs[0]有效时 DQS 信号附加 CODE [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能
Pad_code_dqs	50:48	读写	0x0	DQS 信号附加 CODE 使能设置

				Bit 2：在 Bit 0 有效时，附加在输出及 ODT 上 Bit 1：在 Bit 0 有效时，附加在 SLEWRATE 上 Bit 0：附加 CODE 使能，1 有效
Pad_adj_code_dq	47:40	读写	0x0	设置当 Pad_code_dq[0]有效时 DQ 信号附加 CODE [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能
Pad_code_dq	34:32	读写	0x0	DQ 信号附加 CODE 使能设置 Bit 2：在 Bit 0 有效时，附加在输出及 ODT 上 Bit 1：在 Bit 0 有效时，附加在 SLEWRATE 上 Bit 0：附加 CODE 使能，1 有效
Pad_vref_internal	16:16	读写	0x0	使能内部 VREF 分压电路 1 - 同时使用内部 VREF 分压与外部引脚输出电压 0 - 只使用外部引脚输出电压
Pad_odt_se	8:8	读写	0x0	引脚匹配电阻值控制 0 – 60 欧姆 1 – 120 欧姆
Pad_modezi1v8	24:24	读写	0x0	PAD MODE ZI 1v8 1 – 使用 PAD 的 ZITEST 输入 0 – 使用 PAD 的 ZI 输入
0x150				
Reset_ctrl	49:48	读写	0x0	内存控制器复位引脚输出状态控制 Bit 0：复位信号使能。 为 1 时，保持引脚输出为低； 为 0 时，由 Bit 1 控制由有效电平 Bit 1：复位信号有效电平。 为 0 时，低有效； 为 1 时，高有效
Pad_adj_code_clk	47:40	读写	0x0	设置当 Pad_code_clk[0]有效时 CLK 信号附加 CODE [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能
Pad_code_clk	34:32	读写	0x0	CLK 信号附加 CODE 使能设置 Bit 2：在 Bit 0 有效时，附加在输出及 ODT 上 Bit 1：在 Bit 0 有效时，附加在 SLEWRATE 上 Bit 0：附加 CODE 使能，1 有效
Pad_adj_code_cmd	31:24	读写	0x0	设置当 Pad_code_cmd[0]有效时 CMD 信号附加 CODE [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能
Pad_code_cmd	18:16	读写	0x0	CMD 信号附加 CODE 使能设置 Bit 2：在 Bit 0 有效时，附加在输出及 ODT 上 Bit 1：在 Bit 0 有效时，附加在 SLEWRATE 上 Bit 0：附加 CODE 使能，1 有效
Pad_adj_code_addr	15:8	读写	0x0	设置当 Pad_code_addr[0]有效时 ADDR 信号附加 CODE [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能

Pad_code_addr	2:0	读写	0x0	ADDR 信号附加 CODE 使能设置 Bit 2：在 Bit 0 有效时，附加在输出及 ODT 上 Bit 1：在 Bit 0 有效时，附加在 SLEWRATE 上 Bit 0：附加 CODE 使能，1 有效
0x158				
Pad_comp_code_o	57:48	只读	0x0	引脚补偿单元自动调节调整值 在版本 0x1 中： Pad_comp_code_o 位于 0x158[39:32]
Pad_comp_okn	40:40	只读	0x0	引脚补偿单元自动调节完成标志
Pad_comp_code_i	33:24	读写	0xF0	引脚补偿单元手动设置值 在版本 0x1 中： Pad_comp_code_i 位于 0x158[31:24] [7:4] N_CODE：1 使能，0 关闭 [3:0] P_CODE：0 关闭，1 使能 在版本 0x2/0x3 中： [9:5] N_CODE：1 使能，0 关闭 [4:0] P_CODE：0 关闭，1 使能
Pad_comp_mode	16:16	读写	0x0	引脚补偿单元设置 1 – 手动设置 CODE 0 – 自动调节 CODE
Pad_comp_tm	8:8	读写	0x0	外部引脚测试模块使能 1 – 使用引脚 COMP_NOUT/COMP_POUT 连接电阻 0 – 使用引脚 COMP_REXT 连接电阻
Pad_comp_pd	0:0	读写	0x1	引脚补偿单元 Power Down 1 – Power Down 0 – 正常工作
0x160				
Rdfifo_empty	56:48	只读	0x0	PHY 中收集每个 SLICE 的读 FIFO 错误读出标志，当对应的 FIFO 为空时发生出队列操作时有效。可以用于判断 Rdfifo_valid 无效时，tPHY_RDLAT 的值是否设置过小 每一位分别对应于 SLICE8 … SLICE0
Overflow	40:32	只读	0x0	PHY 中每个 SLICE 中的读 FIFO 溢出标志，每一位分别对应于 SLICE8 … SLICE0
Dram_init	27:24	只读	0x0	DRAM 初始化完成标志，在 Init_start 设置之后才会生效，每一位分别对应于一个片选
Rdfifo_valid	16:16	读写	0x1	表示使用 PHY 内部逻辑控制读数据同步时间 该位无效时，这个同步时间由 tPHY_RDLAT 决定
Cmd_timing	9:8	读写	0x0	控制线 2T/3T 功能使能 0 – 1T 1 – 2T 2 – 3T 与其他几个参数需要满足关系式： tRDDATA - Cmd_delay - Cmd_timing = CASLAT – 3 tPHY_WRLAT – Cmd_delay – Cmd_timing = WRLAT - 4

Ddr3_mode	1:0	读写	0x1	在版本 0x1 中： 使用 DDR2 模式时将该位设为 0 使用 DDR3 模式时将该位设为 1 在版本 0x2/0x3 中： 使用 DDR2 模式时将该位设为 0 使用 DDR3 STD 模式时将该位设为 1 使用 DDR3 ECB 模式时将该位设为 3
0x168				
Ba_xor_row_offset	60:56	读写	0x4	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 表示地址译码时 bank 需要与 row 中的某些位数进行异或 Bit[4]：表示异或使能 Bit[3:0]: 表示 bank 需要与 row 地址的高位进行异或，具体是指的 row 地址的高（16-该值）位
Addr_mirror	51:48	读写	0x0	表示该 CS 对应的地址需要进行地址镜像
Cmd_delay	41:40	读写	0x0	表示命令总线需要的附加延迟 有效值为 0/1/2 与其他几个参数需要满足关系式： tRDDATA - Cmd_delay - Cmd_timing = CASLAT –3 tPHY_WRLAT – Cmd_delay – Cmd_timing = WRLAT - 4
Burst_length	35:32	读写	0x7	表示 DRAM 总线上的突发请求长度，该参数设置应与 MRS 参数一致。 突发长度为 8 时设为 4’h7，突发长度为 4 时设为 4’h3
Bank/cs_resync	27:24	读写	0x7/0x1	在版本 0x1 中： 表示每个片选上的 Bank 数量 Bank 数为 2 时： 需设置 Ba_diff = 2 , Addr_win[3:2] = 2’b01 Bank 数为 4 时： 需设置 Ba_diff = 1 , Addr_win[3:2] = 2’b10 Bank 数为 8 时： 需设置 Ba_diff = 0 , Addr_win[3:2] = 2’b11 在版本 0x2 中： 保留 在版本 0x3 中： 使能对应片选信号的 resync 请求
Cs_zq	19:16	读写	0x1	使能对应片选信号的 ZQ 请求
Cs_mrs	11:8	读写	0x1	使能对应片选信号的 MRS 请求
Cs_enable	3:0	读写	0x1	使能对应片选信号
0x170				
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_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
0x170				
0x180				
Lvl_resp_0	63:56	只读	0x0	Leveling 操作时，第 0 数据组的反馈信号
Lvl_done	48:48	只读	0x0	Leveling 操作时，表示 Lvl_resp_*有效信号
Lvl_ready	40:40	只读	0x0	Leveling 操作时，表示当前控制器已经进入 Leveling 操作模式。（用户程序正确设置 Lvl_mode 的值后，应该对这个寄存器进行采样，如果值为 1 表示可以对控制器发起 Leveling 请求，也就是说，此时才可以将设 Lvl_req 为 1）
Lvl_cs	27:24	读写	0x1	Leveling 操作时，当前控制的片选信号
tLVL_DELAY	23:16	读写	0x10	Leveling 操作时，有效采样延迟周期 单位为时钟周期
Lvl_req	8:8	只写	0x0	Leveling 操作时，向外发起 Leveling 操作请求
Lvl_mode	1:0	读写	0x0	Leveling 模式使能 00 – 正常功能模式 01 – Write Leveling 模式 10 – Gate Leveling 模式
0x188				
Lvl_resp_8	63:56	只读	0x0	Leveling 操作时，第 8 数据组的反馈信号 当 Lvl_mode == 1 时，为数据线上的反馈 当 Lvl_mode == 2 时 Bit[7:5]：内部有效读 DQS 时钟上升沿计数器 Bit[4:2]：内部有效读 DQS 时钟下升沿计数器 Bit[1:0]：内部 gate 采样读 DQS 的反馈 对于一个正确配置的 Gate Leveling 操作来说 Bit[7:5] 与 Bit[4:2] 应该在每一次 Leveling_req 增加 Burst_length/2 个计数，否则需要调整 Dll_gate_x 的值
Lvl_resp_7	55:48	只读	0x0	Leveling 操作时，第 7 数据组的反馈信号
Lvl_resp_6	47:40	只读	0x0	Leveling 操作时，第 6 数据组的反馈信号
Lvl_resp_5	39:32	只读	0x0	Leveling 操作时，第 5 数据组的反馈信号

