



Synopsys Controller IP

LPDDR5X/5/4X Memory Controller

Databook

DWC LPDDR5X/5/4X Controller - Product Code: H507-0
DWC LPDDR5X/5/4X Controller AFP - Product Code: H508-0

Copyright Notice and Proprietary Information

© 2024 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
<https://www.synopsys.com/company/legal/trademarks-brands.html>

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.

www.synopsys.com

Contents

Revision History	11
Preface	15
Databook Organization	15
Web Resources	16
Reference Documentation	16
Synopsys Statement on Inclusivity and Diversity	16
Customer Support	16
Chapter 1	
Product Overview	19
1.1 General Product Description	20
1.1.1 System Block Diagram	20
1.2 Standards Compliance	23
1.3 Features	24
1.3.1 General Features	24
1.3.2 Power Saving and Low-Power Features	25
1.3.3 Performance Features	25
1.3.4 Programmable SDRAM Parameters	26
1.3.5 Refresh Control Features	26
1.3.6 Supported Data Rates	26
1.3.7 LPDDR5-Specific Features	26
1.3.8 System Bus Interface	27
1.3.9 Unsupported Features and Limitations	27
1.4 Area and Power Numbers	34
1.5 Clock And Reset Requirements	35
1.6 Deliverables	37
Chapter 2	
Architecture	39
2.1 Block Diagram	40
2.2 Block Descriptions	41
2.3 Functions of DDRCTL	43
2.4 Signal Naming Conventions	44
Chapter 3	
Interfaces	45
3.1 AXI Interface	46

3.1.1 AXI Port Interface	46
3.1.2 AXI Port Arbiter	59
3.1.3 AXI Queue Status Interface and Port Throttling	66
3.1.4 AXI QoS	69
3.1.5 AXI Exclusive Access	74
3.2 APB Interface	76
3.2.1 Overview of APB Interface	76
3.2.2 Register Classes	76
3.2.3 Signals Related to APB Interface	76
3.3 Host Interface (HIF)	77
3.3.1 Overview of Host Interface (HIF)	77
3.3.2 Dual HIF Functionality	78
3.3.3 Credit Mechanism	79
3.3.4 Valid and Stall	81
3.3.5 Reads	82
3.3.6 Writes	86
3.3.7 Read-Modify-Write Requests	92
3.3.8 WR Versus RMW for HIF Configurations With DBICTL.dm_en=0	93
3.3.9 Half/Quarter Bus Width Mode	93
3.3.10 hif_cmd_latency	94
3.3.11 hif_go2critical	94
3.3.12 hif_cmd_autopre	94
3.3.13 hif_mrr_data_valid and hif_mrr_data	95
3.3.14 Additional Error Signal Outputs	95
3.4 Hardware Low-Power Interfaces	96
Chapter 4	
DFI	97
4.1 DFI Interface	98
4.1.1 Overview of DFI Interface	98
4.1.2 1:2 Frequency Ratio Mode Considerations	99
4.1.3 DFI Write/Read Data to SDRAM Conversion	99
4.1.4 Command Interface	100
4.1.5 Write Data Interface	100
4.1.6 Read Data Interface	101
4.1.7 Update Interface	101
4.1.8 Status Interface	102
4.1.9 DFI Training	103
4.1.10 Low-Power Control Interface	104
4.1.11 PHY Master Interface	105
4.1.12 MC to PHY Message Interface	108
4.1.13 DFI Error Interface	108
4.1.14 Signals Related to DFI Interface	109
4.1.15 Registers Related to DFI Interface	109
4.2 DFI Updates	110
4.2.1 Overview of DFI Updates	110
4.2.2 DFI MC-Initiated Update Requests	110
4.2.3 DFI PHY-initiated Update Requests	111
4.2.4 Registers Related to DFI Updates	112

4.3 DFI Constraints	113
4.3.1 Overview of DFI Constraints	113
4.3.2 WCK Related Constraints (LPDDR5)	113
4.3.3 WCK Constraints (LPDDR5)	114
4.4 DDRCTL to PHY Connections	115
4.4.1 Single DDRC Dual DFI Configuration	115
4.4.2 Single DDRC Quad DFI Configuration	118
Chapter 5	
Channel Modes	125
5.1 Overview of LPDDR4 / LPDDR5 Channel Modes	126
5.2 DDRCTL Configurations	127
5.2.1 Single DDRC Single DFI Configuration	128
5.2.2 Single DDRC Dual DFI Configuration	131
5.2.3 Single DDRC Quad DFI Configuration	143
Chapter 6	
Address Mapping	153
6.1 Overview of Address Mapping.	154
6.1.1 System Address Regions	154
6.2 Application to HIF Address Mapping.	157
6.3 HIF Address to SDRAM Address Mapping	158
6.4 Recommendations for Optimum SDRAM Utilization.	160
6.5 Non-binary Device Densities.	162
6.6 Mixed Packages.	163
6.6.1 Mixed Packages Address Mapping	163
6.6.2 Mixed Packages Limitation	166
6.7 Bank Hashing.	168
6.7.1 Bank Hashing Function	168
6.8 Registers Related to Address Mapper	172
Chapter 7	
Command Scheduling in DDRC	173
7.1 Overview of Command Scheduling in DDRC	174
7.2 Page Policy	175
7.2.1 Explicit Auto-Precharge (Per Command)	175
7.2.2 Intelligent Precharges	175
7.3 Transaction Stores	177
7.3.1 Read Transaction Store	177
7.3.2 Write Transaction Store	177
7.3.3 Transaction Store State Transitions	178
7.3.4 Read Priority Management	180
7.3.5 Read/Write Turnaround	181
7.4 Address Collision Handling	193
7.5 Write Combine.	194
7.5.1 Page-hit Limiter	194
7.5.2 Visible Window Limiter (UMCTL2_VPRW_EN==1 Only)	195
7.6 Registers Related to Command Scheduling in DDRC.	197

Chapter 8

Memory Scheduling	199
8.1 Burst Mode Operation	200
8.1.1 Overview of Burst Mode Operation	200
8.1.2 Bus Width Selection	200
8.1.3 Sequential Operations	200
8.2 Dynamic SDRAM Constraints.....	202
8.2.1 Overview of Dynamic SDRAM Constraints	202
8.2.2 Timing Constraints	202
8.2.3 Preamble and Postamble (LPDDR4)	212
Chapter 9	
Periodic Memory and PHY Maintenance	213
9.1 Refresh Controls	214
9.1.1 Overview of Refresh Controls	214
9.1.2 Refresh Using Direct Software Request of Refresh Command	214
9.1.3 Refresh Using Auto-Refresh Feature	215
9.1.4 Automatic Temperature Derating	219
9.1.5 Signals Related to Refresh Controls	221
9.1.6 Registers Related to Refresh Controls	222
9.1.7 Dynamic SDRAM Rank Constraints	222
9.2 Refresh Management (RFM)	223
9.2.1 Feature Restrictions	223
9.2.2 Rolling Accumulated ACT (RAA) Counter	223
9.2.3 RFM Command Control	223
9.2.4 Single-bank Mode (LPDDR5 only)	226
9.2.5 Adaptive Refresh Management (ARFM) (LPDDR5 only)	227
9.3 ZQ Calibration.....	228
9.3.1 Overview of ZQ Calibration	228
9.3.2 LPDDR4 Devices	228
9.3.3 LPDDR5 Devices	230
9.3.4 Automatic and Software Initiated ZQ Calibration Commands	232
9.3.5 LPDDR4/LPDDR5 ZQ Reset Command	234
9.3.6 Registers Related to ZQ Calibration	234
9.4 Controller Assisted Drift Tracking by MRR Snooping (LPDDR5)	235
9.4.1 Overview of Controller Assisted Drift Tracking	235
9.4.2 PHY MRR Snoop Control	235
9.4.3 Compensate DRAM Clock Stop	236
9.4.4 Registers Related to Controller Assisted Drift Tracking	238
9.5 Enhanced Incremental Periodic Phase Training (PPT2)	239
9.5.1 Overview of PPT2	239
9.5.2 PPT2 Limitations	240
9.5.3 Retraining Interval	241
9.5.4 Normal PPT2	242
9.5.5 Burst PPT2	246
9.5.6 Registers Related to PPT2	248
9.6 Burst DFI Control Update Request for Core VDD Change	249
9.6.1 Overview of Burst DFI Control Update Request	249
9.6.2 Combination With Frequency Change	249
9.6.3 Burst DFI Control Update Limitations	250

Chapter 10

Memory Control	251
10.1 Mode Register Reads and Writes	252
10.1.1 Overview of Mode Register Reads and Writes	252
10.1.2 Mode Register Writes	252
10.1.3 Mode Register Reads	252
10.1.4 Registers Related to Mode Register Reads and Writes	253
10.2 Data Bus Inversion (DBI)	254
10.2.1 Overview of Data Bus Inversion (DBI)	254
10.2.2 Registers Related to DBI	255

Chapter 11

Low-Power and Power-Saving Features	257
11.1 Overview of Power Saving Features	258
11.2 SDRAM Power Saving Features	259
11.2.1 Overview of SDRAM Power Saving Features	259
11.2.2 Precharge Power-Down	260
11.2.3 Self-Refresh	261
11.2.4 Deep Sleep Mode (LPDDR5)	265
11.2.5 Assertion of dfi_dram_clk_disable	267
11.3 Power Saving in PHY Through DFI Low-Power Control Interface	269
11.4 Hardware Low-Power Interfaces	270
11.4.1 Overview of Hardware Low-Power Interfaces	270
11.4.2 DDRC Hardware Low-Power Interface	270
11.4.3 AXI Low-Power Interface	275
11.5 Fast Frequency Change	281
11.6 Hardware Fast Frequency Change (HWFFC) Support	282
11.6.1 Hardware Fast Frequency Change (HWFFC) Overview	282
11.6.2 Limitations of HWFFC	285
11.6.3 LPDDR4 HWFFC Procedure When HWFFCCTL.hwffc_mode=0	287
11.6.4 HWFFC Procedure When HWFFCCTL.hwffc_mode=1	293
11.6.5 HWLP Driven Retraining	299
11.6.6 HWLP Driven LP2 (PHY Fast Standby)	299
11.6.7 DVFSC and DVFSQ Support with HWFFC	300
11.6.8 Registers Related to HWFFC	302
11.6.9 HWFFC with PHY Mission Mode Firmware	303
11.7 Low-Power Optimized Write Data for LPDDR5/4 Masked Write	309
11.8 BSM Clock Removal	310
11.9 APB Clock Gating	312
11.10 Signals Related to Power Saving	313

Chapter 12

LPDDR5 Specific Features	315
12.1 Overview of LPDDR5 Specific Features	316
12.2 Bank Organization	317
12.2.1 Address Mapping	317
12.2.2 Burst Length	318
12.3 WCK Clocking	319
12.3.1 WCK Behavior	319

12.3.2 Enhanced WCK Always On Mode	320
12.4 Link ECC	321
12.4.1 Overview of Link ECC Support	321
12.4.2 Enabling Link ECC	321
12.4.3 Link ECC Restrictions	321
12.4.4 Controller Behavior During Read Link ECC Errors	321
12.4.5 Inline ECC and Link ECC Features Combination	322
12.4.6 Link ECC Error Reporting	324
12.4.7 Link ECC Data Poisoning	325
12.4.8 Performance Impact	326
12.4.9 Registers Related to Link ECC Support	326
12.5 LPDDR5 Post Package Repair	327
12.5.1 Overview of LPDDR5 Post Package Repair	327
12.5.2 Enabling LPDDR5 Post Package Repair Feature (PPR)	327
12.5.3 Limitations of LPDDR5 PPR	327
12.5.4 Software Sequence for LPDDR5 PPR	327
12.5.5 Registers Related to LPDDR5 PPR	329
12.6 LPDDR5X Support With Synopsys LPDDR5X/5/4X PHY	331

Chapter 13

RAS Features	333
13.1 Memory ECC	334
13.1.1 Inline ECC Support	334
13.2 Scrubber	373
13.2.1 Scrubber Overview	373
13.2.2 Scrub Period	374
13.2.3 Restrictions on Scrub Interval With Auto Power-down/ Auto Self-refresh Idle Time	375
13.2.4 Scrubber Normal Operation	375
13.2.5 Scrubber Address Configuration	375
13.2.6 Address Generation	376
13.2.7 Additional Notes for the ECC Scrubber	377
13.2.8 Status	378
13.2.9 Hardware Controlled Low-Power Operation	378
13.2.10 Software Controlled Low-Power Operation	379
13.2.11 Initialization Writes	379
13.2.12 Scrubber Credit Interface	381
13.3 On-Chip Parity (OCPAR)	386
13.3.1 Overview of On-Chip Parity	386
13.3.2 Enabling On-Chip Parity	387
13.3.3 Write Data Parity	387
13.3.4 Read Data Parity	388
13.3.5 Read Modify Write Operation	390
13.3.6 Parity Poisoning	390
13.3.7 Signals Related to On-Chip Parity	391
13.3.8 Registers Related to On-Chip Parity	391
13.4 Registers Parity Protection (REGPAR)	393
13.4.1 Overview of Registers Parity Protection (REGPAR)	393
13.4.2 Enabling Registers Parity Protection	393
13.4.3 Register Parity Generation	393

13.4.4 Register Parity Checking	395
13.4.5 Register Parity Poisoning	396
13.4.6 Registers Related to REGPAR	397
13.5 On-Chip Command and Address Path Protection (OCCAP).....	398
13.5.1 Overview of On-Chip Command and Address Path Protection (OCCAP)	398
13.5.2 Enabling On-Chip Command and Address Path Protection	398
13.5.3 Protection Mechanisms	398
13.5.4 Arbiter Protection	401
13.5.5 DDRC Protection	401
13.5.6 Poisoning	403
13.5.7 Signals Related to OCCAP	405
13.5.8 Registers Related to OCCAP	405
13.6 On-Chip External SRAM Address Protection (OCSAP)	406
13.6.1 Overview of On-Chip External SRAM Address Protection	406
13.6.2 Enabling On-Chip External SRAM Address Protection (OCSAP)	406
13.6.3 On-Chip External SRAM Address Parity	406
13.6.4 On-Chip External SRAM Address Parity Poisoning	406
13.6.5 Signals Related to On-Chip External SRAM Address Protection	407
13.6.6 Registers Related to On-Chip External SRAM Address Protection	407
13.7 DFI Sideband Watchdog Timers.....	408
13.7.1 Overview of DFI Sideband Watchdog Timers	408
13.7.2 Hardware Parameter Related to DFI Sideband Watchdog Timers	408
13.7.3 Registers Related to DFI Sideband Watchdog Timers	408
13.7.4 Interrupts Related to DFI Sideband Watchdog Timers	408
13.7.5 Controller-initiated DFI Update Watchdog Timers	409
13.7.6 DFI Status Interface Watchdog Timers	409
13.7.7 DFI Low Power Interface Watchdog Timers	410
13.7.8 Poison Logic	411

Chapter 14

Programming the Controller	413
14.1 Initialization	414
14.1.1 Register Groups	414
14.1.2 Controller Initialization	415
14.1.3 PHY Initialization	415
14.1.4 SDRAM Initialization	415
14.1.5 LPDDR5X/5/4X Initialization Summary Tables	416
14.2 Registers	418
14.2.1 Dynamic Registers	418
14.2.2 Quasi Dynamic Registers	419
14.2.3 Interrupt Signals and Corresponding Interrupt Status Registers	422
14.3 Modes of Operation	423
14.3.1 SDRAM Selection	423
14.3.2 Low-Power Modes	423
14.4 Software Sequences.....	424
14.4.1 Changing Clock Frequencies	424
14.4.2 Software Sequence for Removal of Clocks	432
14.4.3 Power Removal Flow	433
14.4.4 Special Software Self-Refresh Sequence With Controller Initiated Retraining Followed by Burst	

PPT2	436
14.4.5 Changing RFM Level	438
14.4.6 Sequence of Burst DFI Control Update for Core VDD Change	439
14.4.7 Stopping DFI Sideband Signals During MMFW	440
Chapter 15	
Automotive	443
15.1 Overview of Automotive Safety Feature	444
15.2 Automotive Safety Package Documents	446
15.3 Automotive Specific Features	447
Appendix A	
Area, Speed, and Power	449
A.1 Timing	450
A.2 Area and Power	451
A.3 Latency Analysis	454
A.3.1 AXI - HIF - AXI Write Latency With Controller Idle	454
A.3.2 HIF - DFI - HIF Latency With Controller Idle	458
Appendix B	
Data Lane Mapping Examples	461
Appendix C	
Password Protected Features	477
Appendix D	
Prime Profiles	479
D.1 Supported Prime Profiles	480
D.2 Selecting Prime Profiles	481

Revision History

The following table provides the history of changes to this databook:

Version	Date	Description
1.60a-lca00	May 2024	<p>Added:</p> <ul style="list-style-type: none"> ■ “Area and Power Numbers” on page 34 ■ “Deliverables” on page 37 ■ “Read Data Ordering for BL32” on page 85 ■ “Write Data for the DDRCTL in BL32 Mode (MEMC_BURST_LENGTH=32)” on page 91 ■ Added note in section “Partial Writes” on page 91 ■ Table 4-5 on page 114 ■ “Mixed Packages” on page 163 ■ “Compensate DRAM Clock Stop” on page 236 ■ Added note in section “Removal of DDRC Clock Through Hardware Low-Power Interface Handshaking” on page 271 ■ “Guidance on AXI Clock Gating Using AXI Hardware Low Power Interface” on page 277 ■ Figure 11-25 on page 295 ■ “HWFFC with PHY Mission Mode Firmware” on page 303 ■ “APB Clock Gating” on page 312 ■ “LPDDR5 Post Package Repair” on page 327 ■ “Restriction on Data Mask When ECC is Used” on page 335 ■ Added note in section “Error Types” on page 360 ■ “ECCAP of Inline ECC” on page 360 ■ “SEDDED of Inline ECC” on page 362 ■ “Stopping DFI Sideband Signals During MMFW” on page 440

Version	Date	Description
1.60a-lca00	May 2024	<p><i>(Continued)</i></p> <p>Updated:</p> <ul style="list-style-type: none">■ “General Features” on page 24■ “Supported Data Rates” on page 26■ Table 1-1 on page 27■ “Unsupported Features in LPDDR4” on page 31■ “Unsupported Features in LPDDR5 and LPDDR5X” on page 32■ “Read Data and Response Channel” on page 50■ “Address Mapping” on page 65■ “Credit Mechanism” on page 79■ “Read Requests” on page 82■ “Write Data” on page 88■ “Read-Modify-Write Requests” on page 92■ “Overview of PHY Master Interface” on page 105■ Figure 5-1 on page 128■ “HBW (Half Bus Width) Mode” on page 136■ “System Address Regions Example” on page 155■ “HIF Address to SDRAM Address Mapping” on page 158■ Table 6-3 on page 160■ “Transaction Stores” on page 177■ “Sequential Operations” on page 200■ “Timing Constraints” on page 202■ “Feature Restrictions” on page 223■ Note in section “Automatic and Software Initiated ZQ Calibration Commands” on page 232■ “LPDDR4/LPDDR5 ZQ Reset Command” on page 234■ Note in section “PHY MRR Snoop Control” on page 235■ “Normal PPT2” on page 242■ “Overview of Data Bus Inversion (DBI)” on page 254■ “Entering Deep Sleep Mode” on page 265■ Figure 11-11 on page 279■ Figure 11-12 on page 280■ “Limitations of HWFFC” on page 285■ “HWFFC Procedure When HWFFCCTL.hwffc_mode=1” on page 293■ Figure 11-26 on page 296■ “DVFSC Overview” on page 300■ “Limitations” on page 301■ Figure 11-33 on page 310■ Note in section “Burst Length” on page 318

Version	Date	Description
1.60a-lca00	May 2024	<p>(Continued)</p> <p>Updated:</p> <ul style="list-style-type: none"> ■ “Unsupported Channel Modes and Supported Memory Bit Width” on page 32 ■ “Read-Modify-Write (RMW) Generation” on page 48 ■ “Write Response Channel” on page 56 ■ “Address Collision Handling” on page 193 ■ Note in section “Write Combine” on page 194 ■ “Overview of Burst Mode Operation” on page 200 ■ “Automatic Temperature Derating” on page 219 ■ “Relock and Retrain” on page 298 ■ Table 11-7 on page 303 ■ “WCK Clocking” on page 319 ■ “Link ECC Restrictions” on page 321 ■ Note in section “Enabling Inline ECC” on page 335 ■ “Features and Limitations” on page 363 ■ “Scrubber” on page 373 ■ “Write Data Parity” on page 387 ■ “On-Chip External SRAM Address Parity Poisoning” on page 406 ■ “Software Sequences” on page 424 ■ “Automotive” on page 443 ■ Table A-1 on page 451 ■ Table A-2 on page 452 <p>Removed:</p> <ul style="list-style-type: none"> ■ Section “Constraints on Refresh Burst Registers” <p>Note: Changed instances of DesignWare Cores to Synopsys IP to align with corporate naming conventions.</p>

Version	Date	Description
1.50a-lca00	April 2023	<p>Added:</p> <ul style="list-style-type: none"> ■ “DFI Error Interface” ■ Note in section “Low-Power Control Interface” ■ “Single DDRC Quad DFI Configuration” ■ “Constraints on Refresh Burst Registers” ■ “Burst DFI Control Update Request for Core VDD Change” ■ “Controller Behavior During ECC Errors” ■ “Sequence of Burst DFI Control Update for Core VDD Change” <p>Updated:</p> <ul style="list-style-type: none"> ■ “Features” ■ “Supported Standards” ■ “Registers Related to DFI Interface” ■ “DDRCTL to PHY Connections” ■ “DDRCTL Configurations” ■ “Read to Write Switching” ■ “Write to Read Switching” ■ “Controller Assisted Drift Tracking by MRR Snooping (LPDDR5)” ■ “PPT2 Limitations” ■ “Data Bus Inversion (DBI)” ■ “SDRAM Power Saving Features” ■ “Hardware Fast Frequency Change (HWFFC) Support” ■ “Area and Gate Count for Each LPDDR5X Configuration”
1.40a-lca00	October 2022	Initial release



In some instances, documentation-only updates occur. The Synopsys IP product <https://www.synopsys.com/designware-ip.html> has the latest information.

Preface

This databook describes the implementation and use of the Synopsys DDR Memory Controller (DDRCTL), which is a part of a complete LPDDR5X/5/4X interface solution.

Databook Organization

The chapters of this databook are organized as follows:

- [Chapter 1, "Product Overview"](#) provides an introduction to the DDRCTL, including features list.
- [Chapter 2, "Architecture"](#) describes the DDRCTL architecture, functional blocks, and signal naming conventions.
- [Chapter 3, "Interfaces"](#) describes the DDRCTL interfaces.
- [Chapter 4, "DFI"](#) describes the DFI interface.
- [Chapter 5, "Channel Modes"](#) describes LPDDR4 Channel Modes.
- [Chapter 6, "Address Mapping"](#) discusses DDRCTL address mapping.
- [Chapter 7, "Command Scheduling in DDRC"](#) discusses DDRCTL Port Arbitration, Queue Status Interface, Transaction Control, and Quality of Service.
- [Chapter 8, "Memory Scheduling"](#) describes Burst Mode operation and Dynamic SDRAM Constraints.
- [Chapter 9, "Periodic Memory and PHY Maintenance"](#) describes Refresh Controls, ZQ Calibration, and Controller Assisted Drift Tracking.
- [Chapter 10, "Memory Control"](#) describes Mode Register reads and writes, ODT control, and Data Bus Inversion.
- [Chapter 11, "Low-Power and Power-Saving Features"](#) describes DDRCTL low-power, power saving and Hardware Fast Frequency Change features.
- [Chapter 12, "LPDDR5 Specific Features"](#) describes LPDDR5 specific features of the DDRCTL.
- [Chapter 13, "RAS Features"](#) describes memory ECC, on-chip parity, registers parity protection, and on-chip command and address path protection features of the DDRCTL.
- [Chapter 14, "Programming the Controller"](#) provides details about programming of the DDRCTL.
- [Chapter 15, "Automotive"](#) describes the automotive features supported by the DDRCTL.
- [Appendix A, "Area, Speed, and Power"](#) provides details about the controller performance.
- [Appendix B, "Data Lane Mapping Examples"](#) provides examples of data lane mapping.

- [Appendix C, “Password Protected Features”](#) provides details about the password protected features of the DDRCTL.
- [Appendix D, “Prime Profiles”](#) explains the use of Prime Profiles for predefined configurations provided by Synopsys.

Web Resources

For more information, check the following Synopsys online resources:

- Synopsys IP product information: <https://www.synopsys.com/designware-ip.html>
- Your custom Synopsys IP page: <https://www.synopsys.com/dw/mydesignware.php>
- Documentation through SolvNetPlus: <https://solvnetplus.synopsys.com> (Synopsys password required)
- Synopsys Common Licensing (SCL): <https://www.synopsys.com/support/licensing-installation-computeplatforms/licensing.html>

Reference Documentation

- *JEDEC LPDDR4 SDRAM Specification, JESD209-4B*
- *JEDEC LPDDR4 SDRAM Specification, JESD209-4C*
- *JEDEC LPDDR4X SDRAM Specification, JESD209-4-1*
- *JEDEC LPDDR5 SDRAM Specification, JESD209-5C*
- *Synopsys LPDDR5X/5/4X PHY Utility Block (PUB) Databook*
- *DDR PHY Interface (DFI) Specification, Version 5.0, April 27, 2018*
- *Synopsys Controller IP LPDDR5X/5/4X Memory Controller Reference Manual*
- *Synopsys PHY IP LPDDR5X/5/4X PHY Databook*

JEDEC Specifications are available at <http://www.jedec.org/>.

The DDRCTL may not support all the features outlined in the standards listed above. It only supports the features described in this databook.

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

Customer Support

To obtain support for your product, prepare the required files and contact the support center using one of the methods described:

- Prepare the following debug information, if applicable:

- ❑ For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:
 - **File > Build Debug Tar-file**Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the <*core tool startup directory*>/debug.tar.gz file.
- ❑ For simulation issues outside of coreConsultant or coreAssembler:
 - Create a waveform file (such as VPD, VCD, or FSDB).
 - Identify the hierarchy path to the Design Under Test (DUT).
 - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- For the fastest response, enter a case through SolvNetPlus:
 - a. <https://solvnetplus.synopsys.com>



Note SolvNetPlus does not support Internet Explorer.

- b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).
 - c. Complete the mandatory fields that are marked with an asterisk and click **Save**.
Make sure to include the following:
 - **Product L1:** DesignWare Cores
 - **Product L2:** Memory - Controller
 - d. After creating the case, attach any debug files you created.
For more information about general usage information, refer to the following article in SolvNet-Plus:
<https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources>
- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
 - ❑ Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
 - ❑ For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood.
 - ❑ Attach any debug files you created.
 - Or, telephone your local support center:
 - ❑ North America:
Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
 - ❑ All other countries:
<https://www.synopsys.com/support/global-support-centers.html>

Product Overview

This chapter includes the following sections:

- “[General Product Description](#)” on page [20](#)
- “[Standards Compliance](#)” on page [23](#)
- “[Features](#)” on page [24](#)
- “[Area and Power Numbers](#)” on page [34](#)
- “[Clock And Reset Requirements](#)” on page [35](#)
- “[Deliverables](#)” on page [37](#)

1.1 General Product Description

The Synopsys LPDDR5X/5/4X Memory Controller (DDRCTL) combined with a Synopsys DDR PHY is a complete memory interface solution for DDR memory subsystems. The DDRCTL is delivered as configurable SystemVerilog source and is compatible with all the popular EDA environments. The DDRCTL is a flexible and advanced solution for ASIC and System-on-Chip (SoC) designers who need very low-power memory interface while achieving industry-leading high efficiency, low latency, and high performance.

The DDRCTL supports the following SDRAM types:

- LPDDR4X¹
- LPDDR5/LPDDR5X²

On the host side, there is a choice of an AMBA 4 AXI interface or a custom Host Interface (HIF). The AMBA AXI interface supports single or multi-port configurations, while the HIF supports a single port only per memory channel.

The DDRCTL can accept memory access requests from up to 16 application-side host ports. External AMBA AXI manager ports can be connected to subordinate ports of DDRCTL through the standard AMBA 4 AXI bus interfaces. The configuration registers are programmed through the AMBA 3.0/4.0 APB software interface.



Note Contact Synopsys to check whether your configuration is supported.

1.1.1 System Block Diagram

The DDRCTL and the Synopsys LPDDR5X/5/4X PHY combine to create a complete solution for connecting an SoC application bus to DDR memory devices. The combined DDRCTL and Synopsys LPDDR5X/5/4X PHY solution enables the highest DDR performance and bandwidth, with the DDRCTL initiating DDR commands to transfer data with maximum efficiency.

The DDRCTL SoC application bus interface supports a choice of the following interfaces:

- AMBA 4 AXI4
- Low-latency HIF

Flexible address mapper logic allows application-specific mapping of row, column, bank, bank group, and rank bits. The DDRCTL includes a DFI interface that supports the following standards:

- LPDDR4/LPDDR4X (JEDEC JESD209-4B/JESD209-4-1)
- LPDDR5/LPDDR5X (JEDEC JESD209-5C)

The controller is highly configurable and suitable for a wide range of system architectures. You can use the coreConsultant or coreAssembler (GUI or batch mode) tool to configure and generate the RTL source code.

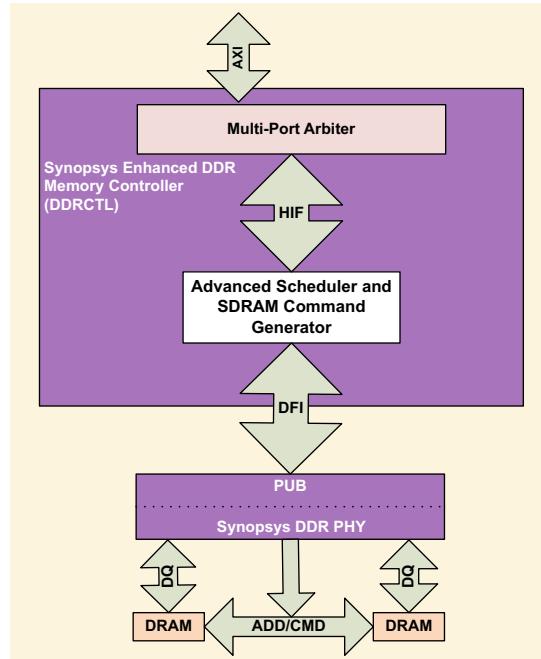
While coreConsultant is the basic tool used to create a “workspace” for a single component, coreAssembler enables you to work with a component within the context of a subsystem. A workspace is your working version of a Synopsys IP component. The coreConsultant and coreAssembler GUI interfaces enable you to run basic sanity testing using the provided System Verilog Verification environment. You can also perform

1. The term LPDDR4 when used in this databook, refers to LPDDR4X SDRAM.
2. The term LPDDR5 when used in this databook, refers to both SDRAM types LPDDR5 and LPDDR5X.

Synthesis runs using these interfaces. Full System Verilog test environment and tests are also supplied. For more information about using coreConsultant to configure, simulate, or synthesize the DDRCTL, see *Synopsys LPDDR5X/5/4X Memory Controller User Guide*.

[Figure 1-1](#) shows a system-level block diagram of the DDRCTL in a multi-port configuration.

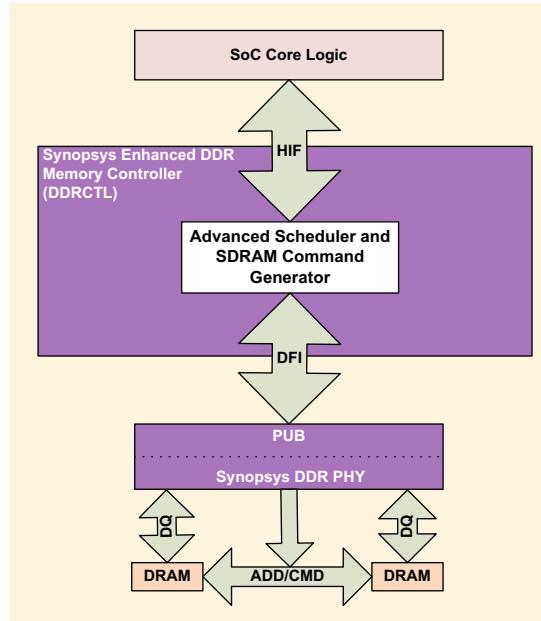
Figure 1-1 DDRCTL (Multi-Port AXI Configuration) in System-Level Diagram



The controller can be configured to use a single custom host port interface with no port arbitration. In this configuration, the DDRCTL receives read requests, write requests, and data through the single-port lowest latency HIF. Write requests are sent to the DDRCTL followed by the associated write data from the SoC. Read requests are made with a tag field. The controller returns the tag field along with read data.

Figure 1-2 shows a system-level block diagram of the DDRCTL in a single-port HIF configuration.

Figure 1-2 DDRCTL (Single-Port HIF Configuration) in System-Level Diagram



The controller receives transactions from the SoC. These transactions are queued internally and scheduled for access while satisfying the SDRAM protocol timing requirements, transaction priorities, and dependencies between the transactions. The DDRCTL in turn issues commands on the DFI interface to the PHY module, which launches and captures data to and from the SDRAM.

1.2 Standards Compliance

The DDRCTL implements the following standards:

- JEDEC LPDDR4 SDRAM Specification, JESD209-4B (unless otherwise specified)
- JEDEC LPDDR4X SDRAM Specification, JESD209-4-1
- JEDEC LPDDR5 Specification, JESD209-5C
- DDR PHY Interface DFI 5.1 Specification, March 29, 2021
- AMBA 4 AXI4 Specification from Arm



Note The DDRCTL may not support all features outlined in the standards listed above. It only supports the features described in this databook.

1.3 Features

This section describes the features of the DDRCTL controller. It consists of the following subsections:

- “General Features” on page 24
- “Power Saving and Low-Power Features” on page 25
- “Performance Features” on page 25
- “Programmable SDRAM Parameters” on page 26
- “Refresh Control Features” on page 26
- “Supported Data Rates” on page 26
- “LPDDR5-Specific Features” on page 26
- “System Bus Interface” on page 27
- “Unsupported Features and Limitations” on page 27
- “Simulators” on page 31

1.3.1 General Features

The DDRCTL controller supports the following features:

- Complete, integrated, single-vendor LPDDR4X, LPDDR5/LPDDR5X solution when combined with Synopsys LPDDR5X/5/4X PHY
- DDR PHY Interface (DFI) for easy integration with industry-standard PHYs:
 - All control, write data, and read data interface signals
 - Update interface:
 - MC-initiated requests
 - PHY-initiated requests
 - Training: PHY-independent training mode
 - Low-power interface
 - DFI PHY master interface
 - DFI Error interface
- Scalable and software-controlled 1:2/1:4 (LPDDR4) or 1:1:2/1:1:4 (LPDDR5) frequency ratio architecture
- For LPDDR4 protocol:
 - Direct software request control or programmable internal control for ZQ calibration
 - MPC (ZQCal Start/Latch) commands can be issued automatically after Self-Refresh-Powerdown exit
 - ZQ Reset MRW command can be issued through software
- Dynamic scheduling to optimize bandwidth and latency
- Read and write buffers in fully associative CAMs, configurable in powers of two, from 16 up to 64 reads and 64 writes
- Delayed writes for optimum performance on SDRAM data bus
- 1, 2, or 4 memory ranks

- Control options to avoid starvation of lower priorities
- Guaranteed coherency for write-after-read (WAR) and read-after-write (RAW) hazards (always on the HIF interface and on the AXI interface only if appropriate hardware configuration parameter and software register are set)
- Flexible address mapper logic to allow application specific mapping of row, column, bank, and rank bits
- Low-area, low-power architecture

1.3.2 Power Saving and Low-Power Features

- Automatic SDRAM power-down entry and exit caused by lack of transaction arrival for a programmable time
- Automatic Clock Stop entry and exit caused by lack of transaction arrival
- Automatic DDRCTL low-power mode operation caused by lack of transaction arrival for a programmable time through the hardware low-power interface
- Advanced power-saving design including no unnecessary toggling of command, address, and data pins (RAS/CAS/WE/BA/A hold last state after each command; DQ does not transition on writes when bytes are disabled)
- Self-refresh entry and exit:
 - Automatic self-refresh entry and exit caused by lack of transaction arrival for a programmable time
 - Self-refresh entry and exit under software control
 - Self-refresh entry and exit using dedicated DDRC hardware low-power interface control (similar to the AMBA 4 AXI protocol low-power control interface)

1.3.3 Performance Features

- For maximum SDRAM efficiency, commands executed out-of-order:
 - Read requests accompanied by a unique token (tag) from HIF
 - Read data returned with token (tag) for SoC to associate read data with correct read request
- Hardware configurable and software programmable Quality-of-Service (QoS) support:
 - Three traffic classes on read commands – high priority reads, variable priority reads, and low priority reads
 - Two traffic classes on write commands – normal priority writes and variable priority writes
 - Port urgent and port throttling control
- If QoS support is not configured in the hardware:
 - Two traffic classes on read commands – high priority reads and low priority reads
 - One traffic class on write commands – normal priority writes
- Write combine to allow multiple writes to the same address to be combined into a single write to SDRAM; supported for same starting address
- 5-clock cycle typical command latency through the DDRCTL (HIF interface):
 - Can be reduced to 4 cycles by choosing not to register DFI outputs (Configuration parameter)

- Leverages out-of-order requests with CAM to maximize throughput

1.3.4 Programmable SDRAM Parameters

- Configurable maximum SDRAM data-bus width (denoted as “full data-bus width”)
- Programmable support for all of the following SDRAM data-bus widths:
 - Full data-bus width, or
 - Half of the full data-bus width (see “[DDRCTL Configurations](#)” on page 127)
- Paging policy selectable by configuration registers as any of the following:
 - Leave pages open after accesses, or
 - Close page when there are no further accesses available in the controller for that page, or
 - Auto-precharge with each access, with an optimization for page-close mode which leaves the page open after a flush for read-write and write-read collision cases
- Explicit SDRAM mode register updates under software control

1.3.5 Refresh Control Features

- Controller-generated auto-refreshes at programmable average intervals.
- In multi-rank designs, an offset can be applied to the refresh timer for each rank to allow rank refreshes to expire at different times (this can increase efficiency by allowing traffic to continue to other ranks while a given rank is being refreshed).
- Ability to group up to eight controller-generated refreshes together to be issued consecutively (this reduces the frequency of page closings, increases overall efficiency).
Per-bank refreshes are scheduled for banks with no traffic in the CAM to minimize impact of refreshes on throughput.
- When controller-generated refreshes are grouped, some refreshes can be issued speculatively when the controller is idle for a programmable period of time.
- Ability to disable controller-generated auto-refreshes.
- Ability to issue a refresh through direct software request.
- Selectable ability to perform per-bank refreshes rather than all-banks refreshes.

1.3.6 Supported Data Rates

DDRCTL supports:

- LPDDR5 protocol up to LPDDR5-6400
- LPDDR5X protocol up to LPDDR5X-6400
- LPDDR5X protocol up to LPDDR5X-8533

1.3.7 LPDDR5-Specific Features

DDRCTL supports the following LPDDR5 features:

- Support for two bank architectures selected by mode register. The maximum data rate depends on the bank architecture.
- WCK clocking.

1.3.8 System Bus Interface

- APB interface for the DDRCTL software accessible registers
- Up to 16 host ports using AMBA AXI
- For host ports with the AXI interface:
 - Compatibility with the AMBA 4 AXI4 protocol
 - AXI burst types: Incremental and wrap
 - AXI clock asynchronous/synchronous to the controller clock
 - Exclusive access support
 - Read reorder buffer with reduced latency options (for example, bypass)

1.3.9 Unsupported Features and Limitations



Note For the most current information about unsupported features and limitations, refer to the Release Notes.

Table 1-1 lists unsupported features and limitations in this version of the LPDDR5X/5/4X Memory Controller.

Table 1-1 Unsupported and Unverified Features in Synopsys LPDDR5X/5/4X Memory Controller

Category	Known Issues and Limitations	Notes
LPDDR5	8-bank mode is not supported.	
LPDDR5	BL32 is supported only with limited configurations: <ul style="list-style-type: none"> ■ AXI configuration is not supported with MEMC_BURST_LENGTH=32. ■ Inline ECC is not supported with MEMC_BURST_LENGTH=32. 	
LPDDR5	Write X is not supported.	
LPDDR5	Data Copy is not supported.	
LPDDR5	CAS(B3) command (that is, Non-Zero Burst Start Address for Read) is not supported.	This limitation is only for HIF configuration (hif_address[3:0] must always be '0').
LPDDR5	Directed Refresh Management (DRFM) is not supported.	According to JESD209-5C, DRFM is an optional feature on LPDDR5X. However, the controller does not support it yet.
LPDDR5	JEDEC defined DRAM timing parameter (tFC) value greater than 250ns is not supported.	Newer version of JEDEC specification may define tFC_Extended to 350ns or so. It is not verified yet.
LPDDR5/4	Supported slowest data rate is 533 Mbps.	Data rates lower than 1066 Mbps is supported only with DFI 1:2 frequency ratio mode.

Category	Known Issues and Limitations	Notes
LPDDR5X	Supported fastest data rate is 9600 Mbps.	There is no JEDEC specification which supports 9600 Mbps yet.
DFI or PHY	Only Synopsys LPDDR5X/5/4X PHY is supported.	
DFI or PHY	The use of Controller Assisted Drift Tracking (DQSOSC) is not recommended for LPDDR4/4X.	Contact Synopsys for more information.
DFI or PHY	Dynamic change of DBI mode after initialization is not supported.	LPDDR54 PHY does not support it.
DFI or PHY	DBI mode in the PHY is not verified (only DFIMISC.phy_dbi_mode=0 is supported).	
DFI or PHY	Controller sets dfi_cke=1 before dfi_init_start becomes '1'. Synopsys DFI VIP sees it as an error (dfi_cke_high_active_during_initialization_check), but there is no harm as long as Synopsys PHY is used.	This is needed only when LPDDR4/4X SDRAM is used.
DFI or PHY	The following VIP error is demoted in testbench because Synopsys LPDDR5X/5/4X PHY does not support tMRW_L and tMRD_L yet: register_fail:mode_register_group:LPDDR5:mode_register_write_command_period_tmrw_check	According to JESD209-5C, if Per Pin DFE is enabled (MR41 OP[0]=1), tMRW_L and tMRD_L are required after issuing MRW command for MR24 and MR70~74.
Config/topology	Only 16-bit or 32-bit SDRAM system is supported.	
Config/topology	64-bit SDRAM system is supported only with limited configuration.	In testbench, two 32-bit PHY instances are used to verify 64-bit Single DDRC Quad DFI configuration in 2-rank system. No limitation for EnableBackDoorInitRegs in coreConsultant GUI testbench option. Four 32-bit PHY instances are used to verify 64-bit Single DDRC Quad DFI configuration in 4-rank system. EnableBackDoorInitRegs in coreConsultant GUI testbench option must be kept disabled (Default).
Config/topology	Dual channel hardware configuration is not supported (only single DDRC can exist in the controller).	See the databook for more details.
Config/topology	UMCTL2_DUAL_HIF=1 is not supported.	
Config/topology	MEMC_REG_DFI_OUT=0 is not supported.	
Config/topology	The configuration of one single PHY that has two DFI interfaces connected with two controller instances ("2 MC + 1 PHY") is supported only with HIF configuration.	Functionality (Simulation) is verified but synthesis is not supported.

Category	Known Issues and Limitations	Notes
Config/topology	CINIT CPU_DPI and STD_ALONE modes are not supported in “2 MC + 1 PHY” configuration.	Testbench does not support access to MC1 yet.
Config/topology	PPT2 feature is not supported in “2 MC + 1 PHY” configuration because it is possible that dfi_ctrlupd_req from both the controllers can be asserted simultaneously, which is not allowed in the PHY.	
Config/topology	PPT2 feature is not supported in 4-rank configuration.	
Config/topology	Only 1:4 hardware configuration is supported (TMGCFG register can choose DFI 1:2 mode or DFI 1:4 mode during reset).	
Config/topology	Quarter Bus Width (QBW) is not supported.	
Config/topology	The following combination is not supported: <ul style="list-style-type: none"> ■ UMCTL2_READ_DATA_INTERLEAVE_EN_n=1 ■ MEMC_DRAM_DATA_WIDTH=16 ■ UMCTL2_PORT_DW_n=512 	
Config/topology	The following combination is not supported: <ul style="list-style-type: none"> ■ UMCTL2_READ_DATA_INTERLEAVE_EN_n=1 ■ MEMC_DRAM_DATA_WIDTH=32 (HBW mode is used) ■ UMCTL2_PORT_DW_n=512 	
Config/topology	The following combination is not supported: <ul style="list-style-type: none"> ■ DFIUPDTMG2.ppt2_en=1 ■ DFIUPDTMG2.ppt2_override=0 ■ DDRCTL_PRODUCT_NAME = 13, 14, or 15 (products for LPDDR5X PHY) 	
Config/topology	DDRCTL_HWFFC_EXT is ‘1’ only when UMCTL2_FREQUENCY_NUM is ‘14’ and DDRCTL_PRODUCT_NAME is ‘15’.	The testbench does not support UMCTL2_FREQUENCY_NUM > 4 with LPDDR5X/5/4X PHY which requires MMFW support.
Config/topology	“Changing clock frequencies” sequences introduced in “Software Sequences” section in the databook are not supported in DDRCTL_HWFFC_EXT==1 configuration. Use HWFFC instead.	
Application Interface	CHI is not supported in LPDDR Controller.	

Category	Known Issues and Limitations	Notes
VC SpyGlass	SDC_OBJECT_NONEXIST errors are reported on multi cycle path constraints from VC SpyGlass in DESIGN_READ stage if configuration parameter DDRCTL_MCP_INCLUDE==1.	These errors does not affect the VC SpyGlass verification and can be ignored. The following command can be executed in coreConsultant CLI before running VC SpyGlass verification to prevent these errors: set_activity_parameter runSpyglass multiCyclePathConstraintsEnabled 0
VC SpyGlass	To avoid “Cannot find file” error, a name should be provided to the job using -N switch in the grid options (Example: -N vspy_job1).	
VC SpyGlass	SETUP_IGNORE_COMMAND, TCL_COMMAND_OPTION_VALUE_INVALID, and NO_OPTION_SPECIFIED errors are reported on XPI BCM66 modules from VC Spyglass if configuration parameter UMCTL2_A_SYNC_n==1 && UMCTL2_XPI_USE2RAQ_n==1 && UMCTL2_OCCAP_EN==1.	
ATPG	TMAX flow reporting an error due to its inability to understand virtual clock syntax.	You can see the issue starting with cT/2021.09-SP1. Following SolvNet article can be consulted: How TestMAX ATPG Interprets SDC File Commands
coreConsultant	In coreConsultant GUI, build debug tar file is not supported.	Contact Synopsys for more information.
coreConsultant	<p>When running Synthesis or Spyglass in coreConsultant, the following manual step is required:</p> <ol style="list-style-type: none"> In coreConsultant GUI, stop at the activity "Set the design, file, and macro prefixes". Execute the following command at the coreConsultant prompt: ::DWC_ddrctl_lpddr54::createInstallLinkForSdcCreation Proceed with the remaining coreConsultant flow as per normal. 	<p>If batch mode is used instead of GUI mode, then the command highlighted at step (2) must be added after the SetDesignAndFilePrefix and before the SpecifyConfiguration activity commands.</p> <p>Note: When using the ::DWC_ddrctl_lpddr54::createInstallLinkForSdcCreation command in coreConsultant, run time can be increased depending on the host cpu used. It is therefore recommended to use the ::DWC_ddrctl_lpddr54::createInstallLinkForSdcCreation command for Synthesis and Spyglass runs only.</p>
coreConsultant	“!” symbol in the IO.html report is expected. In case an IO signal is synchronous to more than one clock, the clock having the “!” symbol is considered as the default clock. For example, if an IO signal “abc” is synchronous to aclk, bclk!, cclk, and dclk, then bclk is considered as the primary clock.	

Category	Known Issues and Limitations	Notes
TB	Warning-[FCPSBU] Invalid values in bin Warning-[FCIBR] Invalid bin range.	To avoid these warnings, if UMCTL2_A_IDW is set to a larger value than '23', the following command needs to be executed in the command line of coreConsultant GUI or included in your configuration batch file: set_activity_parameter DWC_ddrctl_Simulate suppress_warning_noFCPSBU_plus_no FCIBR 1
Tool	PTPX is not working with the tool version listed in the installation guide.	The PrimeTime version must be downgraded to version 2022.03-SP5; if an error message for minimum tool version check pops up, use the following commands to bypass it: namespace import RCE_Public::* RCE_Public::disable_verify_tool {Verdi VCS pt_shell}
Tool	coreConsultant shows an error if FSDB output is enabled at simulation.	If an error message for minimum tool version check pops up, use the following commands to bypass it: namespace import RCEP::setRceAttrVerifyToolBypass {Verdi}
Solution Subsystem (coreAssembler)	Solution Subsystem User Guide is not available.	

1.3.9.1 Simulators

Only Synopsys VCS simulator is supported.

1.3.9.2 Unsupported Features in DFI Interface

- DFI 5.1 Interface to PHY (March 29, 2021)
 - “DFI disconnect protocol” feature
 - LPDDR54 Dual Channel

1.3.9.3 Unsupported Features in LPDDR4

- Multi-purpose command (MPC) is not supported except ZQ Calibration
- Precharge-All (PREA) command

- WR32/RD32/WFF/RFF/RDC commands
- Modified refresh (for derating)
- Self-refresh abort mode
- Density 1 Gb
- Partial array self-refresh (PASR)
- Post Package Repair (PPR)
- RFMab command (only RFMpb is supported regardless of RFSHMOD0.per_bank_refresh)

1.3.9.4 Unsupported Features in LPDDR5 and LPDDR5X

- 8B mode
- Write X
- Data Copy
- Command-based calibration mode
- Only ZQCal Latch, Start WCK2DQI Osc, Start WCK2DQO Osc are the supported Multi-Purpose Command (MPC).
- Precharge-All (PREA) command
- WR32/RD32/WFF/RFF/RDC commands
- RFMab command (only RFMpb is supported regardless of RFSHMOD0.per_bank_refresh)
- First CS toggle (PDX) after power on
- Self-refresh abort mode
- Partial array self-refresh (PASR)
- CAS-WS_FS command is issued only when WCK Always On Mode is enabled (MSTR4.wck_on=1)
- Post Package Repair (PPR)

1.3.9.5 Unsupported Channel Modes and Supported Memory Bit Width

- LPDDR Dual-Channel Mode
- Memory bit width (MEMC_DRAM_DATA_WIDTH) must be set to '16' or '32' in the LPDDR5/4/4X Controller. Memory bit width '64' can be supported only with limited configuration.

1.3.9.6 Unsupported ECC Features

- Sideband ECC

1.3.9.7 Unsupported Features in AXI Interface

- axregion and axcache signals
- Locked accesses
- FIXED transaction type (that is, only INCR and WRAP are supported)

- AxPROT signaling
- ACE (as well as ACE-lite)

1.3.9.8 Limitations

- When the register DRAMSET1TMG4.t_rcd is set to '1', the activate to column timing must be one cycle. However, the actual activate to column timing is two cycles.
- Limitation with RFSHMOD0.refresh_burst update:
 - When the RFSHMOD0.refresh_burst register is changed and the RFSHCTL0.refresh_update_level register is toggled, the DDRCTL sends the first burst of refreshes using the previous value of the register RFSHMOD0.refresh_burst. Subsequent bursts of refreshes use the new value of RFSHMOD0.refresh_burst.
 - The workaround is to toggle the RFSHCTL0.refresh_update_level register twice to avoid this issue. The value of refresh_burst is latched internally on the first toggle, and the DDR controller uses it on the second toggle.

1.4 Area and Power Numbers

For information regarding area and power numbers, see [Appendix A, “Area, Speed, and Power”](#).

1.5 Clock And Reset Requirements

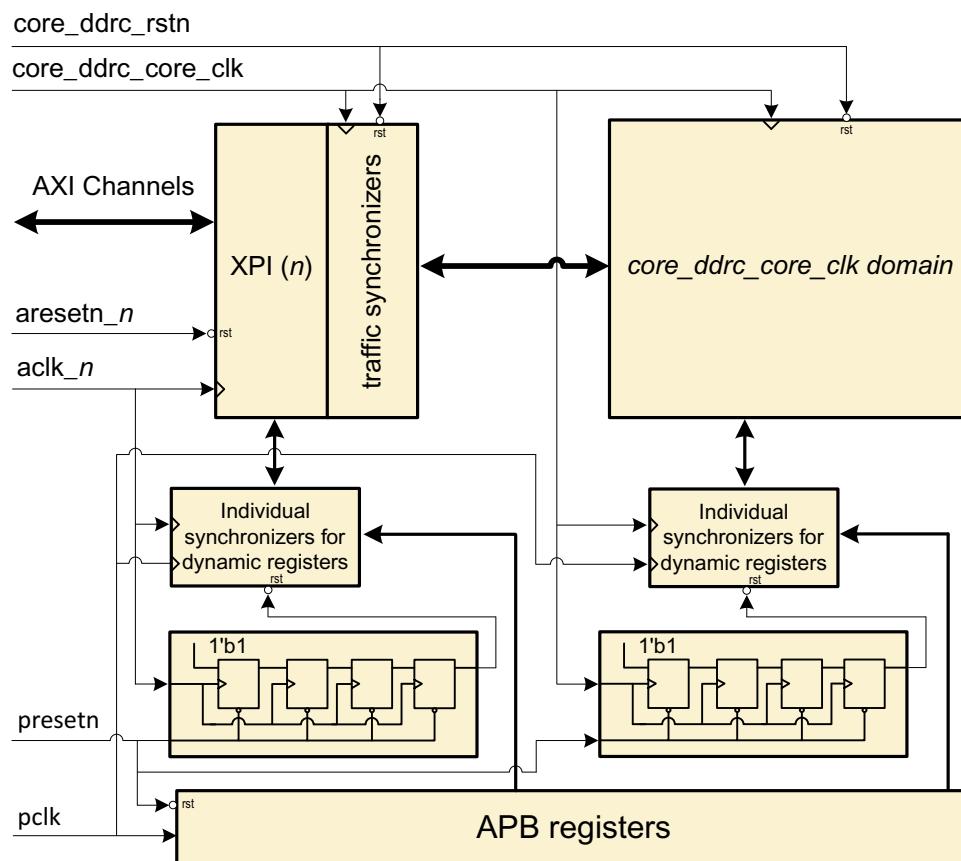
The main clock for the DDRCTL is `core_ddrc_core_clk`. The `core_ddrc_core_clk` clock is synchronous to the PHY clock. It can be driven from an external clock tree driving the DDRCTL and PHY clocks together.

All application port clocks (`aclk_n`) are independently configurable as asynchronous or synchronous with respect to the main DDRCTL clock (`core_ddrc_core_clk`). The APB interface clock (`pclk`) is normally asynchronous to the `core_ddrc_core_clk`. The interfaces that need to be asynchronously configured, incur latency and area increase due to the additional clock domain crossing circuitry.

The number of clock domains within the DDRCTL depends on the setting of the hardware parameters `UMCTL2_P_ASYNC_EN` and `UMCTL2_A_SYNC_n`.

Figure 1-3 shows a typical example with three clock domains—a `core_ddrc_core_clk` domain, a `pclk` domain, and `aclk` domain. The hardware parameter `UMCTL2_P_ASYNC_EN=1` and the hardware parameter `UMCTL2_A_SYNC_n = 0`.

Figure 1-3 Example with Three Clock Domains



An application port's clock (`aclk_n`) is considered synchronous when it is phase-aligned and has equal frequency to the DDRCTL `core_ddrc_core_clk`.



- The clock to the SBR block (`sbr_clk`) is separated from the controller clock (`core_ddrc_core_clk`), but they must be synchronous with each other.

For more information on reset and programming of the registers, see the “Clocks, Resets, and Clocking Scheme” section of the *Synopsys LPDDR5X/5/4X Memory Controller User Guide*.



Note `pclk` frequency must be equal to or less than `core_ddrc_core_clk` frequency.

The following range of ratios of clocks is verified when asynchronous support is enabled:

- `pclk`: `core_ddrc_core_clk`= 1:20 <-> 1:1 (frequency)
- `aclk_n`: `core_ddrc_core_clk`= 1: 10 <-> 9:1 (frequency)

Every input signal is assumed to be synchronous to a specific clock unless otherwise specified.

Port related signals are always synchronous to the correspondent `aclk_n`, while DDRC inputs are synchronous to `core_ddrc_core_clk`.

Exception list:

- `cactive_in_ddrc`, which is present only if no arbiter is instantiated, is asynchronous. This means that you can register the signal in the source clock domain before feeding it to DDRCTL.

Reset requirements:

- The `core_ddrc_rstn` reset must always be asserted before the `presetn` reset.
- If `pclk` is synchronous to `core_ddrc_core_clk` (`UMCTL2_P_ASYNC_EN=0`), then `core_ddrc_rstn` reset assertion must be always followed by `presetn` reset assertion. Only resetting `core_ddrc_core_clk` domain is not allowed.
- The Scrubber reset signal (`sbr_resetn`) must have the same source as `core_ddrc_rstn` (must be asserted at the same moment).
- In dual channel configurations, the `core_ddrc_rstn_dch1` reset signal must have the same source as `core_ddrc_rstn` (must be asserted at the same moment).
- In arbiter configurations (`UMCTL2_INCL_ARB==1`) the `aresetn` signal must always be asserted before `presetn`. The `aresetn` and `core_ddrc_rstn` signals must have the same source.



Note All resets are active low, so "assert" means "logic low".

1.6 Deliverables

The DDR controller is shipped with the following deliverables:

- Custom-configured Verilog RTL source code (using coreConsultant or coreAssembler)
- Synthesis, design-for-test
- Verilog and SystemVerilog, UVM test environment
- Synopsys Controller IP LPDDR5X/5/4X Memory Controller Databook
- Synopsys Controller IP LPDDR5X/5/4X Memory Controller User Guide
- Synopsys Controller IP LPDDR5X/5/4X Memory Controller Reference Manual
- Synopsys Controller IP LPDDR5X/5/4X Memory Controller Installation Guide
- Synopsys Controller IP LPDDR5X/5/4X Memory Controller Release Notes

2

Architecture

This chapter explains the functionality of the DDRCTL controller. The DDR controller consists of the following main architectural blocks:

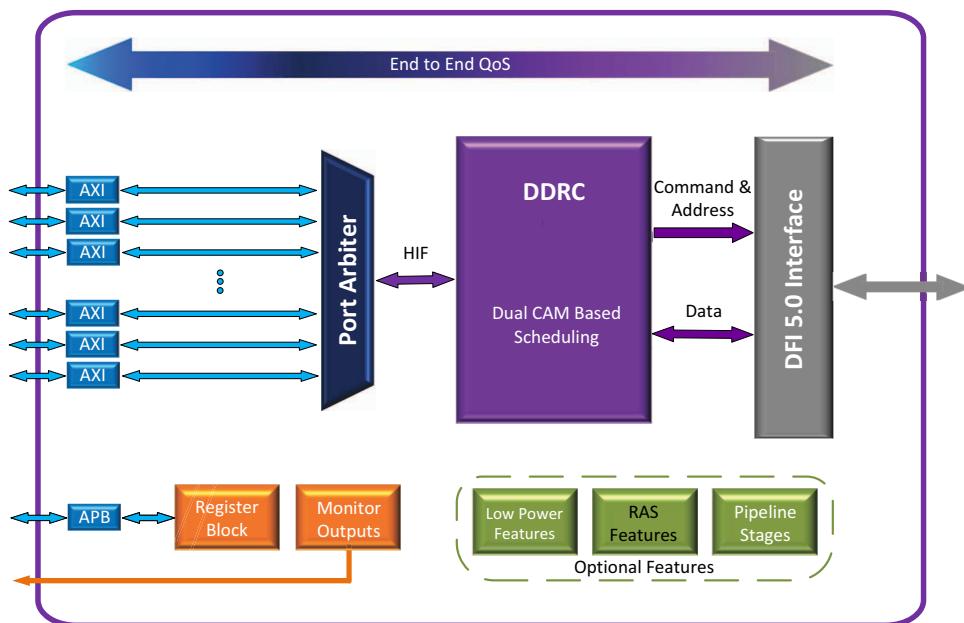
- “[Block Diagram](#)” on page [40](#)
- “[Block Descriptions](#)” on page [41](#)
- “[Functions of DDRCTL](#)” on page [43](#)
- “[Signal Naming Conventions](#)” on page [44](#)

2.1 Block Diagram

The DDRCTL controller converts system bus transactions into memory commands on the DFI interface that are compliant with the DDR protocols.

Figure 2-1 shows the DDRCTL block diagram for AXI configurations.

Figure 2-1 DDRCTL Block Diagram (AXI)



2.2 Block Descriptions

The DDR controller contains the following main architectural components:

- The AXI Port Interface (XPI) block: This block provides the interface to the application ports. It provides bus protocol handling, data buffering and reordering for read data, data bus size conversion (up-sizing or downsizing), and memory burst address alignment. Read data is stored in a SRAM, read re-order buffer and returned in order, to the AXI ports. The SRAM may be instantiated as embedded memory external to the DDRCTL or implemented as flip-flops within the DDRCTL.
- The Port Arbiter (PA) block: This block provides latency sensitive, priority based arbitration between the addresses issued by the XPIs (by the ports).
- The DDR Controller (DDRC) block: This block contains a logical CAM (Content addressable memory), which can be synthesized using standard cells. This holds information on the commands, which is used by the scheduling algorithms to optimally schedule commands to be sent to the PHY, based on priority, bank/rank status and DDR timing constraints.
- The APB Register Block: This block contains the software accessible registers.

Details on DDRC are as follows:

- Write data is stored in a SRAM (internal and external to the DDRCTL) until its associated command is issued to the PHY. The SRAM can be instantiated as embedded memory external to the DDRCTL or internal.
- Read data is handled by the response engine in the DDRC and is returned in the order of scheduled read commands on the HIF.
- ECC handling is an optional function, which is handled by logic modules within the DDRC in the write data path and in the response engine.

The frequency ratio between the DDRC clock (`core_ddrc_core_clk`) and the memory clock is controlled by register `TMGCFG.frequency_ratio`. For further information on the valid values, see TMGCFG section in *Synopsys LPDDR5X/5/4X Memory Controller Reference Manual*.

The interface between the DDRCTL controller and PHY follows the DFI 5.1 Specification.

The terms “DFI clock” and “DFI PHY clock” are used in this document in accordance with the DFI specification:

- The DFI clock is the clock on which the DFI interface runs. It is the `core_ddrc_core_clk`.
- The DFI PHY clock frequency is the same frequency as the SDRAM clock (corresponds to `MEMCLK` for certain Synopsys DDR PHYs).
 - In 1:2 frequency ratio (LPDDR4), this has twice the frequency of the DFI clock. In 1:4 frequency ratio (LPDDR4), this has four times the frequency of the DFI clock.
 - In 1:1:2/1:1:4 frequency ratio (LPDDR5), this has same frequency of the DFI clock.
 - In 1:2 frequency ratio (LPDDR4), this has twice the frequency of the DFI clock. In 1:4 frequency ratio (LPDDR4), this has four times the frequency of the DFI clock.
 - In 1:1:2/1:1:4 frequency ratio (LPDDR5), this has same frequency of the DFI clock.
- The DFI PHY data frequency is the same frequency as DFI PHY clock frequency for memory system other than LPDDR5.
 - For LPDDR5, in 1:1:2 frequency ratio, this has twice the frequency of the DFI PHY clock, in 1:1:4 frequency ratio, this has four times the frequency of the DFI PHY clock.

- The DFI PHY data frequency is the same frequency as DFI PHY clock frequency for memory system other than LPDDR5.

For LPDDR5, in 1:1:2 frequency ratio, this has twice the frequency of the DFI PHY clock, in 1:1:4 frequency ratio, this has four times the frequency of the DFI PHY clock.

2.3 Functions of DDRCTL

The DDRCTL controller performs the following functions:

- Accepts requests from the SoC with system addresses and associated data for writes.
- Performs address mapping from system addresses to SDRAM addresses (rank, bank, bank group, row).
- Prioritizes requests to minimize the latency of reads (especially high priority reads) and maximize page hits.
- SDRAM initialization can be performed for LPDDR4, but is not recommended. SDRAM initialization engine does not exist for LPDDR5.
- Ensures that all requests made to the SDRAM are legal (accounting for associated SDRAM constraints).
- Ensures that refreshes and other SDRAM and PHY maintenance requests are inserted as required.
- Controls that the SDRAM enters and exits the various power saving modes appropriately.
- Generates different interrupt signals. All the interrupt signals are level-sensitive and active-high unless otherwise stated.

For example, *_intr_fault signals are exceptions. Those interrupt signals are defined as 2-bit width, and 2'b10 means fault detected.

2.4 Signal Naming Conventions

The input and output signals of the DDRCTL controller follow the following conventions:

- AXI signals are given the names from the AXI Specification.
- APB signals are given the names from the APB Specification
- DFI signals are given the names from the DFI Specification.
- All HIF signals have the prefix “hif_”:
 - HIF commands are prefixed “hif_cmd_”.
 - HIF write data signals are prefixed “hif_wdata_”.
 - HIF read data signals are prefixed “hif_rdata_”.
 - HIF MRR data signals are prefixed “hif_mrr_data_”.
 - HIF credit signals are named “hif_*_credit”.
 - HIF signals instructing the DDRCTL to go to critical state are prefixed “hif_go2critical_”.
 - The remaining part is a description of the signal.
- The interfaces to the on-chip write data SRAM and read reorder buffer are exceptions to the HIF signal naming conventions. These interfaces use prefixes “wdataram_” and “rdataram_” for both inputs and outputs.

3

Interfaces

This chapter contains the following sections:

- “[AXI Interface](#)” on page [46](#)
- “[APB Interface](#)” on page [76](#)
- “[Host Interface \(HIF\)](#)” on page [77](#)
- “[Hardware Low-Power Interfaces](#)” on page [96](#)

3.1 AXI Interface

This section covers the following topics:

- “AXI Port Interface” on page 46
- “AXI Port Arbiter” on page 59
- “AXI Queue Status Interface and Port Throttling” on page 66
- “AXI QoS” on page 69
- “AXI Exclusive Access” on page 74

3.1.1 AXI Port Interface

This section covers the following topics:

- “Overview of AXI Port Interface” on page 46
- “Read Address Channel” on page 47
- “Write Address Channel” on page 48
- “Wrap Burst” on page 49
- “Read Data and Response Channel” on page 50
- “Write Data Channel” on page 55
- “Data Width Conversion” on page 56
- “Write Response Channel” on page 56
- “Software Coherency for AXI Ports” on page 57
- “Transaction Poisoning” on page 57
- “AXI USER Signals” on page 58
- “Signals Related To AXI Port Interface” on page 58
- “Registers Related To AXI Port Interface” on page 58



Note Unless explicitly mentioned, features described under this section apply only for AXI configurations.

3.1.1.1 Overview of AXI Port Interface

The AXI protocol is burst-based with read and write request channels that specify the host ID for the request, start byte address, burst length, burst size, and burst type. This information is processed by the interface and is used subsequently by the DDRCTL.

The AXI port interface (XPI) connects the AXI application port to the DDRCTL and performs the following main functions:

- Read address generation
- Write address generation
- Write data generation
- Read data and response generation

- Write response generation

The XPI converts AXI bursts into read and write requests, which are forwarded to the Port Arbiter (PA). In the opposite direction, the XPI converts the responses from the DDRC into appropriate AXI responses.

The interface between XPI and PA is the same as HIF (but with independent channels for read and write commands). Each AXI port can be independently configured to operate with a clock that is synchronous or asynchronous to the memory controller clock (refer to the hardware parameter “HW Configuration / Multiport Parameters”).

The AXI Specification requires that transactions must not cross a 4K address boundary. AXI compliant managers must meet this requirement. However, if a manager does not comply with this AXI protocol requirement, you can relax this restriction in the DDRCTL and set the boundary between 4K and 4G using the hardware parameter `UMCTL2_AXI_ADDR_BOUNDARY`.

3.1.1.2 Read Address Channel

The AXI read address channel has the following features:

- The read transaction can be of any length up to 256 for incremental bursts, and 16 for wrapping bursts.
- The burst types supported are incremental and wrapping.
- The burst start address can be unaligned to the AXI data width boundaries.
- The size of the burst can be less than the full width of the AXI data bus (also known as sub-sized transfers).

The signals on the read address channel are registered into the XPI in accordance with the AXI valid/ready handshaking protocol and are synchronous to the AXI clock (`ac1k`).

Read requests are accepted if the request can be written into the Read Address Queue (RAQ). This is used to store all the read address requests. If `UMCTL2_XPI_USE2RAQ_n` is set to '0', there is single address queue on a given port. If `UMCTL2_XPI_USE2RAQ_n` is set to '1', there are two address queues on a given port, named Blue and Red address queues. For more information on the usage of dual address queues, see “[Dual Read Address Queue](#)” on page [72](#). Clock domain crossing from `ac1k` to DDRCTL clock (`core_ddrc_core_clk`) is performed in the RAQ block. The depth of the RAQ can be configured to suit your system requirements (refer to the hardware configuration parameter `UMCTL2_AXI_RAQD_n` in “[HW Configuration / AXI Parameters](#)”).

After registering the read address channel by the RAQ, the following operations are performed:

- Generation of new read requests is based on alignment (derived from AXI address and size), burst lengths (derived from AXI length and memory burst length), and burst type (derived from AXI incremental or wrapping). Each AXI burst is divided into packets of length equal to the memory burst length (BL8, BL16).
- Generation of new HIF address in the case of unaligned burst and burst expansion.

A burst is aligned to the memory burst boundary when:

- $A(R|W) ADDR[2+X:X]=0$ if BL8 or
- $A(R|W) ADDR[3+X:X]=0$ if BL16.

Where X is log2 of the number of data bytes in the SDRAM interface.

Therefore:

- If `MEMC_DRAM_DATA_WIDTH=8`, $X=0$.

- If MEMC_DRAM_DATA_WIDTH=16, X=1.
- If MEMC_DRAM_DATA_WIDTH=32, X=2, and so on.

In case of reads, all the HIF read requests are always aligned to HIF burst boundary.

In general, realignment to a memory burst boundary potentially causes some data beats to be discarded (affecting bandwidth) and potentially introduces additional latency on the read data and response channel.

The arready output is defaulted to logic one and remains in logic one unless the RAQ becomes full or a WRAP burst is requested (arready is low on the next cycle after a WRAP burst is accepted). The read transaction is popped from the RAQ when it is not empty.

The XPI handles the generation of the token which is used by the DDRC for identifying the read command and corresponding data. For more information on how the DDRC handles read commands, see “[Port Timeout](#)” on page [63](#).

3.1.1.3 Write Address Channel

The AXI write address channel has the following features:

- The write transaction can be of any length up to 256 for incremental bursts, and 16 for wrapping bursts.
- The burst types supported are incremental and wrapping.
- The burst start address can be unaligned to the AXI data width boundaries.
- The size of the burst can be less than the full width of the AXI data bus (also known as sub-sized transfers).

Write Address Queue (WAQ) is used to store all the addresses for write requests from a given port. There is a single queue for all AXI IDs from a given port. In case of unaligned writes, the first HIF request can be unaligned to HIF burst boundary, but aligned to HIF beat boundary. The remaining HIF requests are aligned to HIF burst boundary.

The depth of the WAQ can be configured to suit your system requirements (see hardware configuration parameter UMCTL2_AXI_WAQD_n in “[HW Configuration / AXI Parameters](#)”). Write address and read address channels are independent and the ordering between the write and read requests may not be preserved.

To preserve the sequence, a higher-level protocol needs to wait for read/write response before sending the next transaction.

Transactions across the ports are independent and can be issued in any order.

3.1.1.3.1 Read-Modify-Write (RMW) Generation

The read-modify-write (RMW) feature is enabled when ECC is enabled in the hardware (parameter MEMC_ECC_SUPPORT is set to greater than 0) and the software (register ECCCFG0.ecc_mode is set to 3b'100), and RMW is enabled in the hardware (parameter MEMC_USE_RMW is set to ‘1’). RMW generation hardware (MEMC_USE_RMW) is always enabled in AXI configurations.

RMWs are generated for Inline ECC mode (MEMC_INLINE_ECC=1) consideration when:

- The AXI burst start or end address is unaligned with 64 bits, or
- When data strobes are used to mask data bytes falling within 64 bits.

RMWs are generated when data mask is disabled or it is not available (for example, x4 devices) when:

- There is insufficient valid write data available from the AXI transaction to write every byte of data for the memory burst (including burst chop if it is enabled)
- The AXI burst start or end address is unaligned with the DDR memory burst address boundaries

When RMW is enabled, the write command is not forwarded to the DDRC until the write data is collected and the strobes evaluated.

3.1.1.3.2 Read Modify Write (RMW) Bypass Path

XPI_RMW block exists in XPI to handle the generation of HIF Read Modify Write (RMW) commands. XPI_RMW chooses either HIF write or RMW after evaluating write data strobes, ECC enabled/disabled conditions, and register field `DBICTL.dm_en`. This block adds one cycle latency to write command and data paths by default. Read Modify Write (RMW) Bypass Path feature enables a bypass path to avoid the one cycle latency in cases where RMW prediction is not needed.

Read Modify Write (RMW) Bypass Path can be enabled by setting configuration parameter `DDRCTL_XPI_USE_RMWR=0`. Enabling `DDRCTL_XPI_USE_RMWR`, that is, setting to '1' disables the bypass path.

When the RMW Bypass Path is enabled (`DDRCTL_XPI_USE_RMWR=0`), there is a parallel bypass path to the RMW generation logic in `DWC_ddr_umct12_xpi_rmw` module. When there are no RMW generation conditions enabled, this bypass path is activated and write command and data flows through this bypass path. This saves one cycle latency for write command and data paths. The RMW generation conditions are:

- Inline/Sideband ECC is enabled in hardware and software.
- Data mask is disabled (`DBICTL.dm_en=0`).

If any of the above conditions are true, RMW generation logic cannot be bypassed and hence the bypass path is disabled.



Note Enabling the RMW Bypass Path in multi-port configurations will introduce longer timing paths.

3.1.1.4 Wrap Burst

A WRAP burst may be expanded by the XPI into multiple SDRAM transactions. This is also true for cases where wrap burst size is the same as DRAM burst size.

Table 3-1 MEMC_BURST_LENGTH 16 Wrap Expansion

Direction	Port Data Width	SDRAM Burst Length 16 && Full Burst
Read	Native-Sized	Single
	Up-Sized	Single
	Down-Sized	Single

Direction	Port Data Width	SDRAM Burst Length 16 && Full Burst
Write	Native-Sized	Single
	Up-Sized	Single (only if awaddr is aligned to the HIF data width)
	Down-Sized	Single

For information about size conversion definitions, see “[Data Width Conversion](#)” on page [56](#).

In [Table 3-1](#) full burst is when DDR_bytes=AXI_bytes.

Where DDR_bytes refers to the number of bytes transferred in a DDR burst:

MSTR0.burst_rdwr*2*MEMC_DRAM_DATA_WIDTH/8

AXI_bytes refers to the number of bytes transferred in an AXI burst:¹ (ALEN+1) * (UMCTL2_PORT_DW_n / 8)

Effective HIF Bus width depends on MSTR0.data_bus_width setting. When the current effective HIF bus width is larger than the AXI port data width its in up-sized mode. When the current effective HIF bus width is smaller than the AXI port data width its in down-sized mode. When the current effective HIF bus width is equal to AXI port data width its in native-sized mode.

3.1.1.5 Read Data and Response Channel

The XPI handles the common response interface to the DDRC to process the read data from the memory. For details of the common response interface, see “[Host Interface \(HIF\)](#)” on page [77](#).

AXI read data and response channel has a single data storage queue, which is called read data queue (RDQ) with two clocks – AXI clock for reading and DDRCTL clock for writing.

Data from different IDs are stored in the same queue and are returned in the order of read address acceptance per AXI ID.

You can configure the depth of the queue independently for each port (see hardware parameter UMCTL2_AXI_RDQD_n in “[HW Configuration / AXI Parameters](#)”).

The FSM handling the AXI handshake as well as the FIFO push and pop has the following features:

- Based on the information stored, filter the invalid beats.
- Based on the signal hif_rdata_last, generate the rlast.

The controller provides an OKAY response for each read, except for exclusive read transactions. The DDRCTL controller provides an EXOKAY response. For more information about this, see “[AXI Exclusive Access](#)” on page [74](#).

SLVERR response may be returned for read transactions (both normal and exclusive) in the following cases:

- On-chip parity address or data error (see “[On-Chip Parity \(OCPAR\)](#)” on page [386](#))
- Invalid LPDDR5/4 row address (see “[Non-binary Device Densities](#)” on page [162](#))
- Transaction has been poisoned (see “[Transaction Poisoning](#)” on page [57](#))
- Link ECC uncorrected error detected (see “[Link ECC](#)” on page [321](#)).

1. Every transfer is considered as full-sized and calculations always take into consideration the maximum size allowed. Sub-sized transfers are always expanded into multiple DRAM transactions.

- When address error occurs in Non-binary Device Densities (see “[Non-binary Device Densities](#)” on page [162](#))

Note, that SLVERR has the highest priority.



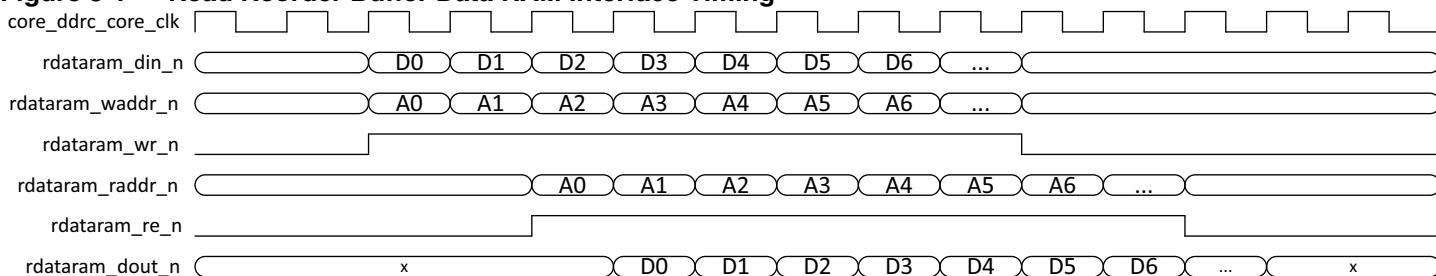
Note For sub-sized transfer, when a parity error is detected at the AXI read data interface, SLVERR is going to be asserted for all beats where the failing bytes are presented, even though they are not valid for that specific cycle. rparity information is provided at the output along with the data so that specific failing bytes can be distinguished by the AXI Manager.

3.1.1.5.1 Read Reorder Buffer

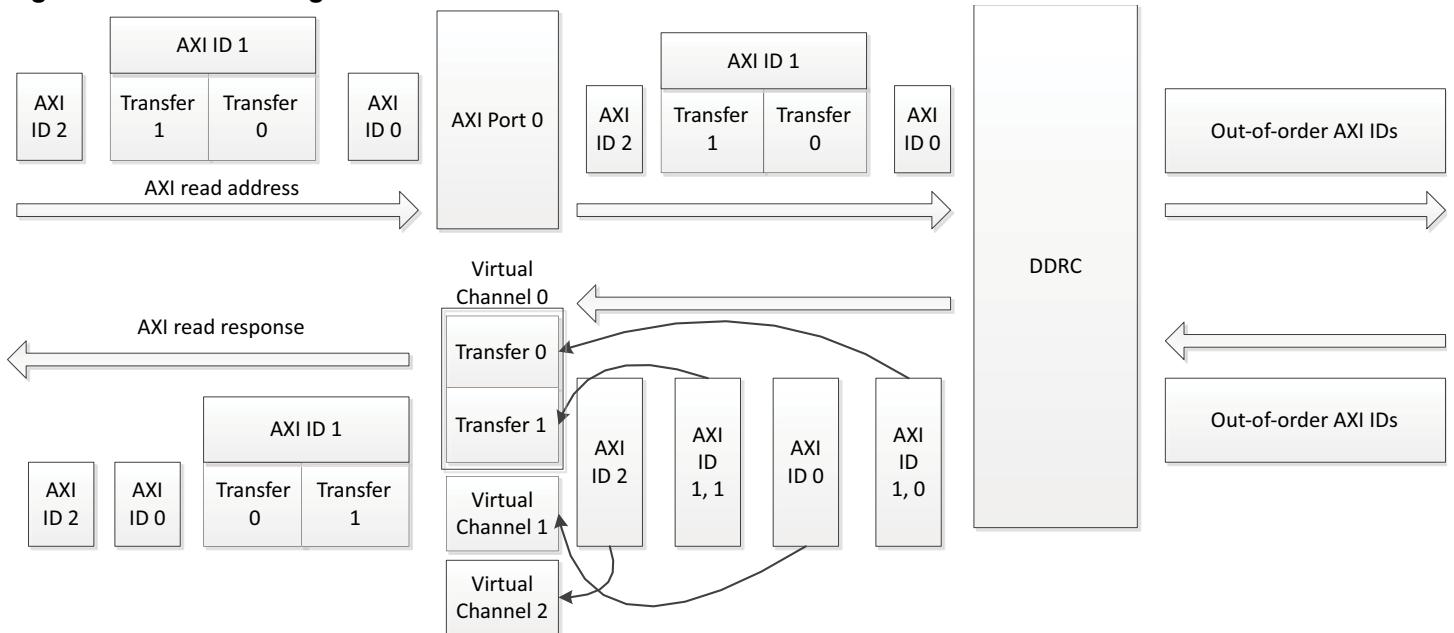
The read data can be returned from the DDRC in a different order from which the read commands are forwarded from the XPI. (due to the re-ordering of read commands in the DDRC to maximize SDRAM bandwidth). A read reorder buffer is implemented in each port to reorder the read data for that port to the same order (per AXI ID) as the order of the AXI read commands.

The read reorder buffer SRAM holds the same number of entries as the read CAM and each entry holds the read data corresponding to a DDR command. Storage for the reorder buffer can be implemented internally in the DDRCTL or implemented externally to the DDRCTL as embedded SRAM and accessed through an External RAM Interface. In either case, the control circuits for the read reorder SRAM are clocked by the DDRCTL clock (and not AXI clock). If implemented as embedded SRAM, a two-port SRAM is required.

Figure 3-1 Read Reorder Buffer Data RAM Interface Timing



The AXI protocol allows the read data for transactions of different IDs to be interleaved. To reduce potential delays where read data for one ID is blocked waiting for data associated with another ID, the read reorder buffer is organized as number of virtual channels (see [Figure 3-2](#)). The Read Reorder Virtual Channel is a mechanism to allow independent read data reordering between multiple groups of AXI IDs.

Figure 3-2 RRB Configured with Three Virtual Channels

3.1.1.5.2 AXI Read Data Interleaving

DDRCTL supports AXI read data interleaving between beats of different IDs. This feature is controlled per port by the parameter `UMCTL2_READ_DATA_INTERLEAVE_EN_$`. The following points have to be considered while enabling this feature:

- To enable read data interleaving for a port:
 - `UMCTL2_READ_DATA_INTERLEAVE_EN_$` must be set to '1' for that port.
 - `UMCTL2_NUM_VIR_CH_$` must be greater than 1 for that port.
- If enabled, read data interleaving can happen between the beats of `UMCTL2_NUM_VIR_CH_$` number of unique IDs at a time.
- Efficiency of interleaving increases as the number of virtual channels increases.

When Dual Read Address Queue is selected (`UMCTL2_XPI_USE2RAQ_n==1`) and AXI Read Data Interleaving is disabled (`UMCTL2_READ_DATA_INTERLEAVE_EN_n==0`), there is a restriction: the number of RRB virtual channels (`UMCTL2_NUM_VIR_CH_n`) must be equal to the number of CAM entries (`MEMC_NO_OF_ENTRY`). The number of virtual channels is not a configurable parameter in this case, its value defaults to `MEMC_NO_OF_ENTRY`. If AXI Read Data Interleaving is enabled, there is no such restriction.

3.1.1.5.3 RRB Threshold Based VC Selection Feature for Read Data Interleaving Disabled Configuration

When AXI read data interleaving is disabled, there are two hardware configuration options to pop-out read data from RRB:

- `UMCTL2_RRB_THRESHOLD_EN==0`

In this mode, RRB starts popping the read data when any data is available on a virtual channel (VC) of RRB.

- UMCTL2_RRB_THRESHOLD_EN==1

In this mode, RRB starts popping the read data when any of the following conditions is satisfied:

- The number of read data bursts available in the virtual channel is more than the value specified by PCFGR_n.rrb_lock_threshold.
- All read data bursts of a given AXI read transaction are available in the virtual channel.

For better performance, it is recommended to turn-on this feature for AXI read data interleaving disabled with multiple VC configurations.



Note The effective size of HIF burst becomes half and quarter in Half and Quarter Bus Width modes respectively. You need to calculate the value to be programmed into the PCFGR_n.rrb_lock_threshold register field based on this effective HIF burst size.

3.1.1.5.4 Virtual Channels: Dynamic Mapping

The number of virtual channels is determined by the hardware parameter UMCTL2_NUM_VIR_CH_n (see "HW Configuration / AXI Parameters"), which can be up to 64. By default, the hardware parameter UMCTL2_STATIC_VIR_CH_n is set to '0', so that AXI read IDs are assigned dynamically to RRB virtual channels. Dynamic mapping mode of operation is recommended since it is completely transparent and there are no software registers to program.

In this mode, the distribution of AXI read ID across virtual channels is maximized to reduce the reordering dependencies between different read IDs.

When a new AXI read ID cannot be assigned to a free virtual channel, a round-robin is used to select one of the virtual channels already in use. As a consequence, the read data for the new ID may be blocked waiting for the data associated with the other IDs in the same virtual channel. Increase the number of virtual channels to minimize this effect. To optimize the average latency and minimize unnecessary reordering, MEMC_NO_OF_ENTRY (CAM depth) number of RRB virtual channels are required, in other words, if MEMC_NO_OF_ENTRY is 64 or less, set UMCTL2_NUM_VIR_CH_n = MEMC_NO_OF_ENTRY. If the number of AXI IDs supported for a given port is less than the CAM depth, then UMCTL2_NUM_VIR_CH_n must be set to that smaller number.

The following algorithm can be used to set the number of virtual channels (UMCTL2_NUM_VIR_CH_n) in an optimum way:

```

if (number of AXI IDs <= 64) && (number of AXI IDs <= CAM depth)
    (number of Virtual channels) = (number of AXI IDs); // optimum setting
else if (CAM depth <= 64)
    (number of Virtual channels) = (CAM depth); // optimum setting
else
    (number of Virtual channels) = 64; // non-optimum setting since CAM depth = 64

```

Note that the number of virtual channels can always be set to less than the optimum settings to save area.

3.1.1.5.5 Virtual Channels: Static Mapping

The dynamic virtual channel mode can be replaced by a static virtual channel mode, when UMCTL2_STATIC_VIR_CH_n=1. In the static virtual channel mode, AXI read IDs are assigned statically to the virtual channels through registers.

When static virtual channel mode is selected, an extra bypass virtual channel is added and the maximum number of static virtual channels is restricted to 16.

The registers `PCFGIDMASKCHn.id_mask` and `PCFGIDVALUECHn.id_value` (see “`REGB_ARB_PORTp`” in “Register Descriptions” chapter of the *Synopsys LPDDR5X/5/4X Memory Controller Reference Manual*) are programmed to assign AXI transaction IDs to these virtual channels. Within a single virtual channel, data is reordered in the same sequence as the originating read commands for the IDs programmed for that particular channel.

The AXI read ID is assigned to a virtual reorder channel m for a port n if the following two conditions are true:

- $(\text{ARID} \& \text{PCFGIDMASKCHn.id_mask}) == (\text{PCFGIDVALUECHn.id_value} \& \text{PCFGIDMASKCHn.id_mask})$
- $\text{PCFGIDMASKCHn.id_mask} != 0$

In the first condition, ‘&’ is a bit wise operator. The first bullet identifies the matching ID through mask and value. The second bullet expresses the condition where a given channel must be enabled. In other words, to enable a channel, its “`id_mask`” must always be set to a non-zero value.

Table 3-2 shows example virtual channel assignments when `UMCTL2_A_NPORTS=3`, `UMCTL2_A_IDW=4`, `UMCTL2_NUM_VIR_CH_0=1`, `UMCTL2_NUM_VIR_CH_1=2`, `UMCTL2_NUM_VIR_CH_2=8`.

Table 3-2 Virtual Channel Registers Programming Example

Port	AXI IDs	Virtual Channel	id_mask Register	id_value Register
0	0	0	<code>PCFGIDMASKCH0_0=0xF</code>	<code>PCFGIDVALUECH0_0=0x0</code>
1	1,3,5,7	0	<code>PCFGIDMASKCH0_1=0x9</code>	<code>PCFGIDVALUECH0_1=0x1</code>
	0,2,4,6	1	<code>PCFGIDMASKCH1_1=0x9</code>	<code>PCFGIDVALUECH1_1=0x0</code>
2	0	0	<code>PCFGIDMASKCH0_2=0xF</code>	<code>PCFGIDVALUECH0_2=0x0</code>
	1	1	<code>PCFGIDMASKCH1_2=0xF</code>	<code>PCFGIDVALUECH1_2=0x1</code>
	2	2	<code>PCFGIDMASKCH2_2=0xF</code>	<code>PCFGIDVALUECH2_2=0x2</code>
	3	3	<code>PCFGIDMASKCH3_2=0xF</code>	<code>PCFGIDVALUECH3_2=0x3</code>
	4	4	<code>PCFGIDMASKCH4_2=0xF</code>	<code>PCFGIDVALUECH4_2=0x4</code>
	5	5	<code>PCFGIDMASKCH5_2=0xF</code>	<code>PCFGIDVALUECH5_2=0x5</code>
	6	6	<code>PCFGIDMASKCH6_2=0xF</code>	<code>PCFGIDVALUECH6_2=0x6</code>
	7	7	<code>PCFGIDMASKCH7_2=0xF</code>	<code>PCFGIDVALUECH7_2=0x7</code>

If the AXI read ID verifies the condition for more than one channel, it is always assigned to the channel with the lower channel number. For example, if a given ID matches both VC2 and VC3, it is always assigned to VC2.

If the AXI read ID does not verify the condition for any of the virtual channels, it is deemed to be ‘unassigned’.

Two modes of operation are possible for AXI read IDs that are not assigned by the register mapping. These modes are selected by the register PCFGR.read_reordered_bypass_en (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the *Synopsys LPDDR5X/5/4X Memory Controller Reference Manual*):

- If REGB_ARB_PORTp.PCFGR.read_reordered_bypass_en=0, the unassigned AXI read IDs are associated to virtual channel 0 (VC0) in addition to all the other IDs that are mapped to VC0.
- If REGB_ARB_PORTp.PCFGR.read_reordered_bypass_en=1, the unassigned AXI read IDs are associated to the bypass path and are not re-ordered.



Note The AXI read ID is assigned to a channel m only when PCFGIDMASKCHn.id_mask is not equal to '0'. After controller reset, all traffic is assigned to VC0 as PCFGIDMASKCHn.id_mask, PCFGIDVALUECHn.id_value and PCFGR.read_reordered_bypass_en are initialized to '0'.

The “bypass path” bypasses the read reorder buffer, and the read data for those transactions are not re-ordered back to the order of the originating commands. When using the bypass path, the following restrictions apply to ensure compliance with the AXI ID ordering rules:

- The AXI burst length must correspond to one command on the DDR interface (if an AXI burst is split into more than one DDR command, the data ordering for that AXI burst cannot be guaranteed and leads to a protocol violation).
- Each outstanding AXI read burst must have a unique ID. The same ID is only be reused, when the AXI Manager (rlast) accepts the last beat of the read data.
- DDR memory burst (BL) boundary crossing for INCR type bursts is not allowed. In other words, AXI start address (ARADDR) and AXI end address (ARADDR + (ALEN+1) * 2*ARSIZE - 1) must be contained within the same DDR burst (BL).

For example, if memory data width is 64 bits and BL8, AXI read must be bound within 64 bytes.

An AXI transfer starts at Start_addr=ARADDR and ends at End_addr=ARADDR + (ALEN+1) * 2*ARSIZE - 1. The burst is bound to DDR burst boundary if Start_addr[UMCTL2_A_ADDRW-1:6]==End_addr[UMCTL2_A_ADDRW-1:6].

- For AXI ports performing data size conversion (UMCTL2_PORT_DW_n!=MEMC_DRAM_DATA_WIDTH*(MEMC_FREQ_RATIO*2)), WRAP bursts must not cross the DDR BL boundary and must be address aligned with the DDR BL.
- For AXI ports not performing data size conversion (UMCTL2_PORT_DW_n==MEMC_DRAM_DATA_WIDTH*(MEMC_FREQ_RATIO*2)), WRAP bursts must be full size (ARSIZE==LOG2(AXI_DW/8)) and their length (ARLEN) must be less than or equal to MEMC_BURST_LENGTH/(MEMC_FREQ_RATIO*2)-1.

3.1.1.6 Write Data Channel

The AXI write data channel has a data storage queue, which is called write data queue (WDQ) with the following clocks:

- AXI clock (aclk_n) for writing
- DDRCTL clock (core_ddrc_core_clk) for reading

This queue is used for clock domain crossing between the AXI clock domain and the controller clock domain and acts as a registering layer in the case of a synchronous interface.

At the output of WDQ, some beats of a write data packet can be masked depending on alignment, burst size and burst length.

Write data from each port is forwarded to the DDRC, which forwards the data to a common data storage. For more information, see “[Write Requests](#)” on page [86](#).

3.1.1.7 Data Width Conversion

The XPI performs the data width conversion between the data width at the AXI interface and effective internal HIF data width. The AXI data width can be set by the hardware parameter UMCTL2_PORT_DW (see “[HW Configuration / Multiport Parameters](#)”).

Both data width down-conversion and data width up-conversion are supported.

The value of effective HIF bus width depends on the DRAM data width and MSTR0.data_bus_width setting. Based on the current effective HIF bus width, an AXI PORT can be in upsize mode, downsize mode, or native mode. When Dual Channel with data channel interleaving is used (UMCTL2_DATA_CHANNEL_INTERLEAVE_EN=1), multiply effective HIF bus width by 2 to determine Upsize/Native/Downsize mode.

Figure 3-3 PDBW Wdata Conversion

DRAM DW (bits)	FREQ_RATIO	Bus_width	Effective HIF BUS WIDTHS (bits)	AXI Data Widths (bits)				
				32	64	128	256	512
8	1:4	FBW	64	Upsize mode	Native	Downsize mode	Downsize mode	Downsize mode
16	1:4	HBW						
32	1:4	QBW						
16	1:4	FBW	128	Upsize mode	Upsize mode	Native	Downsize mode	Downsize mode
32	1:4	HBW						
64	1:4	QBW						
32	1:4	FBW	256	Upsize mode	Upsize mode	Upsize mode	Native	Downsize mode
64	1:4	HBW						
64	1:4	FBW	512	Upsize mode	Upsize mode	Upsize mode	Upsize mode	Native

The width of the queues, write data queue (WDQ) and read data queue (RDQ), are always set to the wider data width. In case of data width down-conversion, these queues are set to the width of the AXI data bus. In case of data width up-conversion, these queues are set to the width of the internal HIF data-bus width.

For data lane mapping examples, see “[Data Lane Mapping Examples](#)” on page [461](#).

3.1.1.8 Write Response Channel

The write response is generated once the last beat of write data, for a given AXI burst, is accepted by the DDRC. The write response queue can be instantiated with configurable depth which is set by the hardware parameter UMCTL2_AXI_WRQD_n (see the “[HW Configuration / AXI Parameters](#)” section).

The write response generation makes use of the result of the exclusive access monitor. For a write transaction, the response is always returned as OKAY. For an exclusive write transaction, the response can be returned as OKAY or EXOKAY. For more information about exclusive transactions, see “[AXI Exclusive Access](#)” on page [74](#).

SLVERR response may be returned in the following cases:

- Invalid LPDDR5/4 row address (see “[Non-binary Device Densities](#)” on page [162](#))
- On-chip parity address or data error (see “[On-Chip Parity \(OCPAR\)](#)” on page [386](#))

- Transaction is poisoned (see “[Transaction Poisoning](#)” on page [57](#))
- When address error occurs in Non-binary Device Densities (see “[Non-binary Device Densities](#)” on page [162](#))



- Note**
- SLVERR has the highest priority.
 - A false SLVERR may be reported if parity error is detected for invalid bytes when `OCPARCFG0.oc_parity_en=1` and `OCPARCFG0.par_rdata_slverr_en=1`. This is due to the fact that parity check is performed on the complete data bus without the knowledge of valid bytes in that specific cycle.

3.1.1.9 Software Coherency for AXI Ports

“[Address Collision Handling](#)” on page [193](#) describes how the DDRC handles in-order execution of commands to the same address.

For the commands issued at the AXI port, the logic in the DDRC protects against all types of software coherency hazards when the AXI Manager waits for write (read) response before sending the next same address read (same address write) or vice versa.

Some non-native AXI Managers expect an ordering relationship between the reads and writes when the opposite direction transactions are sent without waiting for the previous response. For these special cases, additional logic must be instantiated in the XPI to enforce the order of acceptance within the reads and writes between the AXI and HIF interfaces at the DDRC. This logic is included by setting of the hardware parameter `UMCTL2_RDWR_ORDERED_n` (see “[HW Configuration / Multiport Parameters](#)”) and enabled by the register `REGB_ARB_PORTp.PCFG.RDWR_ORDERED_EN` (see “[REGB_ARB_PORTp](#)” in “[Register Descriptions](#)” chapter of the *Synopsys LPDDR5X/5/4X Memory Controller Reference Manual*). When this feature is enabled, AXI port ensures that opposite direction transactions (a read and a write) to the same address are executed on the DFI interface in the same order as they were received on the AXI interface. Transactions to different addresses will still be executed in out of order as per the scheduling. This is ensured by passing the Read and Write commands from System interface to HIF in the same order as they are received irrespective of their addresses. DDRC ensures the order of same address transactions once it is received on the HIF interface. When this feature is enabled and read and write transactions are presented at the same cycle, the pre-arbiter gives precedence to read transactions, incrementing the write transaction order token by 1 with respect to the read transaction order token. For example, if the last order token issued is equal to ‘0’, then the read order token is equal to ‘1’ and the write order token is equal to ‘2’.

3.1.1.10 Transaction Poisoning

Sideband signals `arpoison`/`awpoison`, render an AXI transaction (read or write) invalid and the DDRCTL signals that AXI transaction poisoning has occurred. The input signal must be asserted coinciding with the associated AXI transaction. If a write is poisoned, all of its strobes are de-asserted, making the write effectively transparent to the memory. If a read is poisoned, the command is issued to the DDR memory and all the read data beats are overwritten and returned as zeros.

The DDRCTL controller may be programmed to signal that an AXI transaction poisoning has occurred by setting an interrupt or by driving the AXI SLVERR response for that transaction response. The AXI Poison Configuration Register `POISONCFG` is used to define whether an interrupt and/or AXI SLVERR is generated when an AXI transaction poisoning has occurred. There is separate control for read and write transactions. The port specific AXI Poison status register is asserted whenever an AXI transaction is

poisoned on that port. There is a separate status for read and write transactions. The interrupt and status register is cleared by the register

`POISONCFG.rd_poison_intr_clr`/`POISONCFG.wr_poison_intr_clr`. Interrupt status is stored in the AXI Poison Status register `POISONSTAT`. This register is a mirror in the APB clock domain of the actual interrupts, so status is updated with 2 or 3 `pclk` cycles of delay depending on the CDC logic.

An external block is expected to drive these new sideband signals depending on the access security requirements on transaction by transaction basis. Therefore, this feature replaces the need to implement ARM Trustzone.

3.1.1.11 AXI USER Signals

The DDRCTL provides AXI USER signals to carry user defined information. The input user data is routed in the controller along with each command, from the input to its corresponding response:

- Data provided on ARUSER is routed to RUSER.
- Data provided on AWUSER is routed to BUSER.
- WUSER is unused and must be left unconnected (it is added only for compatibility with the interface specification).

For the reads, the mechanism to carry sideband data from the address channel goes through `rxcmd_token` to the DDRC and back to the read data/response channel through `rdata_token`.

For the writes, the DDRC provides a mechanism called `wdata_ptr` to carry the sideband data from write command back to the write pointer return path which in turn is used to generate the write response to the AXI B-channel.

3.1.1.12 Signals Related To AXI Port Interface

The following are the signals related to the AXI Port Interface:

- AXI Port n Global Signals (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)
- AXI Port n Write Address Channel (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$) Signals
- AXI Port n Write Address On-Chip Parity Signals (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)
- AXI Port n Write Data Channel (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$) Signals
- AXI Port n Write Data On-Chip Parity Signals (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)
- AXI Port n Write Response Channel (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$) Signals
- AXI Port n Read Address Channel (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$) Signals
- AXI Port n Read Address On-Chip Parity Signals (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)
- AXI Port n Read Data Channel (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$) Signals
- AXI Port n Read Data On-Chip Parity Signals (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)

For more information about these signals, refer to the “Signal Descriptions” chapter.

3.1.1.13 Registers Related To AXI Port Interface

The following are the registers related to the AXI Port Interface:

- POISONCFG
- POISONSTAT

For more information about these registers, see “Register Descriptions” chapter in Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

3.1.2 AXI Port Arbiter

This section covers the following topics:

- [“Overview of Port Arbitration \(PA\)” on page 59](#)
- [“Read/Write Arbitration” on page 62](#)
- [“Port Timeout” on page 63](#)
- [“DDRC Read Priorities \(HPR/LPR/VPR\) and Write Priorities \(NPW/VPW\) for Ports” on page 64](#)
- [“Port Command Priority” on page 64](#)
- [“Round-Robin Arbitration” on page 65](#)
- [“Page Match” on page 65](#)
- [“Write Exclusive Access Lock” on page 66](#)
- [“Signals Related to Port Arbiter” on page 66](#)
- [“Registers Related to Port Arbiter” on page 66](#)

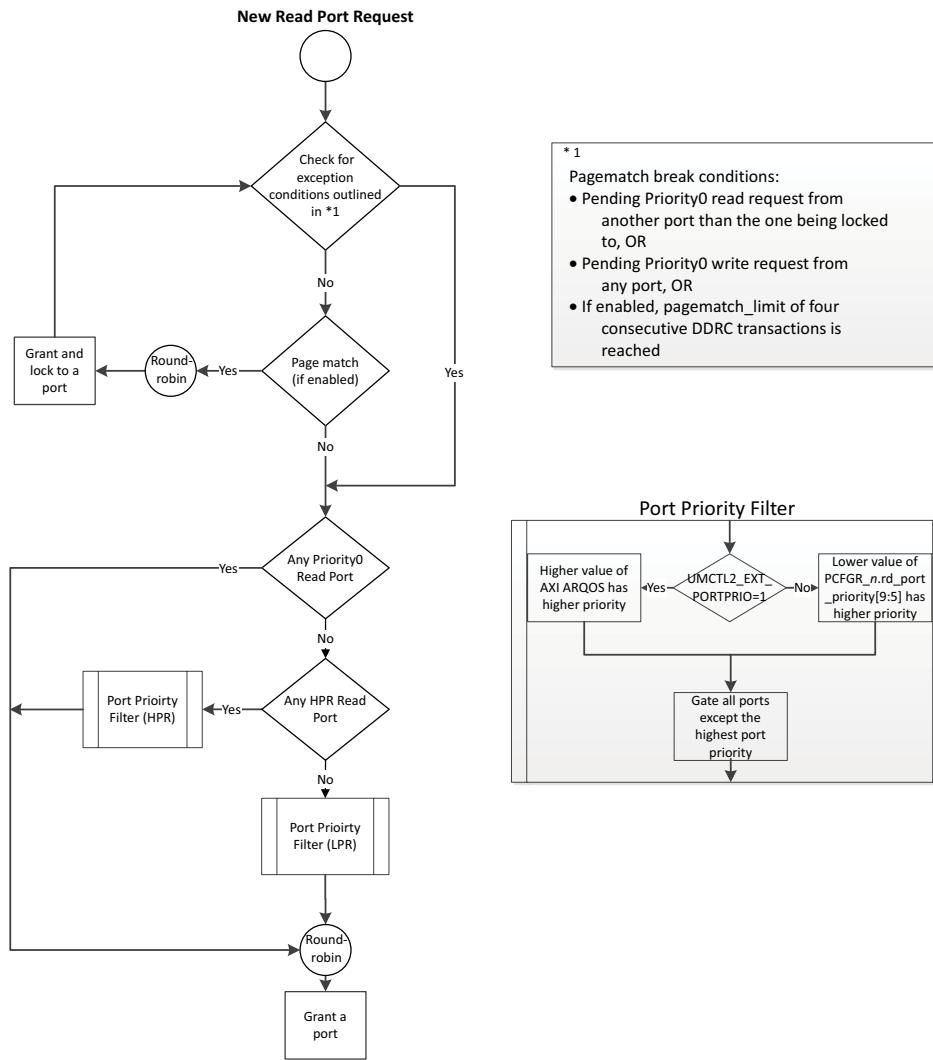
3.1.2.1 Overview of Port Arbitration (PA)

The Port Arbiter (PA) block arbitrates command requests from up to 16 AXI ports to the HIF of the DDR Controller (DDRC). PA is comprised of multiple tiers of arbitration stages which include:

- 2-priority level arbitration based on port aging and expired-VPR/expired-VPW commands (timeout - priority0)
- 2-priority level arbitration for read requests based on DDRC read priorities (HPR/LPR-VPR)
- 32-priority level arbitration based on internal port aging or 16-priority level arbitration based external AXI QoS inputs (selectable by hardware parameter)
- Round-robin arbitration to resolve ports having the same priority after passing all stages of arbitration

Figure 3-4 provides a high-level overview of how a Read port is selected.

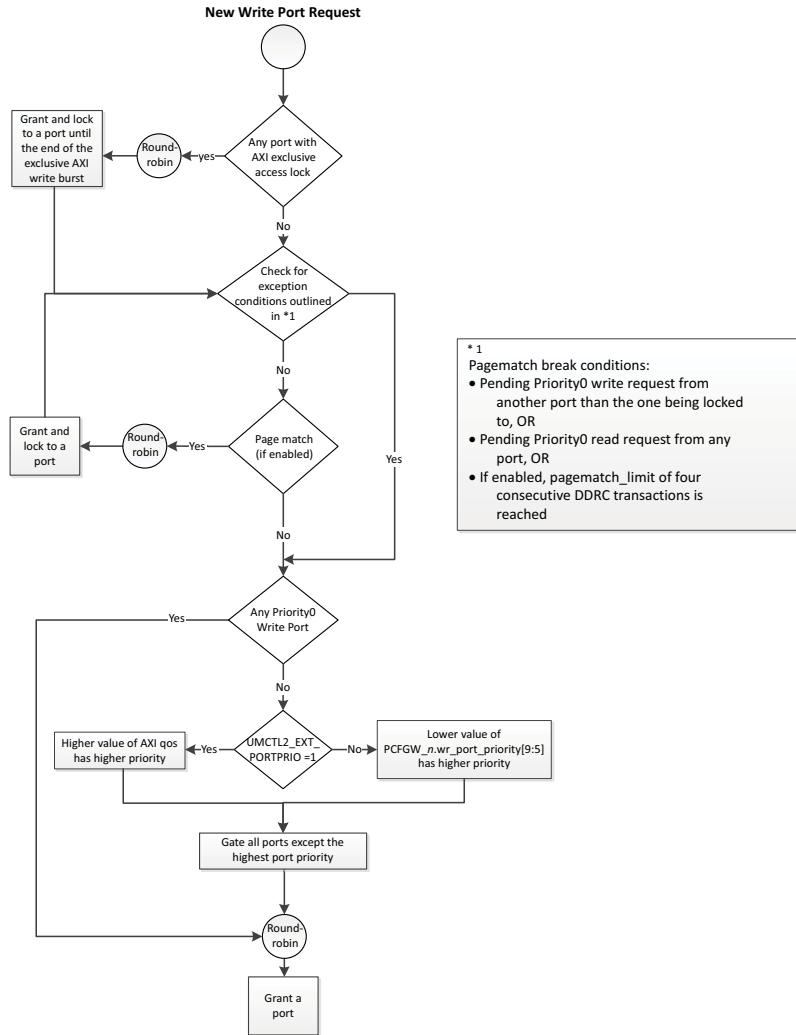
Figure 3-4 Selecting a Read Port to Grant to DDRC



- Read port priority0 is either write port is timed out by port aging or VPW is expired.
- The awurgent_n signal leads to port time out immediately and hence leads to port priority0.

Figure 3-5 provides a high-level overview of how a Write port is selected.

Figure 3-5 Selecting a Write Port to Grant to DDRC



- Write port priority0 is either write port is timed out by port aging or VPW is expired.
- The awurgent_n signal leads to port time out immediately and hence leads to port priority0.

Port aging refers to counting down the time when a port has a request, but is not granted access to the DDRC. Port timeout refers to when the port age becomes 0, which is also referred as the highest priority - 'priority0'. Expired-VPR refers to Variable Priority Read commands whose counter has expired. Read/write credits mentioned in "Read/Write Arbitration" on page 62 refers to the DDRC 'Credit Mechanism' employed at the HIF interface.

3.1.2.2 Read/Write Arbitration

This is the first level of arbitration where all requests to the ports are observed and a decision is made based on:

- Read (LPR-VPR, HPR) and write credits
- Timeout/Expired-VPR/Expired-VPW (Priority0)

The main goal of the read/write arbiter is to combine reads and writes together as long as the selected direction has available credits and the timeout has not occurred for any port of the opposite direction. If all conditions are equal, reads are prioritized over writes. Minimize direction switches to improve the memory bus efficiency.

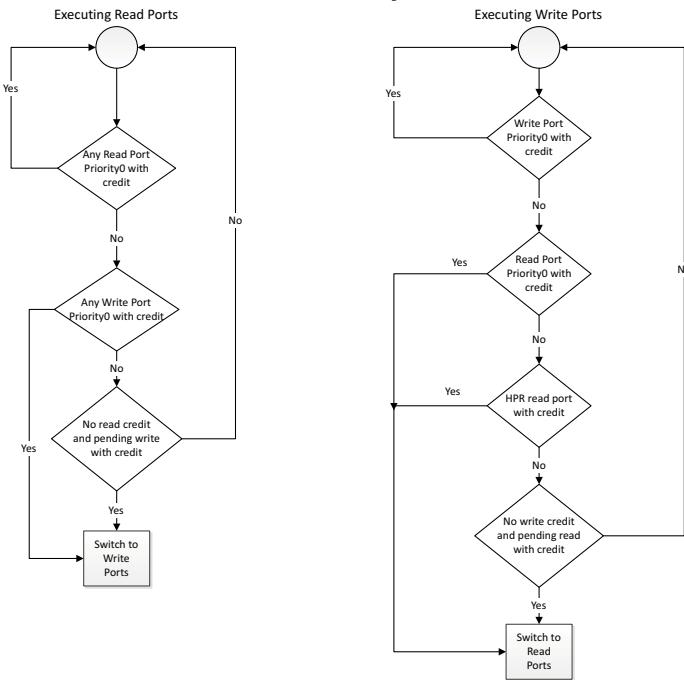
The algorithm can be summarized as:

- While executing reads:
 - Stay on the reads as long as there is a timed-out read port or expired-VPR (which are both Priority0 conditions) with available credit, else
 - Switch to writes if there is a timed-out write port or expired-VPW (Priority0) with available credit, else
 - Switch to writes when there is no read credit left and there is a pending write with available credit.
- While executing writes:
 - Stay on the writes as long as there is a timed-out write port or expired-VPW (Priority0) with available credit, else
 - Switch to reads if there is a timed-out read port or expired-VPR (which are both Priority0 conditions) with available credit, else
 - Switch to reads if there is an HPR read port with available credit, else
 - Switch to reads when there is no write credit left and there is a pending read with available credit.

AXI off-band signals, arurgent and awurgent, when asserted, cause the read/write direction to switch immediately in the Port Arbiter if enabled by `PCFGR.rd_port_urgent_en` and `PCFGW.wr_port_urgent_en` registers (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). Urgent signals also cause the `hif_go2critical_wr` / `hif_go2critical_lpr` signals to be asserted at the HIF interface, which in turn force the read/write direction switching in the DDRC, if enabled by `PCFG.go2critical_en` register.

[Figure 3-6](#) illustrates how the switching occurs between the Read and Write ports when `UMCTL2_DUAL_HIF=0`.

The PA makes sure that a grant is not given to a requester type (Write, HPR, LPR-VPR) that has no credits. The PA does not give any grants when there is a ‘stall’ indication from the DDRC.

Figure 3-6 Switching Between Read and Write Ports – Only Valid if UMCTL2_DUAL_HIF = 0

- Read port priority0 is either read port is timed out by port aging or VPR is expired.
- Write port priority0 is either write port is timed out by port aging or VPR is expired.
- arurgent_n/awurgent_n will lead to port time out immediately and hence lead to port priority0.

3.1.2.3 Port Timeout

The per port, per direction (read/write) aging counters count down the time when a port is requesting but is not granted access. When a port aging counter becomes 0, timeout condition occurs and the port becomes the highest priority requester (Priority0) to the Port Arbiter. The initial value of the counters is determined by registers PCFGR.rd_port_priority and PCFGW.wr_port_priority. The aging feature and the timeout are enabled by PCFGR.rd_port_aging_en and PCFGW.wr_port_aging_en registers. For a port with two RAQs (that is, UMCTL2_XPI_USE2RAQ_n is set for that port), an aging counter is present per RAQ, although only one port priority register is present to set the initial value of the aging counter for both queues.

Each manager can indicate to the Port Arbiter that it is going to starve soon by asserting the AXI off-band signals, arurgent and awurgent. These sideband signals can be asserted anytime and are not associated with any particular command. As long as the a[r/w]urgent signal is set, the priority of that port is the highest (Priority0). This feature is enabled by PCFGR.rd_port_urgent_en and PCFGW.wr_port_urgent_en registers.

Even if the aging feature is disabled, the a[r/w]urgent signal is functional if enabled. The system must ensure that no manager abuses the urgent signal mechanism as this could result in a significant drop of available bandwidth and/or starve other aging ports.

3.1.2.4 DDRC Read Priorities (HPR/LPR/VPR) and Write Priorities (NPW/VPW) for Ports

The read channel of a port can be set to operate as high priority reads (HPR), low priority reads (LPR), or variable priority reads (VPR). The write channel of a port can be set as normal priority writes (NPW) or variable priority writes (VPW). Priority mapping is done through the `arqos` and `awqos` input based on the `PCFGQOS0` and `PCFGWQOS0` configuration registers (see “`REGB_ARB_PORTp`” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). The PA gives higher priority to a HPR read port than to a LPR/VPR read port. The initial priorities for NPW and VPW are the same.

If any port is set to issue an HPR, the `SCHED0.lpr_num_entries` register must be programmed to a suitable value to reserve locations in the high priority queue of the read CAM.

VPR that are not expired are treated as LPR from the arbiter point of view. If a VPR transaction expires in the XPI, it has higher priority than HPR or writes.

VPW that are not expired are treated as NPW from the arbiter point of view. If a VPW transaction expires in the XPI, it has higher priority than normal writes. When multiple VPR/VPW commands expire simultaneously, the PA executes them in round-robin order to the DDRC.

Maximum timeout for VPR and VPW commands per port and per read queue are set by the `PCFGQOS1_n` and `PCFGWQOS1_n` registers (see “`REGB_ARB_PORTp`” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). VPR and VPW time-out values are loaded in the XPI and decremented at each cycle. If the counter becomes 0 before reaching the DDRC, the command is considered as expired and is referred as the highest priority - ‘priority0’. Otherwise, VPR command is treated as a LPR and VPW command, as NPW and the remaining latency are forwarded to the DDRC. For more information about this, see “[AXI QoS](#)” on page [69](#).

3.1.2.5 Port Command Priority

The next tier of arbitration policy is the multiple-level port command priorities. The priorities are per-command and can dynamically change for a given port based on AXI AxQoS signals (`arqos`/`awqos`). Alternatively, the internal port aging counters can drive priorities if the hardware parameter `UMCTL2_EXT_PORTPRIO` is disabled.

This tier of arbitration has lower- level priority than the Timeout tier. In addition, for reads, Port Priority tier has lower priority than HPR/LPR-VPR tier.

3.1.2.5.1 External Port Priorities (`UMCTL2_EXT_PORTPRIO = 1`)

If the QoS is managed end-to-end in a system, the port priorities can be adaptively determined by the average latency requirements of a particular port using closed-loop feedback. Therefore, providing external controllability to priorities allows the DDRCTL to be easily employed in these applications.

Even if port priorities are driven externally, the port timeout feature can still be used through the port aging counters to prevent starvation when required (see “[Port Timeout](#)” on page [63](#)). There are 16-level priorities set by 4-bit wide `arqos`/`awqos` signals where the higher binary value represents a higher priority.

3.1.2.5.2 Internal Port Priorities (`UMCTL2_EXT_PORTPRIO = 0`)

Another way of setting port priorities is through port aging counters. When a request is pending and not serviced, a decrementing aging counter kicks off. The starting value of this counter is derived from a 10-bit value in the port priority registers `PCFGR.rd_port_priority` and `PCFGW.wr_port_priority`. The

register resets to the start value when the request is serviced. The upper-most 5 bits of the counter determine the port priority out of 32 levels which approximates an age-based priority for each port. The lower is the value of this counter, the higher the priority.

3.1.2.6 Round-Robin Arbitration

After passing all tiers of arbitration, a tie is resolved by the final round-robin arbitration stage. The round-robin pointer starts from a port which has the lowest port index and after a grant, the pointer is moved to the first active requester after the one which has just received the grant. When page match feature is enabled, the PA locks on a port and grants to that port. The pointer of the round-robin arbiter continues to subsequent ports, but, they are not granted in that round. They are granted in the subsequent rounds.

3.1.2.7 Page Match

The Page Match feature is the ability of the Port Arbiter locking on a port when there are consecutive transactions to the same rank, bank and row (same page). The grouping increases the DDR efficiency of the controller by adaptively choosing the back-to-back DDRC transactions to be granted at a time from a given port. Along with the request, each port also sends a ‘pagematch’ signal to the Port Arbiter. The pagematch attribute of a command indicates that it is a continuation of the previous page. The Page Match feature does not introduce an additional tier of arbitration. It does not affect which port is granted first. After the grant of a port, if there is an immediate transaction with a pagematch attribute set to ‘1’, port lock condition occurs.

A lock on a write port is released if any of the following conditions occur, with the assumption that there is continuous write request with the pagematch attribute set to ‘1’:

1. Pending Priority0 (timed-out) write request from another port than the one being locked to,
2. Pending Priority0 (timed-out) read request from any port,
3. Pending HPR read request from any port,
4. If enabled, `pagematch_limit` of four consecutive DDRC transactions is reached,
5. There is no write credit available and there is read credit available.

A lock on a read port is released if any of the following conditions occurs, with the assumption that there is continuous read request with pagematch attribute set:

1. Pending Priority0 (timed-out) read request from another port than the one being locked to,
2. Pending Priority0 (timed-out) write request from any port,
3. If enabled, `pagematch_limit` of four consecutive DDRC transactions is reached,
4. There is no read credit available and there is write credit available.

The Page Match function can be enabled by the `PCFGR.rd_port_pagematch_en` and `PCFGW.wr_port_pagematch_en` registers. The same page command limit is set by the `PCCFG.pagematch_limit` register (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

3.1.2.7.1 Address Mapping

To calculate the pagematch attribute of a port, the XPI needs to know the logical to physical address mapping. Logical address is the command address of a transaction as presented on one of the ports. A physical address is the set of bank group, bank, row and column address as presented to the SDRAM

memory. The actual conversion from logical to physical address is performed by the DDRC (see ADDRMAP x registers with $x=0$ to 12). The XPI block uses these registers to determine the pagematch attribute of each DDRC command forwarded to the PA.

3.1.2.8 Write Exclusive Access Lock

An AXI write exclusive access can be comprised of more than one DDRC command. The PA locks on to a port while there are DDRC commands that belong to the same AXI write exclusive access transaction. The write exclusive access lock cannot be released by any other condition and is required for functional correctness. This type of lock occurs transparently if exclusive access monitors are enabled and if there are long write exclusive bursts.

3.1.2.9 Signals Related to Port Arbiter

For more information on signals related to the Port Arbiter, see “Signal Descriptions” chapter.

3.1.2.10 Registers Related to Port Arbiter

The following are the registers related to the PA:

- SCHED0
- SCHED1
- SCHED2

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

3.1.3 AXI Queue Status Interface and Port Throttling

This topic contains the following subsections:

- [“Overview of Queue Status Interface and Port Throttling” on page 66](#)
- [“CAM Credit Information” on page 66](#)
- [“XPI Queues Information” on page 67](#)
- [“Port Arbiter \(PA\) Port Throttling” on page 68](#)

3.1.3.1 Overview of Queue Status Interface and Port Throttling

The Information about the internal usage of resources, such as address queues and CAM slots, is provided at the output. This information can be used by an external bridge between the interconnect and the controller to manage virtual channels. For this purpose, port mask inputs are provided as well. These inputs enable you to force the Port Arbiter to grant a specific ports masking all the others.

3.1.3.2 CAM Credit Information

The controller interface provides the information about the available credits in the CAM.

The information is provided per data channel. In case a dual-channel configuration is selected, *_dch1 version of the signals are available at the output and they refer to data channel 1.

The following are the output signals:

- lpr_credit_cnt: indicates the number of available low priority read CAM slots (LPR and VPR reads share the same CAM resources and every burst consumes 1 LPR slot).
- hpr_credit_cnt: indicates the number of available high priority read CAM slots (HPR does not share resources with other traffic classes, only HPR reads consume HPR slots).
- wr_credit_cnt: indicates the number of available write CAM slots (both NPW and VPW writes share the same CAM resources and every write burst consumes 1 WR slot).

LPR/VPR and HPR resources in the CAM are separate. Total number of read CAM slots can be specified through the parameter MEMC_NO_OF_ENTRY (same as for writes). Resources can then be split between LPR/VPR and HPR by programming the register SCHED0.lpr_num_entries.

The number of CAM slots available for LPR/VPR is indicated by:

```
SCHED0.lpr_num_entries + 1
```

While, the number of slots available for HPR is indicated by:

```
MEMC_NO_OF_ENTRY - (SCHED0.lpr_num_entries + 1)
```

This means that the maximum value for the signal lpr_credit_cnt is always SCHED0.lpr_num_entries + 1.

From the CAM, any command can be chosen out-of-order based on scheduling criteria, and a credit is returned whenever a command within a CAM section is scheduled out. Therefore, whenever CAM has empty spaces, there are credits available.



Note By the definition, the maximum out-of-order depth is not limited within a CAM. It is therefore recommended to enable anti-starvation mechanisms within a CAM (page-hit limiter, max_rank_rd/max_rank_wr, and visible window limiter) to minimize worst case latency.

3.1.3.3 XPI Queues Information

The controller interface provides the information about the address queues in the XPI.

The information involves address queues for both read and write, and for reads separate information for blue/red queues in case dual queue is selected for that specific port (UMCTL2_XPI_USE2RAQ_\$=1).

Information is provided per AXI port, the suffix at the end of each signal indicates which port the signal refers to (*_\$ with \$ the port index).

Signals naming convention is as follows:

- Read address queues:
 - **raq_*** (if single queue)
 - **raqb_*/raqr_*** (blue/red queue in dual queue configuration)
- Write address queue:
 - **waq_***

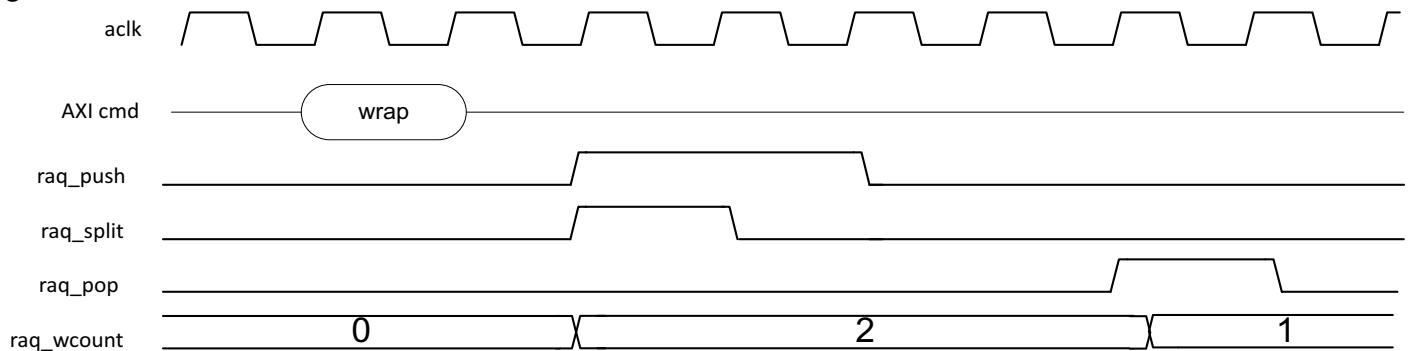
For each of them, the main information provided are:

- * _push_* (command is pushed into the queue, sync to `aclk`)
- * _split_* (first portion of a wrap pushed into the queue, sync to `aclk`)
- * _pop_* (command popped from the queue, sync to `core_ddrc_core_clk`)
- * _wcount_* (number of outstanding commands in the queue, sync to `core_ddrc_core_clk`)

In some cases, an AXI WRAP command at the input may be split into two internal INCR commands and are pushed separately into the queue. This can be recognized when split is asserted. When split and push are high, it means that the first generated INCR command has been pushed into the queue, and the following push means that the second generated INCR command has been pushed into the queue.

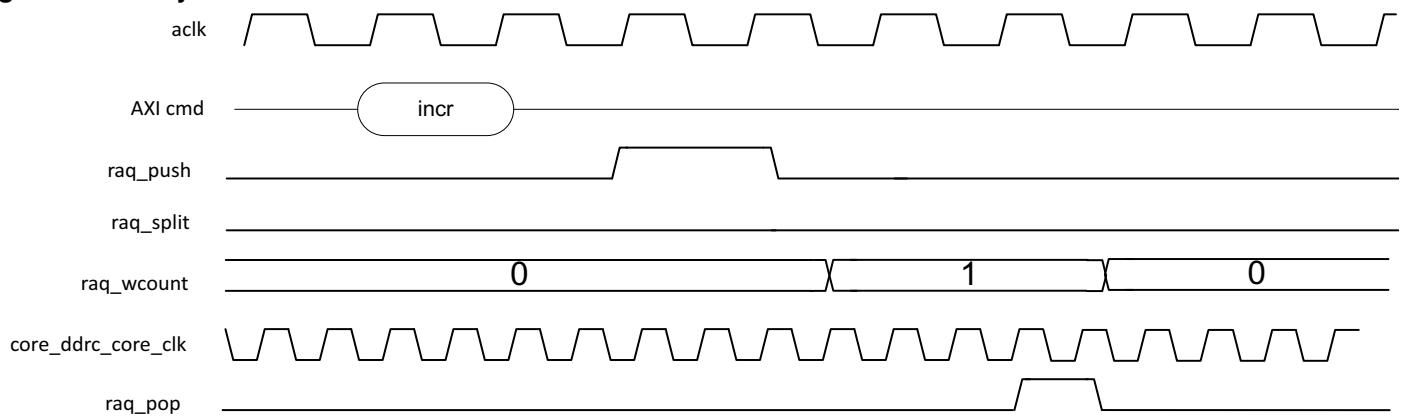
[Figure 3-7](#) shows an example of a WRAP command being pushed to the read address queue.

Figure 3-7 WRAP Pushed into the RAQ



The word count of the queue is synchronous to the destination clock domain (that is, core clock), same as the pop signal. Correct value may take some cycles before it gets propagated from the push side. [Figure 3-8](#) shows an example of an INCR command being pushed to an asynchronous queue.

Figure 3-8 Asynchronous Queue



3.1.3.4 Port Arbiter (PA) Port Throttling

Inputs are provided to control the PA throttle feature:

- `pa_rmask` (read port mask)
- `pa_wmask` (write port mask)

Width of `pa_rmask` is twice the number of ports, 2 bit per ports. First bit is for the blue queue, and the second bit is for the red queue:

- `pa_rmask[0]` -> mask blue queue port 0
- `pa_rmask[1]` -> mask red queue port 0
- `pa_rmask[2]` -> mask blue queue port 1
and so on.

If port is a single queue, the corresponding `pa_rmask` bit is unused.

Width of `pa_wmask` is equal to the number of ports, 1 bit per port.

- `pa_wmask[0]` -> mask port 0
- `pa_wmask[1]` -> mask port 1
and so on.

Asserting the corresponding bit prevents that port/queue to be granted at the PA level.

The only exception to the rule is for exclusive access. When an AXI exclusive access write is split into multiple DDR commands, once the first is granted, the exclusive access gets the PA lock until the burst is finished. No masking is possible when an exclusive access has the lock, the mask is applied once the exclusive access is finished.

3.1.4 AXI QoS

This topic contains the following sections:

- “[Overview of Quality of Service](#)” on page [69](#)
- “[Traffic Classes](#)” on page [69](#)
- “[Dual Read Address Queue](#)” on page [72](#)
- “[VPR/VPW Timeout](#)” on page [72](#)
- “[Urgent Signaling](#)” on page [74](#)
- “[Enabling QOS](#)” on page [74](#)

3.1.4.1 Overview of Quality of Service

In a SoC based sub-systems, Quality of Service (QoS) are extremely important in distributing the available resources (bandwidth or latency) amongst the different devices, in a fair mechanism to meet the overall performance requirements.

In any SoC based on-chip communication system, there are multiple Managers trying to communicate with different Subordinates through Interconnect. Interconnect is used to route the traffic between the Managers and Subordinates. There are potential chances of bandwidth sensitive Manager starving for data and latency sensitive Manager requirements are not met. This impacts the overall performance of the sub-system. QoS addresses these kinds of performance issues in sub-system.

3.1.4.2 Traffic Classes

The DDRCTL supports different traffic classes that are distinguished from each other by the `arqos` signal.

3.1.4.2.1 Read Classes

- Low Priority Read (LPR): It is also called as the best effort traffic. There is no resource allocation and it shares the resources with the other traffic types. LPR is always treated as low priority in the PA and in the DDRC. There are timeout mechanisms that can be used to prevent starvation for both the PA and the DDRC. When there is a timeout in the PA, that port becomes the highest priority (priority0). When there is a timeout in the LPR store in the DDRC (in other words, LPR store becomes critical), the LPR entries are served before HPR entries.
- Variable Priority Read (VPR): It is also called as the maximum latency bound traffic for meeting real time deadlines in video or audio applications. VPR traffic shares the same resources with LPR in the DDRC. But, based on the configuration, it may have a dedicated queue (red or blue) in the XPI. It has the same priority as LPR, but lower priority than HPR for the PA. For the DDRC, VPR has the same initial priority as LPR. Each command tagged as VPR has an associated latency timer. When expired, VPR transactions have the highest priority in the controller, both in the PA and in the DDRC. The purpose of this traffic class is to limit the maximum latency, where this latency bound is expected to be a few hundred clock cycles.
- High Priority Read (HPR): It has allocated resources. If the XPIs are configured to have dual read address queues, then the red queue can be allocated to HPR. For DDRC, HPR traffic goes to its own dedicated store. HPR traffic has higher priority than LPR. HPR is meant for latency critical but not real time applications such as CPU.

3.1.4.2.2 Write Classes

- Normal Priority Write (NPW): It is also called as the best effort traffic. There is no resource allocation and it shares the resources with the other traffic types. It is always treated as normal priority in the PA and DDRC. There are timeout mechanisms that can be used to prevent starvation for the PA. When there is a timeout in the PA, that port becomes the highest priority (priority0).
- Variable Priority Write (VPW): It is also called as the maximum latency bound traffic to meet real time deadlines in video or audio applications. VPW traffic shares the same resources with NPW in the DDRC. For the DDRC, VPW has the same initial priority as NPW. Each command tagged as VPW has an associated latency timer. When expired, VPW transactions have the highest priority in the controller, both in the PA and in the DDRC. The purpose of this traffic class is to limit the maximum latency, where this latency bound is expected to be a few hundred clock cycles.

3.1.4.2.3 QoS Mapping

The priority of an initiator (manager) is susceptible to change by the intermediate agents (for example, interconnect) before reaching the destination subordinate depending on the service levels provided with respect to the required targets. Therefore, the relationship between the IDs and their class representations must be dynamically determined.

arqos/awqos signal determines both port priorities and DDRC priorities dynamically. 16 QoS levels are divided to three regions for reads, two regions for writes. Each region can be assigned to any of the following traffic class – LPR, VPR, and HPR for reads, NPW and VPW for writes.

Region 0 and 1 are assigned to the blue address queue (see “[Dual Read Address Queue](#)” on page [72](#)). Typically, LPR/NPW and VPR/VPW may share this resource.

Region 2 is assigned to the red queue for reads (when dual queue is selected), while for writes it is assigned to the same queue as Region 0 and Region 1 (being writes always single queue). Region 2 can be mapped to HPR or VPR traffic for reads, VPW or NPW traffic for writes. In the DDRC, LPR/NPW and VPR/VPW share the same CAM store called LPR/NPW store. HPR has its own store in the DDRC.

The regions are mapped per port using PCFGQOS0 and PCFGWQOS0 registers (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

Fields for PCFGQOS0_n register are as follows:

- `r qos_map_level<x>` (where x is 1 to 2) indicates two separation levels for three regions. Possible values 0 to 14 correspond to an ar qos value. These registers indicate the upper end of the region. For example, if `r qos_map_level2` is set to 14, then Region 2 is identified as all transactions with ar qos value of 15.
 - `r qos_map_region<y>` (where y is 0 to 2) for region identifier. This register indicates the traffic class of each of the three regions. `r qos_map_region2` is present only in dual read queue configurations.

Valid values are:

- 0-LPR
 - 1-VPR
 - 2-HPR

For dual read address queue configurations, Region 0 and Region 1 map to the blue queue, and Region 2 maps to the red queue. The blue queue can only be set to LPR, VPR or both. The red queue can only be set to VPR or HPR.

For single address queue configurations, Region 0 and Region 1 map to the blue queue. Being single queue, Region 2 is not present. In this case registers `rqos_map_level12` and `rqos_map_region2` do not exist. In this case Region 0 can be set to HPR or VPR, Region 1 to VPR or LPR. Fields for `PCFGWQOS0_n` register are as follows:

- `wqos_map_level<x>` (where x is 1 to 2) indicates the separation level for three regions. Possible values 0 to 14 correspond to an awqos value. These registers indicate the upper end of the region. For example, if `wqos_map_level2` is set to 14, then Region 2 is identified as all transactions with awqos value of 15.
 - `wqos_map_region<y>` (where y is 0 to 2) for region identifier. This register indicates the traffic class of each of the two regions.

Valid values are:

- 0-NPW
 - 1-VPW

Writes are always single queue: Region 0, Region 1 and Region 2 map all to the same queue. All fields always exist. Figure 3-9 shows an example dynamic mapping from AXI QoS input to internal controller traffic classes.

Figure 3-9 Example AxQoS Mapping to Traffic Classes

AxQoS Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HPR	HPR	HPR	HPR	VPR	LPR	LPR	LPR	LPR							

Region 2

Region 1

Region 0

Level 2

Level 1

Red Queue

Blue Queue

3.1.4.3 Dual Read Address Queue

There are two separate read address queues in the XPI block – red and blue queue. They can be enabled by the parameter UMCTL2_XPI_USE2RAQ_n for each port.

Both read queues request independently to the PA; therefore if enabled, maximum configurable number of XPI ports which is 16 is reduced. Each dual-queue XPI consumes two consecutive PA ports. The worst case is that there can be 8 ports where each XPI is configured to have dual read address queue.

Red queue can be used for one traffic class only and associated to Region 2 in the QoS mapper (HPR or VPR). Blue queue can be used for two traffic classes and associated to Region 0 and Region 1 in the QoS mapper (VPR and/or LPR).

There are separate time-outs for the blue and red queues. Timers are started to down count when the transaction is accepted in the XPI, which are then forwarded to DDRC together with the command (see “[VPR/VPW Timeout](#)” on page [72](#)).

Optional retime block (register slice for pipelining) at the input of the XPI read address channel is enabled by UMCTL2_XPI_USE_INPUT_RAR. Optional retime block at the output of the XPI read address channel (enabled by UMCTL2_XPI_USE_RAR) cannot be used if dual queue is enabled.

When Dual Read Address Queue is selected (UMCTL2_XPI_USE2RAQ_n==1) and AXI Read Data Interleaving is disabled (UMCTL2_READ_DATA_INTERLEAVE_EN_n==0), there is a restriction: the number of RRB virtual channels (UMCTL2_NUM_VIR_CH_n) defaults to the number of CAM entries (MEMC_NO_OF_ENTRY). This is necessary to guarantee the UMCTL2_READ_DATA_INTERLEAVE_EN_n==0 functionality and is managed automatically. If AXI Read Data Interleaving is enabled, there is no such restriction.

3.1.4.3.1 ID Collisions

Due to the dynamic nature of the QoS with respect to the IDs, two transactions of the same ID can potentially exist in two queues. The collisions can be classified as:

- Blue after Red (BAR): A command with ID x is presented to the blue queue where a command with the same ID x is outstanding in the red queue.
- Red after Blue (RAB): A command with ID x is presented to the red queue where a command with same ID x is outstanding in the blue queue.

In the XPI, for both collision types, the ordering consistency within a given ID is preserved by stalling the incoming transaction to be pushed into the address queue it is mapped to until the colliding transaction in the opposite queue no longer exists. This is required for functional correctness.

In case of any ID collision, the DDRCTL does not speed up the drain of the stored transactions in the address queue causing collision even if the incoming transaction is HPR. In other words, address queues request arbitration to the Port Arbiter with their normal mapped priorities.

3.1.4.4 VPR/VPW Timeout

There are separate timeouts for the blue and red queues for read transactions, and one timeout for write transactions. Timers are started to down count when the transaction is accepted in the XPI, which are then forwarded to DDRC together with the command.

Timeouts are set per port and queue using PCFGQOS1_n and PCFGWQOS1_n registers (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

For PCFGQOS1_n register:

- rqos_map_timeoutb: specifies the timeout value for transactions mapped to the blue queue.
- rqos_map_timeoutr: specifies the timeout value for transactions mapped to the red queue.

For PCFGWQOS1_n register:

- wqos_map_timeout1: specifies the timeout value for write transactions for region 0 and 1.
- wqos_map_timeout2: specifies the timeout value for write transactions for region 2.

When expired, VPR/VPW transactions from XPI are tagged as expired-VPR or expired-VPW instead of LPR/NPW during normal priority transaction flow. The Port Arbiter treats these transactions with the highest priority (priority0). In addition, the Port Arbiter asserts hif_go2critical_lpr signal to the DDRC if there are no LPR credits available (LPR store of the read CAM is full) when an expired-VPR port is pending, and hif_go2critical_wr signal to the DDRC if there are no write credits available when an expired-VPW is pending. If an expired-VPW is issued as RMW at the HIF interface, the Port Arbiter asserts hif_go2critical_lpr signal to the DDRC if there are no LPR credits, and/or hif_go2critical_wr signal if there are no write credits available. If both hif_go2critical_lpr and hif_go2critical_wr are asserted at the same time, priority in the DDRC is given to the read.

If VPR/VPW timeout registers are set to ‘0’, the VPR/VPW transactions expire immediately as they enter the DDRCTL, thereby making them the highest priority transaction class within the device.

If multiple VPR/VPW transactions expire at the same time in the XPI, they are executed by the PA in round robin order.

To understand how the DDRC handles the VPR/VPW transactions, see “[Command Scheduling in DDRC](#)” on page [173](#).

3.1.4.4.1 Head of Line Blocking

Blue queue can be assigned to LPR and/or VPR if single or dual queue is selected, HPR and/or VPR if single queue is selected. Head of line blocking can occur in the XPI if more than one traffic class is mapped to the same address queue.

If a VPR expires inside the read address queue there can be two scenarios:

- Expired-VPR is blocked by one or more LPR transactions (single or dual queue configuration)
- Expired-VPR is blocked by one or more HPR transactions (only single queue configuration)

In case an LPR is blocking, that port becomes the highest priority (priority0) and if there are no LPR credits available, hif_go2critical_lpr is asserted.

In case an HPR is blocking, that port becomes the highest priority (priority0) and if there are no HPR credits available, hif_go2critical_hpr is asserted.

If a VPW expires inside the write address queue, expired-VPW may be blocked by one or more NPW or a non expired VPW transaction.

In case another transaction is blocking, that port becomes the highest priority (priority0) and if there are no write credits available, hif_go2critical_wr is asserted.

This behavior allows the queue where the VPR/VPW is expired to be flushed as quickly as possible.

This mechanism is applied only for the expired-VPR and expired-VPW commands.

For single read address queue configurations, HPR commands may be blocked by LPR or VPR commands (not-expired); in this case, the port priority is not increased.

Even in configurations with Data channel interleaving enabled

(UMCTL2_DATA_CHANNEL_INTERLEAVE_EN set to '1'), if a VPR/VPW expires within the XPI, that port becomes the highest priority - priority0. An expired VPR can be blocked by one or more LPR or HPR commands. If a VPW expires within the XPI, it can be blocked by a NPW or unexpired VPW command.

- If a VPR expires within XPI and a HPR is blocking it, that port becomes the highest priority (priority0) and if there are no HPR credits available, hif_go2critical_hpr is asserted to one or both the data channels.
- If a VPR expires within XPI and a LPR is blocking it, that port becomes the highest priority (priority0) and if there are no LPR credits available, hif_go2critical_lpr is asserted to one or both the data channels.
- If a VPW expires within the XPI and a NPW or an unexpired VPW is blocking it, that port becomes the highest priority (priority0) and if there are no write credits available, then hif_go2critical_wr is asserted to one or both the data channels. This will ensure that the queue with the expired VPR/VPW will be flushed as quickly as possible.

3.1.4.5 Urgent Signaling

AXI off-band signals per port, arurgent (arurgentb, arurgentn when UMCTL2_XPI_USE2RAQ_n is defined) and awurgent, when asserted (and as long as asserted), set a given port to the highest priority (priority0) and when applicable- cause the read/write direction to switch immediately in the Port Arbiter if enabled by PCFGR.rd_port_urgent_en and PCFGW.wr_port_urgent_en registers (see "REGB_ARB_PORTp" in "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). Urgent signals also cause the hif_go2critical_wr / hif_go2critical_lpr / hif_go2critical_hpr signals to be asserted at the HIF interface, which in turn force the read/write direction switching in the DDRC, if enabled by the PCFG.go2critical_en register. Urgent signals are ignored by the PA if there are no requesters from the asserted address channel and port. Similarly, hif_go2critical* signals are ignored by the DDRC if the corresponding store is empty. When urgent feature is enabled for reads by PCFGR.rd_port_urgent_en, arurgent_n signal causes a hif_go2critical_hpr signal to be asserted at the HIF interface if the read command at the head of the RAQ (Read Address Queue) is of HPR traffic class. If the read command at the head of the RAQ is not of HPR traffic class, then this arurgent_n signal causes a hif_go2critical_lpr signal to be asserted at the HIF interface.

3.1.4.6 Enabling QOS

The feature-specific to QOS can be configured in the coreConsultant GUI under the DDRC parameters/QOS option.

3.1.5 AXI Exclusive Access

The DDRCTL controller supports the AXI Exclusive Access feature. This feature is enabled using the hardware parameter UMCTL2_EXCL_ACCESS. This parameter defines the number of addresses the DDRCTL can monitor.

All exclusive read transactions have an EXOKAY response (except in the case of an ECC uncorrectable error). Successful exclusive write accesses have an EXOKAY response, unsuccessful exclusive write accesses return an OKAY response. If an exclusive write fails, the data mask for the exclusive write is forced low so the data is not written. Masked writes or RMW must be supported when using exclusive access. If system does not support RMW (`MEMC_USE_RMW=0`), data mask must be enabled (`DBICTL.dm_en=1`), so that the controller can properly mask failing exclusive writes.

One address range per AXI ID per port is monitored for exclusivity. Therefore, if a manager does not complete the write portion of an exclusive operation, a subsequent exclusive read to the same ID changes the address that is being monitored for exclusivity.

Once an exclusive access monitor for a given address is enabled, all write transactions are monitored for violation, regardless of the originating port. In other words, the violation check operates across the ports.

The exclusive access monitor compares the exclusive write transaction address, size, length, ID and port number against the exclusive read transaction address, size, length, ID and port number, and only accepts an exclusive write when these parameters match. Otherwise, the exclusive write is considered as fail.

If the `UMCTL2_EXCL_ACCESS` parameter is configured to support no (0) exclusive accesses, the DDRCTL behaves like an AXI subordinate that does not support exclusive accesses. Therefore, all exclusive accesses return OKAY responses.

In some cases, SLVERR response may be returned for exclusive access transactions. For more information about this, see “[Read Data and Response Channel](#)” on page 50 and “[Write Response Channel](#)” on page 56.



- The read response SLVERR is generated for an exclusive read transaction for ECC configurations, when ECC is enabled and an uncorrectable error is detected. The AXI Manager, upon receiving a SLVERR to an exclusive read command, must not complete the exclusive operation by sending an exclusive write command. If it does, the DDRCTL monitors return EXOKAY if there is no violation.
- The maximum number of bytes that are monitored as a region per address cannot exceed 128 bytes. The AXI exclusive transaction length must always be a power of two.
- When all the monitors are active, further exclusive read must not be issued. If this happens, one of the active monitors is evicted using a round robin control and new exclusive read is accepted. The selection of the monitor to be evicted is based on a pointer which rotates across all the locations when a new exclusive read is received.

3.2 APB Interface

This section contains the following sections:

- “[Overview of APB Interface](#)” on page [76](#)
- “[Register Classes](#)” on page [76](#)
- “[Signals Related to APB Interface](#)” on page [76](#)

3.2.1 Overview of APB Interface

The DDRCTL provides a dedicated APB 3.0 bus interface that is used to access the DDRCTL software programmable registers. The DDRCTL converts APB reads and writes into accesses to the internal register file. The APB data width is fixed to 32 bits for compatibility reasons with the Synopsys DDR PHYs. The APB address width is 24 bits.

All APB interface signals (p^*) are synchronous to `pclk`. `pclk` can be asynchronous to the `core_ddrc_core_clk` if the configuration parameter `UMCTL2_P_ASYNC_EN` is set. However, `pclk` frequency must be the same or less than the `core_ddrc_core_clk` frequency.

3.2.2 Register Classes

[Table 3-3](#) describes three classes of registers in the DDRCTL:

Table 3-3 Register Classes

Register Type	Description
Dynamic	Can be written at any time during operation.
Quasi Dynamic	Can be written when the controller is in reset and some specific conditions outside reset. To write them, <code>SWCTL.sw_done</code> has to be set to ‘0’ at the beginning of the programming sequence and set back to ‘1’ at the end. After that, <code>SWSTAT.sw_done_ack</code> has to be polled for acknowledge. For more information about <code>SWCTL.sw_done</code> and <code>SWSTAT.sw_done_ack</code> , refer to the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual .
Static	Can be written only when the controller is in reset.

For more information about dynamic and quasi dynamic registers, see the “Dynamic Registers” section in the [Synopsys LPDDR5X/5/4X Memory Controller Reference Manual](#).

3.2.3 Signals Related to APB Interface

For more information about signals related to the APB Interface, see “APB Subordinate Interface Signals” in the “Signal Descriptions” chapter.

3.3 Host Interface (HIF)

This topic contains the following sections:

- “[Overview of Host Interface \(HIF\)](#)” on page [77](#)
- “[Dual HIF Functionality](#)” on page [78](#)
- “[Credit Mechanism](#)” on page [79](#)
- “[Valid and Stall](#)” on page [81](#)
- “[Reads](#)” on page [82](#)
- “[Writes](#)” on page [86](#)
- “[Read-Modify-Write Requests](#)” on page [92](#)
- “[WR Versus RMW for HIF Configurations With DBICTL.dm_en=0](#)” on page [93](#)
- “[Half/Quarter Bus Width Mode](#)” on page [93](#)
- “[hif_cmd_latency](#)” on page [94](#)
- “[hif_go2critical](#)” on page [94](#)
- “[hif_cmd_autopre](#)” on page [94](#)
- “[hif_mrr_data_valid and hif_mrr_data](#)” on page [95](#)
- “[Additional Error Signal Outputs](#)” on page [95](#)

3.3.1 Overview of Host Interface (HIF)

The HIF is an internal interface in AXI configurations but is available as a HIF system interface when the hardware parameter `UMCTL2_INCL_ARB` is set to '0'. In this case there is no AXI system interfaces.

This section must be viewed when the hardware parameter `UMCTL2_INCL_ARB` is set to '0', with your logic required to interface to this HIF system interface. However, HIF signals still exist (as internal only) in designs with AXI system interfaces, and signal descriptions and functionality remain the same.

Signals related to the HIF interface are described in the following sections of the “Signal Descriptions” Chapter:

- HIF Read Command Interface Signals
- HIF Write Command Interface Signals
- HIF Command Interface Signals
- HIF Write Data Interface Signals
- HIF Read Data Interface Signals

The HIF system interface can be configured as Dual HIF to provide concurrent write (including RMW) and read interfaces, which is only enabled when `UMCTL2_DUAL_HIF` set to '1'. Those signals are defined in the following sections of Signal Descriptions” chapter (replacing “HIF Command Interface Signals”):

- HIF Read Command Interface Signals
- HIF Write Command Interface Signals

This feature converts HIF single command channel into separate/dual HIF command channels for read and write commands.

- RMW commands are performed on the Write HIF command channel.

- Read/Write port arbitration does not occur as there are separate Read and Write command channels.
- Enabling this logic improves the SDRAM utilization, depending on your traffic profile. However, it increases the overall area due to additional logic.

Following the receipt of a request and acceptance (`hif_cmd_valid==1 && hif_cmd_stall==0`) of a write or RMW request, the DDRCTL requests write data from the interface. The interface subsequently provides the write data in the requested order (matches the order in which requests are made to the DDRCTL). For more information, see “[Writes](#)” on page [86](#).

When `UMCTL2_DUAL_HIF=1`, reads are on one command bus and writes/RMWs are on another command bus. For more information, see “[Dual HIF Functionality](#)” on page [78](#).

When `UMCTL2_DUAL_HIF=0`, read, write, and RMW requests are provided through a common command bus. Both read and write requests can be reordered later, before being output on the DFI interface. Requests to the DDRCTL are throttled in two ways:

- A credit mechanism which ensures that buffer space is available for any request the SoC makes to the DDRCTL prior to the request being made. For more information, see “[Credit Mechanism](#)” on page [79](#).
- An independent stall mechanism that throttles requests when address collisions take place or if entering self-refresh through the software self-refresh or hardware low-power self-refresh is provided by the output signal `hif_cmd_stall`. HIF commands are not accepted starting from the first clock edge after the stall goes high until the clock edge on which stall is detected low. Note that `hif_cmd_stall` cannot be used instead of credit mechanism, that is, `hif_cmd_stall` will not be asserted even if CAM is full.

Data is handled separately. The DDRCTL throttles write data by potentially waiting after it receives a write or RMW request before requesting the associated data. If there is collision, then the throttling lasts until the collision is cleared in the DDRCTL. Once write data is requested, it is always accepted by the DDRCTL. However, if RMW requests are present, the write data can be throttled using the `hif_wdata_stall` signal. For more information, see “[Write Data](#)” on page [88](#).

Read data is sent to the host through a read response interface. For more information, see “[Read Response Interface](#)” on page [84](#).

When `UMCTL2_DUAL_CHANNEL` is true for dual channel configurations, there are two sets of signals with second channel signals suffixed with '_dch1'.

3.3.2 Dual HIF Functionality

Dual HIF functionality (selected when `UMCTL2_DUAL_HIF=1`) is as follows:

- Replaces the single (shared) HIF command channel with two separate HIF command channels, one for Read Commands and one for Write commands. RMW commands are performed on the Write HIF command channel.
- Dual HIF functionality continues to guarantee coherency for write-after-read (WAR) and read-after-write (RAW) hazards. Also, if a Read and Write/RMW to the same address are taken at the same time (both `hif_rcmd_stall=0` and `hif_wcmd_stall=0`), the Write/RMW is performed first.

Table 3-4 provides the details of the signals modified by the UMCTL2_DUAL_HIF parameter.

Table 3-4 Comparison of Signals Modified by UMCTL2_DUAL_HIF Parameter

UMCTL2_DUAL_HIF = 0 Single HIF Command Channel	UMCTL2_DUAL_HIF = 1 Separate HIF Command Channels One for Reads (*_rd) One for Writes/RMW (*_wr)
hif_cmd_valid	hif_rcmd_valid hif_wcmd_valid
hif_cmd_type	hif_rcmd_type hif_wcmd_type
hif_cmd_addr	hif_rcmd_addr hif_wcmd_addr
hif_cmd_pri	hif_rcmd_pri hif_wcmd_pri
hif_cmd_token	hif_rcmd_token hif_wcmd_token
hif_cmd_length	hif_rcmd_length hif_wcmd_length
hif_cmd_wdata_ptr	hif_rcmd_wdata_ptr hif_wcmd_wdata_ptr
hif_cmd_autopre	hif_rcmd_autopre hif_wcmd_autopre
hif_cmd_stall	hif_rcmd_stall hif_wcmd_stall
hif_cmd_latency	hif_rcmd_latency hif_wcmd_latency



The databook uses UMCTL2_DUAL_HIF=0 notation.

If you are using UMCTL2_DUAL_HIF=1, the internal signals must be interpreted in the following way:

- hif_cmd_valid refers to hif_rcmd_valid and/or hif_wcmd_valid.
- hif_cmd_stall refers to hif_rcmd_stall and/or hif_wcmd_stall, and so on.

3.3.3 Credit Mechanism

The DDRCTL employs a credit mechanism to control the request and data flow thus ensuring that buffers do not overflow. The interface making the request to the DDRCTL, can only request commands for which it has been granted “credits”.

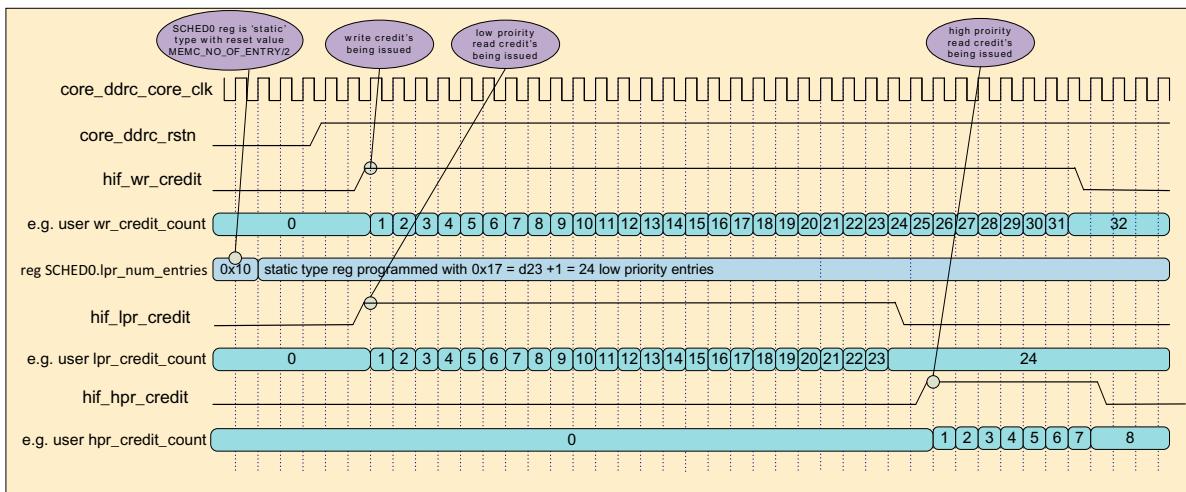
Credits are tracked separately for the following three command types:

- High Priority Read (HPR)

- Low Priority Read/Variable Priority Read/High Priority Read in LPR CAM (LPR/VPR/HPR)
- Write

Credits are counted for each command type independently according to the following rules:

- Initially the interface has zero credits.
- The interface logic must include counters to track the number of credits granted by the DDRCTL and decremented due to commands issued by the interface logic to the DDRCTL. The interface logic must have separate credit counters for each individual command type. The maximum number of LPR credits is configured by `SCHED0.lpr_num_entries` register out of the total number of entries (depth of CAM) defined by the parameter `MEMC_NO_OF_ENTRY`. The number of CAM slots available for LPR/VPR is indicated by `SCHED0.lpr_num_entries + 1`. While the number of slots available for HPR is indicated by:
 - `MEMC_NO_OF_ENTRY - (SCHED0.lpr_num_entries + 1)`.
 - This means that the maximum value for the signal `lpr_credit_cnt` is always `SCHED0.lpr_num_entries + 1`.
- Following the de-assertion of reset to the DDRCTL, credits are issued to the interface for each command type. A given credit count increments every time the DDRCTL issues a credit, indicated by the assertion of the appropriate `*_credit` signal on the rising edge of `core_ddrc_core_clk`. The credit counting logic implemented externally to the DDRCTL must run on the same clock.
- When the credit count is greater than zero, the interface can issue requests of that type to the DDRCTL. Each time a request is issued to the DDRCTL, the associated credit count is decremented. An RMW command decrements both the LPR and write credit counters by one.
- When `UMCTL2_VPRW_EN` is set to '1', the arbiter or the host can send Variable Priority Read (VPR) and Variable Priority Write (VPW) commands.
- VPR commands do not have a pre-allocated storage resource in the Read CAM (unlike HPR commands). VPR commands share the same resource with the LPR commands. As far as the credit mechanism is concerned, the VPR commands are counted in the LPR credit bucket.
- Similarly, VPW commands do not have a pre-allocated storage resource in the Write CAM. VPW commands share the same resource with the Normal Write commands. As far as credit mechanism is concerned, the VPW commands are counted in the WR credit bucket.
- For reads, the `*pr_credit` is issued when that request has been sent on the DFI interface.
- For writes, the `*pw_credit` is issued when that request and data has been sent on the DFI interface.

Figure 3-10 Issued Credits Post Reset Negation

Note

Ensure that if the DDRCTL is reset via `core_ddrc_rstn`, the customer logic credit counters are also zero. If this is not ensured, then the customer logic credit counts may be misaligned with the available credits causing lost data/fatal errors.

3.3.4 Valid and Stall

A request is indicated by the assertion of `hif_cmd_valid` but is only accepted when `hif_cmd_stall==0`. The DDRCTL can throttle the requests by asserting `hif_cmd_stall`. For any cycle in which `hif_cmd_stall` is asserted on the rising edge of the clock, the `hif_cmd_valid` and all other associated request signals are ignored by the DDRCTL. When this happens, the interface must not yet decrement the associated credit counter. When the DDRCTL de-asserts `hif_cmd_stall` on the rising edge of the clock, the request is accepted and the interface can then de-assert `hif_cmd_valid`, decrement the requester credit counter and issue the next read, write or RMW request (credits allowing).

The DDRCTL asserts `hif_cmd_stall` in any of the following conditions:

1. Software driven self-refresh entry requests. This is to ensure that DDRCTL is empty when memories are in self-refresh. For more information, see “[Low-Power and Power-Saving Features](#)” on page 257.
2. Successful DDRCTL hardware low-power entry requests to self-refresh. This is to ensure that the DDRCTL is empty when memories are in self-refresh. For more information, see “[Low-Power and Power-Saving Features](#)” on page 257.
3. Address collisions in the DDRCTL, where flow control is applied to prevent further commands from entering the DDRCTL. For more information, see “[Address Collision Handling](#)” on page 193.
4. In case when Write CAM is full and no corresponding data for any of the write commands has arrived.
5. When setting `OPCTRL1.dis_hif=1`, which causes `hif_cmd_stall` to be asserted.

Note

If Dual HIF functionality is enabled (`UMCTL2_DUAL_HIF=1`), `hif_cmd_stall/hif_wcmd_stall` is asserted as follows:

- Conditions (1), (2), (3), and (5) cause both `hif_rcmd_stall` and `hif_wcmd_stall` to assert at the same time.
- Condition (4) only affects `hif_wcmd_stall`.

3.3.5 Reads

3.3.5.1 Read Requests

When the LPR or HPR credit count is greater than zero, the customer logic can issue read requests to the DDRCTL accordingly via this HIF system interface.

A command is issued to the DDRCTL by asserting `hif_cmd_valid` on the rising edge of the clock. All read request fields must be driven to the appropriate values at the same time. These fields are:

- `hif_cmd_type`: specifies the request type as follows:
 - '00' indicates a write request.
 - '01' indicates a read request.
 - '10' indicates an RMW request.
 - '11' is not supported.
- `hif_cmd_pri`:
 - In HIF-only configurations (==0) where `UMCTL2_VPRW_EN` is set to '0', this signal is 1-bit wide. A '1' indicates the read request is high priority. '0' indicates the read request is low priority. This field is meaningless for write commands.
 - In configurations where `UMCTL2_VPRW_EN` is set to '1', this signal is 2-bit wide.

The encoding for Read is:

- 2'b00 - LPR (Low Priority Read)
- 2'b01 - VPR (Variable Priority Read)
- 2'b10 - HPR (High Priority Read)
- 2'b11 - HPRL (High Priority Read in LPR CAM)

VPR commands are sent only in configurations with `UMCTL2_VPRW_EN=1`.

HPRL commands are supported only in configurations with `DDRCTL_SYS_INTF=0` and `UMCTL2_VPRW_EN=1`.

- `hif_cmd_addr`: indicates the address for the read. For more information, see "[HIF Address to SDRAM Address Mapping](#)" on page [158](#).
- `hif_cmd_length` is 2 bits.

Note the following:

- For `MEMC_BURST_LENGTH=16`, a normal read is 16 SDRAM words.
If `MEMC_BURST_LENGTH=16`:
 - 2'b10 - Partial (Half) Read
 - 2'b00 - Full Read
- For `MEMC_BURST_LENGTH=32`, a normal read is 32 SDRAM words.
If `MEMC_BURST_LENGTH=32`:
 - 2'b11 - Partial (Quarter) Read
 - 2'b10 - Partial (Half) Read

- 2'b00 – Full Read

In each case, a partial (half) read is half the length of a normal full read (where a word is the amount of data transferred on one edge of the DQS to the SDRAM in a fully configured system).

If `MEMC_BURST_LENGTH=32`, a partial (quarter) read is possible and is quarter the length of a normal full read. This field is meaningless for write or RMW commands.

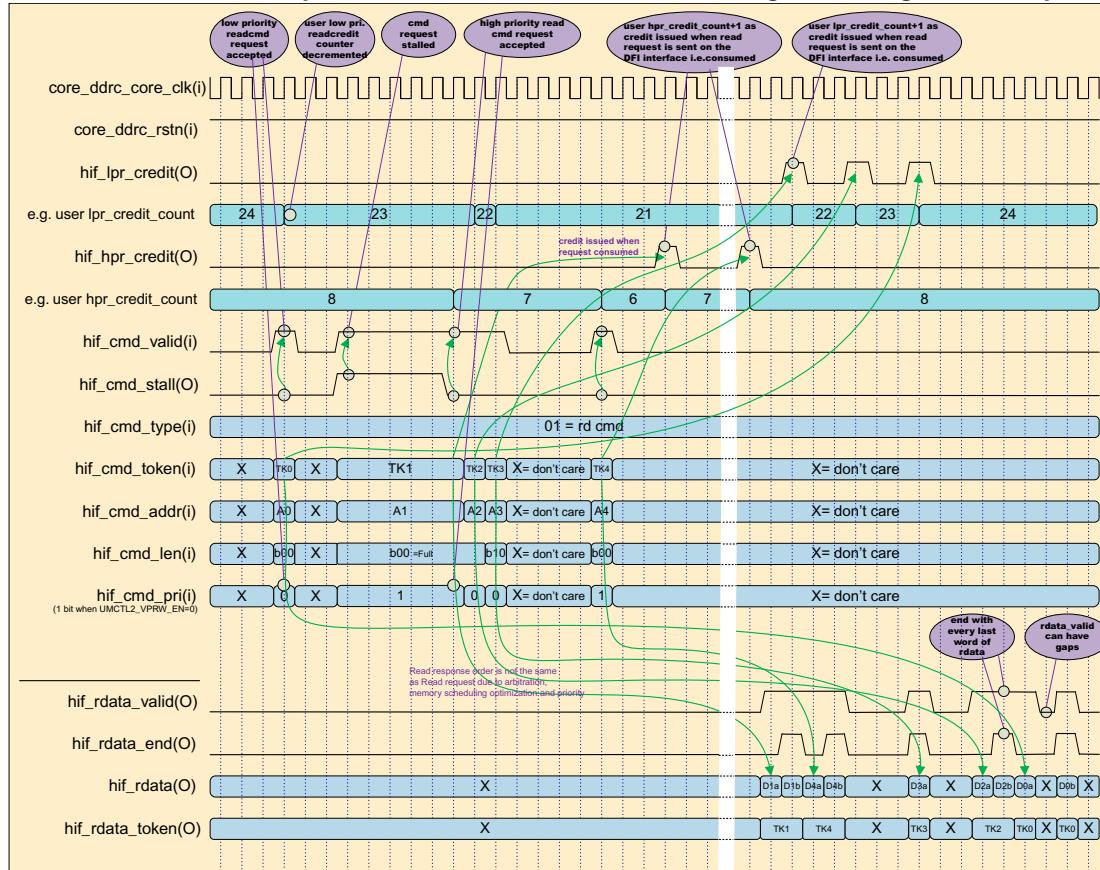
- `hif_cmd_token`: a bit field that is presented with the read command and is returned with the read response. As the read responses may be presented out-of-order, the token is the identifier that indicates the read for which data is being returned on the response side. Therefore, multiple reads with the same token must never be pending in the DDRCTL simultaneously, as the responses would be indecipherable. This field is meaningless for write or RMW commands. It is sized by the parameter `MEMC_TAGBITS`. The `hif_cmd_token` signal is carried internally as a sideband and returned with the corresponding read response `hif_rdata_*` signal group, but otherwise has no internal use.



`hif_cmd_token` must be a customer logic supplied unique number specific for their application with a minimum number of values of `MEMC_NO_OF_ENTRY`. As reads may be returned out of order, a read reorder buffer is required in the customer logic to provide ordered read responses with the correct requester ID.

[Figure 3-11](#) on page 84 shows a series of low and high priority reads and demonstrates the reordering of the read response data due to memory scheduling optimization including priority.

The matching credit will be issued as soon as the read request command is selected and issued to the DFI interface (that is, “consumed” from a CAM perspective). The delay in the data is the read latency through the PHY to the actual DDR back through the PHY to `core_ddrc_core_clk` domain to the HIF interface.

Figure 3-11 LPR and HPR Read Requests with Stalls and Credit Counting, Including Read Responses Example

3.3.5.2 hif_rdata_addr_err

hif_rdata_addr_err is asserted when the requested address for a read is not valid. Invalid address can be caused by using ADDRMAP12.nonbinary_device_density. In these cases, DDRCTL performs an aliased read by changing the physical address to a valid one (row [MSB:MSB-1] changes from 2'b11 to 2'b10). The returned data is masked to '0' and hif_rdata_addr_err is asserted along with the relevant hif_rdata_valid/hif_rdata_token. For more information, see “[Non-binary Device Densities](#)” on page [162](#).

3.3.5.3 Read Response Interface

Figure 3-11 shows multiple read requests and read responses.

The response token indicates which read data is being provided. Once started, read data for a given token always completes before the next token's read data is returned. The hif_rdata_valid signal indicates valid data is on the bus and the hif_rdata_end signal indicates that the last data packet is being transmitted. The hif_rdata_valid can de-assert in the middle of returning read data for one or more cycles, so hif_rdata_valid must be monitored to validate each cycle of returning read data.

There is no stall supported on the read response data phase so the user logic must use every hif_rdata, hif_rdata_end, and hif_rdata_token when the hif_rdata_valid is asserted.

3.3.5.4 Response Data Ordering

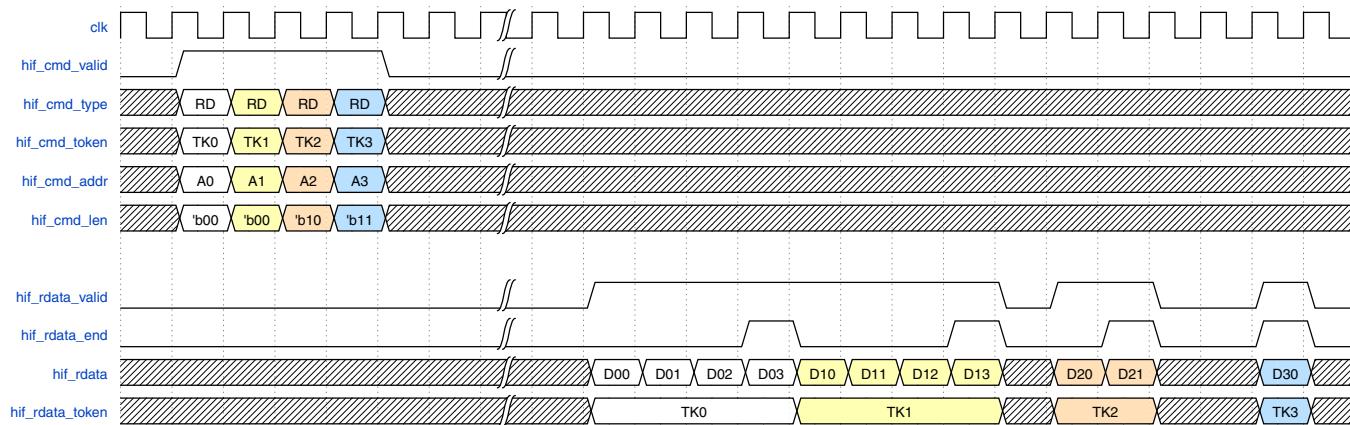
Read data is returned on the HIF in the same order in which it is received from the PHY on the DFI interface. It is independent of whether the read address is aligned or unaligned. For information on how unaligned addresses are handled, see “[Sequential Operations](#)” on page [200](#).

For data lane mapping examples where the addresses are aligned, see “[Data Lane Mapping Examples](#)” on page [461](#).

3.3.5.5 Read Data Ordering for BL32

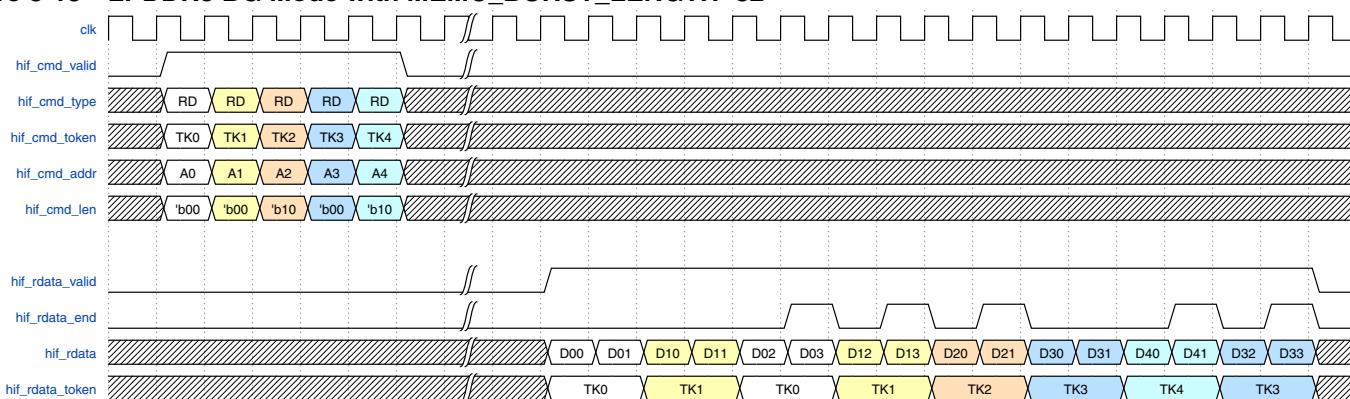
When using LPDDR4 or LPDDR5 16B mode with `MEMC_BURST_LENGTH=32`, the DDRCTL will output four consecutive beats of read data when BL32 RD is issued.

Figure 3-12 LPDDR4 or LPDDR5 16B Mode with `MEMC_BURST_LENGTH=32`



However, when using LPDDR5 BG mode with `MEMC_BURST_LENGTH=32`, the DDRCTL will output two BL16 read data and it can be interleaved with another read data. This read data order is the same as LPDDR5. As `hif_rddata_end` is only asserted on the last read data, `hif_rdata_token` can be changed without `hif_rddata_end`.

Figure 3-13 LPDDR5 BG Mode with `MEMC_BURST_LENGTH=32`



3.3.6 Writes

3.3.6.1 Write Requests

When the user Write credit count is greater than zero, the interface can issue write requests to the DDRCTL accordingly. A command is issued to the DDRCTL by asserting `hif_cmd_valid` on the rising edge of the clock.

All write request fields must be driven to the appropriate values at the same time. These fields are:

- `hif_cmd_type`: specifies the request type as follows:
 - '00' Write request.
 - '01' Read request.
 - '10' RMW request.
 - '11' Reserved.
- `hif_cmd_pri`: specifies the priority of the write request:
 - When `UMCTL2_VPRW_EN` is set to '0', this signal is 1-bit wide. This field is meaningless for write commands.
 - When `UMCTL2_VPRW_EN` is set to '1', this signal is 2-bit wide.

The encoding for Write is:

- 2'b00 - NPW (Normal Priority Writes)
- 2'b01 - VPW (Variable Priority Writes)
- 2'b10 - Reserved
- 2'b11 - Reserved

VPW commands are sent only in configurations where `UMCTL2_VPRW_EN=1`.

- `hif_cmd_addr`: indicates the address for the write. For more information, see "[HIF Address to SDRAM Address Mapping](#)" on page [158](#).
- `hif_cmd_wdata_ptr`: a pointer into the requester own buffers that can be used to subsequently retrieve the write data. Once a write or RMW request is accepted, this pointer is returned to the interface to retrieve the associated write data. (This field is not required, as requests are always accepted in order; the interface can choose to FIFO these pointers internally and ignore this field).

The stall mechanism for write or RMW requests is identical to that for read requests as explained in the section "[Valid and Stall](#)" on page [81](#).

Writes have three phases of operation: a write command, a write data pointer return, and finally write data provided to the DDRCTL. The write command phase indicates to the DDRCTL that a write transfer is required. The write data pointer return phase indicates to the SoC that the DDRCTL is ready to receive the write data. The write data phase provides the write data to the DDRCTL.

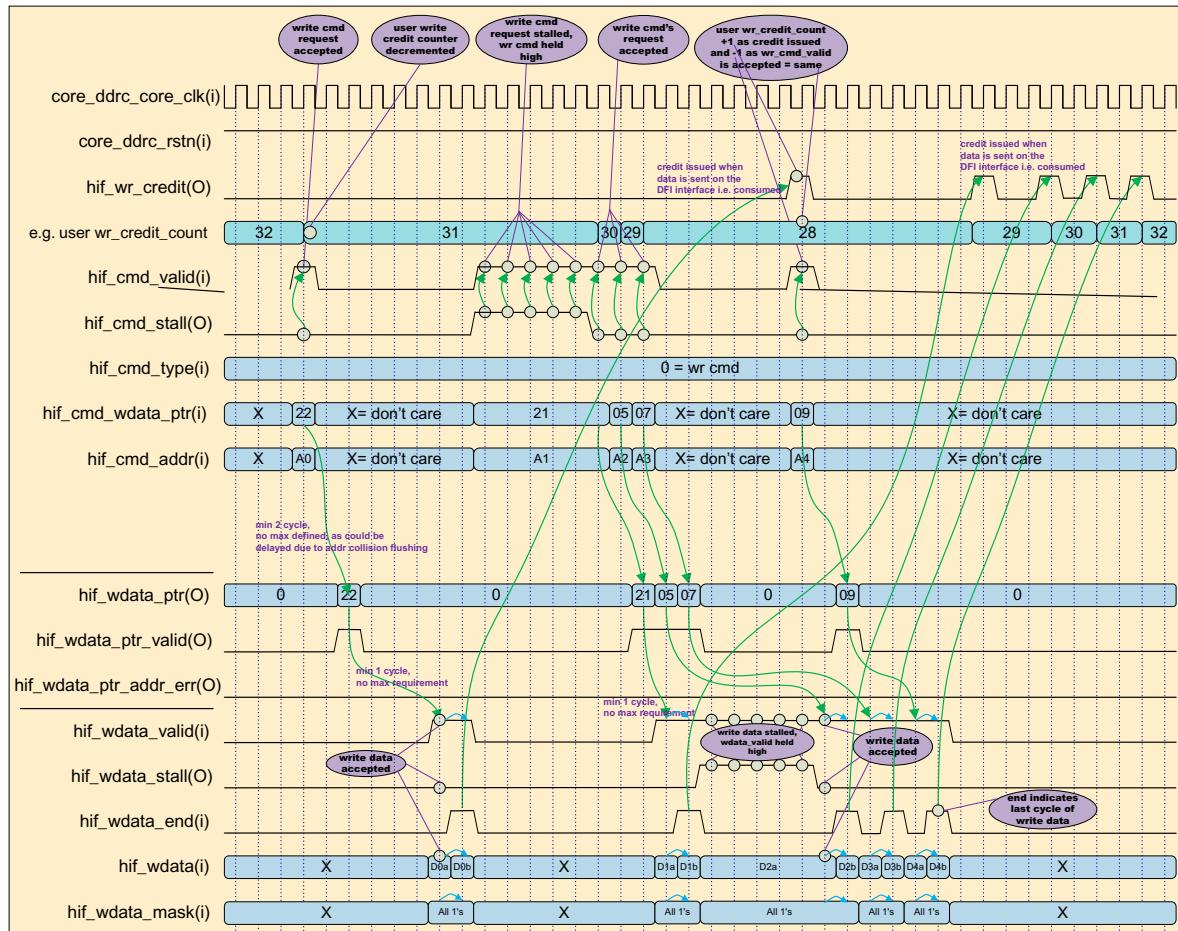
In all these three phases, the transactions must follow the same order. The data pointer is returned in the same order in which the command is sent. The write data must also be sent in the same order. The DDRCTL can perform re-ordering to the commands later, before sending them on the DFI interface, to make efficient use of the SDRAM bandwidth.

Figure 3-14 on page 87 is an example of five writes, with two stalls, of which one write cmd request and one wdata phase is stalled, each with the behavior for MEMC_BURST_LENGTH=16, FREQ_RATIO 1:4 configurations, where there are two cycles of data flagged by hif_wdata_valid with the hif_wdata_end asserted in the second cycle.

For MEMC_BURST_LENGTH=32, FREQ_RATIO 1:4 configurations, a full write command has four cycles of data flagged by hif_wdata_valid and hif_wdata_end is asserted on the fourth cycle.

The credit is issued when that request has been sent on the DFI interface.

Figure 3-14 Write Request and Data – One cmd Stall, One wdata Stall, Each wdata Phase with Two Clocks of Data (Full Writes), MEMC_BURST_LENGTH=16



3.3.6.2 Write Data Pointer Return

Following the acceptance of a write or RMW request, the DDRCTL then returns the write data pointer to the interface logic to retrieve the associated write data. This is done by asserting hif_wdata_ptr_valid on the rising edge of clock. In the same cycle, hif_wdata_ptr indicates the value of the write data pointer. This is same as a pointer that is presented to the DDRCTL as hif_cmd_wdata_ptr during the write command phase.

There is no mechanism for stalling the acceptance of the write data pointer from the DDRCTL; the interface logic must present write data for every cycle in which hif_wdata_ptr_valid is asserted by the DDRCTL.

The mechanism to throttle `hif_wdata_ptr_valid` assertions is for the interface to throttle write requests presented to the DDRCTL.

There is no required timing between a write or RMW command presented to the DDRCTL, and the same pointer is returned as `hif_wdata_ptr`. Multiple pointers can also be presented to the DDRCTL with multiple requests before the first is returned to the interface. The order of the pointers returned matches the order of the requests presented to the DDRCTL.

The use of the output signal `hif_wdata_ptr` returned write pointer is optional for the customer logic as the order of the write data must be in the same order as the write commands.

The next `hif_wdata` must not be sent until the next `hif_wdata_ptr_valid` has been received, indicating that the DDRCTL is ready to receive the next `hif_wdata`, and accept that `wdata`, subject to `hif_wdata_stall`.

3.3.6.3 `hif_wdata_ptr_addr_err`

There is an extra output for DDRCTL, named `hif_wdata_ptr_addr_err`. This is asserted only when the requested address for a write or RMW is not valid. Invalid address can be caused by using `ADDRMAP12.nonbinary_device_density` or LUT based heterogeneous table. When this happens, `hif_wdata_ptr_addr_err` is asserted along with the relevant pair of `hif_wdata_ptr_valid/hif_wdata_ptr`. In these cases, DDRCTL expects to receive write data, but discards them along with the relevant write or RMW request. For more information, see “[Non-binary Device Densities](#)” on page [162](#).

3.3.6.4 Write Data

After a write data pointer is returned to the interface logic, the interface logic must retrieve the associated data and present it to the DDRCTL. The write data must not be presented to the DDRCTL until the corresponding write data pointer is returned to the interface logic. There is no other requirement for timing between write data and write data pointer. However, the write data must be presented in the same order that the pointers returned to the interface (same order in which the original requests are made).

The write data can be throttled using `hif_wdata_stall` signal. The write data is accepted by the DDRCTL when `hif_wdata_valid` is detected high and `hif_wdata_stall` is detected low at the same positive edge of the clock. The signal `hif_wdata_stall` is asserted by the DDRCTL during an RMW operation.

When `MEMC_BURST_LENGTH=32` and write combine is disabled (`OPCTRL0.dis_wc=1`), the signal `hif_wdata_stall` may also be asserted by the write address collision while colliding WR data is being output.

When the DDRCTL writes the read data for the RMW operation into the write data SRAM, it blocks the SoC interface from sending any write data during those cycles by asserting this signal.

3.3.6.5 Write Data SRAM

The write data SRAM can be implemented internally as synthesized flip-flops or implemented externally to the DDRCTL as embedded SRAM using `UMCTL2_WDATA_EXTRAM` parameter. In either case, the control circuits for the Write Data RAM are clocked by the core DDRCTL clock. For more information about `wdataram` signals, refer to the “Signal Descriptions” chapter.

For data lane mapping examples, see “[Data Lane Mapping Examples](#)” on page [461](#).

Figures 3-15 to 3-17 show the timings of write and read data RAM interfaces.

Figure 3-15 Write Data RAM Interface Timing (DDRCTL_WDATARAM_WR_LATENCY=0, DDRCTL_WDATARAM_RD_LATENCY=1)

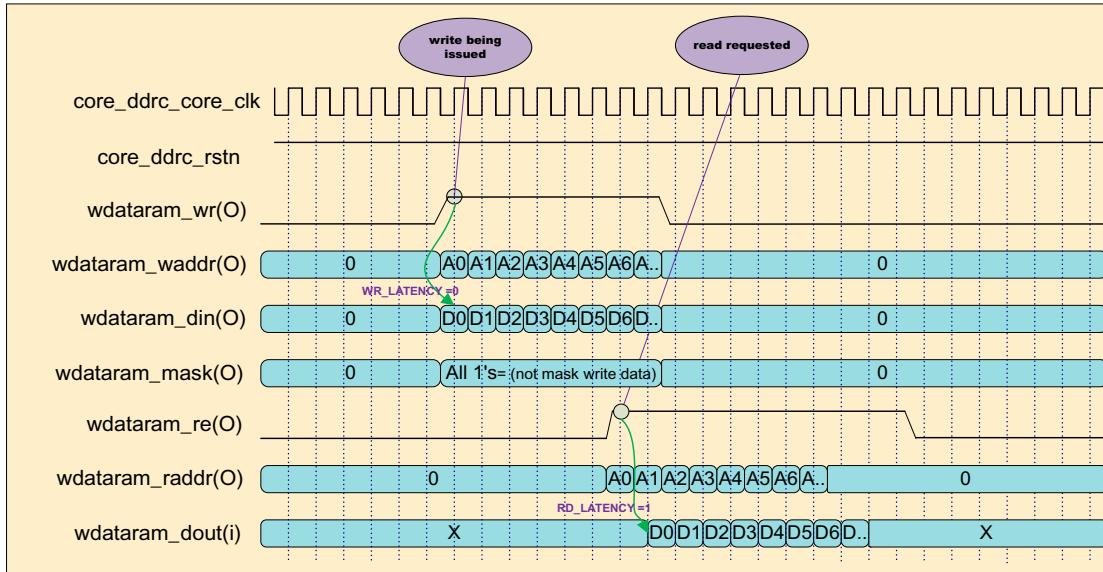
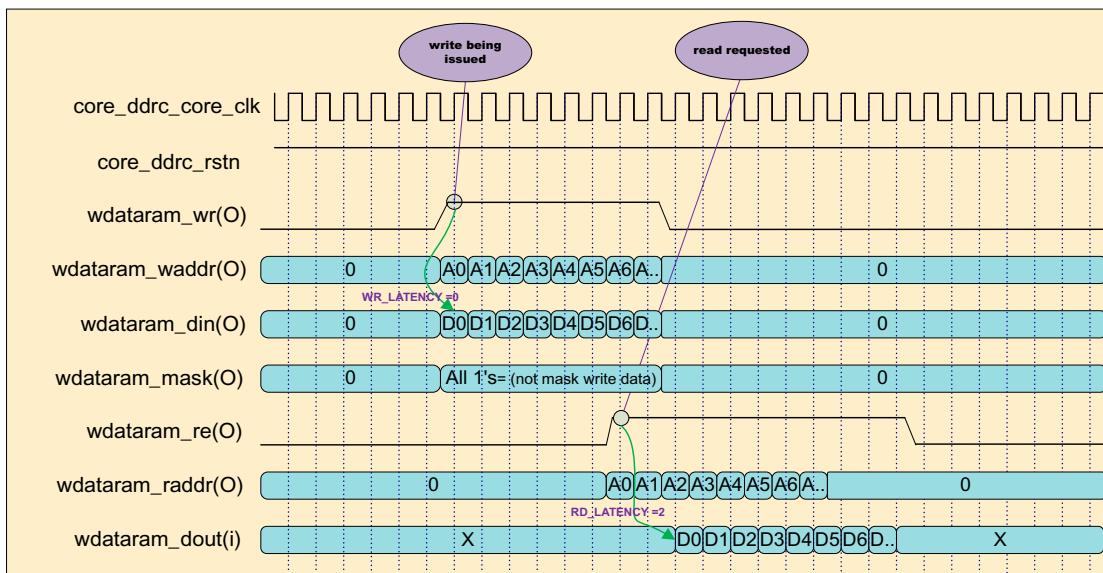


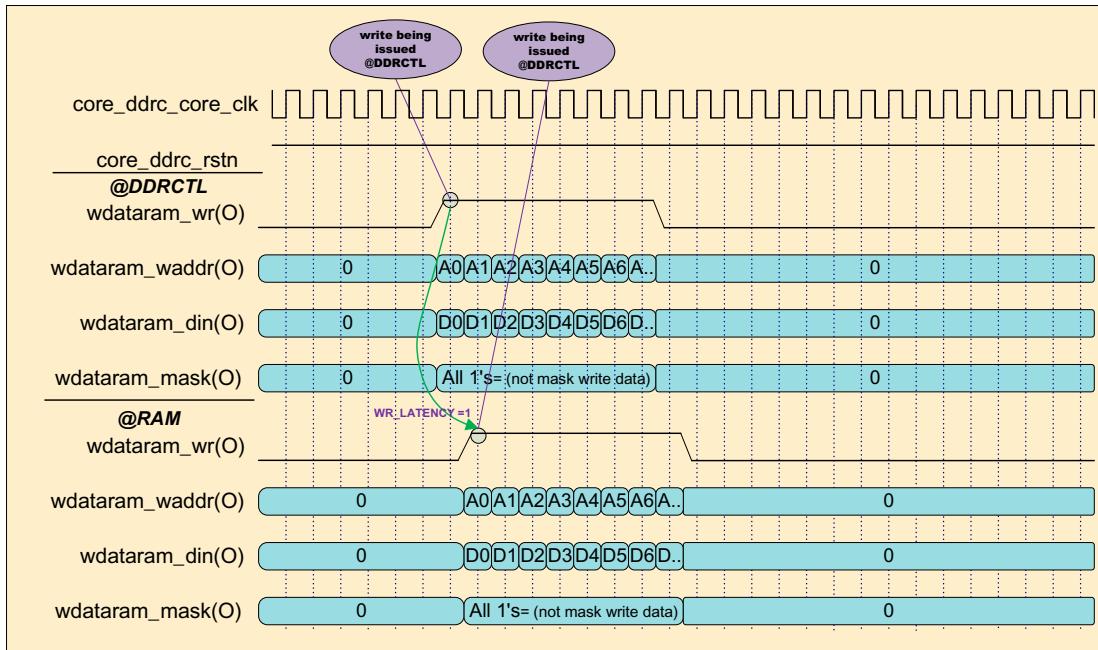
Figure 3-16 Write Data RAM Interface Timing (DDRCTL_WDATARAM_WR_LATENCY=0, DDRCTL_WDATARAM_RD_LATENCY=2)



The parameter **DDRCTL_WDATARAM_WR_LATENCY** specifies the number of clock cycles for external Write Data SRAM write that can be delayed before the RAM is written to. It is enabled when **(UMCTL2_WDATA_EXTRAM==1) && (DDRCTL_DDR==1)**.

Figure 3-17 shows signals as seen at the DDRCTL and at the RAM.

Figure 3-17 Write Data RAM Interface Timing (DDRCTL_WDATARAM_WR_LATENCY=1)



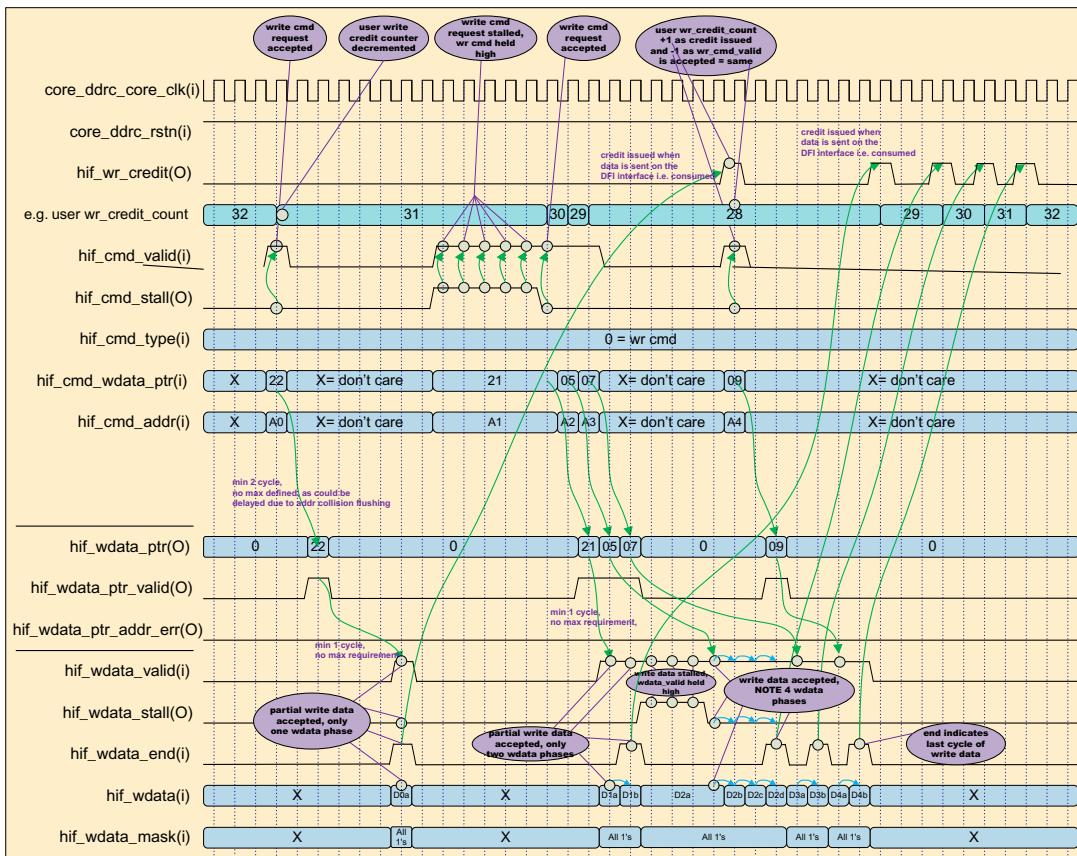
3.3.6.6 Write Data for the DDRCTL in BL16 Mode (MEMC_BURST_LENGTH=16)

Write data is presented to the DDRCTL with the assertion of `hif_wdata_valid`. Each request can have one to two (1:4 frequency ratio) or four (1:2 frequency ratio) data beats. `hif_wdata_valid` must be asserted for each data phase and `hif_wdata_end` must be asserted on the last data beat. `hif_wdata` is the data. Both `hif_wdata_end` and `hif_wdata` are valid only when `hif_wdata_valid` is '1'.

In 1:4 frequency ratio mode (`MEMC_FREQ_RATIO=4`), a normal write or RMW command has two clocks of data associated with it. In this case, `hif_wdata_end` must be asserted on the second data beat. If the SoC has less than two cycles of write data for a write command, it can choose to send one cycle of data, and `hif_wdata_end` must be asserted in the appropriate last data beat - these are referred to as Partial Writes. In this case, the DDRCTL has all the data that it requires one clock cycle earlier than in case of a full write.

Figure 3-18 shows sample timing diagrams.

Figure 3-18 Write Request and Data - One cmd Stall, One Write Data Stall With Mix of Partial Writes With 1, 2, 4 wdata Phases (4 wdata Phases Are Only Applicable When MEMC_FREQ_RATIO = 2)



3.3.6.7 Write Data for the DDRCTL in BL32 Mode (MEMC_BURST_LENGTH=32)

Write data is presented to the DDRCTL with the assertion of hif_wdata_valid. Each request can have one to four (1:2 frequency ratio) data beats. hif_wdata_valid must be asserted for each data phase and hif_wdata_end must be asserted on the last data beat. hif_wdata is the data. Both hif_wdata_end and hif_wdata are valid only when hif_wdata_valid is '1'.

In 1:4 frequency ratio mode (MEMC_FREQ_RATIO=4), a normal write or RMW command has four clocks of data associated with it. In this case, hif_wdata_end must be asserted on the fourth data beat. If the SoC has less than four cycles of write data for a write command, it can choose to send one, two, or three cycles of data, and hif_wdata_end must be asserted in the appropriate last data beat – these are referred to as Partial Writes. In this case, the DDRCTL has all the data that it requires three, two, or one clock cycle earlier than in case of a full write.

3.3.6.8 Partial Writes

A Partial Write is where the number of HIF write data beats is less than the number required for a normal (full) write for the MEMC_BURST_LENGTH.

The DDRCTL issues the minimum number of SDRAM beats on the DDR interface required, depending on the number of HIF write data beats and the HIF address alignment with respect to the SDRAM Column address - as SDRAM Writes must be sent BL-aligned. This additional logic impacts the achievable synthesis timing and increases the area.

A partial write will result in a masked write if `DBICTL.dm_en=1`, else an RMW if `MEMC_USE_RMW=1`, otherwise partial write are not permitted. For more information, see “[Burst Mode Operation](#)” on page 200.



Note When `MEMC_BURST_LENGTH=32` and there are two cycles of data without mask (`hif_wdata_mask=all ('1')`) with `hif_addr[4]=0`, the DDRCTL will issue WR16 (BL16) command.

3.3.7 Read-Modify-Write Requests

Read-modify-write (RMW) functionality in the DDRCTL is intended to support sub-sized write accesses in the following cases, where it is not possible to perform masked writes:

- Data Mask (DM) is disabled: If DM function is disabled through MR13 OP[5], it is necessary to use an RMW to perform a sub-sized write access.

`MEMC_USE_RMW` can be always set to '1'.

HIF configurations, if RMW is not used, only full BL16 bursts are allowed when `MEMC_BURST_LENGTH=16` and full BL32 burst and BL16 burst with `hif_addr[4]=0` are allowed when `MEMC_BURST_LENGTH=32`.

More specifically, to perform an RMW, the HIF input signal, `hif_cmd_type` must be driven to "10".

RMW requests result in a low priority read (LPR) and a write in the DDRCTL. The SoC must decrement one credit each for LPR and write credit counters for every RMW request sent to the DDRCTL. There is no RMW that results in a high priority read (HPR) part.

The DDRCTL allocates one CAM entry in the write CAM and the LPR CAM to handle this request. There is no read data going back to the SoC for RMW commands. The DDRCTL first schedules the read part of the RMW request. The read data from the SDRAM is merged with the write data of the RMW command in the on-chip write data SRAM (external to the DDRCTL). The merged data is then written to the SDRAM when the write is scheduled. The read and write requests of the RMW commands are treated as normal commands in the DDRCTL. The only difference is that the write command is not enabled in the CAM until the read data is merged with the write data.

The following signals are ignored for RMW commands:

- `hif_cmd_length`: the read associated with RMW command is always full read.
- `hif_cmd_token`: there is no read data going back to the SoC for RMW.
- `hif_cmd_pri`: when `UMCTL2_VPRW_EN` is set to '0', this signal is 1-bit wide. This field must be '0' for RMW commands.