Lvl_resp_4	31:24	只读	0x0	Leveling 操作时，第 4 数据组的反馈信号
Lvl_resp_3	23:16	只读	0x0	Leveling 操作时，第 3 数据组的反馈信号
Lvl_resp_2	15:8	只读	0x0	Leveling 操作时，第 2 数据组的反馈信号
Lvl_resp_1	7:0	只读	0x0	Leveling 操作时，第 1 数据组的反馈信号
0x190				
Cmd_a	63:48	读写	0x0	命令发送模式下，对 DRAM 发出的地址线信号
Cmd_ba	42:40	读写	0x0	命令发送模式下，对 DRAM 发出的 ba 线信号
Cmd_cmd	34:32	读写	0x0	命令发送模式下，对 DRAM 发出的控制信号 bit2 – RASn bit1 – CASn bit0 – WEn
Cmd_cs	27:24	读写	0x0	命令发送模式下，对 DRAM 发出的片选信号
Status_cmd	16:16	只读	0x0	表示控制器进入命令发送模式，在 command_mode 设置之后才会生效
Cmd_req	8:8	只写	0x0	命令发送模式下，对 DRAM 发出一次控制命令
Command_mode	0:0	读写	0x0	使控制器进入命令发送模式
0x198				
Status_sref	43:40	只读	0x0	已经进入自刷新模式，每位分别对应一个片选
Srefresh_req	35:32	读写	0x0	自刷新控制信号，设 1 进入自刷新，设 0 退出自刷新
Pre_all_done	27:24	只读	0x0	Precharge All 操作完成
Pre_all_req	19:16	只写	0x0	请求发出 Precharge All 命令，每位分别对应一个片选
Mrs_done	8:8	只读	0x0	命令模式下，表示 MRS 命令发送完毕
Mrs_req	0:0	只写	0x0	命令模式下，向 DRAM 发出一次 MRS 命令，发送的命令序列为 MRS2、MRS3、MRS1、MRS0
0x1A0				
Mr_3_cs_0	63:48	读写	0x0000	向 DRAM CS 0 发送 MRS 3 命令时对应的值
Mr_2_cs_0	47:32	读写	0x0018	向 DRAM CS 0 发送 MRS 2 命令时对应的值
Mr_1_cs_0	31:16	读写	0x0004	向 DRAM CS 0 发送 MRS 1 命令时对应的值
Mr_0_cs_0	15:0	读写	0x0d60	向 DRAM CS 0 发送 MRS 0 命令时对应的值
0x1A8				
Mr_3_cs_1	63:48	读写	0x0000	向 DRAM CS 1 发送 MRS 3 命令时对应的值
Mr_2_cs_1	47:32	读写	0x0018	向 DRAM CS 1 发送 MRS 2 命令时对应的值
Mr_1_cs_1	31:16	读写	0x0004	向 DRAM CS 1 发送 MRS 1 命令时对应的值
Mr_0_cs_1	15:0	读写	0x0d60	向 DRAM CS 1 发送 MRS 0 命令时对应的值
0x1B0				
Mr_3_cs_2	63:48	读写	0x0000	向 DRAM CS 2 发送 MRS 3 命令时对应的值
Mr_2_cs_2	47:32	读写	0x0018	向 DRAM CS 2 发送 MRS 2 命令时对应的值
Mr_1_cs_2	31:16	读写	0x0004	向 DRAM CS 2 发送 MRS 1 命令时对应的值
Mr_0_cs_2	15:0	读写	0x0d60	向 DRAM CS 2 发送 MRS 0 命令时对应的值
0x1B8				
Mr_3_cs_3	63:48	读写	0x0000	向 DRAM CS 3 发送 MRS 3 命令时对应的值
Mr_2_cs_3	47:32	读写	0x0018	向 DRAM CS 3 发送 MRS 2 命令时对应的值

Mr_1_cs_3	31:16	读写	0x0004	向 DRAM CS 3 发送 MRS 1 命令时对应的值
Mr_0_cs_3	15:0	读写	0x0D60	向 DRAM CS 3 发送 MRS 0 命令时对应的值
0x1C0				
tRESET	63:56	读写	0x28	DRAM 初始化前的复位时间 单位为 4096 个时钟周期
tCKE	55:48	读写	0x70	DRAM 初始化从复位释放到 CKE 有效时间 单位为 4096 个时钟周期
tXPR	47:40	读写	0x80	DRAM 初始化从 CKE 有效到 MRS 命令的时间 单位为时钟周期
tMOD	39:32	读写	0x0C	发送 MRS 命令后至下一条命令的时间间隔 单位为时钟周期
tZQCL	31:24	读写	0x03	发送 ZQCL 命令后至下一条命令的时间间隔 单位为 256 个时钟周期
tZQ_CMD	23:16	读写	0x04	不同片选之间发送 ZQ 命令的时间间隔 单位为时钟周期
tWLDQSEN	15:8	读写	0x20	Write Leveling 中，从 MRS 到 DQS 为低的时间间隔 单位为时钟周期
tRDDATA	7:0	读写	0x08	从发送读命令到发送读数据有效命令的时间间隔。 单位为时钟周期 与其他几个参数需要满足关系式： tRDDATA - Cmd_delay - Cmd_timing = CASLAT – 3 该参数最小设置值为 2
0x1C8				
tFAW	61:56	读写	0x30	连续打开 4 个 Bank 的最小允许时间 单位为时钟周期
tRRD	50:48	读写	0x06	打开两个行之间的最小间隔时间 单位为时钟周期
tRCD	43:40	读写	0x09	打开行到对应行的读写操作之间的最小间隔时间 单位为时钟周期
tRP	39:32	读写	0x09	Precharge 操作需要时间 单位为时钟周期
tREF	31:24	读写	0x0C	在版本 0x1 中： 同一片选刷新操作之间的时间间隔 单位为 256 个时钟周期 在版本 0x2/0x3 中： 同一片选刷新操作之间的时间间隔的 Bit[11:4]，与 0x320 寄存器的 tREF_low 组成完整的 tREF 在该版本中，单位为 16 个时钟周期
tRFC	23:16	读写	0x85	刷新操作需要时间 单位为时钟周期
tZQCS	15:8	读写	0x40	ZQCS 操作需要时间 单位为时钟周期
tZQperiod	7:0	读写	0x04	同一片选 ZQCS 操作之间的时间间隔 单位为 tREF（刷新操作之间的时间间隔）