When `UMCTL2_VPRW_EN` is set to '1', `hif_cmd_pri` signal is 2-bit wide.

The encoding for RMW is:

- 2'b00 - NPW (Normal Priority RMW)
- 2'b01 - VPW (Variable Priority RMW, both read part and write part of RMW are Variable Priority)
- 2'b10 - Reserved

- 2'b11 - Reserved

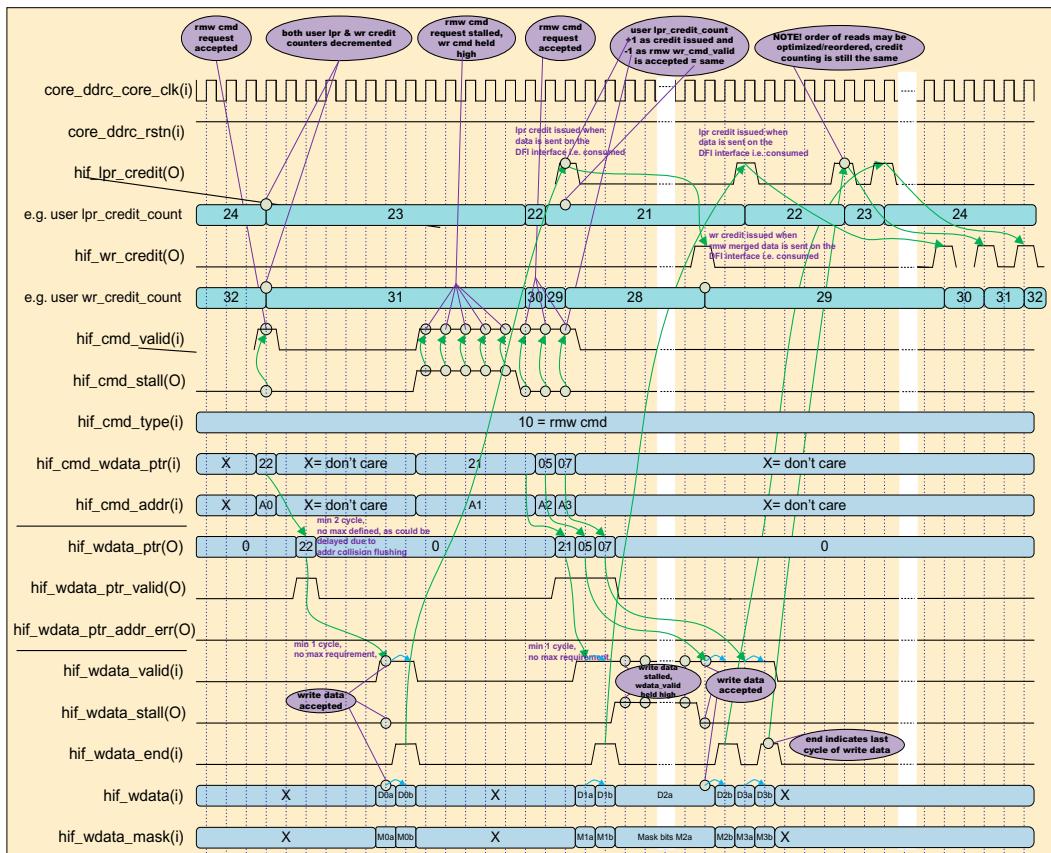
VPW commands are sent only in configurations where UMCTL2_VPRW_EN=1.

The SoC must decrement a low priority read credit and a write credit for every RMW command that it sends to the DDRCTL.



Note When MEMC_BURST_LENGTH=32, the DDRCTL will issue RD32 and WR32 regardless of the write data size when it receives an RMW command.

Figure 3-19 RMW Request and WData with wmask - One cmd Stall, One Write Data Stall with Four RMW Requests with User LPR and Write Credit Counter Examples



3.3.8 WR Versus RMW for HIF Configurations With DBICTL.dm_en=0

If RMW is not used, only full BL16 aligned bursts are allowed.

More specifically, writes are allowed only in specific HIF address and number of HIF data beats combinations. Otherwise, RMW is required.

3.3.9 Half/Quarter Bus Width Mode

When half bus width mode is used (MSTR0.data_bus_width=2'b01), the least significant half data for each DRAM beat of HIF read/write data is valid.

For example, if `MEMC_DRAM_DATA_WIDTH=32`, in half bus width case, the lower side 16 bits for each beat is valid data.

Invalid data of `hif_wdata` have to be set to '0'. For invalid data of `hif_wdata`, corresponding bits of `hif_wdata_mask` must be set to '1'.

For more information, see “[Bus Width Selection](#)” on page [200](#).



- The HBW feature is currently unsupported in configurations with OCECC (`UMCTL2_OCECC_EN==1`).

3.3.10 hif_cmd_latency

Valid when `hif_cmd_valid` is high and `hif_cmd_pri` indicates variable priority read/write (VPR/VPW). Don't care for other priority types.

Specifies the timeout value of VPR/VPW request with a vector size of $[(HIF_RQOS_TW-1):0]$. The controller starts counting down VPR/VPW timer when the request is accepted in the controller. If VPR/VPW timeout value is set to '0', the VPR/VPW request expires immediately as it enters the controller, thereby making it the highest priority transaction class within the device. While in the controller the VPR/VPW timer for that transaction is > 0 , it is treated a low_priority transaction, when the VPR/VPW timer for that transaction is $= 0 = \text{expired-VP}^*$, it is treated a high_priority (priority0) transaction.

3.3.11 hif_go2critical

When the DDRCTL is configured as HIF-only (`UMCTL2_INCL_ARB==0`), it provides a feature where the SoC can force this state transition using external signals. It is useful in cases where the SoC may have additional information that is helpful in determining state transitions. For example, if the data stream has real-time requirements and the SoC has knowledge about FIFO depths or time-till-failure, it can use this information to explicitly request the DDRCTL to prioritize a particular data stream when it is critical to do so. The signals associated with this feature are `hif_go2critical_lpr`, `hif_go2critical_hpr` and `hif_go2critical_wr`.

For example, if the user logic implemented timing monitoring (duration of outstanding transaction waiting for credits) in the queue of transactions which are requested via the HIF interface, if any critical transaction is delayed (held for longer than is critical) due to 0 credits available, it can set the corresponding `hif_go2critical_(lpr/hpr/wr)` signal. These must be asserted only when necessary as it will cause a switch to service that queue (to make credits available) which can lead to less than optimal scheduling decisions.

3.3.12 hif_cmd_autopre

The explicit auto-precharge feature can enable auto-precharge on a per-command basis. If the HIF signal `hif_cmd_autopre` is set during a valid command, then the auto-precharge bit for that command is set when it is sent to the SDRAM.

For more information, see “[Page Policy](#)” on page [175](#) and for options, see [Table 7-1](#) on page [176](#).

3.3.13 hif_mrr_data_valid and hif_mrr_data

There are additional output signals `hif_mrr_data[(MEMC_MRR_DATA_TOTAL_DATA_WIDTH-1) : 0]` and `hif_mrr_data_valid` which make the mode register contents available only when `hif_mrr_data_valid` is asserted.

For more information, see “[Mode Register Reads](#)” on page 252.

3.3.14 Additional Error Signal Outputs

There are additional error output signals (subject to configuration) for indicating crc (`hif_rdata_crc_err`), ecc (`hif_rdata_uncorr_ecc_err`), parity (`hif_rdata_eapar_err`), and kbd (`hif_rdata_kbd`).

For more information about these error signals, see “[HIF Read Data Interface Signals](#)” section in the “Signal Descriptions” chapter.

3.4 Hardware Low-Power Interfaces

For information about Hardware low-power Interfaces, refer to “[Hardware Low-Power Interfaces](#)” on page [270](#).

4

DFI

This chapter contains the following sections:

- “[DFI Interface](#)” on page [98](#)
- “[DFI Updates](#)” on page [110](#)
- “[DFI Constraints](#)” on page [113](#)
- “[DDRCTL to PHY Connections](#)” on page [115](#)

4.1 DFI Interface

This topic contains the following sections:

- “Overview of DFI Interface” on page 98
- “1:2 Frequency Ratio Mode Considerations” on page 99
- “DFI Write/Read Data to SDRAM Conversion” on page 99
- “Command Interface” on page 100
- “Write Data Interface” on page 100
- “Read Data Interface” on page 101
- “Update Interface” on page 101
- “Status Interface” on page 102
- “DFI Training” on page 103
- “Low-Power Control Interface” on page 104
- “PHY Master Interface” on page 105
- “MC to PHY Message Interface” on page 108
- “DFI Error Interface” on page 108
- “Signals Related to DFI Interface” on page 109
- “Registers Related to DFI Interface” on page 109

4.1.1 Overview of DFI Interface

The DDRCTL controller contains a DFI MC (Memory Controller) interface, which is used to connect to a DFI-compliant PHY, and to transfer address, control, and data to the PHY. In case of a Synopsys DDR PHY, the DDRCTL interfaces to a PUB block, which acts as the DFI PHY interface. The DDRCTL controller including its DFI Interface, runs at:

- Half data rate (HDR) clock (referred to as 1:2 frequency ratio mode in the DFI Specification)
- Quarter data rate (QDR) clock (referred to as 1:4 frequency ratio mode in the DFI Specification)

You can get more information about the DFI interface from the following specification:

- DDR PHY Interface (DFI) Specification, version 5.0, April 27, 2018

The DFI interface is sub-divided into the following interface groups:

- Command Interface: The command interface is a reflection of the SDRAM control interface including address, bank, chip select, row strobe, column strobe, write enable, clock enable, and ODT control, as applicable for the memory technology.
- Write Data Interface: The write data interface handles the transmission of write data across the DFI interface. The DFI Specification defines signals and timing relationships. The timing of this interface is configurable, to support all DFI-compliant PHYs through the DFITMG0.dfi_tphy_wrlat and DFITMG0.dfi_tphy_wrdata registers.
- Read Data Interface: The read data interface handles the return of read data across the DFI interface. The DFI Specification defines signals and timing relationships. The timing of this interface is configurable, to support all DFI-compliant PHYs through the DFITMG0.dfi_t_rddata_en register.
- Update Interface: During system operation, the system may require updates to the internal settings to compensate for environmental conditions. To ensure that updates do not interfere with signals on the

SDRAM interface, the DFI interface supports update modes where the DFI read, write, and command interface is suspended from the normal activity.

- Status Interface: The DFI interface requires status information for initialization and clock control to the SDRAM devices. These signals are used to convey information between the MC and PHY.
- Training (PHY-Independent mode supported by the DDRCTL): Write and read leveling function performed in a DFI-compliant PHY with no DDRCTL interaction.
- Low-power control interface: this interface consists of signals that are used to inform the PHY of a low-power mode opportunity, as well as how quickly the DDRCTL requires the PHY to resume normal operation.

4.1.2 1:2 Frequency Ratio Mode Considerations

The DDRCTL controller, including its DFI interface runs at the half data rate (HDR) clock (referred to as 1:2 frequency ratio mode in the DFI Specification), so all DFI command and address signals are twice the width of the equivalent DDR SDRAM signals. DFI data signals are four times the width of the equivalent DDR SDRAM signals. The DDRCTL controller operates with an HDR clock. The PHY handles HDR-to-SDR conversion on the address bus to memory and HDR/DDR conversion on the data buses. In 1:2 frequency ratio mode, `core_ddrc_core_clk` runs at half the frequency of CK/CK#.

4.1.3 DFI Write/Read Data to SDRAM Conversion

This conversion is performed in the PHY, and therefore PHY-Specific. The following details are applicable only to Synopsys DDR PHYs; other PHYs may differ from this.

The DFI data (both read and write) must contain data for multiple byte lanes and for multiple data beats (4 in 1:2 mode, 8 in 1:4 mode). The DFI Specification does not explicitly state how this data must be packed within `dfi[0|1]_rddata_W[0|1|2|3]` and `dfi[0|1]_wrdata_P[0|1|2|3]`. The Synopsys DDR PHYs and the DDRCTL DFI interfaces are designed with the assumption that the data is ordered by beat and then (within each beat) by byte.

For a 4-byte configuration with 32 bits of SDRAM in 1:2 mode, it is more complicated than 16 bits of SDRAM because both DFI0 and DFI1 need to be used. The DFI data order is as follows (from MSB to LSB):

- DFI1 (Phase 1)=[byte3-beat3, byte2-beat3, byte3-beat2, byte2-beat2]
- DFI0 (Phase 1)=[byte1-beat3, byte0-beat3, byte1-beat2, byte0-beat2]
- DFI1 (Phase 0)=[byte3-beat1, byte2-beat1, byte3-beat0, byte2-beat0]
- DFI0 (Phase 0)=[byte1-beat1, byte0-beat1, byte1-beat0, byte0-beat0]

Therefore, the PHY sends the data it receives out to the memory in the following way:

- In beat 0, the data from `dfi0_wrdata_P0[15:0]` are sent on DQ[15:0]
- In beat 1, the data from `dfi0_wrdata_P0[31:16]` are sent on DQ[15:0]
- In beat 2, the data from `dfi0_wrdata_P1[15:0]` are sent on DQ[15:0]
- In beat 3, the data from `dfi0_wrdata_P1[31:16]` are sent on DQ[15:0]
- In beat 0, the data from `dfi1_wrdata_P0[15:0]` are sent on DQ[31:16]
- In beat 1, the data from `dfi1_wrdata_P0[31:16]` are sent on DQ[31:16]
- In beat 2, the data from `dfi1_wrdata_P1[15:0]` are sent on DQ[31:16]
- In beat 3, the data from `dfi1_wrdata_P1[31:16]` are sent on DQ[31:16]

For more information on addressing, see “[Channel Modes](#)” on page [125](#).

Similar ordering is true for `dfi_wrdata_mask` and `dfi_rddata/dfi_rddata_dbi`.

The `dfi_wrdata_en`, `dfi_rddata_en`, and `dfi_rddata_valid` signals have one bit corresponding to each byte of DQ (not each byte of the DFI data signals).

This means, for a 4-byte configuration (32 bits of SDRAM) in 1:2 mode, the DFI ordering for `dfi_wrdata_en` is as follows (from MSB to LSB):

```
[byte3-phase1, byte2-phase1, byte1-phase1, byte0-phase1, byte3-phase0,
byte2-phase0, byte1-phase0, byte0-phase0]
```

The DFI ordering for `dfi_rddata_en` signal is same as `dfi_wrdata_en`. The DFI ordering of `dfi_rddata_valid` is same except that phase0/1 is actually word0/1.

There may be confusion due to these different mappings:

- `dfi_wrdata/dfi_wrdata_mask/dfi_rddata/dfi_rddata_dbi`
- `dfi_wrdata_en/dfi_rddata_en/dfi_rddata_valid`

For clarification, refer to the provided example.

For a 4-byte configuration (32 bits of SDRAM) in 1:2 mode, `dfi_wrdata_en` and `dfi_wrdata` mapping are as follows:

```
dfi0_wrdata_en_P0[0] -> {dfi0_wrdata_P0[23:16], dfi0_wrdata_P0[ 7: 0]} (byte0-phase0)
dfi0_wrdata_en_P0[1] -> {dfi0_wrdata_P0[31:24], dfi0_wrdata_P0[15: 8]} (byte1-phase0)
dfi1_wrdata_en_P0[0] -> {dfi1_wrdata_P0[23:16], dfi1_wrdata_P0[ 7: 0]} (byte2-phase0)
dfi1_wrdata_en_P0[1] -> {dfi1_wrdata_P0[31:24], dfi1_wrdata_P0[15: 8]} (byte3-phase0)

dfi0_wrdata_en_P1[0] -> {dfi0_wrdata_P1[23:16], dfi0_wrdata_P1[ 7: 0]} (byte0-phase1)
dfi0_wrdata_en_P1[1] -> {dfi0_wrdata_P1[31:24], dfi0_wrdata_P1[15: 8]} (byte1-phase1)
dfi1_wrdata_en_P1[0] -> {dfi1_wrdata_P1[23:16], dfi1_wrdata_P1[ 7: 0]} (byte2-phase1)
dfi1_wrdata_en_P1[1] -> {dfi1_wrdata_P1[31:24], dfi1_wrdata_P1[15: 8]} (byte3-phase1)
```



If non-Synopsys DDR PHY is used, these signals may be reordered depending on the bytes/beats ordering of the non-Synopsys DDR PHY.

4.1.4 Command Interface

The command interface is a reflection of the SDRAM control interface including address, bank, chip select, row strobe, column strobe, write enable, and clock enable, as applicable for the memory technology.

4.1.5 Write Data Interface

The write data interface handles the transmission of write data across the DFI interface. The DFI Specification defines signals and timing relationships.

The timing of this interface is configurable, to support all DFI-compliant PHYs through the `DFITMG0.dfi_tphy_wrlat` and `DFITMG0.dfi_tphy_wrdata` register.

For DFI 1:2, if HDR, timing for `dfi_wrdata*` generation for write commands on phase 0/phase 1 are the same and `dfi_wrdata*` signals are aligned to HDR clock. If SDR, timing for `dfi_wrdata*` generation for write commands on phase 0/phase 1 are treated separately and `dfi_wrdata*` signals are not guaranteed to be aligned to HDR clock. Check your PHY requirements for correct programming.

The signal `dfi_wrdata_mask` can act as Write DBI signal depending on the programming of the SDRAM mode register. For more information on DBI, see “[Data Bus Inversion \(DBI\)](#)” on page [254](#).

4.1.5.1 Support For `dfiN_wrdata_cs_Px`/`dfiN_rddata_cs_Px` Signals

The polarity of these signals is defined by `DFIMISC.dfi_data_cs_polarity`.

The signal `dfi_wrdata_cs` is driven as follows:

- The `dfi_wrdata_cs` is driven according to the relevant chip select `tphy_wrcslat` DFI PHY clock cycles after any command that generates `dfi_wrdata`. This includes a write. It is guaranteed to remain at this value for a minimum of `tphy_wrcsgap` plus the time for the write data (usually `MSTR0.burst_rdwr`).

Similarly, for `dfi_rddata_cs`:

- The `dfi_rddata_cs` is driven according to the relevant chip select `tphy_rdcslat` DFI PHY clock cycles after any command that generates `dfi_rddata`. This includes a read, in LPDDR4, and a MRR. It is guaranteed to remain at this value for a minimum of `tphy_rdcsgap` plus the time for the read data (usually `MSTR0.burst_rdwr`).

[Table 4-1](#) outlines the DFI timing value and the equivalent register in the DDRCTL.

Table 4-1 DFI Timing Values for `dfi_wrdata_cs`/`dfi_rddata_cs`

DFI Timing Value	DDRCTL Register
<code>tphy_wrcslat</code>	<code>DFITMG2.dfi_tphy_wrcslat</code>
<code>tphy_wrcsgap</code>	<code>RANKTMG0.diff_rank_wr_gap</code>
<code>tphy_rdcslat</code>	<code>DFITMG2.dfi_tphy_rdcslat</code>
<code>tphy_rdcsgap</code>	<code>RANKTMG0.diff_rank_rd_gap</code>

4.1.6 Read Data Interface

The read data interface handles the return of read data across the DFI interface. The DFI Specification defines signals and timing relationships.

The timing of this interface is configurable, to support all DFI-compliant PHYs through the `DFITMG0.t_rddata_en` register.

For DFI 1:2, in case of HDR, timing for `dfi_rddata_en` generation for read commands on phase 0/phase 1 is the same and `dfi_rddata*` signals are aligned to HDR clock. In case of SDR, timing for `dfi_rddata*` generation for read commands on phase 0/phase 1 are treated separately and `dfi_rddata*` signals are not guaranteed to be aligned to HDR clock. Check your PHY requirements for correct programming.

The DDRCTL controller has been verified with the Synopsys PHYs. It assumes that the DFI parameter `tphy_rdlat` does not exceed 48 cycles.

4.1.7 Update Interface

During system operation, the system may require updates to the internal settings to compensate for environmental conditions. To ensure that updates do not interfere with signals on the SDRAM interface, the

DFI interface supports update modes where the DFI read, write, and command interface is suspended from the normal activity. The DFI Specification defines both MC-initiated and PHY-initiated updates.

4.1.7.1 MC-initiated Updates

MC-initiated updates can be acknowledged or ignored by the PHY. DFI MC-initiated updates are performed periodically by the DDRCTL. For more details, see “[DFI MC-Initiated Update Requests](#)” on page [110](#).

4.1.7.2 PHY-initiated Updates

For more details, see “[DFI PHY-initiated Update Requests](#)” on page [111](#).

4.1.8 Status Interface

The DFI interface requires status information for initialization and clock control to the SDRAM devices. These signals are used to convey information between the MC and PHY.

4.1.8.1 Initialization

The `dfi_init_start` signal is used to trigger the PHY initialization by setting it to ‘1’. It is driven by the DDRCTL through the APB interface with the `DFIMISC.dfi_init_start` read-write register field. The signal `dfi_init_complete` indicates that the PHY has completed its initialization. This information can be read/polled by the DDRCTL through the APB interface with the `DFISTAT.dfi_init_complete` read-only register field.

4.1.8.2 Clock Disabling

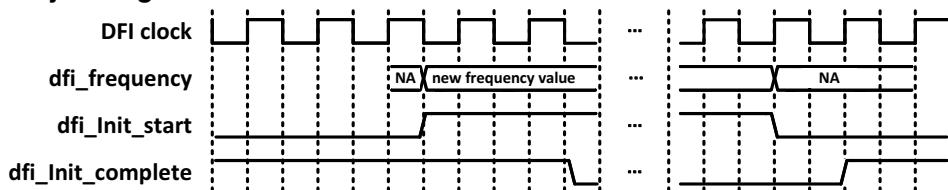
The `dfi_dram_clk_disable` signal depends on setting of `PWRCTL.en_dfi_dram_clk_disable`. For more information about clock disabling, see “[Assertion of dfi_dram_clk_disable](#)” on page [267](#).

4.1.8.3 Frequency Change

The interface consists of three signals

- `dfi_init_start`: Output from the controller to the PHY
- `dfi_init_complete`: Output from the PHY to the controller
- `dfi_frequency`: Output from the controller to the PHY

Figure 4-1 Frequency Change



The procedure is as follows:

`dfi_frequency` indicates the operating frequency of the system. This signal must change only at initialization, during a DFI frequency change operation, or other times that the system defines. This signal must be constant during normal operation. The number of supported frequencies and the mapping of signal values to clock frequencies are defined by the PHY, system, or both. For timing, the `dfi_frequency` signal must be set to a legal value and remain unchanged when `dfi_init_start` is asserted. The `dfi_frequency` signal can change any time when `dfi_init_start` is low and must be ignored at this time.

The behavior of the `dfi_init_start` signal is dependent on the `dfi_init_complete` signal.

If the PHY accepts the frequency change request, it must de-assert the `dfi_init_complete` signal within `tinit_start` cycles of the `dfi_init_start` assertion. The MC continues to hold the `dfi_init_start` signal asserted until the clock frequency change has been completed. The de-assertion must be used by the PHY to re-initialize on the new clock frequency.

If the frequency change is not acknowledged (the `dfi_init_complete` signal remains asserted), the `dfi_init_start` signal must de-assert after `tinit_start` cycles. This requirement needs to be handled by the system as `dfi_init_start` is programmed through the software.



Note Once DDRCTL asserts HWFFC driven `dfi_init_start`, it will not be de-asserted even if `tinit_start` is elapsed until the PHY de-asserts `dfi_init_complete` followed by assertion of `csysreq_ddrc`.

4.1.9 DFI Training

The additional functions of SDRAM memories allow accurate alignment of critical timing signals. These are trained through write and read leveling functions in a DFI-compliant PHY.

The supported modes in DDRCTL are described in “[PHY Independent Mode](#)” on page 103.

If you are using the Synopsys DDR PHY, use PHY independent mode. While interfacing to Synopsys DDR PHYs, the training is initiated by the PHY Utility Block (PUB) present in the PHY, and PHY independent training mode is used.

4.1.9.1 PHY Independent Mode

PHY independent mode are enabled by default and no dedicated signals are required.

Training is assumed to be performed automatically by the PHY, without any intervention from the DDRCTL.

If you are using the Synopsys DDR PHY, the PUB block (part of the PHY) performs all required MRW commands to move the SDRAM to the correct state for each training stage.

If you are using a non-Synopsys PHY that supports PHY independent training mode, DDRCTL may have to perform these MRW commands.

However, DDRCTL has no knowledge on when the PHY is ready to execute each training stage, so this must be controlled by the user.

4.1.10 Low-Power Control Interface

In a DDR memory subsystem, it is advantageous to place the PHY in a low-power state when the DDRCTL has knowledge that the memory subsystem remains idle for a period of time. Depending on the state of the system, the DDRCTL communicates state information to the PHY allowing the PHY to enter the appropriate power saving state. This interface consists of signals that are used to inform the PHY of a low-power mode opportunity, as well as how quickly the DDRCTL requires the PHY to resume normal operation.

Software control is provided in DFILPCFG0/DFILPCFG1 registers for the following:

- This is an optional interface for a DFI-compliant PHY. The interface signals always exist in the DDRCTL. But the software can enable both dfi_lp_ctrl_req and dfi_lp_data_req interfaces in self-refresh power down and/or power down and/or deep sleep mode - DFILPCFG0.dfi_lp_en_pd, DFILPCFG0.dfi_lp_en_sr, and DFILPCFG0.dfi_lp_en_dsm respectively.

The software can also enable dfi_lp_data_req interface in data bus idle DFILPCFG0.dfi_lp_en_data.

- Software control over what is driven on dfi_lp_ctrl_wakeup and dfi_lp_data_wakeup signals (and its associated timing) in self-refresh power down and/or power down and/or deep sleep mode - DFILPTMG0.dfi_lp_wakeup_pd, DFILPTMG0.dfi_lp_wakeup_sr, and DFILPCFG0.dfi_lp_wakeup_dsm respectively.

The software can also control what is driven on dfi_lp_data_wakeup signal (and its associated timing) in data bus idle DFILPTMG1.dfi_lp_wakeup_data.

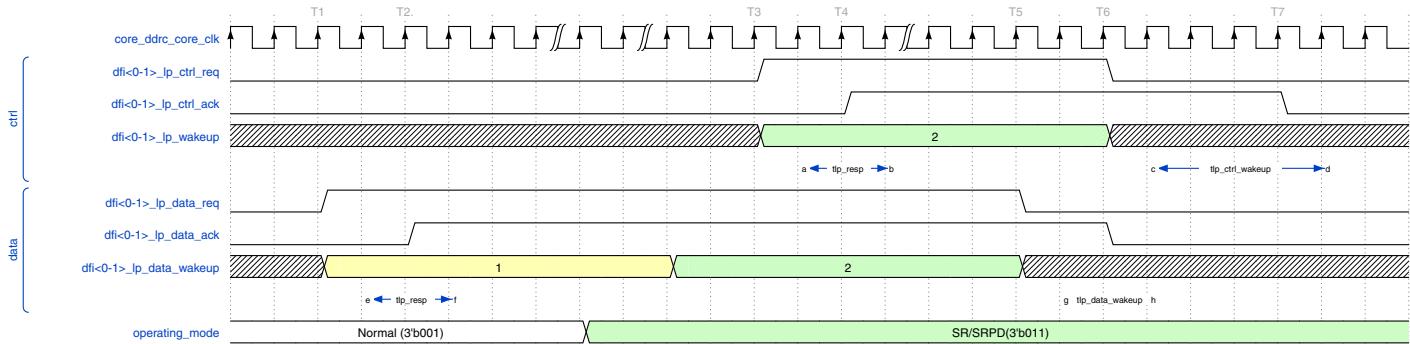
- The timing of this DFI low-power interface - DFILPTMG0.dfi_tlp_resp.

dfi_lp_ctrl_req/dfi_lp_ctrl_ack and dfi_lp_data_req/dfi_lp_data_ack work independently, but when dfi_lp_data_req/dfi_lp_data_ack handshake has already been completed and dfi_lp_ctrl_req/dfi_lp_ctrl_ack handshake is initiated, both interfaces coordinate each other and the value of dfi_lp_data_wakeup can be changed as shown in [Figure 4-2](#).

In figure, the timing parameters defined in the DFI specification are shown as follows:

- A gap between T1-T2: tlp_resp
- A gap between T3-T4: tlp_resp
- A gap between T5-T6: tlp_data_wakeup
- A gap between T6-T7: tlp_ctrl_wakeup

Figure 4-2 DFI Low Power Control



The following limitations apply to `dfi_lp_data_req/dfi_lp_data_ack`:

1. `dfi_lp_data_req/dfi_lp_data_ack` in data bus idle (no data transmission) can be supported only if `tlp_resp` is less than or equal to 2.

`dfi_lp_data_req/dfi_lp_data_ack` in self-refresh power down and/or power down and/or deep sleep mode can be supported even if `tlp_resp` is greater than '2'.

Since the minimum assert duration of `dfi_lp_data_req` in the controller in data bus idle is two `core_ddrc_core_clk` cycles regardless of `DFILPTMG1.dfi_tlp_resp`, it causes DFI violation if `tlp_resp` of the PHY is greater than '2'.

To avoid the issue, `DFILPCFG0.dfi_lp_en_data` shall be set to '0' so that `dfi_lp_data_req` is not asserted in data bus idle.

When the DDRCTL is in power down, self-refresh mode or DSM, the controller will continue to assert `dfi_lp_data_req` at least during

`DFILPTMG1.dfi_tlp_resp`

cycles, hence the issue does not happen.

2. The DDRCTL can support `dfi_lp_data_req/dfi_lp_data_ack` in data bus idle only if `tlp_data_wakeup` is less than or equal to '2'.

If `tlp_data_wakeup` in PHY is larger than '2', `DFILPCFG0.dfi_lp_en_data` shall be set to '0' so that `dfi_lp_data_req` is not asserted in data bus idle.



Note Synopsys LPDDR5/4/4X PHY can be configured to add pipeline by setting `DWC_DDRPHY_PIPE_DFI_MISC`, `DWC_DDRPHY_PIPE_DFI_RD`, and `DWC_DDRPHY_PIPE_DFI_WR` to 1. In this case, `tlp_resp` and `tlp_data_wakeup` is 3. As the controller can support DFI LP data only when `tlp_resp` and `tlp_data_wakeup <= 2`. Hence, if Synopsys PHY with pipeline is used, `DFILPCFG0.dfi_lp_en_data` must be set to 0 to disable DFI LP data.
If Synopsys PHY with pipeline is used, `DFILPCFG0.dfi_lp_en_data` must be set to 0. If you want to use DFI LP data, Synopsys PHY without pipeline must be used.

4.1.11 PHY Master Interface

This topic contains the following sections:

- “[Overview of PHY Master Interface](#)” on page 105
- “[LPDDR5/4 Considerations](#)” on page 107
- “[Registers Related to PHY Master Interface](#)” on page 107

4.1.11.1 Overview of PHY Master Interface

The PHY uses the PHY Master Interface for requesting DDRCTL to relinquish the DFI bus and take control of the DRAM bus. When the PHY has control of the DFI bus, it performs all operations independent of the DDRCTL. This interface is only supported in LPDDR4 or LPDDR5.

The PHY can request the memory to be placed in a specific state (such as, IDLE or self-refresh through `dfi_phymstr_cs_state/dfi_phymstr_state_sel`). However, the DDRCTL puts the SDRAM into Self-Refresh, without consideration of `dfi_phymstr_cs_state/dfi_phymstr_state_sel`.

DDRCTL is not expected to acknowledge `dfi_phymstr_req` in the following situations:

- The PHY does not issue a PHY Master request during SDRAM Initialization, or Deep Sleep Mode (LPDDR5 only).
- The PHY does not issue PHY Master request and PHY Update request simultaneously.

Support for this interface can be enabled/disabled using `DFIPHYMSTR.dfi_phymstr_en`. If enabled through the software, when a PHY Master request occurs, DDRCTL performs the following steps:

1. Ensures that there are no DFI low-power requests pending or in-service and de-asserts both `dfi_lp_ctrl_req` and `dfi_lp_data_req`.
2. Precharges all banks.
3. If `dfi_phymstr_req` is asserted after a SRX and there has been no REF after SRX, DDRCTL issues (`RFSHMOD0.refresh_burst+1`) REF/REFpb commands to speed up the SR entry. There is a JEDEC requirement for at least a REF (or 8 REFpb) to occur between SRX and SRE.
4. Enters self-refresh mode without draining the CAMs (to speed up the response time).
5. Acknowledges the PHY Master request.

After `dfi_phymstr_req` is dropped, DDRCTL de-asserts `dfi_phymstr_ack` signal and initiates Self-Refresh Exit.

The DDRCTL is expected to respond a `dfi_phymstr_req` in `tphymstr_resp = 8k` clock cycles. This is not guaranteed to be satisfied in the following situations:

- Automatic refreshes are disabled (that is, `RFSHCTL0.dis_auto_refresh=1`); if at least one REF command is needed before entering Self-Refresh, response time depends on software sending refreshes.
- While sending burst of automatic REF commands (needed before entering Self-Refresh); these REF commands are postponed due to JEDEC requirement of 16 refreshes in 2xtREFI.
- While sending burst of automatic REF commands (needed before entering Self-Refresh); these REF commands are postponed due to JEDEC requirement of 8 refreshes in a rolling tREFBW window.
- LPDDR4 case:
 - `RFSHMOD0.per_bank_refresh = 1`
 - `MEMC_NUM_RANKS > 1`
 - `MSTR0.active_ranks = 0x3 or 0xF`
 - `ZQCTL2.zq_resistor_shared = 1`
 - `ZQSET1TMG.t_zq_long_nop > 900`
- While DFI LP is active. In this situation, upon DFI LP exit, the following needs to hold:
 - `dfi_lp_ctrl_wakeup`
 - `dfi_lp_data_wakeup`

Note that, if clock removal feature is used (`core_ddrc_core_clk` is removed from the DDRCTL):

- It is not guaranteed to satisfy DFI's `tphymstr_resp` requirement of PHY, as `dfi_phymstr_req=1` is only observed once the clock has been re-enabled.
- If `dfi_phymstr_req` transitions from '1' to '0' and back to '1' while the clock has been removed, logic does not ensure that `dfi_phymstr_req` assertion is acknowledged through `dfi_phymstr_ack`.



- Enable DFI low-power interface for self-refresh (`DFILPCFG0.dfi_lp_en_sr`) setting is ignored during self-refresh entry caused by PHY Master request. DFI low-power interface needs to remain inactive during PHY Master request.
- If the PHY does not support PHY Master, it is recommended to set the following inputs:
 - `DFIPHYMSTR.dfi_phymstr_en` = 1'b0
 - `dfi_phymstr_req` = 1'b0
 - `dfi_phymstr_cs_state` = {`MEMC_NUM_RANKS`{1'b0}}
 - `dfi_phymstr_state_sel` = 1'b0
 - `dfi_phymstr_type` = 2'b00

4.1.11.2 LPDDR5/4 Considerations

In LPDDR5/4, there are two flavors of Self-Refresh state:

- Self-Refresh (SR): CKE is high, clock frequency cannot be changed, clock cannot be removed, some commands such as MRW/MRR can be performed.
- Self-Refresh Power Down (SRPD): CKE is low, clock frequency can be changed, clock can be removed, no commands can be performed.

LPDDR5/4 SRPD is the equivalent of Self-Refresh in other protocols. LPDDR5/4 SR is unique to LPDDR5/4 protocol. According to DFI 5.1, `dfi_phymstr_req` must put SDRAM into SR, not SRPD. To support this, DDRCTL is able to move:

- Into and out of SR only (IDLE → SR → IDLE)
- Out of and into SRPD (SRPD → SR → SRPD)

If hardware low-power interface is active, SRPD → SR → SRPD is not performed upon `dfi_phymstr_req` assertion, because in this situation, for most cases, the clock is removed, so transition is not possible. When this happens, DDRCTL informs external clock controller to re-enable the clock, by setting `cactive_ddrc` to '1'.

If software Self-Refresh is in progress (`PWRCTL.selfref_sw=1`), SRPD → SR → SRPD is performed if possible and if enabled by the `PWRCTL.lppddr4_sr_allowed` register (see the "Register Descriptions" chapter of the [DWC DDRCTL LPDDR5/4 Programming Guide](#)).



`DFIPHYMSTR.dfi_phymstr_blk_ref_x32` must be used with the default value (0x80) unless Synopsys suggests to update it.

4.1.11.3 Registers Related to PHY Master Interface

The following are the registers related to the PHY Master Interface:

- `DFIPHYMSTR.dfi_phymstr_en`
- `PWRCTL.lppddr4_sr_allowed`
- `DFIPHYMSTR.dfi_phymstr_blk_ref_x32`

For more information about these registers, refer to the "Register Descriptions" chapter in the LPDDR5X/5/4X Memory Controller Reference Manual.

4.1.12 MC to PHY Message Interface

Synopsys PHYs have no support for DFI MC to PHY Message Interface, therefore `DFI0MSGCTL0.dfi0_ctrlmsg_req` must be set to '0'.

Instead of DFI MC to PHY Message Interface, periodic re-training can be performed by DFI PHY Master interface, Controller Assisted Drift Tracking (LPDDR5) or Enhanced Incremental Periodic Phase Training (PPT2).

MC to PHY Message Interface (`dfi*_ctrlmsg_*`) is removed if the hardware parameter `DDRCTL_DFI_CTRLMSG` is '0'. Currently, the parameter is '0' only when `DDRCTL_PPT2=1&&UMCTL2_HWFFC_EN=1`.

4.1.13 DFI Error Interface

The DFI Error interface handles the transmission of error information from PHY to DDRCTL. PHY asserts `dfi_error` signal with `dfi_error_info[3:0]` after error condition occurs.

The `dfi_error` signal definition is design specific. See Synopsys DDR PHY databook for more details regarding this signal.

The PHY may support the `dfi_error` signal without `dfi_error_info` and it may also use a subset of the `dfi_error_info` signal bits. Signals that are not being driven must be tied low at the input of DDRCTL.

DDRCTL stores the input coming on the `dfi_error_info` signal in an APB register whenever `dfi_error` is asserted. An interrupt is flagged, and interrupt status register bit is set when this happens. No new error is captured until the previous interrupt is cleared.



DFI Error Interface is supported in limited configurations. For more information, contact Synopsys.

4.1.13.1 Hardware Parameter Related to DFI Error Interface

The following parameter specifies the Controller to include DFI Error Interface.

- `DDRCTL_DFI_ERROR`

4.1.13.2 Registers Related to DFI Error Interface

The following registers are related to the DFI Error Interface:

- `DFIERRINTRPTCFG`
- `DFIERRORSTAT`
- `DFIERRORSTAT1`

4.1.13.3 Interrupts Related to DFI Error Interface

The following interrupts are associated with the DFI Error Interface:

- `dfi_error_intr`
- `dfi_error_intr_fault`

4.1.14 Signals Related to DFI Interface

The following are the signals related to the DFI Interface:

- DFI Command Interface Signals
- DFI Write Data Interface Signals
- DFI Read Data Interface Signals
- Non-DFI DDRCTL PHY Sideband Interface Signals
- DFI Update Interface Signals
- DFI Status Interface Signals
- DFI Low-Power Control Interface Signals
- DFI PHY Master Interface Signals
- DFI MC to PHY Message Interface Signals
- DFI Error Interface Signals

For more information about these signals, refer to the “Signal Descriptions” chapter.

4.1.15 Registers Related to DFI Interface

The following are the registers related to the DFI Interface:

- DFITMG x
- DFILPCFG x
- DFIUPDX
- DFIMISC
- DFISTAT
- DFIPHYMSTR
- DFIMSGTMG0
- DFIOMSGCTL0
- DFIOMSGSTAT0
- DFIERINTRPTCFG
- DFIERROSTAT
- DFIERROSTAT1

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

4.2 DFI Updates

This topic contains the following sections:

- “[Overview of DFI Updates](#)” on page [110](#)
- “[DFI MC-Initiated Update Requests](#)” on page [110](#)
- “[DFI PHY-initiated Update Requests](#)” on page [111](#)
- “[Registers Related to DFI Updates](#)” on page [112](#)

4.2.1 Overview of DFI Updates

DFI update interface handles the transmission of updates to internal settings to compensate for environmental conditions. The interface includes signals and timing parameters. To ensure that updates do not interfere with signals on the DRAM interface, the DFI supports update modes where the DFI bus is placed must be in an idle state.

4.2.2 DFI MC-Initiated Update Requests



Note The DDRCTL supports two types of DFI MC-initiated update.

DFI MC-initiated update means the type 0 of `dfi_ctrlupd_req` unless otherwise stated.

The type 1 of `dfi_ctrlupd_req` is described in “[Enhanced Incremental Periodic Phase Training \(PPT2\)](#)” on page [239](#).

DFI MC-initiated update request (`dfi_ctrlupd_req`) is issued by the DDRCTL on the DFI interface so that the PHY can update delay line values from the master delay line. This signal must be asserted periodically to ensure that PVT (Process/Voltage/Temperature) variations are accounted for in the delay lines in PHY over time. An acknowledgment signal comes from PHY (`dfi_ctrlupd_ack`) to indicate that the delay line update process in PHY is complete. The DDRCTL follows the DFI Specification minimum and maximum requirements for duration to keep the DFI update request asserted.

It is important to control this carefully to ensure that the delay line is never updated during a read or a write as this could degrade the data eye.

The DDRCTL delays the assertion of `dfi_ctrlupd_req` if a PHY-initiated update is in progress (`dfi_phyupd_ack=1`).

There are two methods for performing the DFI updates:

- “[Automatic MC-Initiated Update Request](#)” on page [110](#)
- “[Direct Software Request of MC-initiated Update Request](#)” on page [111](#)

4.2.2.1 Automatic MC-Initiated Update Request

This method is used when `DFIUPD0.dis_auto_ctrlupd` is set to ‘0’. The DDRCTL must carefully control the assertion of `dfi_ctrlupd_req`. The DDRCTL asserts `dfi_ctrlupd_req` any time the DDRCTL is idle, where idle is defined as having no read or write cycles in the CAMs, no read responses buffered in the RT (Response Tracker) block, and no write data pending in the MR (memory read) block.

To force the DFI update requests, when the DDRCTL is not idle for a long period of time (this depends on the PHY - how frequently it requires an update request), the DDRCTL uses a timeout mechanism. When a

DFI MC-initiated update timeout occurs, the DDRCTL ceases to schedule reads and writes until the MR and RT blocks are idle and then `dfi_ctrlupd_req` is asserted.

4.2.2.2 Direct Software Request of MC-initiated Update Request

This method is used when `DFIUPD0.dis_auto_ctrlupd` is set to '1' through the software. In this case, the SoC decides when to do the DFI MC-initiated update requests. This is useful in cases where the bandwidth utilization of the SDRAM bus is extremely crucial and the DDRCTL does not want any break in transactions.

To perform the request, SoC must set `OPCTRLCMD.ctrlupd` to '1'. When the request is stored in the DDRCTL, the register bit is automatically cleared. The SoC can perform this request only if `OPCTRLSTAT.ctrlupd_busy` is low. The `OPCTRLSTAT.ctrlupd_busy` signal goes high in the clock after the DDRCTL accepts the request. It goes low when the DFI update operation is initiated in the DDRCTL.

The DDRCTL may assert `dfi_ctrlupd_req` at the same time as refresh assertion. For both the refreshes and the DFI MC-initiated update request, the DDRCTL ensures that these happen in a functionally-correct manner. The SoC logic ensures that they are scheduled frequently, and done at a time when the controller can tolerate the associated break in read/write scheduling.

4.2.3 DFI PHY-initiated Update Requests

DFI PHY-initiated update requests must be acknowledged by the DDRCTL. DFI specifies four different update PHY-initiate request modes. Each mode differs only in the number of cycles that the DFI interface must be suspended when the update occurs.

Support for this interface can be enabled/disabled using `DFIUPD0.dfi_phyupd_en`. If enabled through the software, when a PHY-initiated update request occurs, the DFI command, read data and write data channels are stalled as soon as possible (within a maximum of 64 clock cycles in 1:2 frequency ratio mode), and the DDRCTL drives `dfi_phyupd_ack` high in response.

The only exception to this is when both update request signals (`dfi_ctrlupd_req` and `dfi_phyupd_req`) are asserted at the same time. When both request signals are driven, the DDRCTL and the PHY could violate the DFI protocol by simultaneously acknowledging the other's request. To prevent this situation, the DDRCTL does not assert `dfi_phyupd_ack` while `dfi_ctrlupd_req` is asserted.

Note that if burst refreshes are being used with PHY-initiated updates, care must be taken to avoid tREFI violations, which could occur if a PHY update request is received shortly before a refresh burst is due to be transmitted. In this situation, the refresh burst is delayed until after the PHY update is complete. This can be avoided by reducing the value of `RFSHMOD0.refresh_burst`.

Note that if clock removal feature is exercised (`core_ddrc_core_clk` is removed from the DDRCTL):

- It is not guaranteed to satisfy DFI's `tphyupd_resp` requirement of PHY, as `dfi_phyupd_req=1` is only observed once clock has been re-enabled.
- If `dfi_phyupd_req` transitions from '1' to '0' and back to '1' while clock has been removed, logic does not guarantee that `dfi_phyupd_req` assertion is acknowledged through `dfi_phyupd_ack`.



- If PHY does not support the PHY-initiated updates, it is recommended to set the following inputs:
 - DFIUPD2.dfi_phyupd_en=1'b0
 - phy_ddrc_dfi_phyupd_req=1'b0
 - phy_ddrc_dfi_phyupd_type=2'b00
- The DDRCTL controller supports the following tPHYUPD_RESP value in terms of DFI clocks (controller clocks):

MSTR.lpddr4	0	1	
Frequency Ratio Mode	1:2	1:4	1:2
tPHYUPD_RESP	64	32	128

4.2.4 Registers Related to DFI Updates

The following are the registers related to the DFI Updates:

- DFIUPD0.dfi_phyupd_en
- DFITMG0.dfi_t_ctrl_delay

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

4.3 DFI Constraints

This topic contains the following sections:

- “[Overview of DFI Constraints](#)” on page [113](#)
- “[WCK Related Constraints \(LPDDR5\)](#)” on page [113](#)
- “[WCK Constraints \(LPDDR5\)](#)” on page [114](#)

4.3.1 Overview of DFI Constraints

Some constraints apply to every command regardless of the rank or bank targeted by the command. These constraints are primarily related to gaining access to the data bus while avoiding bus contention. A global constraints block enforces each of these constraints. Using the global constraints block, the scheduler dynamically obeys all of the constraints in [Table 4-2](#) when scheduling transactions.

Table 4-2 DFI Constraints

Control Register	Constraint Name	Description
DFITMG0.dfi_tphy_wrlat	Write to write-enable	This constraint is the time after a write command that dfi_wrdata_en must be driven to SDRAM. This corresponds to the DFI parameter tphy_wrlat.
DFITMG0.dfi_tphy_wrdata	Write-enable to write data time	This constraint is the time after dfi_wrdata_en that the data is driven to SDRAM. This corresponds to the DFI parameter tphy_wrdata.
DFITMG0.dfi_t_rddata_en	Read command to read enable time	This constraint is the time from the assertion of a read command on the DFI interface to the assertion of the dfi_rddata_en signal. This constraint corresponds to the DFI parameter trddata_en.
DFITMG0.dfi_t_ctrl_delay	Time for command to pass through PHY	This constraint corresponds to the DFI parameter tctrl_delay.

4.3.2 WCK Related Constraints (LPDDR5)

WCK related constraints are listed in [Table 4-3](#):

Table 4-3 WCK Related Constraints (LPDDR5)

Control Register	Constraint Name	Description
DFITMG4.dfi_twck_en_rd	t _{wck_en_rd}	WCK Enable Read Timing. Defines the timing from the CAS_WS_RD command to driving of the dfi_wck_en=ENABLED.
DFITMG4.dfi_twck_en_wr	t _{wck_en_wr}	WCK Enable Write Timing. Defines the timing from the CAS_WS_WR command o driving of the dfi_wck_en=ENABLED.
DFITMG4.dfi_twck_dis	t _{wck_dis}	WCK Off Timing. Defines the timing from the last command opportunity to the de-assertion of dfi_wck_en and dfi_wck_toggle_en assuming that no command is being sent.

Control Register	Constraint Name	Description
DFITMG5.dfi_twck_fast_toggle	$t_{wck_fast_toggle}$	This constraint defines the number of clock cycles between the <code>dfi_wck_signal</code> being driven to TOGGLE to when the <code>dfi_wck_signal</code> is driven to FAST_TOGGLE. This timing is only applicable when the WCK transitions from the slow to fast toggle. Otherwise, this timing parameter must be set to 0x0.
DFITMG5.dfi_twck_toggle	t_{wck_toggle}	This constraint defines the number of clock cycles between the <code>dfi_wck_en</code> signal being enabled to when the <code>dfi_wck_toggle</code> signal is driven to TOGGLE.
DFITMG5.dfi_twck_toggle_cs	$t_{wck_toggle_cs}$	This constraint defines the number of clock cycles between a read or write command to when the <code>dfi_wck_cs</code> signal must be stable. This timing is applicable when the WCK is synchronized for multiple CS's and commands are to different CS's. During WCK synchronization, the CS must be static from the CAS command to the completion of the synchronization sequence.
DFITMG5.dfi_twck_toggle_pos_t	$t_{wck_toggle_post}$	This constraint defines the number of clock cycles after a read or write command data burst completion during which the WCK must remain in the current toggle state. During this time, the <code>dfi_wck_cs</code> signal must also remain stable.

4.3.3 WCK Constraints (LPDDR5)

WCK constraints are listed in [Table 4-4](#):

Table 4-4 WCK Constraints (LPDDR5)

Control Register	Description
DRAMSET1TMG24.max_rd_sync	Maximum time between BL16 read commands (RD16) without a new WCK2CK sync start $RL + BL/n_max + RD(tWCKPST/tCK)$
DRAMSET1TMG24.max_wr_sync	Maximum time between BL16 write commands (WR16) without a new WCK2CK sync start $WL + BL/n_max + RD(tWCKPST/tCK)$

When `MEMC_BURST_LENGTH` is set to 32, the WCK constraints listed in [Table 4-5](#) are added.

Table 4-5 WCK Constraints (LPDDR5) (`MEMC_BURST_LENGTH=32`)

Control Register	Description
DRAMSET1TMG37.max_rd_sync_blx2	Maximum time between BL32 read commands (RD32) without a new WCK2CK sync start $RL + BL/n_max + RD(tWCKPST/tCK)$
DRAMSET1TMG37.max_wr_sync_blx2	Maximum time between BL32 write commands (WR32) without a new WCK2CK sync start $WL + BL/n_max + RD(tWCKPST/tCK)$

4.4 DDRCTL to PHY Connections

This section shows the diagrams which indicate how DFI sideband signals are connected between DDRCTL and PHY. Those DFI signals which come straight through are not displayed.

4.4.1 Single DDRC Dual DFI Configuration

This section shows the diagrams which indicate how DFI sideband signals are connected between DDRCTL and PHY in case of Single DDRC Dual DFI configuration. Those DFI signals which come straight through are not displayed.

[Figure 4-3](#) shows DDRCTL to PHY connections for Command Interface.

Figure 4-3 Command Interface

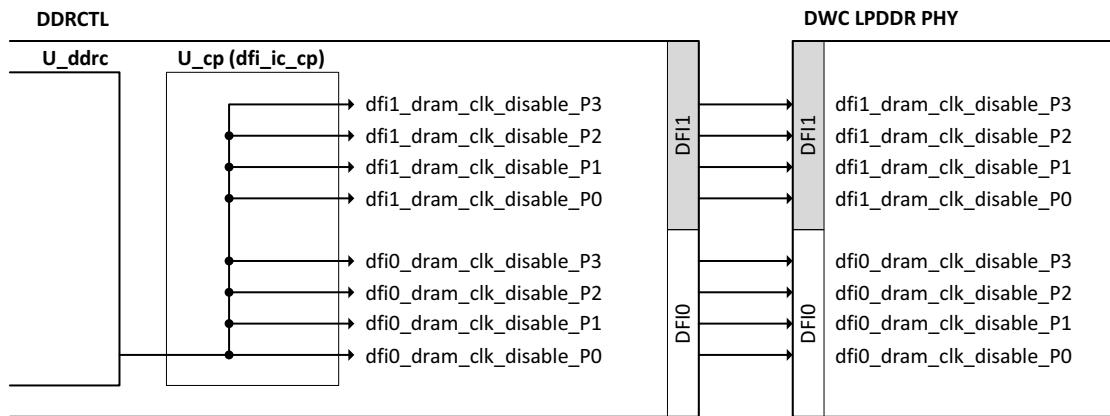
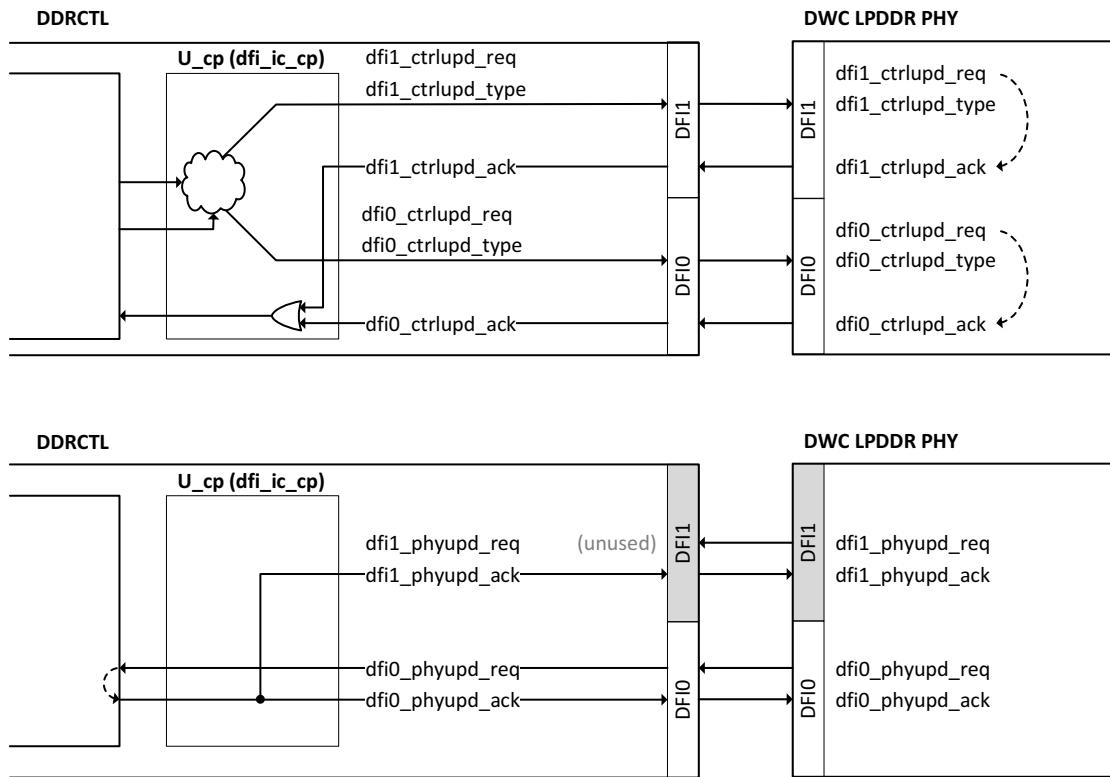


Figure 4-4 shows DDRCTL to PHY connections for Update Interface.

Figure 4-4 Update Interface



Note When **DDRCTL_PPT2** is not defined or **PPT2CTRL0.ppt2_en** is 0, **dfi1_ctrlupd_ack** is unused in DDRCTL, and **dfi1_ctrlupd_req** is the same as **dfi0_ctrlupd_req**.

Figure 4-5 shows DDRCTL to PHY connections for Status Interface.

Figure 4-5 Status Interface

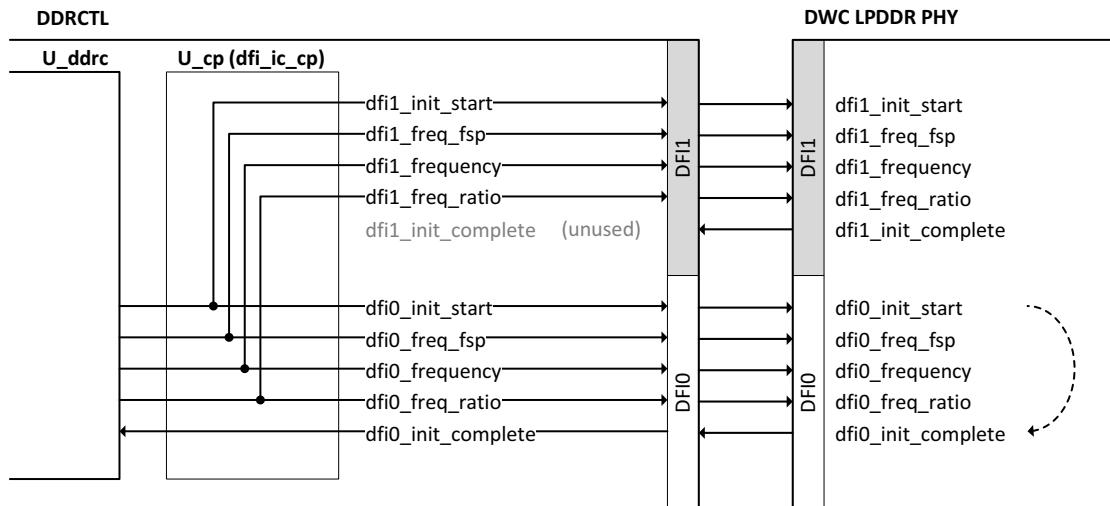


Figure 4-6 shows DDRCTL to PHY connections for low-power control interface.

Figure 4-6 Low-Power Control Interface

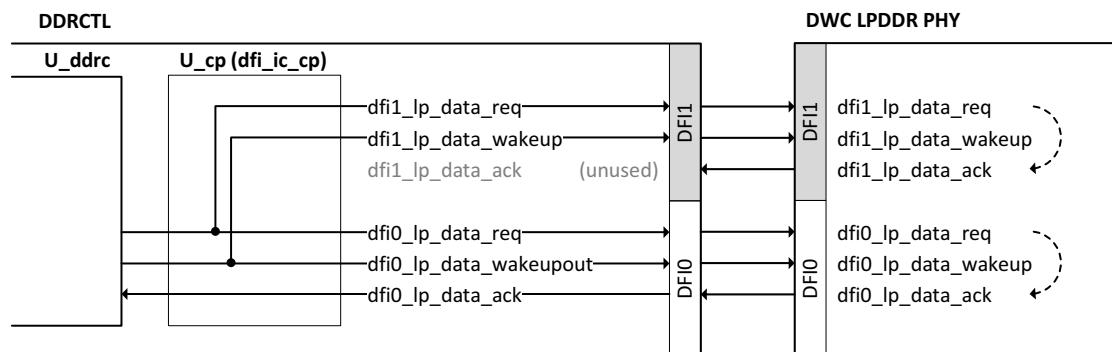
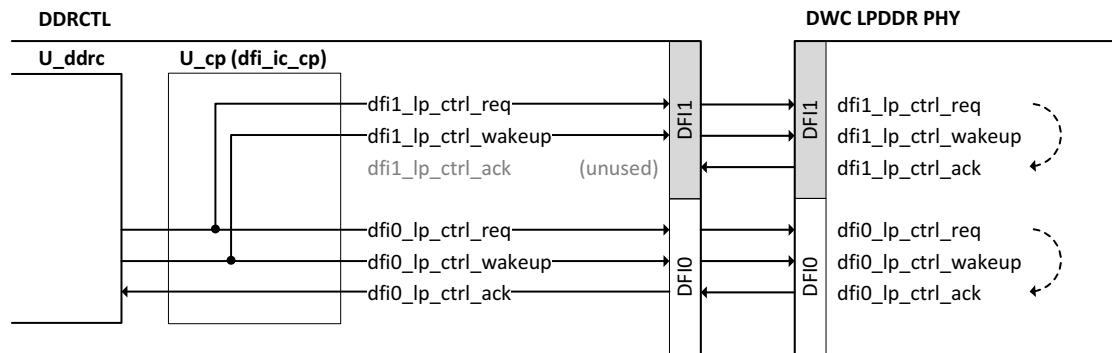


Figure 4-7 shows DDRCTL to PHY connections for PHY Master Interface.

Figure 4-7 PHY Master Interface

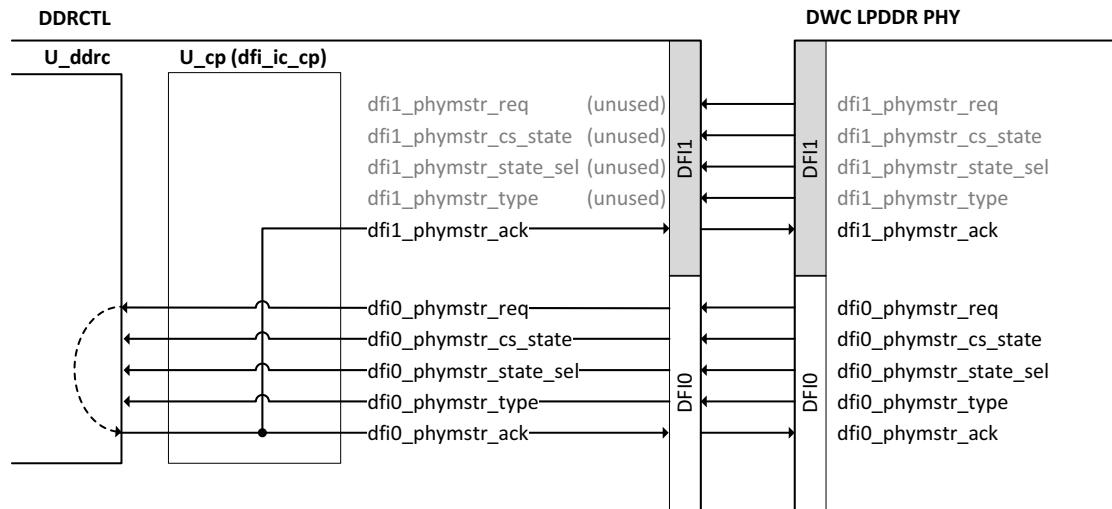
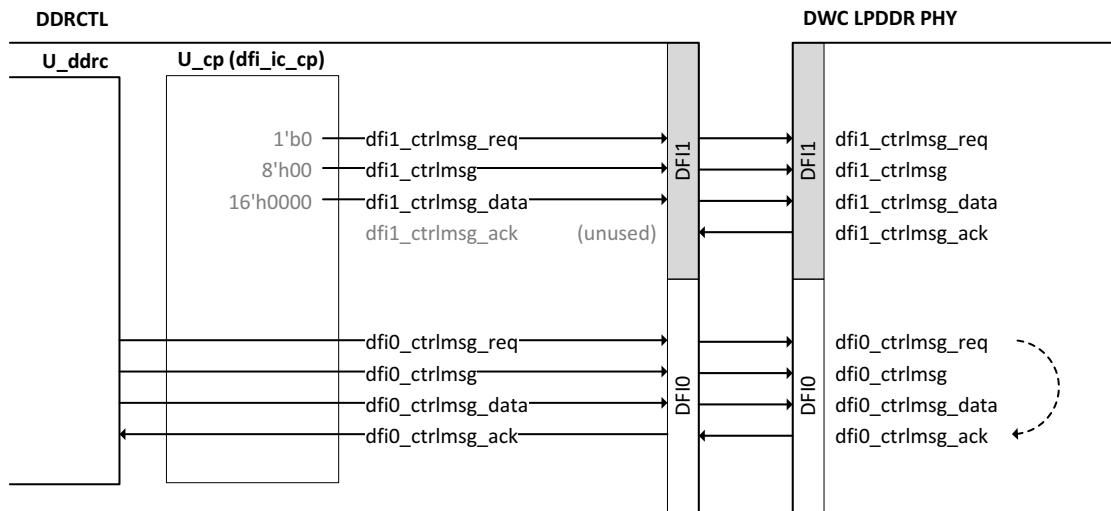


Figure 4-8 shows DDRCTL to PHY connections for MC PHY Message Interface.

Figure 4-8 MC PHY Message Interface



4.4.2 Single DDRC Quad DFI Configuration

This section shows the diagrams which indicate how DFI sideband signals are connected between DDRCTL and PHY in case of Single DDRC Quad DFI configuration. Those DFI signals which come straight through are not displayed.

Figure 4-9 shows DDRCTL to PHY connections for Command Interface.

Figure 4-9 Command Interface

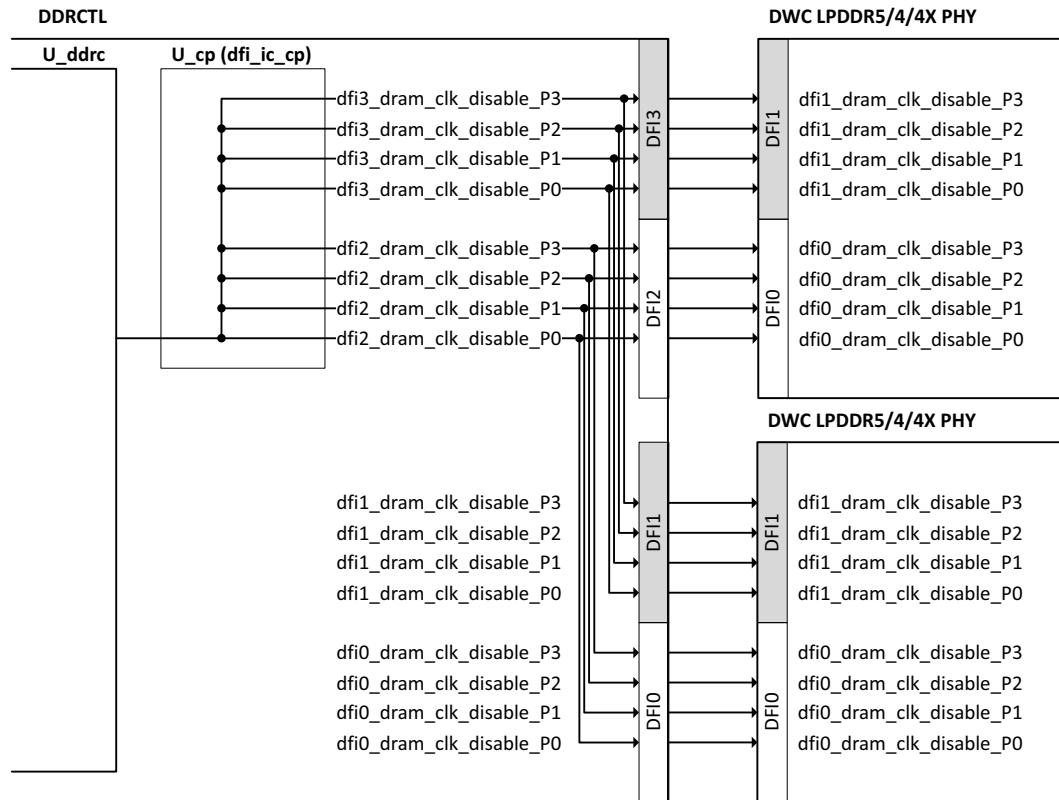
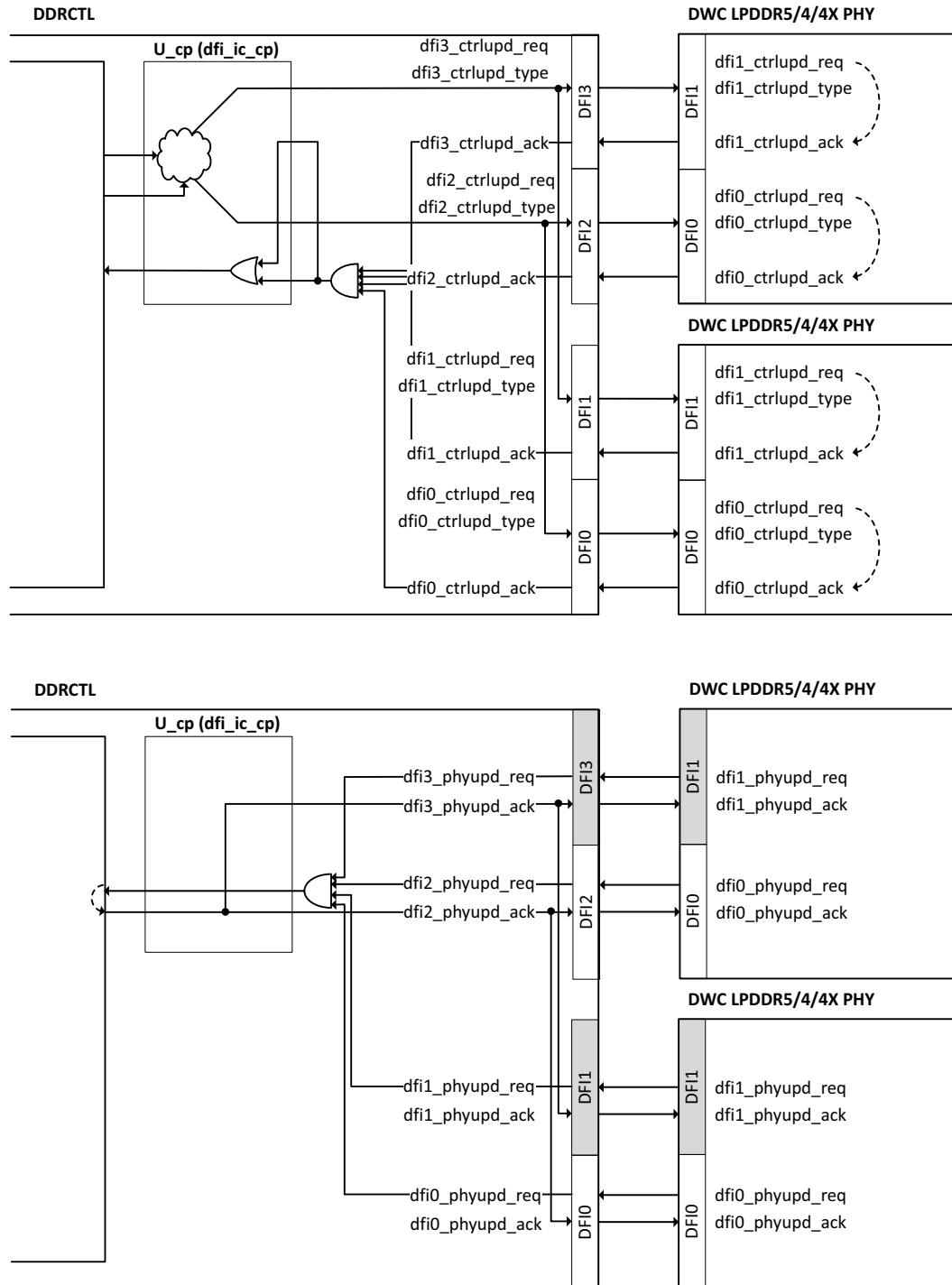


Figure 4-10 shows DDRCTL to PHY connections for Update Interface.

Figure 4-10 Update Interface

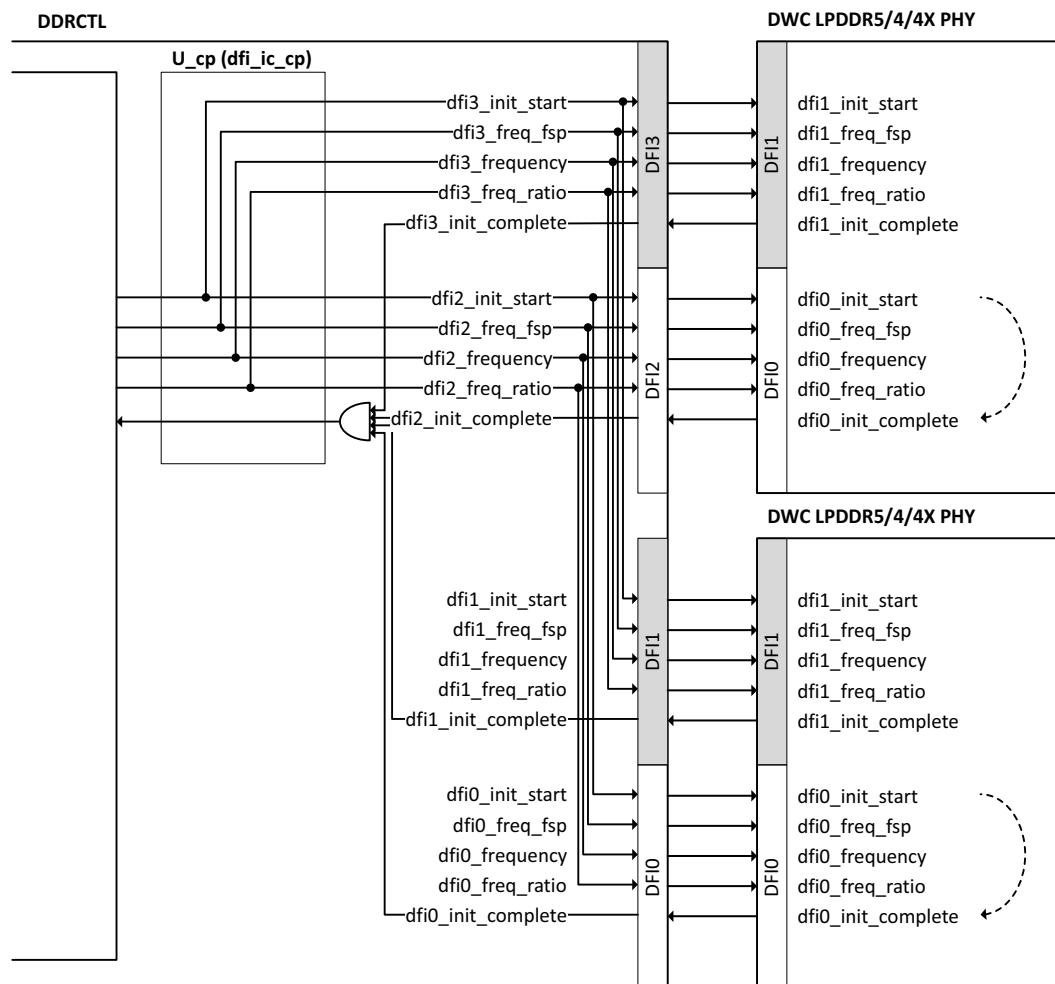


Note

- When DDRCTL_PPT2 is not defined or PPT2CTRL0.ppt2_en is 0, dfi1_ctrlupd_ack is unused in DDRCTL, and dfi1_ctrlupd_req is the same as dfi0_ctrlupd_req.
- dfi*_ctrlupd_ack signals from all DFI are internally ANDed, that is, the controller completes the control update when all acknowledge signals are asserted.
- dfi*_phyupd_req signals from all DFI are internally ANDed, that is, the controller accepts the PHY update when all request signals are asserted.

Figure 4-11 shows DDRCTL to PHY connections for Status Interface.

Figure 4-11 Status Interface

**Note**

- dfi*_i_init_complete signals from all DFI are internally ANDed, that is, the controller accepts the INIT complete when all complete signals are asserted.

Figure 4-12 shows DDRCTL to PHY connections for low-power control interface.

Figure 4-12 Low-Power Control Interface

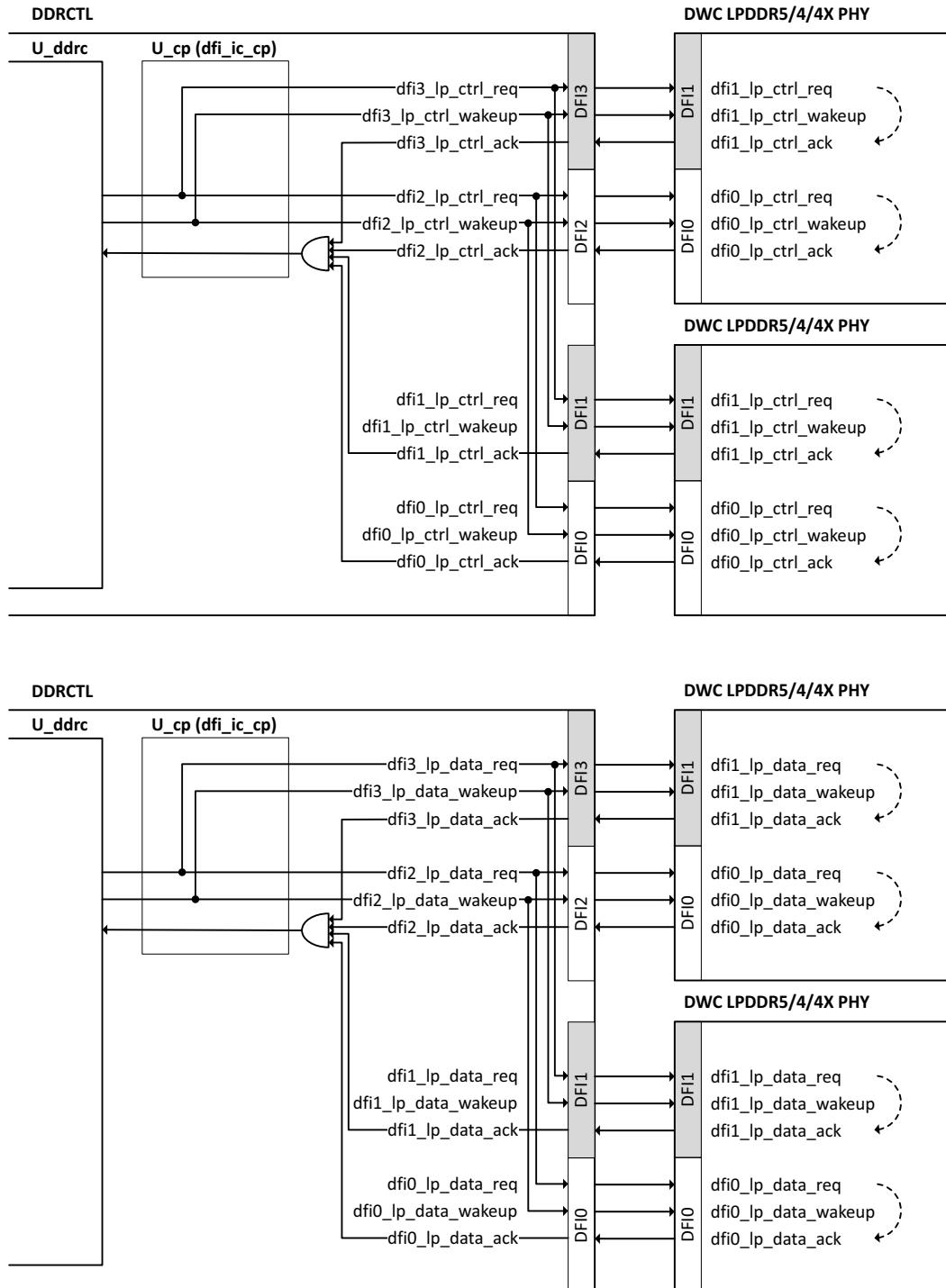
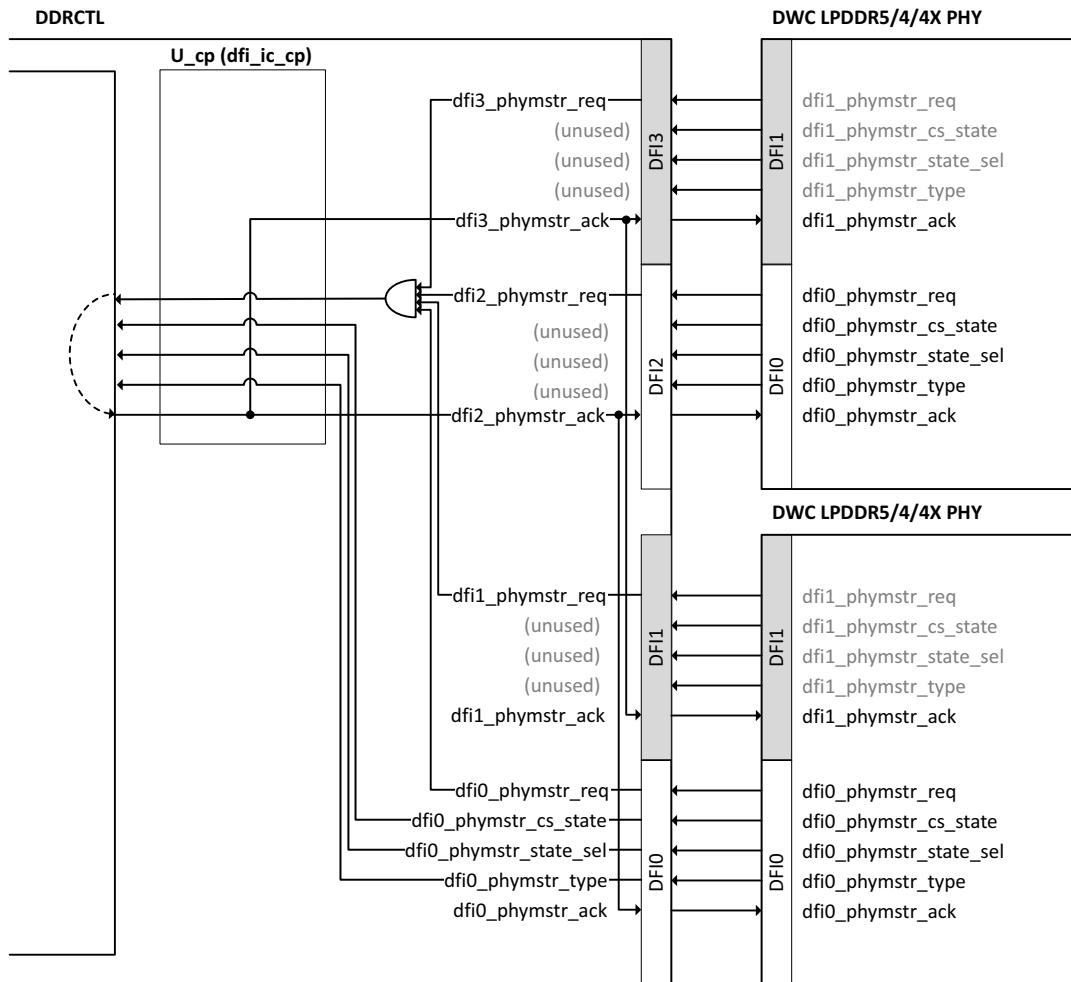


Figure 4-13 shows DDRCTL to PHY connections for PHY Master Interface.

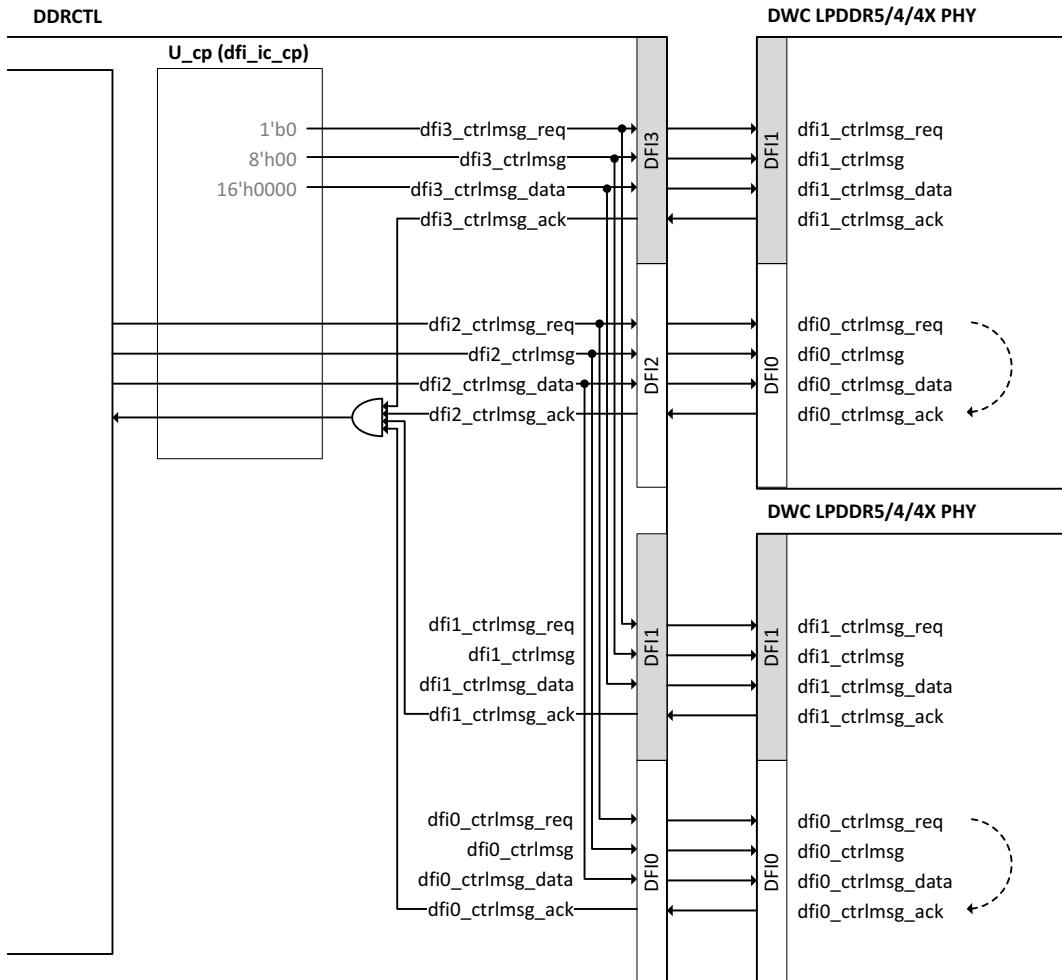
Figure 4-13 PHY Master Interface



Note dfi*_phymstr_req signals from all DFI are internally ANDed, that is, the controller accepts the PHY master when all request signals are asserted.

Figure 4-14 shows DDRCTL to PHY connections for MC PHY Message Interface.

Figure 4-14 MC PHY Message Interface



Note d_i*_ctrlmsg_ack signals from all DFI are internally ANDed, that is, the controller completes the control message when all acknowledge signals are asserted.

5

Channel Modes

This chapter contains the following sections:

- “[Overview of LPDDR4 / LPDDR5 Channel Modes](#)” on page 126
- “[DDRCTL Configurations](#)” on page 127

5.1 Overview of LPDDR4 / LPDDR5 Channel Modes

LPDDR4 SDRAM consists of two independent channels, and Synopsys DDR PHY has two sets of DFI interfaces. LPDDR 5/4 controller has three sets of DFI interfaces, called “DFI channels” in this document. Each DFI channel is to connect to its respective SDRAM channel (that is, DFI channel 0 connects to SDRAM channel A, DFI channel 1 connects to SDRAM channel B).

In terms of memory/DFI topology, Single DDRC Single DFI configuration, Single DDRC Dual DFI configuration, and Single DDRC Quad DFI configuration are supported, as shown in [Figure 5-1](#) on page 128, [Figure 5-4](#) on page 131, [Figure 5-7](#) on page 137, [Figure 5-8](#) on page 138, and [Figure 5-11](#) on page 144.

5.2 DDRCTL Configurations

The DDRCTL controller currently supports the following hardware configurations:

1. Single DDRC Single DFI configuration
2. Single DDRC Dual DFI configuration
3. Single DDRC Quad DFI configuration

[Table 5-1](#) shows the relationship between `MEMC_DRAM_DATA_WIDTH` and `UMCTL2_NUM_DFI`.

Table 5-1 Relationship Between `MEMC_DRAM_DATA_WIDTH` and `UMCTL2_NUM_DFI`

<code>MEMC_DRAM_DATA_WIDTH</code>	<code>UMCTL2_NUM_DFI</code>	Notes
16	1	Single DDRC Single DFI configuration
32	2	Single DDRC Dual DFI configuration
64	4	Single DDRC Quad DFI configuration



The hardware parameter `UMCTL2_NUM_DFI` is automatically selected.

This section contains the following subsections:

- “[Single DDRC Single DFI Configuration](#)” on page [128](#)
- “[Single DDRC Dual DFI Configuration](#)” on page [131](#)
- “[Single DDRC Quad DFI Configuration](#)” on page [143](#)

5.2.1 Single DDRC Single DFI Configuration

Figure 5-1 shows the single DDRC, single DFI Configuration.



Only FBW (Full Bus Width) Mode is supported in this configuration.

Figure 5-1 Single DDRC Single DFI Configuration

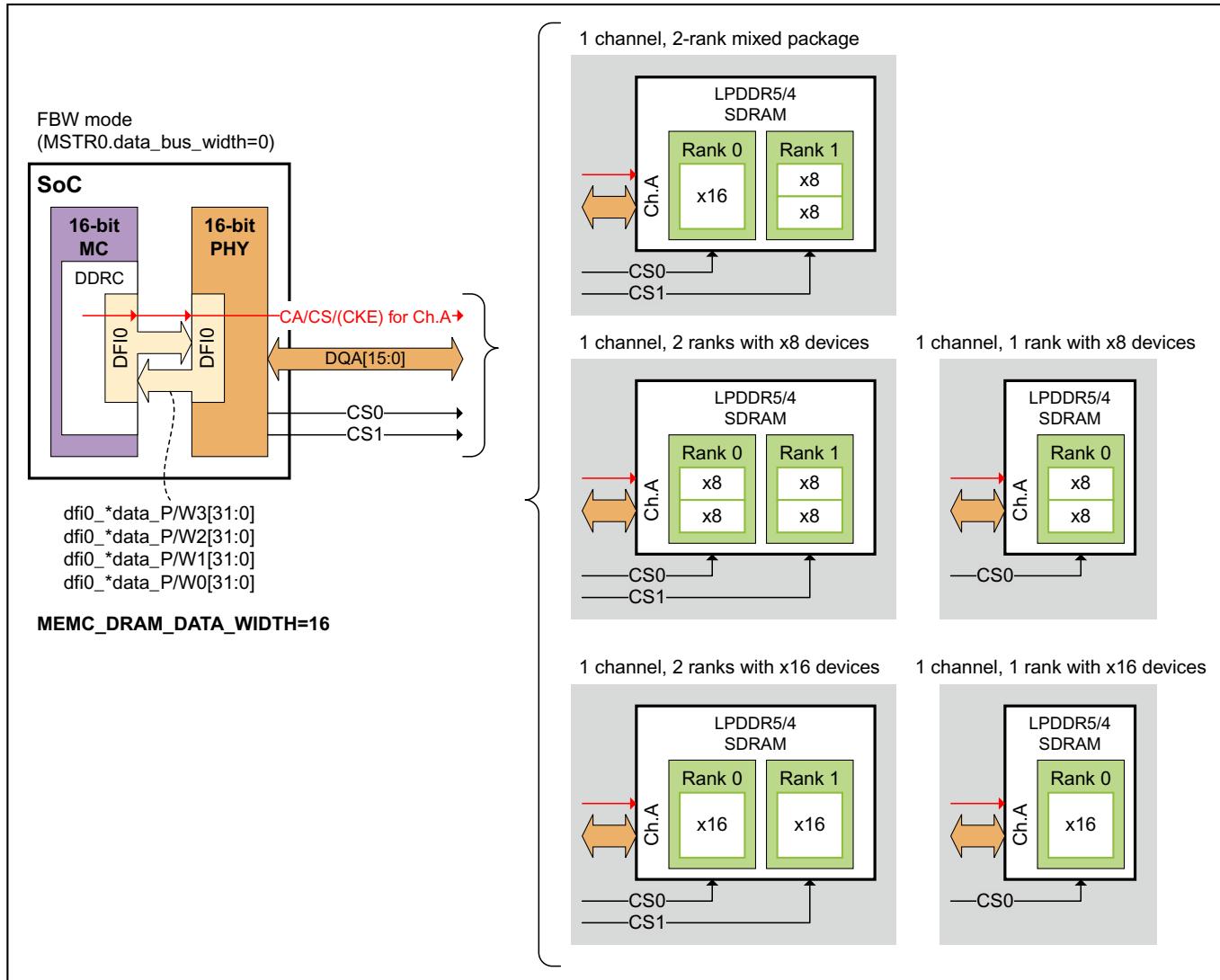


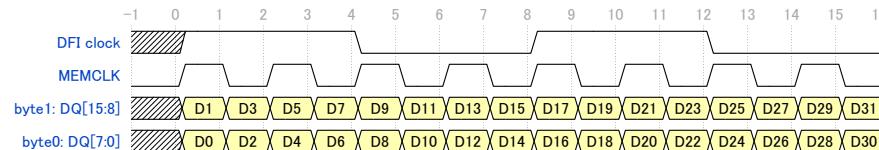
Figure 5-2 shows the DFI Clocks/Data of single DDRC, single DFI configuration for 1:4/1:1:4 frequency ratio mode.

Figure 5-2 DFI Clocks/Data of Single DDRC Single DFI Configuration for 1:4/1:1:4 Frequency Ratio Mode

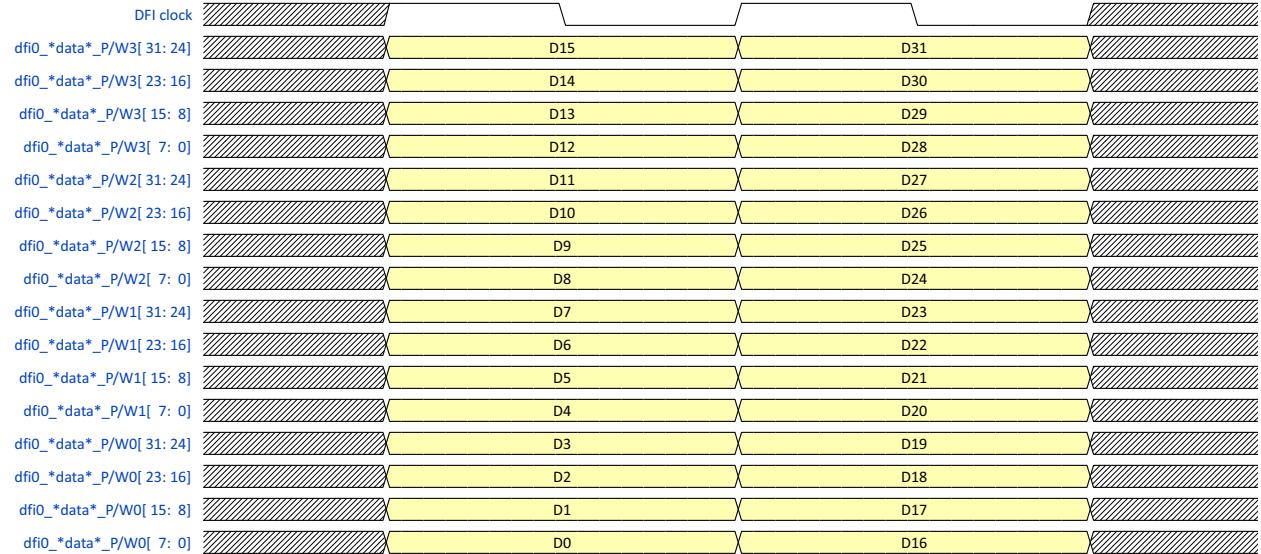
TMGCFG.frequency_ratio=1 (DFI 1:4/1:1:4 frequency ratio mode)

MSTRO.data_bus_width=0 (Full Bus Width mode)

SDRAM



DFI



HIF

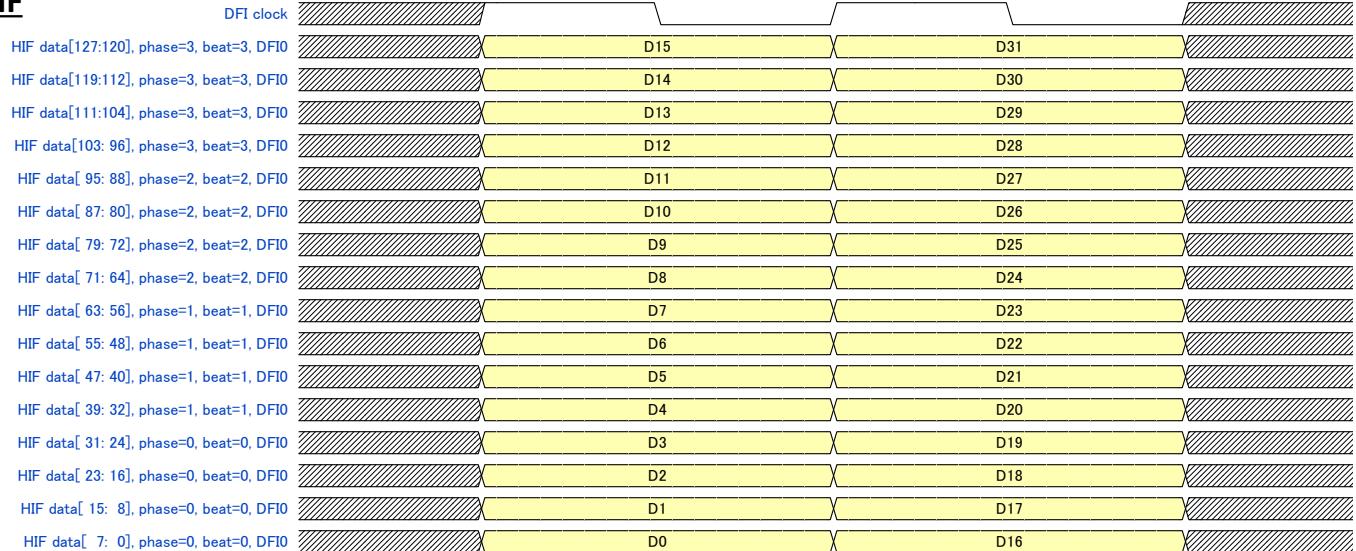
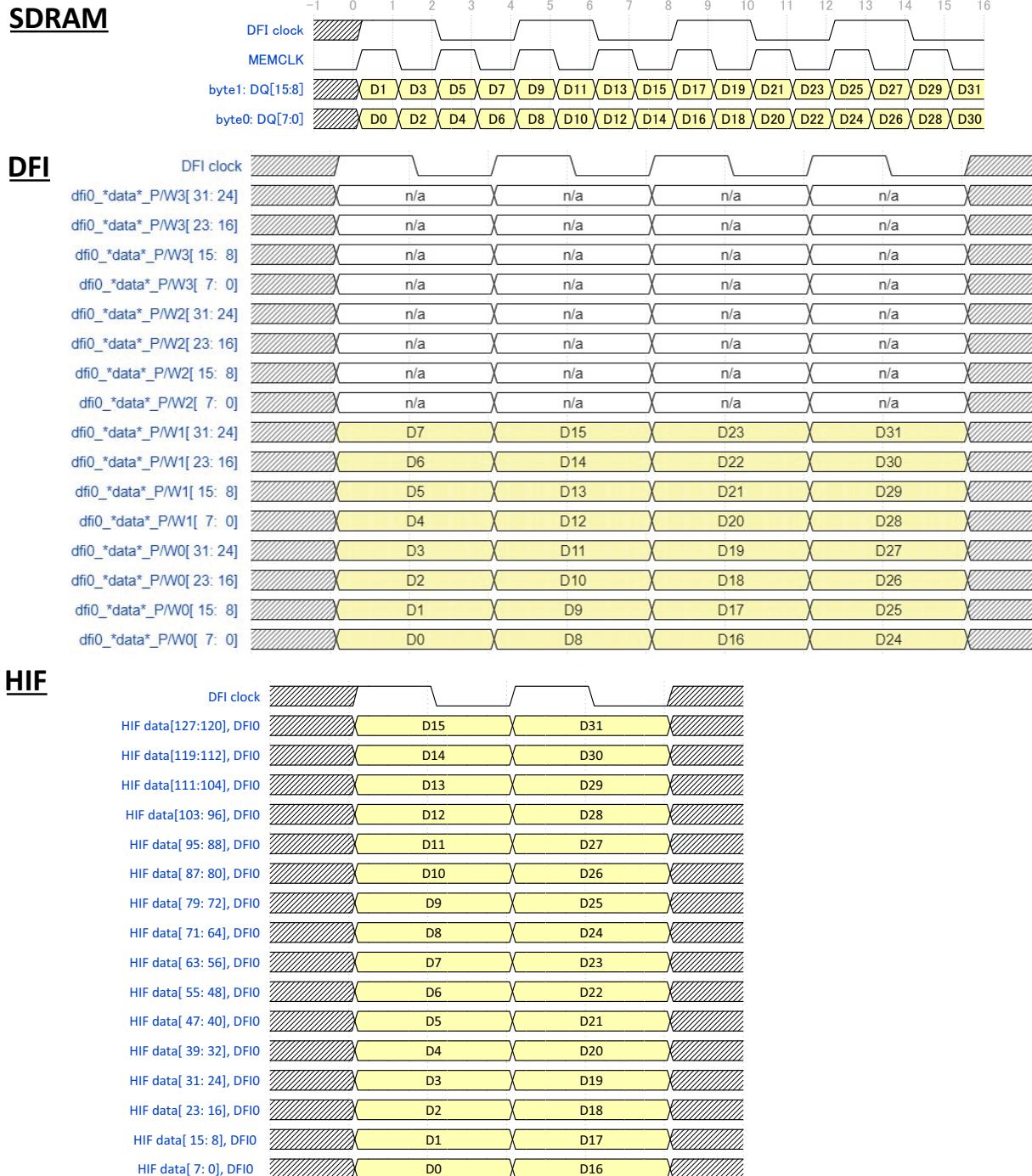


Figure 5-3 shows the DFI clocks of single DDRC, single DFI configuration for 1:2/1:1:2 frequency ratio mode.