				无论在什么模式下，该参数都不可设为 0。
0x1D0				
tODTL	59:56	读写	0x0A	Write Leveling 之从 ODT 无效到发 MRS 命令时间间隔 单位为时钟周期
tXSRD	55:48	读写	0x02	从自刷新模式恢复到第一条访问之间的最小时间间隔 单位为 256 个时钟周期
tPHY_RDLAT	43:40	读写	0x09	读内存操作的 PHY 内部读数据同步时间 0x5 时一般可以正常工作，极端情况下可以将这个值增加或减小。减小可能会影响到读数据的正确性，增加会增加读操作的延迟 当 Rdfifo_valid 有效时，这个配置不起作用 单位为时钟周期
tPHY_WRLAT	36:32	读写	0x05	从发送写命令到发送写数据的时间间隔。 单位为时钟周期 与其他几个参数需要满足关系式： tPHY_WRLAT – Cmd_delay – Cmd_timing = WRLAT – 4 该参数最小设置值为 2
tRAS_max	25:8	读写	0x00000	在版本 0x1 中： 行打开的最长有效时间 单位为时钟周期 在版本 0x2/0x3 中： 保留
tRAS_min	5:0	读写	0x1C	行打开的最短有效时间 单位为时钟周期
0x1D8				
tXPDLL	63:56	读写	0x14	从离开 Power down （DLL 关闭）状态到下一个命令的间隔时间 单位为始终周期
tXP	55:48	读写	0x05	从离开 Power down （DLL 打开）状态到下一个命令的间隔时间
tWR	44:40	读写	0x0C	写恢复时间 单位为时钟周期 该参数设置值应大于等于 MRS 中设置值
tRTP	34:32	读写	0x06	读到 Precharge 操作的间隔时间 单位为时钟周期
tRL	27:24	读写	0x0A	读操作延迟，相当于 CASLAT 单位为时钟周期 该参数设置值应大于等于 MRS 中设置值
tWL	19:16	读写	0x08	写操作延迟，相当于 WRLAT 单位为时钟周期 该参数设置值应大于等于 MRS 中设置值
tCCD	11:8	读写	0x04	两个读写操作之间的最小间隔时间 单位为时钟周期
tWTR	3:0	读写	0x06	写操作到读操作之间的最小间隔时间

				单位为时钟周期
0x1E0				
tW2R_diffCS	61:56	读写	0x03	不同 CS 上的写操作到读操作之间的间隔时间减去 1 的值 单位为时钟周期，最小值等于 tCCD+tWL-tRL，一般应增加 2 或更多
tW2W_diffCS_adj	53:48	读写	0x0	不同 CS 上的写操作到写操作之间的附加间隔时间 单位为时钟周期，一般应增加 2 或更多
tR2P_sameBA_adj	45:40	读写	0x0	相同 Bank 上的读操作到 Precharge 之间的附加间隔时间 单位为时钟周期
tW2P_sameBA_adj	37:32	读写	0x0	相同 Bank 上的写操作到 Precharge 之间的附加间隔时间 单位为时钟周期
tR2R_sameBA_adj	29:24	读写	0x0	相同 Bank 上的读操作到读操作之间的附加间隔时间 单位为时钟周期
tR2W_sameBA_adj	21:16	读写	0x0	相同 Bank 上的读操作到写操作之间的附加间隔时间 单位为时钟周期
tW2R_sameBA_adj	13:8	读写	0x0	相同 Bank 上的写操作到读操作之间的附加间隔时间 单位为时钟周期
tW2W_sameBA_adj	5:0	读写	0x0	相同 Bank 上的写操作到写操作之间的附加间隔时间 单位为时钟周期
0x1E8				
tR2R_diffCS_adj	61:56	读写	0x01	不同 CS 上的读操作到读操作之间的附加间隔时间 单位为时钟周期，一般应增加 2 或更多
tR2W_diffCS	53:48	读写	0x06	不同 CS 上的读操作到写操作之间的间隔时间减去 1 的值 单位为时钟周期，最小值等于 tCCD+tRL-tWL，一般应增加 2 或更多
tR2P_sameCS	44:40	读写	0x0	相同 CS 上的读操作到 Precharge 之间的间隔时间减去 1 的值 单位为时钟周期
tW2P_sameCS	37:32	读写	0x0	相同 CS 上的读操作到读操作之间的附加间隔时间 单位为时钟周期
tR2R_sameCS_adj	27:24	读写	0x0	相同 CS 上的读操作到读操作之间的附加间隔时间 单位为时钟周期
tR2W_sameCS_adj	21:16	读写	0x0	相同 CS 上的读操作到写操作之间的附加间隔时间 单位为时钟周期
tW2R_sameCS_adj	13:8	读写	0x0	相同 CS 上的写操作到读操作之间的附加间隔时间 单位为时钟周期
tW2W_sameCS_adj	5:0	读写	0x0	相同 CS 上的写操作到写操作之间的附加间隔时间 单位为时钟周期
0x1F0				
Power_up	59:56	读写	0x0	分别对应四个 CS。设为 1 时，可以使对应的 CS 离开或者不进入 Power down 状态。
Age_step	55:48	读写	0x8	Power down 计数器步长。
tCPDED	47:40	读写	0x1	CKE 为 0 后，命令和地址总线失效时间

				单位为时钟周期
Cs_map	39:32	读写	0xE4	CS 地址映射控制，每两位分别对应地址译码后的 CS 与真实 CS 之间的映射关系
Bs_config	31:24	读写	0xFF	命令调度 CS 状态使能 Bit7：CS3 对应状态机状态使能 1-使能；0-禁用（下同） Bit6：CS3 对应状态机状态重置 1-解复位；0-重置（下同） Bit5：CS2 对应状态机状态使能 Bit4：CS2 对应状态机状态重置 Bit3：CS1 对应状态机状态使能 Bit2：CS1 对应状态机状态重置 Bit1：CS0 对应状态机状态使能 Bit0：CS0 对应状态机状态重置
Nc	18:16	读写	0x0	多通道模式使能 000 – 普通 64 位模式 001 – 多通道模式 011 – 普通 16 位模式 101 – 普通 32 位模式 其它 – 保留
Pr_r2w	11:8	读写	0x1	在版本 0x1 中： 读操作优先级是否高于写操作 在版本 0x2/0x3 中： 保留
Placement_en	0:0	读写	0x1	使能读写命令重排逻辑
0x1F8				
Hw_pd_3	59:56	读写	0x0	从低到高分别对应 Active Standby，Fast Power Down，Slow Power Down 和 Self Refresh。设为 1 表示允许 CS3 进入对应的低功耗状态。 需要注意的是，该值不能配置为 4‘h8 和 4‘h9，因为 SelfRefresh 不能在没有进行 precharge 的情况下进行。 另外，bit2 和 bit1 只能不能同时有效，具体与 MR0 的 A12 相匹配： MR0 的 A12 ： 为 0，仅能使能 bit2， 为 1，仅能使能 bit1。
Hw_pd_2	51:48	读写	0x0	设为 1 表示允许 CS2 进入对应的低功耗状态。
Hw_pd_1	43:40	读写	0x0	设为 1 表示允许 CS1 进入对应的低功耗状态。
Hw_pd_0	35:32	读写	0x0	设为 1 表示允许 CS0 进入对应的低功耗状态。
Credit_16	29:24	读写	0x4	在版本 0x1 中： 16 位通道优先级设置 在版本 0x2/0x3 中： 保留
Credit_32	21:16	读写	0x8	在版本 0x1 中： 32 位通道优先级设置

				在版本 0x2/0x3 中： 保留
Credit_64	13:8	读写	0x10	在版本 0x1 中： 64 位通道优先级设置 在版本 0x2/0x3 中： 保留
Selection_en	0:0	读写	0x1	在版本 0x1 中： 不同通道优先级调度使能 在版本 0x2/0x3 中： 保留
0x200				
Cmdq_age_16	59:48	读写	0xC00	16 位通道调度最大超时时间
Cmdq_age_32	43:32	读写	0xC00	32 位通道调度最大超时时间
Cmdq_age_64	27:16	读写	0xC00	64 位通道调度最大超时时间
tCKESR	15:8	读写	0x07	自刷新时，CKE 为低的最短时间 单位为时钟周期
tRDPDEN	7:0	读写	0x0C	从发出 RD/RDA 命令到进入低功耗状态的时间间隔 单位为时钟周期
0x208				
Wfifo_age	59:48	读写	0xC00	在版本 0x1 中： 写队列中命令最大超时时间。 在版本 0x2/0x3 中： 保留
Rfifo_age	43:32	读写	0xF80	读队列中命令最大超时时间。
Power_stat3	27:24	只读	0x0	从低到高分别对应 Active Standby，Fast Power Down，Slow Power Down 和 Self Refresh。设为 1 表示 CS3 处于对应的低功耗状态。
Power_stat2	19:16	只读	0x0	设为 1 表示 CS2 处于对应的低功耗状态。
Power_stat1	11:8	只读	0x0	设为 1 表示 CS1 处于对应的低功耗状态。
Power_stat0	3:0	只读	0x0	设为 1 表示 CS0 处于对应的低功耗状态。
0x210				
Active_age	63:48	读写	0x08	Active Standby 低功耗状态计数器
Cs_place_0	40:40	读写	0x0	普通模式或窗口 0 译码时 CS 在地址中的位置 0 – 译码方式为{CS、ROW、BA、COL} 1 – 译码方式为 关闭 addr_new：{ROW[x-1:0]、ROW[:x]、CS、BA、COL}； 打开 addr_new：{ROW[y+x-1:y]、CS、ROW[:y]、BA、ROW[y-1:0]、COL}； 其中，x 表示 CS 的位宽，y 表示 ROW 的低位位宽，y 由 addr_new[3:0]决定。
Addr_win_0	35:32	读写	0xF	普通模式或窗口 0 地址命中及配置 Bit [3:2]：窗口使用 DRAM 的 Bank 数 11 – 8 bank；10 – 4 bank；01 – 2 bank；00 – 保留 Bit [1:0]：窗口使用 DRAM 位数

				11 – 64 位；10 – 32 位；01 – 16 位；00 – 保留
Cs_diff_0	27:24	读写	0x0	普通模式或窗口 0 实际使用的 CS 译码前地址与 2 之差 当 Pm_nc 有效时， 对于 64 位窗口，应该为 2； 对于 32 位窗口，应该为 1； 对于 16 位窗口，应该为 0
Row_diff_0	19:16	读写	0x2	普通模式或窗口 0 实际使用的行地址线个数与 16 之差 这个值等于 16 – 实际使用的行地址线个数
Ba_diff_0	9:8	读写	0x0	普通模式或窗口 0 实际使用的 BA 线个数与 3 之差 这个值等于 3 – 实际使用的 BA 线个数
Col_diff_0	3:0	读写	0x6	普通模式或窗口 0 实际使用的列地址线个数与 16 之差 这个值等于 16 – 实际使用的列地址线个数
0x218				
Fastpd_age	63:48	读写	0x08	Fast Powerdown 低功耗状态计数器
Cs_place_1	40:40	读写	0x0	在版本 0x1 中： 普通模式或窗口 1 译码时 CS 在地址中的位置 0 – 译码方式为{CS、ROW、BA、COL} 1 – 译码方式为{ROW、CS、BA、COL} 在版本 0x2/0x3 中： 保留
Addr_win_1	35:32	读写	0xF	在版本 0x1 中： 普通模式或窗口 1 地址命中及配置 Bit [3:2]：窗口使用 DRAM 的 Bank 数 11 – 8 bank；10 – 4 bank；01 – 2 bank；00 – 保留 Bit [1:0]：窗口使用 DRAM 位数 11 – 64 位；10 – 32 位；01 – 16 位；00 – 保留 在版本 0x2/0x3 中： 保留
Cs_diff_1	27:24	读写	0x0	在版本 0x1 中： 普通模式或窗口 1 实际使用的 CS 译码前地址与 2 之差 当 Pm_nc 有效时， 对于 64 位窗口，应该为 2； 对于 32 位窗口，应该为 1； 对于 16 位窗口，应该为 0 在版本 0x2/0x3 中： 保留
Row_diff_1	19:16	读写	0x2	在版本 0x1 中： 普通模式或窗口 1 实际使用的行地址线个数与 16 之差 这个值等于 16 – 实际使用的行地址线个数 在版本 0x2/0x3 中： 保留
Ba_diff_1	9:8	读写	0x0	在版本 0x1 中： 普通模式或窗口 1 实际使用的 BA 线个数与 3 之差 这个值等于 3 – 实际使用的 BA 线个数

				在版本 0x2/0x3 中： 保留
Col_diff_1	3:0	读写	0x6	在版本 0x1 中： 普通模式或窗口 1 实际使用的列地址线个数与 16 之差 这个值等于 16 – 实际使用的列地址线个数 在版本 0x2/0x3 中： 保留
0x220				
Slowpd_age	63:48	读写	0x08	Slow Powerdown 低功耗状态计数器
Cs_place_2	40:40	读写	0x0	在版本 0x1 中： 普通模式或窗口 2 译码时 CS 在地址中的位置 0 – 译码方式为{CS、ROW、BA、COL} 1 – 译码方式为{ROW、CS、BA、COL} 在版本 0x2/0x3 中： 保留
Addr_win_2	35:32	读写	0xF	在版本 0x1 中： 普通模式或窗口 2 地址命中及配置 Bit [3:2]：窗口使用 DRAM 的 Bank 数 11 – 8 bank；10 – 4 bank；01 – 2 bank；00 – 保留 Bit [1:0]：窗口使用 DRAM 位数 11 – 64 位；10 – 32 位；01 – 16 位；00 – 保留 在版本 0x2/0x3 中： 保留
Cs_diff_2	27:24	读写	0x0	在版本 0x1 中： 普通模式或窗口 2 实际使用的 CS 译码前地址与 2 之差 当 Pm_nc 有效时， 对于 64 位窗口，应该为 2； 对于 32 位窗口，应该为 1； 对于 16 位窗口，应该为 0 在版本 0x2/0x3 中： 保留
Row_diff_2	19:16	读写	0x2	在版本 0x1 中： 普通模式或窗口 2 实际使用的行地址线个数与 16 之差 这个值等于 16 – 实际使用的行地址线个数 在版本 0x2/0x3 中： 保留
Ba_diff_2	898	读写	0x0	在版本 0x1 中： 普通模式或窗口 2 实际使用的 BA 线个数与 3 之差 这个值等于 3 – 实际使用的 BA 线个数 在版本 0x2/0x3 中： 保留
Col_diff_2	3:0	读写	0x6	在版本 0x1 中： 普通模式或窗口 2 实际使用的列地址线个数与 16 之差 这个值等于 16 – 实际使用的列地址线个数

				在版本 0x2/0x3 中： 保留
0x228				
Selfref_age	63:48	读写	0x08	Selfrefresh 低功耗状态计数器
Cs_place_3	40:40	读写	0x0	在版本 0x1 中： 普通模式或窗口 3 译码时 CS 在地址中的位置 0 – 译码方式为{CS、ROW、BA、COL} 1 – 译码方式为{ROW、CS、BA、COL} 在版本 0x2/0x3 中： 保留
Addr_win_3	35:32	读写	0xF	在版本 0x1 中： 普通模式或窗口 3 地址命中及配置 Bit [3:2]：窗口使用 DRAM 的 Bank 数 11 – 8 bank；10 – 4 bank；01 – 2 bank；00 – 保留 Bit [1:0]：窗口使用 DRAM 位数 11 – 64 位；10 – 32 位；01 – 16 位；00 – 保留 在版本 0x2/0x3 中： 保留
Cs_diff_3	27:24	读写	0x0	在版本 0x1 中： 普通模式或窗口 3 实际使用的 CS 译码前地址与 2 之差 当 Pm_nc 有效时， 对于 64 位窗口，应该为 2； 对于 32 位窗口，应该为 1； 对于 16 位窗口，应该为 0 在版本 0x2/0x3 中： 保留
Row_diff_3	19:16	读写	0x2	在版本 0x1 中： 普通模式或窗口 3 实际使用的行地址线个数与 16 之差 这个值等于 16 – 实际使用的行地址线个数 在版本 0x2/0x3 中： 保留
Ba_diff_3	9:8	读写	0x0	在版本 0x1 中： 普通模式或窗口 3 实际使用的 BA 线个数与 3 之差 这个值等于 3 – 实际使用的 BA 线个数 在版本 0x2/0x3 中： 保留
Col_diff_3	3:0	读写	0x6	在版本 0x1 中： 普通模式或窗口 3 实际使用的列地址线个数与 16 之差 这个值等于 16 – 实际使用的列地址线个数 在版本 0x2/0x3 中： 保留
0x230				
Win_mask_0	59:32	读写	0xFFFFF00	在版本 0x1 中： 0 号窗口 MASK，对应地址[47:20]