Figure 5-3 DFI Clocks/Data of Single DDRC Single DFI Configuration (1:2/1:1:2 Frequency Ratio Mode)

TMGCFG.frequency_ratio=0 (DFI 1:2/1:1:2 frequency ratio mode)
MSTR0.data_bus_width=0 (Full Bus Width mode)





For HIF write data (hif_wdata), "n/a" shown in the diagrams needs to be set to '0'. That is, invalid data of hif_wdata have to be set to '0'.
For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

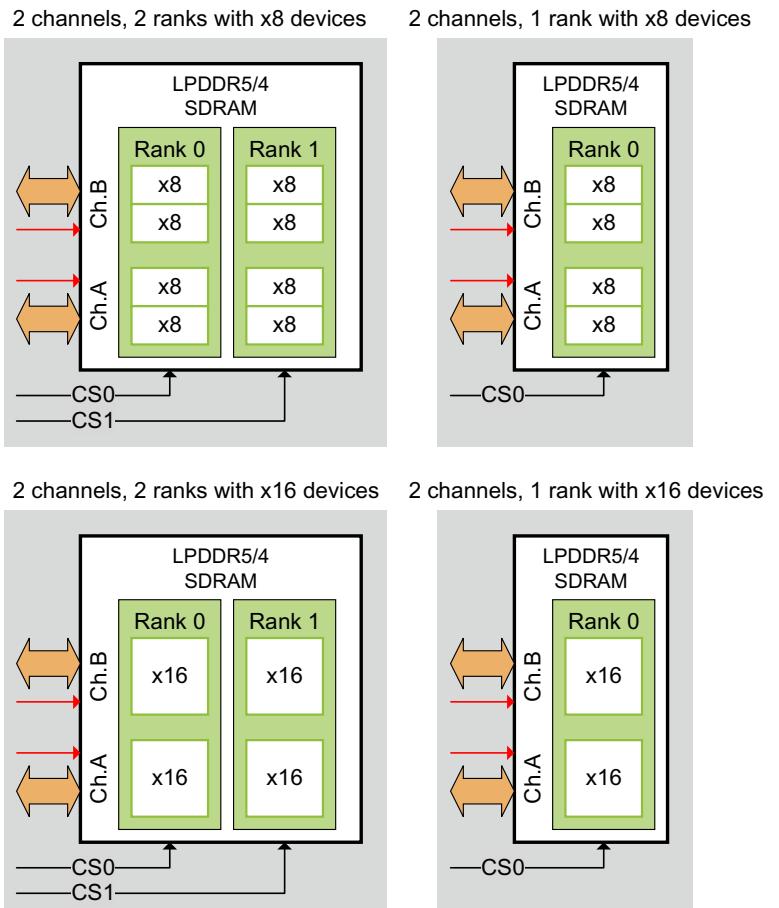
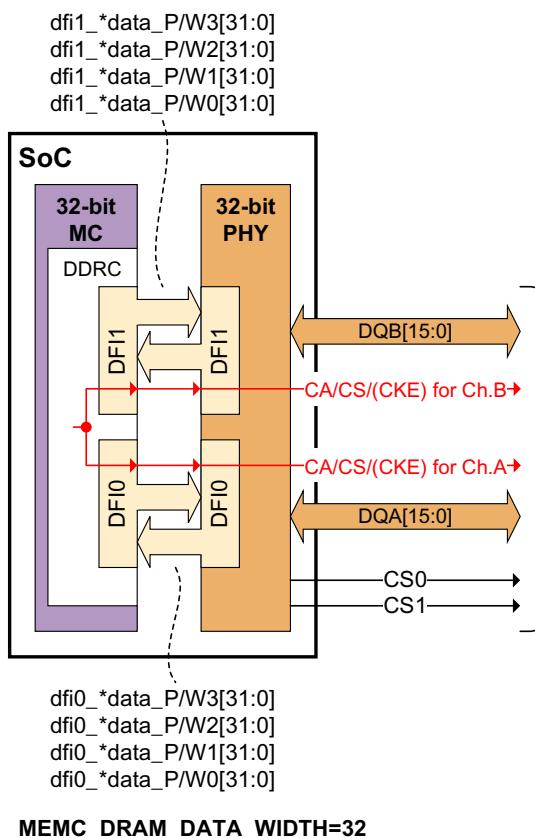
5.2.2 Single DDRC Dual DFI Configuration

5.2.2.1 FBW (Full Bus Width) Mode

Figure 5-4 shows the Single DDRC, Dual DFI configuration.

Figure 5-4 Single DDRC Dual DFI Configuration

FBW mode
(MSTR0.data_bus_width=0)

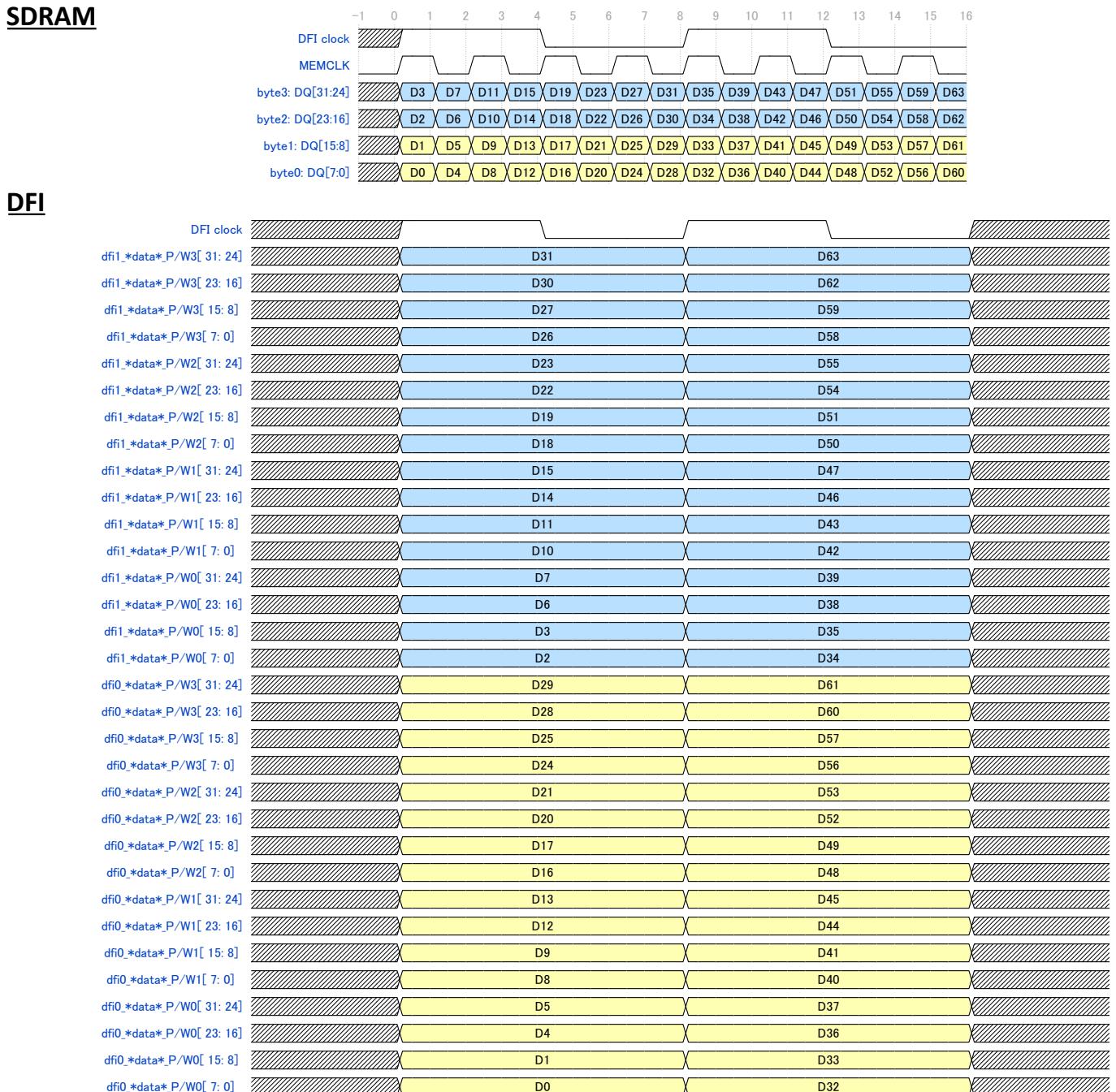


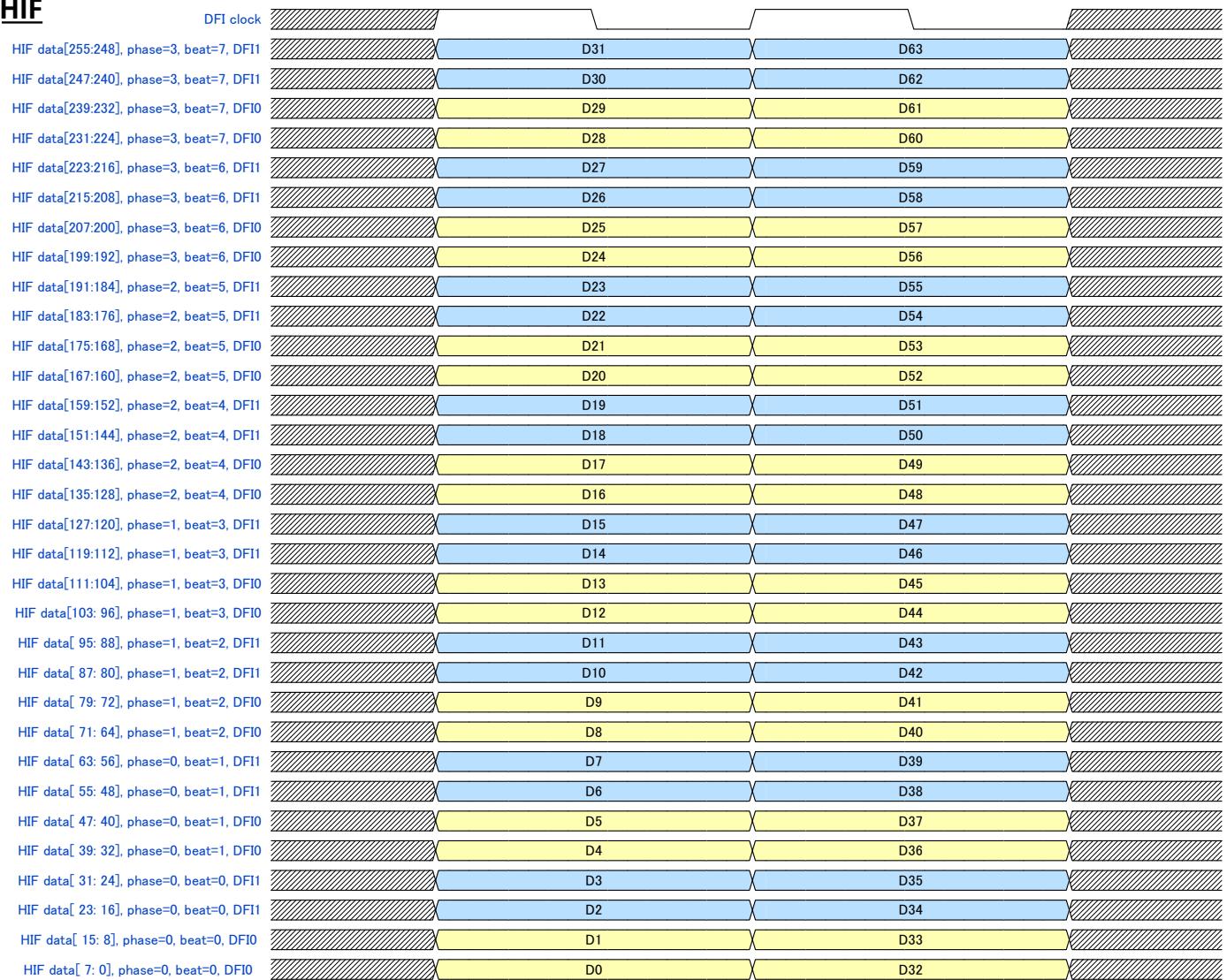
[Figure 5-5](#) shows the DFI clocks/data of Single DDRC, Dual DFI configuration for 1:4/1:1:4 frequency ratio mode.

Figure 5-5 DFI Clocks/Data of Single DDRC Dual DFI Configuration for 1:4/1:1:4 Frequency Ratio Mode

TMGCFG.frequency_ratio=1 (DFI 1:4/1:1:4 frequency ratio mode)

MSTRO. data_bus_width=0 (Full Bus Width mode)



HIF

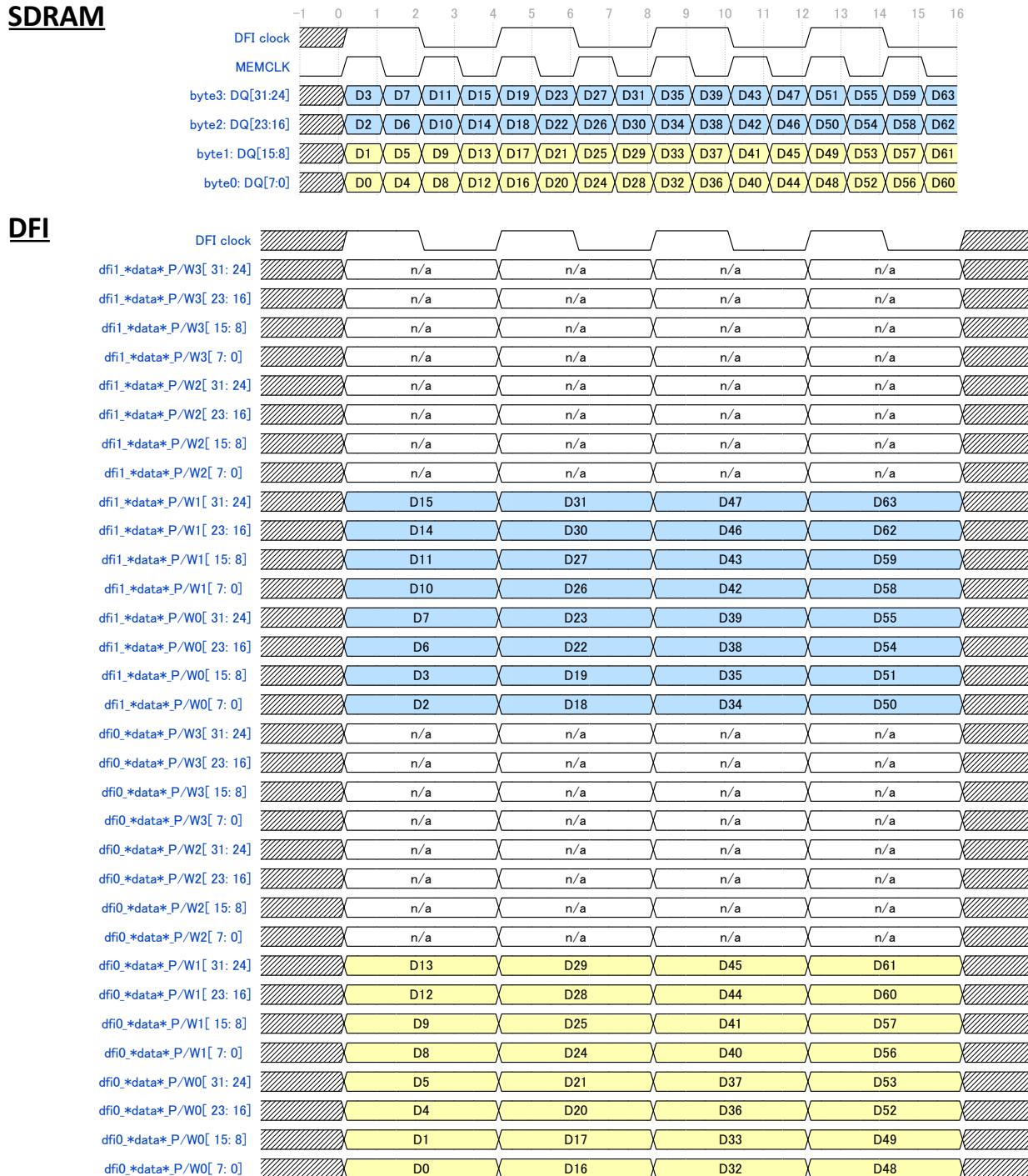
Note For HIF write data (hif_wdata), "n/a" shown in the diagrams needs to be set to '0'. That is, invalid data of hif_wdata have to be set to '0'.

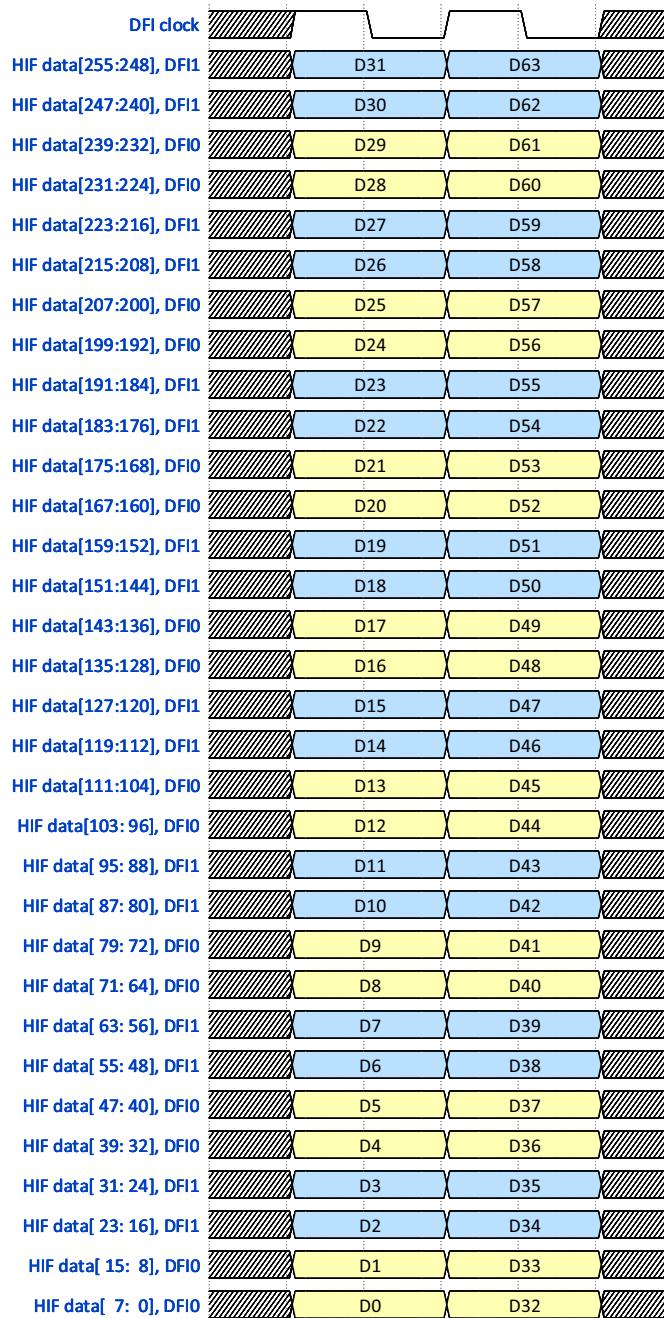
For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

Figure 5-6 shows the DFI clocks/data of Single DDRC, Dual DFI configuration for 1:2/1:1:2 frequency ratio mode.

Figure 5-6 DFI Clocks/Data of Single DDRC Dual DFI Configuration (1:2/1:1:2 Frequency Ratio Mode)

TMGCFG.frequency_ratio=0 (DFI 1:2/1:1:2 frequency ratio mode)
MSTRO. data_bus_width=0 (Full Bus Width mode)



HIF

Note For HIF write data (hif_wdata), "n/a" shown in the diagrams needs to be set to '0'. That is, invalid data of hif_wdata have to be set to '0'.

For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

5.2.2.2 HBW (Half Bus Width) Mode

LPDDR5/4/4X Controller operates either in FBW mode (`MSTR0.data_bus_width=0`) or in HBW mode (`MSTR0.data_bus_width=1`). If HBW mode is selected, the relevant signals for DFI1 in the controller are all driven into inactive state (set to '0'):

- `dfi1_address_P*1`
- `dfi1_cke_P*1`
- `dfi1_cs_P*1`
- `dfi1_ctrlupd_req`
- `dfi1_ctrlupd_type`
- `dfi1_dram_clk_disable_P*1`
- `dfi1_freq_fsp*1`
- `dfi1_freq_ratio`
- `dfi1_frequency`
- `dfi1_init_start`
- `dfi1_lp_ctrl_req`
- `dfi1_lp_ctrl_wakeup`
- `dfi1_lp_data_req`
- `dfi1_lp_data_wakeup`
- `dfi1_phymstr_ack`
- `dfi1_phyupd_ack`
- `dfi1_rddata_cs_P*1`
- `dfi1_rddata_en_P*1`
- `dfi1_wck_cs_P*1`
- `dfi1_wck_en_P*1`
- `dfi1_wck_toggle_P*1`
- `dfi1_wrdata_cs_P*1`
- `dfi1_wrdata_en_P*1`
- `dfi1_wrdata_ecc_P*1`
- `dfi1_wrdata_mask_P*1`
- `dfi1_wrdata_P*1`
- `dwc_lpddr5xphy1_snoop_en_P*1`
- `dwc_lpddr5xphy1_snoop_osc_running`

If a 32-bit controller and 32-bit PHY are implemented together in a SoC, and the same SoC needs to be used in different products (i.e. one for 16-bit SDRAM, and the other for 32-bit SDRAM), this requirement is satisfied logically by setting `MSTR.data_bus_width` and `PHY register` according to SDRAM data width.

- `MSTR.data_bus_width=0` (FBW mode) for 32-bit SDRAM

1. The value of phase * is 0~3

- MSTR.data_bus_width=1 (HBW mode) for 16-bit SDRAM

Depending on the application, power saving for the DFI1 of the 32-bit PHY in case of HBW mode may need to be considered, but this is outside the scope of controller functionality.

This is supposed to be used only for specific cases. For example, when a 16-bit PHY is implemented only to minimize the footprint in test chip. In such cases, mass production chip must use 32-bit PHY and 32-bit controller, but if each configuration parameter in the controller is identical between mass production chip and test chip, basic functionality of the 32-bit controller can be validated briefly with 16-bit PHY by using the test chip.

[Figure 5-7](#) and [Figure 5-8](#) show a Single DDRC Dual DFI configuration (32-bit) with 16-bit PHY and 32-bit PHY in HBW mode, respectively.

Figure 5-7 Single DDRC Dual DFI Configuration (32-bit) With 16-bit PHY in HBW mode

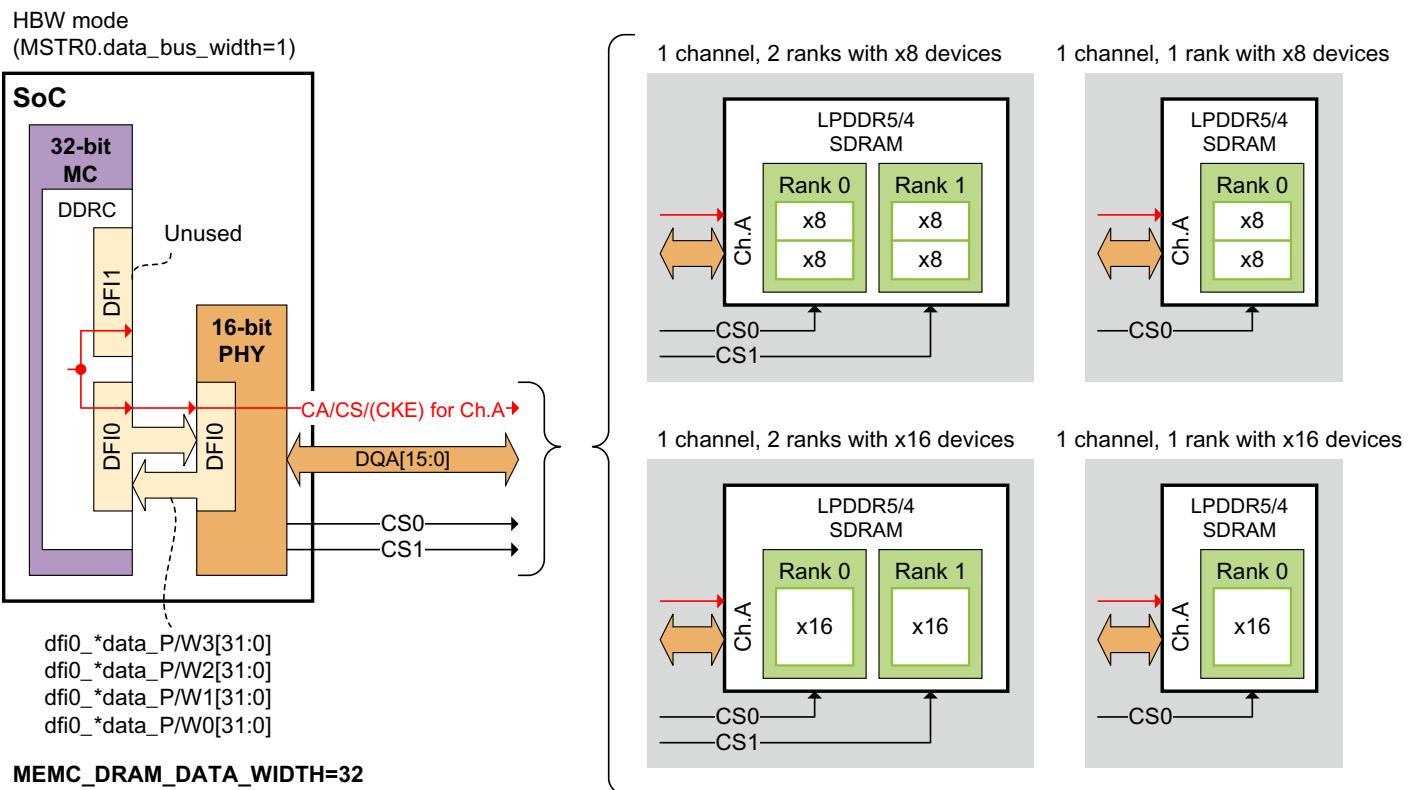


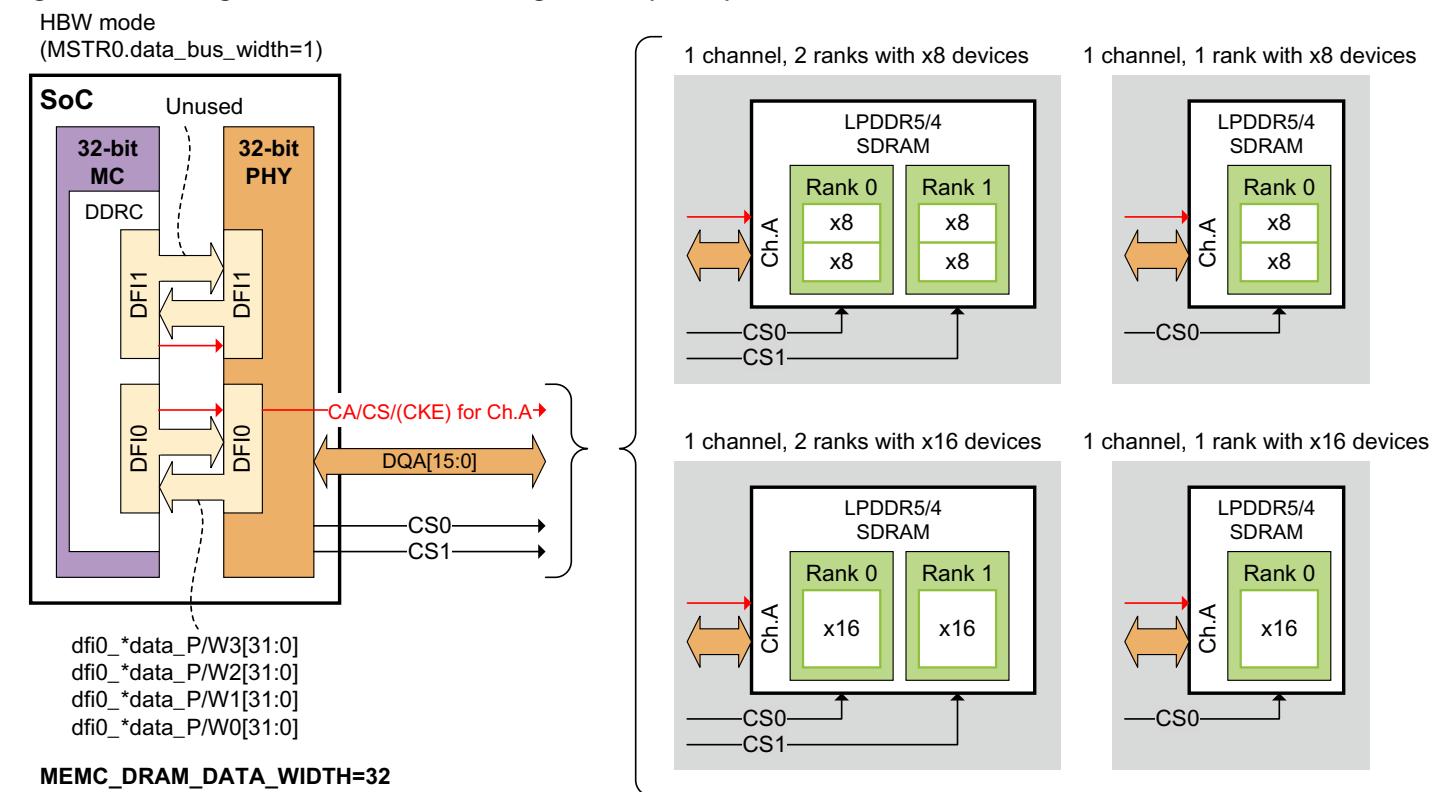
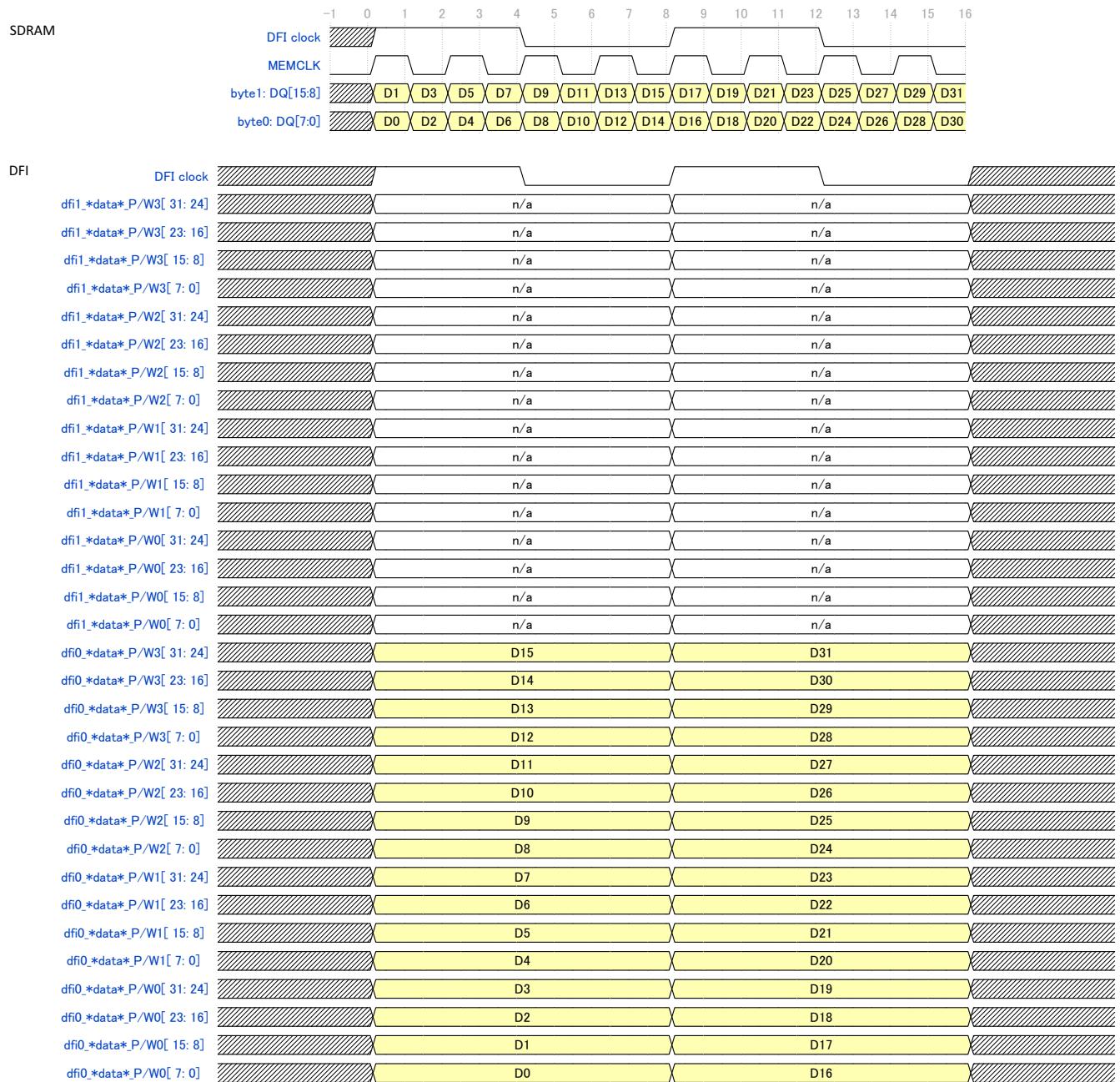
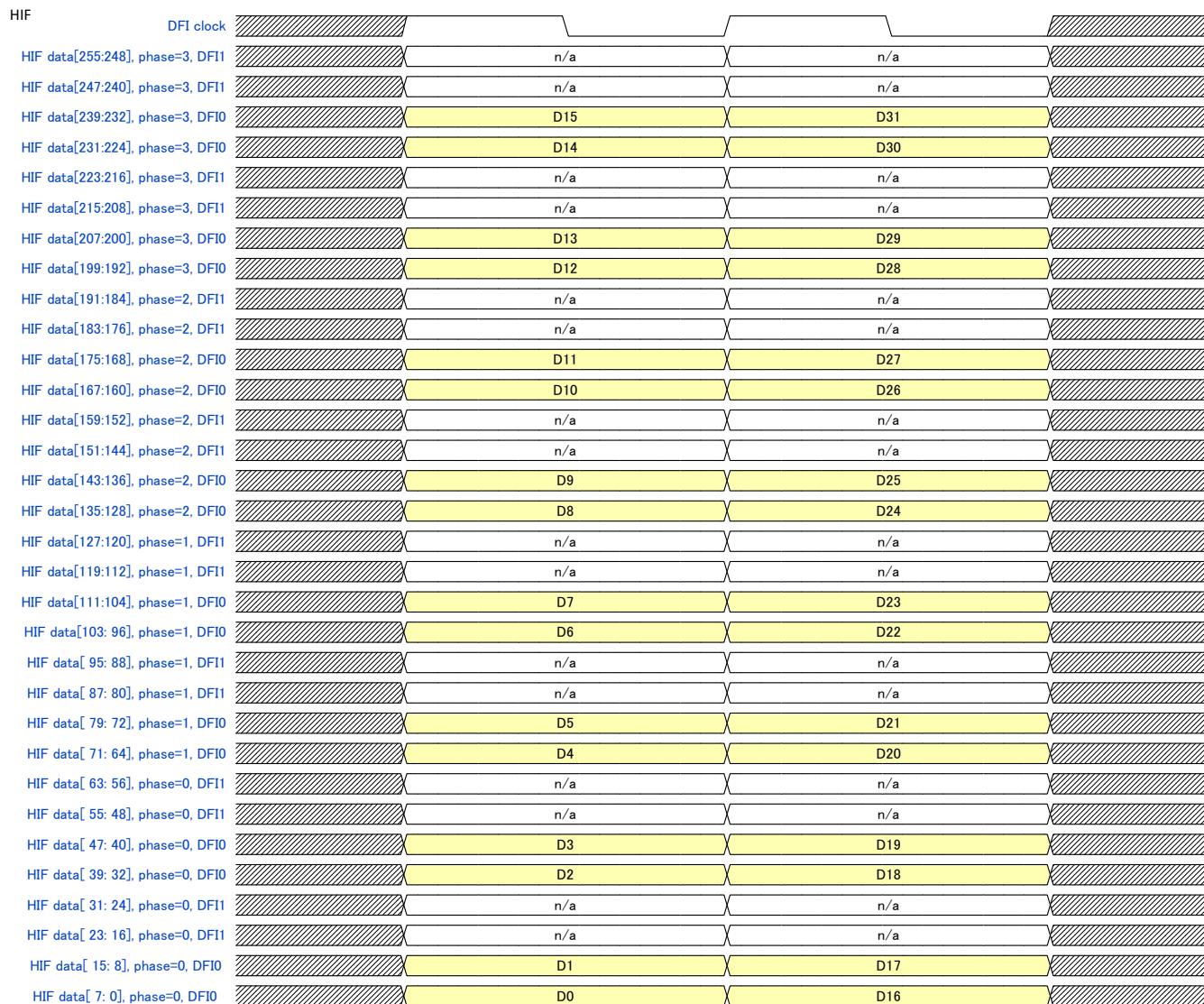
Figure 5-8 Single DDRC Dual DFI Configuration (32-bit) With 32-bit PHY in HBW mode

Figure 5-9 shows the DFI clocks/data of Single DDRC Dual DFI configuration for 1:4/1:1:4 frequency ratio mode.

Figure 5-9 DFI Clocks/Data of Single DDRC Dual DFI Configuration (1:4/1:1:4 Frequency Ratio Mode)

TMGCFG.frequency_ratio=1 (DFI 1:4/1:1:4 frequency ratio mode)
MSTRO.data_bus_width=1 (Half Bus Width mode)





For HIF write data (hif_wdata), "n/a" shown in the diagrams needs to be set to '0'. That is, invalid data of hif_wdata have to be set to '0'.

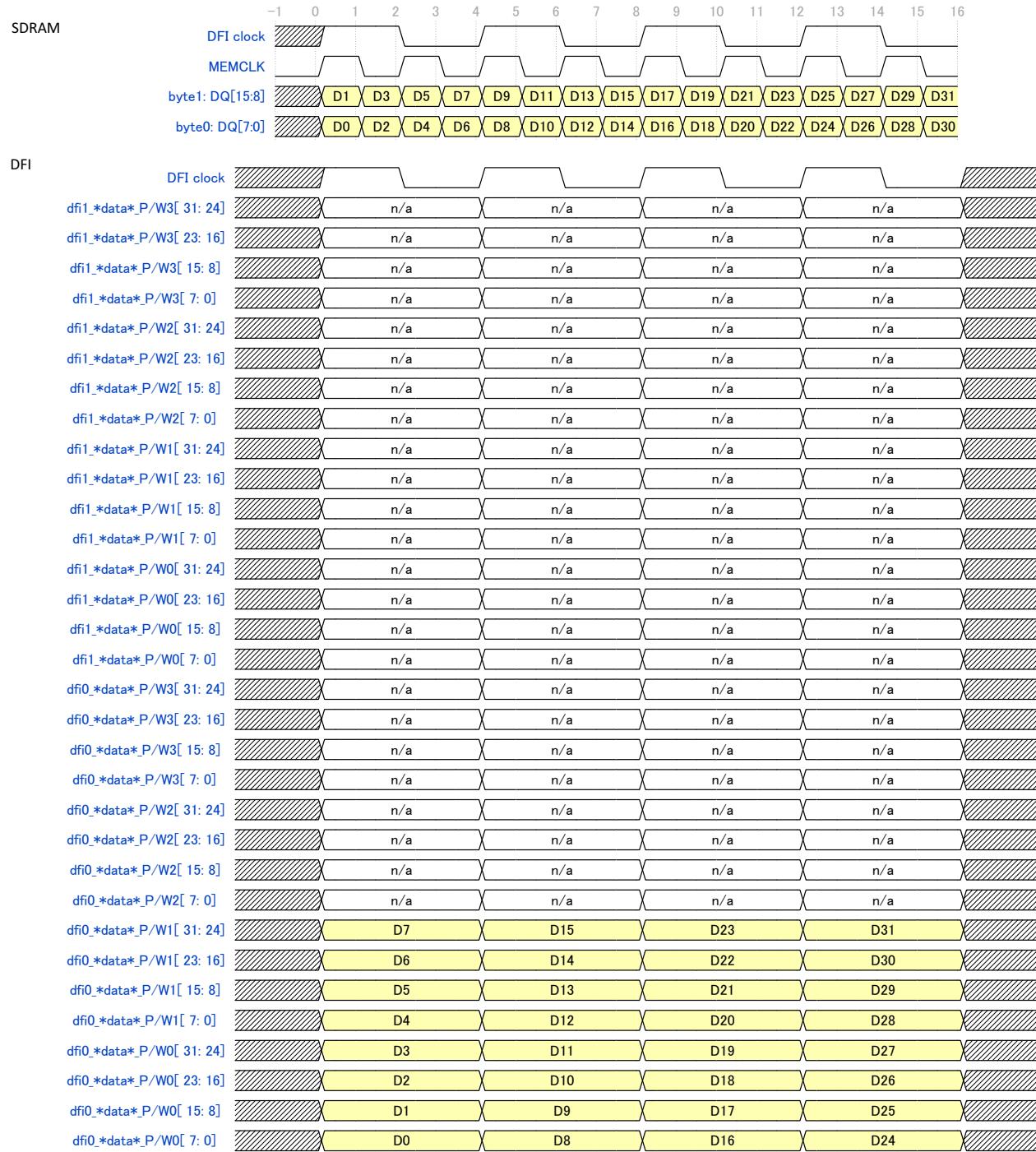
For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

Figure 5-10 shows the DFI clocks/data of Single DDRC Dual DFI configuration for 1:2/1:1:2 frequency ratio mode.

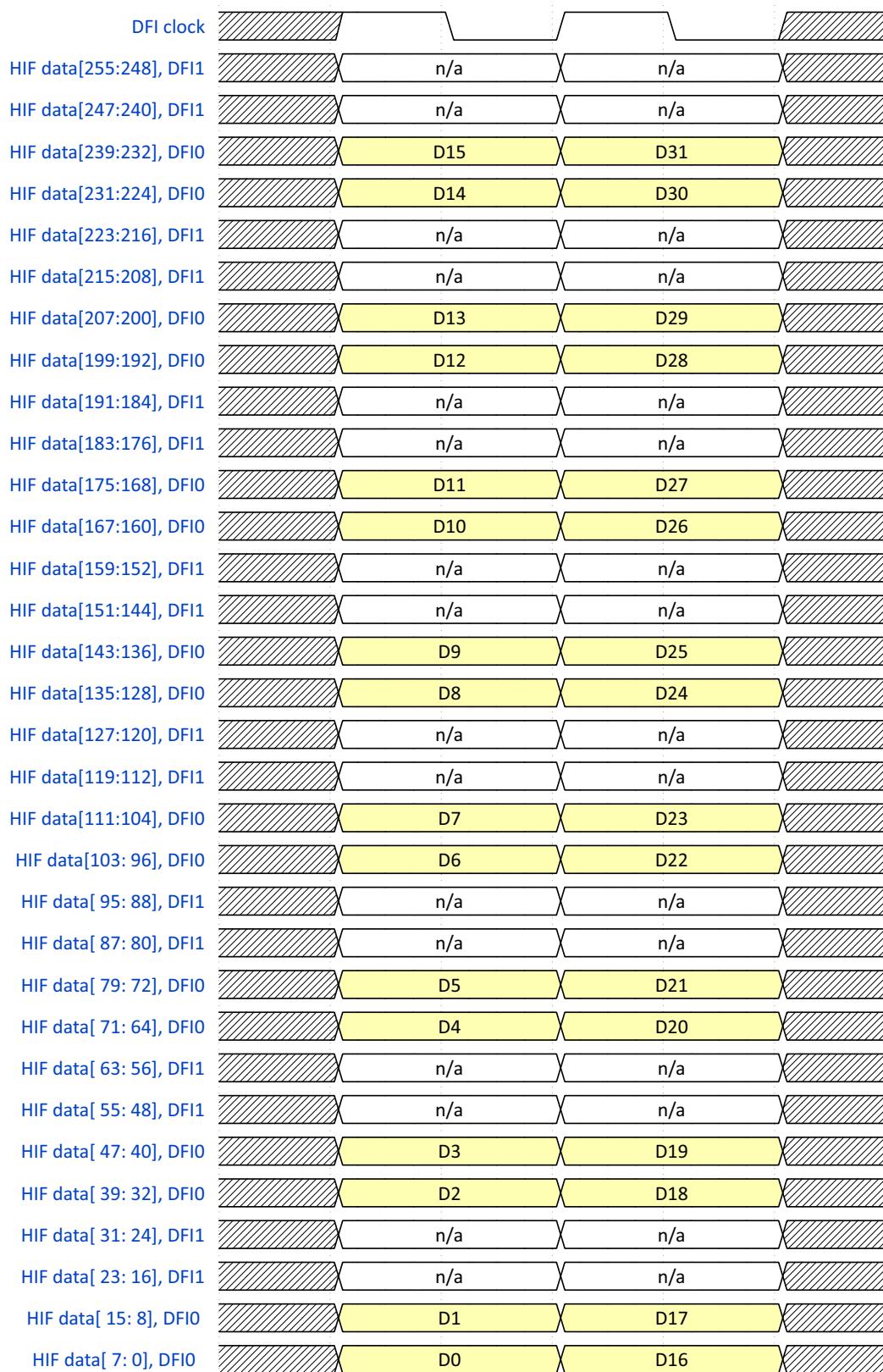
Figure 5-10 DFI Clocks/Data of Single DDRC Dual DFI Configuration (1:2/1:1:2 Frequency Ratio Mode)

TMGCFG.frequency_ratio=0 (DFI 1:2/1:1:2 frequency ratio mode)

MSTRO.data_bus_width=1 (Half Bus Width mode)



HIF





Note For HIF write data (hif_wdata), "n/a" shown in the diagrams needs to be set to '0'. That is, invalid data of hif_wdata have to be set to '0'.

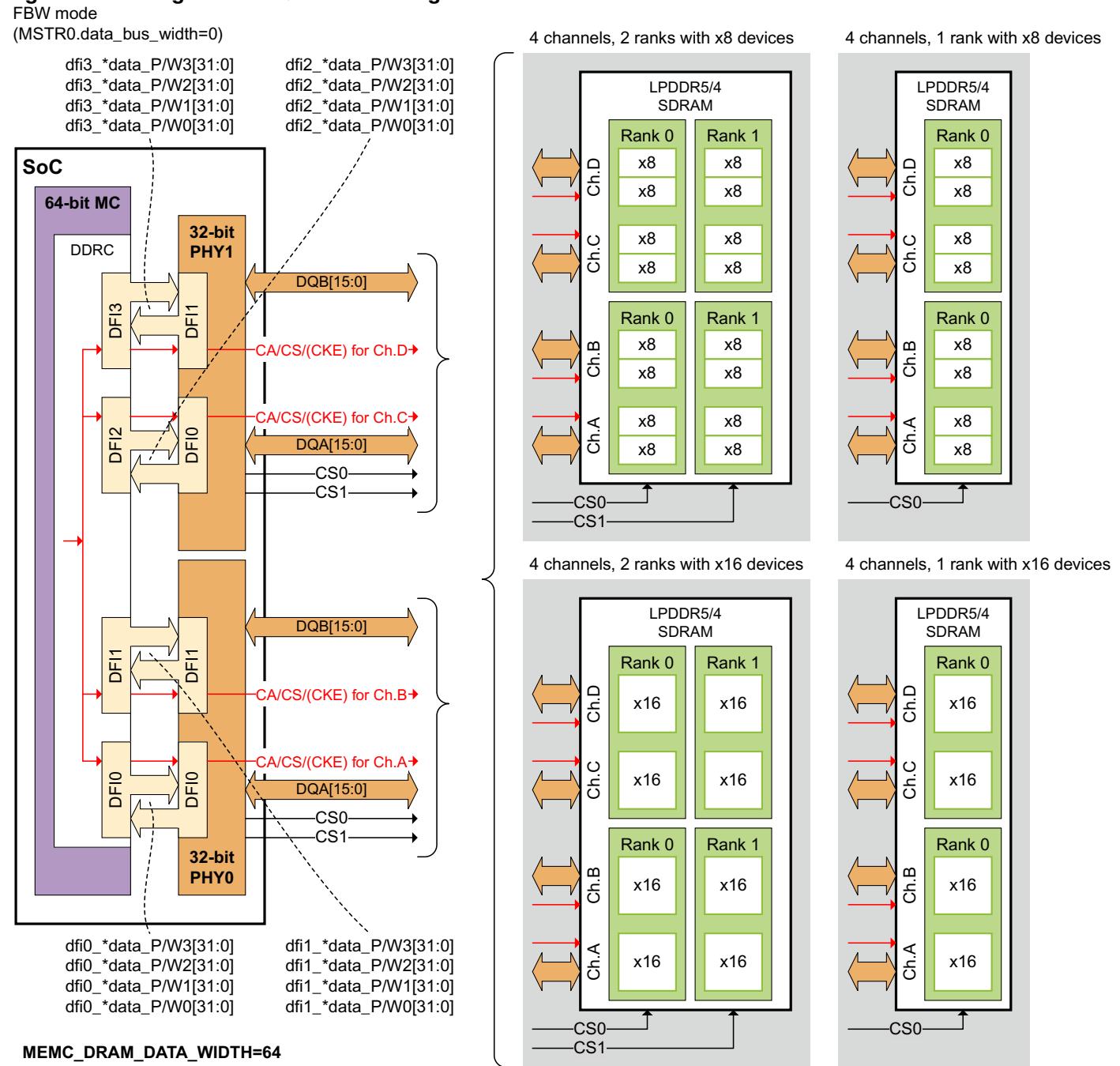
For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

5.2.3 Single DDRC Quad DFI Configuration

This feature is supported only with limited hardware configurations. At least, the following configuration parameter conditions need to be satisfied.

- DDRCTL_SYS_INTF == 1 (AXI)
- UMCTL2_PORT_DW_* == 256
- DDRCTL_PPT2 == 0
- UMCTL2_DUAL_HIF == 0
- MEMC_ECC_SUPPORT == 0
- UMCTL2_SBR_EN == 0
- UMCTL2_OCPAR_EN == 0
- UMCTL2_OCECC_EN == 0
- DDRCTL_OCSAP_EN == 0
- UMCTL2_OCCAP_EN == 0
- UMCTL2_FREQUENCY_NUM == 1
- UMCTL2_HWFFC_EN == 0

[Figure 5-11](#) on page [144](#) shows the single DDRC quad DFI configuration.

Figure 5-11 Single DDRC Quad DFI Configuration

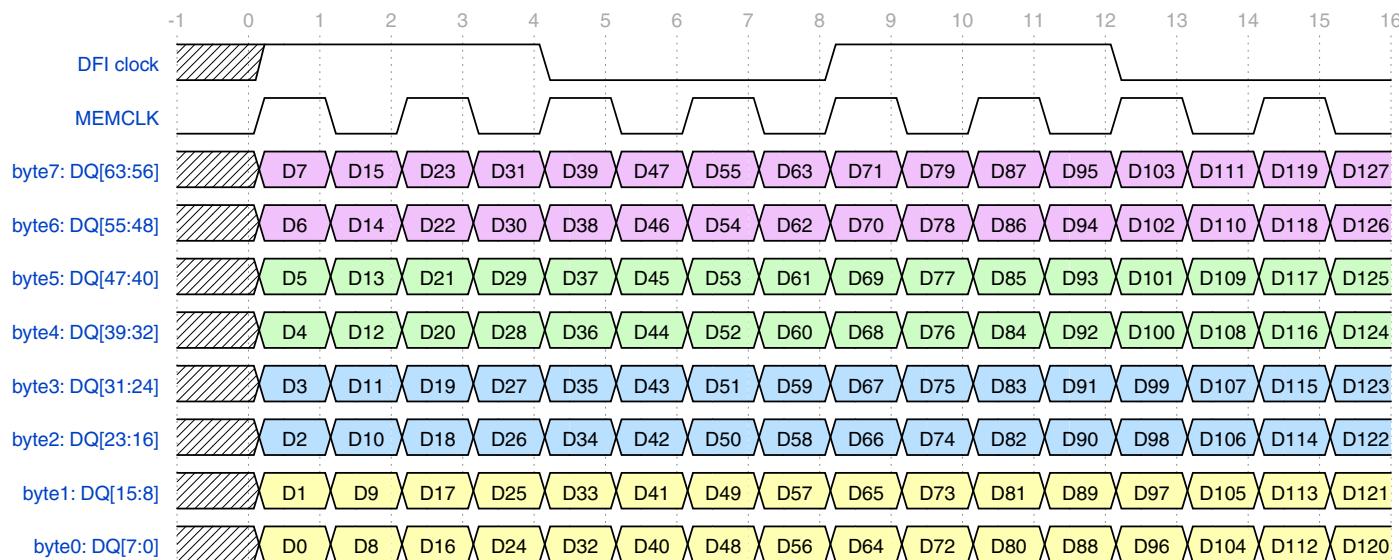


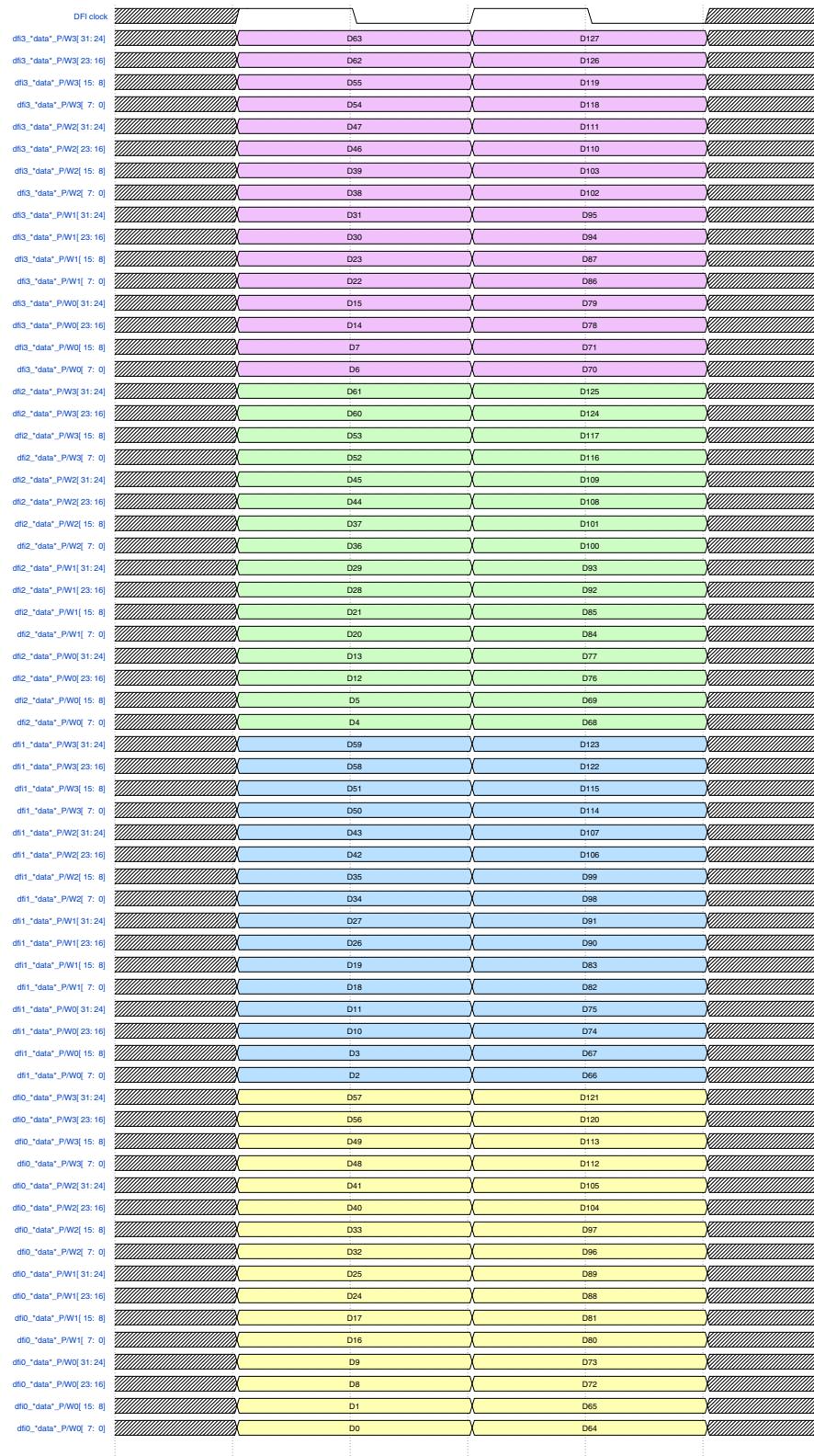
- This is supported only with limited configuration. For more information, contact Synopsys.
- Only FBW (Full Bus Width) Mode is supported in this configuration.
- If a 64-bit controller (Single DDRC Quad DFI configuration) is implemented with multiple PHY instances, the same PHY type and the same setting for all PHY instances is a must.
- If multiple PHY instances are used, DRAM device type must be the same for all PHY instances.
- The DDRCTL does not have de-skew mechanism for DFI read data path between different PHY instances, so this must be treated outside of the DDRCTL. It is required that all external PHYs operate in lock-step mode so they appear to the DDRCTL as a single PHY. The DDRCTL can use only one `dfi_rddata_valid` signal from any of the PHYs to indicate DFI read data is valid for all the PHYs. The DDRCTL requires the DFI read data to be presented in a unified fashion: all data bytes must be presented as valid in the same DFI phase even when originating from just one PHY.

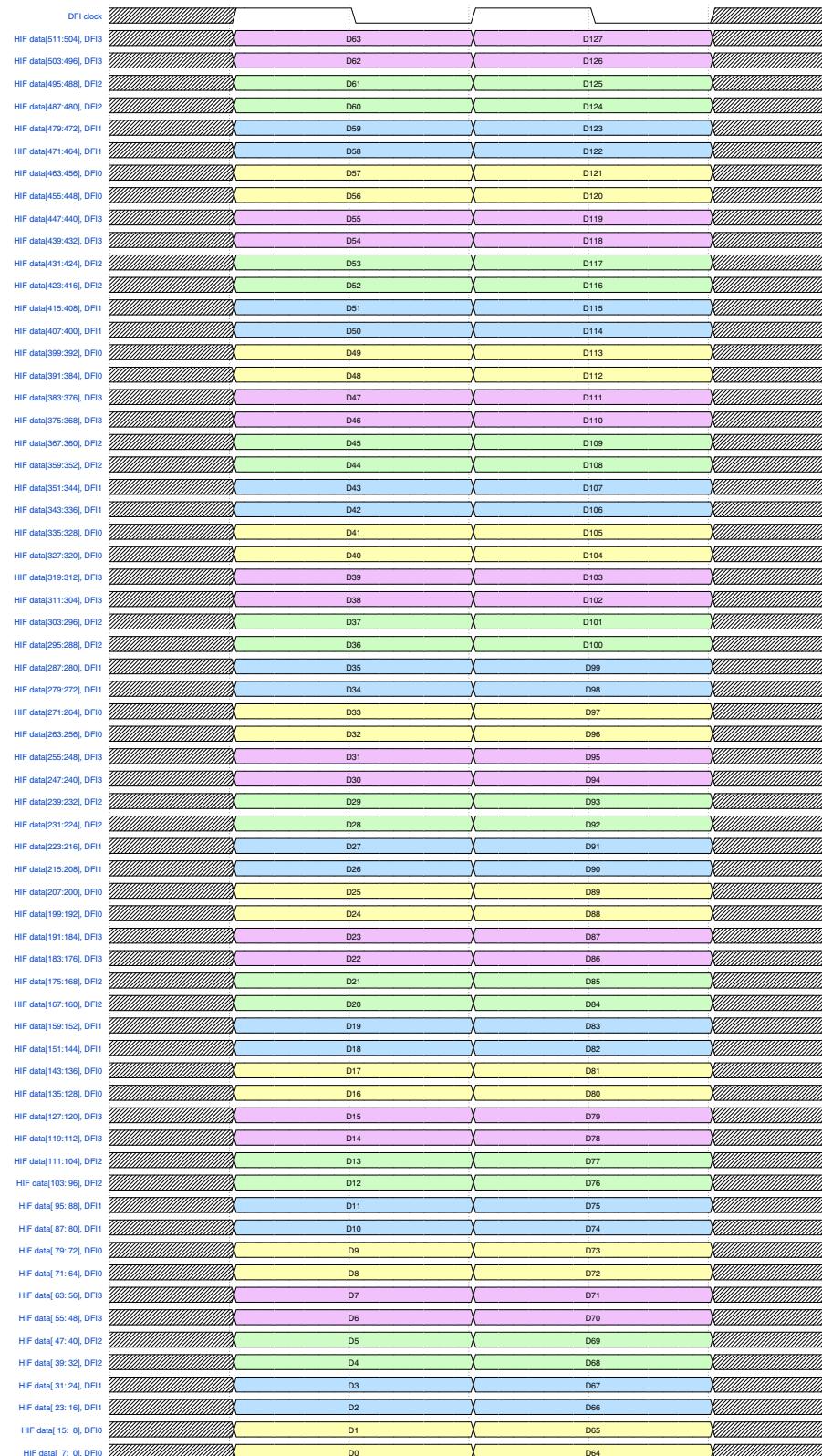
[Figure 5-12](#) on page [145](#) shows the DFI Clocks / Data of Single DDRC Quad DFI Configuration for 1:4/1:1:4 frequency ratio mode.

Figure 5-12 DFI Clocks / Data of Single DDRC Quad DFI Configuration (1:4/1:1:4 Frequency Ratio Mode)

SDRAM



DFI

HIF



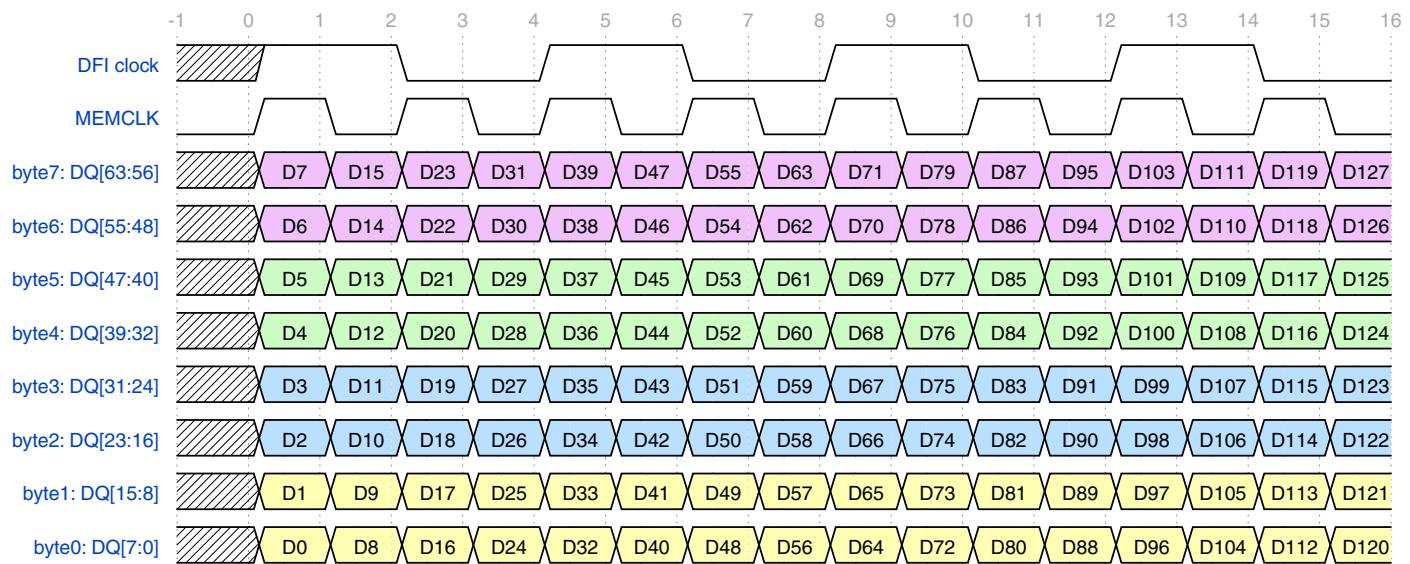
For HIF write data (`hif_wdata`), "n/a" shown in the diagrams must be set to '0'. That is, invalid data of `hif_wdata` must be set to '0'.

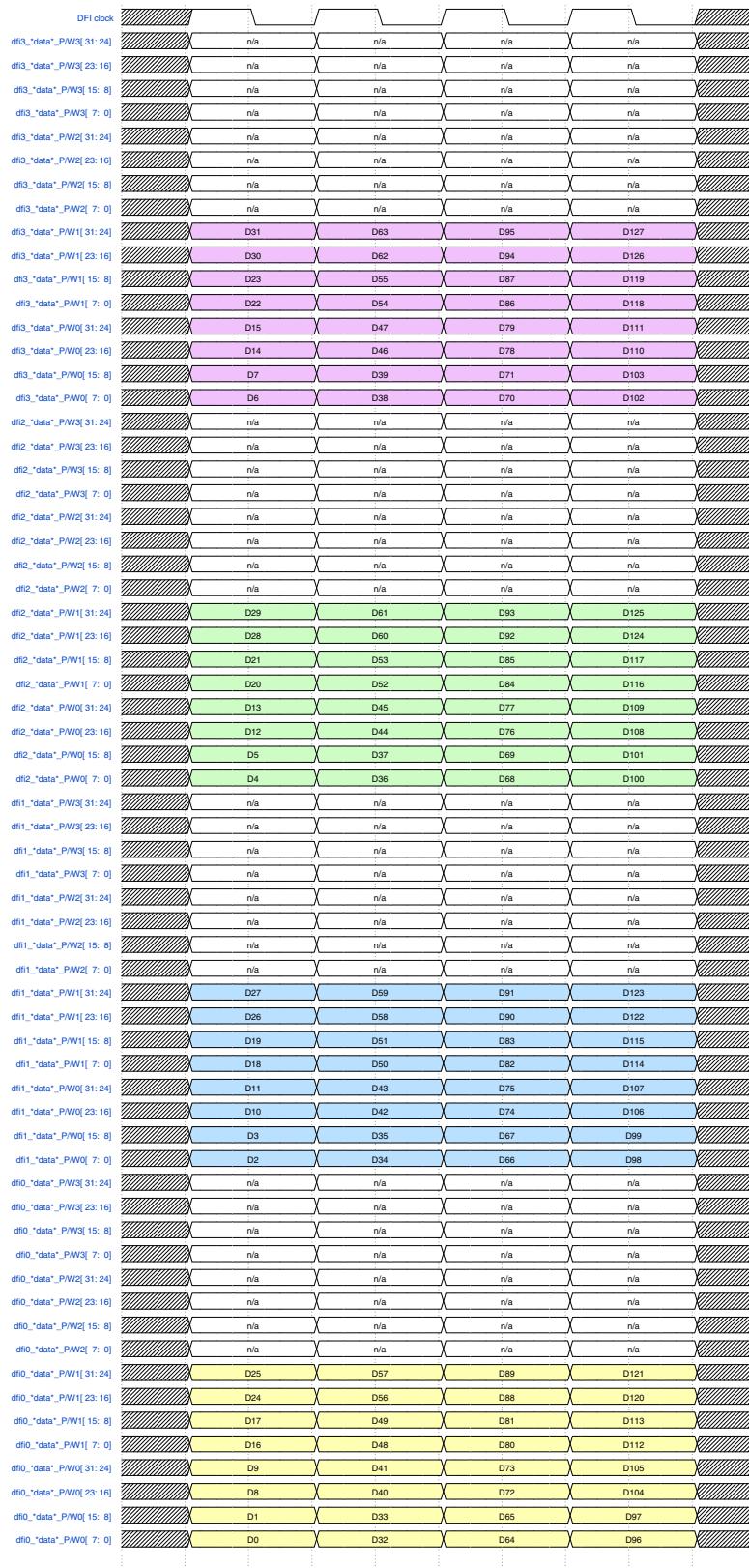
For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

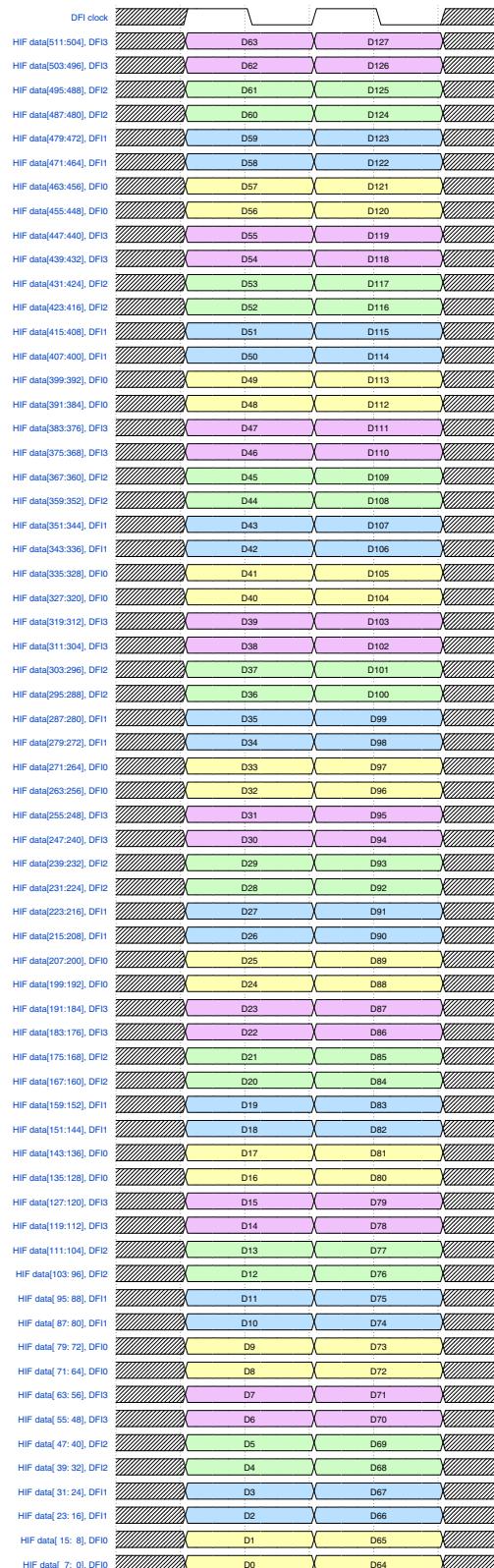
[Figure 5-13](#) shows the DFI Clocks of Single DDRC Quad DFI Configuration for 1:2/1:1:2 frequency ratio mode.

Figure 5-13 DFI Clocks of Single DDRC Quad DFI Configuration (1:2/1:1:2 Frequency Ratio Mode)

SDRAM



DFI

HIF



Note For HIF write data (`hif_wdata`), "n/a" shown in the diagrams must be set to '0'. That is, invalid data of `hif_wdata` must be set to '0'.

For HIF write/read data, diagrams show continuous beat, but invalid beat can be inserted between beats (non-back-to-back).

6

Address Mapping

This chapter contains the following sections:

- “[Overview of Address Mapping](#)” on page [154](#)
- “[Application to HIF Address Mapping](#)” on page [157](#)
- “[HIF Address to SDRAM Address Mapping](#)” on page [158](#)
- “[Recommendations for Optimum SDRAM Utilization](#)” on page [160](#)
- “[Non-binary Device Densities](#)” on page [162](#)
- “[Bank Hashing](#)” on page [168](#)
- “[Registers Related to Address Mapper](#)” on page [172](#)

6.1 Overview of Address Mapping

Read and write requests are provided to the DDRCTL with a system address. The system address is the command address of a transaction as presented on one of the data ports. The DDRCTL converts this system address to a physical address. It maps the system address to the SDRAM rank, bank group, bank, row, and column addresses.

If the system address regions are enabled (`UMCTL2_A_NSAR > 0`), the first part of the mapping is conversion of system address to AXI byte address. The controller maps disjointed address regions to internal consecutive addresses. Otherwise, the controller assumes that the DRAM is always mapped as a monolithic block. For more information about this, see “[System Address Regions](#)” on page [154](#).

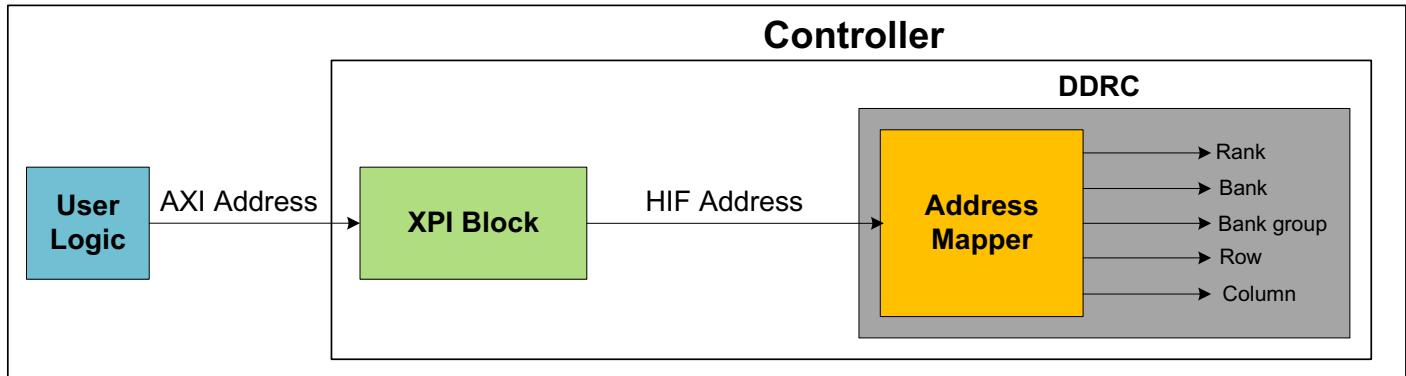
The second part of this mapping is conversion of AXI byte address to HIF word address. This is performed in the XPI block in AXI configuration. For more information about this, see “[Application to HIF Address Mapping](#)” on page [157](#).

The last part is the conversion of HIF word address to SDRAM address. A flexible address mapper maps the HIF word address to the SDRAM rank/bank/bank group/row/ column address. This address mapper is located within the DDRC.

The address mapping to be used depends on the use-case. The DDRCTL provides a set of registers that allows flexible re-programming of logical to physical address mapping. For more information about this, see “[HIF Address to SDRAM Address Mapping](#)” on page [158](#).

[Figure 6-1](#) explains the conversion of AXI to HIF to the SDRAM rank/bank/row/column/bank group addresses.

Figure 6-1 Conversion of AXI/CHI to HIF to SDRAM Rank/Bank/Row/Column/Bank Group Addresses



6.1.1 System Address Regions

The system address regions add the capability to define up to four disjoint memory regions mapping to the DRAM as consecutive addresses. The number of regions and minimum block size are determined by the hardware parameters `UMCTL2_A_NSAR` and `UMCTL2_SARMINSIZE`.

Each region is defined by:

- Base Address: Starting address of the region aligned to the minimum block size.
- Number of Blocks: Size of the region in multiples of minimum block size.

The base address of each region is specified by the register `SARBASEs.base_addr` and the total number of blocks is specified by the register `SARSIZES.nblocks` (see “[REGB_ARB_PORTp](#)” in “[Register Descriptions](#)” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

System address region specification is the same for all ports. Error response is not generated by the controller for addresses falling outside the specified address regions and the same address translation is applied from one base address to the next base address.

The base addresses must be specified such that they are in the ascending order ($SARBASE0 < SARBASE1 < SARBASE2 < SARBASE3$) and such that regions do not overlap. It is assumed that an outside agent can generate address decode errors, if they happen to occur.

6.1.1.1 System Address Regions Example

Consider the following example: the controller is set to have three system address regions (as specified on [Table 6-1](#)), with a minimum block size of 2 GB. Therefore, the `UMCTL2_SARMINSIZE` parameter must be set to 4.

The system address bits [`UMCTL2_A_ADDRW-1 : 31`] is used to represent the base addresses in the multiples of the minimum block size.

The registers must be programmed as follows:

- `SARBASE0.base_addr=1` (see “`REGB_ARB_PORTp`” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual)
- `SARBASE1.base_addr=17`
- `SARBASE2.base_addr=272`
- `SARSIZE0.nblocks=0`
- `SARSIZE1.nblocks=14`
- `SARSIZE2.nblocks=15`

The specifications in [Table 6-1](#) translates to the register settings mentioned in [Table 6-2](#).

Table 6-1 Address Region Mapping Example - Specification

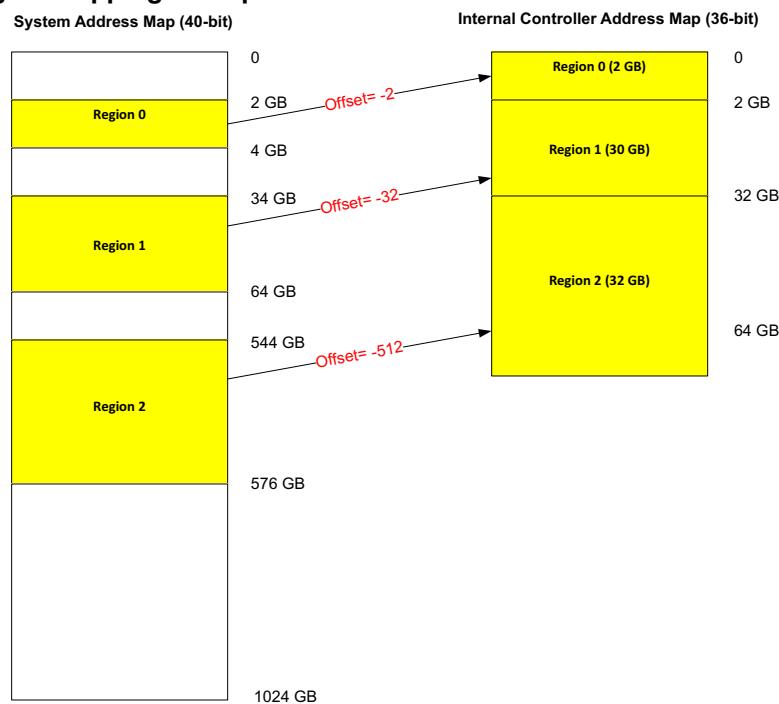
Region	System Address (40-bit)	Offset	Internal Controller Address (36-bit)
Region 0 (2 GB)	0x00 8000 0000–0x00 FFFF FFFF	0x00 8000 0000	0x00 0000 0000–0x00 7FFF FFFF
Region 1 (30 GB)	0x08 8000 0000–0x0F FFFF FFFF	0x08 0000 0000	0x00 8000 0000–0x07 FFFF FFFF
Region 2 (32 GB)	0x88 0000 0000–0x8F FFFF FFFF	0x80 0000 0000	0x08 0000 0000–0x0F FFFF FFFF

Table 6-2 Example Base Addresses, Number of Blocks and Offset

Region	Base Address (Binary)	Register Value (base_addr)	Register Value (nblocks)	Calculated Base Address	Calculated Offset (Decimal/GB)
Region 0	9'b0_0000_0001	1	0	2 GB	1 / 2 GB
Region 1	9'b0_0001_0001	17	14	34 GB	16 / 32 GB
Region 2	9'b1_0001_0000	272	15	544 GB	256 / 512 GB

Figure 6-2 shows the graphical illustration of the system address regions as specified in Table 6-1.

Figure 6-2 Address Region Mapping Example



Note When UMCTL2_AXI_BOUNDARY > UMCTL2_SARMINSIZE + 27, portion of the address below the boundary can be changed by the conversion from system to AXI address. In this case, you must ensure that the converted AXI address does not cross the boundary.

The maximum size of a system address region that can be supported by the feature is
 $(\text{SARSIZE}_n.\text{nblocks}+1) * \text{UMCTL2_SARMINSIZE}$

For example, if SARSIZE0.nblocks=7 and UMCTL2_SARMINSIZE=1, then the maximum size of System Address Region 0 is $8 * 256\text{MB} = 2\text{GB}$

If a bigger SAR region size is needed, then use a bigger UMCTL2_SARMINSIZE.



Note Invalid accesses in gaps are not verified in CHI configurations.

6.2 Application to HIF Address Mapping

The {ARADDR | AWADDR} is a byte address. Based on the MEMC_BURST_LENGTH, MEMC_FREQ_RATIO, MEMC_DRAM_DATA_WIDTH, and MSTR0.data_bus_width, XPI maps the MSB bits of the application address to the HIF address (hif_cmd_addr) in the following ways:

- $\text{hif_cmd_addr}[40:\log_2(\text{MEMC_BURST_LENGTH})] = \{\text{ARADDR} | \text{AWADDR}\}$
[UMCTL2_A_ADDRW-1: $\log_2(\text{MEMC_BURST_LENGTH}) + \log_2(\text{MEMC_DRAM_DATA_WIDTH}/(8*2^{\text{MSTR0.data_bus_width}}))$]
- $\text{hif_cmd_addr}[\log_2(\text{MEMC_BURST_LENGTH})-1:(\text{MEMC_FREQ_RATIO}-1)]$ is internally generated by XPI
- $\text{hif_cmd_addr}[\text{MEMC_FREQ_RATIO}-2:0] = 0$

6.3 HIF Address to SDRAM Address Mapping

The address mapper maps HIF word addresses to SDRAM addresses by selecting the HIF address bit that maps to each and every applicable SDRAM address bit. While it is possible to map HIF address bits to a SDRAM address in any desired manner, the full available address space is accessible only when no two SDRAM address bits are determined by the same HIF address bit. Each SDRAM address bit has an associated register vector to determine its source.

Registers ADDRMAP x ($x=0$ to 12) are used to program the address mapper. For more information on ADDRMAP registers, see REGB_ADDR_MAP0 in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

The HIF address bit number is determined by adding the internal base of the ADDRMAP x ($x=0$ to 12) register to the programmed value for that register, as described in the following equation:

$$\text{HIF address bit number} = [\text{internal base}] + [\text{register value}]$$

For example, for ADDRMAP5.addrmap_col_b7, the internal base is 7. When full data bus is in use, column bit 7 is determined by the following equation:

$$7 + [\text{register value}]$$

If this register is programmed to 2, the HIF address bit can be calculated by using following equation:

$$[\text{HIF address bit number}] = 7 + 2 = 9$$

In other words, the column address bit 7 sent to SDRAM would always be equal to hif_cmd_addr[9] of the corresponding HIF source address.

The system address to physical address mapping can be done by choosing any one of the possible combinations from [Figure 6-3](#) on page [159](#). It explains the different possible ways to map the HIF address bits to the SDRAM rank/bank/bank group/row/column address.

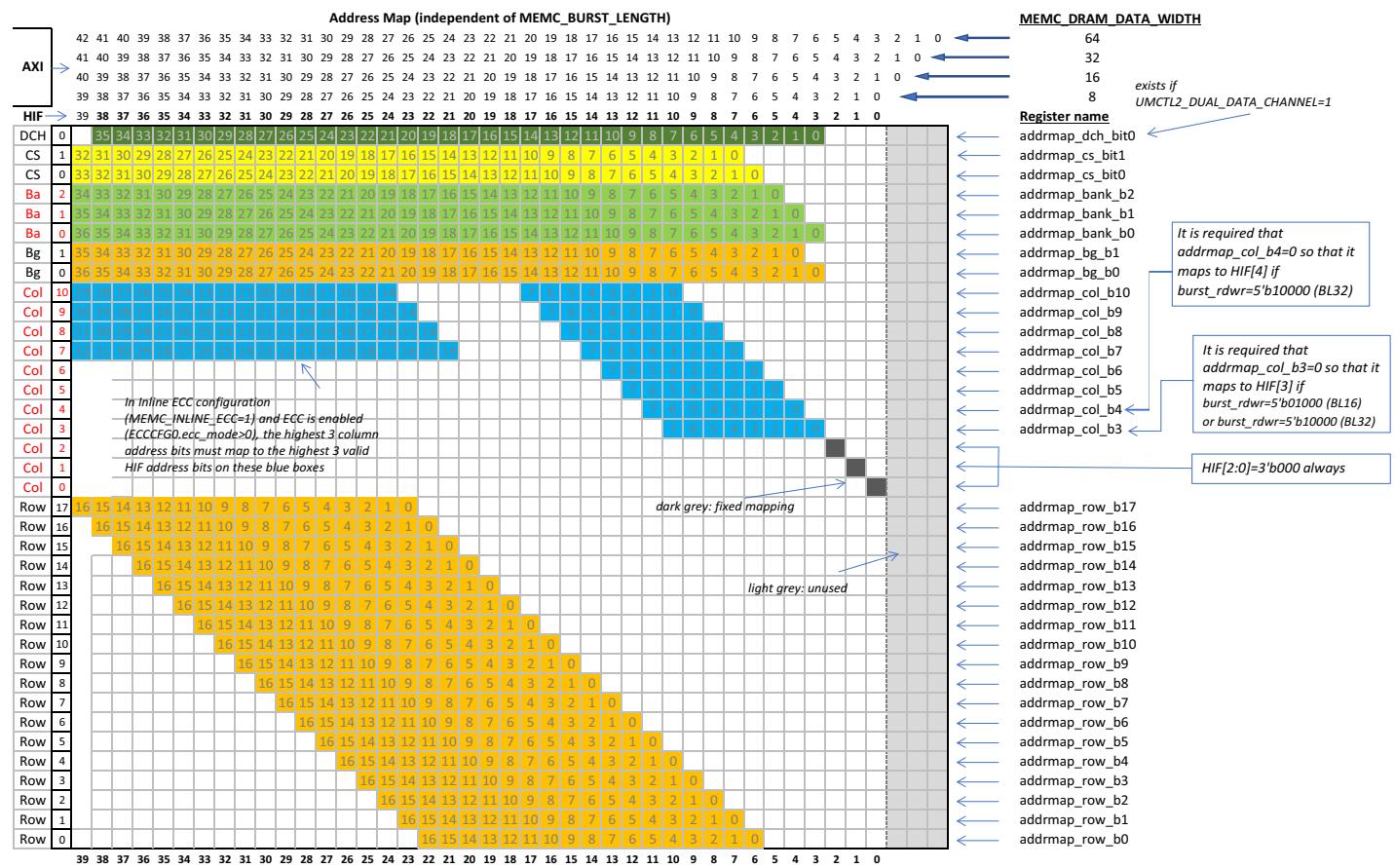


Note

- COL[3:0]==HIF[3:0] must be true for both LPDDR4 and LPDDR5 protocols.
- burst_rdwr==5'b01000 must be true for both LPDDR4 and LPDDR5 protocols.
- Register constraints are independent of ARB or HIF configuration and bus width mode.
- SDRAM column address [2:0] does not exist and hence is not addressable. If UMCTL2_INCL_ARB==1, this is automatically the case. The HIF master must generate hif_cmd_addr[2:0]=000.
- For any address bits which cannot be in use in all cases, all bits of the associated address map register field must be set to ‘1’ when the associated SDRAM address bit is not in use.



Ensure that no two SDRAM address bits are determined by the same HIF address bit.

Figure 6-3 System Address to Physical Address Mapping (Independent of MEMC_BURST_LENGTH)

MEMC_DRAM_DATA_WIDTH must be set to '16' or '32' in LPDDR5/4/4X Controller.

6.4 Recommendations for Optimum SDRAM Utilization

Write or Masked Write to Masked Write have a t_{CCDMW} penalty instead of the minimum t_{CCD} . There is no performance hit for reads as t_{CCDMW} is not related to them. These performance hits can be minimized/avoided by using address mapping recommendations in arbiter configurations and traffic recommendations in HIF configurations, as outlined in [Table 6-3](#) and [Table 6-4](#).

It is recommended to perform bank or bank group rotation between back to back SDRAM commands. For LPDDR4, t_{CCDMW} is equal to $4 * t_{CCD}$, so two bank bits need to be rotated. For LPDDR5, t_{CCDMW} in the case of BG mode is equal to $8 * t_{CCD}$, so two bank group bits and one bank bit need to be rotated.

Description of terms used in the columns of the tables are as follows:

- In the “HIF behavior” and “HIF burst size” columns:
 - “Full writes, reads, RMWs corresponds to writes/RMWs with HIF burst size equal to $MEMC_BURST_LENGTH / (MEMC_FREQ_RATIO * 2)$ in write data channel, and to reads with `hif_cmd_length=2'b00`.
 - “Half writes, reads, RMWs” corresponds to writes/RMWs with HIF burst size equal to $MEMC_BURST_LENGTH / (MEMC_FREQ_RATIO * 4)$ in write data channel, and to reads with `hif_cmd_length=2'b10`.
- In the “Performance” column:
 - “Data of each CAM entry used” refers to the data size stored in each CAM entry.
 - Full means that each entry stores data corresponding to $MEMC_BURST_LENGTH$ size.
 - Half means that each entry stores data corresponding to $MEMC_BURST_LENGTH / 2$ size.

For setting the Address Map register, `ADDRMAP6.addrmap_col_b3` must be set to ‘0’.

Table 6-3 Address Map Recommendations for Arbiter Configurations

Address Map	HIF Behavior	Performance
LPDDR5		
<code>addrmap_col_b3=0 (HIF[3])</code> <code>addrmap_col_b4=0 (HIF[4])</code> <code>addrmap_bg_b0=2 (HIF[5])</code>	Full writes, reads, RMWs	SDRAM utilization: full Data of each CAM entry used: full
LPDDR4 (MEMC_BURST_LENGTH=16)		
<code>addrmap_col_b3=0 (HIF[3])</code> <code>addrmap_bank_b0=1 (HIF[4])</code> <code>addrmap_bank_b1=1 (HIF[5])</code>	Full writes, reads, RMWs	SDRAM utilization: full Data of each CAM entry used: full
LPDDR4 (MEMC_BURST_LENGTH=32)		
<code>addr_map_col_b3=0 (HIF[3])</code> <code>addr_map_col_b4=0 (HIF[4])</code> <code>addr_map_bank_b0=2 (HIF[5])</code>	Full writes, reads, RMWs	SDRAM utilization: full Data of each CAM entry used: full

Table 6-4 Address Map Recommendations for HIF Configurations

Traffic	Maximum HIF Burst Size	Performance
LPDDR5		
Rotate 2 bank group bits and 1 bank bit between HIF commands	Full writes, reads, RMWs	SDRAM utilization: full Data of each CAM entry used: full
LPDDR4		
Rotate 2 bank bits between HIF commands	Full writes, reads, RMWs	SDRAM utilization: full Data of each CAM entry used: full



If Mask write is not predominant in the traffic, different address map might be fine to the SDRAM utilization.

6.5 Non-binary Device Densities

LPDDR4 3Gb/6Gb/12Gb per channel devices, LPDDR5 3Gb/6Gb/12Gb/24Gb per channel devices do not support addresses which have both MSB and MSB-1 row bits high. Any attempt to read or write to this address space results in memory misbehavior or system halts. When such devices are used, you must set the register ADDRMAP12.nonbinary_device_density accordingly. This prevents the DDRCTL from accessing this address space when there is a read or write request from you/SoC.

Table 6-5 Non-binary Device Densities

Non-binary Device Densities	Component Type	Inaccessible Region	
		LPDDR4	LPDDR5
3Gb	X16	[R14:R13]!=2'b11	[R13:R12]!=2'b11
3Gb	X8	[R15:R14]!=2'b11	[R14:R13]!=2'b11
6Gb	X16	[R15:R14]!=2'b11	[R14:R13]!=2'b11
6Gb	X8	[R16:R15]!=2'b11	[R15:R14]!=2'b11
12Gb	X16	[R16:R15]!=2'b11	[R15:R14]!=2'b11
12Gb	X8	[R17:R16]!=2'b11	[R16:R15]!=2'b11
24Gb	X16	N/A	[R16:R15]!=2'b11
24Gb	X8	N/A	[R17:R16]!=2'b11
24Gb	X4	N/A	N/A

When ADDRMAP12.nonbinary_device_density is set to non-zero, any write or RMW request with row[MSB:MSB-1]==2'b11 is discarded, while the HIF output hif_wdata_ptr_addr_err is asserted. Also, any read request with row[MSB:MSB-1]==2'b11 is executed (spare read) by changing row[MSB:MSB-1] from 2'b11 to 2'b10. The return data for such reads are masked to zeros, while the HIF output hif_rdata_addr_err is asserted.

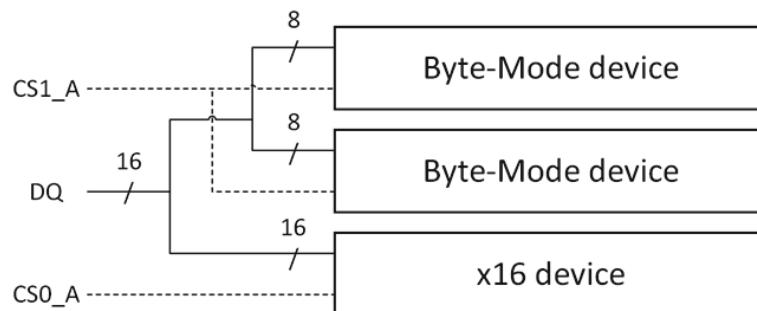
For more information, see “[hif_wdata_ptr_addr_err](#)” on page 88 and “[hif_rdata_addr_err](#)” on page 84.

6.6 Mixed Packages

Mixed packages are defined in the JEDEC specifications. In this configuration, some ranks contain x16 device and other ranks contain byte-mode device.

The JEDEC specification says that CS0 (rank0) is connected to x16 device and CS1 (rank1) is connected to byte-mode device. The controller supports only this connection.

Figure 6-4 Signal Assignment for Mixed Packages



In mixed packages, both ranks use byte-mode latency parameters.

For refresh related registers, both ranks must be set to different values if density is different.

You must set the following registers for rank1 which is byte-mode device:

- RFSHSET1TMG11.t_pbr2pbr_mp
- RFSHSET1TMG12.t_rfc_min_ab_mp
- RFSHSET1TMG12.t_rfc_min_mp
- RFMSET1TMG1.t_rfmpb_mp



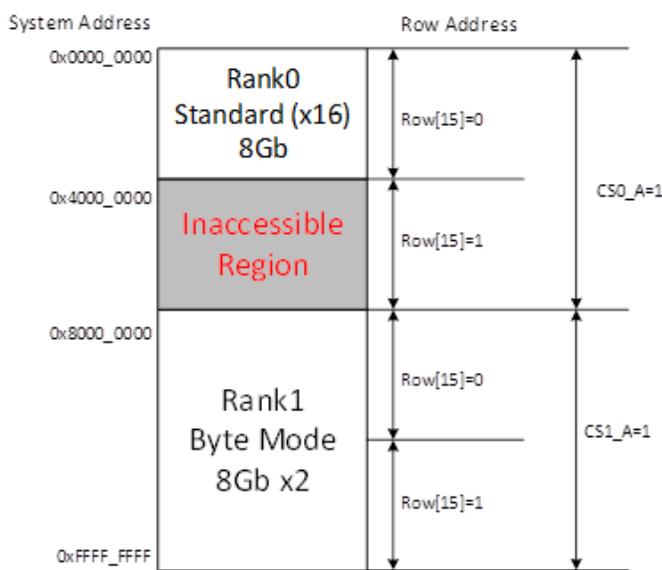
Note Four ranks do not support mixed packages.

6.6.1 Mixed Packages Address Mapping

x16 device and byte-mode device have different address ranges. Byte-mode device has wider address range than x16 device if both have the same density. This means that, x16 device has inaccessible region which is not allowed to access.

Any attempt to read or write to this address space results in memory misbehavior or system halts. When such devices are used, you must set the register ADDRMAP12.1pddr_mixed_pkg_x16_size accordingly and ADDRMAP12.1pddr_mixed_pkg_en to 1'b1. This prevents the DDRCTL from accessing this address space when there is a read or write request from SoC.

Figure 6-5 shows inaccessible region in terms of systems address and row address.

Figure 6-5 Inaccessible Region on Rank0:8Gb Rank1:8Gb for LPDDR5

[Table 6-6](#) and [Table 6-7](#) show inaccessible region for each density combination.