				在版本 0x2/0x3 中： 内存地址范围的 MASK，对应地址[47:20]
Win_base_0	27:0	读写	0x0000000	在版本 0x1 中： 0 号窗口 BASE，对应地址[47:20] 在版本 0x2/0x3 中： 内存地址范围的 BASE，对应地址[47:20]
0x238				
Win_mask_1	59:32	读写	0xFFFFF00	在版本 0x1 中： 1 号窗口 MASK，对应地址[47:20] 在版本 0x2/0x3 中： 内存地址范围的 MASK，对应地址[47:20]，为支持 3CS 的情况而设置
Win_base_1	27:0	读写	0x0000100	在版本 0x1 中： 1 号窗口 BASE，对应地址[47:20] 在版本 0x2/0x3 中： 内存地址范围的 BASE，对应地址[47:20]，为支持 3CS 的情况而设置
0x240				
Win_mask_2	59:32	读写	0xFFFFF00	在版本 0x1 中： 2 号窗口 MASK，对应地址[47:20] 在版本 0x2/0x3 中： 保留
Win_base_2	27:0	读写	0x0000200	在版本 0x1 中： 2 号窗口 BASE，对应地址[47:20] 在版本 0x2/0x3 中： 保留
0x248				
Win_mask_3	59:32	读写	0xFFFFF00	在版本 0x1 中： 3 号窗口 MASK，对应地址[47:20] 在版本 0x2/0x3 中： 保留
Win_base_3	27:0	读写	0x0000300	在版本 0x1 中： 3 号窗口 BASE，对应地址[47:20] 在版本 0x2/0x3 中： 保留
0x250				
Cmd_monitor	55:48	读写	0x0	在版本 0x1 中： Bit 7：使能命令队列 3 监控功能 Bit 6：复位命令队列 3 性能计数值 Bit 5：使能命令队列 2 监控功能 Bit 4：复位命令队列 2 性能计数值 Bit 3：使能命令队列 1 监控功能 Bit 2：复位命令队列 1 性能计数值

				Bit 1：使能命令队列 0 监控功能 Bit 0：复位命令队列 0 性能计数值 在版本 0x2/0x3 中： Bit 1：使能命令队列监控功能 Bit 0：复位命令队列性能计数值 其他：保留
Axi_monitor	41:32	读写	0x0	使能 AXI 命令队列性能监控，每两位控制一个 AXI 监控模块，控制方法与 Cmd_monitor 相同
Ecc_code	31:24	只读	0x0	第一次发生 ECC 错误时从内存读出的校验码 记录的出错信息的时机由 Int_vector[0]或 Int_vector[1]由 0 变 1 时触发，使用 Ecc_enable[3]进行配置
Ecc_enable	20:16	读写	0x0	ECC 功能使能 Bit-5：使能读后写模式，即所有的带数据屏蔽的写行为都需要进行先读数据再写入的过程。 Bit-3：设置保存 ECC 出错信号时机 0 - 出现 ECC 错时触发；1 – 出现两位错时触发 Bit-2：使能写时 ECC 校验错的内部总线报错（异常） Bit-1：使能读时 ECC 校验错的内部总线报错(异常)Bit-0：使能 ECC 功能（只在 64 位模式下有效）
Int_vector	9:8	读写	0x0	中断向量寄存器 Bit-1：ECC 两位校验错 Bit-0：ECC 校验错（包括一位错与两位错） 对这个寄存器的读操作将得到当前的 ECC 出错情况，对这个寄存器的“写 1”操作将清除对应的位
Int_enable	1:0	读写	0x0	中断使能寄存器 Bit-1：ECC 两位校验错中断使能 Bit-0：ECC 校验错中断使能（包括一位错与两位错）
0x258				
0x260				
Ecc_addr	63:0	只读	0x0	第一次发生 ECC 错误时向内存读的出错地址 记录的出错信息的时机由 Int_vector[0]或 Int_vector[1]由 0 变 1 时触发，使用 Ecc_enable[3]进行配置
0x268				
Ecc_data	63:0	只读	0x0	第一次发生 ECC 错误时从内存读出的数据 记录的出错信息的时机由 Int_vector[0]或 Int_vector[1]由 0 变 1 时触发，使用 Ecc_enable[3]进行配置
0x270				
Lpbk_ecc_mask	57:56	只读	0x0	自循环测试第一次出错时的 ECC MASK 值 Bit 1：对应于 ECC MASK 的上升沿数据 Bit 0：对应于 ECC MASK 的下降沿数据
Prbs_init	54:32	读写	0x10	自循环测试时使用的 PRBS 初始值
Lpbk_error	24:24	只读	0x0	自循环测试出错
Prbs_23	16:16	读写	0x0	自循环测试时使用的编码方式

				1 – PRBS 23 0 – PRBS 7
Lpbk_start	8:8	读写	0x0	自循环测试开始
Lpbk_en	0:0	读写	0x0	自循环测试模式使能
0x278				
Lpbk_ecc	63:48	只读	0x0	自循环测试第一次出错时的 ECC 值 Bit [63:54]：对应于 ECC 的上升沿数据 Bit [53:48]：对应于 ECC 的下降沿数据
Lpbk_data_mask	47:32	只读	0x0	自循环测试第一次出错时的 DQM 值 Bit [47:40]：对应于 DQM 的上升沿数据 Bit [39:32]：对应于 DQM 的下降沿数据
Lpbk_correct	31:16	只读	0x0	自循环测试第一次出错时的 PRBS 编码 Bit [31:24]：对应于上升沿数据 Bit [23:16]：对应于下降沿数据
Lpbk_counter	15:0	只读	0x0	自循环测试第一次出错时的计数周期
0x280				
Lpbk_data_r	63:0	只读	0x0	自循环测试第一次出错时的 DQ 上升沿数据
0x288				
Lpbk_data_f	63:0	只读	0x0	自循环测试第一次出错时的 DQ 下降沿数据
0x290				
Axi0_bw_w	63:32	只读	0x0	AXI0 写带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
Axi0_bw_r	31:0	只读	0x0	AXI0 读带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
0x298				
Axi0_latency_w	63:32	只读	0x0	AXI0 写延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
Axi0_latency_r	31:0	只读	0x0	AXI0 读延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
0x2A0				
Axi1_bw_w	63:32	只读	0x0	AXI1 写带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
Axi1_bw_r	31:0	只读	0x0	AXI1 读带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
0x2A8				
Axi1_latency_w	63:32	只读	0x0	AXI1 写延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
Axi1_latency_r	31:0	只读	0x0	AXI1 读延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
0x2B0				
Axi2_bw_w	63:32	只读	0x0	AXI2 写带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
Axi2_bw_r	31:0	只读	0x0	AXI2 读带宽性能计数值

				这个值表示 1M 个时钟周期里总线数据有效的周期数
0x2B8				
Axi2_latency_w	63:32	只读	0x0	AXI2 写延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
Axi2_latency_r	31:0	只读	0x0	AXI2 读延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
0x2C0				
Axi3_bw_w	63:32	只读	0x0	AXI3 写带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
Axi3_bw_r	31:0	只读	0x0	AXI3 读带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
0x2C8				
Axi3_latency_w	63:32	只读	0x0	AXI3 写延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
Axi3_latency_r	31:0	只读	0x0	AXI3 读延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
0x2D0				
Axi4_bw_w	63:32	只读	0x0	AXI4 写带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
Axi4_bw_r	31:0	只读	0x0	AXI4 读带宽性能计数值 这个值表示 1M 个时钟周期里总线数据有效的周期数
0x2D8				
Axi4_latency_w	63:32	只读	0x0	AXI4 写延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
Axi4_latency_r	31:0	只读	0x0	AXI4 读延迟性能计数值 这个值表示 256K 个访问的总延迟周期之和
0x2E0				
Cmdq0_bw_w	63:32	只读	0x0	命令队列 0 写带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数
Cmdq0_bw_r	31:0	只读	0x0	命令队列 0 读带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数
0x2E8				
Cmdq0_latency_w	63:32	只读	0x0	命令队列 0 写延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和
Cmdq0_latency_r	31:0	只读	0x0	命令队列 0 读延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和
0x2f0				
Cmdq1_bw_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 1 写带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
Cmdq1_bw_r	31:0	只读	0x0	在版本 0x1 中：

				命令队列 1 读带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
0x2f8				
Cmdq1_latency_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 1 写延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
Cmdq1_latency_r	31:0	只读	0x0	在版本 0x1 中： 命令队列 1 读延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
0x300				
Cmdq2_bw_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 2 写带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
Cmdq2_bw_r	31:0	只读	0x0	在版本 0x1 中： 命令队列 2 读带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
0x308				
Cmdq2_latency_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 2 写延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
Cmdq2_latency_r	31:0	只读	0x0	在版本 0x1 中： 命令队列 2 读延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
0x310				
Cmdq3_bw_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 3 写带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
Cmdq3_bw_r	31:0	只读	0x0	在版本 0x1 中：

				命令队列 3 读带宽性能计数值 这个值表示 64K 个时钟周期里总线数据有效的周期数 在版本 0x2/0x3 中： 保留
0x318				
Cmdq3_latency_w	63:32	只读	0x0	在版本 0x1 中： 命令队列 3 写延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
Cmdq3_latency_r	31:0	只读	0x0	在版本 0x1 中： 命令队列 3 读延迟性能计数值 这个值表示 64K 个访问的总延迟周期之和 在版本 0x2/0x3 中： 保留
0x320				
tRESYNC_length	56:63	读写	0x8	在版本 0x1/0x2 中： 保留 在版本 0x3 中： Resync 命令请求保持的时间。当 cs_resync 不为 4’b0 时，该值不得小于 0
tRESYNC_shift	55:48	读写	0x8	在版本 0x1/0x2 中： 保留 在版本 0x3 中： 该参数与 tRESYNC_max/tRESYNC_max 配合使用，表示两个 resync 命令之间的时间间隔。当 cs_resync 不为 4’b0 时，该值不得小于 0
tRESYNC_max	47:40	读写	0x30	在版本 0x1/0x2 中： 保留 在版本 0x3 中： 该参数移位 tRESYNC_shift 后，表示两个 resync 命令之间的最大间隔。当 cs_resync 不为 4’b0 时，该值不得小于 0
tRESYNC_min	39:32	读写	0x10	在版本 0x1/0x2 中： 保留 在版本 0x3 中： 该参数移位 tRESYNC_shift 后，表示两个 resync 命令之间的最小间隔。当 cs_resync 不为 4’b0 时，该值不得小于 0
Pre_predict	31:16	读写	0x1	在版本 0x1/0x2 中： 保留 在版本 0x3 中： Bit3-0：使能对应片选信号的预 PRECHARGE 操作 Bit15-4：该 BANK 在空闲此段时间后，可以发出

				PRECHARGE 的操作
tXS	15:8	读写	0x60	在版本 0x1/0x2 中： 保留 在版本 0x3 中： 从自刷新模式恢复到第一条不需要锁定 DLL 的访问之间的最小时间间隔
tREF_low	3:0	读写	0x0	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 同一片选刷新操作之间的时间间隔的 Bit[3:0] 在该版本中，单位为 16 个时钟周期
0x328				
tRESYNC_delay	7:0	读写	0x10	在版本 0x1/0x2 中： 保留 在版本 0x3 中： 命令总线空闲该时间后，才能发出 resync 命令请求。当 cs_resync 不为 4'b0 时，该值不得小于 0
0x330				
Stat_en	56:56	读写	0x0	在版本 0x1 中： 保留； 在版本 0x2/0x3 中： 在 retry 使能的情况下，使能对重发送（retry）次数的统计。
Rdbuffer_max	54:48	只读	0x0	在版本 0x1 中： 保留； 在版本 0x2/0x3 中： 表示读数据 FIFO 中用到的 FIFO 项数最大值
Retry	40:40	读写	0x0	在版本 0x1 中: 保留； 在版本 0x2/0x3 中： 使能读队列的重发送功能。
Wr_pkg_num	37:32	读写	0x10	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 写队列连续发送写请求的个数
Rwq_rb	24:24	读写	0x0	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 使能读写轮转的切换模式
Stb_en	16:16	读写	0x1	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 流模式读预取使能
Addr_new	12:8	读写	0x0	在版本 0x1 中：

				保留 在版本 0x2/0x3 中： Bit4 表示新的译码方式使能， Bit[3:0]表示地址偏移量。 译码方式具体为： { cs、 row[:x]、 bank、 row[x-1:0]、 col}, 其中 x 由 addr_new[3:0]决定。Addr_new[3:0]需要满足以下条件： Col_width + channel_width ≤ 12 + addr_new[3:0] ≤ Col_width + channel_width + row_width。 其中，Col_width = 16 – col_diff_0； Row_width = 16 – row_diff_0； 当 nc 为 0，时，Channel_width 为 3；nc 为 3 时，channel_width 为 2；nc 为 5 时，Channel_width 为 1。
tRDQidle	7:0	读写	0xA	在版本 0x1 中： 保留 在版本 0x2 中： Bit3-0：表示当读操作占用总线时，其请求空闲时间为(15-该值)个时间周期时，将切换到写操作 Bit7:4：保留 在版本 0x3 中： 表示当读操作占用总线时，其请求空闲时间为该值表示的时间周期时，将切换到写操作
0x338				
Rd_fifo_depth	37:34	读写	0x0	在版本 0x1 中： 保留； 在版本 0x2/0x3 中： Bit3：表示此时使能的读数据 FIFO 深度是 32； Bit2：表示此时使能的读数据 FIFO 深度是 16； Bit1：表示此时使能的读数据 FIFO 深度是 8； Bit0：表示此时使能的读数据 FIFO 深度是 4； 当 bit[3:0]有两个及其以上使能时，高位的优先级较高。
Retry_cnt	31:0	只读		在版本 0x1 中： 保留 在版本 0x2/0x3 中： 在 stat_en 使能的情况下，统计的重发送（retry）次数。
0x340				
tREFretention	63:32	读写	0x30D400	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 它与内存周期的积应稍小于内存的 refresh retention time。 通常内存的 refresh retention time 为 64ms 或 32ms。 单位为时钟周期
Ref_num	18:16	读写	0x7	在版本 0x1 中：

				保留 在版本 0x2/0x3 中： 该值加 1 后，表示刷新可提前/推后的个数。
tREF_IDLE	15:8	读写	0x0F	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 表示在命令队列空闲该段时间后，控制器发送 REF 命令
Ref_sch_en	1:0	读写	0x1	在版本 0x1 中： 保留 在版本 0x2 中： Bit0 使能刷新调度功能。需要注意的是，该功能在低功耗时需要关闭。 Bit1 保留 在版本 0x3 中： Bit0 使能刷新调度功能。 Bit1 使能低功耗时刷新调度功能。当低功耗使能时（hw_pd0/1/2/3 有效），bit1 应该与 bit0 保持一致。
0x348				
0x350				
Lpbk_data_en	63:0	读写	0xffff_ffff_ffff_ffff	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 使能自循环测试模式下数据比较，设为 0 时不比较相应位
0x358				
Lpbk_ecc_mask_en	16:16	读写	0x1	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 使能自循环测试模式下 ECC 屏蔽比较，设为 0 时不比较对应位
Lpbk_ecc_en	15:8	读写	0xff	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 使能自循环测试模式下 ECC 比较，设为 0 时不比较对应位
Lpbk_data_mask_en	7:0	读写	0xff	在版本 0x1 中： 保留 在版本 0x2/0x3 中： 使能自循环测试模式下数据屏蔽比较，设为 0 时不比较对应位
0x360				
Ecc_int_cnt_fatal	47:40	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中：