Table 6-6 Mixed Packages Device Densities for LPDDR4

Density of Rank0 x16 Device / Row Address Range	Density of Rank1 Byte-Mode Device / Row Address Range	Inaccessible Region in Rank0
1Gb/R0-R12	1Gb/R0-R13	[R13]==1'b1
	2Gb/R0-R14	[R14:R13]!=2'b00
2Gb/R0-R13	2Gb/R0-R14	[R14]==1'b1
	3Gb/R0-R15(R14=0 when R15=1)	[R15:R14]!=2'b00
	4Gb/R0-R15	[R15:R14]!=2'b00
3Gb/R0-R14(R13=0 when R14=1)	3Gb/R0-R15(R14=0 when R15=1)	[R15:R13]>3'b010
	4Gb/R0-R15	[R15:R13]>3'b010
	6Gb/R0-R16(R15=0 when R16=1)	[R16:R13]>4'b0010
4Gb/R0-R14	4Gb/R0-R15	[R15]==1'b1
	6Gb/R0-R16(R15=0 when R16=1)	[R16:R15]!=2'b00
	8Gb/R0-R16	[R16:R15]!=2'b00
6Gb/R0-R15(R14=0 when R15=1)	6Gb/R0-R16(R15=0 when R16=1)	[R16:R14]>3'b010
	8Gb/R0-R16	[R16:R14]>3'b010
	12Gb/R0-R17(R16=0 when R17=1)	[R17:R14]>4'b0010

Density of Rank0 x16 Device / Row Address Range	Density of Rank1 Byte-Mode Device / Row Address Range	Inaccessible Region in Rank0
8Gb/R0-R15	8Gb/R0-R16	[R16]==1'b1
	12Gb/R0-R17(R16=0 when R17=1)	[R17:R16]!=2'b00
	16Gb/R0-R17	[R17:R16]!=2'b00
12Gb/R0-R16(R15=0 when R16=1)	12Gb/R0-R17(R16=0 when R17=1)	[R17:R15]>3'b010
	16Gb/R0-R17	[R17:R15]>3'b010
16Gb/R0-R16	16Gb/R0-R17	[R16]==1'b1

Table 6-7 Mixed Packages Device Densities for LPDDR5

Density of Rank0 x16 Device / Row Address Range	Density of Rank1 Byte-Mode Device / Row Address Range	Inaccessible Region in Rank0
2Gb/R0-R12	2Gb/R0-R13	[R13]==1'b1
	3Gb/R0-R14(R13=0 when R14=1)	[R14:R13]!=2'b00
	4Gb/R0-R14	[R14:R13]!=2'b00
3Gb/R0-R13(R12=0 when R13=1)	3Gb/R0-R14(R13=0 when R14=1)	[R14:R12]>3'b010
	4Gb/R0-R14	[R14:R12]==3'b11
	6Gb/R0-R15(R14=0 when R15=1)	[R15:R12]>4'b0010
4Gb/R0-R13	4Gb/R0-R14	[R14]==1'b1
	6Gb/R0-R15(R14=0 when R15=1)	[R15:R14]!=2'b00
	8Gb/R0-R15	[R15:R14]!=2'b00
6Gb/R0-R14(R13=0 when R14=1)	6Gb/R0-R15(R14=0 when R15=1)	[R15:R13]>3'b010
	8Gb/R0-R15	[R15:R13]>3'b010
	12Gb/R0-R16(R15=0 when R16=1)	[R16:R13]>4'b0010
8Gb/R0-R14	8Gb/R0-R15	[R15]==1'b1
	12Gb/R0-R16(R15=0 when R16=1)	[R16:R15]!=2'b00
	16Gb/R0-R16	[R16:R15]!=2'b00
12Gb/R0-R15(R14=0 when R15=1)	12Gb/R0-R16(R15=0 when R16=1)	[R16:R14]>3'b010
	16Gb/R0-R16	[R16:R14]>3'b010
	24Gb/R0-R17(R16=0 when R17=1)	[R17:R14]>4'b0010

Density of Rank0 x16 Device / Row Address Range	Density of Rank1 Byte-Mode Device / Row Address Range	Inaccessible Region in Rank0
16Gb/R0-R15	16Gb/R0-R16	[R16]==1'b1
	24Gb/R0-R17(R16=0 when R17=1)	[R17:R16]!=2'b00
	32Gb/R0-R17	[R17:R16]!=2'b00
24Gb/R0-R16(R15=0 when R16=1)	24Gb/R0-R17(R16=0 when R17=1)	[R17:R15]>3'b010
	32Gb/R0-R17	[R17:R15]>3'b010
32Gb/R0-R16	32Gb/R0-R17	[R16]==1'b1

When ADDRMAP12.lpddr_mixed_pkg_x16_size is set to non-zero and ADDRMAP12.lpddr_mixed_pkg_en is set to 1'b1, any write or RMW request to inaccessible region is discarded, while the HIF output hif_wdata_ptr_addr_err is asserted. Also, any read request to inaccessible region is executed (spare read) by changing the following row addresses:

- If rank0 (x16 standard device) is a non-binary device (3Gb, 6Gb, 12Gb, or 24Gb):
 - Row[MSB¹:MSB¹-1] to 2'b10.
 - Row bits above MSB¹ to 0.
- If rank0 (x16 standard device) is a binary device (1Gb, 2Gb, 4Gb, 8Gb, 16Gb, or 32Gb)
 - Row bits above MSB¹ to 0.

The return data for such reads are masked to zeros, while the HIF output hif_rdata_addr_err is asserted. For more information, see “[hif_wdata_ptr_addr_err](#)” on page [88](#) and “[hif_rdata_addr_err](#)” on page [84](#).

6.6.2 Mixed Packages Limitation

The following list shows limitation of mixed packages:

- The following combinations of density is not supported:
 - LPDDR4
 - Rank0: 1Gb and Rank1: 3Gb, 4Gb, 6Gb, 8Gb, 12Gb, or 16Gb
 - Rank0: 2Gb and Rank1: 1Gb, 6Gb, 8Gb, 12Gb, or 16Gb
 - Rank0: 3Gb and Rank1: 1Gb, 2Gb, 8Gb, 12Gb, or 16Gb
 - Rank0: 4Gb and Rank1: 1Gb, 2Gb, 3Gb, 12Gb, or 16Gb
 - Rank0: 6Gb and Rank1: 1Gb, 2Gb, 3Gb, 4Gb, or 16Gb
 - Rank0: 8Gb and Rank1: 1Gb, 2Gb, 3Gb, 4Gb, or 6Gb
 - Rank0: 12Gb and Rank1: 1Gb, 2Gb, 3Gb, 4Gb, 6Gb, or 8Gb
 - Rank0: 16Gb and Rank1: 1Gb, 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, or 12Gb
 - LPDDR5

1. MSB here indicates the most significant bit of rank0 (For example, MSB is 12 when density of rank0 is 2Gb for LPDDR5)

- Rank0: 2Gb and Rank1: 6Gb, 8Gb, 12Gb, 16Gb, 24Gb, or 32Gb
- Rank0: 3Gb and Rank1: 2Gb, 8Gb, 12Gb, 16Gb, 24Gb, or 32Gb
- Rank0: 4Gb and Rank1: 2Gb, 3Gb, 12Gb, 16Gb, 24Gb, or 32Gb
- Rank0: 6Gb and Rank1: 2Gb, 3Gb, 4Gb, 16Gb, 24Gb, or 32Gb
- Rank0: 8Gb and Rank1: 2Gb, 3Gb, 4Gb, 6Gb, 24Gb, or 32Gb
- Rank0: 12Gb and Rank1: 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, or 32Gb
- Rank0: 16Gb and Rank1: 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, or 12Gb
- Rank0: 24Gb and Rank1: 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, 12Gb, or 16Gb
- Rank0: 32Gb and Rank1: 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, 12Gb, 16Gb, or 24Gb
- System other than two ranks is not supported.
- Non-JEDEC compliant connection (rank0: byte-mode device, rank1: x16 device) is not supported.
- The ECC scrubber is not supported.
- Inline ECC is not supported.

6.7 Bank Hashing

DDRCTL currently maps the HIF address to SDRAM address - rank, bank, bank group, Column, and row bits using the address mapping registers. The bank hashing function is situated after the HIF address to DRAM address field mapper. When bank hashing feature is enabled, the DDRCTL performs an XOR function on the physical address bits - row and bank. The output of this XOR function is the final bank bits. The functionality is explained in detail in further sections.

6.7.1 Bank Hashing Function

The bank hashing function in DDRCTL performs fixed XOR functions on the SDRAM row and bank-BG bits. Based on the SDRAM, the number of bank and BG bits may vary, and the XOR function changes accordingly. Each bank is uniquely identified by concatenating the BG and bank bits. For each bank bit, a set of row bits are selected. An XOR operation is performed between the bank bit and the selected row bits. The output of this XOR is the final bank bit.

Depending on SDRAM devices, three, four, or five bank bits can be valid. The maximum number of row bits is 18. All row bits will be used in the bank hashing function. Based on the total number of valid bank bits, the structure of the bank hash function changes. The Hashing function currently supports three cases:

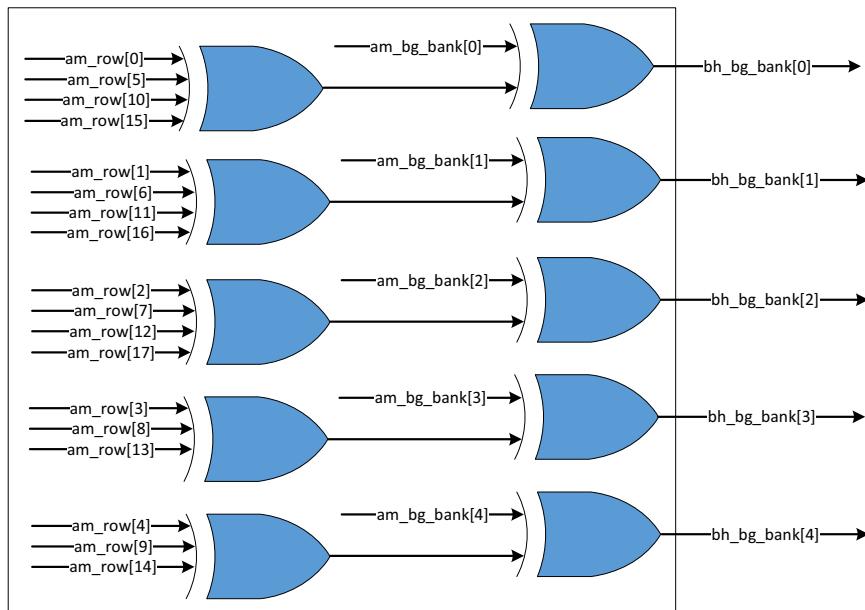
1. The number of bank bits is equal to 5. In [Figure 6-6](#), the signals am_bg_bank and am_row indicate the mapped BG-bank bits and row bits from the address mapper respectively. These signals are propagated to bank hashing function. bh_bg_bank indicates the output from the bank hashing function. The function performs the following XOR operation on the am_bg_bank and am_row signals:

```

bh_bg_bank [0] = (am_bg_bank [0] ^ am_row[0] ^ am_row[5] ^ am_row[10] ^ am_row[15]);
bh_bg_bank [1] = (am_bg_bank [1] ^ am_row[1] ^ am_row[6] ^ am_row[11] ^ am_row[16]);
bh_bg_bank [2] = (am_bg_bank [2] ^ am_row[2] ^ am_row[7] ^ am_row[12] ^ am_row[17]);
bh_bg_bank [3] = (am_bg_bank [3] ^ am_row[3] ^ am_row[8] ^ am_row[13]);
bh_bg_bank [4] = (am_bg_bank [4] ^ am_row[4] ^ am_row[9] ^ am_row[14]);

```

Figure 6-6 Bank Hashing Function With 5 Bank Bits



2. The number of bank bits is equal to 4. In [Figure 6-7](#), the signals am_bg_bank and am_row indicate the mapped BG-bank bits and row bits from the address mapper respectively. These signals are propagated

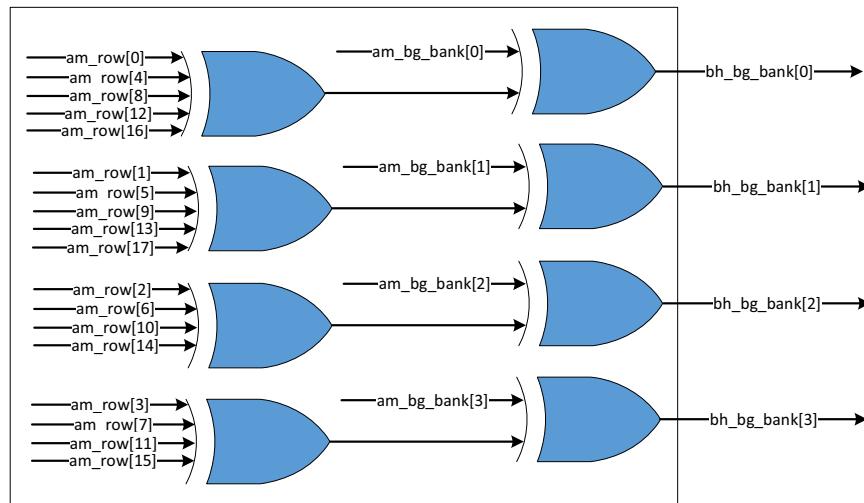
to bank hashing function. `bh_bg_bank` indicates the output from the bank hashing function. The function performs the following XOR operation on the `am_bg_bank` and `am_row` signals:

```

bh_bg_bank[0] = (am_bg_bank[0] ^ am_row[0] ^ am_row[4] ^ am_row[8] ^ am_row[12] ^ am_row[16]);
bh_bg_bank[1] = (am_bg_bank[1] ^ am_row[1] ^ am_row[5] ^ am_row[9] ^ am_row[13] ^ am_row[17]);
bh_bg_bank[2] = (am_bg_bank[2] ^ am_row[2] ^ am_row[6] ^ am_row[10] ^ am_row[14]);
bh_bg_bank[3] = (am_bg_bank[3] ^ am_row[3] ^ am_row[7] ^ am_row[11] ^ am_row[15]);

```

Figure 6-7 Bank Hashing Function With 4 Bank Bits



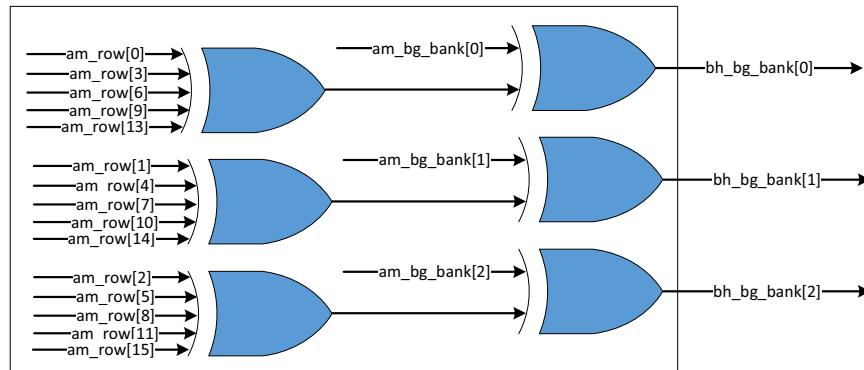
- The number of bank bits is equal to 3. In [Figure 6-8](#), the signals `am_bg_bank` and `am_row` indicate the mapped BG-bank bits and row bits from the address mapper respectively. These signals are propagated to bank hashing function. `bh_bg_bank` indicates the output from the bank hashing function. The function performs the following XOR operation on the `am_bg_bank` and `am_row` signals:

```

bh_bg_bank[0] = (am_bg_bank[0] ^ am_row[0] ^ am_row[3] ^ am_row[6] ^ am_row[9] ^ am_row[12] ^ am_row[15]);
bh_bg_bank[1] = (am_bg_bank[1] ^ am_row[1] ^ am_row[4] ^ am_row[7] ^ am_row[10] ^ am_row[13] ^ am_row[16]);
bh_bg_bank[2] = (am_bg_bank[2] ^ am_row[2] ^ am_row[5] ^ am_row[8] ^ am_row[11] ^ am_row[14] ^ am_row[17]);

```

Figure 6-8 Bank Hashing Function With 3 Bank Bits



The total number of bank and BG bits is determined by the parameter `MEMC_BG_BANK_BITS`. The total number of valid bits is derived based on the device configuration and the bank/BG address mapping registers. When bank hashing is enabled, the total number of bank bits can't be less than 3. For some devices, the higher row bits are not valid, in which case the corresponding input to bank hashing function will not toggle.

In LPDDR5 BG/16B mode, the independent bank and BG bits are concatenated as {BG [1:0]/BA [3:2], Bank [1:0]} and are then used by the bank hashing function. In LPDDR4 mode, the bank bits {Bank} are directly used by the bank hashing function. The concatenation is always bank at LSB and BG at MSB (if applicable). The valid combinations are listed in [Table 6-8](#).

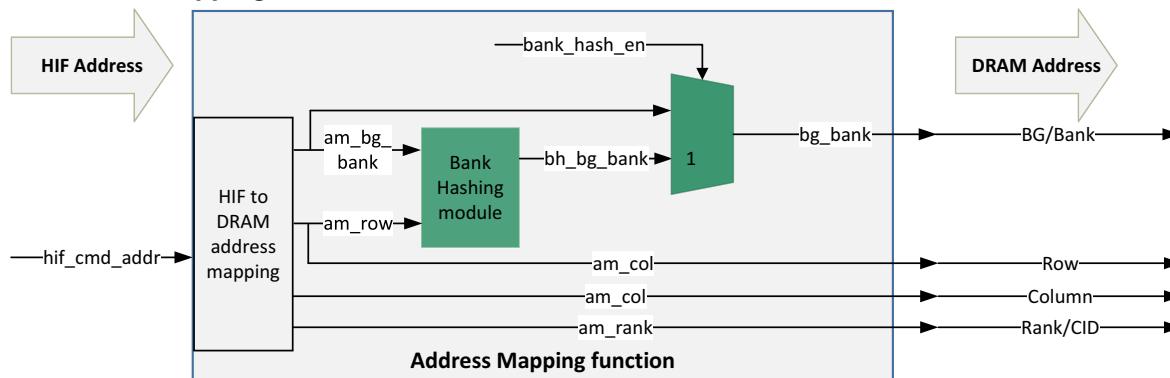
Table 6-8 Valid Combinations of Numbers of BG and Bank Bits

Number of BG Bits	Number of Bank Bits	Total Number of Bank Bits
0	4	4
2	2	4
2	1	3
1	2	3
0	3	3

Bank hashing function is performed by the Bank Hashing Module which is instantiated after HIF to DRAM address mapper. Software register `ADDRMAP12.bank_hash_en` provides bypass path of the module.

The module output bank/BG address (`bh_bg_bank`) is based on input bank/BG address (`am_bg_bank`) and row address (`am_row`).

Figure 6-9 Address Mapping Function



The address map registers determine if bank/BG bits are mapped (used). The number of mapped BG/bank bits determines the polynomial used for bank hashing.

The bank hashing feature is enabled by the parameter `DDRCTL_BANK_HASH`. The register bit `bank_hash_en` determines which bank values will be used by the downstream design.

**Note**

- When a non-binary density DRAM device has been selected, the invalid row address is converted into a valid one. However, the bank hashing function uses the incoming invalid row address directly.
- The bank hashing feature provides improved DRAM utilization with traffic patterns targeting same bank and row address toggling.
- The DDRCTL may report a DRAM location. With bank hashing the DDRCTL reports the hashed bank and BG bits.
 - In LPDDR54 controller the BG and bank values for CE, UE events are logged in APB register fields if the respective features are enabled.
 - The BG and bank values are provided as performance logging outputs.
 - In LPDDR4 configurations, per-bank refresh bank values are provided as outputs.
- The bank hashing function, enabled by `ADDRMAP12.bank_hash_en`, should be considered by any software functions which translate logged DRAM address to a HIF/AXI address.

6.8 Registers Related to Address Mapper

The following are the registers related to the Address Mapper:

- ADDRMAP0
- ADDRMAP2
- ADDRMAP1
- ADDRMAP3
- ADDRMAP4
- ADDRMAP5
- ADDRMAP6
- ADDRMAP7
- ADDRMAP8
- ADDRMAP9
- ADDRMAP10
- ADDRMAP11
- ADDRMAP12

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

7

Command Scheduling in DDRC

This chapter contains the following sections:

- “[Overview of Command Scheduling in DDRC](#)” on page [174](#)
- “[Page Policy](#)” on page [175](#)
- “[Transaction Stores](#)” on page [177](#)
- “[Address Collision Handling](#)” on page [193](#)
- “[Write Combine](#)” on page [194](#)
- “[Registers Related to Command Scheduling in DDRC](#)” on page [197](#)

7.1 Overview of Command Scheduling in DDRC

Command scheduling mechanism in DDRC allows you to carefully manage the following:

- Costly read/write bus turnaround
- Priorities of read requests to generally favor high priority traffic while also preventing starvation of low priority traffic

This functionality is implemented in a simple 2-state machine for each traffic type with completely configurable controls. The states of the 2-state machine determine when reads/writes are serviced and the relative priority (high priority versus low priority reads) at any given moment.

In this controller, the enhanced RD/WR switching features are always enabled to use.

The key features of the command scheduling mechanism are:

- During read mode, issue ACT commands pro-actively for a write request as page preparation in certain condition so that write command can be issued soon after switching to write mode (and vice versa).
- Prefer page-hit on the other direction rather than executing page-miss command on the current direction to reduce number of PRE-ACT cycles in certain conditions.
- Autonomous switching to write mode when Write CAM reaches a certain fill level to avoid Write CAM full as well as keeping read mode to optimize read latency if the WR CAM has enough space.
- In Inline ECC configuration, Write ECC CAM reaches a certain fill level that causes switching to write mode and avoids Write ECC CAM full.

7.2 Page Policy

This section contains the following subsections:

- “[Explicit Auto-Precharge \(Per Command\)](#)” on page 175
- “[Intelligent Precharges](#)” on page 175

7.2.1 Explicit Auto-Precharge (Per Command)

The explicit auto-precharge feature can enable auto-precharge on a per-command basis. If the HIF signal `hif_cmd_autopre` is set during a valid command, then the auto-precharge bit for that command is set when it is sent to the SDRAM.

If you do not want to use the per-command auto-precharge feature, then this bit can be tied to 1'b1 or 1'b0. Tying it to 1'b1 causes all the commands to be executed with auto-precharge, and tying it to 1'b0 causes no commands to be executed with auto-precharge.

In cases where a HIF transaction is translated into multiple DFI transactions, only the last DFI transaction is executed with auto-precharge. The earlier ones keep the page open, to allow subsequent transactions to benefit from the page hit.

7.2.2 Intelligent Precharges

The `SCHED0.pageclose` register enables precharge commands to be issued smartly through auto-precharges or explicit precharges. The exact functionality depends on the value programmed in `SCHEDTMG0.pageclose_timer`:

- If `SCHED0.pageclose` is set to '1' and `SCHEDTMG0.pageclose_timer=0`, a bank is kept open while there are page hit transactions available in the CAM to that bank. The last read or write command in the CAM with a bank and page hit is executed with auto-precharge. The intelligent precharge logic considers only the CAM or CAM region that is currently active, that is, while executing reads it looks at the LPR or HPR region of the read CAM (whichever is active at the time) and does not look at transactions in the other region of the read CAM or in the write CAM; while executing writes the logic does not look at transactions in the LPR or HPR regions of the read CAM. Whenever a mode switch occurs between LPR, HPR and writes, the page context is lost and therefore, a redundant auto-precharge may be issued followed by an explicit activate to the same bank or only an explicit precharge may be issued instead of an auto-precharge. The read and write commands that are executed as part of the ECC scrub requests are also executed with explicit precharge if the page needs to be closed.
- `SCHED0.pageclose` is set to '1' and `SCHEDTMG0.pageclose_timer >0`, a bank is kept open while there are page hit transactions available in the CAM to that bank. The last read or write command in the CAM with a bank and page hit is not executed with auto-precharge. Instead, a timer is started with `pageclose_timer` as the initial value. There is a timer on a per bank basis. The timer decrements unless the next read or write in the CAM to a bank is a page hit. It resets to `pageclose_timer` value if the next read or write in the CAM to a bank is a page hit. Once the timer reaches zero, an explicit precharge is scheduled. Prior to a bank's timer expiring, an explicit precharge may also be scheduled if the transaction scheduler chooses a CAM entry that is a row miss to the same bank.
- If `SCHED0.pageclose` is set to '0', the bank remains open only until there is a need to close it (to open a different page or for page timeout or refresh timeout). This is also known as open page policy. The open page policy can be overridden by setting the per command auto pre bit on the HIF interface (`hif_cmd_autopre`). When the multi-port arbiter is configured, the HIF signal `hif_cmd_autopre` is tied to '0' and you do not have control over it.

The intelligent precharge feature provides a midway between open and close page policies. SCEDTMG0.pageclose_timer gives you control over the time waited when there are no page hits to a bank in the CAM before auto-precharge or explicit precharge is scheduled. It is useful when the multi-port arbiter is configured. This timer allows the page to be kept open for a configurable number of clock cycles after there are no commands pending in the CAM to a bank. If there are pending commands higher up in the stream (for example, XPI/PA of multi-port arbiter) to the same bank/page, it gives a chance to be scheduled by the DDRCTL as an open page command.

Table 7-1 provides a summary of all paging policy options available in the DDRCTL.

Table 7-1 Paging Policy Options

Paging Policy	Availability	Description
Open page policy	All configurations	Setting SCED0.pageclose to '0' enables the open page policy. In HIF configurations, the open page policy can be overridden by sending commands with hif_cmd_autopre=1. In AXI configurations, the open page policy can be overridden in certain cases by sending commands with arautopre_n=1.
Intelligent precharge (Midway between Closed page and Open page policy)	All configurations	Setting SCED0.pageclose to '1' enables this smart paging policy as described in “ Intelligent Precharges ” on page 175. The exact functionality is dependent on SCEDTMG0.pageclose_timer value. In HIF configurations, this paging policy can be overridden by sending commands with hif_cmd_autopre=1. In AXI configurations, the open page policy can be overridden in certain cases by sending commands with arautopre_n=1.
Closed page policy	Only in HIF configurations. Not in multi-port configuration.	Control the paging policy per-command by asserting or de-asserting the auto precharge signal on the HIF interface – hif_cmd_autopre. If all commands are to be executed with auto-precharge, hif_cmd_autopre can be tied to '1'. In AXI configurations, the open page policy can be overridden in certain cases by sending commands with arautopre_n=1.

7.3 Transaction Stores

Transactions in the DDRCTL are separated into six categories:

- Low Priority Reads (LPR)
- Variable Priority Reads (VPR)
- High Priority Reads (HPR)
- Normal Priority Writes (NPW)
- Variable Priority Writes (VPW)
- High Priority Reads in LPR CAM (HPRL)



- Note**
- Hardware parameter UMCTL2_VPRW_EN must be defined to enable the VPR/VPW feature.
 - Hardware parameters DDRCTL_SYS_INTF=0 and UMCTL2_VPRW_EN=1 must be required to enable the HPRL feature.

7.3.1 Read Transaction Store

This section describes how LPR, VPR, HPR, and HPRL traffic classes work inside the DDRC. For information on how to map the AXI arqos value to LPR, VPR, and HPR traffic queues, refer to “[AXI QoS](#)” on page [69](#).

SCHED0.lpr_num_entries register splits the Read CAM in the controller into LPR and HPR sections. The LPR, VPR, and HPRL commands are sent to the LPR section of the CAM, and the HPR commands are sent to the HPR section. The VPR commands come in with an associated ‘timeout’ value. The LPR, VPR, and HPRL commands do not have ‘timeout’ value associated with them. The ‘timeout’ value of the VPR commands are counted down every cycle in which the commands are pending in the LPR store. If any VPR command has not been serviced and its ‘timeout’ value reaches 0, the command is considered as expired-VPR command. When there are any expired-VPR commands in the DDRC, those commands are given higher priority than HPR and LPR commands.

The Read CAM handles four traffic classes – LPR, VPR, HPR, and HPRL. But the Scheduling Engine that gets its input from the Read CAM, handles only two traffic classes - LPR and HPR. When there are no expired-VPR commands, all the VPR commands are treated as LPR. HPRL commands are treated as HPR.

7.3.2 Write Transaction Store

This section describes how NPW and VPW traffic classes work inside the DDRC. For information on how to map the AXI awqos value to NPW and VPW traffic queues, refer to “[AXI QoS](#)” on page [69](#).

The NPW and VPW commands are sent to the Write CAM. VPW commands do not have reserved space in the Write CAM. The VPW commands come in with an associated ‘timeout’ value. The NPW commands do not have ‘timeout’ value associated with them. The ‘timeout’ value of the VPW commands are counted down every cycle in which the commands are pending in the WR store. If any VPW command has not been serviced and its ‘timeout’ value reaches 0, the command is considered as expired-VPW command. When there are any expired-VPW commands in the DDRC, those commands are given higher priority than NPW commands.

The Write CAM handles two traffic classes - NPW and VPW. But the Scheduling Engine that gets its input from the Write CAM, handles only one traffic class - NPW. When there are no expired-VPW commands, all

the VPW commands are treated as NPW. When there are any expired-VPW commands in DDRC, all expired-VPW commands are given higher preference over NPW commands.

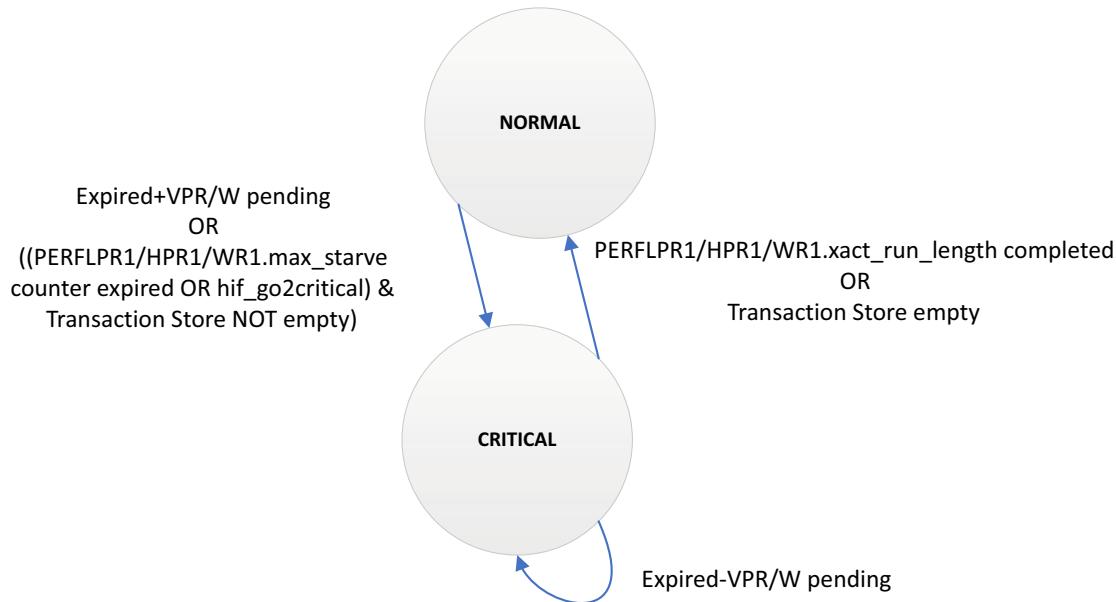
7.3.3 Transaction Store State Transitions

Each class of transaction store (LPR, HPR, or WR) can be in any one of the following two states:

- Normal: the state where the transaction store starts.
- Critical: indicates that the transaction store must be prioritized for service.

The transaction stores move between these two states under the control of *_max_starve and *_xact_run_length registers.

Figure 7-1 Transaction Store FSM (One FSM per store - WR, LPR, HPR)



Note: When there is more than one transition from a given state, the priority is shown in each state, with smaller number representing higher priority

Table 7-2 describes the transaction store state transitions.

Table 7-2 Transaction Store State Transitions

Current State	Next State	State Transition
Normal	Critical	This transaction store is not serviced for a count of *_max_starve clock cycles, or if an address collision happens.
Critical	Normal	*_xact_run_length number of transactions is serviced from this transaction store.

Taking the low priority read transaction store as an example, it is expected that the transaction store generally functions independently based on the following registers (see “REGB_FREQf_CHc Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual):

- PERFLPR1.lpr_max_starve

- `PERFLPR1.lpr_xact_run_length`

In the normal mode of operation, the FSM moves from normal to critical state by the starvation timer timeout. The store remains in Critical state until the number of serviced transactions becomes equal to `PERFLPR1.lpr_xact_run_length`, or when no more commands are available for that store, whichever happens first.

When the DDRCTL is configured as HIF-only (`UMCTL2_INCL_ARB==0`), it provides a feature where the SoC can force this state transition using external signals. It is useful in cases where the SoC may have additional information that is helpful in determining state transitions. For example, if the data stream has real-time requirements and the SoC has knowledge about FIFO depths or time-till-failure, it can use this information to explicitly request the DDRCTL to prioritize a particular data stream when it is critical to do so. The signals associated with this feature are `hif_go2critical_lpr` and `hif_go2critical_wr`.

In case of multi-port arbiter configuration (`UMCTL2_INCL_ARB==1`), the `hif_go2critical_*` signals are driven by the Port Arbiter. The generation of three signals involved in this case - `hif_go2critical_wr`, `hif_go2critical_lpr` and `hif_go2critical_hpr` is described in “[AXI QoS](#)” on page [69](#) and “[Urgent Signaling](#)” on page [74](#).

The assertion of `hif_go2critical_*` signals cause their respective queue FSMs to go to Critical state. The change in Read/Write mode switching and the Read priority level selection based on the presence of the `hif_go2critical_*` signals is mentioned in the following section.

**Note**

If `SCHED.opt_wrcam_fill_level==1`,
`SCHED3.wrcam_highthresh/SCHED5.wrecc_cam_highthresh` and
`SCHED3.wrcam_lowthresh/SCHED5.wrecc_cam_lowthresh` are used for the transition
in addition to `PERFWR1.w_max_starve` and `PERFWR1.w_xact_run_length`.

7.3.3.1 Store Critical

Depending factors, levels of critical are introduced on top of the original FSM.

`Critical_L1` is the highest priority and `Critical_L3` is the lowest priority within `Critical_L1/2/3`.

Table 7-3 Priorities for Configurations with `UMCTL2_INCL_ARB==1` (AXI Configuration)

Priority	External Signal	Source
<code>Critical_L1</code>	<code>awurgent_n</code> <code>arurgent_n</code> (<code>arurgentb_n</code> , <code>arurgentr_n</code> when <code>UMCTL2_XPI_USE2RAQ_n</code> is defined)	AXI sideband signals from SoC. Drive these make corresponding store <code>Critical_L1</code> . For more information, see “ Urgent Signaling ” on page 74 .
Between L1 and L2		If <code>opt_wrcam_fill_level==1</code> , write CAM fill-level makes write store Critical/Normal, the priority is higher than <code>Critical_L2</code> , and lower than <code>Critical_L1</code> .
<code>Critical_L2</code>	N/A	Asserted by XPI/PA for VPR/VPW timeout with corresponding CAM credit. For more information, see “ VPR/VPW Timeout ” on page 72 .

Priority	External Signal	Source
Critical_L3	N/A	Critical caused by <code>PERF.*_max_starve</code> . AXI port timeout only force read write switching inside PA, does not impact on transaction store FSM directly.

Table 7-4 Priorities for Configurations with UMCTL2_INCL_ARB==0 (HIF Configuration)

Priority	External Signal	Source
Critical_L1	<code>hif_go2critical_wr</code> <code>hif_go2critical_hpr</code> <code>hif_go2critical_lpr</code>	HIF sideband signals from SoC. Drive these make corresponding store Critical_L1.
Between L1 and L2		If <code>opt_wrcam_fill_level ==1</code> , write CAM fill-level makes write store Critical/Normal, the priority is higher than Critical_L2, lower than Critical_L1.
Critical_L2	N/A	Not used source for the configurations.
Critical_L3	N/A	Critical caused by <code>PERF.*_max_starve</code> .

7.3.4 Read Priority Management

In a read mode, read priority is controlled by `SCHED1.opt_hit_gt_hpr` and critical state.

If `SCHED1.opt_hit_gt_hpr==0`, or the register does not exist in your configuration, the preferred priority is:

1. Page-hit high priority read
2. Page-miss high priority read
3. Page-hit low priority read
4. Page-miss low priority read

If `SCHED1.opt_hit_gt_hpr==1`, the preferred priority is:

1. Page-hit high priority read
2. Page-hit low priority read
3. Page-miss high priority read
4. Page-miss low priority read

If the low priority read transaction store is in Critical state due to a transaction that is pending for a long time (see [Table 7-2](#) on page [178](#)) and the high priority read transaction store is not, the low priority read requests are preferred over high priority read requests. This prevents starvation of low priority reads.

The exceptions to the previous rule are as follows:

- When there are any expired-VPR commands in the DDRC, these are given priority ahead of any LPR commands, even if the LPR queue is in a critical state.
- If `hif_go2critical_hpr` is high, the HPR queue is given priority over LPR queue, even if LPR queue is in Critical state.

- If `hif_go2critical_lpr` is high and `hif_go2critical_hpr` is low, the LPR queue is given priority over HPR queue, even if HPR queue is in Critical state.

7.3.5 Read/Write Turnaround

The following terminology is used to describe the read/write turnaround algorithm:

- Write/Read pending: A write or a read command is pending in the CAM.
- Write/LPR/HPR critical: Indicates that the write, LPR or HPR queue is in critical state due to starvation.
- Read/Write idle timeout: A read/write idle timer based on the register `SCHEDTMG0.rdwrd_idle_gap` has expired.
- Page-miss: A request targeting to a closed bank or an opened bank with different page.
- Page-hit: A request targeting to an opened page on a bank.
- Page-hit timing based on ACT (page-hit timing 0): A page-miss request in Next Transaction Table (NTT), which stores the next candidate per bank/per direction, becomes page-hit once an ACT command is issued to the page. A read/write command cannot be issued until tRCD is satisfied. For more information, see [Figure 7-2](#) on page [183](#).
- Page-hit timing based on ACT + tRCD (page-hit timing 1): A page-miss request in Next Transaction Table becomes page-hit once an ACT command is issued to the page and tRCD is elapsed. Now, read/write command can be issued.
- Read Collision: An incoming command (write or RMW) on HIF has the same address as the existing read commands in the CAM. These existing reads must be scheduled-out timely (flush) to accept the incoming command.
- Write Collision: An incoming command (read/write/RMW) on HIF has the same address as an existing Write command in the CAM. This existing Write must be scheduled-out timely (flush) to accept the incoming command.
- Colliding Request: Existing entries in WR CAM or RD CAM, which has same address as incoming command.
- Collision Page-hit: Indicates the page-hit for colliding bank. In other words, during flush for address collision, the bank of the colliding request to be flushed is a page-hit. When a collision happens, the colliding request may not or may be loaded in NTT.
 - If colliding request is loaded in NTT, when the colliding bank is a page-hit, the colliding request is a page-hit request and is scheduled out.
 - If colliding request is not loaded in NTT, even when the colliding bank is a page-hit, it does not mean colliding command is a page-hit because the page is opened for non-colliding request. Once the request loaded in 'Next Transaction Table' and it is scheduled out, then the colliding request is loaded to 'Next Transaction Table' and another page can be opened if this is a page-miss against the previous one. In this case, collision page-hit may have two periods, one is previous request in 'Next Transaction Table' is a page-hit, and another is colliding request in 'Next Transaction Table' is a page-hit. Both the periods are collision page-hit.
- Critical: Critical is for each store (LPR, HPR or write). A read critical includes LPR store critical or HPR store critical. Write critical indicates write store critical.
 - Critical could be caused by starvation timer or sideband signals (`hif_go2critical_*` in HIF configuration or `arurgentr/arurgentb` and `awurgent` in AXI configuration)

Page-hit status is used as hint of global read/write switching. Depending on conditions, the DDRCTL selectively uses page-hit timing 0 or page-hit timing 1 as shown in [Table 7-5](#).

Table 7-5 Page-hit Timing

Transaction	Expired-VPR/W	Collision	Critical	Normal
Read	Page-hit timing 0	Page-hit timing 0	Page-hit timing 0	Page-hit timing 1
Write	Page-hit timing 0	Page-hit timing 0	Page-hit timing 1	Page-hit timing 1

Some examples:

- There is Expired-VPR, it is treated as a page-hit once ACT for the Expired-VPR is issued.
- Write store is critical, it is treated as page-hit once ACT for a write command is issued and tRCD is elapsed.

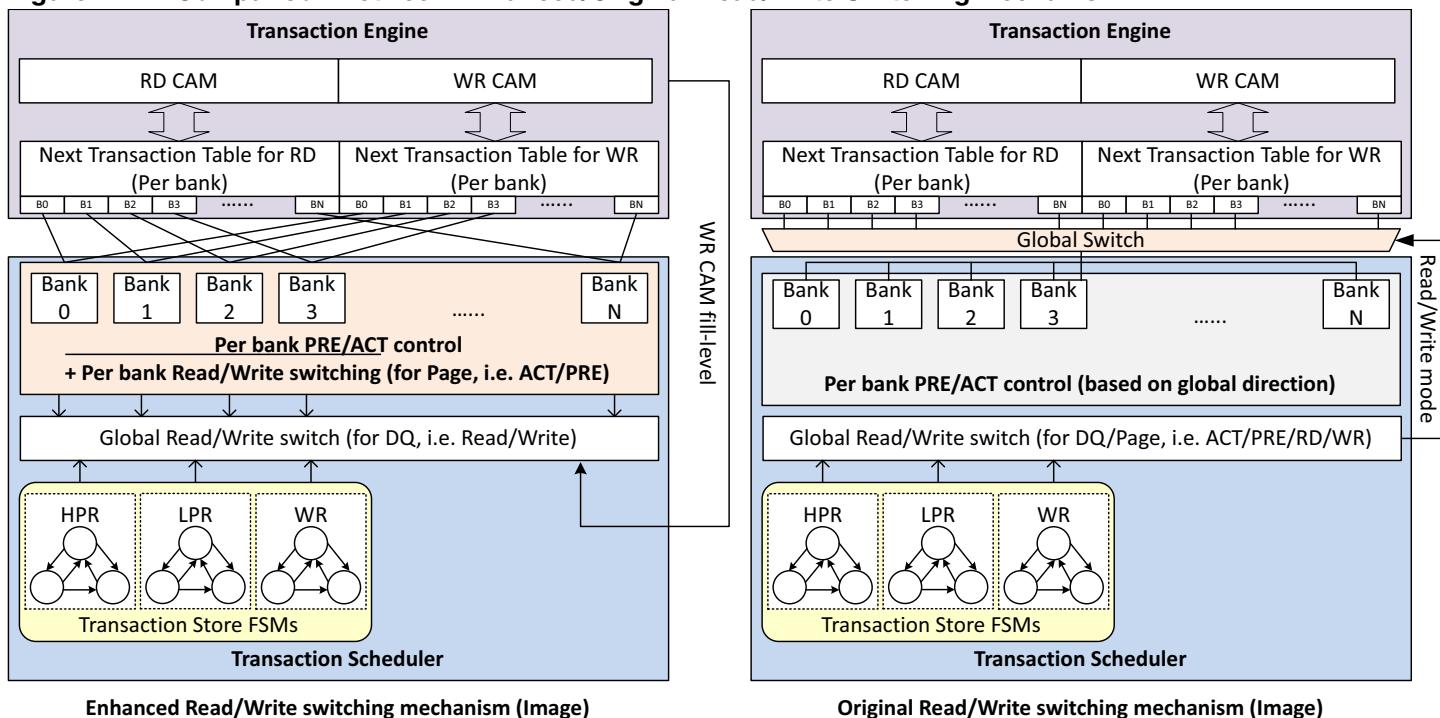
7.3.5.1 Overview of Enhanced Read Write Switching

As described in “[Overview of Command Scheduling in DDRC](#)” on page [174](#), the following new features are introduced:

- During read mode, issue an ACT command pro-actively for a write request as page preparation in certain conditions, so that the write command can be issued soon after switching to write mode (and vice versa).
- Prefer page-hit on the other direction rather than executing page-miss command on the current direction to reduce the number of PRE-ACT cycles in certain conditions.
- Autonomous switching to write mode when Write CAM reaches a certain fill level to avoid Write CAM full as well as keeping read mode to optimize read latency if WR CAM has enough space.

Figure 7-2 displays the concept of the enhancements.

Figure 7-2 Comparison Between Enhanced/Original Read/Write Switching Mechanism



For these enhancements, per-bank read/write switching (for page, that is, PRE/ACT) is introduced in addition to global read/write switching control (for DQ, that is, read/write).

The per-bank read/write switching mechanism decides which direction of the page to prepare (issue ACT/PRE) based on the criteria, which include the current direction from global read/write switching mechanism.

The global read/write switching mechanism decides the direction of read or write based on the criteria, which include page-hit status on both directions which is the result of the per-bank read/write switching.

Therefore, global read/write switching mechanism and per-bank read/write switching mechanism are interdependent.

The following sections explain the criteria of both the per-bank read/write switching and global read/write switching.

7.3.5.2 Per-bank Read/Write Switching for Page (ACT/PRE)

For a given bank, issue activate for the current direction request or the other direction request will be decided by the following information:

- Global read/write mode for DQ (that is, read/write commands)
- Critical status of Transaction Store FSMs, that is, HPR/LPR/WR store
- Collision status for each bank
- Page-hit status in read and write Next Transaction Table (NTT)

Figure 7-3 shows the decision for page management per bank.

Figure 7-3 Per-bank Page Control



- Expired-VPR/W is the highest priority.
 - If there are Expired-VPR requests, prepare a page of the bank for read direction regardless of the current global direction.
 - If there are Expired-VPW requests and no Expired-VPR requests, prepare a page of the bank for write direction regardless the current global direction.
- Collision is the next priority.
 - If only one direction has colliding request to be flushed, prepare page of the bank for collision direction.
 - If both directions have colliding request to be flushed, prefer to prepare a page of the bank for current direction if any page of the bank is not open for the other direction.
- Transaction Store FSM Critical is the next priority after collision. The behavior is similar to collision.
 - If both directions are critical, compare the critical level, prefer to prepare a page of the bank for higher level critical direction (L1>L2>L3).



Note Write CAM fill level or Write ECC CAM fill level exceeding high-threshold, causes write store Critical. The critical level is lower than Critical_L1 but higher than Critical_L2.

- If both directions are at the same critical level, prefer to prepare a page of the bank for current direction unless the bank is already open for the other direction.



Note Transaction Store FSM critical is per-store, and not per-bank. Therefore, this applies to all the banks.

- Except previous cases, prefer to prepare a page of the bank for the current direction if page of the bank is not open for the other direction.



Note While global read/write switching mechanism is in an intermediate state from read mode to write mode, in other words, waiting for SCHED1.delay_switch_write is expired, nothing is done for the bank. That is, do not prepare any page of the bank because it is uncertain which direction is finally chosen by global read/write switching.

- When the register rd_act_idle_gap is set to non-zero value, and in the read mode, do not prepare page of the bank for the write direction until the number of cycles determined by the rd_act_idle_gap is elapsed with no read request to the bank. This is intended to reduce the frequent read to write switching.
- Similarly, when the register wr_act_idle_gap is set to non-zero value, and in the write mode, do not prepare page of the bank for the read direction until the number of cycles determined by the wr_act_idle_gap is elapsed with no write request to the bank. This is intended to reduce the frequent write to read switching.
- The intention of the two separated activate idle gap registers, namely rd_act_idle_gap and wr_act_idle_gap, are to make hysteresis for page preparation for the other direction.

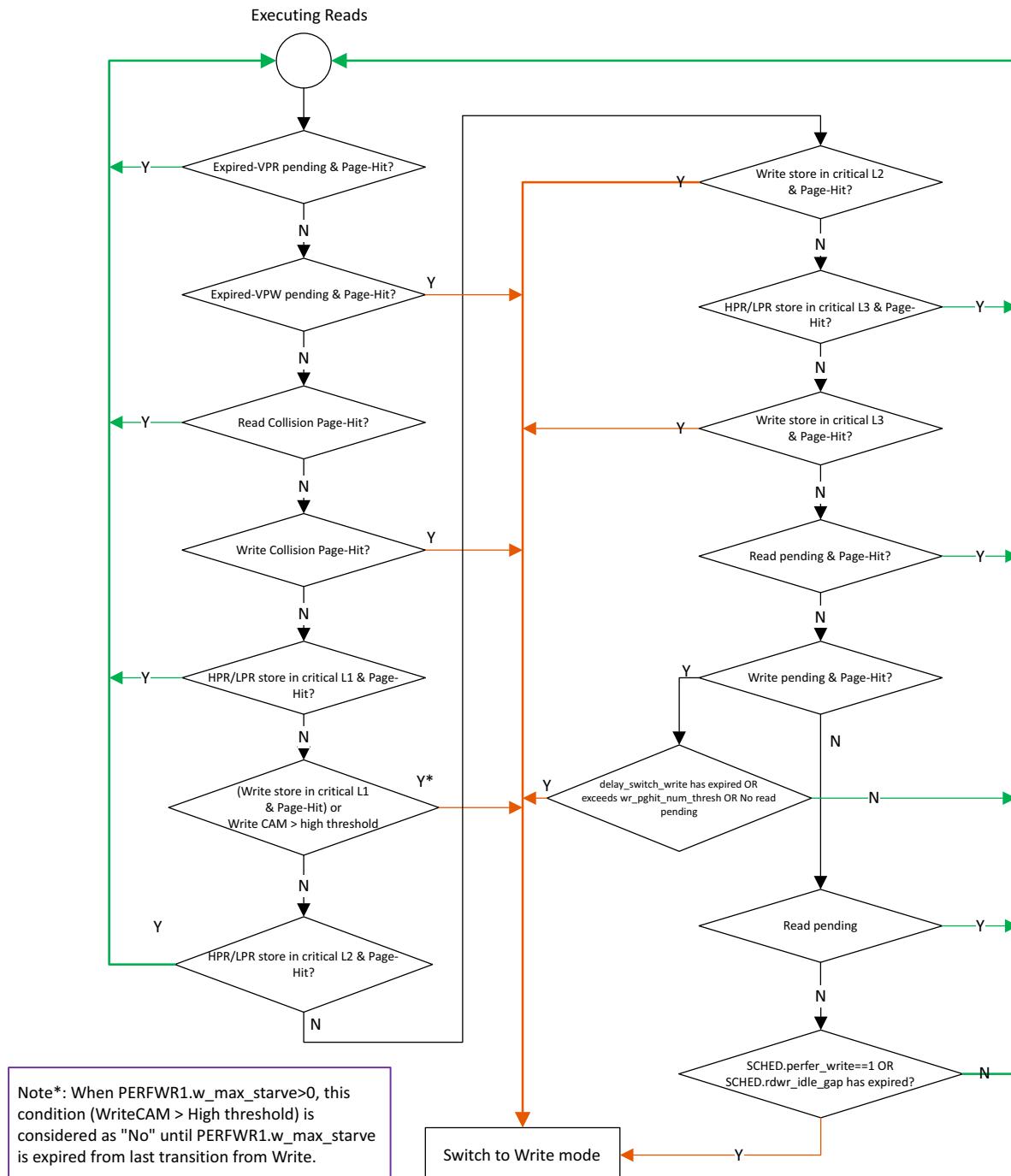
- `SCHED4.rd_page_exp_cycles` indicate number of cycles to keep the bank opened for read direction in the write mode when both directions have requests to the bank and to different pages. Similarly, `SCHED4.wr_page_exp_cycles` indicate number of cycles to keep the bank opened for write direction in read mode when both directions have requests to the bank and to different pages.
 - The purpose is to avoid a bank being opened for a long time in the opposite direction, while there is a page-miss requesting in the current direction, so that more banks can be used for the current direction to improve utilization. This is critical especially for bank-group rotation case, if only one bank group can be used for current direction, and the other bank-groups are opened for the opposite direction, the `tCCD_L` penalty cannot be avoided.
 - While global read/write switching mechanism is in an intermediate state from read mode to write mode, in other words, waiting for `SCHED1.delay_switch_write` to expire, pause timer for `SCHED4.wr_page_exp_cycles` for all the banks because it is uncertain which direction is finally chosen by global read/write switching.

7.3.5.3 Global Read/Write Switching for DQ (Read/Write Commands)

Read to Write Switching

Figure 7-4 shows the decision for global Read to Write switching.

Figure 7-4 Read to Write Switching



- Switch to write once a write request becomes a page-hit even if there are pending read requests but all are page-miss. This is intended to reduce the number of PRE-ACT cycles.
- Keep reading direction if both read and write requests are same priority, such as both read and write are collision and page-hit, both are in same critical level and page-hit, or both are page-miss.
- Expired-VPR/W is the highest priority, that is, if there are any Expired-VPR or Expired-VPW, only Expired-VPR and Expired-VPW could be candidate to cause read write switching, as the other requests are masked.



Note If both Expired-VPR and Expired-VPW are present and one of them is a page-hit, the page-hit one is higher priority than the other (If Expired-VPR and Expired VPW are on the same bank, Expired-VPR has priority for the page preparation, this is per-bank control).

- Collision page-hit is higher priority than store critical. The write collision page hit causes read to write switching if no read collision page hit exists even there is a read critical page-hit.
- Critical page-hit is higher priority than normal read/write page-hit. Write store critical page-hit causes read to write switching if LPR or HPR store are not critical.
 - Critical is separated into 3 levels, the priority is Critical_L1 > Critical_L2 > Critical_L3. Prefer to switch to the direction with higher level critical. For more information, see “[Store Critical](#)” on page [179](#).
 - If the register SCHED0.opt_wrcam_fill_level=1, write store starvation is managed by SCHED3.wrcam_highthresh/SCHED3.wrcam_lowthresh/SCHED5.wrecc_cam_high-thresh/SCHED5.wrecc_cam_lowthresh in addition to PERFWR1.w_max_starve and PERFWR1.w_xact_run_length. The switch from read to write is higher priority than Critical_L2, a lower priority than Critical_L1.
- A normal page-hit is of higher priority than page-miss. The write page-hit causes read to write switching if read request is not page-hit.
- Read page-miss is higher priority than SCHED0.prefer_write register. Even if SCHED0.prefer_write=1, keep read mode until all the read request are served regardless of a page-hit or page-miss as long as there is no write page-hit.
- To delay switching to write mode SCHED1.delay_switch_write is provided. It is possible that the incoming read request could be a page-hit or the pending read request will be a page-hit in the next several cycles even if there are no page-hit right now. With this function, frequent ‘read -> write -> read’ turnaround can be avoidable (keep read mode), and therefore read latency can be reduced.
- While waiting for SCHED1.delay_switch_write timeout, switch to write mode immediately once the number of write page-hit requests (total in all banks) exceeds the threshold set in SCHED1.wr_pghit_num_thresh. As this gives more confidence to switch to write as there are several page-hits command on write.

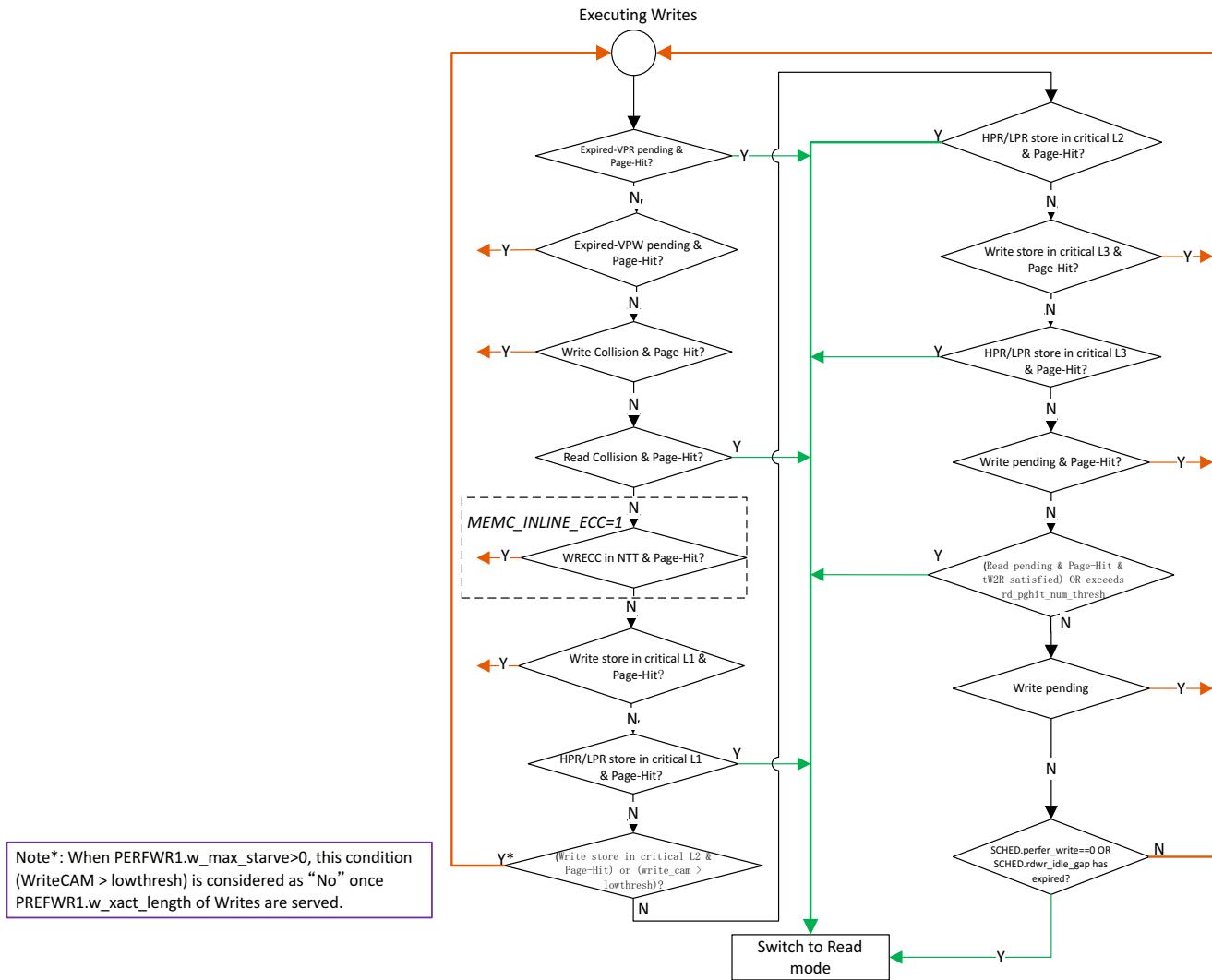


Note SCHED3.wr_pghit_num_thresh cannot be used when SCHED1.delay_switch_write=0.

7.3.5.4 Write to Read Switching

Figure 7-5 shows the decision for global Write to Read switching.

Figure 7-5 Write to Read Switching



Write to read switching is almost symmetrical with read to write switching. The differences are:

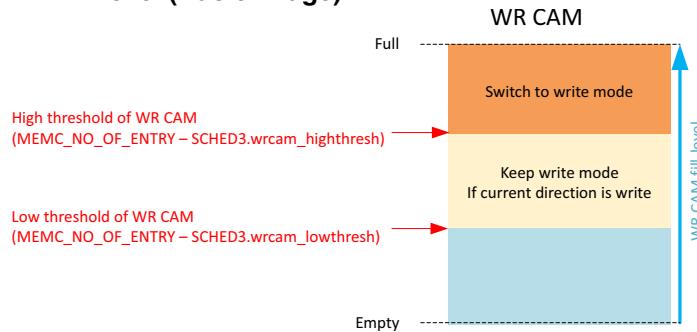
- Expired-VPR is higher priority than Expired-VPW even if it is a write mode.
- If there are page-hit in read direction and no page-hit in write direction, switch to read after tW2R time is satisfied. During waiting for write to read turnaround time specified in DRAMSET1TMG2.wr2rd (tW2R) is satisfied:
 - If a write becomes a page-hit, keep write mode.
 - If number of read page-hit exceeds threshold (SCHED3.rd_pghit_num_thresh), switch to read mode without waiting for tW2R is satisfied. As this gives more confidence to switch to read as there are several page-hits command on read.

When write CAM fill-level exceeds low threshold (SCHED3.wrcam_lowthresh), do not switch to read until write CAM fill level is under low threshold. For more information, see [Figure 7-6](#).

7.3.5.5 Optimize WR CAM Fill-Level

[Figure 7-6](#) shows the basic image of optimize WR CAM fill-level.

Figure 7-6 Optimize WR CAM Fill-Level (Basic Image)



When write CAM is almost full, switch to write mode to prevent write CAM full.

The two thresholds register provided for this feature are:

1. SCHED3.wrcam_highthresh
2. SCHED3.wrcam_lowthresh

In Inline ECC configurations (MEMC_INLINE_ECC==1), in addition to write CAM fill-level, write ECC CAM fill-level is also optimized by observing the number of valid write ECC entries and number of loaded write ECC entries:

- Valid write ECC entry indicates that the write ECC entry can be served as write CAM has no write entries belonging to the same Block Token (BT).
- Loaded write ECC entry indicates that the write ECC entry is loaded in the write ECC CAM but cannot be served as write CAM has the write entries belonging to the same Block Token.

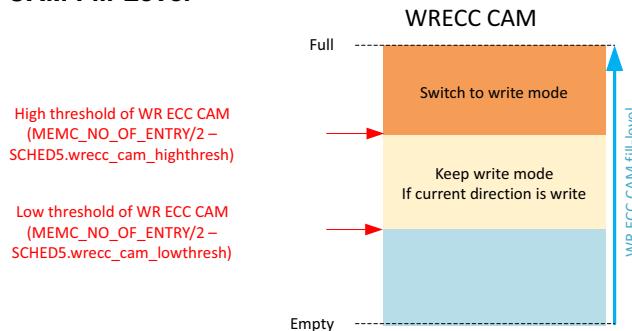
This feature can also be controlled by the following two registers:

- PERFWR1.w_max_starve
- PERFWR1.w_xact_run_length

If PERFWR1.w_max_starve > 0, after PERFWR1.w_xact_run_length number of write commands are served while WR CAM threshold exceeds the low threshold, the controller can switch to read even if the number of WR CAM entries is higher than the low threshold. After that, once the PERFWR1.w_max_starve is expired and WR CAM fill level exceeds the high threshold, the controller switches to write mode again. This is to avoid extreme read latency in case write demand is very high.

If PERFWR1.w_max_starve=0, keep write mode as long as the WR CAM fill level exceeds the low threshold (with the exception of exVPR and collisions).

[Figure 7-7](#) shows the basic image of optimize valid write ECC CAM fill-level.

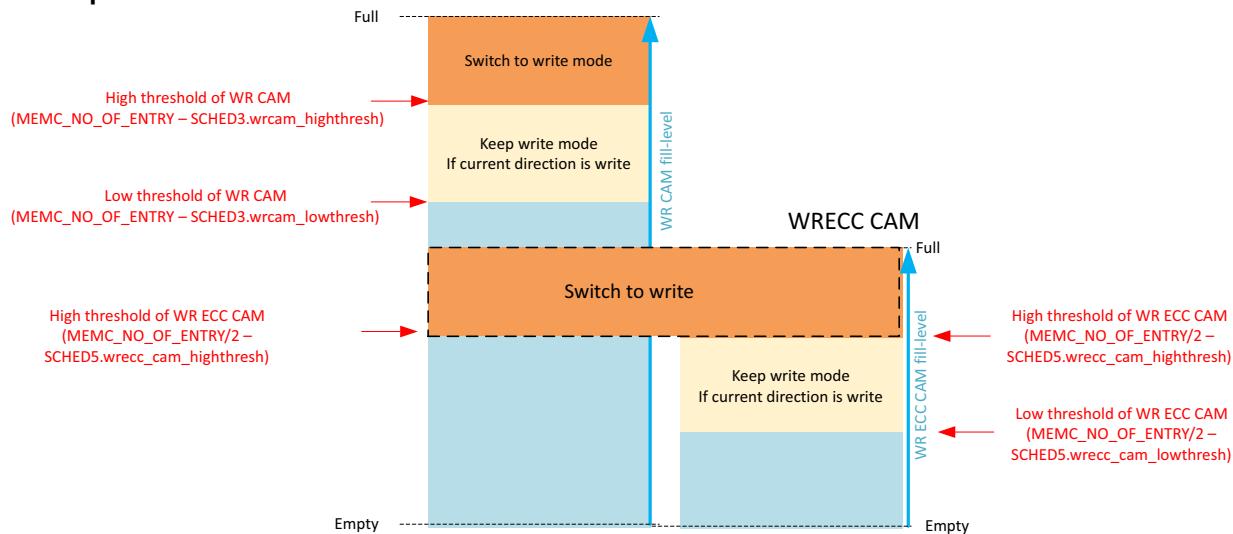
Figure 7-7 Optimize WR ECC CAM Fill-Level

When a valid write ECC CAM exceeds high threshold, switch to write mode to prevent write ECC CAM full.

The two threshold registers provided for this feature are:

- SCHED5.wrecc_cam_highthresh
- SCHED5.wrecc_cam_lowthresh

[Figure 7-8](#) shows the basic image of optimize loaded write ECC CAM fill-level.

Figure 7-8 Optimize Loaded WR ECC CAM Fill-Level

In some extreme cases, for example, one write entry corresponds to one write ECC entry, while write ECC CAM depth is half of write CAM depth. As a result, write ECC CAM is full while write CAM is not full (such as half of entries are used, not exceeds high threshold), and the write ECC entries are just loaded. In this case SoC cannot send write requests to the protected region because no write ECC credits are available.

To optimize write ECC CAM fill level, the DDRCTL switches to write mode to serve write entries when loaded write ECC entries exceed the high threshold of WR ECC CAM

($\text{MEMC_NO_OF_ENTRY} / 2 - \text{SCHED5.wrecc_cam_highthresh}$) and write entries exceed the higher threshold of WR ECC CAM ($\text{MEMC_NO_OF_ENTRY} / 2 - \text{SCHED5.wrecc_cam_highthresh}$). This function is disabled by setting SCHED5.dis_opt_load_wrecc_cam_fill_level as default.

This optimized write CAM and write ECC CAM fill level feature is enabled/disabled by SCHED.opt_wrcam_fill_level.

- In read direction, when the following condition is satisfied, switch to write direction.

- ❑ The number of write entries exceeds high threshold of WR CAM, and if `PERFWR1.w_max_starve > 0`, `PERFWR1.w_max_starve` timer is expired if previous read to write transition was done while WR CAM fill level exceeded the low threshold.
 - ❑ In Inline ECC configuration, the number of valid write ECC entries exceeds high threshold of WR ECC CAM.
 - ❑ In Inline ECC configuration, the number of loaded write ECC entries exceeds high threshold of WR ECC CAM and the number of write entries exceeds high threshold of WR ECC CAM.
 - In write direction, do not switch to read mode and do not prepare page of banks for read direction while number of write entries exceeds the low threshold of WR CAM unless `PERFWR1.w_xact_run_length` number of writes is served if `PERFWR1.w_max_starve > 0`, or the number of valid write ECC entries exceeds the low threshold of WR ECC CAM.
 - If the read to write switching is caused by the third condition, in write direction, do not switch to read mode and do not prepare page of banks for read direction until at least one write entry or write ECC entry is issued.
 - Expired-VPR and read collision are of higher priority than write CAM fill-level, and therefore it is subject to switch to read.
- For example, when the number of write entries exceed high threshold of WR CAM, switch to write direction. But, if a read collision occurs after that, switch back to read direction to flush the colliding command. Same for Expired-VPR as well.
- The priority is lower priority than write `Critical_L1`, but higher than write `Critical_L2` and write `Critical_L3`.



- A write command cannot be scheduled-out if the corresponding write data is not provided. The write data is provided by the system (AXI or HIF) or read data path if this is RMW.
- The thresholds are for number of WR CAM entries with write data ready. Therefore, if provided the write data is delayed, it is possible situation that WR CAM is (almost) full in terms of credit, but WR CAM is (almost) empty in terms of number of WR CAM entries with write data ready.

7.4 Address Collision Handling

The DDRC can execute transactions out-of-order while ensuring that all transactions appear as if they are executed in the order in which they are received. Every transaction that requires a response from the DDRC arrives with a token number which is provided back to the SoC as part of the response. Since the DDRC queues transactions prior to execution, it is possible that multiple transactions to the same SDRAM address can arrive before the first transaction to that address is issued.

For address collision, two HIF addresses are considered the same address if all of the HIF address bits (except for the LSB column values) are the same. That is, the “collision” ignores the LSB column values when comparing addresses.

So the following HIF address bits are ignored during comparison.

- HIF[4+addrmap_col_b4] if MEMC_BURST_LENGTH=32
- HIF[3+addrmap_col_b3]
- HIF[2:0]

To enforce ordering of accesses to the same address, the DDRC uses the following algorithm:

1. **New read colliding with queued read:** This collision causes no problems. The two reads can end up being executed out-of-order.
2. **New write colliding with queued write:** If write combine is enabled, the DDRC overwrites the data for the old write with that from the new write and only performs one write transaction (write combine). For more information, see “[Write Combine](#)” on page [194](#).
3. **New read (or write) colliding with queued write (or read) respectively:** In this case, the DDRC performs the following sequence:
 - a. Holds the new transactions in two deep temporary buffer.
 - b. Once the two deep temporary buffer is filled by subsequent command, applies flow control back to the SoC to prevent more transactions from arriving (hif_cmd_stall).
 - c. Flushes the internal queue holding the colliding transaction until that transaction is serviced.
 - d. Accepts the new transaction from the two deep temporary buffer and removes the flow control (hif_cmd_stall).
4. **New read colliding with both read and write:** This can happen when a read collides with an RMW command. In this case, the reads are flushed until the read collision is cleared, then the writes are flushed in the same manner as described in Step 3.
5. **New write colliding with both read and write:** This can happen when a write collides with an RMW command.
 - If write combine is enabled, the new write is combined with write part of queued RMW.
 - If write combine is disabled, the new write is held in two deep temporary buffer until the RMW has completed.
6. **New RMW colliding with queued write:** In this case, the new RMW is stored in two deep temporary buffer until the queued write is completed.

7.5 Write Combine

The write combine feature can combine multiple writes to the same address into a single write to SDRAM.

When a new write collides with a queued write in the CAM:

- If write combine is enabled, the DDRC overwrites the data for the old write with that from the new write and only performs one write transaction (write combine).
- If write combine is disabled, the DDRC performs the following sequence:
 - a. Holds the new write transaction in a temporary buffer.
 - b. Applies flow control back to the SoC to prevent more transactions from arriving.
 - c. Flushes the internal queue holding the colliding transaction until that transaction is serviced.
 - d. Accepts the new transaction and removes flow control.

If for example, there are 5 transactions, namely A,B,B,B,C (A,B, and C are addresses):

- If write combine is enabled, 3 commands go to DRAM as follows: (A-B-C).
- If write combine is disabled, 5 commands go to DRAM as follows: (A-B-B-B-C).



- Note**
- When a write combine occurs, QOS information (Write Classes, that is, NPW, VPW latency timer) of a queued write inside the DDRC is not updated by a new write from the HIF Interface.
 - When `MEMC_BURST_LENGTH=32` and the combined data has masked data, the DDRCTL will split the WR command into two commands including MWR.

7.5.1 Page-hit Limiter

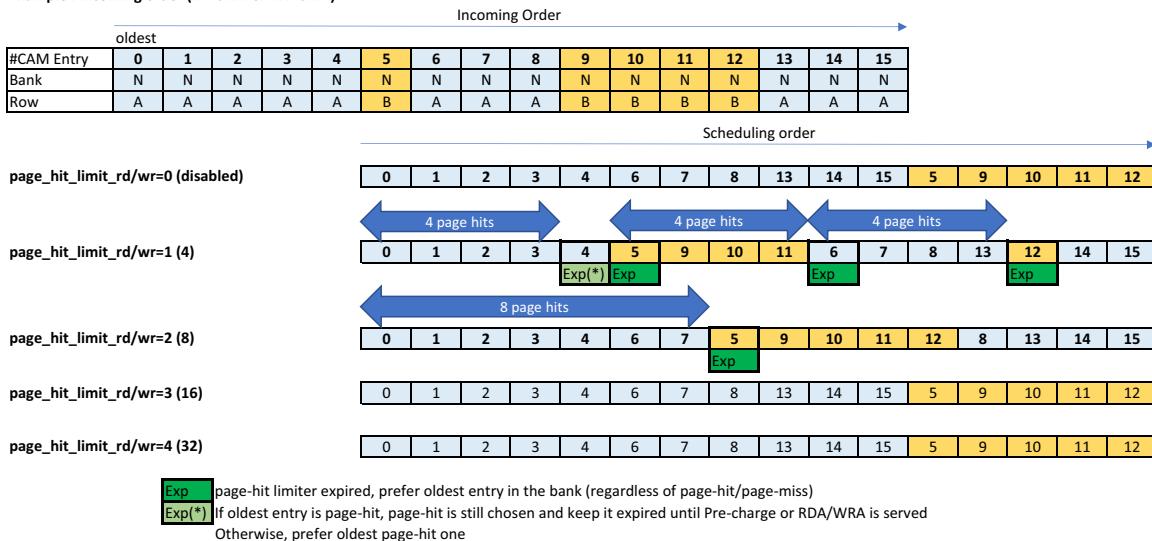
The Page-hit Limiter is an anti-starvation mechanism.

When this feature is enabled, after a certain number of page-hits are executed on a bank without PRE-ACT cycle, the next command loaded into the NTT is the oldest entry regardless of the page-hit or page-miss. This eliminates starvation caused by preferring page-hit.

[Figure 7-9](#) shows an example incoming order (RD CAM or WR CAM).

Figure 7-9 Incoming Order (RD CAM or WR CAM) Example

Example : Incoming order (RD CAM or WR CAM)



This feature works only within same priority:

- LPR and non-expired VPR
- HPR
- Expired VPR
- NPW and non-expired VPW
- Expired VPW

7.5.2 Visible Window Limiter (UMCTL2_VPRW_EN==1 Only)

The Visible Window Limiter feature is introduced to eliminate following extreme starvation:

- NPW against expired-VPW
- LPR against expired-VPR
- HPR against expired-VPR

With the enhanced CAM pointer mechanism, the visible window (age differences between the newest entry and the oldest entry) can be more than MEMC_NO_OF_ENTRY (unlimited by definition). This feature works as follows:

- Each CAM entry has a counter for this feature.
- When the entry is pushed, the counter is initialized to the value set by register SCHED1.visible_window_limit_rd/wr.
- Whenever younger entry is scheduled-out from its CAM, the counter is decremented by one.
- Once the counter reaches to '0', the entry becomes expired VPR/expired VPW.
- Within exVPR or exVPWs, the priorities are the same, therefore, this can eliminate the extreme starvation.



- Note**
- This feature works also for VPR/VPW. Depending on the latency timer value, this feature makes it expired before the latency timer expires.
 - This feature does not guarantee the actual visible window.
-

7.6 Registers Related to Command Scheduling in DDRC

The following are the registers related to the Command scheduling in DDRC:

- Low priority read transaction store:
 - PERFLPR1.lpr_max_starve
 - PERFLPR1.lpr_xact_run_length
 - SCHED1.page_hit_limit_rd
 - SCHED1.visible_window_limit_rd (UMCTL2_VPR_EN==1)
- High priority read transaction store:
 - PERFHPR1.hpr_max_starve
 - PERFHPR1.hpr_xact_run_length
 - SCHED1.page_hit_limit_rd
 - SCHED1.visible_window_limit_rd (UMCTL2_VPR_EN==1)
- Write transaction store:
 - PERFWR1.w_max_starve
 - PERFWR1.w_xact_run_length
 - SCHED1.page_hit_limit_wr
 - SCHED1.visible_window_limit_wr (UMCTL2_VPR_EN==1)
- Bus turn-around control:
 - SCHED0.prefer_write
 - SCHEDTMG0.rdwrd_idle_gap

For more information about these registers, see in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

8

Memory Scheduling

This chapter contains the following sections:

- “[Burst Mode Operation](#)” on page [200](#)
- “[Dynamic SDRAM Constraints](#)” on page [202](#)

8.1 Burst Mode Operation

This topic contains the following sections:

- “[Overview of Burst Mode Operation](#)” on page [200](#)
- “[Bus Width Selection](#)” on page [200](#)
- “[Sequential Operations](#)” on page [200](#)

8.1.1 Overview of Burst Mode Operation

When `MEMC_BURST_LENGTH=16`, DDRCTL only supports BL16.

When `MEMC_BURST_LENGTH=32`, DDRCTL supports BL16/BL32 on-the-fly.



Note When using LPDDR4/4X with `MEMC_BURST_LENGTH=32`, MR1 OP[1:0] must be set to 'b10 (BL16/BL32 on-the-fly).

8.1.2 Bus Width Selection

The DDRCTL allows you to select whether all or part of the DQ data-bus width is connected to the SDRAM. This allows the DDRCTL to connect to “full width” SDRAMs (where the width of the DQ bus is equal to the width configured in `MEMC_DRAM_DATA_WIDTH`), half of that width, or a quarter of that width. To select the DQ data bus width, set `MSTR0.data_bus_width`.

In case of half bus width mode, the half data width on the HIF is used.



Note In configurations that contain the AXI Port Interface, the XPI block generates multiple bursts on the HIF interface.

The DDRCTL does not support the optional DFI signal `dfi_data_byte_disable`, so it may be necessary to program the PHY to indicate that certain bytes are not used. Check the Synopsys PHY documentation and program the PHY as required.

8.1.3 Sequential Operations

The DDRCTL provides support for sequential burst mode operations.

SDRAM writes are always executed as aligned operations on the DFI interface in all modes. If the HIF write has an unaligned address, DDRCTL reorders the data so that the data beats appear in the correct order when the aligned write is done on the DFI interface (and hence the SDRAM interface).

[Table 8-1](#) shows examples of addressing for sequential and interleaved burst modes.

This table applies to HIF-only configurations (`UMCTL2_INCL_ARB=0`). For Arbiter configurations (`UMCTL2_INCL_ARB=1`), a fixed addressing sequence is used. Therefore, `MSTR.burst_mode` does not exist for Arbiter configurations.



Note For a sequential burst, bit 2 to bit 0 of the starting HIF address must be set to 'b000.

Table 8-1 Addressing for MSTR0.burst_rdwr = 5'b01000 (BL16) in MEMC_BURST_LENGTH = 16 (LPDDR4/5³ Configurations)

Starting HIF Address (A3 ^a A2 A1 A0)	Starting SDRAM Address (A3 A2 A1 A0)	Sequential Addressing
0000	0000	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
1000	1000	8,9,A,B,C,D,E,F,0,1,2,3,4,5,6,7

a. HIF[3] must be mapped to SDRAM A3 as ADDRMAP6.addrmap_col_b3 = 0

Table 8-2 Addressing for MSTR0.burst_rdwr = 5'b10000 (BL32) in MEMC_BURST_LENGTH = 32 (LPDDR4/5 Configurations)

Starting HIF Address (A4 ^a A3 ^b A2 A1 A0)	Starting SDRAM Address (A4 A3 A2 A1 A0)	Sequential Addressing
00000	00000	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F
01000	01000	8,9,A,B,C,D,E,F,0,1,2,3,4,5,6,7,18,19,1A,1B,1C,1D,1E,1F,10,11,12,13,14,15,16,17
10000	10000	10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
11000	11000	18,19,1A,1B,1C,1D,1E,1F,10,11,12,13,14,15,16,178,9,A,B,C,D,E,F,0,1,2,3,4,5,6,7

a. HIF[4] must be mapped to SDRAM A4 as ADDRMAP6.addrmap_col_b4 = 0.

b. HIF[3] must be mapped to SDRAM A3 as ADDRMAP6.addrmap_col_b3 = 0.



Note For LPDDR4/5, only Sequential Addressing is supported.

For LPDDR5 read, only HIF[3:0] equal to 0000 and 1000 are supported (CAS-B3 command is not supported).

8.2 Dynamic SDRAM Constraints

This topic contains the following sections:

- “[Overview of Dynamic SDRAM Constraints](#)” on page [202](#)
- “[Timing Constraints](#)” on page [202](#)

8.2.1 Overview of Dynamic SDRAM Constraints

Dynamic SDRAM constraints are the restrictions placed on the transaction scheduler by the rules of the SDRAM specification. The exact value of each constraint is programmable and varies with the specification for the exact SDRAM parts being used. These constraints must be set up before traffic is sent to the DDRCTL.

Dynamic SDRAM constraints can be subdivided into three basic categories:

- Bank constraints: It affects the transactions that can be scheduled to a given bank. See “[Timing Constraints](#)” on page [202](#).
- Rank constraints: It affects the transactions that can be scheduled to a given rank.
- Global constraints: It affects all the transactions.

Write latency (WL) and read latency (RL) descriptions for different SDRAMs are defined as follows:

- For LPDDR4/5:
 - RL and WL are directly defined in the mode register.
- For LPDDR5X:
 - the minimum gap from Activate to Write can be set separately from the minimum gap from Activate to Read.
- All commands are sent from the lower half of the bus, corresponding to the even cycle on the SDRAM bus (LPDDR4 only).
- For 1:4 frequency ratio mode, the following limitations apply on the DFI bus (LPDDR4 only):
 - All commands except refresh are sent on the first phase of the bus, corresponding to the 4n cycle on the SDRAM bus.
 - Refresh command is sent on the third phase of the bus, corresponding to the 4n+2 cycle on the SDRAM bus.

As a result of these limitations gaps occur between some commands on the SDRAM bus being one cycle longer than programmed by the timing registers as described in this section.

8.2.2 Timing Constraints

Sections “[LPDDR4 Command Timing Constraints](#)” on page [203](#), and “[LPDDR5 Command Timing Constraints](#)” on page [205](#) describe command timing constraints for LPDDR4, and LPDDR5 respectively.

8.2.2.1 LPDDR4 Command Timing Constraints

[Table 8-3](#) to [Table 8-5](#) list the command timing control registers and command timing constraints for LPDDR4.

Table 8-3 LPDDR4 Command Timing Control Registers for BL16

Control Register	Value
DRAMSET1TMG4.t_ccd	8
DRAMSET1TMG13.t_ccd_mw	tCCDMW
DRAMSET1TMG4.t_rcd	RU(tRCD/tCK)
DRAMSET1TMG2.wr2rd	WL + 1 + BL/2 + RU(tWTR/tCK)
DRAMSET1TMG0.t_ras_min	RU(tRAS/tCK)
DRAMSET1TMG1.rd2pre	BL/2 + max{(8, RU(tRTP/tCK)} – 8
DRAMSET1TMG0.wr2pre	WL + 1 + BL/2+RU(tWR/tCK)
DRAMSET1TMG4.t_rp	RU(tRP/tCK)
DRAMSET1TMG4.t_rrd	RU(tRRD /tCK)
DRAMSET1TMG13.t_ppd	tPPD
DRAMSET1TMG25.rda2pre	
DRAMSET1TMG2.rd2wr (see Table 101 in JESD209-4C)	DQ ODT is disabled RL + RU(tDQSCK(max)/tCK) + BL/2 - WL + tWPRE + RD(tRPST)
	DQ ODT is enabled RL + RU(tDQSCK(max)/tCK) + BL/2 + RD(tRPST) – ODTLon – RD(tODTon,min/tCK) + 1



RANKTMG1 . rd2wr_dr must be larger than or equal to the value of DRAMSET1TMG2 . rd2wr.

Table 8-4 LPDDR4 Command Timing Control Registers for BL32

Control Register	Value
DRAMSET1TMG34.t_ccd_blx2	16
DRAMSET1TMG34.t_ccd_mw_blx2	tCCDMW + 8
DRAMSET1TMG34.wr2rd_blx2	WL + 1 + BL/2 + RU(tWTR/tCK)
DRAMSET1TMG35.rd2pre_blx2	BL/2 + max{8, RU(tRTP/tCK)} – 8
DRAMSET1TMG35.wr2pre_blx2	WL + 1 + BL/2 + RU(tWR/tCK)

Control Register	Value
DRAMSET1TMG35.rda2pre_blx2	WL + 1 + BL/2 + nWR
DRAMSET1TMG34.rd2wr_blx2	DQ ODT is disabled. RL + RU(tDQSCK(max)/tCK) + BL/2 - WL + tWPRE + RD(tRPST)
	DQ ODT is enabled. RL + RU(tDQSCK(max)/tCK) + BL/2 + RD(tRPST) - ODTLon - RD(tODTon,min/tCK) + 1



Note Table 8-4 is available only if MEMC_BURST_LENGTH=32.

Table 8-5 Command Timing Constraints for Same Banks

Next CMD	Active	Read	Write	Mask Write	Precharge
Current CMD					
ACTIVE	Illegal	t_rcd	t_rcd	t_rcd	t_ras_min
READ (RD16)	Illegal	t_ccd	rd2wr	rd2wr	rd2pre
WRITE (WR16)	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
READ (RD32) ^a	Illegal	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	rd2pre_blx2
WRITE (WR32) ^a	Illegal	wr2rd_blx2	t_ccd_blx2	t_ccd_mw_blx2	wr2pre_blx2
MASK WRITE	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
PRECHARGE	t_rp	Illegal	Illegal	Illegal	t_ppd
READ (RD16) with AP	rda2pre + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR16) with AP	wra2pre + t_rp	Illegal	Illegal	Illegal	N/A
READ (RD32) ^a with AP	rda2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR32) ^a with AP	wra2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
MASK WRITE with AP	wra2pre + trp	Illegal	Illegal	Illegal	N/A

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

Table 8-6 Command Timing Constraints for Different Banks

Next CMD	Active	Read	Write	Mask Write	Precharge
Current CMD					
ACTIVE	t_rrd	1	1	1	1
READ (RD16)	1	t_ccd	rd2wr	rd2wr	1

Next CMD					
Current CMD	Active	Read	Write	Mask Write	Precharge
WRITE (WR16)	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1
WRITE (WR32) ^a	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE	1	wr2rd	t_ccd	t_ccd	1
PRECHARGE	1	1	1	1	t_ppd
READ (RD16) with AP	1	t_ccd	rd2wr	rd2wr	1
WRITE (WR16) with AP	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a with AP	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1
WRITE (WR32) ^a with AP	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE with AP	1	wr2rd	t_ccd	t_ccd	1

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

8.2.2.2 LPDDR5 Command Timing Constraints

Sections “[BG Mode](#)” on page 205 and “[16B Mode](#)” on page 208 describe the LPDDR5 command timing constraints for BG mode and 16B mode, respectively.

8.2.2.2.1 BG Mode

[Table 8-7](#) to [Table 8-15](#) list the command timing control registers and command timing constraints for LPDDR5 in BG mode.

Table 8-7 LPDDR5 Command Timing Control Registers for BL16 in BG Mode

Control Register	Value
t_ccd	BL/n (same BG)
t_ccd_s	BL/n (different BG)
t_ccd_mw	4*BL/n_max
t_rcd	RU(tRCD/tCK)
t_rcd_write (for LPDDR5X)	RU(tRCD(for write)/tCK)
wr2rd	WL + BL/n_max + RU(tWTR_L/tCK)
wr2rd_s	WL + BL/n_min + RU(tWTR_S/tCK)
t_ras_min	RU(tRAS/tCK)
rd2pre	BL/n_min + RU(tRBTP/tCK)

Control Register	Value
wr2pre	WL + BL/n_min + RU(tWR/tCK)
t_rp	RU(tRPpb/tCK)
t_rrd	RU(tRRD_L/tCK)
t_rrd_s	RU(tRRD_S/tCK)
t_ppd	2
DQ ODT is disabled	
rd2wr	RL + BL/n_max + RU(tWCKDQO(max)/tCK) - WL
rd2wr_s	RL + BL/n_min + RU(tWCKDQO(max) /tCK) - WL
DQ ODT is enabled	
rd2wr	RL + BL/n_max + RU(tWCKDQO(max)/tCK) + RD(tRPST/tCK) - ODTLon - RD(tODTon(min)/tCK)
rd2wr_s	RL + BL/n_min + RU(tWCKDQO(max)/tCK) + RD(tRPST/tCK) - ODTLon - RD(tODTon(min)/tCK)



RANKTMG1.rd2wr_dr must be larger than or equal to the value of DRAMSET1TMG24.rd2wr_s.

Table 8-8 LPDDR5 Command Timing Control Registers for BL32 in BG Mode

Control Register	Value
t_ccd_blx2	BL/n (same BG, BL32)
t_ccd_s_blx2	BL/n (different BG, BL32)
t_ccd_mw_blx2	2.5*BL/n_max(BL32)
wr2rd_blx2	WL + BL/n_max(BL32) + RU(tWTR_S/tCK)
wr2rd_s_blx2	WL + BL/n_min(BL32) + RU(tWTR_S/tCK)
rd2pre_blx2	BL/n_min(BL32) + RU(tRBTP/tCK)
wr2pre_blx2	WL + BL/n_min + 1 + RU(tWR/tCK)
rd2wr_blx2	tRTW (See table 347 and table 350 in JESD209-5C)
rd2wr_s_blx2	tRTW (See table 348 and table 351 in JESD209-5C)



Table 8-8 is available only if MEMC_BURST_LENGTH=32.

Table 8-9 Command Timing Constraints for Same Banks in Same Bank Group

Next CMD Current CMD	Active	Read	Write	Mask Write	Precharge
ACTIVE	Illegal	t_rcd	t_rcd (for LPDDR5) t_rcd_write (for LPDDR5X)	t_rcd	t_ras_min
READ (RD16)	Illegal	t_ccd	rd2wr	rd2wr	rd2pre
WRITE (WR16)	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
READ (RD32) ^a	Illegal	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	rd2pre_blx2
WRITE (WR32) ^a	Illegal	wr2rd_blx2	t_ccd_blx2	t_ccd_mw_blx2	wr2pre_blx2
MASK WRITE	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
PRECHARGE	t_rp	Illegal	Illegal	Illegal	t_ppd
READ (RD16) with AP	rda2pre + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR16) with AP	wra2pre + t_rp	Illegal	Illegal	Illegal	N/A
READ (RD32) ^a with AP	rda2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR32) ^a with AP	wra2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
MASK WRITE with AP	wra2pre + t_rp	Illegal	Illegal	Illegal	N/A

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

Table 8-10 Command Timing Constraints for Different Banks in Same Bank Group

Next CMD Current CMD	Active	Read	Write	Mask Write	Precharge
ACTIVE	t_rrd	1	1	1	1
READ (RD16)	1	t_ccd	rd2wr	rd2wr	1
WRITE (WR16)	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1
WRITE (WR32) ^a	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE	1	wr2rd	t_ccd	t_ccd	1
PRECHARGE	1	1	1	1	t_ppd
READ (RD16) with AP	1	t_ccd	rd2wr	rd2wr	1
WRITE (WR16) with AP	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a with AP	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1

Next CMD					
Current CMD	Active	Read	Write	Mask Write	Precharge
WRITE (WR32) ^a with AP	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE with AP	1	wr2rd	t_ccd	t_ccd	1

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

Table 8-11 Command Timing Constraints for Different Bank Groups

Next CMD					
Current CMD	Active	Read	Write	Mask Write	Precharge
ACTIVE	t_rrd	1	1	1	1
READ (RD16)	1	t_ccd_s	rd2wr_s	rd2wr_s	1
WRITE (WR16)	1	wr2rd_s	t_ccd_s	t_ccd_s	1
READ (RD32) ^a	1	t_ccd_s_blx2	rd2wr_s_blx2	rd2wr_s_blx2	1
WRITE (WR32) ^a	1	wr2rd_s_blx2	t_ccd_s_blx2	t_ccd_s_blx2	2
MASK WRITE	1	wr2rd_s	t_ccd_s	t_ccd_s	1
PRECHARGE	1	1	1	1	t_ppd
READ (RD16) with AP	1	t_ccd_s	rd2wr_s	rd2wr_s	1
WRITE (WR16) with AP	1	wr2rd_s	t_ccd_s	t_ccd_s	1
READ (RD32) ^a with AP	1	t_ccd_s_blx2	rd2wr_s_blx2	rd2wr_s_blx2	1
WRITE (WR32) ^a with AP	1	wr2rd_s_blx2	t_ccd_s_blx2	t_ccd_s_blx2	1
MASK WRITE with AP	1	wr2rd_s	t_ccd_s	t_ccd_s	1

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

8.2.2.2.2 16B Mode

Table 8-12 to Table 8-15 list the command timing control registers and command timing constraints for LPDDR5 in 16B mode.

Table 8-12 LPDDR5 Command Timing Control Registers in 16B Mode

Control Register	Value
t_ccd	BL/n
t_ccd_mw	4*BL/n
t_rcd	RU(tRCD/tCK)
t_rcd_write (for LPDDR5x)	RU(tRCD(for write)/tCK)

Control Register	Value
wr2rd	WL + BL/n + RU(tWTR /tCK)
t_ras_min	RU(tRAS/tCK)
rd2pre	BL/n + RU(tRBTP/tCK)
wr2pre	WL + BL/n + RU(tWR/tCK)
t_rp	RU(tRP/tCK)
t_rrd	RU(tRRD /tCK)
t_ppd	2
rd2wr	DQ ODT is disabled RL + BL/n + RU(tWCKDQQ(max)/tCK) – WL DQ ODT is enabled RL + BL/n + RU(tWCKDQQ(max) /tCK) + RD(tRPST/tCK) – ODTLon – RD(tODTon(min)/tCK)



Note RANKTMG1.rd2wr_dr must be larger than or equal to the value of DRAMSET1TMG2.rd2wr.

Table 8-13 LPDDR5 Command Timing Control Registers for BL32 in 16B Mode

Control Register	Value
t_ccd_blx2	BL/n(BL32)
t_ccd_mw_blx2	2.5*BL/n(BL32)
wr2rd_blx2	WL + BL/n(BL32) + RU(tWTR /tCK)
rd2pre_blx2	BL/n(BL32) + RU(tRBTP/tCK)
wr2pre_blx2	WL + BL/n(BL32) + 1 + RU(tWR/tCK)
rd2wr_blx2	tRTW (See table 349 and table 352 in JESD209-5C)



Note Table 8-13 is available only if MEMC_BURST_LENGTH=32.

Table 8-14 Command Timing Constraints for Same Banks

Next CMD Current CMD	Active	Read	Write	Mask Write	Precharge
ACTIVE	Illegal	t_rcd	t_rcd (for LPDDR5) t_rcd_write (for LPDDR5X)	t_rcd	t_ras_min

Next CMD					
Current CMD	Active	Read	Write	Mask Write	Precharge
READ (RD16)	Illegal	t_ccd	rd2wr	rd2wr	rd2pre
WRITE (WR16)	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
READ (RD32) ^a	Illegal	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	rd2pre_blx2
WRITE (WR32) ^a	Illegal	wr2rd_blx2	t_ccd_blx2	t_ccd_mw_blx2	wr2pre_blx2
MASK WRITE	Illegal	wr2rd	t_ccd	t_ccd_mw	wr2pre
PRECHARGE	t_rp	Illegal	Illegal	Illegal	t_ppd
READ (RD16) with AP	rda2pre + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR16) with AP	wra2pre + t_rp	Illegal	Illegal	Illegal	N/A
READ (RD32) ^a with AP	rda2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
WRITE (WR32) ^a with AP	wra2pre_blx2 + t_rp	Illegal	Illegal	Illegal	N/A
MASK WRITE with AP	wra2pre + t_rp	Illegal	Illegal	Illegal	N/A

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

Table 8-15 Command Timing Constraints for Different Banks

Next CMD					
Current CMD	Active	Read	Write	Mask Write	Precharge
ACTIVE	t_rrd	1	1	1	1
READ (RD16)	1	t_ccd	rd2wr	rd2wr	1
WRITE (WR16)	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1
WRITE (WR32) ^a	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE	1	wr2rd	t_ccd	t_ccd	1
PRECHARGE	1	1	1	1	t_ppd
READ (RD16) with AP	1	t_ccd	rd2wr	rd2wr	1
WRITE (WR16) with AP	1	wr2rd	t_ccd	t_ccd	1
READ (RD32) ^a with AP	1	t_ccd_blx2	rd2wr_blx2	rd2wr_blx2	1
WRITE (WR32) ^a with AP	1	wr2rd_blx2	t_ccd_blx2	t_ccd_blx2	1
MASK WRITE with AP	1	wr2rd	t_ccd	t_ccd	1

a. RD32 and WR32 are available only if MEMC_BURST_LENGTH=32.

8.2.2.3 LPDDR5X Command Timing Constraints

The DDRCTL has LPDDR5X specific register fields, which are implemented only when the hardware parameter `MEMC_LPDDR5X` is defined:

- `MSTR0.lpddr5x`
- `DRAMSET1TMG1.t_rcd_write`
- `DERATEVAL1.derated_t_rcd_write`

Both `DRAMSET1TMG1.t_rcd_write` and `DERATEVAL1.derated_t_rcd_write` are applied only when `MSTR0.lpddr5x` is set to '1' (LPDDR5X devices).

- The minimum gap between an ACT-2 command and a WR command (i.e. tRCD for write) is shorter in LPDDR5X devices than LPDDR5 devices, which results in a slight improvement of DDR utilization in LPDDR5X devices.
- The minimum gap between an ACT-2 command and a RD/Masked WR command is the same for LPDDR5X and LPDDR5

8.2.2.4 Limitation With DRAMSET1TMG4.t_rcd = 1

The input `DRAMSET1TMG4.t_rcd` indicates the minimum time from Activate to a read or write to the same bank, except LPDDR5X. The input `DRAMSET1TMG1.t_rcd_write` indicates the minimum time from active to write to the same bank for LPDDR5X. The DDRCTL has the following limitation when `DRAMSET1TMG4.t_rcd` is programmed to '1': in 1:2 frequency ratio mode setting this register is programmed to '1' when the Activate to Read/Write timing requirement at the DRAM is either one or two cycles. The DDRCTL puts a minimum of two `core_ddrc_core_clk` cycles between an Activate and Read/Write request, and this results in four SDRAM clock cycles gap at the SDRAM interface.

This limitation can be avoided by programming DRAM Additive Latency appropriately as described as follows:

`DRAMSET1TMG4.t_rcd` is programmed as: tRCD

where tRCD is RAS-to-CAS delay of the DRAM.

`DRAMSET1TMG1.t_rcd_write` is programmed as : tRCD (for write, LPDDR5X only)

8.2.2.5 Byte Mode (x8) Consideration

DDRCTL supports byte mode (x8) devices.

For details, refer to [Figure 5-1](#) on page 128, [Figure 5-4](#) on page 131, and [Figure 5-7](#) on page 137 in the section “[DDRCTL Configurations](#)” on page 127.

To support byte mode devices, the following timing parameters must be calculated properly and used to set appropriate DDRCTL registers:

- LPDDR4: WL, RL, nWR, nRTP, tWR, tWTR.
- LPDDR5: RL, nWR, nRBTP, tWR, tWTR (tWTR_S, tWTR_L for BG mode), tWCKENL_RD.