				ECC 两位出错引起的中断次数统计
Ecc_int_cnt_error	39:32	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中： ECC 出错引起的中断次数统计
Ecc_cnt_cs_3	31:24	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中： CS3 的 Ecc 出错次数
Ecc_cnt_cs_2	23:16	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中： CS2 的 Ecc 出错次数
Ecc_cnt_cs_1	15:8	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中： CS1 的 Ecc 出错次数
Ecc_cnt_cs_0	7:0	读写	0x00	在版本 0x1 中： 保留 在版本 0x2/0x3 中： CS0 的 Ecc 出错次数
0x368				
0x370				
Prior_age3	15:0	读写	0x55	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 优先级为 3 的命令，(0xff-该值)为命令在队列中等待的时间。该优先级的等待时间最短
Prior_age2	15:0	读写	0x44	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 优先级为 2 的命令，(0xff-该值)为命令在队列中等待的时间。
Prior_age1	15:0	读写	0x33	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 优先级为 1 的命令，(0xff-该值)为命令在队列中等待的时间。
Prior_age0	15:0	读写	0x22	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 优先级为 0 的命令，(0xff-该值)为命令在队列中等待的时间。该优先级的等待时间最长

0x378				
Row_hit_place	0:0	读写	0x0	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 命令队列中出现有前端行命中，后端行冲突的情况，该值为 1 新命令插在行命中前面，反之则插在新命令后面。
0x380				
Zq_cnt_1	63:32	只读	0x0	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 表示 cs1 进行 ZQ 校准的次数。
Zq_cnt_0	31:0	只读	0x0	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 表示 cs0 进行 ZQ 校准的次数。
0x388				
Zq_cnt_3	63:32	只读	0x0	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 表示 cs3 进行 ZQ 校准的次数。
Zq_cnt_2	31:0	只读	0x0	在版本 0x1/0x2 中： 保留 在 版本 0x3 中： 表示 cs2 进行 ZQ 校准的次数。

2.软件编程指南

2.1. 初始化操作

初始化操作由软件向寄存器 Init_start（0x018）写入 1 时开始，在设置 Init_start 信号之前，必须将其它所有寄存器设置为正确的值。

软硬件协同的 DRAM 初始化过程如下：

(1) 软件向所有的寄存器写入正确的配置值，但是 Init_start（0x018）在这一过程中必须保持为 0；

(2) 软件将 Init_start（0x018）设置为 1，这将导致硬件初始化的开始；

(3) PHY 内部开始初始化操作，DLL 将尝试进行锁定操作。如果锁定成功，则可以从 Dll_init_done（0x000）读出对应状态，并可以从 Dll_value_ck（0x000）读写当前锁定延迟线个数；如果锁定不成功，则初始化不会继续进行（此时可以通过设置 Dll_bypass（0x018）使得初始化继续执行）；

(4) DLL 锁定（或者 bypass 设置）之后，控制器将根据对应 DRAM 的初始化要求向 DRAM 发出相应的初始化序列，例如对应的 MRS 命令，ZQCL 命令等等；

(5) 软件可以通过采样 Dram_init（0x160）寄存器来判断内存初始化操作是否完成。

2.2. 复位引脚的控制

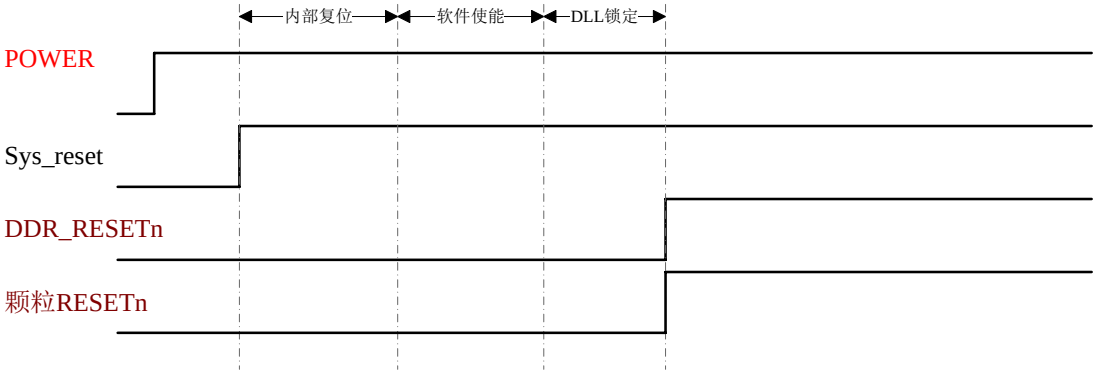
为了在 STR 等状态下更加简单地控制复位引脚，可以通过 reset_ctrl 寄存器进行特别的复位

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。

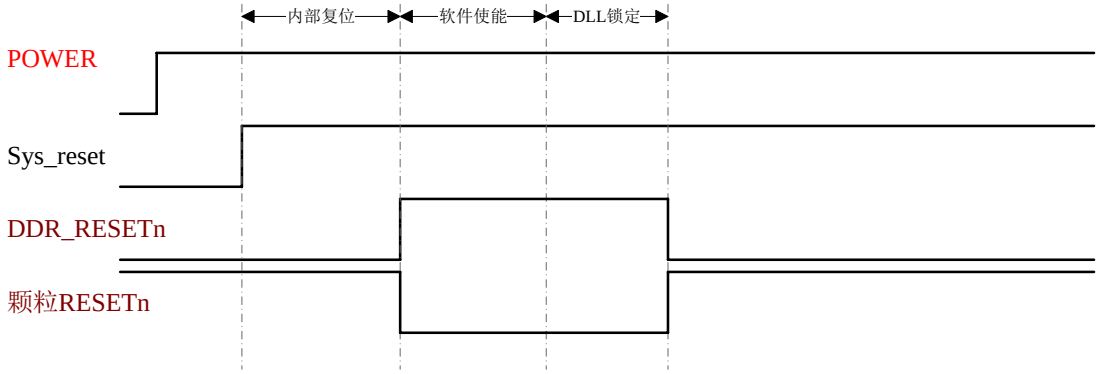
时序如下图所示：



(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。

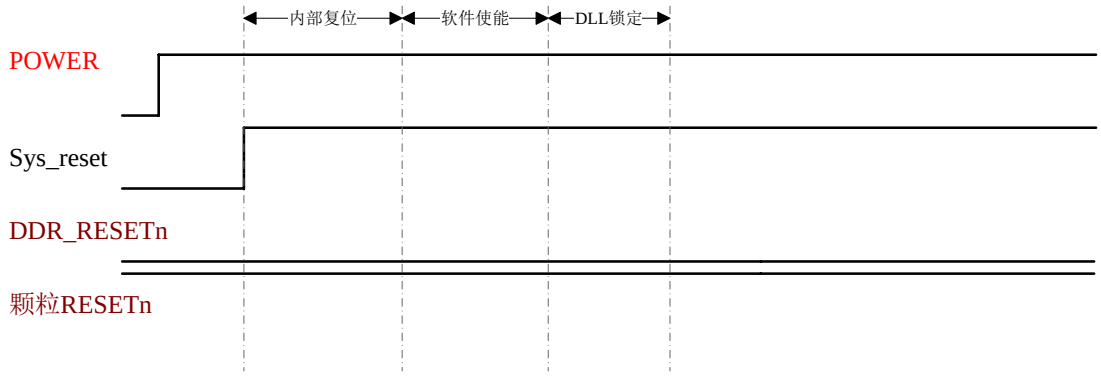
时序如下图所示：



(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。

2.3. Leveling

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。在本控制器中，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。

2.3.1 Write Leveling

Write Leveling 用于配置写 DQS 与时钟之间的相位关系，软件编程需要参照如下步骤。

(1) 完成控制器初始化，参见上一小节内容；

(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（0x180）寄存器，如果为 1，表示可以开始 Write Leveling 请求；

(6) 设置 Lvl_req（0x180）为 1；

(7) 采样 Lvl_done（0x180）寄存器，如果为 1，表示一次 Write Leveling 请求完成；

(8) 采样 Lvl_resp_x（0x180、0x188）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 5-7；如果为 1，则表示 Write Leveling 操作已经成功；

(9) 此时 Dll_wrdqs_x 的值就应该是正确的设置值。

至此 Write Leveling 操作结束。如果这个过程中，第一次采样就发现 Lvl_resp_x 为 1，则这个结果是有问题的，应该检查其它的寄存器是否有错误的设置，这些寄存器可能包括 Wrdqs_lt_half、Dqs_start_edge、Dqs_stop_edge、Dqs_oe_begin、Dqs_oe_end。