Note, that the following Mode Registers in SDRAMs are not directly related to the DDRCTL behavior, but they need to be programmed by software to support byte mode Vref/ODT/DQ Calibration/Training appropriately during initialization:

- LPDDR4
 - MR12: CBT mode for byte mode
 - MR22: ODTD for x8 2ch (Byte) mode
 - MR32: Byte mode Vref Selection
- LPDDR5
 - MR12: VBS (V REF (CA) Byte Select)
 - MR14: VLDC (V REF DQ Lower byte copy)
 - MR17: X8 ODTD Lower (CA/CS/CK ODT termination disable, Lower Byte select), X8 ODTD Upper (CA/CS/CK ODT termination disable, Upper Byte select)
 - MR30: DCA for Lower Byte (DCAL), DCA for Upper Byte (DCAU)
 - MR31: Lower-byte per-bit control Register for DQ Calibration
 - MR32: Upper-byte per-bit control Register for DQ Calibration

8.2.3 Preamble and Postamble (LPDDR4)

8.2.3.1 LPDDR4 Constraints

LPDDR4 supports write preamble of 2*tCK and programmable write postamble of 0.5 tCK or 1.5 tCK:

- Write postamble is selected through MR3[1] - adjust INT4.emr2.
- If write postamble of 1.5 tCK is used, the following timing values are affected and you must program the registers accordingly:
 - Adjust RANKTMG0.diff_rank_wr_gap for multi-rank systems: program this with N+1.

LPDDR4 supports read preamble of 2*tCK and programmable read postamble of 0.5 tCK or 1.5 tCK:

- Read preamble is selected "Static" or "Toggle" through MR1[3]. Both preambles are 2*tCK. - adjust INITMG3.mr.
- Read postamble is selected through MR1[7] - adjust INITMG3.mr.
- If read postamble of 1.5 tCK is used, the following timing values are affected and you must program the registers accordingly:
 - Adjust RANKTMG0.diff_rank_rd_gap for multi-rank systems: program this with N+1.

9

Periodic Memory and PHY Maintenance

This chapter contains the following sections:

- “Refresh Controls” on page [214](#)
- “Refresh Management (RFM)” on page [223](#)
- “ZQ Calibration” on page [228](#)
- “Controller Assisted Drift Tracking by MRR Snooping (LPDDR5)” on page [235](#)
- “Enhanced Incremental Periodic Phase Training (PPT2)” on page [239](#)
- “Burst DFI Control Update Request for Core VDD Change” on page [249](#)

9.1 Refresh Controls

This topic contains the following sections:

- “[Overview of Refresh Controls](#)” on page [214](#)
- “[Refresh Using Direct Software Request of Refresh Command](#)” on page [214](#)
- “[Refresh Using Auto-Refresh Feature](#)” on page [215](#)
- “[Automatic Temperature Derating](#)” on page [219](#)
- “[Signals Related to Refresh Controls](#)” on page [221](#)
- “[Registers Related to Refresh Controls](#)” on page [222](#)
- “[Dynamic SDRAM Rank Constraints](#)” on page [222](#)

9.1.1 Overview of Refresh Controls

Refresh can be issued using the auto-refresh feature in the DDRCTL or using the direct software request of the refresh command. The RFSHCTL0.dis_auto_refresh register bit selects the refresh method.



Note Even if the contents of DRAM are not to be preserved, some devices may require periodic refresh operations while VDDQ, VDD, and VPP are asserted and the DRAM state is IDLE or Powerdown. See the DRAM documentation before configuring it.

The purpose of refresh control is to:

- Reduce the bandwidth impact of refresh cycles.
- Increase the likelihood of refreshes being serviced during idle periods.
- Provide fine-gain control of the trading-off the previous benefits (to gather refreshes) versus the increased worst case latencies associated with gathering refreshes.
- Allow traffic to flow to other ranks while a given rank is being refreshed (for multi-rank configurations of the DDRCTL only).

9.1.2 Refresh Using Direct Software Request of Refresh Command

Follow these steps to put the DDRCTL in direct software request of refresh command mode:

1. Set the RFSHCTL0.dis_auto_refresh bit to ‘1’. When the register bit set, the DDRCTL checks for any pending refreshes. Any pending refreshes are issued right away using the ‘critical refresh’ feature inside the DDRCTL. After these refreshes are issued, all the refresh timers inside the DDRCTL are reset to ‘0’. They are re-activated only when the auto-refresh feature is enabled.

Any pending refreshes are issued in same way, in this case after exiting self-refresh.

2. The SoC must keep track of the refresh requirements of the SDRAM.
3. The refresh command can be issued by setting the register bits OPREFCTRL*.rank*_refresh to ‘1’. When the rank*_refresh request is stored in the DDRCTL, the corresponding register bit is automatically cleared. The SoC can initiate a rank*_refresh operation only if OPCTRLSTAT.rank*_refresh_busy is low. The DDRCTL issues refresh to the SDRAM at the earliest.
4. Software-driven refresh commands for each rank are loaded into a buffer, and are issued by the DDRCTL on the DFI as soon as it is legal to do so (the DDRCTL controller must wait tRFC(min) between

each refresh request). If the buffer saturates, the OPCTRLSTAT.rank*_refresh_busy remains asserted to prevent software from initiating further refreshes.

Depending on all-bank or per-bank refresh mode, the buffer size is different.

Table 9-1 Buffer Size Depending on All-Bank and Per-Bank Refresh Modes

All-Bank and Per-Bank Refresh Mode	Buffer Size
Per-bank refresh	65 entries
All-bank refresh	9 entries



- You can give a burst of back-to-back refresh commands without polling the OPCTRLSTAT.rank*_refresh_busy register field to reduce the time between the software refresh commands to a minimum with the following risk:
 - If the refresh buffer is not empty when the burst starts or the burst is longer than 9 refresh commands, the buffer saturates (OPCTRLSTAT.rank*_refresh_busy is asserted) and some APB writes given in the burst are not added to the buffer. This means that the number of APB writes do not reflect the number of REF commands sent out to the memory.
- For LPDDR4/LPDDR5 configurations manual refreshes (RFSHCTL0.dis_auto_refresh=1) may not be used if the PHY Master Interface is enabled (DFIPHYMSTR.dfi_phymstr_en=1).

9.1.3 Refresh Using Auto-Refresh Feature

The DDRCTL provides advanced refresh controls. Besides fully-configurable refresh constraints (tRFC(min) and tREFI), the DDRCTL can also be programmed to gather refreshes to each rank of SDRAM to reduce the bandwidth consumed by refreshes and to increase the likelihood that refreshes can be serviced during an idle period.

Fine-grain control of the refreshes ensures these benefits can be balanced against worst case latencies associated with servicing refreshes together. Staggered refresh timers for multi-rank configurations of the DDRCTL allow transactions to continue to other ranks while refreshes are taking place to just one rank.

To minimize the worst case impact of a forced refresh cycle, the DDRCTL can be programmed to issue single refreshes at a time by forcing RFSHMOD0.refresh_burst=0. It can be programmed to burst up to 8 refreshes (RFSHMOD0.refresh_burst=7). With per-bank refresh enabled (RFSHMOD0.per_bank_refresh=1), the maximum refresh burst supported by DDRCTL is 64 (RFSHMOD0.refresh_burst=63).

9.1.3.1 Single Refresh

When using single refresh (RFSHMOD0.refresh_burst=0), the DDRCTL issues refreshes every time the refresh timer (tREFI) expires. This is the optimal mode of operation for systems that minimize the maximum latency associated with refresh cycles.

9.1.3.2 Burst Refresh

When burst refresh is enabled (`RFSHMOD0.refresh_burst > 0`), the DDRCTL issues refreshes in bursts of (`RFSHMOD0.refresh_burst+1`) refreshes at one time. Bursting refreshes reduces the total latency associated with those refreshes by reducing the number of pre-charges and activates required for refresh, as banks must be precharged only once to perform the entire group of refreshes, instead of once for each refresh.

9.1.3.3 Speculative Refresh

When t_{REFI} has expired at least once, the DDRCTL can also perform speculative refreshes. This is done by automatically inserting refreshes when there are no transactions pending in the CAM to a rank/bank address.

The `RFSHSET1TMG0.refresh_to_x1_x32` register determines how long the transactions must not be stored in the CAM before considering inserting these speculative refreshes. Each time a speculative refresh is performed, the count of t_{REFI} expirations is decremented, and thereby increasing the time before a critical refresh is required. This also ensures that speculative refreshes never occur more often than is required to keep the SDRAM properly refreshed.

If a new read or write transaction is accepted by the DDRCTL during speculative refresh, the DDRCTL services it as soon as legally possible. Most often it entails waiting for the required NOP cycles after a refresh before performing an activate and then servicing the read or write. If the DDRCTL has begun closing pages for a speculative refresh but has not yet issued the refresh when the new transaction arrives, the speculative refresh is canceled.

9.1.3.4 Per-Bank Refresh

If `RFSHMOD0.per_bank_refresh` is set to '1', the DDRCTL performs per-bank refreshes instead of all-bank refreshes. In this case, `RFSHTMG1.t_rfc_min` must be set to the appropriate values for per-bank refresh (t_{RFCpb}). In this mode, the DDRCTL keeps track of which bank is being refreshed at any time, and is able to schedule commands to other banks immediately before and after the per-bank refresh commands, resulting in potential efficiency gains. To improve the accuracy of per-bank refresh timing, set `RFSHTMG0.t_refi_x1_sel` to '1' and program `RFSHSET1TMG0.t_refi_x1_x32` accordingly.

The per-bank refresh command can be issued in any bank order. The DDRCTL can send the per-bank refresh to banks as a priority, where no transactions pending in the CAM or the bank is precharged. Also, the DDRCTL does not send the per-bank refresh command to multiple ranks at the same time, as the command contains a bank address, and the DDRCTL cannot specify multiple bank addresses with one command.

If the refresh burst (`RFSHMOD0.refresh_burst`) is set to its maximum value of 63, it means that the DDRCTL can postpone up to 64 per-bank refreshes, in accordance with the JEDEC specification. However, in this case the time between two per-bank refreshes to a specific bank can possibly exceed $9 \times t_{REFI}$. It is not clear from the JEDEC specification whether this is a violation or not; to avoid this, `RFSHMOD0.refresh_burst` must be set to the maximum value of 55.

If the refresh burst (RFSHMOD0.refresh_burst) is programmed with a large value in per-bank Refresh mode, and bursting per-bank refreshes start, it may take a longer time than bursting all-bank refreshes. The three possible cases where bursting per-bank refreshes may occur are:

1. Random traffic
- The DDRCTL may not be able to issue a speculative refresh because the CAM always has transactions to some banks. This can be avoided with appropriate register settings. For example, setting smaller values to RFSHCTL0.refresh_burst or RFSHMOD0.refresh_to_x1_x32.
2. Toggling RFSHCTL0.refresh_update_level
3. Hardware Fast Frequency Change (HWFFC)

After the 2 or 3 operation, the DDRCTL starts bursting refreshes to compensate for refresh timing parameters which may have been updated.

If DERATEEN.derate_enable is set to 1'b1 and RFSHMOD0.auto_refab_en is set to a value greater than 0, the DDRCTL may switch automatically from per-bank refresh to all-bank refresh if the refresh period is too small due to temperature derating.

In this case, the DDRCTL adjusts the following refresh parameters internally when sending REFab instead of REFab:

- RFSHSET1TMG0.t_refi_x1_x32 is multiplied by 8 (tREFIab instead of tREFIpB).
- RFSHSET1TMG1.t_rfc_min_ab is used instead of RFSHSET1TMG1.t_rfc_min (tRFCab instead of tRFCpb).
- RFSHSET1TMG3.refresh_to_ab_x32 is used instead of RFSHSET1TMG1.refresh_to_x1_x32.
- RFSHMOD0.refresh_burst is divided by 8.

Once temperature derating allows it, the DDRCTL switches back to sending REFab and adjusts above internal timings to their original software programmed values for per-bank refresh.

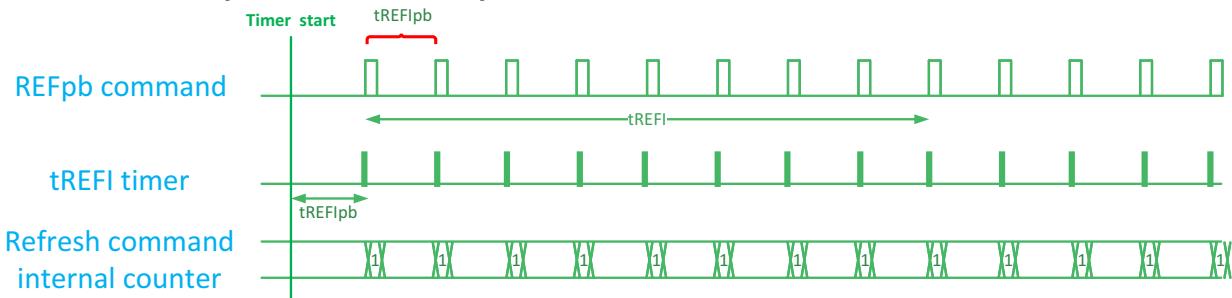
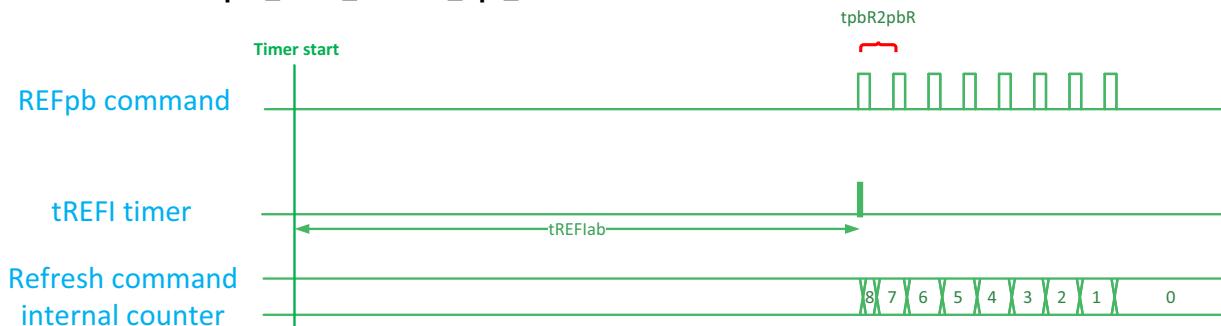


Note In LPDDR5 mode, bank address BA3 and bank group address BG1 are "don't care" for 16B and BG mode, and then per-bank refresh is performed to two banks simultaneously.

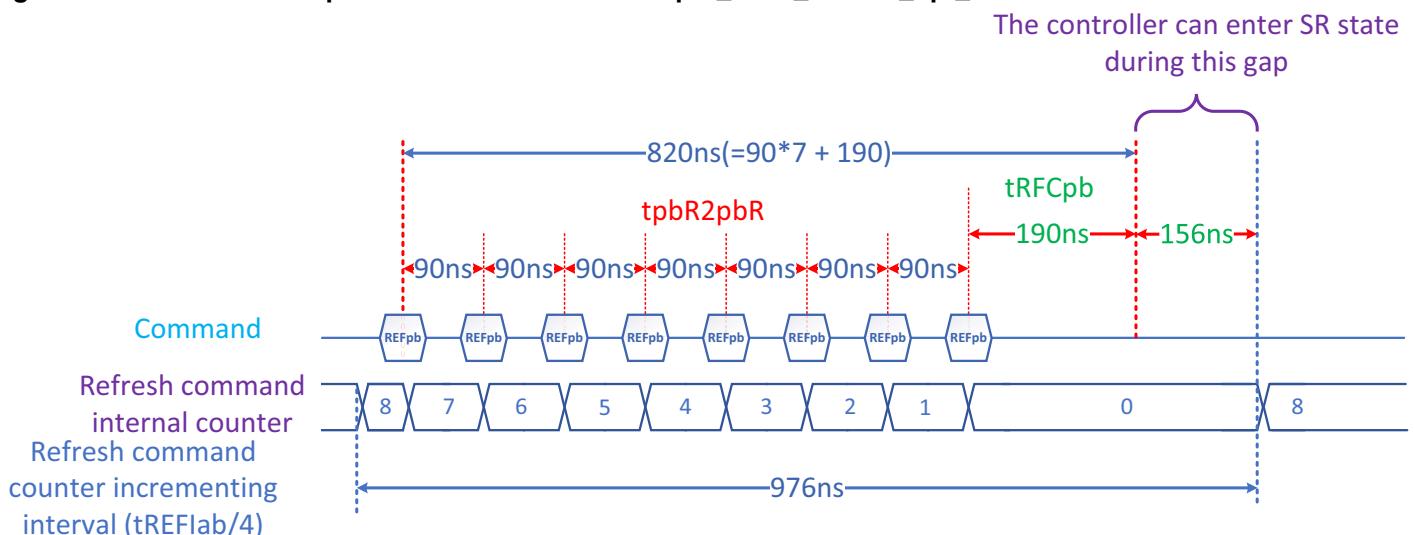
9.1.3.5 Per-Bank Refresh Optimization

The controller supports optimized per-bank refresh behavior for LPDDR4 and LPDDR5 when RFSHMOD0.per_bank_refresh_opt_en=1. If optimized per-bank refresh is enabled, the internal per-bank refresh command counter is incremented by 8 every tREFIab timing, and per-bank refresh commands can be issued more efficiently. If the automatic temperature derating feature is enabled (DERATECTL0.derate_enable=1) along with per-bank refresh (RFSHMOD0.per_bank_refresh=1), optimized per-bank refresh should be enabled. Also, if Optimized Refresh Mode (MR0:OP[3] and MR25:OP[7]) is enabled in LPDDR5 SDRAM, it is recommended to enable the optimized per-bank refresh.

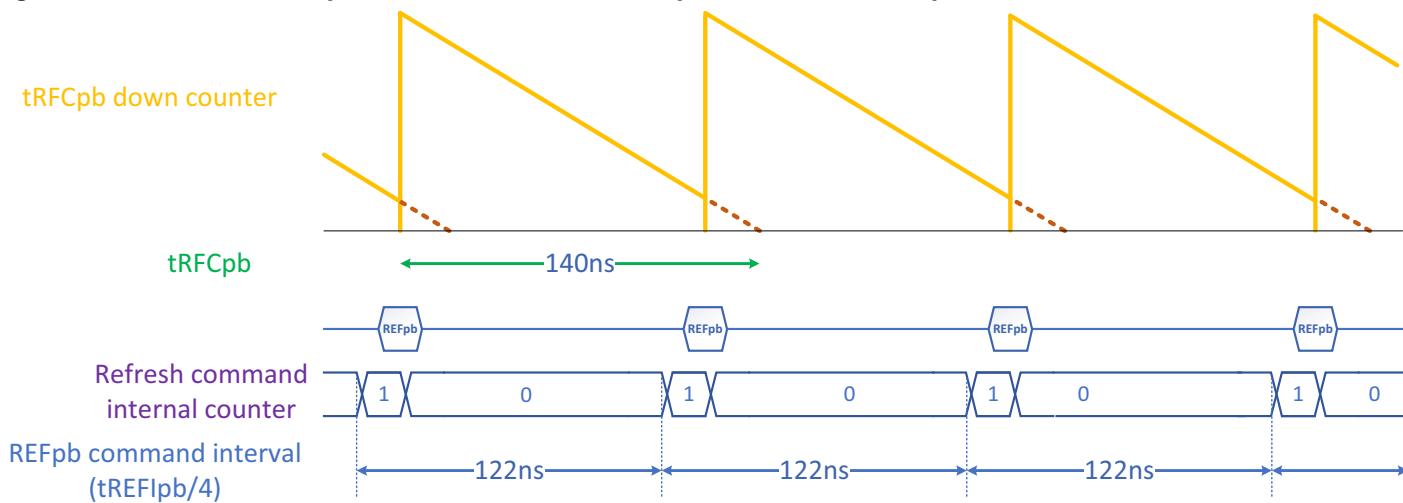
Figure 9-1 and Figure 9-2 illustrate when optimized per-bank refresh is disabled and enabled, respectively.

Figure 9-1 RFSHMOD0.per_bank_refresh_opt_en=0**Figure 9-2 RFSHMOD0.per_bank_refresh_opt_en=1**

The controller cannot enter SR state when Refresh Multiplier is 0.25x and `RFSHMOD0.per_bank_refresh_opt_en=0`, because REFpb commands are issued constantly ([Figure 9-4](#)).

Figure 9-3 Refresh Multiplier 0.25x and RFSHMOD0.per_bank_refresh_opt_en=1

If Refresh Multiplier is 0.25x and `RFSHMOD0.per_bank_refresh_opt_en=1`, REFpb commands are issued with a minimum gap of `tpbR2pbR`, and the controller can enter SR state. An example with `tRFCpb=140ns` is shown in [Figure 9-4](#).

Figure 9-4 Refresh Multiplier 0.25x and RFSHMOD0.per_bank_refresh_opt_en=0

9.1.3.6 Constraints on refresh_timerX_start_value_x32

The `refresh_timerX_start_value_x32` register fields control the relative timing of refreshes to different ranks. The register fields must obey the following constraint:

```
refresh_timerX_start_value_x32 + RoundUp(RFSHTMG1.t_rfc_min/32) <
RFSHSET1TMG0.t_refi_x1_x32
```

If `RFSHTMG0.t_refi_x1_sel` is set, `RoundDown(RFSHSET1TMG0.t_refi_x1_x32/32)` must be used in the previous constraint.

Also, note that the register field `refresh_timerX_start_value_x32` cannot be changed after initialization. So the value of the register `RFSHSET1TMG0.t_refi_x1_x32` used previously must be anticipated as the minimum value which is used, considering frequency change, fine granularity refresh, and so on.

If `DERATECTL0.derate_enable=1`, use minimum `RFSHSET1TMG0.t_refi_x1_x32/4` (for LPDDR4) or `RFSHSET1TMG0.t_refi_x1_x32/8` (for LPDDR5) as value in calculation of `refresh_timerX_star_value_x32` to consider possible effect of derating in high temperature case. All ranks of `refresh_timerX_star_value_x32` should have the same value.

9.1.4 Automatic Temperature Derating

The DDRCTL includes functionality to automatically read the MR4 register in SDRAMs, and to derate the refresh rate and certain timing parameters accordingly. This functionality can be enabled by setting the `DERATECTL0.derate_enable` input to '1'.

The read interval with which the DDRCTL performs MRR operations from MR4 is defined (in clock periods) by the `DERATEINT.mr4_read_interval` input. This value depends on the maximum expected temperature gradient. For more information, refer to the LPDDR4/LPDDR5 JEDEC Specification.

For multi-rank systems, the derated timing values (including refresh rate) are adjusted for all ranks in accordance with the temperature derating value of the worst device.

By default, derating logic uses MR4's TUF flag (`MR4[7]`) as a valid flag before evaluating a new MR4 value from `MR4[2:0]`. If `MR4[7]=1'b0`, then `MR4[2:0]` is ignored and existing refresh rate/timing parameters

are maintained. The use of MR4[7] as a valid flag can be disabled through a software (DERATECTL0.derate_mr4_tuf_dis). For example, if the system is hot or cold prior to DERATECTL0.derate_enable=1, the first MR4 value read by derating logic may have MR4[7]=0 as there was a software driven MR4 previously, even though MR4[2:0]!=3'b011 (1 x tREFI case). It is recommended to set DERATECTL0.derate_mr4_tuf_dis=1.

Once DERATECTL0.derate_enable=1 is set, it is required not to perform MRR to MR4 through software (MRCTRL*/MRSTAT).

If the value read from MR4 indicates that derating is required, the effective refresh rate is modified as shown in [Table 9-2](#) and [Table 9-3](#) on page 220. In LPDDR4 and the case of MR4[2:0] = "110" or in LPDDR5 and the case of MR4[4:0] = "01101" or "01111", the effective values of the following timing registers are used:

- DERATEVAL0.derated_t_rcd
- DERATEVAL0.derated_t_ras_min
- DERATEVAL0.derated_t_rp
- DERATEVAL0.derated_t_rrd
- DERATEVAL0.derated_t_rc

In case of MR4[2:0] = "000" or "111" (for LPDDR4) and MR4[4:0] = "00000" or "11111" (for LPDDR5), or invalid value, the DDRCTL can generate maskable interrupt derate_temp_limit_intr and non-maskable interrupt derate_temp_limit_intr_fault. If DERATECTL1.derate_temp_limit_intr_en is set to '0', derate_temp_limit_intr is never asserted, although there is no effect on derate_temp_limit_intr_fault. The status register DERATESTAT.derate_temp_limit_intr stores the pre-masked derate_temp_limit_intr. Both interrupts and the status register are cleared by setting DERATECTL1.derate_temp_limit_intr_clr to '1'. Furthermore, for testing or debug purposes, you can force the derate_temp_limit_intr by setting DERATECTL1.derate_temp_limit_intr_force to '1'.

Table 9-2 Effect of Values of MR4 Read from LPDDR4 SDRAM

MR4[2:0]	Refresh Rate	Derate DRAM Timing Values
001	RFSHSET1TMG0.t_refi_x1_x32 * 4	No derating performed
010	RFSHSET1TMG0.t_refi_x1_x32 * 2	No derating performed
011	RFSHSET1TMG0.t_refi_x1_x32	No derating performed
100	RFSHSET1TMG0.t_refi_x1_x32 / 2	No derating performed
101	RFSHSET1TMG0.t_refi_x1_x32 / 4	No derating performed
110	RFSHSET1TMG0.t_refi_x1_x32 / 4	Derating performed - timing parameters incremented (see above)
111	RFSHSET1TMG0.t_refi_x1_x32 / 4	Derating performed - timing parameters incremented (see above)
Other values	RFSHSET1TMG0.t_refi_x1_x32	No derating performed

Table 9-3 Effect of Values of MR4 Read from LPDDR5 SDRAM

MR4[4:0]	Refresh Rate	Derate DRAM Timing Values
00001	RFSHSET1TMG0.t_refi_x1_x32 * 8	No derating performed

MR4[4:0]	Refresh Rate	Derate DRAM Timing Values
00010	RFSHSET1TMG0.t_refi_x1_x32 * 6	No derating performed
00011	RFSHSET1TMG0.t_refi_x1_x32 * 4	No derating performed
00100	RFSHSET1TMG0.t_refi_x1_x32 * 3.3	No derating performed
00101	RFSHSET1TMG0.t_refi_x1_x32 * 2.5	No derating performed
00110	RFSHSET1TMG0.t_refi_x1_x32 * 2	No derating performed
00111	RFSHSET1TMG0.t_refi_x1_x32 1.7	No derating performed
01000	RFSHSET1TMG0.t_refi_x1_x32 * 1.3	No derating performed
01001	RFSHSET1TMG0.t_refi_x1_x32	No derating performed
01010	RFSHSET1TMG0.t_refi_x1_x32 * 0.7	No derating performed
01011	RFSHSET1TMG0.t_refi_x1_x32 * 0.5	No derating performed
01100	RFSHSET1TMG0.t_refi_x1_x32 * 0.25	No derating performed
01101	RFSHSET1TMG0.t_refi_x1_x32 * 0.25	Derating performed - timing parameters incremented (see above)
01110	RFSHSET1TMG0.t_refi_x1_x32 * 0.125	No derating performed
01111	RFSHSET1TMG0.t_refi_x1_x32 * 0.125	Derating performed - timing parameters incremented (see above)

If derating is enabled, and is later disabled by the DERATECTL0.derate_enable register, the refresh rate and other timing parameters revert to their nominal values - the derated values are not retained.

During frequency change process, to retain the derated values, while stopping to send MRR commands for MR4, DERATEEN.derate_mr4_pause_fc must be set to '1' instead of setting DERATEEN.derate_enable to '0'.

Automatic temperature derating shall be disabled/paused before enabling OCPAR poisoning.

If per-bank refresh (RFSHMOD0.per_bank_refresh) and derating (DERATEEN.derate_enable) are enabled, RFSHMOD0.per_bank_refresh_opt_en should be set to '1'.

For LPDDR5, RFSHMOD0.auto_refab_en should be greater than '0' if derating (DERATEEN.derate_enable) is enabled, because JEDEC specification does not expect REFpb commands issued with refresh multiplier 0.125x.

9.1.5 Signals Related to Refresh Controls

The following signals are related to the Refresh Controls:

- Per-bank refresh Bank number Signals

For more information about these signals, refer to the "Signal Descriptions" chapter.

9.1.6 Registers Related to Refresh Controls

The following are the registers related to the Refresh Controls:

- RFSHMOD0
- RFSHCTL0
- RFSHSET1TMG0
- RFSHSET2TMG0
- RFSHSET3TMG0
- RFSHSET4TMG0
- RFSHSET5TMG0

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

9.1.7 Dynamic SDRAM Rank Constraints

One rank constraints block enforces all of the following constraints for each rank supported by the system. Using the rank constraints blocks, the scheduler dynamically obeys all of the constraints on a per-rank basis when scheduling transactions. The values for the timing parameters can be obtained from the relevant JEDEC Specification.

9.2 Refresh Management (RFM)

Periods of high SDRAM activity may require additional Refresh commands to protect the integrity of the SDRAM data. The SDRAM devices that require additional activity-based refreshes include support for an Activation-based refresh management (RFM) command.

The RFM feature is configured by setting configuration parameter `DDRCTL_HW_RFm_CTRL`, and enabled by setting `RFMMOD0.rfm_en` to '1' when SDRAM requires RFM (`MR27:OP[0]==1` in LPDDR5, `MR24:OP[0]==1` in LPDDR4).

9.2.1 Feature Restrictions

- All-bank Refresh Management (RFMab) command is not supported.
- Directed Refresh Management (DRFM) is not supported.

9.2.2 Rolling Accumulated ACT (RAA) Counter

Rolling Accumulated ACT (RAA) counter counts the number of ACT commands, and RFM command is scheduled based on RAA count. The controller has RAA counters per bank-pair. The RAA counter is implemented in the following way:

- Increments by 1 when issuing ACT command to bank corresponding to the RAA counter.
- Decrements by RAAIMT when issuing REFab/REFpb to bank(s) corresponding to the RAA counter.
- Decrements by RAAIMT * RAADEC when issuing RFM command to bank(s) corresponding to the RAA counter.

Valid range of the RAA count is from 0 to RAAMMT (= RAAIMT * RAAMULT). When RAA count is 0, and any decrement event happens, the count keeps 0. When RAA count reaches RAAMMT, the DDRCTL stops scheduling ACT to that bank, and increment event never happens.

Table 9-4 RAA Counter Relevant Registers are Programmed Based on Mode Registers in the SDRAM

	DDRCTL Register	LPDDR4 Mode Register	LPDDR5 Mode Register
RAAIMT	<code>RFMMOD0.raaimt</code>	<code>MR24:OP[5:1]</code>	<code>MR27:OP[5:1]</code>
RAADEC	<code>RFMMOD0.raadec</code>	<code>MR36:OP[1:0]</code>	<code>MR57:OP[1:0]</code>
RAAMULT	<code>RFMMOD0.raamult</code>	<code>MR24:OP[7:6]</code>	<code>MR27:OP[7:6]</code>

9.2.3 RFM Command Control

DDRCTL starts issuing per-bank RFM (RFMpb) command when RAA count reaches RAAMMT. This is to minimize an impact on DDR utilization if a refresh interval (tREFIe) satisfies a Refresh Management Threshold (RFMTH) requirement.

JEDEC specification requirement:

- RAA count <= RAAMMT
- tREFIe > RFMTH

If RAA count == RAAMMT, ACT is not allowed, that is, transaction is blocked. It is possible to issue RFM beforehand.

However, the DDRCTL does not schedule RFM command when current Refresh Multiplier (RM) is greater than or equal to RFMMOD0.rfmth_rm_thr.

DDRCTL specification requirement:

- RAA count == RAAMMT
- RM < RFMMOD0.rfmth_rm_thr

The controller supports rfmth_rm_thr as indicated in [Table 9-5](#) (LPDDR5) and [Table 9-6](#) (LPDDR4). "Yes" in the table means that the controller supports enabling RFM command, and value of rfmth_rm_thr is described in the same row.

Table 9-5 rfmth_rm_thr support for LPDDR5

RAAIMT	RFMTH	tREFle													rfmth_rm_t hr
		8x	6x	4x	3.3x	2.5x	2x	1.7x	1.3x	1x	0.7x	0.5x	0.25x	0.125x	
8	480	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
16	960	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
24	1440	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
32	1920	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
40	2400	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
48	2880	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
56	3360	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
64	3840	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
72	4320	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	5'b11111
80	4800	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
88	5280	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
96	5760	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
104	6240	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
112	6720	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
120	7200	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
128	7680	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
136	8160	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
144	8640	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	5'b01111
152	9120	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
160	9600	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
168	10080	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100

RAAIMT	RFMTH	tREFle													rfmth_rm_t hr
		8x	6x	4x	3.3x	2.5x	2x	1.7x	1.3x	1x	0.7x	0.5x	0.25x	0.125x	
176	10560	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
184	11040	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
192	11520	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
200	12000	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
208	12480	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
216	12960	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
224	13440	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
232	13920	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
240	14400	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100
248	14880	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	5'b01100

Table 9-6 rfmth_rm_thr support for LPDDR4

RAAIMT	RFMTH	Refresh rate					rfmth_rm_thr
		4x	2x	1x	0.5x	0.25x	
8	480	Yes	Yes	Yes	Yes	Yes	5'b00111
16	960	Yes	Yes	Yes	Yes	Yes	5'b00111
24	1440	Yes	Yes	Yes	Yes	Yes	5'b00111
32	1920	Yes	Yes	Yes	Yes	Yes	5'b00111
40	2400	Yes	Yes	Yes	Yes	Yes	5'b00111
48	2880	Yes	Yes	Yes	Yes	Yes	5'b00111
56	3360	Yes	Yes	Yes	Yes	Yes	5'b00111
64	3840	Yes	Yes	Yes	Yes	Yes	5'b00111
72	4320	Yes	Yes	Yes	Yes	Yes	5'b00111
80	4800	Yes	Yes	Yes	Yes	Yes	5'b00111
88	5280	Yes	Yes	Yes	Yes	Yes	5'b00111
96	5760	Yes	Yes	Yes	Yes	Yes	5'b00111
104	6240	Yes	Yes	Yes	Yes	Yes	5'b00111
112	6720	Yes	Yes	Yes	Yes	Yes	5'b00111
120	7200	Yes	Yes	Yes	Yes	Yes	5'b00111

RAAIMT	RFMTH	Refresh rate					rfmth_rm_thr
		4x	2x	1x	0.5x	0.25x	
128	7680	Yes	Yes	Yes	Yes	Yes	5'b00111
136	8160	Yes	Yes	Yes	Yes	Yes	5'b00111
144	8640	Yes	Yes	Yes	Yes	Yes	5'b00111
152	9120	Yes	Yes	Yes	Yes	No	5'b00101
160	9600	Yes	Yes	Yes	Yes	No	5'b00101
168	10080	Yes	Yes	Yes	Yes	No	5'b00101
176	10560	Yes	Yes	Yes	Yes	No	5'b00101
184	11040	Yes	Yes	Yes	Yes	No	5'b00101
192	11520	Yes	Yes	Yes	Yes	No	5'b00101
200	12000	Yes	Yes	Yes	Yes	No	5'b00101
208	12480	Yes	Yes	Yes	Yes	No	5'b00101
216	12960	Yes	Yes	Yes	Yes	No	5'b00101
224	13440	Yes	Yes	Yes	Yes	No	5'b00101
232	13920	Yes	Yes	Yes	Yes	No	5'b00101
240	14400	Yes	Yes	Yes	Yes	No	5'b00101
248	14880	Yes	Yes	Yes	Yes	No	5'b00101

The controller does not issue RFM when RAA count < RAAMMT.

Both JEDEC and DDRCTL specifications must be satisfied.



Note The controller does not enable RFM command, and allows ACT command even when RAA count == RAAMMT, if refresh rate exceeds high temperature limit (5'b11111 in LPDDR5, 3'b111 in LPDDR4). To enable RFM command if high temperature limit is exceeded, set DERATECTL0.dis_trefi_x0125 to '1' for LPDDR5 or set RFMMOD0.rfmth_rm_thr to 5'b11111 for LPDDR4.

9.2.4 Single-bank Mode (LPDDR5 only)

The DDRCTL supports single-bank based RAA counter in DDRCTL_LPDDR_RFMSBC==1 configuration. This feature is enabled by setting RFMMOD0.rfmsbc to '1' and setting MR5[7:OP[5:4]] (RFMSBC, RFM Single-Bank Counters Implemented) to 2'b01 (one RAA counter per one bank) when LPDDR5 supports single-bank mode (MR5[7:OP[3:2]] == 2'b01).

9.2.5 Adaptive Refresh Management (ARFM) (LPDDR5 only)

The DDRCTL can use Adaptive Refresh Management (ARFM) in RFM configuration, DDRCTL_HW_RFMI_CTRL==1, if LPDDR5 SDRAM supports ARFM (MR1:OP[1]==1). To enable the ARFM feature, change RFM level in the SDRAM (MR57:OP[7:6]) and program RFM threshold registers in the DDRCTL (RFMMOD0.raaimt, RFMMOD0.raamult, RFMMOD0.raadec, and RFMMOD0.rfmth_rm_thr) for selected RFM level. If the SDRAM supports ARFM (MR1:OP[1]==1), RFMMOD0.rfm_en needs to be set to '1' for ARFM even when the SDRAM does not support RFM (MR27:OP[0]==0).

The RFM level can be set in the initialization sequence, and changed at any time. Changing the RTL level sequence in normal operation is described in the "Software Sequences" section of the "Programming" chapter.

9.3 ZQ Calibration

This topic contains the following sections:

- “[Overview of ZQ Calibration](#)” on page [228](#)
- “[LPDDR4 Devices](#)” on page [228](#)
- “[LPDDR5 Devices](#)” on page [230](#)
- “[Automatic and Software Initiated ZQ Calibration Commands](#)” on page [232](#)
- “[LPDDR4/LPDDR5 ZQ Reset Command](#)” on page [234](#)
- “[Registers Related to ZQ Calibration](#)” on page [234](#)

9.3.1 Overview of ZQ Calibration

The DDRCTL controller uses ZQ calibration command to calibrate SDRAM RON (Resistor ON) and ODT (On-die termination) values over PVT (Process, Voltage, Temperature). LPDDR4 SDRAMs need more time to calibrate RON and ODT at initialization and relatively less time to perform periodic calibrations. For more information, refer to the LPDDR4 (JEDEC JESD209-4B) specification.

9.3.2 LPDDR4 Devices

The DDRCTL supports all the ZQ calibration commands that are supported by the JEDEC Specification.

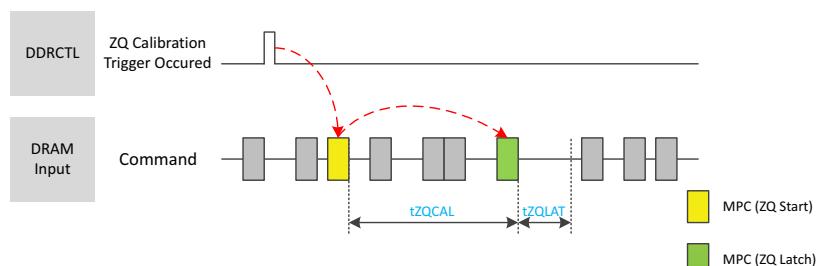
ZQ calibration sequence uses the following MPC commands:

- ZQCAL Start
- ZQCAL Latch

The ZQCAL Start command is used to initiate SDRAM’s calibration procedure.

The ZQCAL Latch command is used to capture the result and load it into SDRAM’s drivers. ZQCAL Latch command can be issued after tZQCAL (determined by `zqset1tmgo.t_zq_long_nop`) has expired and all DQ Bus operations have completed. If a ZQ calibration is triggered, those commands are sent out consecutively to the DRAM by DDRCTL.

Figure 9-5 ZQ Calibration



ZQCAL Start and ZQCAL Latch command can be performed automatically at a regular interval or through direct software request. For more information, see “[Automatic and Software Initiated ZQ Calibration Commands](#)” on page [232](#). Furthermore, both commands are issued automatically after SR-Powerdown exit if `ZQCTL2.dis_srx_zqc1` is set to ‘0’. To disable issuing of both commands after SR-Powerdown exit, set `ZQCTL2.dis_srx_zqc1` to ‘1’.

Do not change the following mode register fields following a ZQCal Start command and before tZQCAL has expired:

- PU-Cal (Pull-up Calibration VOH Point) - MR3 : OP [0]
- PDDS (Pull Down Drive Strength and Rx Termination) - MR3 : OP [5 : 3]
- DQ-ODT (DQ ODT Value) - MR11 : OP [2 : 0]

To change these mode register fields, follow this procedure:

1. Disable the following automatic controls (if using):
 - a. Set PWRCTL.selfref_en to '0'.
 - b. Set ZQCTL0.dis_auto_zq to '1'.
2. Ensure that DDRC is not in self-refresh or Self-Refresh-Powerdown:
 - a. Wait for STAT.operating_mode not to be 3'b011 (self-refresh /Self-Refresh-Powerdown).
3. Enter self-refresh 1 state:
 - a. Set PWRCTL.stay_in_selfref to '1'.
 - b. Set PWRCTL.selfref_sw to '1'.
 - c. Wait until STAT.selfref_state is higher than 3'b000 (any self-refresh state).



- When polling the field STAT.selfref_state, do an APB RD for the entire STAT register. That is, selfref_state and selfref_type must be read at the same time (in the same APB read).
- The reading of the field STAT.selfref_type is done along with the reading of the field STAT.selfref_state.

- d. If STAT.selfref_state is 3'b001 and STAT.selfref_type is not 2'b01 ("self-refresh 1" state caused solely by software, not by DFI PHY Master hardware state machine) then jump to step 4, else: program PWRCTL.selfref_sw=0, program PWRCTL.stay_in_selfref=0 and return to step 2.
4. Send MRW commands using MRCTRL0 and MRCTRL1.
5. Enter Self-Refresh-Powerdown mode:
 - a. Set PWRCTL.stay_in_selfref to '0'.
 - b. Wait for STAT.selfref_state to become 3'b010 (Self-Refresh-Powerdown).
 - c. If DQ-ODT is switched between Disable and Enable, updating DRAMSET1TMG2.rd2wr and RANKTMG0.diff_rank_wr_gap is required accordingly.

If ZQCTL2.dis_srx_zqc1 is programmed to '1':

1. Enter self-refresh 2 state:
 - a. Set PWRCTL.stay_in_selfref to '1'.
 - b. Set PWRCTL.selfref_sw to '0'.
 - c. Wait for STAT.selfref_state to become 3'b011 (self-refresh 2).
2. Send the ZQCal command through direct request from hardware (external IOs) or software.

3. Exit self-refresh mode:
 - a. Set PWRCTL.stay_in_selfref to '0'.
 - b. Wait for STAT.selfref_state to become 3'b000 (not self-refresh).
4. Re-enable automatic controls disabled in first step.

If ZQCTL2.dis_srx_zqc1 is not programmed to '1':

1. Exit self-refresh mode:
 - a. Set PWRCTL.selfref_sw to '0'.
 - b. Wait for STAT.selfref_state to become 3'b000 (not self-refresh).
2. Re-enable automatic controls disabled in the first step.

ZQCal Reset (ZQ Calibration Reset) command is used to reset the output impedance calibration to a default accuracy of +/- 30% across process, voltage, and temperature. This command is used to ensure output impedance accuracy to +/- 30% when ZQCal Start and ZQCal Latch commands are not used and a time period of tZQRESET is allowed, determined by ZQTMG1.t_zq_reset_nop. The command is issued by using the registers ZQTMG1.zq_reset and ZQSTAT.zq_reset_busy. For more information, see ["LPDDR4/LPDDR5 ZQ Reset Command" on page 234](#).

The DDRCTL performs no other activities for the duration of tZQLAT (determined by ZQTMG0.t_zq_short_nop) and tZQRESET (determined by ZQTMG1.t_zq_reset_nop). The quiet time on the SDRAM channel helps in accurate calibration of SDRAM output impedance.

Although not required by the LPDDR4 JEDEC specification, the DDRCTL closes (precharges) all banks before sending ZQCal commands. This implementation is aligned with the ZQ calibration behavior for other protocols.



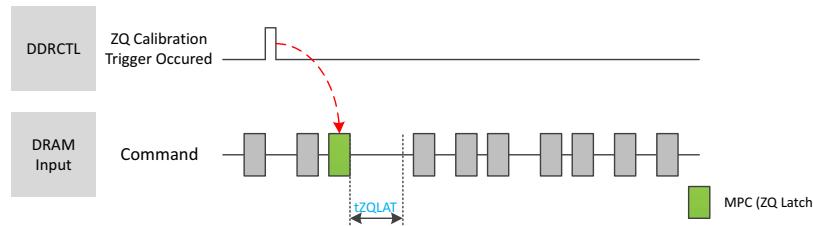
For LPDDR4, the access to the MR10 register from software using the mode register interface is prohibited (see ["Mode Register Reads and Writes" on page 252](#)).

9.3.3 LPDDR5 Devices

LPDDR5 JEDEC specification supports two ZQ calibration mode - Background Calibration or Command-Base Calibration. The DDRCTL supports Background Calibration mode only. Command-Base Calibration is not supported. DDRCTL ZQ calibration sequence uses ZQCal Latch command only.

In Background Calibration mode ZQ calibration is performed in the background and kept up-to-date by the DRAM. Re-calibration will be performed by the LPDDR5 SDRAM within the time interval, tZQINT, specified in MR28.OP[3:2]. The MR28.OP[3:2] can be accessed from software using the mode register interface.

At the completion of ZQ calibration, ZQUF (MR4.OP[5]) bit will be set if the new calibration codes do not match the currently latched codes. It is allowed to issue ZQCal Latch command without monitoring ZQUF bit.

Figure 9-6 LPDDR5 ZQ Calibration

ZQCal Latch command can be performed automatically at a regular interval or through direct software request without monitoring ZQUF bit. For more information, see “[Automatic and Software Initiated ZQ Calibration Commands](#)” on page [232](#). Furthermore, ZQCal Latch is issued automatically after Self-Refresh power-down exit if ZQCTL2.dis_srx_zql is set to ‘0’. To disable issuing of both commands after Self-Refresh exit, set ZQCTL2.dis_srx_zql to ‘1’.

The Background calibration must be halted by setting ZQ Stop (MR28 OP[1]) when DVFSQ is active or VDDQ is going to be power off. Automatically ZQCal latch must not be issued while the Background calibration is halted. Therefore, ZQCTL0.dis_auto_zq and ZQCTL2.dis_srx_zql need to be set to ‘1’ before ZQ Stop is set to ‘1’. ZQ Stop can be accessed from software using the mode register interface.

To change ZQ Stop (MR28 OP[1]) to ‘1’, follow this procedure:

1. Enter self-refresh state:
 - a. Set PWRCTL.selfref_en to ‘0’.
 - b. Check STAT.operating_mode!=3'b011 (or STAT.selfref_state==3'b000). If STAT.operating_mode==3'b011 and STAT.selfref_type==2'b10, set HWLPCTL.hw_lp_en to ‘0’ in order to exit self-refresh.
 - c. Set PWRCTL.stay_in_selfref to ‘1’.
 - d. Set PWRCTL.selfref_sw to ‘1’.
 - e. Wait until STAT.selfref_state is 3'b001 (self-refresh 1 state).
2. Disable the following automatic controls (if using):
 - a. Set SWCTL.sw_done to ‘0’.
 - b. Set ZQCTL0.dis_auto_zq to ‘1’.
 - c. Set ZQCTL2.dis_srx_zql to ‘1’.
 - d. Set SWCTL.sw_done to ‘1’.
 - e. Wait until SWSTAT.sw_done_ack becomes equal to 1'b1.
3. Change ZQ Stop to ‘1’:
 - a. Send MRW commands using MRCTRL0 and MRCTRL1.
4. Exit self-refresh mode:
 - a. Set PWRCTL.selfref_sw to ‘0’.
 - b. Set PWRCTL.stay_in_selfref to ‘0’.
 - c. Wait for STAT.selfref_state to become 3'b000 (not self-refresh).

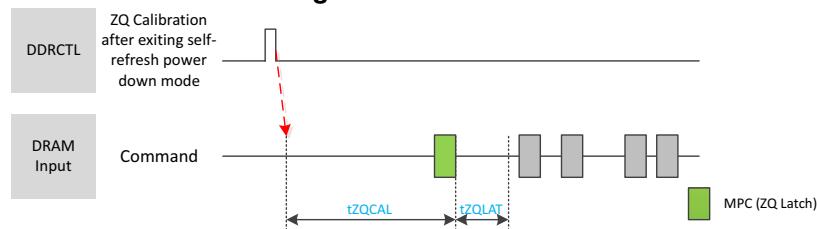
ZQ Stop (MR28 OP[1]) may be reset to ‘0’ when DVFSQ is no longer active or power-down mode is exited. When ZQ Stop is reset to ‘0’, ZQ calibration starts immediately. If ZQCTL0.dis_auto_zq changes from ‘1’

to '0', Read/Write commands are not issued while ZQ calibration is working (the period of tZQCAL + tZQLAT).

To change ZQ Stop (MR28 OP[1]) to '0', follow this procedure:

1. Enter self-refresh state:
 - a. Set PWRCTL.selfref_en to '0'.
 - b. Check STAT.operating_mode!=3'b011 (or STAT.selfref_state==3'b000). If STAT.operating_mode==3'b011 and STAT.selfref_type==2'b10, set HWLPCTL.hw_lp_en to '0' in order to exit from self-refresh.
 - c. Set PWRCTL.stay_in_selfref to '1'.
 - d. Set PWRCTL.selfref_sw to '1'.
 - e. Wait until STAT.selfref_state is higher than 3'b000 (any self-refresh 1 state).
2. Reset ZQ Stop to '0':
 - a. Send MRW commands using MRCTRL0 and MRCTRL1.
3. Enable the following automatic controls (if using):
 - a. Set SWCTL.sw_done to '0'.
 - b. Set ZQCTL0.dis_auto_zq to '0'.
 - c. Set ZQCTL2.dis_srx_zqcl to '0'.
 - d. Set SWCTL.sw_done to '1'.
 - e. Wait until SWSTAT.sw_done_ack becomes equal to 1'b1.
4. Exit self-refresh mode:
 - a. Set PWRCTL.selfref_sw to '0'.
 - b. Set PWRCTL.stay_in_selfref to '0'.
 - c. Wait for STAT.selfref_state to become 3'b000 (not self-refresh).

Figure 9-7 LPDDR5 ZQ Calibration After Exiting Self-Refresh Power-Down Mode



If an LPDDR5 SDRAM package has multiple dies and two or more ZQ pins, for proper operation an external ZQ resistor cannot be shared by those ZQ pins. That is, each ZQ pin needs dedicated external ZQ resistor.

9.3.4 Automatic and Software Initiated ZQ Calibration Commands

The DDRCTL issues ZQ Calibration commands in the following ways:

- **Automatic ZQ Calibration Commands by the DDRCTL**

DDRCTL sends ZQ Calibration commands to SDRAM periodically. The interval is determined by `ZQSET1TMG1.t_zq_short_interval_x1024`.

This method is used if `ZQCTL0.dis_auto_zq` is set to '0'.

- **ZQ Calibration Commands using direct software request:**

The SoC sends ZQ Calibration commands through software by setting `OPCTRLCMD.zq_calib_short` to '1'. When the ZQ Calibration command request is stored in the DDRCTL, the register bit is automatically cleared.

It is recommended not to set `OPCTRLCMD.zq_calib_short` signal in initialization, or Self-Refresh-Powerdown, or Deep Sleep Mode (LPDDR5) operating modes (this can be checked by observing `STAT.operating_mode` and `STAT.selfref_state`).

The SoC can initiate a ZQ Calibration command operation only if `OPCTRLSTAT.zq_calib_short_busy` is low. The `OPCTRLSTAT.zq_calib_short_busy` signal goes high in the next clock cycle after the DDRCTL accepts a ZQ Calibration commands request. It goes low when the ZQ Calibration command operation is initiated in the DDRCTL. For proper SDRAM operation, you or SoC must schedule this command frequently.

This method is used if `ZQCTL0.dis_auto_zq` is set to '1'.



Note If software sets `ZQCTL0.dis_auto_zq` to '1', and then sets `OPCTRLCMD.zq_calib_short` to '1' soon after the DDRCTL schedules periodic MPC commands (ZQCAL Start/Latch) for each rank, the following eight MPC command requests (four pairs of ZQCAL start/latch) are queued in DDRCTL:

- Periodic ZQCAL Start for rank0
- Periodic ZQCAL Latch for rank0
- Periodic ZQCAL Start for rank1
- Periodic ZQCAL Latch for rank1
- Software initiated ZQCAL Start for rank0
- Software initiated ZQCAL Latch for rank0
- Software initiated ZQCAL Start for rank1
- Software initiated ZQCAL Latch for rank1

In this case, even if the PHY sets `dfi0_phymstr_req='1'`, the DDRCTL cannot respond by setting `dfi0_phymstr_ack='1'` in a timely manner.

This could happen when all the following conditions are true:

- `MSTR0.lpddr4` is '1' (LPDDR4 SDRAM is used)
- `ZQCTL0.dis_auto_zq` is '0' until software sets `OPCTRLCMD.zq_calib_short='1'`
- `ZQCTL0.zq_resistor_shared` is '1'
- `MSTR0.active_ranks` is '3' or '15' (2 or 4 ranks)

- **External ZQCAL**

In this case ZQCAL commands are sent through top-level IOs. This functionality is enabled by setting the configuration parameter `UMCTL2_REF_ZQ_IO` to '1' and is switched On by setting the static register `OPCTRLCMD.hw_ref_zq_en` to '1' (see "Register Descriptions" chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual) and has effect only if `ZQCTL0.dis_auto_zq` is set to '1'. If these conditions are not satisfied, the IO pins related to external ZQ commands are ignored. External ZQCAL can be issued by setting `ext_zq_calib_short` signal to

'1' for one clock cycle. It is recommended not to set this signal in initialization, self-refresh, deep power-down operating modes.

`ext_zq_calib_short_busy` signal goes high in the next clock cycle after the DDRCTL accepts a ZQCal request through IOs and it goes low when the ZQCal operation is initiated in the DDRCTL.



Note External ZQCal feature is currently not supported.

For the most current information about unsupported features and limitations, refer to the Release Notes.

In Self-Refresh-Powerdown modes, the command is scheduled after exiting from Self-Refresh-Powerdown.

9.3.5 LPDDR4/LPDDR5 ZQ Reset Command

The ZQ Reset command is issued by setting `ZQCTL1.zq_reset` to '1'. It is recommended not to set this register field to '1' in init or Self-Refresh-Powerdown or Deep Sleep Mode (LPDDR5) operating modes (this can be checked by observing `STAT.operating_mode` and `STAT.selfref_state`). The SoC can initiate a ZQ Reset operation only if `ZQSTAT.zq_reset_busy` is low. This register field goes high in the clock after the DDRCTL accepts a ZQ Reset request. It goes low when the ZQ reset command is issued to the SDRAM and the associated NOP period is completed.

For Self-Refresh-Powerdown, command is scheduled after Self-Refresh-Powerdown is exited.



Note When DDRCTL is going to put SDRAM into Self-Refresh-Powerdown and `ZQCTL1.zq_reset` is about to set to '1', it is possible that the request can be accepted. However, DDRCTL cannot issue ZQ Reset command if SDRAM is already in Self-Refresh-Powerdown.

In this case, ZQCal Start/Latch commands are issued instead of ZQ Reset command just after Self-Refresh-Powerdown is exited.

9.3.6 Registers Related to ZQ Calibration

The following are the registers related to the ZQ Calibration:

- `ZQCTL0`
- `ZQCTL2`
- `ZQTMG1`
- `ZQSTAT`

For more information about these registers, refer to the "Register Descriptions" chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

9.4 Controller Assisted Drift Tracking by MRR Snooping (LPDDR5)

This topic contains the following sections:

- “[Overview of Controller Assisted Drift Tracking](#)” on page [235](#)
- “[PHY MRR Snoop Control](#)” on page [235](#)
- “[Registers Related to Controller Assisted Drift Tracking](#)” on page [238](#)

9.4.1 Overview of Controller Assisted Drift Tracking

This feature is applicable to LPDDR5 SDRAM. As voltage and temperature change on the SDRAM die, the DQS/WCK clock tree will shift and may require re-training. The LPDDR5 includes an internal oscillator to measure the amount of delay over a given time interval. These oscillators started by MPC command with proper opcode. The results are updated in the MR register which can be read using MRR command. This feature enables Synopsys LPDDR5X/5/4X PHY to smartly adjust LCDL (“local calibrated delay line”) to compensate the drift when DFI update event occurs. It is accomplished by allowing Synopsys LPDDR5X/5/4X PHY to periodically snoop the result of these oscillators of SDRAM.

This feature is also referred as “DQS Oscillator” in the controller and it can be used interchangeably with Controller Assisted Drift Tracking.



- DDRCTL does not support MPC command to stop the Oscillator.
- See Synopsys LPDDR PHY documentation to know if there are limitations.

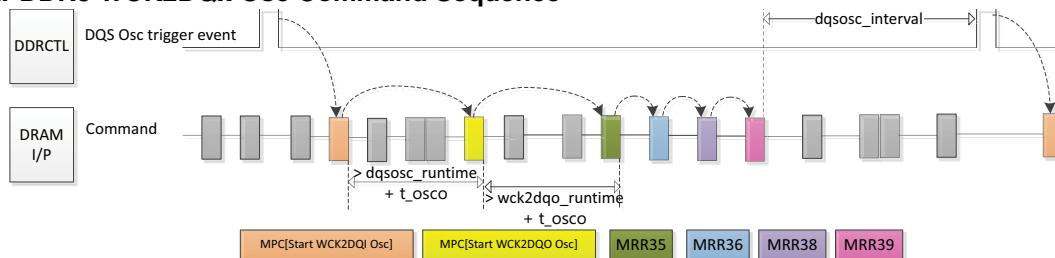
9.4.2 PHY MRR Snoop Control

LPDDR5 SDRAM includes internal oscillators to measure the tWCK2DQI and tWCK2DQO timing parameters.

These oscillators are started by issuing MPC command. After runtime expires, the results are updated in MR register by SDRAM. The DDRCTL takes the control of scheduling MPC command to start these oscillators and MRR commands to read the result of these oscillators periodically as well as during SRE/PDE. The DDRCTL also asserted dwc_ddrphy*_snoop_en_P* flag, along with dfi_rddata_en/dfi_rddata_cs for full burst-length UIs, to enable the Synopsys LPDDR5X/5/4X PHY to capture the MRR data. Subsequently, Synopsys LPDDR5X/5/4X PHY smartly adjusts LCDL (“local calibrated delay line”) to compensate the drift when DFI update event occurs. The DDRCTL also blocks any activate command to the same rank while reading out the result (MRR command) of these oscillators.

For LPDDR5, the DDRCTL first sends out two MPC commands to start the WCK2DQI and WCK2DQO interval oscillators. Subsequently, followed by four MRR commands to get the result of these oscillators.

Figure 9-8 LPDDR5 WCK2DQx Osc Command Sequence



For a multi rank system, the MPC commands to start the oscillators are scheduled to all the active ranks simultaneously. However, the MRR commands to read out the result are scheduled one after the other in the order of higher rank to lower rank.

To enable the DQS Oscillator, follow this procedure below:

1. Program the following register(s) to "interval timer run time setting" defined in the JEDEC specification:
 - `DQSOSCRUNTIME.dqsosc_runtime` for MR3
 - `DQSOSCRUNTIME.wck2dqo_runtime` for MR40
2. Program the `DQSOSCRUNTIME` register. This is non-zero binary value that corresponds to interval timer run time setting of LPDDR5 SDRAM.
3. Program the `DQSOSCCTL0` register:
 - a. Program DQS Oscillator interval time. This defines the frequency of DQS Oscillator trigger event. It must be appropriately programmed based on the system requirement.
 - b. Enable the DQS oscillator by setting `DQSOSCCTL0.dqsosc_enable =1`.

To disable the DQS Oscillator, use the following procedure:

1. Disable DQS oscillator by setting `DQSOSCCTL0.dqsosc_enable=0`.
2. If the DQS oscillator has already started (MPC[Start DQS Osc] is already issued), it will continue until the associated MRR commands are issued. Completely disabled state of the DQS Oscillator is indicated by `DQSOSCSTAT0.dqsosc_per_rank_stat=0` and `DQSOSCSTAT0.dqsosc_state=0`.

After disabling the DQS oscillator and before enabling any other periodic phase training method the software must poll these registers (`DQSOSCSTAT0.dqsosc_per_rank_stat=0` and `DQSOSCSTAT0.dqsosc_state=0`). DQS oscillator must be disabled before enabling OCPAR poisoning.



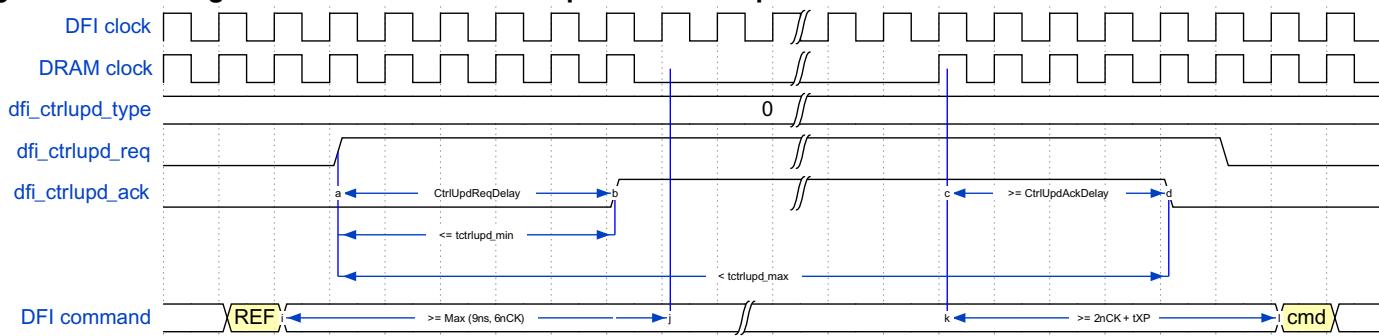
Note Controller assisted drift tracking and PPT (Periodic Phase Training) are mutually exclusive and must not be enabled at the same time. Do not change `DQSOSCCTL0.dqsosc_enable` (that is, keep the default value of '0') if DFI PHY Master interface is used.

9.4.3 Compensate DRAM Clock Stop

Synopsys LPDDR5/4/4X PHY and LPDDR5X/5/4X PHY stops the DRAM clock during MC-initiated DFI update (DFI control update) if MRR snoop is enabled. Follow this section to be compliant with LPDDR5 JEDEC specification that has limitations against stopping the clock.

9.4.3.1 PHY Configuration

LPDDR5 JEDEC specification defines the timing gap between some command and DRAM clock stop and resume. PHYINIT sets its CSR `CtrlUpdReqDelay` and `CtrlUpdAckDelay` to delay `dfi_ctrlupd_req` and `dfi_ctrlupd_ack` handshake and ensures the gap requirement, as shown in [Figure 9-9](#). Contact Synopsys PHY support for more details.

Figure 9-9 Timing for DRAM CK/CA and ctrlupd with CK Stop

9.4.3.2 DFI Control Update with Bank Close

LPDDR5 JEDEC specification declares that all banks need to be closed before stopping DRAM clock so that periodic DFI control update aligns with it. Follow these instructions to use PPT2 override:

1. Set PHY pub register `PPTTrainSetup_pX::PhyMstrTrainInterval` to '0'.
 - a. Controller register `DFIUPHYMSTR.dfi_phymstr_en` can also be '0'.
2. Set PHY pub register `DFIUPHYUPD::DFIUPHYUPDCNT` to '0'.
 - a. Controller register `DFIUPD0.dfi_phyupd_en` can also be '0'.
3. Set `PWRCTL.lpddr4_sr_allowed` to '1'.
4. Set DFI timing parameters `DFIUPDTMG0.dfi_t_ctrlup_min` and `DFIUPDTMG0.dfi_t_ctrlup_max` according to the value determined in “[PHY Configuration](#)” on page 236.
5. Set `DFIUPD0.dis_auto_ctrlupd` to '1' and `OPCTRLCMD.ctrlupd` to '0'.
6. Set `DFIUPDTMG2.ppt2_override` to '1' and `DFIUPDTMG2.ppt2_en` to '1'.
7. Set `DFIUPDTMG2.dfi_t_ctrlupd_interval_type1_unit` to '1'.
8. `DFIUPDTMG2.dfi_t_ctrlupd_interval_type1` should be the same as `DFIUPDTMG1.dfi_t_ctrlupd_interval_min_x1024`. Note that it gives DFI control update interval but can get longer than the configured value as long as REFab/REFpb is postponed. This may require consideration.
9. Set `PPT2CTRL0.ppt2_ctrlupd_num_dfi0` to '1' and `PPT2CTRL0.ppt2_ctrlupd_num_dfi1` to '0' regardless of the number of DFI interfaces or hardware configuration parameter `MEMC_DRAM_DATA_WIDTH`.
10. Set `PPT2CTRL0.ppt2_burst` to '0'.
11. Set `PPT2CTRL0.ppt2_burst_num` to '0'.
12. All data rates are supported.
13. If HWFFC is enabled, only `HWFFCCTL.hwfffc_mode=1` is supported.
14. DFI control update will be sent only when DRAM state is in Idle, Self-refresh without power-down, or Refresh.
15. DDRCTL will close all opened banks before issuing DFI control update.

Compared to the pure DFI control update, PPT2 override may affect performance due to the following facts coming from the PPT2 mechanism:

- Closes all banks before issuing `dfi_ctrlupd_req`
- When `dfi_ctrlupd_req` needs to be issued while in SRPD, DDRCTL exits from SRPD to be in SR w/o PD state
- When `dfi_ctrlupd_req` needs to be issued while in PD, DDRCTL exits from PD to be in Normal state

To temporarily disable the periodic DFI control update, set `DFIUPDTMG2.ppt2_en` to '0'. `DFIUPDTMG2.ppt2_override` is a static register and cannot be changed from '1'.



Note PPT2 is another existing feature that issues DFI control update with `dfi_ctrlupd_type=1`, while, in contrast, `dfi_ctrlupd_type` is '0' in regular DFI control update. As PPT2 supports bank close, you can use it to send DFI control update with bank close.
`DFIUPDTMG2.ppt2_override==1` configuration will override `dfi_ctrlupd_type` to be '0'. Additionally, DDRCTL does not issue any DFI control update request while DQS Oscillator in SDRAM device is running if `DFIUPDTMG2.ppt2_override=1`. If `DFIUPDTMG2.ctrlupd_after_dqsosc` is set to '1', Single DFI control update request is issued for each MRR Snoop complete.

9.4.3.3 Burst DFI Control Update

Regardless of “DFI Control Update with Bank Close” on page 237, burst DFI control update does not automatically close banks. In this case, DDRCTL will assert `dwc_*_snoop_osc_running`. It is a level signal and PHY does not stop DRAM clock if it is '1' even during DFI control update. The signal is a non DFI-standard signal and only applicable to proprietary Synopsys LPDDR 5/4/4X PHY and 5X/5/4X PHY.

9.4.4 Registers Related to Controller Assisted Drift Tracking

The following are the registers related to Controller Assisted Drift Tracking

- `DQSOSCRUNTIME`
- `DQSOSCCTL0`
- `DQSOSCSTAT0`

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

9.5 Enhanced Incremental Periodic Phase Training (PPT2)



This feature is currently not supported with Synopsys LPDDR5X/5/4X PHY. The information in this section is provided for reference only.

This section covers the following topics:

- “[Overview of PPT2](#)” on page [239](#)
- “[PPT2 Limitations](#)” on page [240](#)
- “[Retraining Interval](#)” on page [241](#)
- “[Normal PPT2](#)” on page [242](#)
- “[Burst PPT2](#)” on page [246](#)
- “[Registers Related to PPT2](#)” on page [248](#)

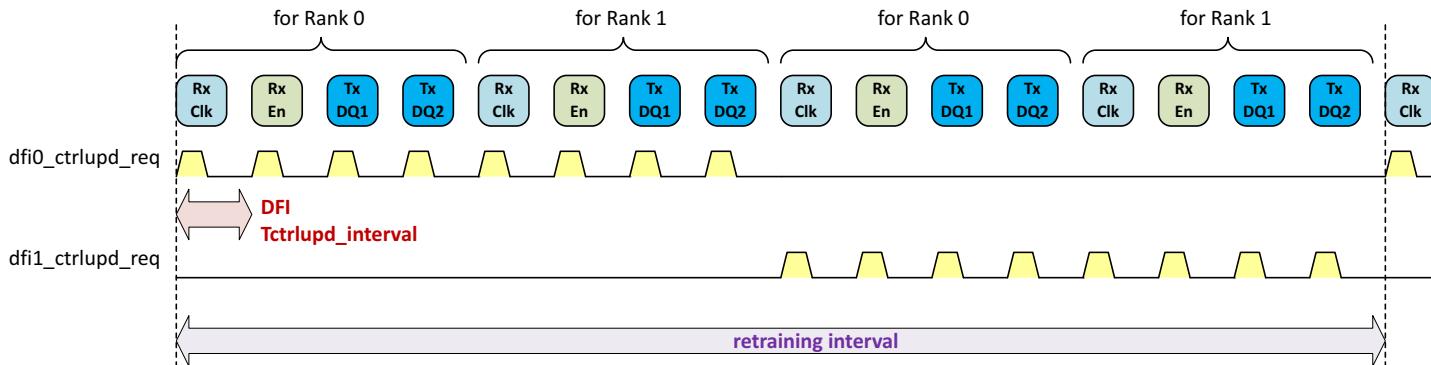
9.5.1 Overview of PPT2

PPT2 is a MC initiated, low latency and accurate DQS/DQ retraining scheme done along with Synopsys LPDDR5/4/4X PHY. DDRCTL issues a number of `dfin_ctrlupd_req/ack` handshake with `dfin_ctrlupd_type=1`¹ to trigger the PHY PPT2 within a given interval. For each handshake, PHY cyclically retrains its four interfaces of RxClk, RxEn, TxDqWr, and TxDqWrECC (if Link ECC is enabled). Since they need retraining per a rank per a DFI interface, a hardware configuration which is Link ECC enabled & 2-rank & dual DFI needs $4 \times 2 \times 2$ `ctrlupd_req` to complete retraining all devices. [Figure 9-10](#) shows an example of PPT2 scheduling.

Because each `ctrlupd_req` costs less than 500ns (or 1000ns depending on the configuration) and some part of it is during tRFC, PPT2 has lower latency comparing to PHY master of 10us latency. Also, PPT2 is considered to be accurate because PHY does not rely on any built-in oscillators in DRAM to retrain.

Unless otherwise specified, `ctrlupd_req` in this section refers to the `dfin_ctrlupd_req/ack` handshake with `dfin_ctrlupd_type=1`.

Figure 9-10 PPT2 Scheduling - Sending 16 `ctrlupd_req` to Retrain 2-rank & Dual DFI System



1. `dfin_ctrlupd_type` is a DFI interface signal, which is Synopsys only and a non-standard one for now.

9.5.2 PPT2 Limitations

The following are current PHY side limitations. For the latest status of these limitations, refer to the PHY databook.

- Supported for data rates ≥ 1600 Mbps.
- Both Write Link ECC and Read Link ECC are enabled or both Write Link ECC and Read Link ECC are disabled.
- Retraining latency is less than:
 - 500ns if data rates ≥ 3200 Mbps and WL=setA
 - 1000ns if data rates ≥ 1600 Mbps and WL=setA
- PHY configuration userInputBasic.RetrainMode[Pstate]=4 must be set.
- PPT2 is only used at the following DRAM states: Idle, Self-refresh (without Power-down), and Refresh.

The following list shows DDRCTL side limitations.

- When PPT2 is used, phystr, phyupd, ctrlmsg, and dqsosc features are not necessary and must be disabled.
 - PHY pub register PPTTrainSetup_px::PhyMstrTrainInterval must be '0'.
 - Controller register DFIPHYMSTR.dfi_phymstr_en can also be 0.
 - PHY pub register DFIPHYUPD::DFIPHYUPDCNT must be '0'.
 - Controller register DFIUPD0.dfi_phyupd_en can also be '0'.
 - DFI0MSGCTL0.dfi0_ctrlmsg_req must be '0'.
 - DQSOSCCTL0.dqsosc_enable must be '0'.
- Existing DFI MC-Initiated Update Requests via ctrlupd_type=0 does not conflict with PPT2 according to [Table 9-7](#).

Table 9-7 dis_auto_ctrlupd and ppt2_en Combination

dis_auto_ctrlupd	ppt2_en	DDRCTL Functionality
0	0	Only ctrlupd type 0 is sent automatically.
0	1	Both ctrlupd type 0 and type 1 are sent automatically.
1	0	Neither ctrlupd type 0 nor type 1 are sent automatically.
1	1	Only ctrlupd type 1 is sent automatically.

- Connect proprietary DDRCTL DFI sideband ports to the PHY:
 - dfi0_ctrlupd_type
 - dfi1_ctrlupd_type if needed
 - They are not DFI 5.1 standard ports.
- Refresh multiplier 0.125x (LPDDR5 only) is not supported.
- Automatic All-bank / Per-bank refresh switching is recommended to be enabled.

- RFSHMOD0.auto_refab_en=2 (switched to REFab if refresh rate <= 0.25x) is recommended.
- Dynamic Frequency Switch is supported with PPT2, but neither DVFSCL nor DVFSQ is supported.
- PWRCTL.1pddr4_sr_allowed must be '1'.
- DDRCTL will close all opened banks before issuing PPT2 to prepare for DRAM clock stop which PHY may do while retraining.
- When postponed refresh count exceeds RFSHMOD0.refresh_burst, 'critical refresh' happens. It blocks both Normal PPT2 and Burst PPT2 issues, not to postpone too many refresh commands while ctrlupd_req is sent.

9.5.3 Retraining Interval

$t_{retraining_interval}$ is an interval to complete a single, periodical all device retraining. It must be calculated to make the retraining pace catch up with the VT drifts. Otherwise, retraining cannot sufficiently compensate the drifts as time passes, which leads DQS/DQ bus error as a result. Refer to the PHY PUB databook for $t_{retraining_interval}$ calculation.

As illustrated in [Figure 9-10](#) on page [239](#), the PHY requires several `ctrlupd_req` sent from DDRCTL for each retraining interval. There is also $t_{ctrlupd_interval}$, which is an interval between two consecutive `ctrlupd_req`.

The number of `ctrlupd_req` required within $t_{retraining_interval}$ is four, if Link ECC is enabled, otherwise, it is three per a rank per a DFI interface. For Link ECC enabled & 2-rank & dual DFI system, $t_{ctrlupd_interval} = t_{retraining_interval}/(4*2*2)^1$.

[Table 9-8](#) shows how many `ctrlupd_req` is required for each retraining interval.

Table 9-8 The Number of `ctrlupd_req` Required for Each $t_{retraining_interval}$

DFI Interface	MSTR0.active_ranks	LNKECCCTL0.wr_link_ecc_enable	Total number of <code>dfi_ctrlupd_req</code> for each $t_{retraining_interval}$	PPT2CTRL0.ppt2_ctrlupd_num_dfi0	PPT2CTRL0.ppt2_ctrlupd_num_dfi1
Single DFI	1	0	3	3	0
Single DFI	1	1	4	4	0
Single DFI	3	0	6	6	0
Single DFI	3	1	8	8	0
Dual DFI	1	0	6	3	3
Dual DFI	1	1	8	4	4
Dual DFI	3	0	12	6	6
Dual DFI	3	1	16	8	8

Set the register `DFIUPDTMG2.dfi_t_ctrlupd_interval_type1` and `DFIUPDTMG2.dfi_t_ctrlupd_interval_type1_unit` to $t_{ctrlupd_interval}$.

Set the register `PPT2CTRL0.ppt2_ctrlupd_num_dfi0` and `PPT2CTRL0.ppt2_ctrlupd_num_dfi1` to the number of required `ctrlupd_req` for dfi0 and dfi1.

1. This configuration is illustrated in [Figure 9-10](#) on page [239](#).

9.5.4 Normal PPT2

Normal PPT2 sends `ctrlupd_req` periodically and cyclically to dfi0 and dfi1. It is intended for the retraining to persistently compensate a little amount of short-term drift. When large amount of drift is expected, for example, after initialization or SRX from long-term SRPD/DSM, Burst PPT2 is required. For more details, see “[Burst PPT2](#)” on page [246](#).

To enable Normal PPT2, follow this procedure:

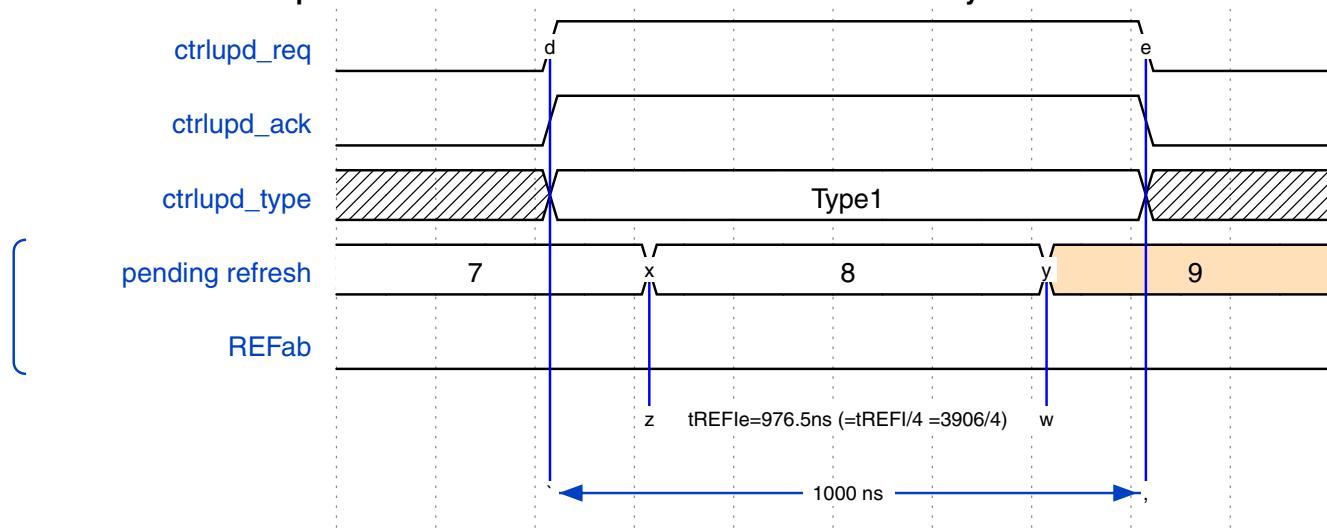
1. Program DFI Update Timing Parameters¹.
 - ❑ Set `DFIUPDTMG0.dfi_t_ctrlup_min`.
 - ❑ Set `DFIUPDTMG0.dfi_t_ctrlup_max`.
2. Program Refresh register.
 - ❑ [Table 9-9](#) shows the maximum `RFSHMOD0.refresh_burst` if `RFSHMOD0.per_bank_refresh` is '0'.

Table 9-9 Upper Limit of All-bank Refresh Burst

Maximum PPT2 Retraining Latency (ns)	Minimum Refresh Rate	tREFI * (Refresh Rate)	# of REFab Which PPT2 Can Postpone	Maximum RFSHMOD0.refresh_burst
500	0.25	977	1	7 (Up to burst-of-8 refresh)
500	0.125	488	2	6 (Up to burst-of-7 refresh)
1000	0.5	1953	1	7 (Up to burst-of-8 refresh)
1000	0.25	977	2	6 (Up to burst-of-7 refresh)
1000	0.125	488	3	5 (Up to burst-of-6 refresh)

- This limitation prevents any bank from violating tREFI.
- [Figure 9-11](#) shows an example of fatal case where burst-of-8 refreshes is allowed in PPT2 retraining latency = 1000ns with 0.25x refresh rate. 1000ns of PPT2 gap can postpone 2 refreshes while 7 refreshes are already postponed so that the number of pending refreshes can be greater than 8. `RFSHMOD0.refresh_burst` must have proper upper bound to avoid this case.

1. Must be programmed while `PPT2CTRL0.ppt2_en` is '0'.

Figure 9-11 PPT2 Can Postpone 2 All-bank Refresh Commands When Its Latency is 1000ns

- Table 9-10 shows the maximum RFSHMOD0.refresh_burst if RFSHMOD0.per_bank_refresh is 1.

Table 9-10 Upper Limit of Per-bank Refresh Burst

Maximum PPT2 Retraining Latency (ns)	Minimum Refresh Rate in REFpb	$t\text{REFI}pb^*$ (Refresh Rate)	# of REFpb Which PPT2 Can Postpone	Maximum RFSHMOD0.refresh_burst
500	1	488	2	54
500	0.7	342	2	54
500	0.5	244	3	53
500	0.25	122	5	51
500	0.125	61	9	47
1000	1	488	3	53
1000	0.7	342	3	53
1000	0.5	244	5	51
1000	0.25	122	9	47
1000	0.125	61	17	39

- This limitation prevents any bank from violating tREFI.
- If RFSHMOD0.auto_refab_en==2, minimum refresh rate in REFpb is 0.5x.
- If RFSHMOD0.auto_refab_en==3, minimum refresh rate in REFpb is 0.25x.

3. Program $t_{ctrlupd_interval}$:

- Set DFIUPDTMG2.dfi_t_ctrlupd_interval_type1 and DFIUPDTMG2.dfi_t_ctrlupd_interval_type1_unit to $t_{ctrlupd_interval}$.

- PPT2 interval can get longer as much as REFab/REFpb is postponed which may require consideration. If `ppt2_wait_ref=1`, Normal PPT2 will wait for any scheduled REFab/REFpb command before sending `ctrlupd_req type1` after the interval timer expiration, which is 35.1 us (= 9 * tREFI) at most if refresh rate is 1x.
 - Set `PPT2CTRL0.ppt2_ctrlupd_num_dfi0`.
 - Set `PPT2CTRL0.ppt2_ctrlupd_num_dfi1`.
4. Start Normal PPT2.
- Set `DFIUPDTMG2.ppt2_en` to '1'¹.
 - PPT2 status can be observed via the register `PPT2STAT0.ppt2_state`.
 - Note: DDRCTL sends single `ctrlupd type1` as soon as possible after the `ppt2_en` is toggled from 0 to 1, except `ppt2_en` is set when DDRCTL is in initialization state.

To disable the Normal PPT2, follow this procedure:

- Set `PWRCTL.selfref_en` to '0', `PWRCTL.selfref_sw` to '0', and `PWRCTL.dsm_en` to '0'.
- Poll `STAT.operating_mode` until it is **not** '3'.
- Poll `PPT2STAT0.ppt2_state` until it is '1'.
- Set `DFIUPDTMG2.ppt2_en` to '0'.
- Poll `PPT2STAT0.ppt2_state` until it is '0'.

Once Normal PPT2 has started, DDRCTL never automatically terminates it until `DFIUPDTMG2.ppt2_en` on current frequency becomes '0'. When DDRCTL has sent `ctrlupd_req` as many as specified at `PPT2CTRL0.ppt2_ctrlupd_num_dfi0/1` to retrain all rank0 and rank1 devices at dfi0 and dfi1, DDRCTL continues sending `ctrlupd_req` from the beginning of rank0 device at dfi0.

Once PPT2 has stopped and resumed, DDRCTL sends single `ctrlupd_req` as soon as possible. Also, DDRCTL does not reset how many `ctrlupd_req` has been sent while it stopped. That is, if the previous `ctrlupd_req` was sent to retrain DFI0 Rank0 TxQD1, it continues sending `ctrlupd_req` from DFI0 Rank0 TxQD2.

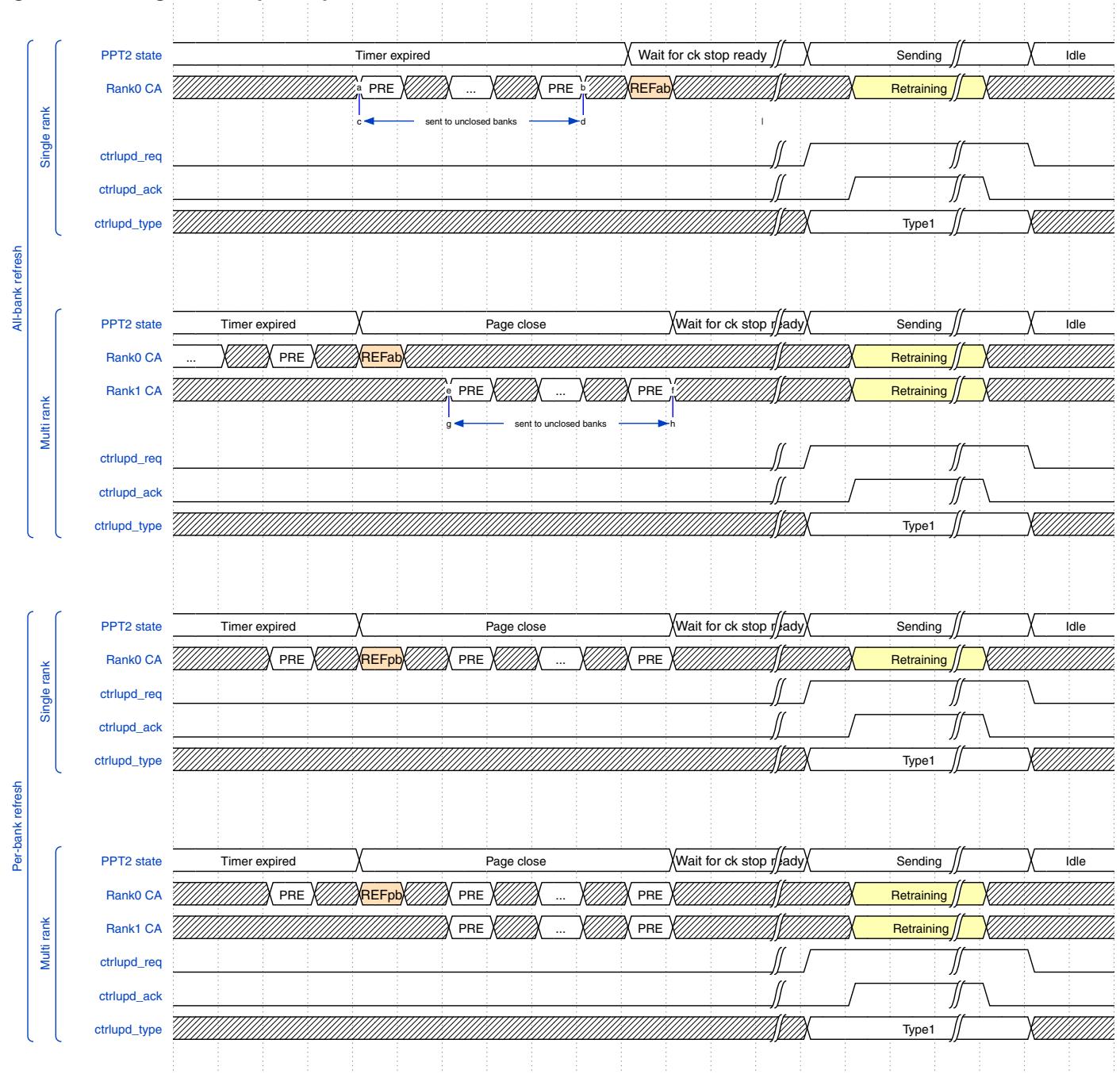
When `dis_auto_ctrlupd` is '0' and `ppt2_en` is '1', DDRCTL issues `ctrlupd_req` with both `dfin_ctrlupd_type=0` and `dfin_ctrlupd_type=1` periodically. Their interval is independent so that they can be issued consecutively.

[Figure 9-12](#) on page 245 shows a typical flow of how DDRCTL sends a `ctrlupd_req` as Normal PPT2 during Normal state. In the figure, PRE always represents PREpb regardless of the REF type (REFab/REFpb) because DDRCTL does not support PREab in general.

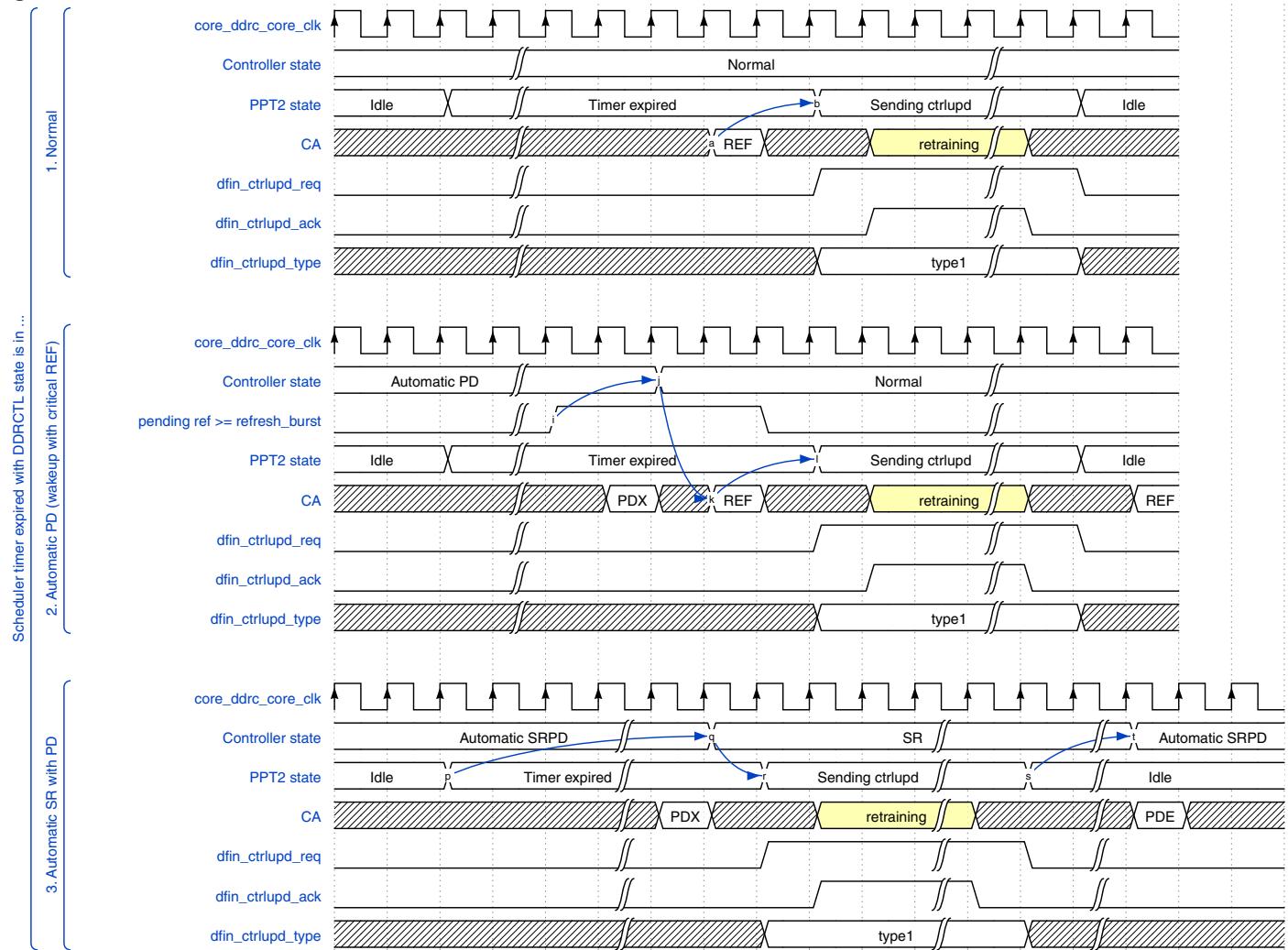
Once DDRCTL PPT2 internal timer that counts $t_{ctrlupd_interval}$ expires,

1. DDRCTL waits for the next (periodic) REFab or REFpb command.
2. Close all opened banks to prepare for clock stop which PHY may do while retraining.
3. Issue `dfin_ctrlupd_req/ack` handshake with `dfin_ctrlupd_type=1`.
4. Reset PPT2 timer to be $t_{ctrlupd_interval}$ and start counting again.

1. `DFIUPDTMG2.ppt2_en` can be set per target frequency.

Figure 9-12 Regular ctrlupd_req Issue

During software or hardware low-power initiated SRPD, DDRCTL does not issue Normal PPT2 because PPT2 is not allowed during SRPD and exiting SRPD is software's responsibility. In contrast, when Normal PPT2 is required ($t_{ctrlupd_interval}$ timer expired) during automatic SRPD, DDRCTL exits SRPD to go to SR without PD state where PPT2 is allowed, sends `ctrlupd_req`, and then re-enters to SRPD. This case is one of the exceptions where `ctrlupd_req` does not follow any REFab/REFpb command due to Normal PPT2. When PPT2 is required during automatic PD, DDRCTL waits for PD exit which occurs eventually and then issues `ctrlupd_req`. Figure 9-13 compares how DDRCTL issues PPT2 when it is triggered under (1) Normal state, (2) Automatic PD state, and (3) Automatic SRPD state.

Figure 9-13 How PPT2 Follows Automatic PD Exit or Automatic SRPD Exit

Note While Normal PPT2 is pending in PD state, it blocks DDRCTL from entering automatic SR even if PWRTMG.selfref_to_x32 timer expires. DDRCTL enters automatic SR once pending PPT2 is issued.

9.5.5 Burst PPT2

Burst PPT2 can compensate a large amount of VT drift by issuing 32¹ set of the entire DFI/rank retraining. DDRCTL sends `ctrlupd_req` up to 32*(4*2*2) times as quickly as possible. Burst PPT2 is useful right after the initialization (after the first `dfi_init_complete=1`) or when returning from a long-term SRPD or DSM, including LP3 I/O retention where the Normal PPT2 is not allowed so it has been a long time since the last PPT2. Figure 9-14 and Figure 9-15 show how the entire `ctrlupd_req` is sent during Burst PPT2. In this period, DDRCTL temporarily ignores `dfi_t_ctrlupd_interval_type1`. Consecutive `ctrlupd_req` will be sent adding 24 cycles gap at minimum between the last `ctrlupd_req` going low and high².

1. You may reduce the number of times. '32 set' expects that the VT-drift is at the maximum.
2. This is a PHY requirement: PHY does not issue an ACK (and does not VT compensate the delay elements) when `dfi_ctrlupd_req` is asserted within 24 cycles of its de-assertion.

Assuming each `ctrlupd_req` consumes 500ns, Burst PPT2 will consume 256us (=500ns*32*(4*2*2)).

Figure 9-14 Burst PPT2 Schedule

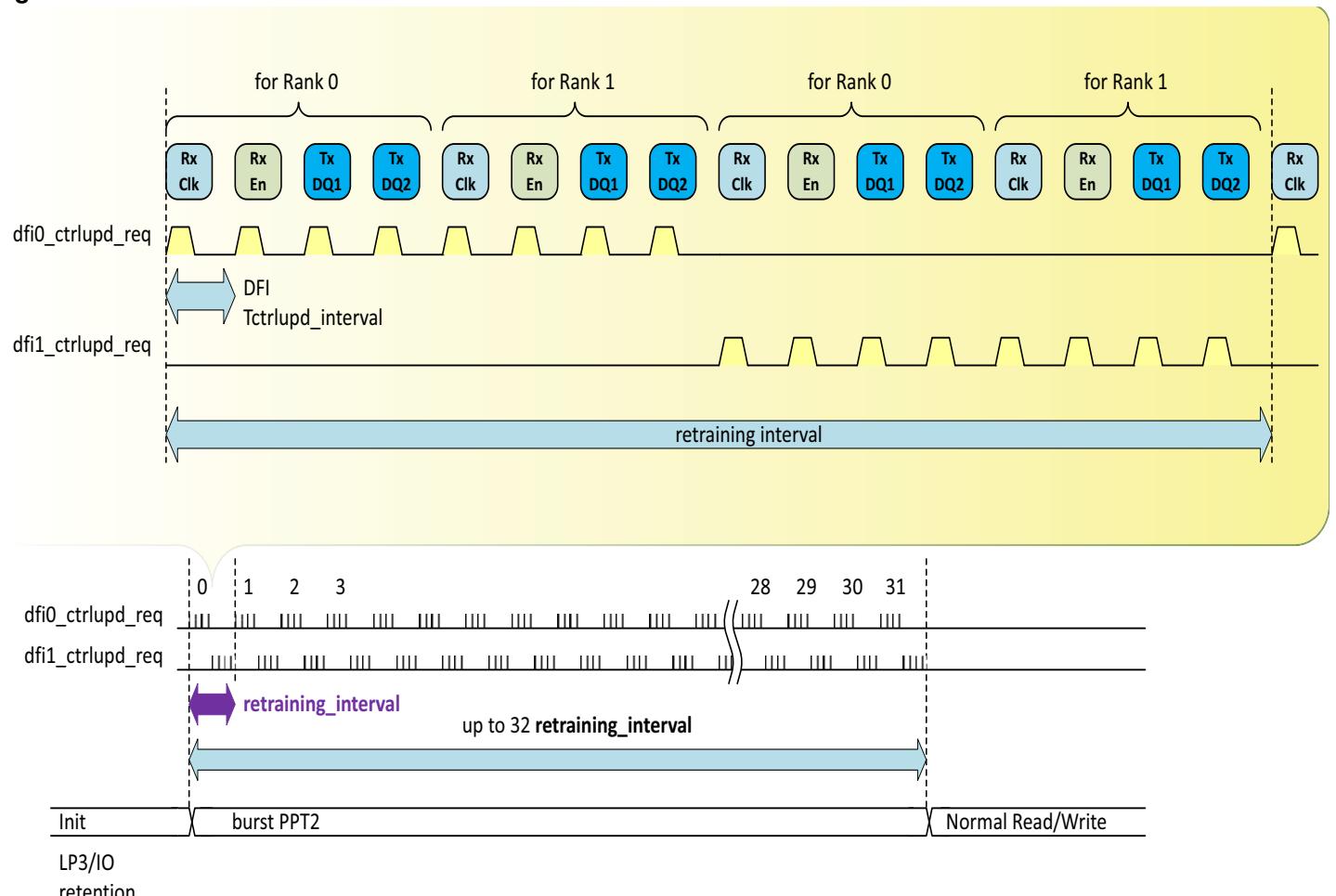
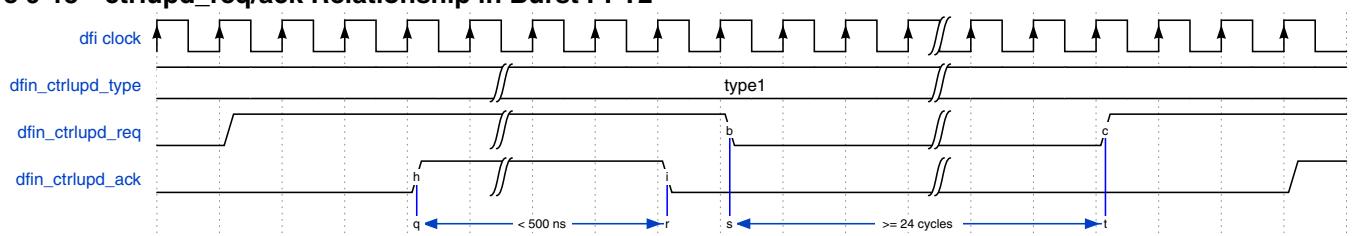


Figure 9-15 ctrlupd_req/ack Relationship in Burst PPT2



Any transaction that uses DQ bus, for example, READ, WRITE and MRR is not allowed once burst PPT2 starts and until it completes. However, other commands such as refresh and ZQCal may get in while burst PPT2 is ongoing and `ctrlupd_req` is low.