(10) 接着根据 Dll_wrdqs_x 的值是否小于 0x40 来设置 Wrdqs_lt_half_x；

(11) 根据 Dll_wrdqs_x 的值是否小于 0x20 来设置 Dll_wrdata_x。如果 Dll_wrdqs_x > 0x20，Dll_wrdata_x = Dll_wrdqs_x – 0x20，否则 Dll_wrdata_x = Dll_wrdqs_x + 0x60；

(12) 根据 Dll_wrdata_x 的值是否小于 0x40 来设置 Wrdata_lt_half_x；

(13) 判断是否存在以下情况：不同的 Dll_wrdata_x 值在 0x40 附近，且有跨越 0x40 边界的情

况出现（指有的 Dll_wrdata_x 略小于 0x40，有的 Dll_wrdata_x 略大于 0x40）。如果出现这种情况，设置对应 Wrdata_lt_half_x == 0 数据组的 Write_clk_delay_x 为 1。再将 tPHY_WRDATA 与 tRDDATA 的值减 1；

(14) 将 Lvl_mode（0x180）设置为 2'b00，退出 Write Leveling 模式；

(15) 在 write leveling 做完后，将 cs_zq(0x168)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。

2.3.2 Gate Leveling

Gate Leveling 用于配置控制器内使能采样读 DQS 窗口的时机，软件编程参照如下步骤。

(1) 完成控制器初始化，参见上一小节内容；

(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) 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 Lvl_resp_x[0]（0x180、0x188）寄存器。如果第一次采样发现 Lvl_resp_x[0]为 1，则将对应的 Dll_gate_x[6:0]增加 1，并重复执行 6-8，直至采样结果为 0，否则进行下一步；

(10) 如果采样结果为 0，则将对应的 Dll_gate_x[6:0]增加 1，并重复执行 6-9；如果为 1，则表示 Gate Leveling 操作已经成功；

至此 Gate Leveling 操作结束，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。

(11) 如果 Dll_gate_x[6:0]与 Dll_wrdata_x[6:0]的和小于 0x20 或者大于 0x60，那么 Dll_rddqs_lt_halt 设置为 1。因为 rddqs 的相位关系实际上等于在输入的读 DQS 基础上再延迟 1/4。

(12) 此时如果 Dll_gate_x 的值大于 0x40，则将 Dll_gate_x 的值减去 0x40；否则将其设为 0 即可。

(13) 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 13 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

(14) 将 Lvl_mode（0x180）设置为 2'b00，退出 Gate Leveling 模式；

(15) 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。

2.4. 单独发起 MRS 命令

内存控制器向内存发出的 MRS 命令次序分别为：

MR2_CS0、MR2_CS1、MR2_CS2、MR2_CS3、

MR3_CS0、MR3_CS1、MR3_CS2、MR3_CS3、

MR1_CS0、MR1_CS1、MR1_CS2、MR1_CS3、

MR0_CS0、MR1_CS1、MR1_CS2、MR1_CS3。

其中，对应 CS 的 MRS 命令是否有效，是由 Cs_mrs 决定，只有 Cs_mrs 上对应每个片选的位

有效，才会真正向 DRAM 发出这个 MRS 命令。对应的每个 MR 的值由寄存器 Mr*_cs*决定。这些值同时也用于初始化内存时的 MRS 命令。

具体操作如下：

(1) 将寄存器 Cs_mrs（0x168）、Mr*_cs*（0x190 – 0x1B8）设置为正确的值；
(2) 设置 Command_mode（0x190）为 1，使控制器进入命令发送模式；
(3) 采样 Status_cmd（0x190），如果为 1，则表示控制器已进入命令发送模式，可以进行下一步操作，如果为 0，则需要继续等待；
(4) 写 Mrs_req（0x198）为 1，向 DRAM 发送 MRS 命令；
(5) 采样 Mrs_done（0x198），如果为 1，则表示 MRS 命令已经发送完毕，可以退出，如果为 0，则需要继续等待；
(6) 设置 Command_mode（0x190）为 0，使控制器退出命令发送模式。

2.5. 任意操作控制总线

内存控制器可以通过命令发送模式向 DRAM 发出任意的命令组合，软件可以设置 Cmd_cs、Cmd_cmd、Cmd_ba、Cmd_a（0x168），在命令发送模式下向 DRAM 发出。

具体操作如下：

(1) 将寄存器 Cmd_cs、Cmd_cmd、Cmd_ba、Cmd_a（0x190）设置为正确的值；
(2) 设置 Command_mode（0x190）为 1，使控制器进入命令发送模式；
(3) 采样 Status_cmd（0x190），如果为 1，则表示控制器已进入命令发送模式，可以进行下一步操作，如果为 0，则需要继续等待；
(4) 写 Cmd_req（0x190）为 1，向 DRAM 发送命令；
(5) 设置 Command_mode（0x190）为 0，使控制器退出命令发送模式。

2.6. 自循环测试模式控制

自循环测试模式可以分别在测试模式下或者正常功能模式下使用，为此，本内存控制器分别实现了两套独立的控制接口，一套用于在测试模式下由测试端口直接控制，另一套用于在正常功能模式下由寄存器配置模块进行配置使能测试。

这两套接口的复用使用端口 test_phy 进行控制，当 test_phy 有效时，使用控制器的 test_*端口进行控制，此时的自测试完全由硬件控制；当 test_phy 无效时，使用软件编程的 pm_*的参数进行控制。使用测试端口的具体信号含义可以参考寄存器参数中的同名部分。

这两套接口从控制的参数来说基本一致，仅仅是接入点不同，在此介绍软件编程时的控制方法。具体操作如下：

(1) 将内存控制器所有的参数全部正确设置；
(2) 将寄存器 Lpbk_en（0x270）设为 1；
(3) 将寄存器 Init_start（0x018）设为 1；
(4) 采样寄存器 Dll_init_done（0x000），如果这个值为 1，表示 DLL 已经锁定，可以进行下一步操作；如果这个值为 0，则需要继续等待；（当使用测试端口进行控制的时候，因为看不到这个寄存器的输出，所以不需要采样这个寄存器，而只需要在此处等待一定的时间，以确保 DLL 锁定完成，再进行下一步操作）；
(5) 将寄存器 Lpbk_start（0x270）设为 1；此时自循环测试正式开始。

到此为止自循环测试已经开始，软件需要经常检测是否有错误发生，具体操作如下：

(6) 采样寄存器 Lpbk_error（0x270），如果这个值为 1，表示有错误发生，此时可以通过 Lpbk_*等观测用寄存器（0x270、0x278、0x280、0x288）来观测第一个出错时的错误数据和正确

数据；如果这个值为 0，表示还没有出现过数据错误。

2.7. ECC 功能使用控制

ECC 功能只有在 64 位模式下可以使用。

Ecc_enable 包括以下 4 个控制位：

Ecc_enable[0]控制是否使能 ECC 功能，只有设置了这个有效位，才会使能 ECC 功能。

Ecc_enable[1]控制是否通过处理器内部的读响应通路进行报错，以使得出现 ECC 两位错的读访问能立即导致处理器核的异常发生。

Ecc_enable[2]控制是否通过处理器内部的写响应通路进行报错，以使得出现 ECC 两位错的写访问（读后写）能立即导致处理器核的异常发生。

Ecc_enable[3]控制寄存器内记录出错信息的触发时机。这些出错信息在没有软件进行处理的情况下不会连续触发，只会记录第一次出错时的信息。这些信息包括 Ecc_code，Ecc_addr，Ecc_data。当 Ecc_enable[3]为 0 的情况下，只要出现了 ECC 错误（包括 1 位错与 2 位错），这个记录就会被触发，当 Ecc_enable[3]为 1 的情况下，只有出现了 ECC 两位错，这个记录才会被触发。而这个“第一次”指的是中断向量寄存器的对应位被置位。也就是说，记录的是导致中断发生的那一次访问。

除此之外，ECC 出错还可以通过中断方式通知处理器核。这个中断通过 Int_enable 进行控制。中断包括两个向量，Int_vector[0]表示出现 ECC 错误（包括 1 位错与 2 位错），Int_vecotr[1]表示出现 ECC 两位错。Int_vector 的清除通过向对应位写 1 实现。