DDRCTL drives `cactive_ddrc` of the hardware low-power interface high during Burst PPT2 in order to deny any external low-power request.

To trigger the Burst PPT2, follow this procedure. Retraining functionalities other than PPT2 are assumed to be disabled (see “[PPT2 Limitations](#)” on page [240](#)).

1. Program DFI Update Timing Parameters, Refresh registers and $t_{ctrlupd_interval}$ according to the procedure on how to enable Normal PPT2 in “[Normal PPT2](#)” on page [242](#) if needed.
2. Program the sum of the number of times to send `ctrlupd_req` for dfi0 and dfi1.
 - Set `PPT2CTRL0.ppt2_burst_num`.
 - It will be ' $512 = 32 * (4 * 2^2)$ ' times for Link ECC enabled & 2-rank & dual DFI system.
3. Disable transactions which use DQ. Disable other features that can get in the burst PPT2 if needed.
 - Set `OPCTRL1.dis_hif` to '1', so that no new commands are accepted by the controller and poll `OPCTRLCAM.dbg_wr_q_empty=1`, `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.dbg_rd_q_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`.
 - Set `dis_dq` to '1'.
 - Set `derate_mr4_pause_fc` to '1' if `DERATECTL0.derate_eanble` is '1'.
 - Set `dis_auto_zq` to '1' if needed.
4. Disable low-power activities.
 - Set `PWRCTL.selfref_en` and `PWRCTL.powerdown_en` to '0'.
5. Start the Burst PPT2.
 - In a separate APB transaction, set `PPT2CTRL0.ppt2_burst` to '1'.
 - Note: `DFIUPDTMG2.ppt2_en` and `DFIUPD0.dis_auto_ctrlupd` can either be '0' or '1'.
6. Wait for Burst PPT2 to complete.
 - Burst PPT2 automatically stops when it completes.
 - Poll `PPT2STAT0.ppt2_burst_busy` until it is '0'.
7. Revert registers changed at step 3 and step 4.



Note While Burst PPT2 is running and once `dfi_ctrlupd_req=1` with `dfi_ctrlupd_type=1` is issued, DDRCTL does not issue `dfi_ctrlupd_req=1` with `dfi_ctrlupd_type=0` even if `dis_auto_ctrlupd` is '0'.

9.5.6 Registers Related to PPT2

The following are the registers related to PPT2:

- `PPT2CTRL0`
- `PPT2STAT0`
- `DFIUPDTMG2`

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5/4/4X Memory Controller Reference Manual.

9.6 Burst DFI Control Update Request for Core VDD Change

9.6.1 Overview of Burst DFI Control Update Request

Burst DFI control update is a controller-initiated multiple DFI control update request by asserting `dfi_ctrlupd_req` every certain period. It is required for core VDD change with Synopsys LPDDR5X/5/4X PHY and LPDDR5/4/4X PHY to change voltage.

The DDRCTL does not stop operation in point of performance during changing VDD except while DFI control update procedure is in progress.

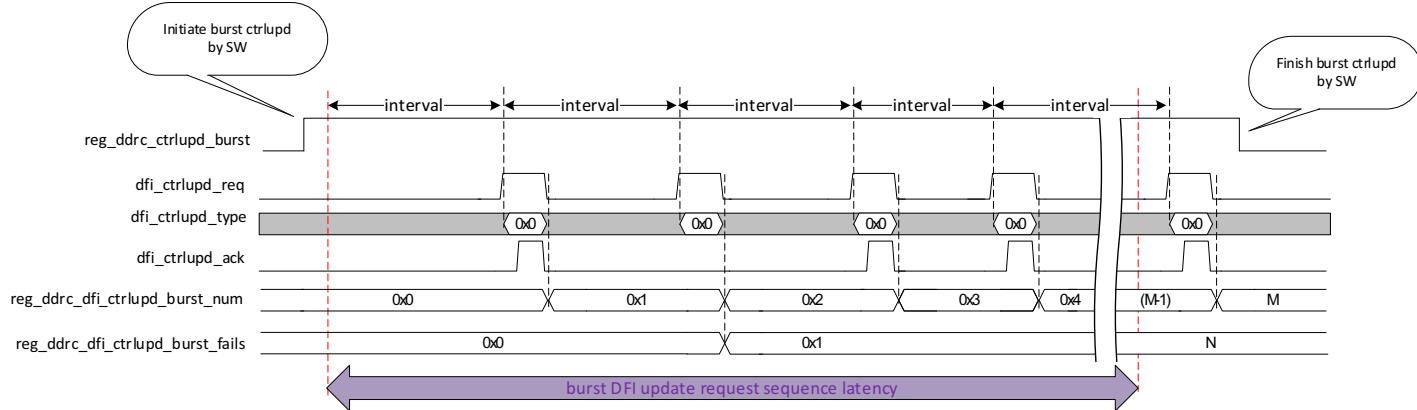
The software can start and stop this sequence through `OPTCTRLCMD.ctrlupd_burst`.

While DFI control update sequence is a part of burst, `OPTCTRLSTAT.ctrlupd_burst_busy` is set to '1'.

Once the sequence starts, a timer starts to track the interval for each `dfi_ctrlupd_req` and the initial value is programmed in `DFIUPDTMG1.dfi_t_ctrlupd_burst_interval_x8`.

When the interval timer is expired, DFI control update request is sent, and then if `OPTCTRLCMD.ctrlupd_burst` is set to '0' by the software, `OPTCTRLSTAT.ctrlupd_burst_busy` is cleared by the DDRCTL.

Figure 9-16 Burst DFI Control Update Request



9.6.2 Combination With Frequency Change

PHY defines {VDD, Data Rate} pairs. This indicates that for a given data rate, what is min VDD level is required. Here is an example: {750 mV, 6400 Mbps} and {700 mV, 4800 Mbps}.

This means 6400 Mbps can run only at 750 mV but 4800 Mbps can run at any VDD from 700 mV to 750 mV.

Therefore, frequency change between 4800 Mbps to 6400 Mbps or vice versa can only be supported when VDD is 750 mV.

In this example, core VDD change must be executed as follows.

Change VDD level (700 mV to 750 mV) before frequency change.

1. SOC can change VDD level slowly while mission mode traffic at 4800 Mbps is running.
2. The only thing SOC needs to ensure is that burst DFI updates are issued to calibrate PHY LCDL/IO codes.
3. Once VDD level reaches 750 mV, the DDRCTL can perform frequency change from 4800 Mbps to 6400 Mbps.

Change VDD level (750 mV to 700 mV) after frequency change.

1. The DDRCTL performs frequency change from 6400 Mbps to 4800 Mbps.
2. SoC can change VDD level slowly with burst DFI updates while mission mode traffic at 4800 Mbps is running.

9.6.3 Burst DFI Control Update Limitations

The following features are disabled automatically while burst DFI control update sequence is in progress.

- DFI PHY Master request
 - Disabled, the DDRCTL does not return acknowledge for PHY master.
- HWLP request
 - `cactive_ddrc` is to be set '1' while burst DFI control update is in progress

The software is required to turn off the following features while burst DFI control update sequence is in progress.

- DQSOSC request
 - Set `DQSOSCCCTL0.dqsosc_enable` to '0'
- ZQCAL request
 - Set `ZQCTL0.dis_auto_zq` to '1'
- Powerdown request
 - Set `PWRCTL.powerdown_en` to '0'
- SR request
 - Set `PWRCTL.selfref_en` to '0'
 - Set `PWRCTL.selfref_sw` to '0'
- PHY update response
 - Set `DFIUPD0.dfi_phyupd_en` to '0' (for more information about `DFIUPD0.dfi_phyupd_en` register settings, see the first Note in section “[Software Sequences](#)” on page [424](#)).
- DFI control message request
 - Set `DFI0MSGCTL0.dfi0_ctrlmsg_req` to '0'
- Disable PPT2 feature
 - Set `DFIUPDTMG2.ppt2_en` to '0'

If these features are changed, set back these registers to original values after burst DFI control update sequence.

10

Memory Control

This chapter contains the following sections:

- “Mode Register Reads and Writes” on page [252](#)
- “Data Bus Inversion (DBI)” on page [254](#)

10.1 Mode Register Reads and Writes

This topic contains the following sections:

- “[Overview of Mode Register Reads and Writes](#)” on page [252](#)
- “[Mode Register Writes](#)” on page [252](#)
- “[Mode Register Reads](#)” on page [252](#)
- “[Registers Related to Mode Register Reads and Writes](#)” on page [253](#)

10.1.1 Overview of Mode Register Reads and Writes

This section explains how to perform mode register reads and writes through software. Mode Register Reads (MRR) are used to read configuration and status data from mode registers in the SDRAM. Mode Register Writes (MRW) are applicable to all supported DDR protocols, and are used to write configuration data to mode registers in the SDRAM.

Access to the mode register is initiated by programming the MRCTRL0 and MRCTRL1 registers. This must be done in the following steps:

1. If performing MRR and using MRRDATA0 / MRRDATA1 registers, write the MRCTRL0.mrr_done_clr to '1'.
This bit is self-clearing, and clears the MRSTAT.mrr_done register.
2. Poll MRSTAT.mr_wr_busy until it is '0'. This checks that there is no outstanding MR transaction. No writes must be performed to MRCTRL0 and MRCTRL1 if MRSTAT.mr_wr_busy=1.
3. Write the MRCTRL0.mr_addr, MRCTRL0.mr_rank and (for MRWs) MRCTRL1.mr_data ([15:8] for MR addr and [7:0] for MR data for write) to define the MR transaction.
4. In a separate APB transaction, write the MRCTRL0.mr_wr to '1'. This bit is self-clearing, and triggers the MR transaction. The DDRCTL then asserts the MRSTAT.mr_wr_busy while it performs the MR transaction to SDRAM, and no further accesses can be initiated until it is de-asserted.
5. If performing MRR, the MRR data is made available on the hif_mrr_data signals and the MRRDATA0 / MRRDATA1 registers.

This section explains how to perform mode register reads and writes through software.



Note The sequences described above are dedicated for MRR and MRW, and they are tailed from general Software Command Interface programming sequence.

For more details, see “Software Command Interface” section.

10.1.2 Mode Register Writes

MRW transactions can be specified for either any single rank, or any combination of several ranks, by programming MRCTRL0.mr_rank.

If any part of the SDRAM’s MR register is updated by software, it is also the responsibility of the software to update corresponding INIT*. *mr* so that it is aligned to the SDRAM’s MR register in LPDDR4/5.

10.1.3 Mode Register Reads

MRR transactions must only be performed to one rank at a time, to avoid bus contention.

When a MRR is performed, the mode register contents are available on `hif_mrr_data`, qualified by `hif_mrr_data_valid` and the `MRRDATA0/MRRDATA1` register, after DDRCTL issues the mode register read command to the SDRAM. Note that the entire width of the SDRAM data is mapped to `hif_mrr_data` and `MRRDATA0/MRRDATA1` register. You must select the appropriate byte, depending on whether the bytes are swapped on the board or not.



Note For Mode Register Read, if you use `hif_mrr_data` rather than `MRRDATA0/MRRDATA1`, you must add SoC logic to capture read data.

10.1.4 Registers Related to Mode Register Reads and Writes

The following are the registers related to the Mode Register Reads and Writes in LPDDR5:

- `MRCTRL0`
- `MRCTRL1`
- `MRSTAT`
- `MRRDATA0`
- `MRRDATA1`

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

10.2 Data Bus Inversion (DBI)

This topic contains the following sections:

- “[Overview of Data Bus Inversion \(DBI\)](#)” on page [254](#)
- “[Registers Related to DBI](#)” on page [255](#)

10.2.1 Overview of Data Bus Inversion (DBI)

Data Bus inversion is a power saving LPDDR5/4 specific feature which aims at minimizing:

- Logic 1s (consume more power than logic 0s)
- Simultaneous switching outputs

This is done by inverting each byte of read or write data, if the number of ‘1’s in that byte is greater than 4.

A DBI signal (1 bit per byte of data) is used to indicate whether this has been done or not. For writes, the DBI signal is shared with `dfi_wrdata_mask` on the DFI interface. For reads, there is a dedicated `dfi_rddata_db1` signal.

Both write DBI and Data Mask are supported at the same time.

In this mode, a masked byte is indicated by DDRCTL setting the `dfi_wrdata` byte to 0xFF or 0xF8 and setting the `dfi_wrdata_mask` bit to ‘0’.

Both read DBI and link ECC cannot be enabled simultaneously.

The DFI Specification indicates that DBI can be implemented either in the controller or in PHY. In the DDRCTL, DBI is controlled by `DFIMISC.phy_dbi_mode`. If this is set to ‘1’, DBI generation and data inversion are performed in the PHY. The read and write data on the DFI interface are same as that of when DBI is disabled (though masked writes are not allowed), and the signal `dfi_rddata_db1` is ignored. When `DFIMISC.phy_dbi_mode` is set to ‘0’, DBI generation and data inversion is done by the DDRCTL. During write operation, the DBI value is put on the signal `dfi_wrdata_mask`. During read operation, `dfi_rddata_db1` is evaluated and used to determine if the polarity of the read data needs to be inverted. The register DBICTL contains three fields namely:

- `rd_dbi_en` - enables read DBI
- `wr_dbi_en` - enables write DBI
- `dm_en` - enables data masking

The software must ensure that these values correspond to the values written to the mode register.



Note Data Mask Disable (DMD) which is defined by `MR13`, has the opposite polarity of `DBICTL.dm_en`.

If read DBI is enabled, the JEDEC Specification indicates that the read latency (RL) and `tWCKENL_RD` (LPDDR5/5X only) must be increased. So, the following DDRCTL registers must be adjusted accordingly as they depend on RL or `tWCKENL_RD`:

- `DFITMG0.dfi_t_rddata_en`
- `DRAMSET1TMG2.read_latency`
- `DRAMSET1TMG2.rd2wr`

- DFITMG2.dfi_tphy_rdcslat
- DRAMSET1TMG24.rd2wr_s (LPDDR5 only)
- INIT4.emr2 (LPDDR4 only)
- DRAMSET1TMG3.rd2mr
- DRAMSET1TMG24.max_rd_sync (LPDDR5 only)

The required sequence for enabling or disabling DBI is as follows:

1. Enter the appropriate mode for writing quasi-dynamic registers.
2. Set DBICTL.rd_dbi_en and DBICTL.wr_dbi_en as required.
3. Set the DRAM mode register to match the previous settings.
4. Adjust the registers depending on CL (or RL) and AL (listed previously) as appropriate.
5. Exit quasi-dynamic mode by exiting self-refresh, or by re-enabling read/write traffic (For more information, see “Group 1: Registers that can be Written when No Read/Write Traffic is Present at the DFI” or “Group 2: Registers that can be Written in Self-refresh” in the “Programming” chapter).

10.2.2 Registers Related to DBI

For more information on registers related to the DBI, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

11

Low-Power and Power-Saving Features

This chapter contains the following sections:

- “[Overview of Power Saving Features](#)” on page [258](#)
- “[SDRAM Power Saving Features](#)” on page [259](#)
- “[Power Saving in PHY Through DFI Low-Power Control Interface](#)” on page [269](#)
- “[Hardware Low-Power Interfaces](#)” on page [270](#)
- “[Fast Frequency Change](#)” on page [281](#)
- “[Hardware Fast Frequency Change \(HWFFC\) Support](#)” on page [282](#)
- “[Low-Power Optimized Write Data for LPDDR5/4 Masked Write](#)” on page [309](#)
- “[BSM Clock Removal](#)” on page [310](#)
- “[Signals Related to Power Saving](#)” on page [313](#)

11.1 Overview of Power Saving Features

The DDRCTL supports various methods to save power within the system:

- Power saving opportunities within the SDRAM. The DDRCTL supports various SDRAM power saving modes such as precharge power-down, self-refresh, deep sleep mode, and support for disabling clock to the DRAM through `dfi_dram_clk_disable`.
For more information about this, see “[SDRAM Power Saving Features](#)” on page [259](#).
- Power saving opportunities within the PHY. For more information about this, see “[Power Saving in PHY Through DFI Low-Power Control Interface](#)” on page [269](#).
- Power saving opportunities from an external SoC low-power controller driven through an external hardware low-power interface (based on AMBA 4 AXI protocol low-power control interface). For more information about this, see “[Hardware Low-Power Interfaces](#)” on page [270](#).
- Power saving opportunities within the internal module BSM (Bank State Machine), whose clock can be gated while SDRAM is idle. For more information, see “[BSM Clock Removal](#)” on page [310](#).

11.2 SDRAM Power Saving Features

This topic contains the following sections:

- “[Overview of SDRAM Power Saving Features](#)” on page [259](#)
- “[Precharge Power-Down](#)” on page [260](#)
- “[Self-Refresh](#)” on page [261](#)
- “[Deep Sleep Mode \(LPDDR5\)](#)” on page [265](#)
- “[Assertion of dfi_dram_clk_disable](#)” on page [267](#)

11.2.1 Overview of SDRAM Power Saving Features

In multi-rank systems, these power saving modes cannot be applied on a per-rank basis. If applied, they are always applied globally.

When enabled, the DDRCTL automatically enters and exits precharge power-down mode based on a programmable idle timeout period.

Self-refresh can be entered/exited through three ways:

- Based on a programmable idle timeout period (similar to precharge power-down idle timeout)
- Explicitly controlled through software
- Through the hardware low-power interfaces

In addition, `dfi_dram_clk_disable` can be asserted to disable the clocks to the DRAM. This can be done in the following modes:

- Self-refresh power-down
- Power-down
- Deep Sleep Mode (LPDDR5 only)

It also includes clock stop.

In relation to the hardware low-power interface, optional support is provided to drive `cactive_ddrc` low under idle conditions in normal or power-down or automatic self-refresh modes. Optional support for power-down, self-refresh, and clock stop to be forcefully exited through `cactive_in_ddrc=1` is also provided.



Note It is valid to enable any combination of Power-down and Self-refresh modes simultaneously:

- Power-down: `PWRCTL.powerdown_en=1`
- Automatic self-refresh: `PWRCTL.selfref_en=1`
- Software self-refresh: `PWRCTL.selfref_sw=1`

Enabling assertion of `ddrc_dfi_dram_clk_disable` is valid in combination with any of the power saving modes.



Note The idle timer fields in PWRTMG/HWLPCCTL register continuously count down the programmed values once the controller enters an idle state irrespective of the enable bits in PWRCTL or HWLPCCTL:

- PWRTMG.powerdown_to_x32
- PWRTMG.selfref_to_x32
- HWLPTMG0.hw_lp_idle_x32

11.2.2 Precharge Power-Down

This section discusses entering and exiting Precharge Power-down.

11.2.2.1 Entering Precharge Power-Down

When PWRCTL.powerdown_en=1, DDRCTL automatically enters precharge power-down when the period specified by PWRTMG.powerdown_to_x32 has passed while the DDRCTL is idle (except for issuing refreshes).

Entering precharge power-down mode involves the following steps:

1. If there is a self-refresh exit previously, wait for at least one refresh command (or 8 per-bank refresh commands if per-bank refresh is enabled) to all active ranks. Auto-refresh logic must be enabled, or refresh must be issued using direct software requests of refresh command through OPREFCTRL*.rank*_refresh.
2. Precharging (closing) all open pages. Pages are closed one-at-a-time in no specified order.
3. Waiting for tRP (row precharge) idle period.
4. Issuing the command to enter precharge power-down. For multi-rank systems, all chip-selects are asserted so that all ranks enter precharge power-down simultaneously. Power-down entry commands are issued separately for even and odd ranks.
5. This step occurs only if DFI low-power interface for power-down is enabled (DFILPCFG0.dfi_lp_en_pd). Attempts an entry to low-power mode through DFI low-power interface and with both dfi_lp_ctrl1_wakeup and DRAMSET1TMG5.t_cksre set by DFILPTMG0.dfi_lp_wakeup_pd. The low-power entry attempt is delayed with DFITMG0.dfi_t_ctrl1_delay + DRAMSET1TMG7.t_cksre clock cycles, this is needed to satisfy SDRAM timings related to disabling clocks when the PHY is programmed to gate the clock to save maximum power.

If the DDRCTL receives a read or write request from the SoC during step 2 or step 3, the power-down entry is immediately canceled. The same is true if PWRCTL.powerdown_en is driven to '0' during step 2 or step 3. Once the power-down entry command is issued, then proper power-down exit is required.



Note Even if PWRCTL.powerdown_en=1 and no outstanding commands are present in DDRCTL, the controller will not enter precharge power-down if all the following conditions are true:

- Either arvalid, awvalid, or wvalid is set to '1' when PCTRL.port_en=0 on any AXI port
- HWLPCCTL.hw_lp_en=1

11.2.2.2 Exiting Precharge Power-Down

Once the DDRCTL has put the DDR SDRAM devices in precharge power-down mode, the DDRCTL automatically performs the precharge power-down exit sequence for any of the following reasons:

- A refresh cycle is required to any rank in the system.
- The DDRCTL receives a new request from the SoC.
- A self-refresh entry is requested.
- PWRCTL.powerdown_en is set to '0'.

The DDRCTL follows these steps when exiting precharge power-down mode:

1. Inserting any NOP/deselect commands required to satisfy the tCKE requirement after entering precharge power-down.
2. This step occurs only if DFI low-power mode entry during power-down entry is successful. Performs an exit from DFI low-power mode. DFI low-power mode is exited after the wakeup time specified by DFILPTMG0.dfi_lp_wakeup_pd, but not earlier than DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG6.t_cksrcx clock cycles.
3. Issuing the power-down exit command. For multi-rank systems, all chip-selects are asserted so that all ranks exit precharge power-down simultaneously.
4. Issuing NOP/deselect for the period defined by tXP.



Note Other supported DDR protocols do not specify the difference between fast and slow power-down exit.

11.2.3 Self-Refresh

This section contains the following subsections:

- “[LPDDR4/LPDDR5 Considerations](#)” on page [261](#)
- “[Entering Self-Refresh Mode](#)” on page [263](#)
- “[Exiting Self-Refresh](#)” on page [264](#)

11.2.3.1 LPDDR4/LPDDR5 Considerations

The following are the self-refresh states for LPDDR4/LPDDR5:

- Self-Refresh
- Self-Refresh Power-Down

During Self-Refresh mode, the external clock input is needed and all input pins of SDRAM are active. SDRAM can accept the following commands, except the PASR Bank/Segment setting:

LPDDR4

- MRR-1
- CAS-2
- SRX

- MPC
- MRW-1
- MRW-2

LPDDR5

- PDE
- DSM
- MRR
- CAS
- SRX
- MPC
- MWR

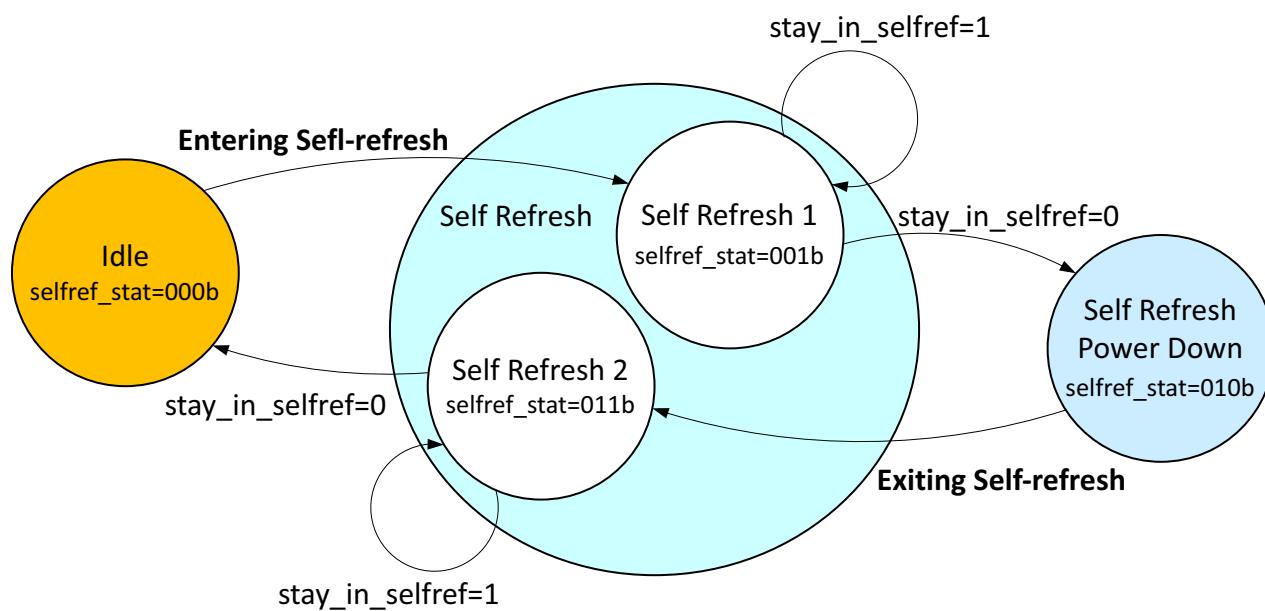
The DDRCTL has `STAT.selfref_state` register which indicates the DDRCTL's state during Self-Refresh and Self-Refresh Power-down modes. Once the DDRCTL enters `STAT.selfref_state=3'b001`, the next state must be `3'b010`. Also, the next state after `STAT.selfref_state=3'b011` must be `3'b000` except for re-entering self-refresh state due to `lpddr4_sr_allowed=1`.

[Table 11-1](#) lists the DDRCTL states and DRAM states for self-refresh.

Table 11-1 List of DDRCTL State and DRAM State for Self-Refresh

STAT.selfref_state	DDRCTL State	DRAM State
000	Neither Self-Refresh nor Self-Refresh Power-Down	Neither Self-Refresh nor Self-Refresh Power-Down
001	Self-Refresh 1	Self-Refresh
010	Self-Refresh Power-Down	Self-Refresh Power-Down
011	Self-Refresh 2	Self-Refresh
100 (LPDDR5 only)	Deep Sleep Mode	Deep Sleep Mode

The DDRCTL has `PWRCTL.stay_in_selfref` register to stay in Self-Refresh mode. During Self-Refresh mode, the DDRCTL can issue MRR, MRW, and SRX commands.

Figure 11-1 State Transition of Self-Refresh for LPDDR4

11.2.3.2 Entering Self-Refresh Mode

The DDRCTL puts the DDR SDRAM devices into Self-Refresh mode in the following cases:

- When the PWRCTL.selfref_en bit is set and no reads or writes are pending in the DDRCTL for the period specified by PWRTMG.selfref_to_x32. This is referred to as automatic Self-Refresh.
- When the PHY Master Interface is enabled by setting DFIPHYMSTR.dfi_phymstr_en bit and there is a dfi_phymstr_req coming from the PHY. In this case the DDRC's hif_cmd_stall is not driven high (the controller can accept commands on HIF) and existing controller commands in the DDRC are not executed before the entering the Self-Refresh mode sequence occurs.
- When the PWRCTL.selfref_sw bit is set and there are no outstanding read or write commands in the DDRC, this is referred to as the software self-refresh entry. This means that the DDRCTL cannot put SDRAM into Self-Refresh as long as write/read commands are being entered into the DDRCTL.
- When a hardware low-power entry request occurs (on csysreq_ddrc/csysack_ddrc) with cactive_in_ddrc=0, no outstanding commands, and as long as the DDRCTL is not in init or deep sleep mode. This is referred to as an accepted hardware low-power self-refresh entry. When accepted, the DDRC's hif_cmd_stall is driven high to stop new commands from being accepted and existing controller commands in the DDRC are performed before the following sequence occurs.

Entering Self-Refresh mode involves the following steps:

1. If there is a self-refresh exit previously, wait for at least one refresh command (or 8 per-bank refresh commands if per-bank refresh is enabled) to all active ranks. Auto-refresh logic must be enabled, or refresh must be issued using direct software requests of refresh command through OPREFCTRL*.rank*_refresh.
2. Precharging (closing) all open pages. Pages are closed one-at-a-time in no specified order.
3. Waiting for tRP (row precharge) idle period. If a new command is received on the HIF during this time, the self-refresh entry is canceled.

4. Issuing the command to enter self-refresh mode. For multi-rank systems, all chip-selects are asserted so that all ranks enter self-refresh simultaneously.

If `PWRCTL.stay_in_selfref` is set to '1' before the SRE command is issued, DDRCTL does not enter the Self-Refresh Power-Down mode. In this case, right after `PWRCTL.stay_in_selfref` is set to '0', DDRCTL enters the Self-Refresh Power-Down mode. When `PWRCTL.stay_in_selfref` is '0' before SRE command issued, DDRCTL enters the Self-Refresh Power-Down mode automatically.

If the PHY Master Interface is enabled and `dfi_phymstr_req` comes before the SRE command is issued, DDRCTL does not enter the Self-Refresh Power-Down mode. In this case, right after `dfi_phymstr_req` is dropped, DDRCTL enters Self-Refresh Power-Down mode.

5. This step occurs only if DFI low-power interface for self-refresh is enabled (`DFILPCFG0.dfi_lp_en_sr`). Attempts an entry to low-power mode through DFI low-power interface with both `dfi_lp_ctrl_wakeup` and `dfi_lp_data_wakeup` set by `DFILPTMG0.dfi_lp_wakeup_sr`. The low-power entry attempt is delayed with `DFITMG0.dfi_t_ctrl_delay + DRAMSET1TMG5.t_ckssre` clock cycles, this is needed to satisfy SDRAM timings related to disabling clocks when the PHY is programmed to gate the clock, to save maximum power.

Note, that `STAT.selfref_type` register field is 2'b11 if automatic self-refresh feature is the only cause of self-refresh. If software self-refresh or hardware low-power self-refresh occurs, `STAT.selfref_type=2'b10`.

If Self-refresh entry is triggered by a PHY Master request, this step is skipped because DFI low-power interface is disabled when there is a `dfi_phymstr_req`.

Automatic self-refresh has the lowest priority followed by both software and hardware low-power self-refresh, PHY Master Interface has the highest priority. A software self-refresh entry means that a self-refresh exit occurs only if a software self-refresh exit occurs. Similarly, a hardware low-power self-refresh entry means a self-refresh exit occurs only if a hardware low-power self-refresh exit occurs. If both software and hardware low-power self-refresh entry occurs, self-refresh exit occurs only if both software and hardware low-power self-refresh exits occur. A self-refresh entry triggered by a PHY Master request means a self-refresh exit occurs only if the PHY Master request is de-asserted.

11.2.3.3 Exiting Self-Refresh

The DDRCTL takes the DDR SDRAM out of self-refresh mode under the following conditions:

- Whenever the `PWRCTL.selfref_en` input is set to '0', or new commands are received by the DDRCTL, as long as automatic Self-Refresh is only cause for self-refresh entry.
- When the PHY Master Interface is enabled by setting `DFIPHYSR.dfi_phymstr_en` bit and `dfi_phymstr_req` is de-asserted by the PHY.
- Whenever the `PWRCTL.selfref_sw` bit is de-asserted. This is referred to as software Self-Refresh exit.
- When a hardware low-power exit request occurs (on `csysreq_ddrc/csysack_ddrc`). This is referred to as hardware low-power self-refresh exit.

Exiting Self-Refresh mode involves the following steps:

1. Inserting any NOP/deselect commands required to satisfy the tCKE/tCKESR/tSR requirements after entering Self-Refresh.
2. If `DFIUPD0.dis_auto_ctrlupd_srx=0` and `DFIUPD0.ctrlupd_pre_srx=1`, issuing a `dfi_ctrlupd_req` in accordance with the DFI specification. If Self-refresh is entered, without

Power-Down, `dfi_ctrlupd_req` is not issued. This situation can occur only when there is a PHY Master request.

3. This step occurs only if DFI low-power mode entry during self-refresh entry is successful. DFI low-power mode is exited after the wakeup time specified by `DFILPTMG0.dfi_lp_wakeup_sr`, but not earlier than `DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG5.t_cksr` clock cycles.
4. Issuing the self-refresh exit command.

If `PWRCTL.stay_in_selfref` is set to '1' before Self-Refresh Power-Down exit command is issued, DDRCTL does not exit Self-Refresh. In this case, right after `PWRCTL.stay_in_selfref` is set to '0', DDRCTL exits Self-Refresh and enters the Idle state. When `PWRCTL.stay_in_selfref` is 0 before self-refresh power-down command issued, DDRCTL exits Self-Refresh and enters the Idle state automatically.

If the PHY Master Interface is enabled and `dfi_phymstr_req` comes before the Self-Refresh Power-Down exit command is issued, DDRCTL does not exit Self-Refresh. In this case, right after `dfi_phymstr_req` is de-asserted, DDRCTL exits Self-Refresh and enters the Idle state.

5. If `DFIUPD0.dis_auto_ctrlupd_srx=0` and `DFIUPD0.ctrlupd_pre_srx=0`, issuing a `dfi_ctrlupd_req` in accordance with the DFI specification. If Self-Refresh is entered, without Power-Down, `dfi_ctrlupd_req` is not issued. This situation can occur only when there is a PHY Master request.
6. Issuing NOP/deselect as determined by protocol requirements before any relevant pending command can be executed (in parallel with the `dfi_ctrlupd_req`, if applicable).



Note When `PWRCTL.stay_in_selfref` is set to '1', DDRCTL does not exit Self-Refresh. Therefore, the SoC must not set it to '1' unless SoC must issue MRR/MRW during Self-Refresh.

11.2.4 Deep Sleep Mode (LPDDR5)

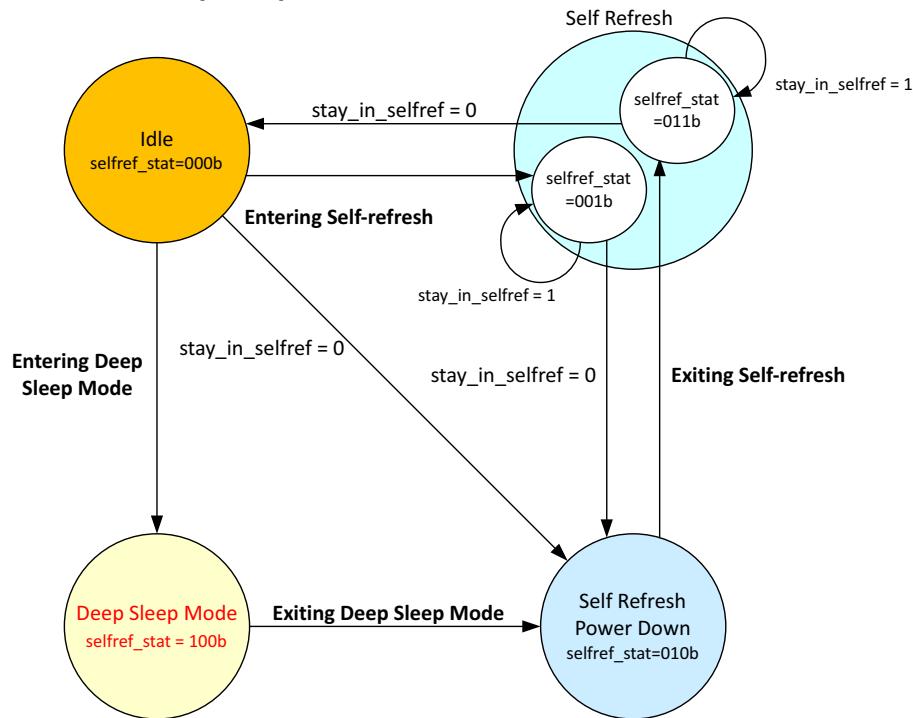
Deep Sleep Mode (DSM) is applicable for LPDDR5 devices only.

11.2.4.1 Entering Deep Sleep Mode

By setting the `PWRCTL.dsm_en` bit, you can put the LPDDR5 devices into Deep Sleep Mode, if all the following conditions are true:

- The DDRCTL is idle (except for issuing refreshes)
- `PWRCTL.selfref_sw=0`
- `PWRCTL.selfref_en=0`
- `HWPCTL.hw_lp_en=0`

When the controller is in Deep Sleep Mode, the controller does not acknowledge the PHY master request (`dfi_phymstr_req`). Therefore, PHY master feature of the PHY must be disabled before entering Deep Sleep Mode. If PHY master feature of the PHY cannot be disabled after initialization, the register `DFIPHYSMSTR.dfi_phymstr_en` needs to be set to '0' so that the DDRCTL does not set `dfi_phymstr_ack=1`. In this case, the PHY does not initiate retraining, and the signal `dfi_error` is asserted if the hardware parameter `DDRCTL_DFI_ERROR` is enabled.

Figure 11-2 State Transition of Deep Sleep Mode and Self-Refresh for LPDDR5

Entering Deep Sleep Mode involves the following steps:

1. If there is a self-refresh exit previously, wait for at least one refresh command (or 8 per-bank refresh commands if per-bank refresh is enabled) to all active ranks. Auto-refresh logic must be enabled, or refresh must be issued using direct software requests of refresh command through `OPREFCTRL*.rank*_refresh`.
2. Precharging (closing) all open pages. Pages are closed one-at-a-time (not in a specified order).
3. Waiting for tRP (row precharge) idle period.
4. Issuing the SRE command with DSM=1 to enter Deep Sleep Mode. For multi-rank systems, SRE commands must be sent to all ranks. This happens simultaneously.
5. This step occurs only if DFI low-power interface for Deep Sleep Mode is enabled (`DFILPCFG0.dfi_lp_en_dsm`). It attempts an entry to low-power mode through DFI low-power interface with both `dfi_lp_ctrl_wakeup` and `dfi_lp_data_wakeup` set by `DFILPTMG0.dfi_lp_wakeup_dsm`. The low-power entry attempt is delayed with `DFITMG0.dfi_t_ctrl_delay + DRAMSET1TMG11.t_ckmpe` clock cycles, this is needed to satisfy SDRAM timings related to disabling clocks when the PHY is programmed to gate the clock, to save maximum power.

If the DDRCTL receives a read or write request from the SoC during step 1 or step 2, the Deep Sleep Mode entry is immediately canceled. The same is true if `PWRCTL.dsm_en` is driven to '0' during step 1 or step 2. Once the Deep Sleep Mode entry command is issued, proper Deep Sleep Mode exit is required as described in the following section.

11.2.4.2 Exiting Deep Sleep Mode

Once the DDRCTL puts the DDR SDRAM devices in Deep Sleep Mode, and when the following software sequence is performed, the DDRCTL exits Deep Sleep Mode followed by Self-Refresh Power-Down mode, and eventually the DDRCTL exits Self-Refresh mode:

1. Before exiting DSM, it is assumed that `PWRCTL.dsm_en` is set to '1'.
2. Program `PWRCTL.selfref_sw=1'b1`.
3. Program `PWRCTL.dsm_en=1'b0` to exit power-down mode.
4. Polling until `STAT.selfref_state=3'b010` (it means self-refresh power-down).
5. Wait for `tXDSM_XP`.
6. Program `PWRCTL.selfref_sw=1'b0` to exit self-refresh power-down.



Note In the DDRCTL, an exit from DFI low-power mode is performed prior to exiting the Deep Sleep Mode (occurs only if DFI low-power mode entry during Deep Sleep Mode is successful). DFI low-power mode exits after the wakeup time specified by `DFILPCFG0.dfi_lp_wakeup_dsm`, but not earlier than `DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG5.t_cksrcx` clock cycles (`tCKMPX` value is the same as `tCKSRX`).

11.2.5 Assertion of `dfi_dram_clk_disable`

Assertion of `dfi_dram_clk_disable` occurs only if `PWRTL.en_dfi_dram_clk_disable=1`.

`dfi_dram_clk_disable` is also dependent on the operating mode:

- `dfi_dram_clk_disable` can be asserted in the following modes:
 - Deep Sleep Mode (LPDDR5 only)
 - Self-refresh power-down
 - Power-down
 - Normal mode

This is the "Clock Stop" feature.



`dfi_dram_clk_disable` cannot be asserted in WCK always on mode (`MSTR4.wck_on=1`) for LPDDR5.

The timing of the assertion and de-assertion of `dfi_dram_clk_disable` in various modes is as follows:

- In Self-Refresh and Self-Refresh power-down:
 - Asserted at least `DFITMG0.dfi_t_ctrl_delay + DRAMSET1TMG5.t_cksre - DFITMG1.dfi_t_dram_clk_disable` cycles after SRE command.
 - De-asserted at least `DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG5.t_cksrcx - DFITMG0.dfi_t_ctrl_delay` cycles before SRX command.
- In Power-down:

- ❑ Asserted at least DFITMG0.dfi_t_ctrl_delay + DRAMSET1TMG5.t_ckssre - DFITMG1.dfi_t_dram_clk_disable cycles after PDE command.
- ❑ De-asserted at least DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG5.t_ckssrx - DFITMG0.dfi_t_ctrl_delay cycles before PDX command.
- In Normal mode (Clock Stop):
 - ❑ Asserted at least DFITMG0.dfi_t_ctrl_delay - DFITMG0.dfi_t_dram_clk_disable cycles after any command other than SRPDE/PDE/DSME.
 - ❑ De-asserted at least DFITMG1.dfi_t_dram_clk_enable + DRAMSET1TMG6.t_ckcsx - DFITMG0.dfi_t_ctrl_delay cycles before any command other than SRPD/X/PDX/DSMX.

For more information about all of the modes, see REGB_FREQf_CHc in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

11.3 Power Saving in PHY Through DFI Low-Power Control Interface

Based on whether SDRAM is in self-refresh power-down, power-down, deep sleep mode (LPDDR5 only) (see “[SDRAM Power Saving Features](#)” on page [259](#)), the PHY can be placed in power saving modes through the DFI low-power interface. For more information, see “[Low-Power Control Interface](#)” on page [104](#).

11.4 Hardware Low-Power Interfaces

This topic contains the following sections:

- “[Overview of Hardware Low-Power Interfaces](#)” on page [270](#)
- “[DDRC Hardware Low-Power Interface](#)” on page [270](#)
- “[AXI Low-Power Interface](#)” on page [275](#)

11.4.1 Overview of Hardware Low-Power Interfaces

Power saving opportunities from an external SoC low-power controller are driven through an external hardware low-power interface (based on AMBA 4 AXI protocol low-power control interface). There is a separate hardware low-power interface for each AXI port and dedicated hardware low-power interface for the DDRC.

11.4.2 DDRC Hardware Low-Power Interface

The hardware low-power interface can be used to enter/exit self-refresh mode. For this four-signal interface, the functionality is enabled by setting `HWLPCTL.hw_lp_en=1`.



- For a single port, HIF only, `cactive_in_ddrc` is 1-bit wide and is driven by an external port. Else, `cactive_in_ddrc` is 1-bit per port, and each bit is driven by `cactive_n` of each port.
- `cactive_in_ddrc` is used internally to evaluate whether self-refresh entry is allowed. It might have more registering stages than `csysreq_ddrc` to avoid CDC issues.

11.4.2.1 Entering Self-Refresh Using DDRC

Once the `HWLPCTL.hw_lp_en=1` bit is set, the input `csysreq_ddrc` can trigger the entry into the self-refresh operating mode. A hardware low-power entry trigger is ignored/denied if any bit of the input `cactive_in_ddrc=1`, or if the DDRC is not empty, or if a controller driven R/W/RMW command is on the controller interface (`hif_cmd_valid=11`). Otherwise, it can be accepted depending on the current state of the DDRCTL. If accepted, the DDRC’s `hif_cmd_stall=1` to stop new commands from being accepted, existing controller commands in DDRC are performed before self-refresh entry occurs.

In [Figure 11-3](#) on page [272](#), at T1, an external low-power controller drives `csysreq_ddrc` to '0' to request the DDRCTL entry into low-power state. After the DDRCTL recognizes the request, it performs a power-down sequence. At T2, it drives `cactive_ddrc` to '0', signaling that `core_ddrc_core_clk` can be removed. At T3, the DDRCTL drives `csysack_ddrc` to '0', signaling that it has completed the entry into low-power state.

The `cactive_ddrc` output can also be used by an external low-power controller to decide when to request a transition to self-refresh. When `HWLPTMG0.hw_lp_idle_x32 > 0`, `cactive_ddrc=0` indicates that the DDRCTL is idle for at least `HWLPTMG0.hw_lp_idle_x32 * 32 * DRAM_clock cycles`.

1. `hif_cmd_valid` refers to `hif_cmd_valid` if `UMCTL2_DUAL_HIF=0`, or `hif_rcmd_valid || hif_wcmd_valid` if `UMCTL2_DUAL_HIF=1`.

11.4.2.2 Exiting Self-Refresh Using DDRC

In self-refresh mode, the `cactive_ddrc` output can be used by an external low-power controller to trigger a self-refresh exit. Always, `cactive_ddrc` is driven high when any bit of `cactive_in_ddrc=1` or `hif_cmd_valid=1` or `(dfi_phyupd_req=1 && DFIUPD0.dfi_phyupd_en=1)` or `(dfi_phymstr_req=1 && DFIPHYMSTR.dfi_phymstr_en=1)` or `PPT2STAT0.ppt2_burst_busy=1` or if there are outstanding commands in the DDRC. The paths from `cactive_in_ddrc`, `hif_cmd_valid`, `dfi_phymstr_req` and `dfi_phyupd_req` to `cactive_ddrc` are asynchronous. Therefore, `cactive_ddrc` asserts even if the clocks are removed. The low-power controller must re-enable the clocks when `cactive_ddrc` is driven high while in the self-refresh state.



- The paths from `cactive_in_ddrc`, `hif_cmd_valid`, `dfi_phyupd_req`, and `dfi_phymstr_req` to `cactive_ddrc` are asynchronous.
- `cactive_in_ddrc` and `hif_cmd_valid` are input ports of the DDRCTL in HIF only configurations. Otherwise, they are internal signals of the DDRCTL.
- `dfi_phyupd_req` is an input of the DDRCTL in all configurations. `dfi_phyupd_req=1` generates `cactive_ddrc=1` asynchronously (as long as `DFIUPD0.dfi_phyupd_en=1`).

The behavior of the DDRCTL with respect to automatic clock stop, automatic precharge power-down or automatic self-refresh and `cactive_in_ddrc` is selectable through software (`HWLPCCTL.hw_lp_exit_idle_en`). If `HWLPCCTL.hw_lp_exit_idle_en=1`, automatic clock stop, automatic precharge power-down and automatic self-refresh is exited (if `cactive_in_ddrc=1`) and they do not occur until `cactive_in_ddrc=0`. Note that it does not cause the exit of self-refresh by DDRC hardware low-power interface and/or software (`PWRCTL.selfref_sw`). This improves memory utilization performance. `cactive_in_ddrc` informs DDRC that there is RD/WR command imminent, and DDRC exits the various SDRAM power saving modes early so that it can perform the RD/WR sooner.

11.4.2.3 Hardware Removal Of Clock From DDRC

`core_ddrc_core_clk` can be removed by:

- Hardware low-power interface handshaking

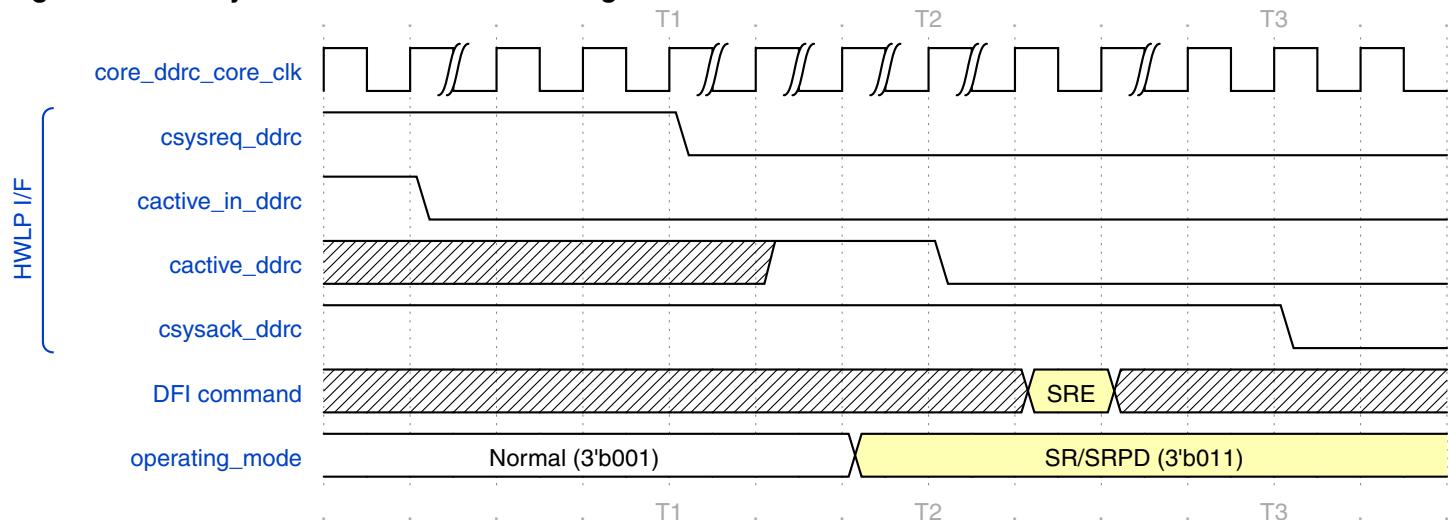


- Dynamic and quasi dynamic registers can not be programmed if any of the clocks has been removed. Clocks must be turned back on before starting the programming sequence. Also the clock gating logic must ensure there are no glitches on the clocks when there are removed/enabled.

11.4.2.4 Removal of DDRC Clock Through Hardware Low-Power Interface Handshaking

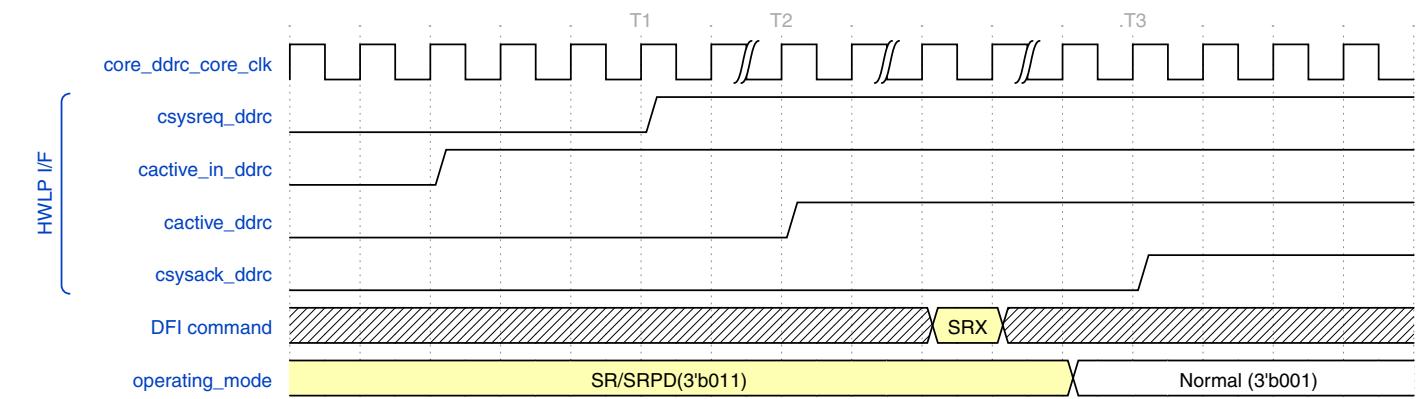
The `core_ddrc_core_clk` can be removed through hardware low-power interface handshaking. An entry to self-refresh mode occurs as shown in [Figure 11-3](#). In this example, the system initiates the low-power state entry through de-assertion of `cysreq_ddrc`.

For HIF only, `core_ddrc_core_clk` can be removed from the DDRC if `cactive_ddrc=0` when `cysack_ddrc` goes low. For more information about DDRC clock removal for non-HIF only configurations, see “[Hardware Removal of Clocks from AXI Ports](#)” on page [278](#).

Figure 11-3 Entry to Self-Refresh Mode Through Hardware Low-Power Interface of DDRC

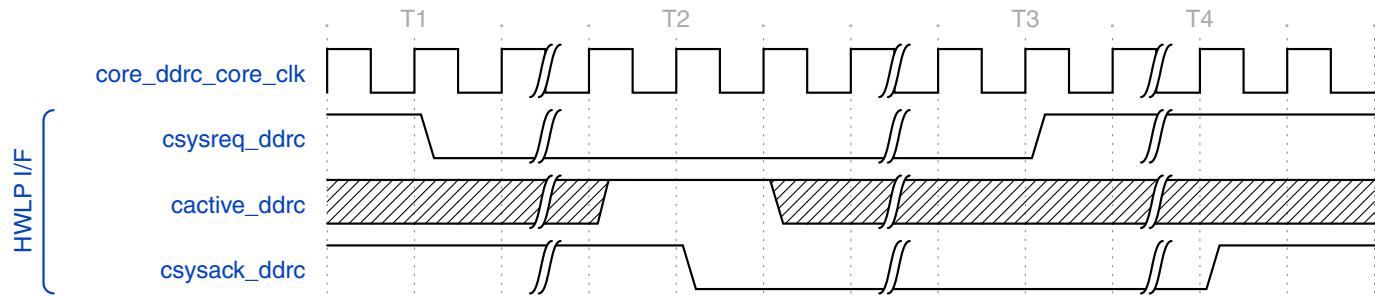
Note Once `csysreq_ddrc` is set to '0', `csysreq_ddrc` must be kept as '0' until `csysack_ddrc` becomes '0'.

An exit from self-refresh occurs as shown in [Figure 11-4](#). In this example, the system initiates the low-power state exit by asserting `csysreq_ddrc`.

Figure 11-4 Exit From Self-Refresh Mode Through Hardware Low-Power Interface of DDRC

A denial of low-power request occurs as shown in [Figure 11-5](#).

Figure 11-5 Denial of Hardware Low Power Request



At T1, an external low-power controller drives `csysreq_ddrc` to '0', requesting the DDRCTL to enter low-power state. At T2, the DDRCTL acknowledges the low-power request by driving `csysack_ddrc` to '0', but denies the request by setting `cactive_ddrc` to '1'. The low-power controller needs to recognize that the request is denied at T2 and the next clock cycle. In this case, the low-power controller must maintain the `core_ddrc_core_clk`, and must go through the low-power state exit sequence before it can initiate another low-power request. At T3, the low-power controller begins the low-power state exit sequence by driving `csysreq_ddrc` to '1'. At T4, the DDRCTL completes the exit sequence by driving `csysack_ddrc` to '1'.



`cactive_ddrc` can be set to '0' after T2 depending on the internal state of the DDRCTL, and the low-power controller cannot expect `cactive_ddrc` keeps '1'.

11.4.2.5 Example of Power Saving Modes of DDRC

[Table 11-2](#) shows an expected use case of the various power saving modes of the DDRC.

Table 11-2 Example of Power Saving Modes of the DDRC

Register or Hardware Low-Power I/F Action	Register Value	Behavior
PWRCTL.powerdown_en = 1 and PWRTMG.powerdown_to_x32 = X	Depending on the traffic profile, you can set the value of X.	<p>Puts SDRAM into power-down after X*32 cycles of idleness within DDRC.</p> <p>Note:</p> <p>This is Automatic Power-down.</p> <p>The DDRCTL puts the SDRAM into the power-down immediately, if PWRCTL.powerdown_en = 1 and PWRTMG.powerdown_to_x32 = 0.</p> <ul style="list-style-type: none"> ■ This assumes that there are no commands in progress or on the HIF. Power-down entry is based on idleness, and occurs only if the DDRCTL is idle. ■ There may be some delay before the actual power-down entry command is sent, as there may be other commands which need to be sent first (For example, precharge, refresh, ZQ, ctrlupd, phyupd). ■ The DFI low-power protocol may also need to be followed (if enabled).
PWRCTL.selfref_en = 1 and PWRTMG.selfref_to_x32 = Y	Value of Y must be greater than X. This ensures that power-down is entered prior to self-refresh.	<p>Puts SDRAM into self-refresh after Y*32 cycles of idleness within DDRC.</p> <p>This is Automatic Self-refresh.</p> <p>SDRAM moves from power-down to self-refresh. cactive_ddrc is not directly affected by Automatic Self-refresh.</p>
HWLPTMG0.hw_lp_idle_x32 = Z	Value of Z must be greater than Y. This ensures that system must already be in Automatic Self-refresh when cactive_ddrc = 0 occurs.	cactive_ddrc = 0 occurs after Z*32 cycles of idleness within DDRC.

Register or Hardware Low-Power I/F Action	Register Value	Behavior
External Hardware Low-Power controller sees cactive_ddrc going low and knows that a hardware low-power request on csysreq_ddrc/csysack_ddrc must be accepted (unless the DDRC stops being idle). Hardware Low-Power controller sends a DDRC hardware low-power entry request by driving csysreq_ddrc = 0 and waiting for csysack_ddrc = 0 and checking cactive_ddrc value at the time.	-	DDRC accepts hardware low-power entry request. This means that if DDRC stops being idle, this does not move the SDRAM out of self-refresh. Once a DDRC hardware low-power entry request is accepted, self-refresh can only be exited through a hardware low-power exit request.
If DDRC hardware low-power entry request is successful, the DDRC clock can be removed. If non-HIF only, this may not be the case. See “ Hardware Removal of Clocks from AXI Ports ” on page 278	-	-
If DDRC is receiving, or about to receive, a new command, due to asynchronous path from cactive_in_ddrc/hif_cmd_valid ^a of DDRC to cactive_ddrc, cactive_ddrc = 1 occurs, even if DDRC clock is removed. External Hardware Low-Power controller must observe this, re-start DDRC clock and perform DDRC hardware low-power exit request.	-	SDRAM exits self-refresh and performs RD/WR commands. idleness counters reset.

a. hif_cmd_valid refers to hif_cmd_valid if UMCTL2_DUAL_HIF=0, or hif_rcmd_valid || hif_wcmd_valid if UMCTL2_DUAL_HIF=1

11.4.3 AXI Low-Power Interface

The hardware low-power interface can be used to enter/exit a low-power state for the port. The input csysreq_n can trigger the entry into the low-power state. A hardware low-power entry trigger is accepted if the port is idle. Idle means there are no outstanding read or write commands from port n and there is no data accepted in the Write Data Queue. Otherwise, the request is rejected.

The outputs csysack_n and cactive_n provide feedback as required by the AXI low-power interface specification (this interface operation is defined by the AXI Specification). The port csysack_n acknowledges the request to go into a low-power state. The port csysack_n goes low for one cycle after csyseq_n goes low. The port cactive_n indicates whether the request is accepted or denied.

If the low-power request is accepted, the awready_n and arready_n signals are driven low and they stay low for one cycle, after cactive_n goes high.

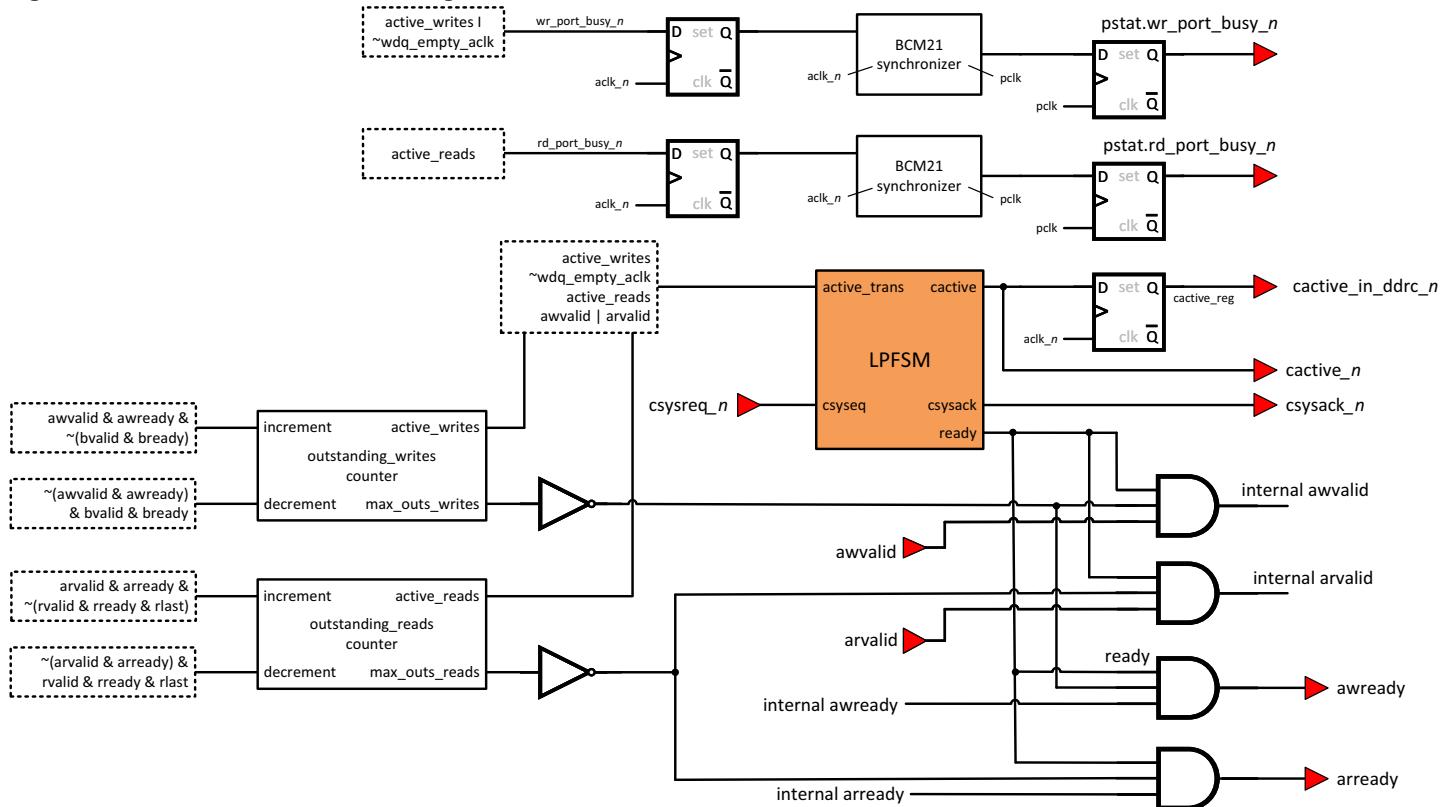
The cactive_n output can also be used by an external low-power controller to decide when to request a transition to the low-power state for a port. When cactive_n is low, it indicates that there are no outstanding read or write commands from port n, there is no data in the Write Data Queue, and the port is idle for at least UMCTL2_AXI_LOWPWR_NOPX_CNT*aclk_n cycles.

In the low-power state, the `cactive_n` output can be used by an external low-power controller to trigger a low-power state exit. Always, `cactive_n` is driven high when `arvalid_n` or `awvalid_n` is high, or if there are outstanding commands in the port. The paths from `arvalid_n`, `awvalid_n` and `wvalid_n` to `cactive_n` are asynchronous. Therefore, `cactive_n` asserts, even if the clocks are removed. The low-power controller must re-enable the clocks when `cactive_n` is driven high while in the low-power state.

The `cactive_n` signals are registered on `aclk_n` and input to the DDRC (`cactive_in_ddrc`).

[Figure 11-6](#) shows the block diagram of XPI LPFSM.

Figure 11-6 XPI LPFSM Block Diagram



The LPFSM has two different implementations depending on the setting of the parameter `UMCTL2_AXI_LOWPWR_NOPX_CNT`.

[Figure 11-7](#) shows the LPFSM implementation when `UMCTL2_AXI_LOWPWR_NOPX_CNT=0`.

Figure 11-7 LPFSM Implementation when `UMCTL2_AXI_LOWPWR_NOPX_CNT = 0`

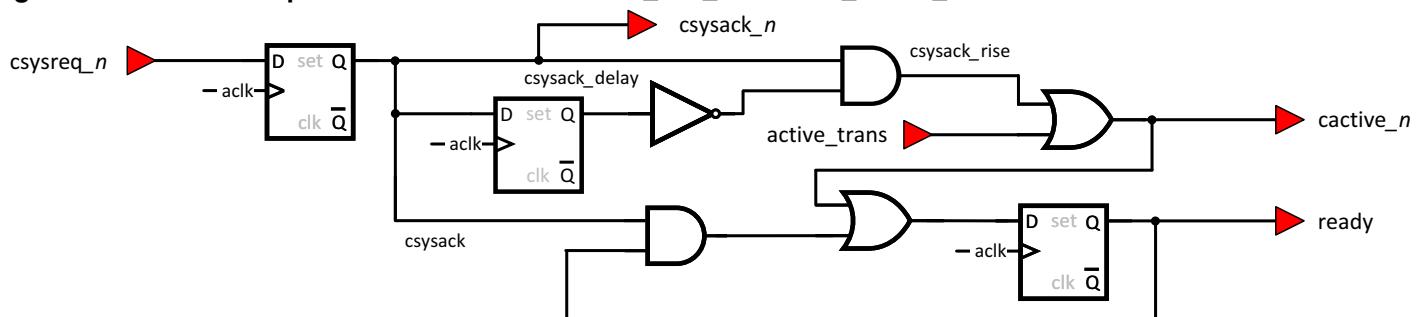
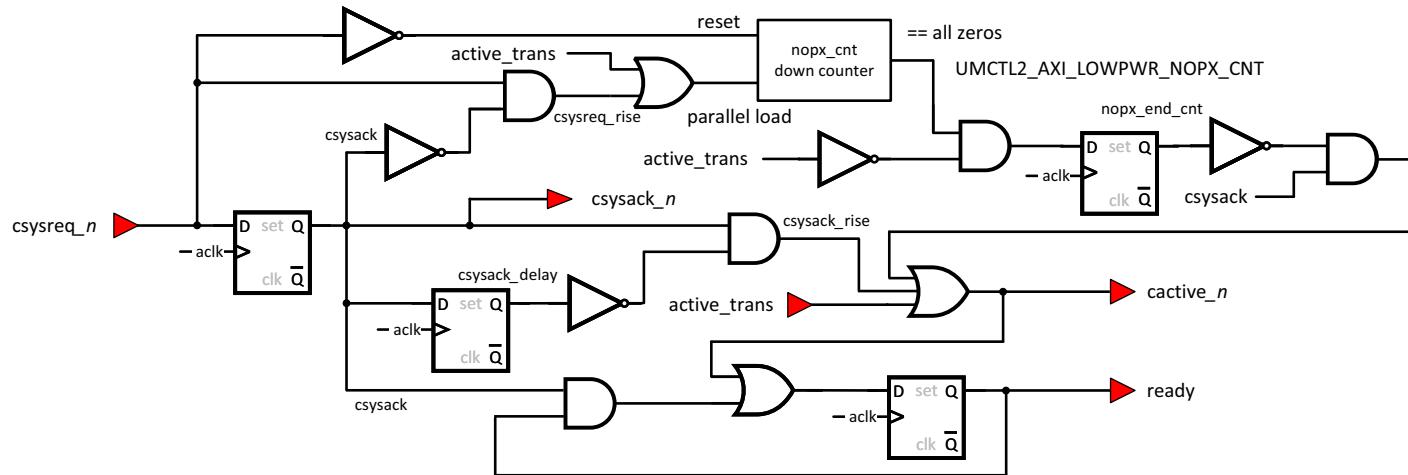


Figure 11-8 shows LPFSM implementation when UMCTL2_AXI_LOWPWR_NOPX_CNT > 0.

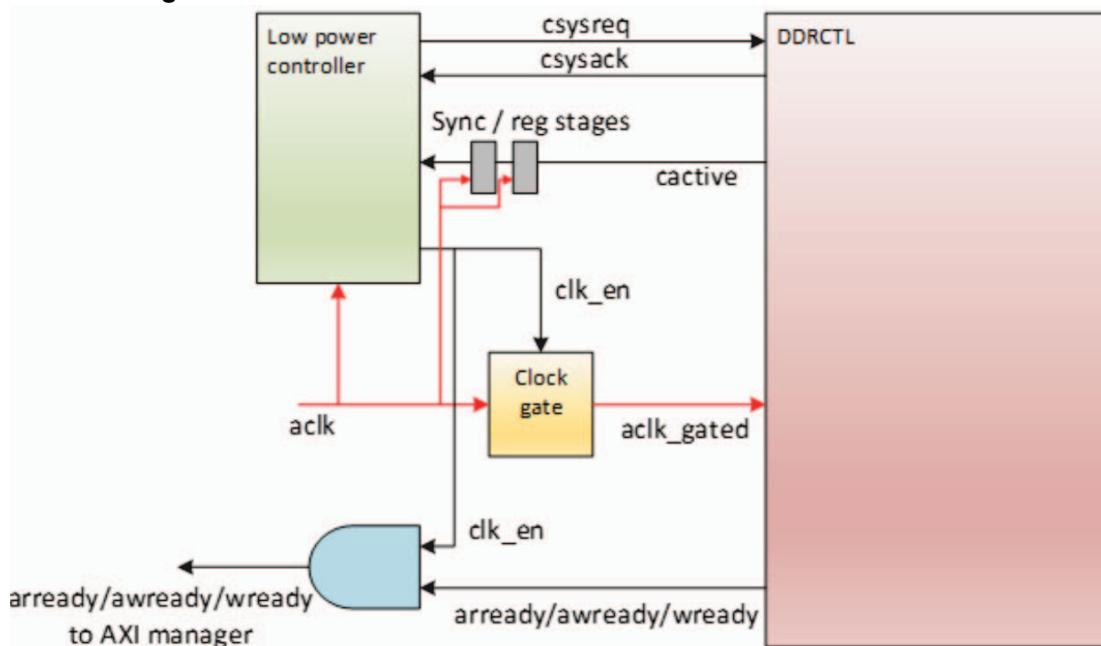
Figure 11-8 LPFSM Implementation when UMCTL2_AXI_LOWPWR_NOPX_CNT > 0



11.4.3.1 Guidance on AXI Clock Gating Using AXI Hardware Low Power Interface

The clock gating scheme shown in Figure 11-9 must be implemented for the following cases:

- `cactive_n` is asynchronous to `aclk_n` when hardware fast frequency change is enabled (`UMCTL2_HWFFC_EN==1`) and `aclk_n` is asynchronous to core clock (`UMCTL2_A_SYNC_n==0`). Hence, there is a need for synchronizers on the `cactive_n` output of the controller before being used in the clock gating logic.
- If there is a need of register stages on `cactive_n` when hardware fast frequency change is disabled (`UMCTL2_HWFFC_EN==0`), irrespective of `aclk_n` being synchronous or asynchronous to core clock (`UMCTL2_A_SYNC_n==0/1`).

Figure 11-9 Clock Gating Scheme

Not implementing the recommended clock gating scheme may lead to loss of AXI transactions during the clock gating sequence.



Figure 11-9 represents one AXI port. The same needs to be followed for all AXI ports.

11.4.3.2 Hardware Removal of Clocks from AXI Ports

There are two methods by which `aclk_n` can be removed:

- AXI hardware low-power interface handshaking



Dynamic and quasi dynamic registers can not be programmed if any of the clocks has been removed. Clocks must be turned back on before starting the programming sequence. Also, the clock gating logic must ensure there are no glitches on the clocks when they are removed/enabled.

11.4.3.3 Removal of AXI Clock through AXI Hardware Low-Power Interface Handshaking

Removal of clocks using the AXI hardware low-power interface is preferred, as this method places the ports in a low-power state. `arready_n` and `arready_n` signals are driven low once the low-power request is accepted.

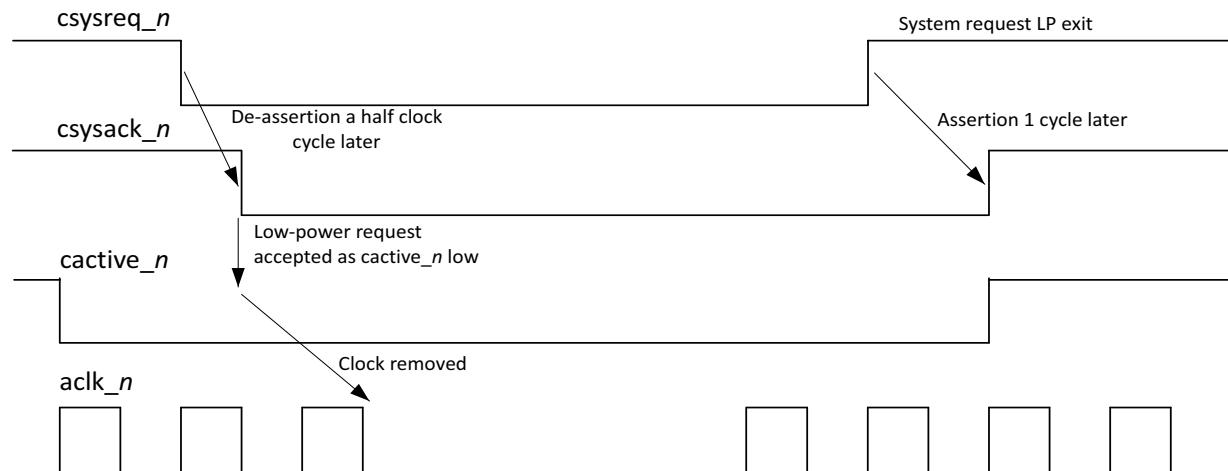
The low-power interface of all the AXI ports are independent. `aclk_n` for a port can be removed once the low-power requested has been accepted by the port.

If all the AXI ports are asynchronous (`UMCTL2_A_SYNC_n=0`), the DDRC clock can be removed once the DDRC has accepted the low-power request.

If any of the ports are synchronous (UMCTL2_A_SYNC_n=1), the DDRC clock can only be removed when `aclk_n` is removed for all synchronous ports. It must be re-enabled before any of the `aclk_n` for the synchronous ports are enabled.

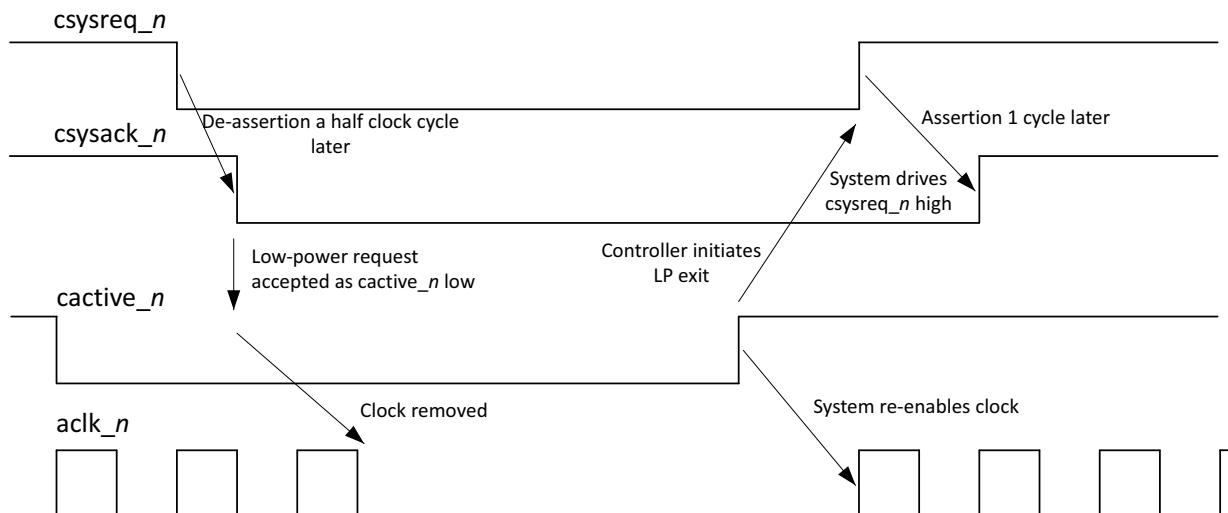
Clock removal using the AXI hardware low-power interface is shown in [Figure 11-10](#). In this example, the system initiates the low-power state exit though assertion of `csysreq_n`.

Figure 11-10 Clock Removal Using Hardware Low-Power Interface, Exit Triggered by Assertion of `csysreq_n`



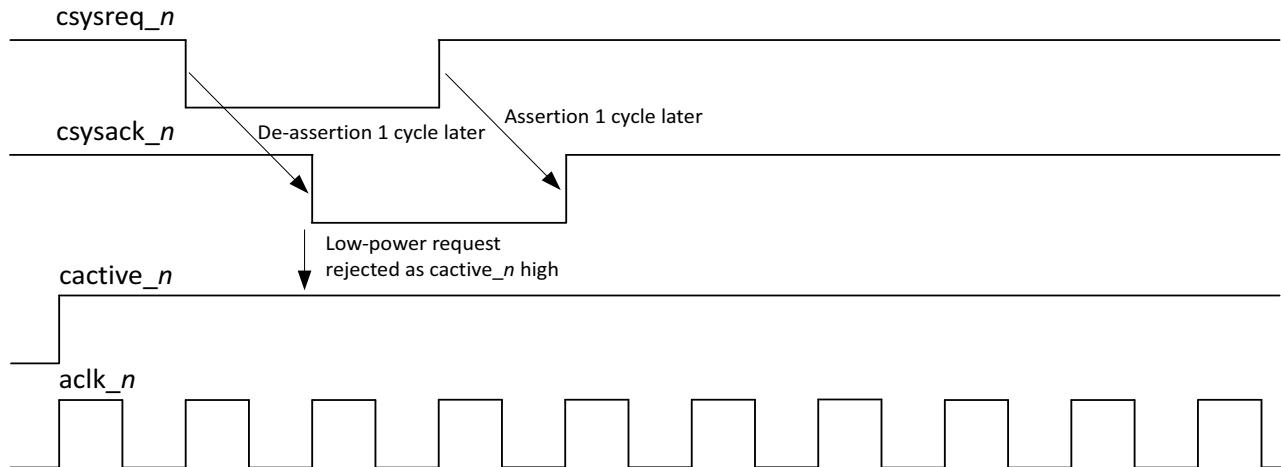
Clock removal using the AXI hardware low-power interface is shown in [Figure 11-11](#). In this example, the controller initiates the low-power state exit though assertion of `cactive_n`.

Figure 11-11 Clock Removal Using Hardware Low-Power Interface, Exit Triggered by Assertion of `cactive_n`



Rejection of the request is shown in [Figure 11-12](#). In this case, `aclk_n` cannot be removed.

Figure 11-12 Rejection of Low-Power State Request



With respect to interaction with “[DDRC Hardware Low-Power Interface](#)” on page [270](#), it is recommended to stagger a low-power entry of the DDRC until after all port's AXI low-power interface have successfully entered low-power. This is to allow the `cactive_n` to propagate to `cactive_ddrc_in` of DDRC hardware low-power interface.

11.5 Fast Frequency Change

The DDRCTL supports Fast Frequency Change, using up to four sets of timing registers. The alternative sets of timing registers can be found in REGB_FREQf_CHc registers. These registers may be written while the traffic is in progress using the first set of timing registers, thus reducing the software overhead at the time of frequency change.

The Fast Frequency Change sequence is described in the “Software Sequences” section of the “Programming” chapter.

11.6 Hardware Fast Frequency Change (HWFFC) Support



Note When switching from higher frequency to lower frequency, the controller may violate the JEDEC requirement that no more than 16 refreshes must be issued within $2 \cdot t_{REFI}$. These extra refreshes are not expected to cause problems in the SDRAM.

11.6.1 Hardware Fast Frequency Change (HWFFC) Overview

Hardware Fast Frequency Change (HWFFC) functionality in DDRCTL is intended to support clock frequency change procedure without the software intervention. HWFFC can support up to four sets of timing registers.

The following is a simplified HWFFC procedure that DDRCTL performs automatically:

1. Blocks all the AXI ports from taking further commands and data.
2. Drains commands from the controller and stops sending further transactions to SDRAMs.
3. Puts all the SDRAMs into self-refresh without powerdown.
4. Programs several mode registers (MR) in all the SDRAMs as necessary.
5. In LPDDR4 configurations, puts all the SDRAMs into Self-Refresh-Powerdown, if required.
6. Interacts with PHY by using DFI frequency change protocol.
7. In LPDDR4 configurations, exits all the SDRAMs from self-refresh powerdown, if required.
8. Programs several mode registers (MR) in all the SDRAMs as necessary.
9. Exits all the SDRAMs from self-refresh without powerdown.

During step (6), when `csysack_ddrc` becomes 0, clock frequency can be changed to a new target frequency. For more details, see “[LPDDR4 HWFFC Procedure When HWFFCCTL.hwffc_mode=0](#)” on page [287](#).

The HWFFC process can be managed by a hardware low-power controller (see section “[Hardware Low-Power Interfaces](#)” on page [270](#)) external to DDRCTL, using the DDRC Hardware Low-Power Interface. This interface is extended by the addition of five signals:

- `csysmode_ddrc`
- `csysdiscard_ddrc` (This input signal needs to be fixed to 0)
- `csysfrequency_ddrc`
- `csysfsp_ddrc` (this input signal is used only when `HWFFCCTL.hwffc_mode` is set to '1')
- `csystarget_frequency_ddrc` (this input signal exists only when `DDRCTL_HWFFC_EXT` is '1')

When the signal `csysreq_ddrc` is de-asserted, '1' on `csysmode_ddrc` indicates that a frequency change of the `core_ddrc_core_clk` is required. When `DDRCTL_HWFFC_EXT` is '0', the signal `csysfrequency_ddrc` indicates which set of DDRCTL timing register is used after the change of frequency. When `DDRCTL_HWFFC_EXT` is '1', the signal `csystarget_frequency_ddrc` indicates the same instead. These signals must be held constant while `csysreq_ddrc` is de-asserted.

When `HWFFCCTL.hwffc_en` is set to '0', even if the signal `csysreq_ddrc` is de-asserted and `csysmode_ddrc` is set to '1', then the DDRCTL does not initiate the HWFFC procedure.

Setting `HWFFCCTL.hwffc_mode` to '0' enables LPDDR4 support for HWFFC. Setting `HWFFCCTL.hwffc_mode` to '1' enables LPDDR5/4 support for HWFFC and allows DDRCTL to delegate

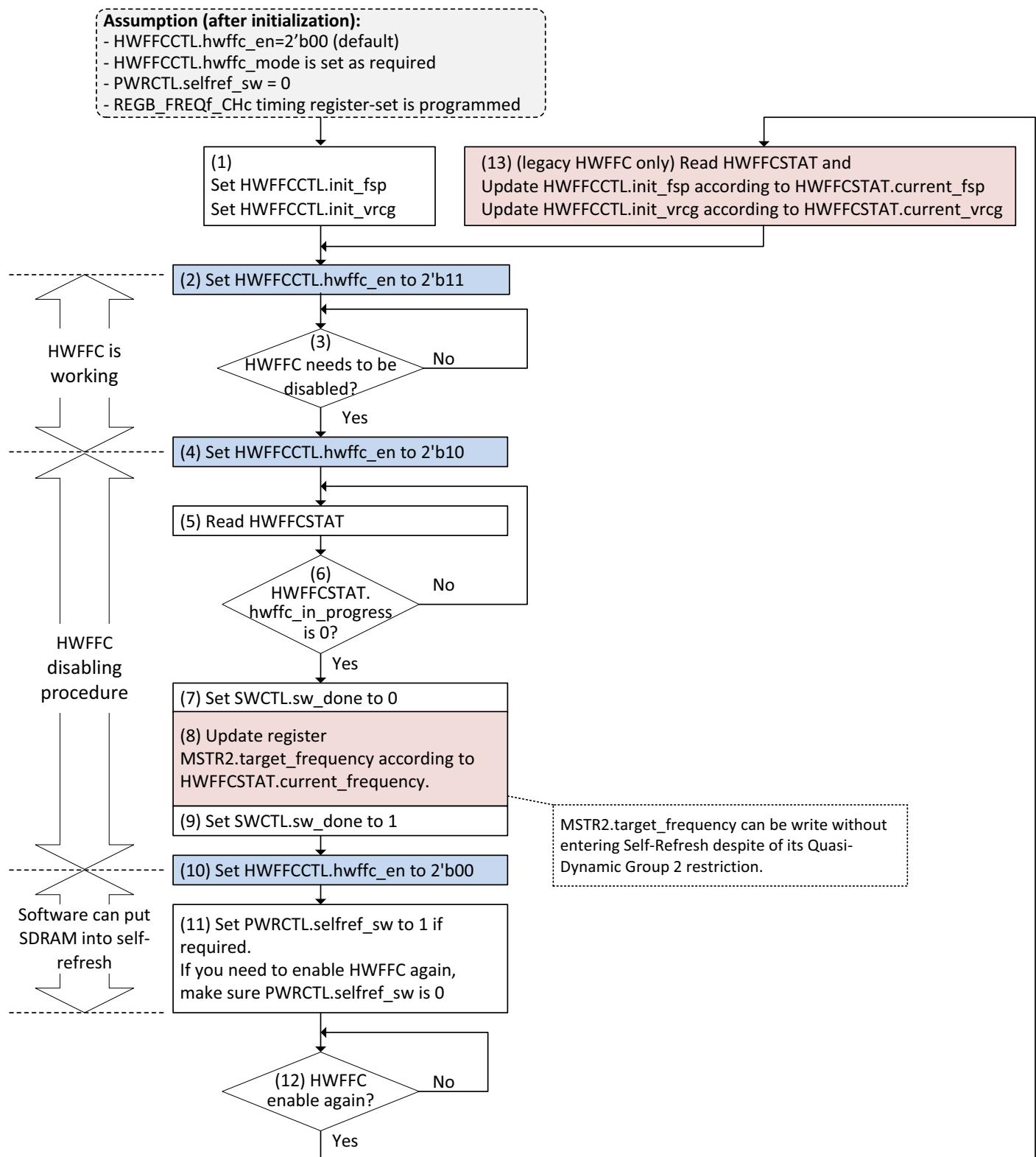
most of HWFFC procedure to Synopsys LPDDR5X/5/4X PHY. The PHY can be configured based on the value on `dfi_frequency`. With `csysfrequency_ddrc` you can request not only frequency change, but also other PHY features, such as entry into LP2 (PHY Fast Standby) mode.

If `HWFFCCTL.hwfffc_mode` is set to '1' and DDRCTL was in the automatic self-refresh with power-down state at the beginning of HWFFC procedure, the DDRCTL brings DRAM back to self-refresh without power-down state, then continues the procedure without going through the normal state, if possible.



- A hardware low-power entry request and a HWFFC request cannot be accepted simultaneously. The signal `csmode_ddrc` indicates which action to take place. When the signal `csmreq_ddrc` is de-asserted, '0' on `csmode_ddrc` indicates that a hardware low-power entry request is required and the DDRCTL attempts to enter the self-refresh operating mode. See the section "[DDRC Hardware Low-Power Interface](#)" on page [270](#). Clocks may then be removed but HWFFC is not executed and the signal `csmfrequency_ddrc` is ignored.
- Whenever you need to put SDRAM into self-refresh mode by software when `HWFFCCTL.hwfffc_en` is `2'b11`, then you have to follow the procedure as shown in [Figure 11-13](#) on page [284](#).
- When the software updates INITMRx, DRAMSETxTMGx, or other timing registers it is strongly recommended to disable the HWFFC in advance, because the software must know which register-set (`REGB_FREQf_CHC`) is currently being used, so that the appropriate registers can be updated correctly. This prevents unexpected behavior from happening. To disable HWFFC follow the procedural steps from 4 to 10, as shown in [Figure 11-13](#) on page [284](#).
- When the HWFFC is requested, the DDRCTL stops new read/write commands from being accepted. All existing commands in the DDRCTL are performed before entering self-refresh.
- All the AXI clocks and the DDRC clock must be changed at the same time when `UMCTL2_A_SYNC_n=1`. The external clock control logic must ensure there are no glitches on both the AXI and the DDRC clocks when clock frequency is changed.
- Synopsys expects that you use the `PHYINIT` utility to create the register writes needed to initialize the PHY and memory, including loading and executing the training firmware.

[Figure 11-13](#) on page [284](#) shows the software sequence for disabling/enabling HWFFC. Software needs to comply with the following sequence if there is a need to disable/enable HWFFC by writing `HWFFCCTL.hwfffc_en`. Software can put SDRAMs into self-refresh mode properly with the sequence.

Figure 11-13 Software Sequence for Disabling or Enabling of HWFFC



- Updating the HWFFCCTL.init_fsp in Step 1 and 13 applies only for LPDDR4.
- Write-DBI/read-DBI settings can be changed only before step 2 or after step 10 (that is, HWFFCCTL.hwfffc_en is '0').

11.6.2 Limitations of HWFFC

This functionality is intended to be used only with Synopsys LPDDR5X/5/4X PHY, and works under the following conditions:

- DFI frequency change (dfi_init_start=1) has priority over dfi_phymstr_req and dfi_phyupd_req.
The PHY must safely cancel the other requests when dfi_init_start is asserted.
- The PHY always acknowledges dfi_init_start (DFI frequency change request is never rejected).
- dfi_init_start can be asserted any time after de-assertion of dfi_cke.
DFI timing parameter tinit_start_min=1 can be assumed always.
- OCECC, REGPAR, OCCAP, OCSAP features are not supported.
- HWFFCCTL.hwfffc_mode must be set to '0' or '1' when core_ddrc_rstn is set to '0'. It cannot be changed after that.
- DERATECTL0.derate_mr4_pause_fc must be '0' while frequency change procedure is ongoing.

There are additional limitations, depending on HWFFC.hwfffc_mode:

- HWFFCCTL.hwfffc_mode=0:
 - Only LPDDR4 protocol is supported, per-rank ODT/Vref feature is not supported.
 - Dynamic frequency ratio change is not supported.
 - Data rate must be within the 1066 Mbps - 4266 Mbps range.
 - Software needs to use userInputAdvanced.DisableFspOp=1. For more details on this, refer to the PUB databook.
 - PPT2 feature is not supported.
 - Up to four operating frequencies (hardware parameter UMCTL2_FREQUENCY_NUM must be less than or equal to 4).
- HWFFCCTL.hwfffc_mode=1:
 - Only LPDDR4, LPDDR5, and LPDDR5X protocol is supported
 - Dynamic frequency ratio change support:
 - Frequency ratio aligns with TMGCFG.frequency_ratio in the REGB_FREQf_CH0 block, where f is determined by csysfrequency_ddrc.
 - The value of TMGCFG.frequency_ratio cannot be changed when core_ddrc_rstn is set to '1'. Either 1:1:4 or 1:1:2 frequency ratio is applicable for single target frequency.
 - MSTR4.wck_on must be equal to '0' (WCK on demand mode) if the hardware parameter MEMC_NUM_RANKS=1.
 - MSTR4.ws_off_en must be equal to '1' if MSTR4.wck_on=1 and the hardware parameter MEMC_NUM_RANKS must be greater than '1'.

- ❑ UMCTL2_FREQUENCY_NUM must be equal to 2, 3, 4, or 14.
- ❑ DDRCTL_HWFFC_EXT is '1' if UMCTL2_FREQUENCY_NUM is 14. Otherwise, DDRCTL_HWFFC_EXT is '0'.
- ❑ DDRCTL_HWFFC_EXT_AND_LPDDR5X is '1' if UMCTL2_FREQUENCY_NUM is '14' and MEMC_LPDDR5X is '1'. Otherwise, DDRCTL_HWFFC_EXT_AND_LPDDR5X is '0'.
- ❑ When DDRCTL_HWFFC_EXT_AND_LPDDR5X is '1', HWFFC only supports LPDDR5 and LPDDR5X protocols. Also, only MEMC_DRAM_DATA_WIDTH==16 is supported.
- ❑ Software needs to use userInputAdvanced.DisableFspOp=0 except for DDRCTL_HWFFC_EXT_AND_LPDDR5X==1 case. For DDRCTL_HWFFC_EXT_AND_LPDDR5X==1, set userInputAdvanced.DisableFspOp=1. For more details, see the PUB Databook.



Note DDRCTL_HWFFC_EXT=1 and DDRCTL_HWFFC_EXT_AND_LPDDR5X=1 are supported only with limited configuration. For more information, contact Synopsys.

- ❑ In LPDDR5, the ZQ Stop DRAM mode register (MR28[1]) must be equal to '0' before HWFFC entry. Do not modify it from software while HWFFCCTL.hwfffc_en is greater than '0' and HWFFCCTL.skip_zq_stop_start is equal to '0'.
- ❑ While HWFFCCTL.hwfffc_en is greater than '0', do not set OPCTRLCMD.zq_calib_short and OPCTRLCMD.zq_reset to '1'. When HWFFCCTL.hwfffc_en is greater than '0', the values of busy indicators OPCTRLSTAT.zq_calib_short_busy and OPCTRLSTAT.zq_reset_busy are not specified.
- ❑ PWRCTL.lppddr4_sr_allowed must be equal to '1'.
- ❑ Software initiated MRW/MRR via MRCTRL0.mr_wr=1 is not recommended when HWFFCCTL.hwfffc_en is greater than '0', and not allowed when HWFCSTAT.hwfffc_in_progress==1.

HWFFC operation may start changing frequency regardless of MRSTAT.mr_wr_busy so that MR access can occur before, during, or after changing frequency.

DDRCTL does not guarantee on when the software-driven MR access can occur, or to which FSP which MR will belong to.

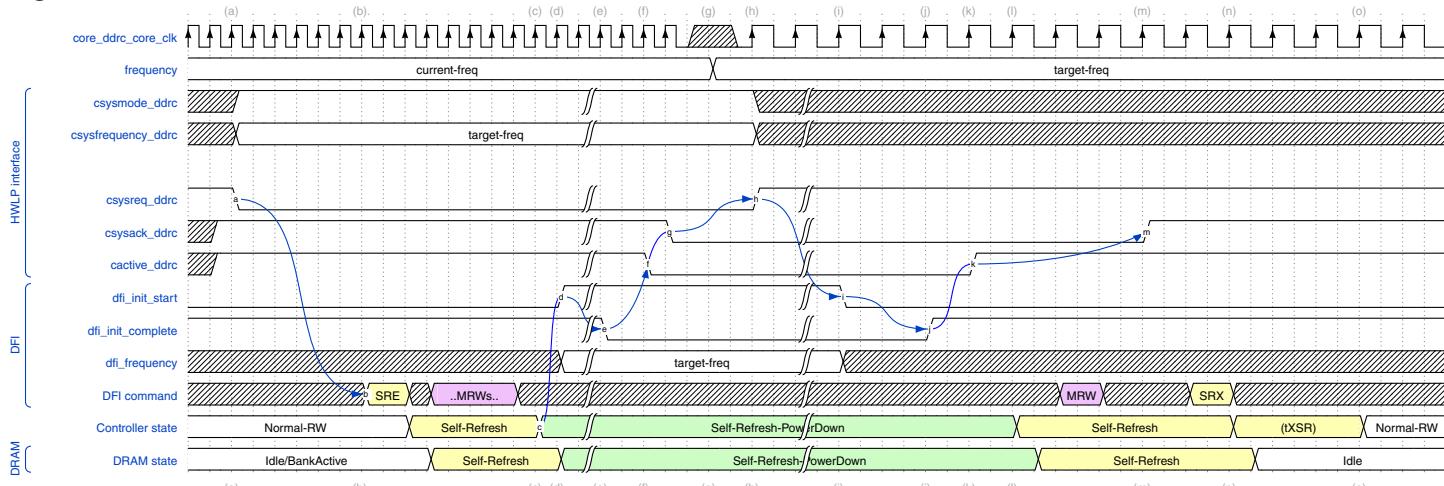
- ❑ LP3 entry request is not supported even though they are described in the PHY databook.
- ❑ PPT2 feature (DDRCTL_PPT2==1) is supported with additional limitations:
 - The hardware parameter DDRCTL_DFI_CTRLMSG=0.
 - The register PPT2CTRL0.reg_ddrc_ppt2_wait_ref=1 (default is '1').
 - Software can update the register DFIUPDTMG2.ppt2_en only when HWFFCCTL.hwfffc_en=0.
 - PPT2CTRL0.ppt2_burst can be set to '1' only when HWFFCCTL.hwfffc_en=0. HWFFCCTL.hwfffc_en must be '0' while PPT2STAT0.ppt2_burst_busy=1. See [Table 11-3](#) for more details.
 - PPT2 is applicable only for >=1600Mbps. If frequency is changed to <1600Mbps via HWFFC, Normal PPT2 request is suppressed even if it has been triggered and postponed over frequency change. See the row 'Trigger Normal PPT2 during HWFFC is ongoing' in [Table 11-3](#) for more details.

Table 11-3 Order of HWFFC and PPT2

Action	Details
Trigger HWFFC while Normal PPT2 is ongoing.	Allowed. HWFFC starts after PPT2 completes.
(Automatically) Trigger Normal PPT2 while HWFFC is ongoing.	Allowed. If Normal PPT2 is not scheduled until dfi_init_start/complete handshaking is initiated, it is postponed. Once handshake is completed, DDRCTL sends PPT2 only if REGB_FREQf_CH0.DFIUPDTIMG2.ppt2_en is '1' where f is HWFFCSTAT.current_frequency at that time.
Trigger HWFFC while Burst PPT2 is ongoing.	Not allowed. ppt2_burst_busy must be '0' before setting hwffc_en>0.
Trigger Burst PPT2 while HWFFC is ongoing.	Not allowed. hwffc_en must be '0' before triggering Burst PPT2.

11.6.3 LPDDR4 HWFFC Procedure When HWFFCCTL.hwffc_mode=0

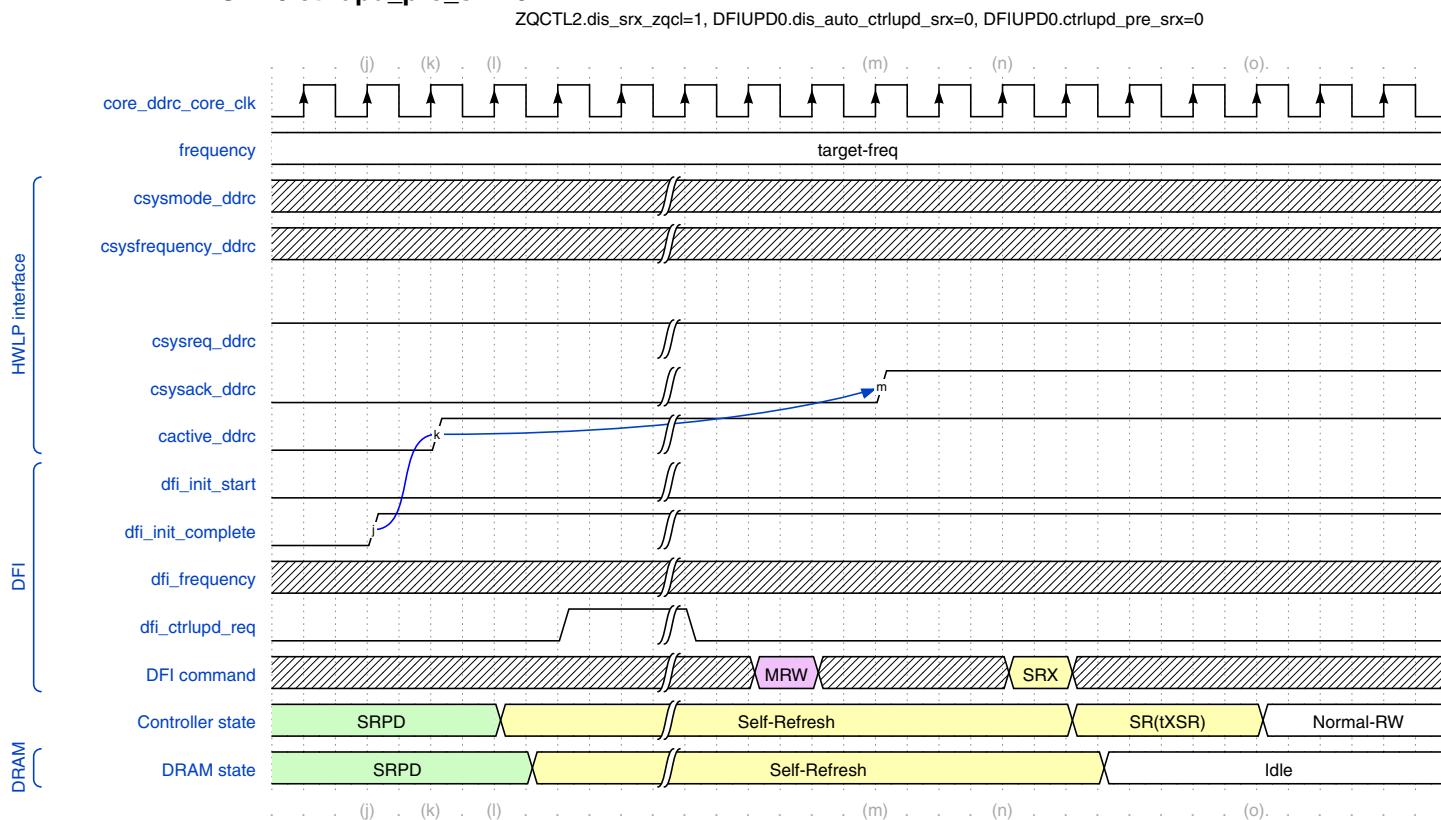
Figure 11-14 on page 287 shows the fundamental timing diagram of LPDDR4 HWFFC procedure.

Figure 11-14 LPDDR4 HWFFC Procedure

- The system requests frequency change by de-asserting `csysreq_ddrc`. `csysmode_ddrc=1`, `csysfrequency_ddrc` and `csysdiscamdrain_ddrc` must remain at constant values while `csysreq_ddrc=0`. `csysreq_ddrc` must be held de-asserted until `csysack_ddrc` is de-asserted.
- The DDRCTL issues SRE to put SDRAMs into self-refresh (without power-down), and then sends several MRW commands to update timing parameters for opposite side of current FSP, and lastly it switches FSP-OP to the opposite side. For more details, see “[MRW Commands Before Frequency Change](#)” on page 291.
- The DDRCTL de-asserts `dfi_cke` to put SDRAMs into Self-Refresh-Powerdown.
- The DDRCTL requests frequency change to the PHY by asserting `dfi_init_start`. `dfi_frequency` is set to the frequency value indicated by `csysfrequency_ddrc` (and holds it during `dfi_init_start=1`).
- The PHY responds to the frequency change request by de-asserting `dfi_init_complete`.
- The DDRCTL de-asserts `cactive_ddrc` and then de-asserts `csysack_ddrc` (frequency change request is accepted).

- g. The system changes clock frequency.
- h. The system asserts `csysreq_ddrc` (stable clock is required here).
- i. The DDRCTL de-asserts `dfl_init_start`.
- j. The PHY acknowledges `dfl_init_start` by asserting `dfl_init_complete`.
- k. The DDRCTL asserts `cactive_ddrc`.
- l. The DDRCTL asserts `dfl_cke` to exit Self-Refresh-Powerdown, and sends an MRW command to program MR13 .VRCG=0 if required. For more details, see “[MRW Command After the Frequency Change](#)” on page 291.
- m. The DDRCTL asserts `csysack_ddrc`. `cactive_ddrc` starts to behave again as described in other sections.
- n. The DDRCTL issues SRX to exit self-refresh.
- o. The DDRCTL starts to issue commands again after tXSR time.

Figure 11-15 Variation After Frequency Change When ZQCTL2.dis_srx_zqcl=1, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=0



Note `dfl_ctrlupd_req` may assert also before SPRD exit at (l) if the periodic ctrlupd interval is expired during HWFFC procedure.

Figure 11-16 Variation After Frequency Change When ZQCTL2.dis_srx_zqcl=1, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=1

ZQCTL2.dis_srx_zqcl=1, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=1

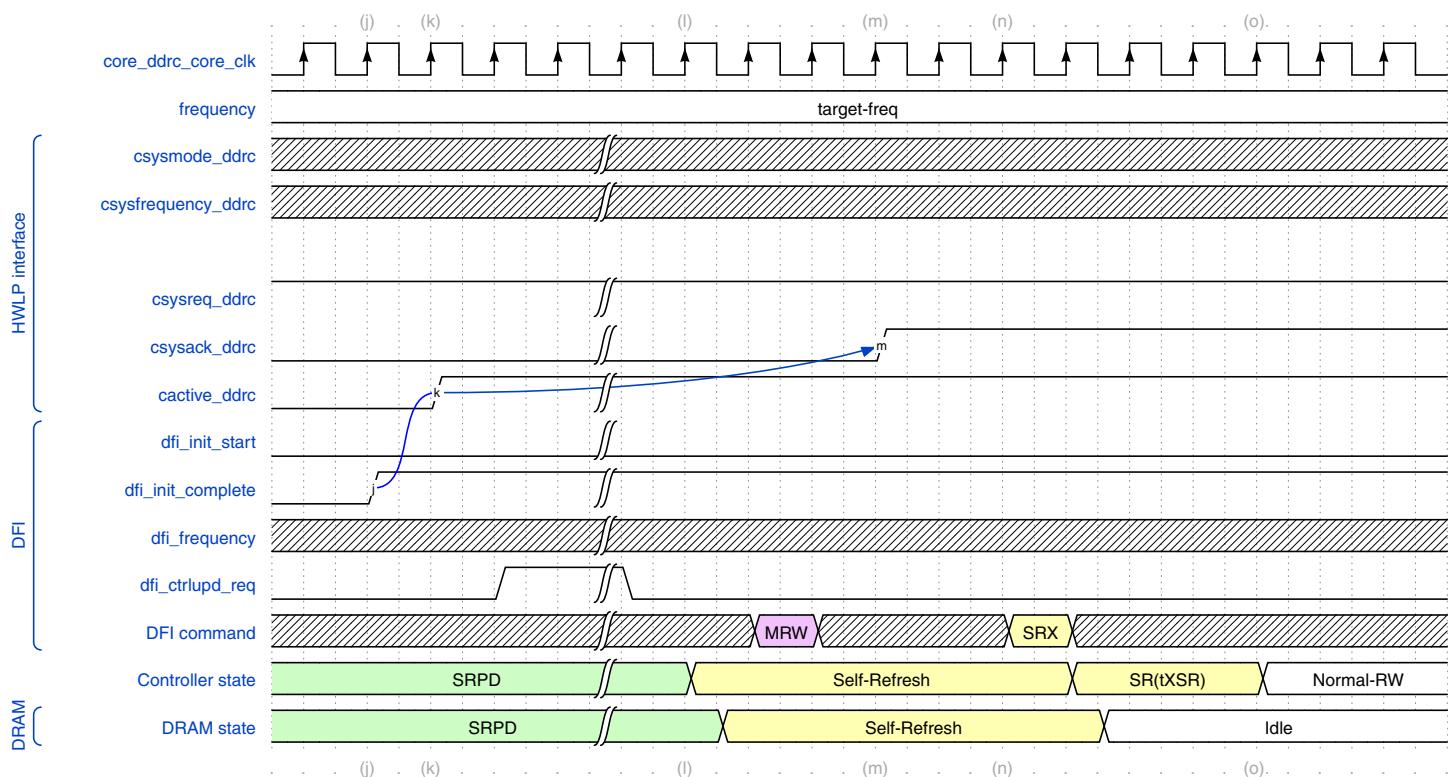


Figure 11-17 Variation After Frequency Change When ZQCTL2.dis_srx_zqcl=0, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=0

ZQCTL2.dis_srx_zqcl=0, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=0

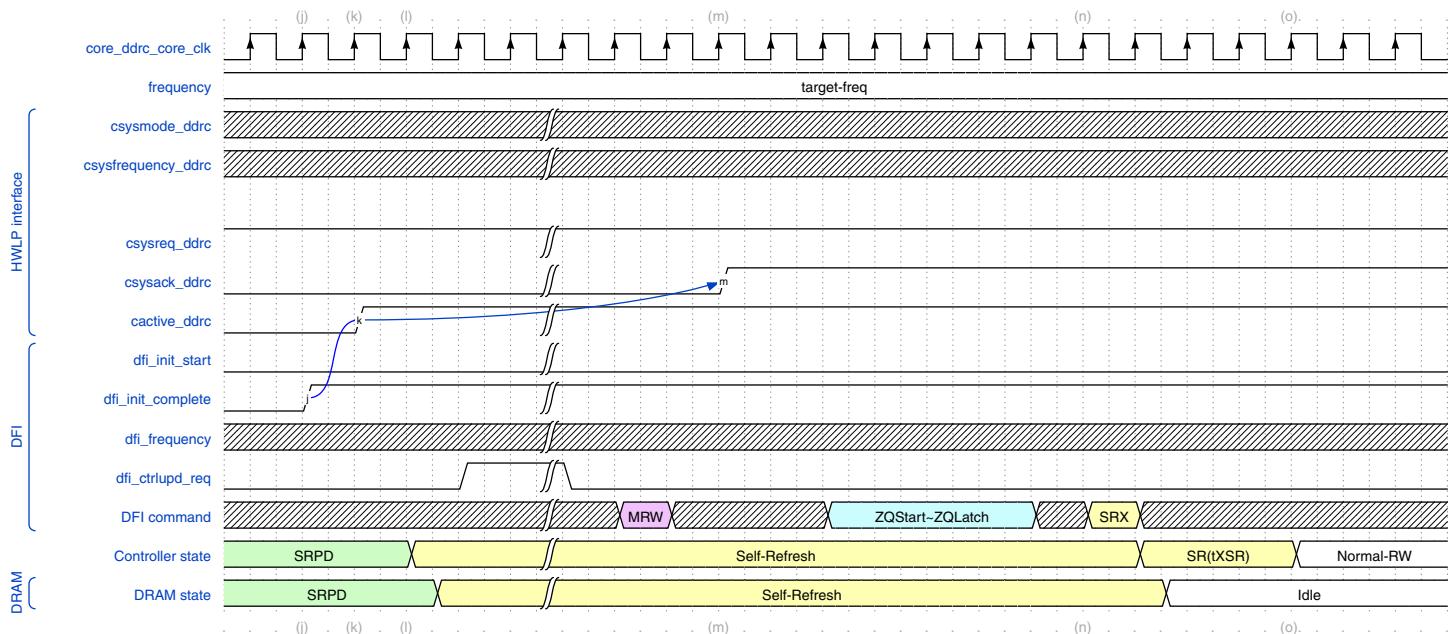
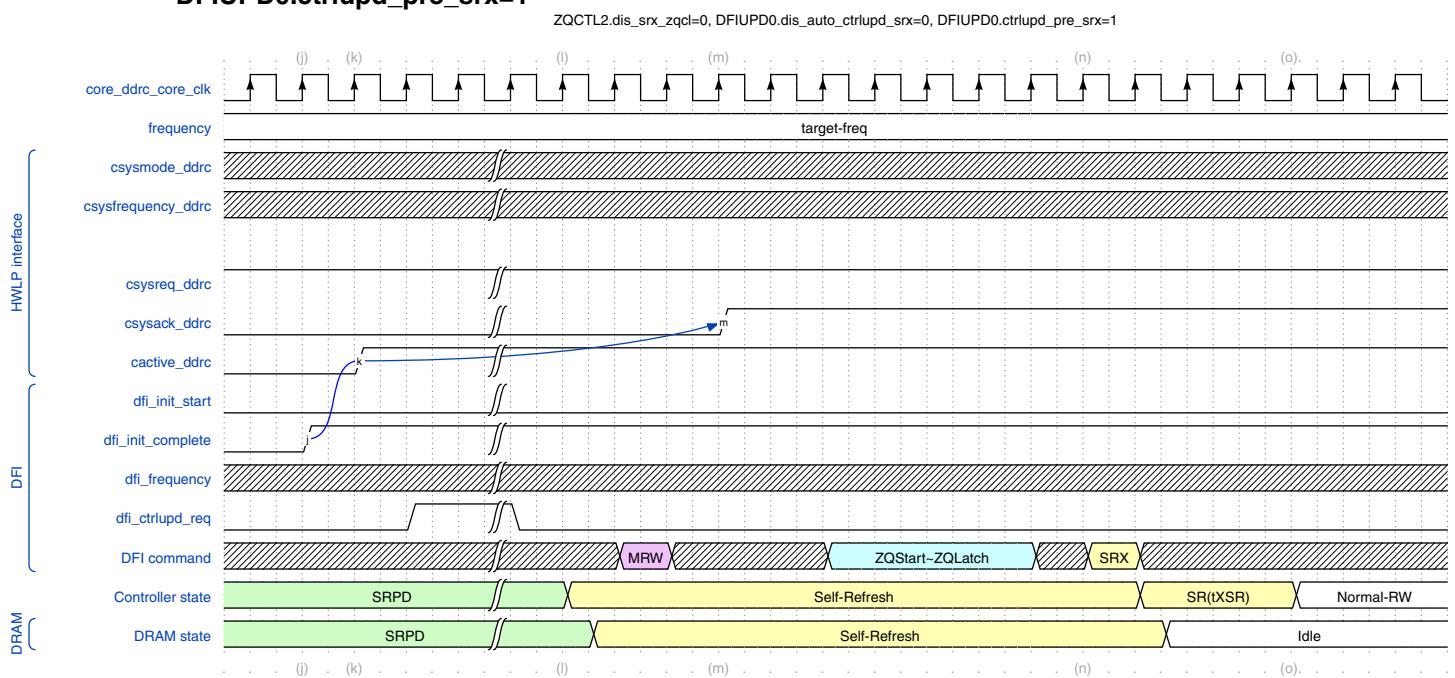


Figure 11-18 Variation After Frequency Change When ZQCTL2.dis_srx_zqcl=0, DFIUPD0.dis_auto_ctrlupd_srx=0, DFIUPD0.ctrlupd_pre_srx=1



Automatic MR Programming for LPDDR4 SDRAM

Frequency-Set-Point in LPDDR4 SDRAM

LPDDR4 SDRAM has two sets of frequency-set-point registers (FSP), FSP-OP[0] and FSP-OP[1], and they are controlled by MR13 in SDRAM as follows:

- FSP-WR(MR13 OP[6]) determines which frequency-set-point registers are accessed with MRW commands
- FSP-OP(MR13 OP[7]) determines which frequency-set-point register values are currently used to specify device operation

HWFFC functionality automatically programs MR13 as well as other MR registers (that is, MR1, MR2, MR3, MR11, MR12, MR14, MR22) during step (b) of [Figure 11-14 on page 287](#).

If FSP-OP indicates FSP-OP[1] then the DDRCTL issues an MRW command to MR13 so that FSP-OP becomes FSP-OP[0], or vice-versa. Every time HWFFC is performed, FSP-OP in SDRAM is toggled alternately, so the DDRCTL needs to track which frequency-set-point is currently being used in SDRAM.

The DDRCTL uses HWFFCCTL.init_fsp value as an initial value when the first HWFFC is performed to determine which FSP is currently being used in SDRAM.



FSP-OP[0] is chosen in default in LPDDR4 SDRAM, but if MR13 is issued and FSP-OP is changed during SDRAM initialization by PHY or controller, HWFFCCTL.init_fsp has to be set accordingly.

When HWFFCCTL.hwffc_mode=0, HWFFC implementation does not use dfi0_freq_fsp/dfi1_freq_fsp, therefore setting DFIMISC.dfi_freq_fsp is not required. To ignore those signals in Synopsys LPDDR5/4/4X PHY, software setting userInputAdvanced.DisableFspOp=1 must be used. For more details, refer to the PUB databook.

11.6.3.1 MRW Commands Before Frequency Change

There are a few options on how to program LPDDR4 SDRAM as shown in the [Table 11-4](#) on page [291](#). MR13 is programmed twice before frequency change. The first one is to set FSP-WR, and the second one is to set VRCG.

Table 11-4 Programming LPDDR4 SDRAM

Step	MRW	MR Field	Corresponding Register Fields to be Referred	Notes
1	MR13	OP[7] FSP-OP	HWFFCCTL.init_fsp	The same value is written to all ranks.
		OP[6] FSP-WR	~HWFFCCTL.init_fsp	
		OP[3] VRCG	HWFFCCTL.init_vrcg	
		Other fields	INITMR1.emr3	
2	MR1	-	INITMR0.mr	The same value is written to all ranks.
3	MR2	-	INITMR0.emr	The same value is written to all ranks.
4	MR3	-	INITMR1.emr2	The same value is written to all ranks.
5	MR11	-	INITMR2.mr4	The same value is written to all ranks.
6	MR12	-	INITMR2.mr5	The same value is written to all ranks.
7	MR14	-	INITMR3.mr6	The same value is written to all ranks.
8	MR22	-	INITMR3.mr22	Unlike LPDDR4 SDRAM, MR22 in LPDDR4X SDRAM defines whether or not ODT is enabled. HWFFC writes the same value (INITMR3.mr22) to MR22 for all ranks. Different ranks must have different ODT settings, so HWFFC cannot be used for multiple ranks system of LPDDR4X.
9	MR13	OP[7] FSP-OP	Inverted version of what was issued in the step 1	Corresponding wait time, which is programmed with DRAMSET1TMG17.t_vrcg_enable, is inserted to ensure tVRCG_ENABLE after this MRW is issued. That wait time is necessary if VRCG has been changed from '0' to '1' when FSP-OP is changed.
		OP[6] FSP-WR	Same as what was issued in the step 1	
		OP[3] VRCG	Always set to '1'	



If HWFFCCTL.skip_mrw_odtvref is set to '1', step 5-8 (MR11, MR12, MR14, MR22) is skipped.

11.6.3.2 MRW Command After the Frequency Change

MR13 is programmed in order to set VRCG to '0' if HWFFCCTL.target_vrcg is 0.

Table 11-5 Programming LPDDR4 SDRAM (Applicable only when HWFFCCTL.target_vrcg is 0)

Step	MRW	MR Field	Corresponding Register Fields to be Referred	Notes
10	MR13	OP[7] FSP-OP	Same as what was issued in step 9	Corresponding wait time, which is programmed with DRAMSET1TMG17.t_vrcg_disable, is inserted to ensure tVRCG_DISABLE after this MRW is issued.
		OP[6] FSP-WR	Same as what was issued in step 9	
		OP[3] VRCG	Always set to '0'	

11.6.3.3 Simplified Timing Diagrams Focusing on MR13 (VRCG)

Figure 11-19 to Figure 11-20 show how DDRCTL programs MR13.OP[3] VRCG according to HWFFCCTL register fields.

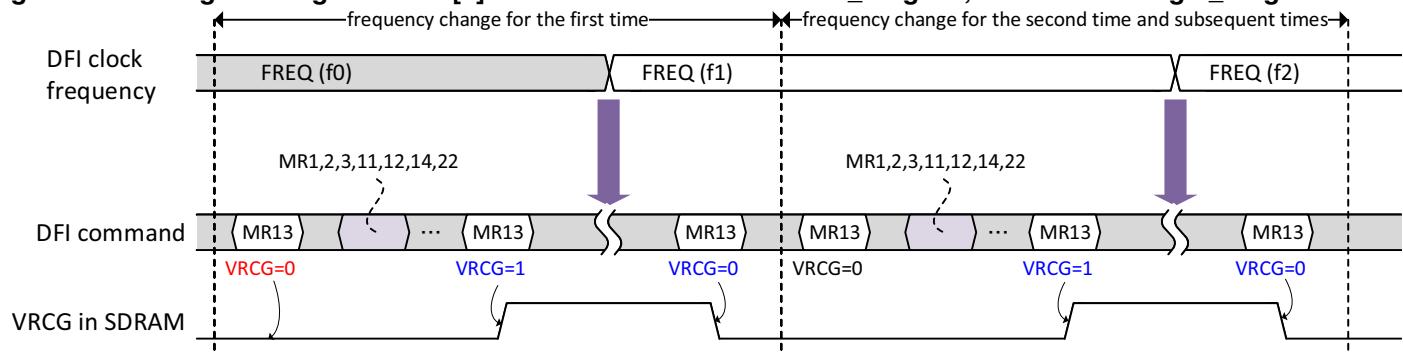
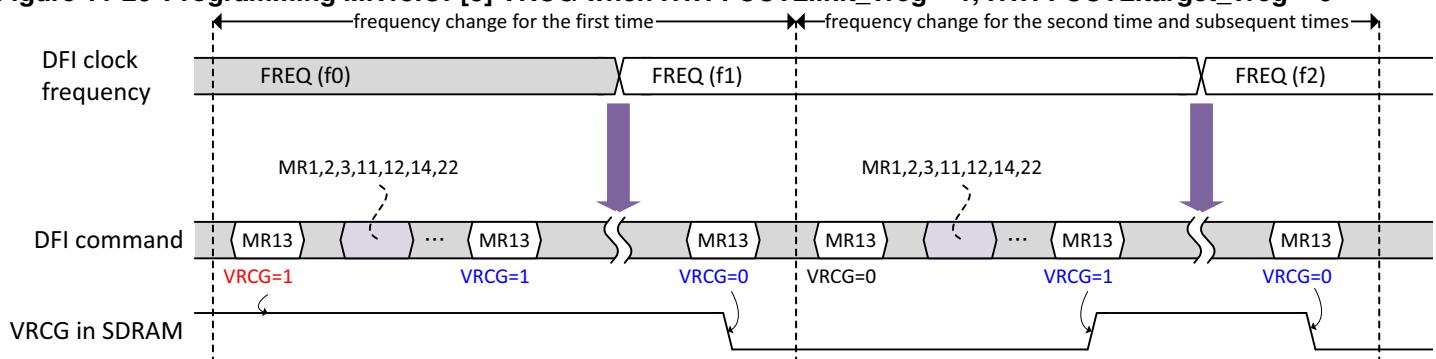
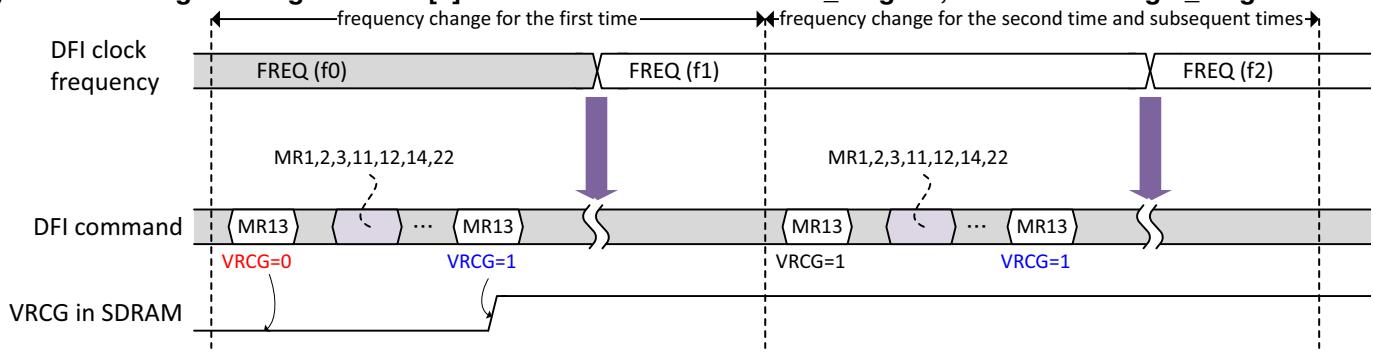
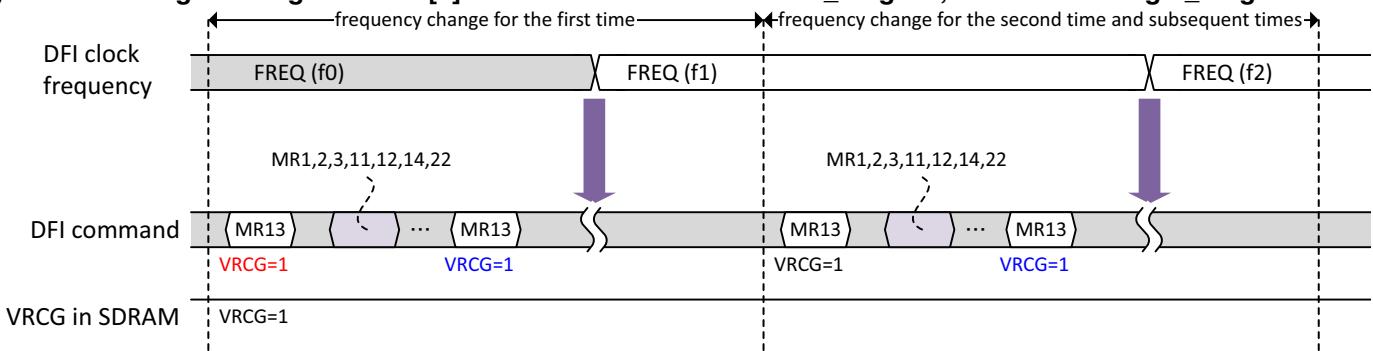
Figure 11-19 Programming MR13.OP[3] VRCG when HWFFCCTL.init_vrcg = 0, HWFFCCTL.target_vrcg = 0**Figure 11-20 Programming MR13.OP[3] VRCG when HWFFCCTL.init_vrcg = 1, HWFFCCTL.target_vrcg = 0**

Figure 11-21 Programming MR13.OP[3] VRCG when HWFFCCTL.init_vrcg = 0, HWFFCCTL.target_vrcg = 1**Figure 11-22 Programming MR13.OP[3] VRCG when HWFFCCTL.init_vrcg = 1, HWFFCCTL.target_vrcg = 1**

11.6.4 HWFFC Procedure When HWFFCCTL.hwffc_mode=1

In contrast to HWFFCCTL.hwffc_mode=0, HWFFC with HWFFCCTL.hwffc_mode=1 drives dfi_freq_fsp according to the system input from csysfsp_ddrc. For more details, refer to the PUB databook.

In this mode the same value input from csysfrequency_ddrc is output to dfi_frequency. When changing frequency and DDRCTL_HWFFC_EXT is '0', HWFFCSTAT.current_frequency is the same to LSB of dfi_frequency. When changing frequency but DDRCTL_HWFFC_EXT is '1', HWFFCSTAT.current_frequency is the same value to the one from added signal csystarget_frequency_ddrc.

Figure 11-23 and Figure 11-24 show the relationship between DDRCTL frequency set and PHY PState.

Figure 11-23 When DDRCTL_HWFFC_EXT = 0, dfi_frequency(==csysfrequency_ddrc) and dfi_freq_ratio Determines the Pair of DDRCTL Frequency Set and PHY PState

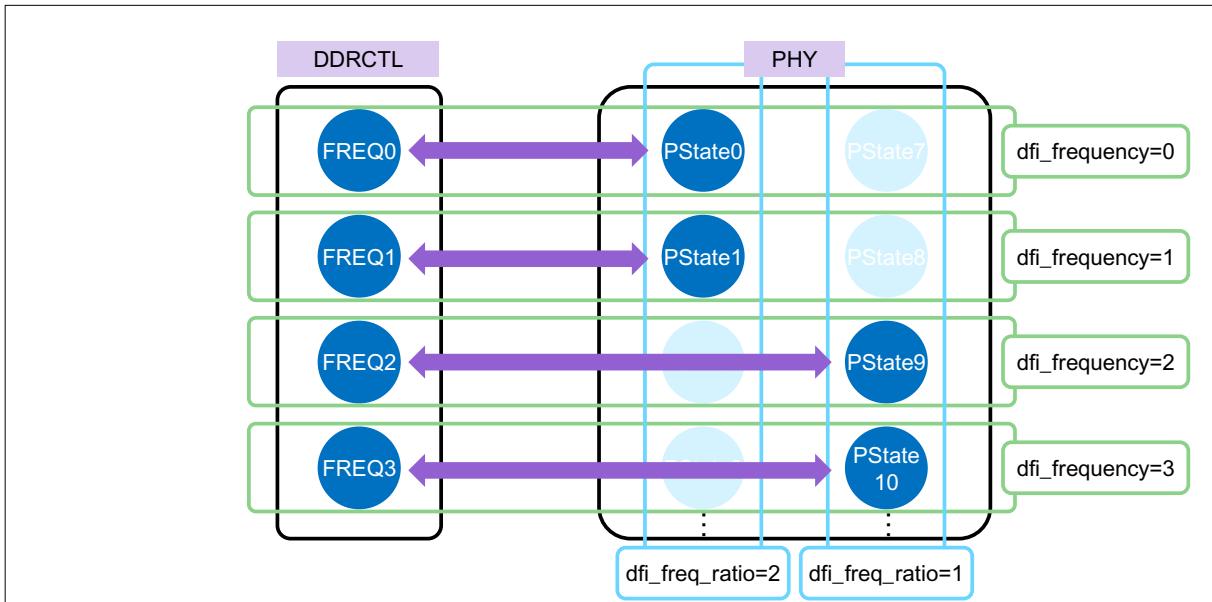


Figure 11-24 When DDRCTL_HWFFC_EXT = 1 and DDRCTL_HWFFC_EXT_AND_LPDDR5X = 0, dfi_frequency(==csysfrequency_ddrc), dfi_freq_ratio, and csystarget_frequency_ddrc Determines the Pair of DDRCTL Frequency Set and PHY PState

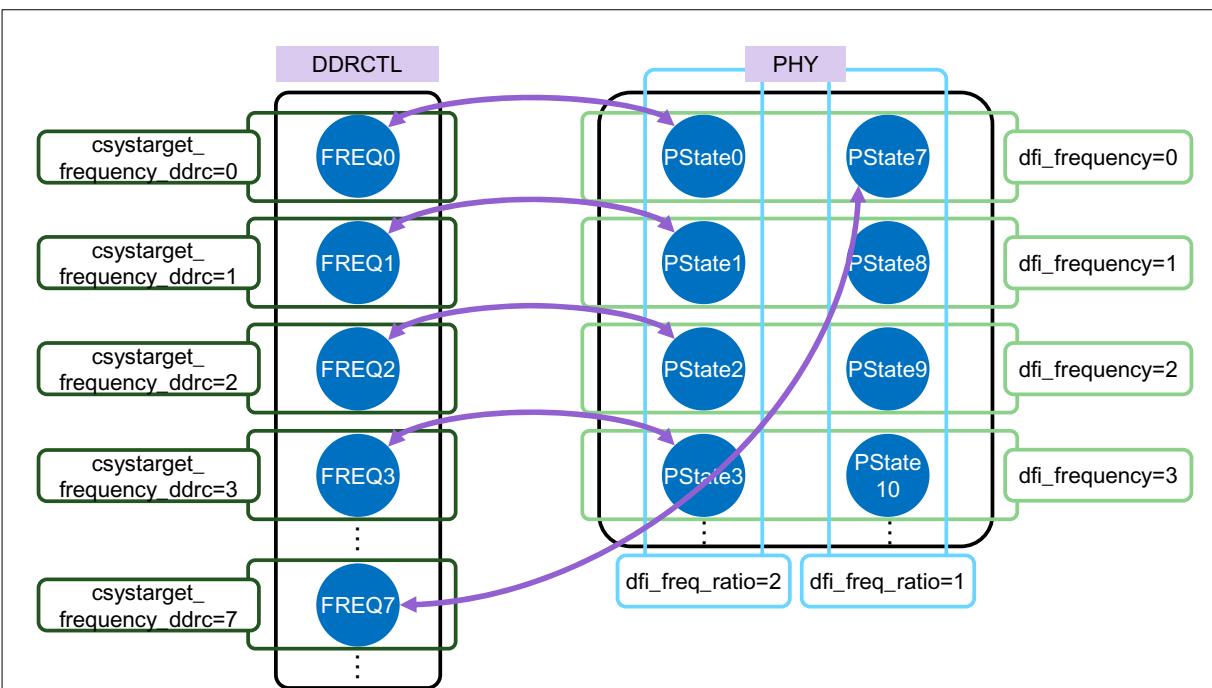


Figure 11-25 When DDRCTL_HWFFC_EXT = 1 and DDRCTL_HWFFC_EXT_AND_LPDDR5X = 1, dfi_frequency(==csysfrequency_ddrc), dfi_freq_ratio, and csystarget_frequency_ddrc Determines the DDRCTL Frequency Set and PHY PState

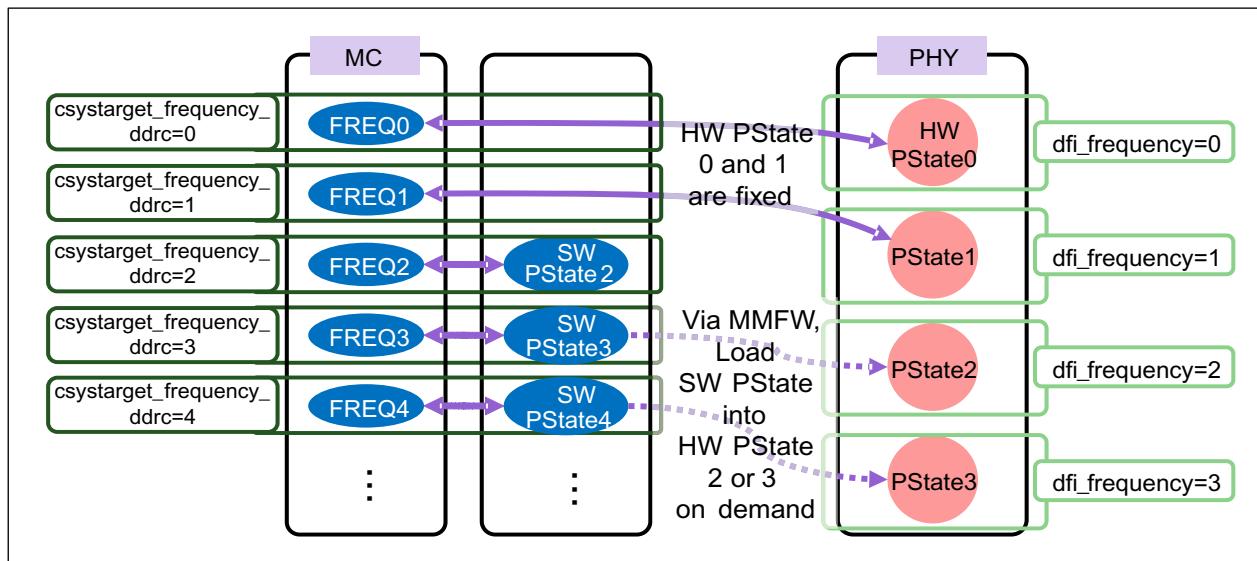


Figure 11-26 shows an example of how DDRCTL and PHY power state is selected.

Figure 11-26 Example: csys* Inputs and DFI Sideband Encoding to Select DDRCTL Frequency Set and PHY PState

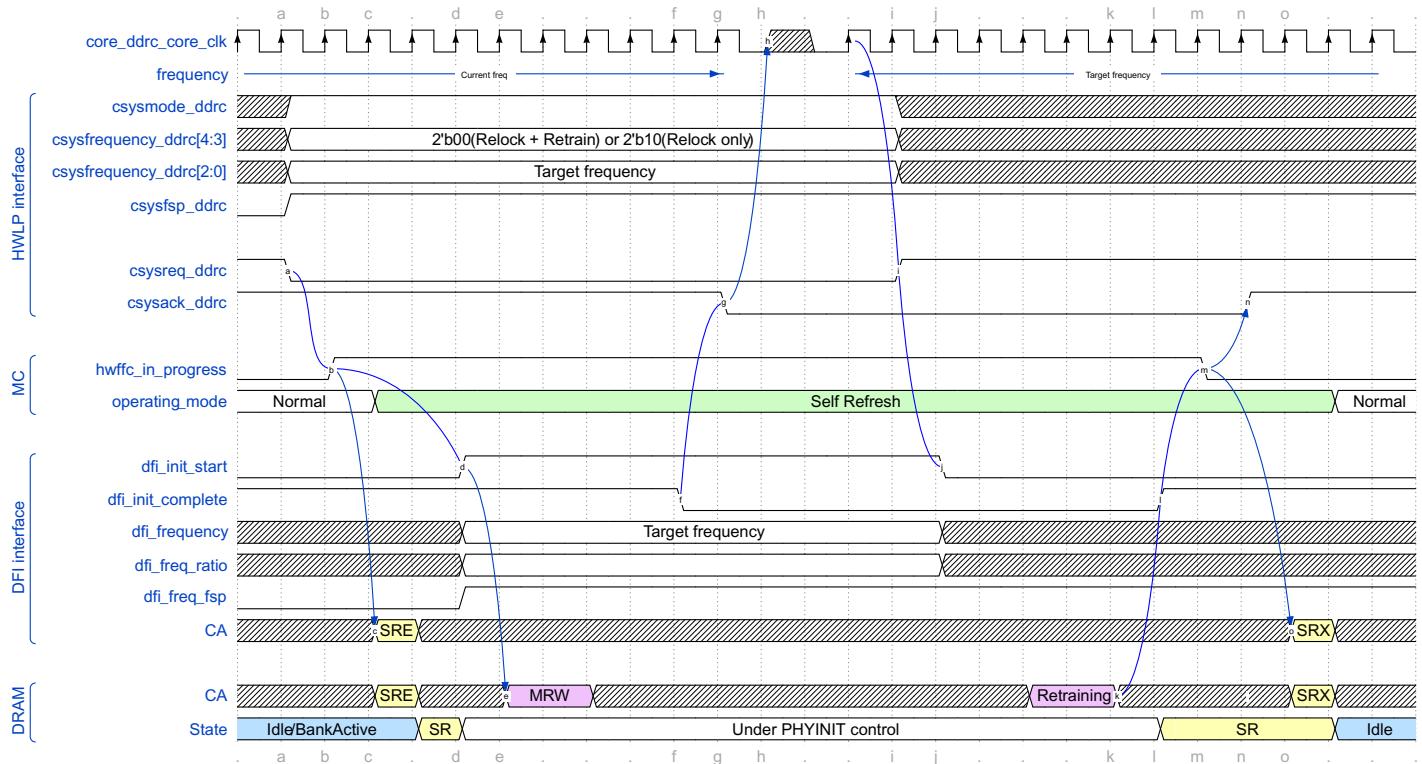
DDRCTL HW config			DDRCTL input	DDRCTL logic	DDRCTL output		
					PHY input	PHY logic	
UMCTL2_FREQUENCY_NUM		DDRCTL_HWFFC_EXT		HWFFCSTAT.current_frequency_ddrc driven by csysfrequency_ddrc or csystarget_frequency_ddrc		dfi_frequency dfi_freq_ratio driven by REGB_FREQf_CHO.TMCFG.frequency_ratio where f is HWFFCSTAT.current_frequency	
4	0	0	0	0	0	2	0
			1	1	1	2	1
			2	2	2	1	9
			3	3	3	1	10
14		1		0	0	2	0
			1	1	1	2	1
			2	2	2	2	2
			3	3	3	2	3
			4	4	4	2	4
			5	5	5	2	5
			6	6	6	2	6
			0	7	7	0	1
			1	8	8	1	8
			2	9	9	1	9
			3	10	10	1	10
			4	11	11	1	11
			5	12	12	1	12
			6	13	13	1	13
14		1		0	0	2 or 1	0
			1	1	1	2 or 1	1
			2	(*1)	(*1)	2 or 1	2 (*2)
			3	(*1)	(*1)	2 or 1	3 (*2)

*1: Software PState value; that is, from 2 to 13.

*2: Selected hardware PState is 2 or 3. Selected software PState is what MMFW has loaded in advance.

Figure 11-27 shows the fundamental timing diagram of the frequency change procedure.

Figure 11-27 Frequency Change Procedure With dfi_freq_fsp Flip



- System requests frequency change by de-asserting `csysreq_ddrc`. It must be held de-asserted until `csysack_ddrc` is de-asserted.

At the negative edge of `csysreq_ddrc`:

- `csysmode_ddrc` must be equal to '1'
- `csysfrequency_ddrc` and `csystarget_frequency_ddrc` to be desired value
- `csysfsp_ddrc` must be flipped to the opposite side of past DRAM FSP-OP value if selected PHY PState is changed from the past one. If not, `csysfsp_ddrc` must not be changed. See the PHY databook for more detailed FSP requirements.

Their values must remain constant while `csysreq_ddrc=0`

- HWFFCSTAT.hwffc_in_progress starts to indicate HWFFC is in progress
- The DDRCTL automatically closes all activated banks then issues SRE to put SDRAMs into self-refresh without power-down state
- The DDRCTL pauses periodic commands and stops WCK if needed. The controller then requests frequency change to the PHY by asserting `dfi_init_start`. It drives `dfi_frequency` and `dfi_freq_ratio` according to HWLP input signals, and also drives `dfi_freq_fsp` according to `TMGCFG.frequency_ratio`.
- PHY sends several MRW commands to update timing related parameters into MRs belongs to current FSP-WR.
- The PHY de-asserts `dfi_init_complete` to indicate DRAM is ready to change frequency
- The DDRCTL de-asserts `csysack_ddrc` to indicate frequency change request is accepted.

- h. The system changes clock frequency
- i. Once the clock becomes stable, the system asserts `csysreq_ddrc`
- j. The DDRCTL de-asserts `dfi_init_start`
- k. Between step 'e' and this step, the PHY flips FSP-OP to `dfi_freq_fsp` and FSP-WR to `!dfi_freq_fsp` to make them the opposite to their former values. The PHY also automatically sets and resets the VREF Fast Response mode. For more details, see the PHYINIT document
- l. The PHY acknowledges `dfi_init_start` by asserting `dfi_init_complete` once the DRAM and PHY are ready to complete the frequency change
- m. `HWFFCSTAT.hwfffc_in_progress` indicates that the HWFFC operation is complete
- n. The DDRCTL asserts `csysack_ddrc` to indicate that the HWFFC operation is complete
- o. Between step 'l' and this step, the DDRCTL issues `ctrlupd_req`, and ZQ calibration if required. The controller then issues SRX to exit self-refresh. The order of `dfi_ctrlupd_req` and ZQ calibration aligns with [Figure 11-15](#) on page 288 and [Figure 11-17](#) on page 289. The DDRCTL starts to issue other commands (for example, those requested from the AXI bus), again after tXSR period has expired.



Note `ctrlupd_req` at SRX after HWFFC is always sent after PDX regardless of `DFIUPD0.ctrlupd_pre_srx` because PDX is sent by PHY while DDRCTL does not have DRAM control.

Instead of `ZQCTL2.dis_srx_zqcl`, HWFFC with `HWFFCCTL.hwfffc_mode=1` can customize whether DDRCTL issues ZQ calibration before the SRX command, or not. If the un-operational period due to changing frequency needs to be shortest, set `ZQCTL2.dis_srx_zqcl_hwfffc` to '1' (default is '0').

The PHY automatically sets and resets the VREF Fast Response mode, therefore VRCG DRAM mode register (MR16[6]) must be '0' (normal operation) before entering HWFFC. Do not modify it from the software while `HWFFCCTL.hwfffc_en` is greater than '0'. The register `HWFFCSTAT.current_vrcg` is not used when `HWFFCCTL.hwfffc_mode` is equal to '1'.

To maximize performance, `dfi_freq_ratio` is recommended to be set at '2' (1:1:4 or 1:4) if datarate is greater than 3200Mbps, and '1' (1:1:2 or 1:2) if datarate is less than or equal to 3200Mbps.

11.6.4.1 Relock and Retrain

At step 'a' in [Figure 11-27](#), you can select whether PHY retrain runs in:

- Relock+Retrain: `csysfrequency_ddrc[4:2] = 3'b000`
- Relock-only: `csysfrequency_ddrc[4:2] = 3'b100`
- Fast Relock+Retrain: `csysfrequency_ddrc[4:2]=3'b001` (valid only when `DDRCTL_HWFFC_EXT_AND_LPDDR5X` is '1')

Relock+Retrain mode is recommended to change the frequency in generic case. Relock-only mode also can be applied if retraining is not required. For more details, see the PHY databook.

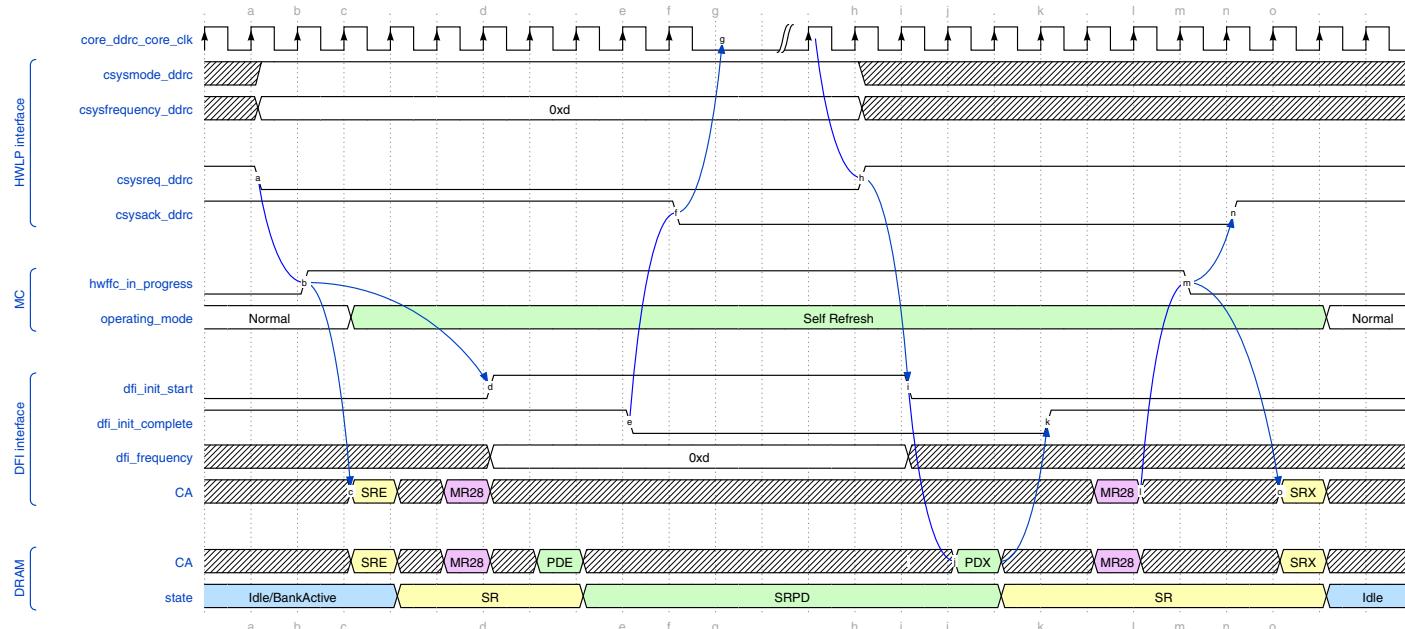
11.6.5 HWLP Driven Retraining

To initiate PHY retraining via HWLP interface, set HWFFCCTL.hwfffc_mode to '1' and specify csysfrequency_ddrc=0x0c at step 'a' in Figure 11-27. In this case frequency will not change and some steps in the HWFFC procedure will be skipped.

11.6.6 HWLP Driven LP2 (PHY Fast Standby)

Figure 11-28 shows the fundamental timing diagram of the LP2 entry and exit procedure. It is applicable only when HWFFCCTL.hwfffc_mode=1.

Figure 11-28 HWLP Driven LP2 Entry/Exit Sequence



1. The system requests LP2 entry by de-asserting csysreq_ddrc. csysreq_ddrc must be held de-asserted until csysack_ddrc is de-asserted.

At the negative edge of csysreq_ddrc:

- csysmode_ddrc must be set to '1'
- csysfrequency_ddrc must be set to '0xd'
- csysfsp_ddrc and csystarget_frequency_ddrc (if exists) must be the same value as their previous one, that is, previous FSP and previous target frequency.

Their values must remain constant while csysreq_ddrc=0

2. HWFFCSTAT.hwffc_in_progress starts to indicate that HWLP driven LP2 is in progress
3. The DDRCTL issues SRE to put the SDRAMs into self-refresh mode. After entering self-refresh, the DDRCTL pauses LPDDR DRAM background ZQ calibration by issuing MRW to MR28 with ZQ_stop=1 if necessary, according to HWFFCCTL.skip_zq_stop_start.
4. The DDRCTL requests the PHY LP2 entry by asserting dfi_init_start.
5. The PHY issues PDE to put the SDRAMs into self-refresh-powerdown, then de-asserts dfi_init_complete to indicate that the PHY is in LP2 state
6. The DDRCTL de-asserts csysack_ddrc to accept the HWLP driven LP2 entry

7. The system may stop the clock
8. Once the clock becomes stable, the system asserts `csysreq_ddrc` to request LP2 exit
9. The DDRCTL de-asserts `dfi_init_start`
10. The PHY issues PDX to exit self-refresh-powerdown
11. The PHY asserts `dfi_init_complete` once PHY and DRAM is ready
12. The DDRCTL requests restarting background ZQ calibration by issuing MRW to MR28 with `ZQ_stop=0` if necessary, according to `HWFFCCTL.skip_zq_stop_start`
13. `HWFFCSTAT.hwffc_in_progress` indicates the completion of HWLP driven LP2
14. The DDRCTL asserts `csysack_ddrc` to accept LP2 exit
15. The DDRCTL issues SRX to exit self-refresh. The controller starts issuing other commands again after the expiry of tXSR period

In this mode ZQ calibration is issued at SRX according to `ZQCTL2.dis_srx_zqc1` (not `ZQCTL2.dis_srx_zqc1_hwffc`). In contrast to frequency change, HWLP driven LP2 supposes long term idle e.g., several ms or seconds without calibration and its behavior is aligned with the general software driven SRPD.

11.6.7 DVFSC and DVFSQ Support with HWFFC

DVFSC (DVFS Core) and DVFSQ (DVFS VDDQ) are low power features defined in the LPDDR5 JEDEC standard spec. DDRCTL supports them with HWFFC to enhance the power saving feature. This feature is not applicable to LPDDR4 devices.

11.6.7.1 DVFSC Overview

The LPDDR5/5X SDRAM operates internal circuitry from either the VDD2H rail or the VDD2L rail while DVFSC or Enhanced DVFSC is enabled. When FSP is changed along with changing frequency, the LPDDR5/5X SDRAM might internally switch some internal circuits from one rail to the other. Unless explicitly stated, DVFSC refers to both DVFSC and Enhanced DVFSC in this section.

- MR19 OP[1:0] controls DVFSC enable/disable.
- DVFSC supports up to 1600 Mbps. Enhanced DVFSC supports up to 3200 Mbps.

11.6.7.2 DVFSQ Overview

LPDDR5 devices allows the VDDQ to be ramped during operation including read/write transactions. You need to determine exact speeds and levels with Synopsys PHY team according to the limits specified in the standard.

- MR19 OP[3:2] controls DVFSQ enable/disable.
- Support up to 3200 Mbps.
- During VDDQ ramp up and down, LPDDR5 operates with DVFSQ enabled condition. Refer to "LPDDR PHY databook".
- ZQ calibration can only be performed if DVFSQ is not active.
- ODT needs to be disabled while DVFSQ is enable.

11.6.7.3 Limitations

- DVFSC is only applicable to LPDDR5/5X.
- DVFSQ is only applicable to LPDDR5/5X.
- HWFFCCTL.hwffc_mode must be '1'.
- HWFFCCTL.skip_zq_stop_start does not affect DVFSQ switch. It only affects HWLP driven LP2 sequence.
- While DVFSCTL0.dvfsq_enable is '1' in current frequency set, ZQ Cal latch command would not be sent because Background ZQ Calibration is halted.
 - ZQCTL0.dis_auto_zq, ZQCTL2.dis_srx_zqc1 and ZQCTL2.dis_srx_zqc1_hwffc are equivalent to '1' when DVFSCTL0.dvfsq_enable is set to '1' even if they are set to '0'.
 - ZQCTL1.zq_reset and OPCTRLCMD.zq_calib_short must be set to '0'.
- DVFSQ switch requires HWFFC with PHY relock + retrain mode.
- PHYINIT DVFSC and DVFSQ enable flag for each PState must be programmed by System.
- PHYINIT ODT enable flag for each PState must be programmed by System.
- Dual VDD2 (MR13 OP[7]) is set to '0' - Dual VDD2 rail used (default).
- DVFSC and DVFSQ must be disabled (High voltage) at The PState which is selected first.
- DVFSCTL0.dvfsq_enable can be changed only at initialization.

11.6.7.4 DVFSC and DVFSQ Switching Sequence

DVFSC and DVFSQ are always enabled or disabled within single frequency set and can be automatically switched during HWFFC frequency change sequence as long as DDRCTL and PHY are properly configured at initialization. System need to do voltage ramp at switching sequence. System lowers or increases the voltage according to JEDEC, PHY and device specification.

[Table 11-6](#) shows the DVFSC and DVFSQ switch related steps.

Table 11-6 DVFSC and DVFSQ Switch Related Steps During HWFFC

Sequence	Before dfi_int_start/complete Handshake	During Handshake	After Handshake
DVFSC High-to-Low	System sets csysreq_ddrc=0	PHY sets MR19[FSP_WR][1:0]=1 (DVFSC=Low speed mode) PHY sets next FSP_OP to current FSP_WR so that DVFSC switch takes effect LPDRAM internally switches some internal circuits from VDD2H rail to VDD2L rail	System sets csysreq_ddrc=1

Sequence	Before dfi_int_start/complete Handshake	During Handshake	After Handshake
DVFSC Low-to-High	System sets csysreq_ddrc=0	PHY sets MR19[FSP_WR][1:0]=0 (DVFSC=High speed mode) PHY sets next FSP_OP to current FSP_WR so that DVFSC switch takes effect LPDRAM internally switches some internal circuits from VDD2L rail to VDD2H rail	System sets csysreq_ddrc=1
DVFSQ High-to-Low	System sets csysreq_ddrc=0 MC sets MR28[0]=1 (ZQ Reset=1) Wait for tZQRESET MC sets MR28[1]=1 (ZQ Stop=1) Wait for tZQSTOP	PHY sets MR19[FSP_WR][3:2]=1 (VDDQ=0.3V) PHY updates MR11, MR17, MR18, and MR41 to disable ODT PHY sets next FSP_OP to current FSP_WR so that DVFSQ switch takes effect	System sets csysreq_ddrc=1 System reduces VDDQ below 0.5V ^a
DVFSQ Low-to-High	System ramps VDDQ up to 0.5V ^{ab} System sets csysreq_ddrc=0 MC sets MR28[1]=0 (ZQ Stop=0) Wait for tZQCAL4=1.5 us ZQCAL Latch Wait for tZQLAT	PHY sets MR19[FSP_WR][3:2]=0 (VDDQ=0.5V) PHY updates MR11,17,18 and 41 to enable ODT PHY sets next FSP_OP to current FSP_WR so that DVFSQ switch takes effect	System sets csysreq_ddrc=1

a. Voltage ramp rate must follow the limitations according to JEDEC and PHY spec.

b. Voltage ramp must be completed 20 us before the dfi_init_start assertion. It is recommended to complete at least 18.5 us (=20us-tZQCAL4) before setting csysreq_ddrc=0. Refer to "LPDDR PHY databook" for details.

DDRCTL automatically writes MR28 to stop and reset background ZQ calibration before the dfi_init_start/complete handshake at DVFSQ switching, according to [Table 11-6](#).

DVFSCTL0.dvfsq_enable is used to indicate the frequency set where DVFSQ is enabled. If DDRCTL detects that DVFSQ is enabled, DVFSQ High-to-Low steps are added to the sequence. If DVFSQ is disabled, DVFSQ Low-to-High steps are added to the sequence. System needs to ramp VDDQ according to [Table 11-6](#).

For DVFSC, nothing special is done by DDRCTL. PHY takes necessary steps at DVFSC High-to-Low or DVFSC Low-to-High in [Table 11-6](#).

ZQ Cal latch command is usually sent according to dis_srx_zqc1_hwffc after the handshake and before DDRCTL goes to normal state, however, it would not occur right after DVFSQ High-to-Low and DVFSQ Low-to-High. In case of High-to-Low, ZQ Cal latch command is not needed because Background ZQ calibration is halted. In case of Low-to-High, ZQ Cal latch command is done right before dfi_init_start/complete handshaking.

11.6.8 Registers Related to HWFFC

The following are the registers related to HWFFC:

- ZQCTL2
- HWFFCCTL
- HWFFCSTAT
- ZQSET1TMG2
- DVFSCTL0

Table 11-7 shows how DDRCTL and PHY behaves according to the HWFFC registers and input signals.

Table 11-7 Valid Register Combinations, csysfrequency_ddrc Values, and Applicable Features for LPDDR 5X/5/4X PHY

HWFFCCTL.hwffc_mode	csysfrequency_ddrc in Binary	Feature to be Launched
0	000_xx	HWFFC followed by relock + retrain
0	100_xx	HWFFC followed by relock
1	000_xx	HWFFC followed by relock + retrain
1	011_00	HWLP driven retraining
1	001_00	HWFFC followed by fast relock + retrain
1	011_01	HWLP driven LP2
1	100_xx	HWFFC followed by relock
	other	Not allowed

Table 11-8 shows how these registers control ZQ calibration after HWFFC sequence.

Table 11-8 Relation Between HWFFCCTL, ZQCTL2 and ZQ Calibration

HWFFCCTL.hwffc_mod_e	ZQCTL2.dis_srx_zqcl_hwff_c	ZQCTL2.dis_srx_zqcl	ZQ calibration after HWFFC sequence
0	Don't care	0	Yes
0	Don't care	1	No
1	0	0	Yes
1	0	1	Yes if NOT HWLP driven LP2 was launched
1	1	0	Yes if HWLP driven LP2 was launched
1	1	1	No

11.6.9 HWFFC with PHY Mission Mode Firmware

Synopsys LPDDR5X/5/4X PHY (5XPHY) supports 14 PStates but in a different way compared to LPDDR5/4/4X PHY.

- 5XPHY has configuration registers for hardware PStates but only for four PStates. Frequency change requires Host to explicitly load software PState into hardware PState in advance, using PHY mission mode firmware (MMFW). See PHY databook for more requirements and limitations.
- All necessary DRAM mode registers (MR) need to be updated at changing frequency. It is done by DDRCTL instead of PHY because PHY does not automatically update MR's if PHY's CSR DisableFspOp is set to '1' and it is required from MMFW limitation. To support it, DDRCTL provides a dedicated mechanism to automatically send MRW, called MRW Buffer.
- While MMFW is in progress, DFI sideband signal must be inactive.
- Only LPDDR5 and LPDDR5X protocols are supported.

11.6.9.1 MRW Buffer

To support issuing vast number of per-PState and per-rank MRW commands, DDRCTL provides a programmable MRW mechanism called MRW Buffer. According to its programming, specified number of MRW is issued with specified address, data, rank, and interval, before changing frequency/PState.

11.6.9.2 Example Mode Registers to be Issued at Changing Frequency

Following is a list of example MRs that is issued to update DRAM operational settings and training settings when changing frequency:

- Operational settings: MR19, 18, 1, 2, 3, 10, 11, 17, 20, 22, 41, 58
MR17 is per-rank setting. Others are common between both ranks.
- Trained value settings: MR12, 14, 15, 24, 30, 69, 70, 71, 72, 73, 74
All MRs are per-channel and per-rank setting.
MR12 is also per-byte setting if channel consists of two x8 devices. MR12[7] is VBS field that selects which range of DQ, that is, DQ[0:7] or DQ[15:8] to be written. Both MRWs with MR12[7]=0 and MR12[7]=1 are needed for x8 device.

Assuming single channel die, total required number of MRW per PState is (MR19, 18, 1, 2, 3, 10, 11, 20, 22, 41, 58) + (MR17)*2rank + (MR14, 15, 24, 30, 69, 70, 71, 72, 73, 74)*2rank + (MR12)*2rank*2device = 37.

Contact Synopsys PHY support for exact MRs list.

11.6.9.3 MRW Buffer Constitution

MRW Buffer requires an external SRAM placed outside of DDRCTL. [Figure 11-29](#) shows the connection between SRAM and DDRCTL. To compensate the SRAM timing, you can insert some stage of registers between SRAM and DDRCTL. The hardware parameters DDRCTL_MRWBUFF_RD_LATENCY and DDRCTL_MRWBUFF_WR_LATENCY should follow the actual number of their stages.



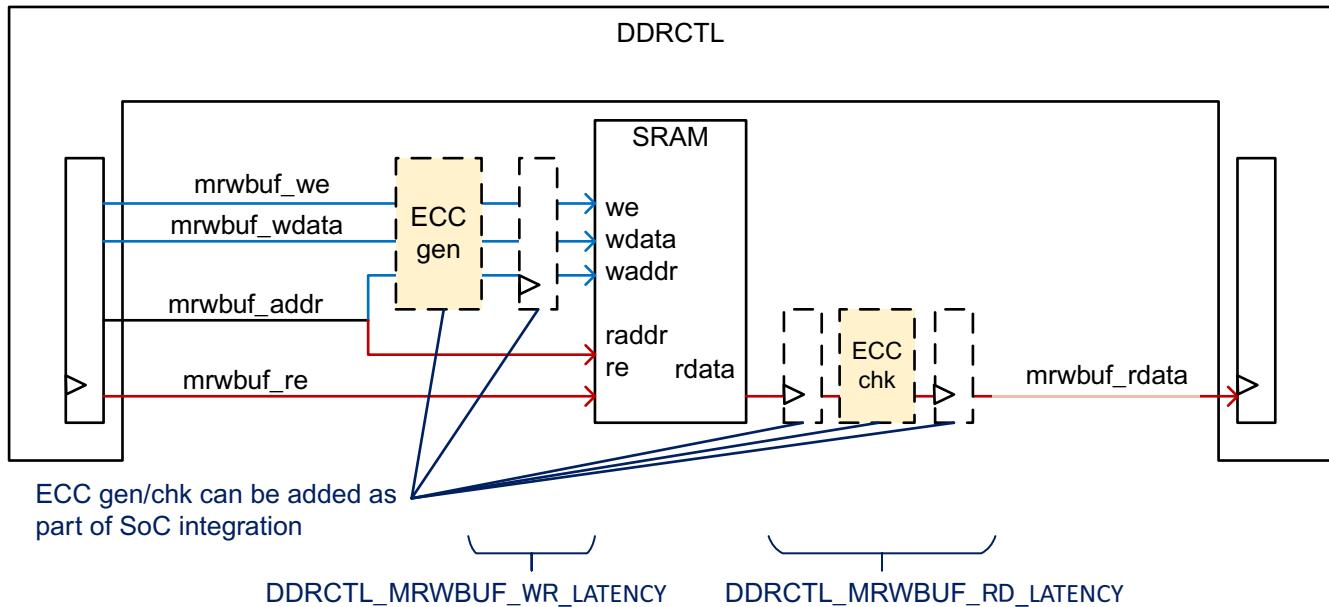
DDRCTL_MRWBUFF_RD_LATENCY must be 2 and DDRCTL_MRWBUFF_WR_LATENCY must be 1.

An optional SRAM wrapper, that has an ECC generation, check, and handle mechanism, may be implemented. As DDRCTL does not provide any recover scheme, the Host should deal with any fatal errors found by the wrapper.



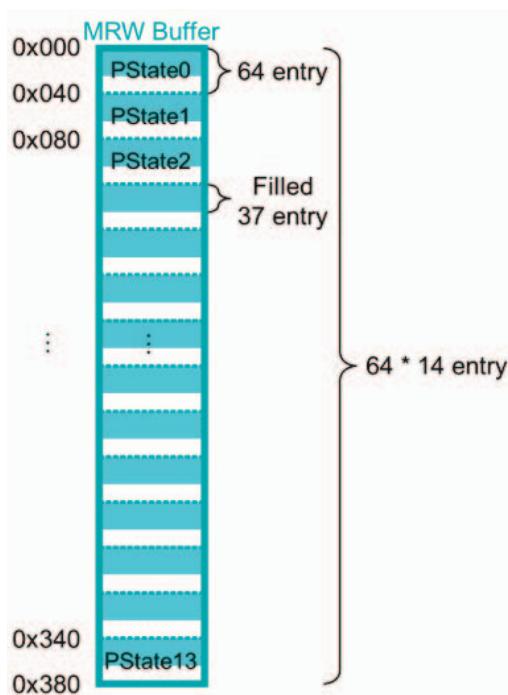
Note SRAM wrapper described here and shown in [Figure 11-29](#) is not delivered as part of the controller IP. You are expected to build it as a part of RAM integration with controller IP effort.

Figure 11-29 External MRWBUF SRAM and its Wrapper



[Figure 11-30](#) describes SRAM addressing. A set of MR value for one PState (37 entries per PState as former assumption) is stored within a block of 64 entries. The block is located the same times as the number of PState, in a linear address space. Total MRW Buffer size will be 64 entry * 14 PState = 896 entries, if there are 14 PStates.

At DDRCTL configuration register, there are two fields that indicate address; namely, `hwffc_mrwbuf_addr` and `hwffc_mrwbuf_select`. They are used when Host programs MRW Buffer. `hwffc_mrwbuf_select` selects which PState to write. `hwffc_mrwbuf_addr` selects what number of MRW value to write. SRAM address will be the concatenation of both when programming. That is, (`hwffc_mrwbuf_addr` at top IO) == {`hwffc_mrwbuf_select` at register field, `hwffc_mrwbuf_addr` at register field}.

Figure 11-30 SRAM Addressing of MRW Buffer

According to the MRW value encoding described in section “[MR Value Encoding](#)”, single MR value is 22-bit width. Actual SRAM size will be (896 entry * 22 bit) = 19712 bit = 19.25 kb.

11.6.9.4 MR Value Encoding

[Figure 11-31](#) shows MR value encoding in MRW Buffer. Each entry contains it.

Figure 11-31 MR Value Encoding

21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rank	Tail	Interval to next	MR address										Data to write								

- Bit[7:0] – 8-bit MR data to be written
- Bit[15:8] – 8-bit destination MR address. Bit[15] is always 0 because LPDDR5 MR address range is 0 to 127 but reserved for future use.
- Bit[18:16] – Selects the interval to next MRW
 - 3'b000: Back-to-back
 - 3'b001: tMRW defined at DRAMSET1TMG3.t_mr
 - 3'b010: tMRW_L defined at DRAMSET1TMG7.t_mrw_l
 - 3'b100: Programmable value defined at DRAMSET1TMG17.t_vrcg_disable
 - 3'b101: Programmable value defined at DRAMSET1TMG17.t_vrcg_enable

- Others: Illegal

If the next MR to be written is for different rank than current MR's, you can set this field to 'Back-to-back' whose interval is DDRCTL_MRWBUFF_RD_LATENCY+3. Otherwise, set to 'tMRW' or 'tMRW_L'. tMRW_L is a variant of tMRW. It is used for DFE related MRW.

If other desired interval is needed, set to 'programmable value', that is, 3'b100 or 3'b101, which refers to DRAMSET1TMG17.t_vrcg_disable or DRAMSET1TMG17.t_vrcg_enable, respectively. These registers need not be the same as the actual JEDEC timing parameter tVRCG_Disable or tVRCG_Enable. These registers would not be used for other usage than 'programmable value'. Host can set them to any value.

- Bit[19] – Tail flag. Tail is equal to '1' at the last MR value among the PState. Otherwise, Tail is equal to '0'.
- Bit[21:20] – Destination rank:
 - 2'b11: MRW will be issued to both rank 0 and rank 1.
 - 2'b01: MRW will be issued to only rank 0.
 - 2'b10: MRW will be issued to only rank 1.
 - 2'b00: Illegal

11.6.9.5 MRW Buffer Write Procedure

Host programs MRW values into MRW Buffer at the initialization of DDRCTL, just before asserting the first `dfi_init_start`.

Follow these steps to write i^{th} MR that is issued right before changing frequency/PState to PState p :

1. Set `HWFFC_MRWBUFF_CTRL1.hwfffc_mrwbuff_wdata` to encoded i^{th} MR value according to [Figure 11-31](#).
2. Set `HWFFC_MRWBUFF_CTRL0.select` to p .
Set `HWFFC_MRWBUFF_CTRL0.addr` to i .
Set `HWFFC_MRWBUFF_CTRL0.rw_type` to '1'.
3. In separated apb access, set `HWFFC_MRWBUFF_CTRL0.rw_start` to '1'.
This field will be automatically cleared.

Repeat the previous steps for all i and all p .

11.6.9.6 MRW Buffer Read Procedure

This is a debug feature. If Host needs to check the data in MRWBUFF SRAM to debug, follow these steps to read i^{th} MR that is issued right before changing frequency/PState to PState p . HWFFC can work without these steps.

1. Set `HWFFC_MRWBUFF_CTRL0.select` to p .
2. Set `HWFFC_MRWBUFF_CTRL0.addr` to i .
Set `HWFFC_MRWBUFF_CTRL0.rw_type` to '0'.
3. In separated apb access, set `HWFFC_MRWBUFF_CTRL0.rw_start` to '1'.
This field will be automatically cleared.

4. Read `HWFFC_MRWBUFF_STAT.hwfffc_mrwbuff_rdata`.

11.6.9.7 MMFW Procedure

PHY MMFW has a functionality to restore unloaded PState into PHY. See PHY documents for more details. DFI frequency change can be done only to the currently loaded PStates in PHY. Typically, MMFW procedure is done right before Host starts HWFFC procedure, so that required target PState is dynamically loaded and selected.

According to PHY documents, DFI sideband signals need to be inactive during MMFW. MMFW procedure is as follows. See section [“Stopping DFI Sideband Signals During MMFW”](#) on page [440](#) for steps 1 and 3.

1. Stop DFI sideband.
2. MMFW: Load software PState X into hardware PState Y
3. Resume DFI sideband.

After this procedure is done, the remaining HWFFC flow follows the procedure described in section [“HWFFC Procedure When HWFFCCTL.hwfffc_mode=1”](#) on page [293](#). `csystarget_frequency_ddrc` needs to be X, and `csysfrequency_ddrc` needs to be Y in the flow.

11.7 Low-Power Optimized Write Data for LPDDR5/4 Masked Write

The LPDDR5/4 JEDEC specification indicates that the LPDDR5/4 device masks the Write data received on the DQ inputs if the total count of '1' data bits on DQ[2 : 7] or DQ[10 : 15] (for lower or upper byte, respectively) is equal to or greater than five, and DMI signal is LOW.

A DQ in the LPDDR5/4 SDRAM consumes less power when the logic level of DQ is '0' than when it is '1'. If DFIMISC. lp_optimized_write is set to '1' and the LPDDR5/4 SDRAM device is used, dfi_wrdata per byte lane in Masked Write with enabling Write DBI is shown in [Table 11-9](#) appears on the DFI bus. That is, setting DFIMISC.lp_optimized_write to '1' can reduce power consumption in DQ.

Table 11-9 Write DBI on DFI Bus

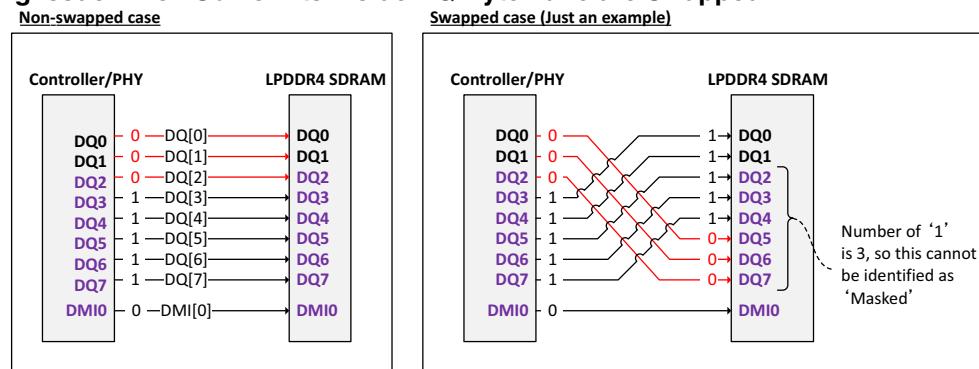
DFIMISC.lp_optimized_write	dfi_wrdata Per Byte Lane in Masked Write with Enabling Write DBI	Notes
0	8'b1111_1111	
1	8'b1111_1000	All the following conditions must be met: <ul style="list-style-type: none"> ■ DBICTL.wr_dbi_en=1 ■ DBICTL.dm_en=1 ■ DBICTL.phy_dbi_mode=0



This feature requires a straight-through DQ wiring between controller/PHY and SDRAM.

For example, it is possible that DQ[7 : 0]=8'b1111_1000 can be mapped as 8'b0001_1111 at SDRAM as shown in [Figure 11-32](#) on page 309. If this happens because of the DQ wiring, then it cannot be identified as 'Masked' by the SDRAM. To optimize power consumption in DQs, a straight-through connection between controller/PHY and SDRAM is necessary. Otherwise, DFIMISC.lp_optimized_write must be set to '0'.

Figure 11-32 Wiring Issue When Some Bits Inside DQ Byte Lane are Swapped



11.8 BSM Clock Removal

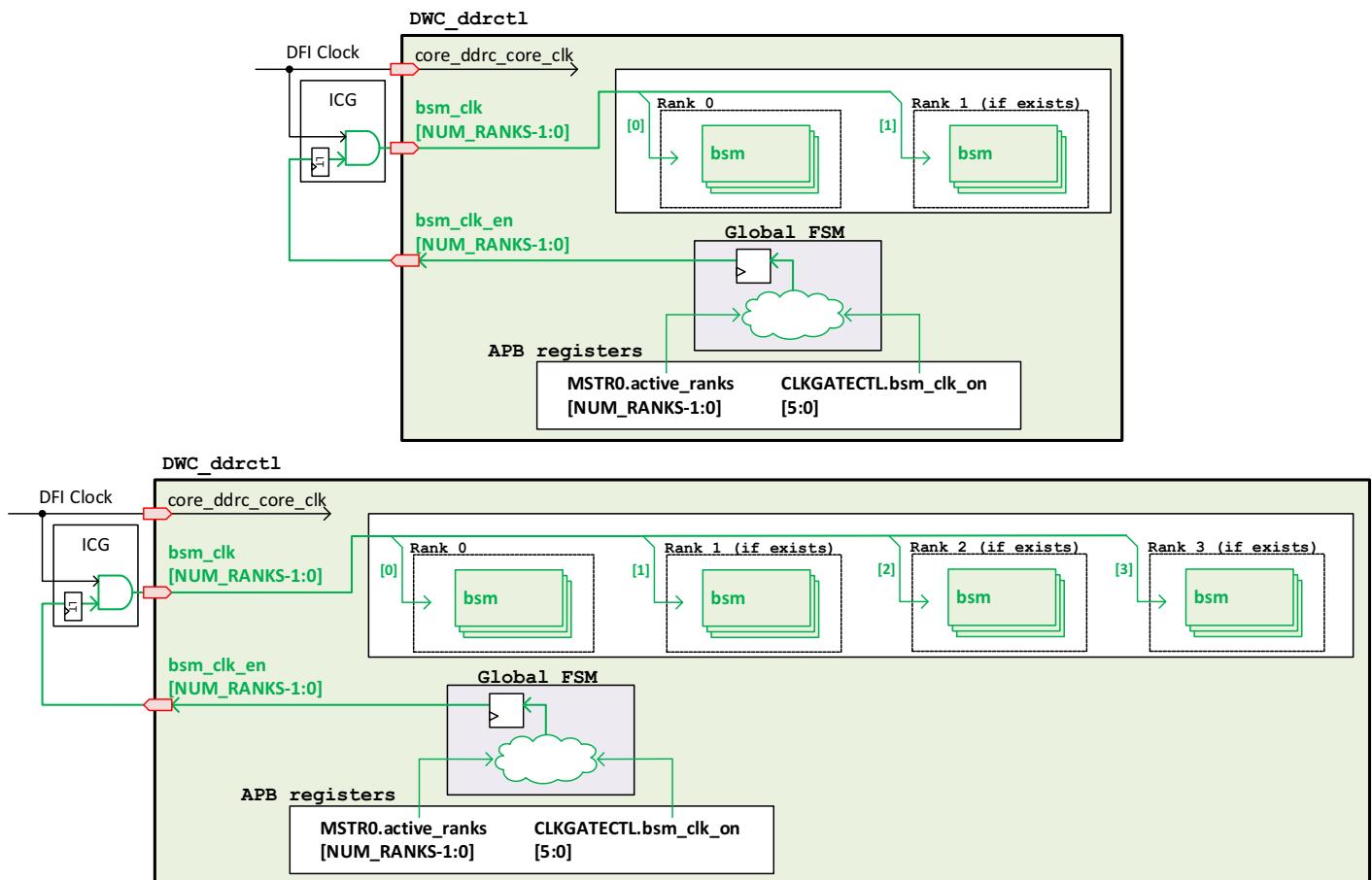
While SDRAM is in power saving mode such as DSM and SRPD, DDRCTL internal logic BSM (Bank State Machine), which manages state for each bank in SDRAM, can also save the power through removing BSM clock.

DDRCTL has two BSM clock related ports `bsm_clk` and `bsm_clk_en` which width is `NUM_RANKS`.

Regardless whether this feature is enabled or disabled, `bsm_clk` must be driven by the clock which is in the same domain as `core_ddrc_core_clk`. When BSM clock removal is enabled, `bsm_clk_en` starts to indicate when `bsm_clk` can be gated, that is, `bsm_clk[i]` can be gated only when `bsm_clk_en[i]` is 0, where '*i*' represents BSM group for SDRAM rank *i*.

Figure 11-33 shows how to configure the behavior of `bsm_clk_en` and how to drive `bsm_clk`.

Figure 11-33 Hardware and Software Configuration to Gate BSM Clock

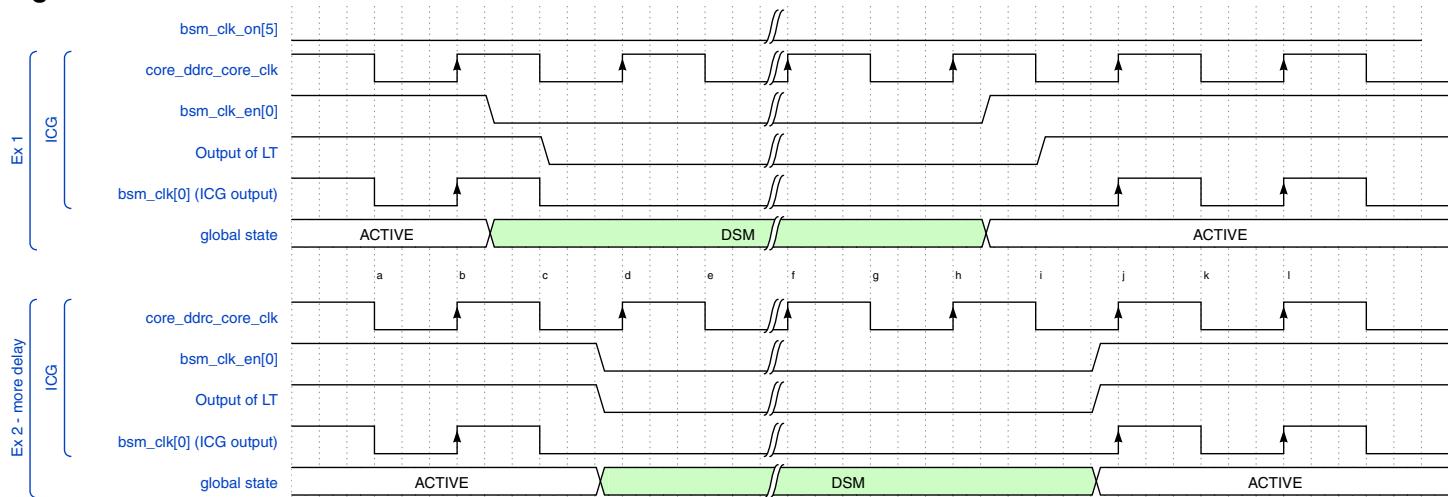


The register `CLKGATECTL.bsm_clk_on` represents how aggressively the BSM clock is removed. For example, setting 1'b0 to `CLKGATECTL.bsm_clk_on[1]`, which is for Deep Sleep Mode control, lets `bsm_clk_en[*]` to be de-asserted while SDRAM is in DSM state. For another example, if `CLKGATECTL.bsm_clk_on[0]` for Unpopulated rank control is 1'b0, `bsm_clk_en[i]` is always 0, where '*i*' is an unused rank according to `MSTR0.active_ranks`.

`CLKGATECTL.bsm_clk_on` is 6'b11_1111 by default, therefore `bsm_clk_en[*]` is always 1 if not configured.

To maximize power saving, this field needs to be set to 6'b000_0000 on initialization. For more information, see Synopsys LPDDR5/4/4X Memory Controller Reference Manual.

Figure 11-34 ICG Behavior with BSM Clock Removal



If BSM clock removal is enabled, `bsm_clk` might be driven by ICG (Integrated Clock Gating) cell which takes `core_ddrc_core_clk` and `bsm_clk_en` as inputs, as shown in Figure 11-33. Figure 11-34 shows how ICG behaves. When the SDRAM state transits to idle states from other active states, `bsm_clk_en[*]` is de-asserted. Once ICG accepts¹ `bsm_clk_en` de-assertion, it starts to drive `bsm_clk` low. Then, ICG resumes `bsm_clk` after `bsm_clk_en` assertion is accepted.

1. See the period c-d and i-j at ‘Output of LT’. A latch in ICG accepts `bsm_clk_en` toggle only when `core_ddrc_core_clk` is low.

11.9 APB Clock Gating

Normally, there are two clocks `core_ddrc_core_clk` and `pclk` related to APB function modules. For power saving purpose, two extra clocks `core_ddrc_core_clk_apbrw` and `pclk_apbrw` are defined for APB function modules, after hardware parameter `DDRCTL_EXTRA_CLK_APB_EN` is set to '1'. Clock `pclk_apbrw` is used for APB read and write operation. Clock `core_ddrc_core_clk_apbrw` is used for register CDC from `core_ddrc_core_clk` clock domain to `pclk` clock domain.

These two extra clocks `core_ddrc_core_clk_apbrw` and `pclk_apbrw` needs gating from clocks `core_ddrc_core_clk` and `pclk`. When there is no APB read or write access, you can disable these two extra clocks. If you want to access APB register, you must enable these two extra clocks in advance.

11.10 Signals Related to Power Saving

For the power saving features, STAT.operating_mode signal can be used to monitor the current operating mode of the DDRCTL.

The following signals are related to:

Hardware low-power interface of the DDRC:

- csysreq_ddrc
- csysack_ddrc
- cactive_n (input pin to the DDRC with one bit per port)
- cactive_ddrc

Hardware low-power interface of the AXI ports:

- csysreq_n
- csysack_n
- cactive_n

For more information about these signals, refer to the “Signal Descriptions” chapter.

12

LPDDR5 Specific Features

This chapter contains the following sections:

- “[Overview of LPDDR5 Specific Features](#)” on page [316](#)
- “[Bank Organization](#)” on page [317](#)
- “[WCK Clocking](#)” on page [319](#)
- “[Link ECC](#)” on page [321](#)
- “[LPDDR5X Support With Synopsys LPDDR5X/5/4X PHY](#)” on page [331](#)

12.1 Overview of LPDDR5 Specific Features

The following are unique features of the LPDDR5 configurations:

- Bank organization:
 - LPDDR5 SDRAM supports three bank architectures which are selected by mode register.
 - The maximum data rate depends on the bank architecture.
- WCK clocking:
 - LPDDR5 SDRAM needs two clocks for command/address (CK, single data rate) and DQ (WCK, double data rate).
 - CAS with WCK sync command is necessary before READ, WRITE, MRR if DRAM is not synchronized with WCK.
- Command encoding:
 - LPDDR5 SDRAM does not have CKE pin.
 - ACTIVATE command is composed of two commands, ACT1 and ACT2. ACT2 command can be separated from ACT1 command within tAAD.
- Link ECC:
 - LPDDR5 SDRAM supports SEC/DED ECC for Link Protection.
 - Link ECC is an optional feature.

12.2 Bank Organization

LPDDR5 SDRAM supports three bank architectures to provide optimal access methods for varied system configurations. The native burst length determined by data prefetch size depends on which bank architecture is enabled.

Each architecture name is abbreviated as follows:

BG Mode = 4 banks, 4 bank groups

8B Mode = 8 banks, no bank groups

16B Mode = 16 banks, no bank groups

The supported operation data rate for each Bank/Bank Group Organization is as below.

BG Mode for more than 3200Mbps (>3200Mbps).

8B Mode for all data rate range.

16B Mode for equal or less than 3200Mbps (<=3200Mbps).

The bank architecture is selected by DRAMSET1TMG24.bank_org. This mode register is replicated for each frequency.

The BG and 16B modes support burst lengths of 16 or 32, while 8B mode supports only burst length 32.

The DRAM array mapping in these three modes is shown in [Table 12-1](#).

Table 12-1 Address Array Mapping

Bank Architecture	BG	BA0	BA1	BG0	BG1
	8B	BA0	BA1	BA2	B4
	16B	BA0	BA1	BA2	BA3



- BA0-3: Bank Address, BG0-1: Bank Group address, B4 Burst Starting Address.
- The DDRCTL does not support 8B mode.

12.2.1 Address Mapping

LPDDR5 SDRAM introduces Burst addresses which indicates starting address within the burst. Since this address is same as lower column address for previous protocols, burst address is mapped by addrmap_col_b3 register.



- Burst address [2:0] is mapped to HIF address [2:0].
- HIF address [2:0] must be 3'b000.

For consistency between 16 bank mode and BG mode, the addrmap_bg register is used for BA2, BA3 for 16B mode and BG0, BG1 for BG mode.

Table 12-2 Address Mapping for BG Mode and 16B Mode

Register	BG mode	16B mode
ADDRMAP6.addrmap_col_b3	B3	B3
ADDRMAP6.addrmap_col_b4	C0	C0
ADDRMAP6.addrmap_col_b5	C1	C1
ADDRMAP6.addrmap_col_b6	C2	C2
ADDRMAP5.addrmap_col_b7	C3	C3
ADDRMAP5.addrmap_col_b8	C4	C4
ADDRMAP5.addrmap_col_b9	C5	C5
ADDRMAP3.addrmap_bank_b0	BA0	BA0
ADDRMAP3.addrmap_bank_b1	BA1	BA1
ADDRMAP4.addrmap_bg_b0	BG0	BA2
ADDRMAP4.addrmap_bg_b1	BG1	BA3

12.2.2 Burst Length

The Read/Write command behavior depends on the bank architecture. Each mode supports the following burst length.

BG mode: BL16, BL32 (interleaved)

8B mode: BL32

16B mode: BL16, BL32



The DDRCTL only supports BL16 when `MEMC_BURST_LENGTH=16`.

12.3 WCK Clocking

The LPDDR5 command and address interface operates from a differential clock (CK_t and CK_c). Commands and addresses are registered double data rate (DDR) at every rising edge of CK_t and CK_c. CS is sampled at the rising edge of CK_t and the falling edge of CK_c.

LPDDR5 uses a DDR data interface. The data interface uses two differential forwarded clocks (WCK_t/WCK_c) that are source synchronous to the DQs.

WCK is used to sample DQ data for write operation and toggle DQ data for read operation. WCK must start toggle before starting write or read DQ data burst. Any commands that require DQ data burst initiate WCK2CK auto-sync sequence in LPDDR5 SDRAM.

WCK is managed by the DDRCTL using the following registers:

```
DFITMG4.dfi_twck_en_rd
DFITMG4.dfi_twck_en_wr
DFITMG4.dfi_twck_dis
DFITMG5.dfi_twck_fast_toggle
DFITMG5.dfi_twck_toggle
DFITMG5.dfi_twck_toggle_cs
DFITMG5.dfi_twck_toggle_post
```

The values of these registers are provided by the PHY. Refer to relevant PHY databook for more information.

LPDDR5 SDRAM utilizes two types of clock with different frequency. The frequency of WCK is four times or twice higher than the command clock.

The supported data rate for CK:WCK==1:2 mode is up to 3200Mbps.

Table 12-3 Supported CK:WCK Frequency Ratio

CK:WCK Ratio	Bank Organization	Data Rate	WCK Frequency	CK Frequency
1:2	16B mode	<= 3200 Mbps	<= 1600 MHz	<= 800 MHz
1:4	16B mode	<= 3200 Mbps	<= 1600 MHz	<= 400 MHz
	BG mode	> 3200 Mbps	> 1600 MHz	> 400 MHz

The CK:WCK frequency ratio is set by TMGCFG.frequency_ratio.

With respect to controller clock (that is, DFI clock), the clocking relationship can be also summarized as DFI:CK:WCK which corresponds to frequency ratios of 1:1:2 and 1:1:4.

12.3.1 WCK Behavior

The controller supports two WCK modes:

- MSTR4.wck_on = 0: WCK always ON mode disabled (WCK on demand mode)
- MSTR4.wck_on = 1: WCK always ON mode enabled (WCK always on mode)

In WCK on demand mode, the controller issues CAS-WS_RD command before READ/MRR or CAS-WS_WR command before WRITE if necessary, and the controller starts to toggle WCK. When the

toggling period is over, the controller stops WCK toggling. In multi-rank system, the controller issues CAS-WS command to one rank only. More than one rank cannot be in WCK2CK synchronization state simultaneously.

In WCK always on mode, the controller issues CAS-WS_RD command before READ/MRR or CAS-WS_WR command before WRITE at single rank configuration and issues CAS-WS_FS command before READ/WRITE/MRR at multi rank configuration, and the controller starts to toggle WCK. WCK is toggling until a power-down, self-refresh power-down, deep sleep mode or CAS-WS_OFF command is issued. The controller issues CAS-WS_OFF command when the following are required:

- DFI Control update
- DFI PHY update
- DFI PHY master

And also the controller issues CAS-WS_OFF command when there are no on-going Read or Write WCK2CK SYNC operations and MSTR4.ws_off_en is set to '1'. If MSTR4.wck_on=1, MSTR4.ws_off_en must be set to '1'.

In a multi-rank configuration, the controller issues CAS-WS_FS to all ranks and all ranks are in WCK2CK synchronization state simultaneously.



Note In a single-rank configuration (MEMC_NUM_RANKS = 1 or MSTR0.active_ranks = 0x1), it is recommended to use WCK on demand mode (MSTR4.wck_on = 0).

12.3.2 Enhanced WCK Always On Mode

If LPDDR5 device supports an enhanced WCK Always On Mode by reading out MR0 OP[2]=1, the controller is able to issue a CAS-WCK_SUSPEND command to reduce some of internal WCK clock net power consumption inside LPDDR5 devices. CAS-WCK_SUSPEND is a standalone command and may be issued anytime unless it interrupts on-going Read or Write WCK2CK SYNC operation when WCK always on mode is enabled. WCK SUSPEND mode keeps requiring WCK toggling input and exits automatically by following Read or Write, or Mask Write commands without issuing any additional explicit command.

The controller issues a CAS-WCK_SUSPEND only when MSTR4.wck_on = 1 and MSTR4.wck_suspend_en = 1.

If LPDDR5 device does not support an enhanced WCK Always On Mode (MR0 OP[2]=0), MSTR4.wck_suspend_en must be set to '0'.

If DDRCTL_ENHANCED_WCK is not defined, MSTR4.wck_on must be set to '0'.

12.4 Link ECC

This topic contains the following sections:

- “[Overview of Link ECC Support](#)” on page 321
- “[Enabling Link ECC](#)” on page 321
- “[Link ECC Restrictions](#)” on page 321
- “[Controller Behavior During Read Link ECC Errors](#)” on page 321
- “[Inline ECC and Link ECC Features Combination](#)” on page 322
- “[Link ECC Error Reporting](#)” on page 324
- “[Link ECC Data Poisoning](#)” on page 325
- “[Performance Impact](#)” on page 326
- “[Registers Related to Link ECC Support](#)” on page 326

12.4.1 Overview of Link ECC Support

The DDRCTL supports Link ECC feature for Read and Write. If supported by the DRAM, Link ECC may then be enabled or disabled by the software as required by the system configuration, operating speed, or other requirements. For Read, the DDRCTL detects single/double bit error and stored the error count which can be read via APB register. If Link ECC error is single bit error, the DDRCTL corrects the data and sends to the host.

12.4.2 Enabling Link ECC

To enable Link ECC, set the hardware configuration parameter `MEMC_LINK_ECC` to ‘1’. This feature can then be enabled/disabled by software using the `LNKECCCTL0.wr/rd_link_ecc_enable` register.

12.4.3 Link ECC Restrictions

The following restrictions apply to Link ECC feature:

- Read-DBI must be disabled
- CAS(B3) must be ‘0’ at read command
- BL16 or BL32 only
- 1:1:4 mode only
- Partial Read is not supported (`hif_cmd_length` must be set to 2'b00 (Full Read)) in BL16. Quarter Read is not supported (`hif_cmd_length` must be set to 2'b00 (Full Read) or 2'b01 (Half Read)) in BL32.

12.4.4 Controller Behavior During Read Link ECC Errors

When the DDRCTL detects a correctable Read Link-ECC error, it performs the following:

- Sends the corrected data to the host as part of the read data.
- Sends the Link ECC error information to the APB register.
- Asserts the interrupt signal.

When the DDRCTL detects an uncorrectable Read Link-ECC error, it does the following:

- Sends the data with error to the SoC as part of the read data.
- Sends the Link ECC error information to the APB register.
- Asserts the interrupt signal.
- Assert `hif_rdata_uncorr_linkecc_err`.
- Generates a SLVERR response on the AXI interface. Note, that in case of Inline ECC overhead (RE_B) command and Read part of RMW, DDRCTL does not assert SLVERR because they do not support SLVERR.

12.4.5 Inline ECC and Link ECC Features Combination

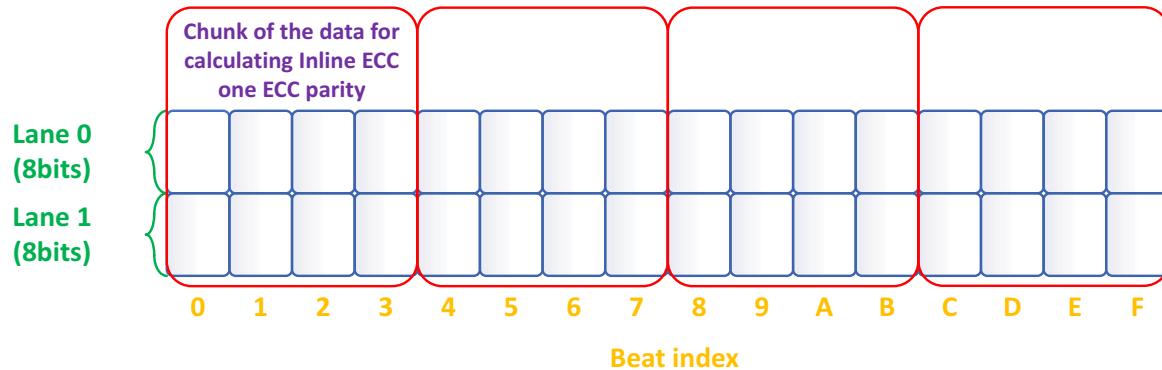
Read

When both Link ECC and Inline ECC are enabled, Link ECC check and correction is performed before Inline ECC check, correction, and detection. The Inline ECC error happens on the SDRAM devices. Link ECC parity is calculated by using SDRAM saving data even if it has already broken. This means Link ECC feature cannot detect the Inline ECC error.

If 2-bits Link ECC error happens, it cannot be corrected and this uncorrected data is propagated to Inline ECC checking mechanism.

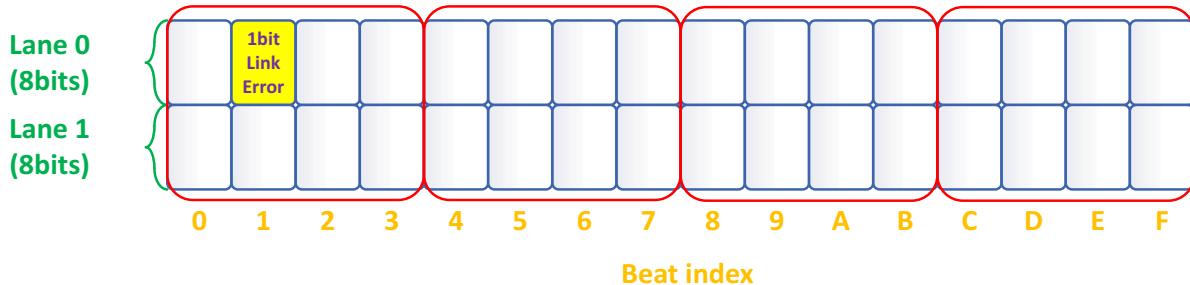
Inline ECC generates ECC parity per 8-byte data. If `MEMC_DRAM_DATA_WIDTH = 16`, 1-byte ECC parity is generated by 4-beats data.

Figure 12-1 Link ECC and Inline ECC Data Chunk Structure



If 1-bit Link ECC error happens, this error can be corrected by Link ECC parity. Then, the Inline ECC mechanism does not detect any errors.

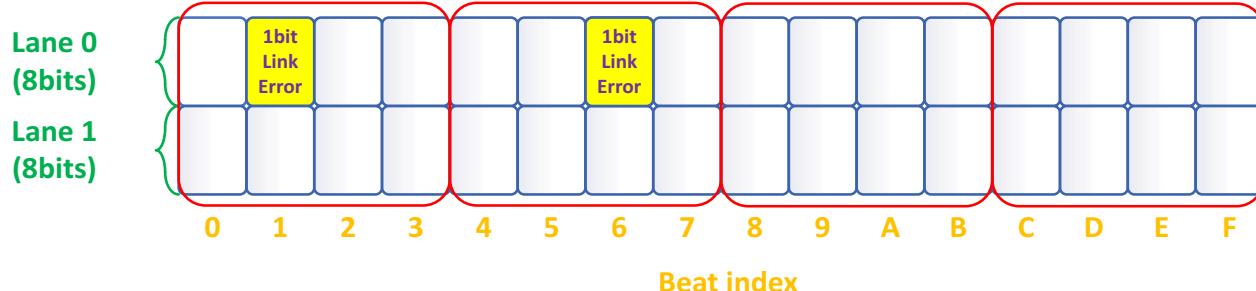
Figure 12-2 Link ECC 1-Bit Error



It is possible that Inline ECC can correct data where Link ECC indicates the uncorrected error, if the reason of Link ECC error is on the different 1-bit error on the chunk of 4-beats data. In the following case, Inline

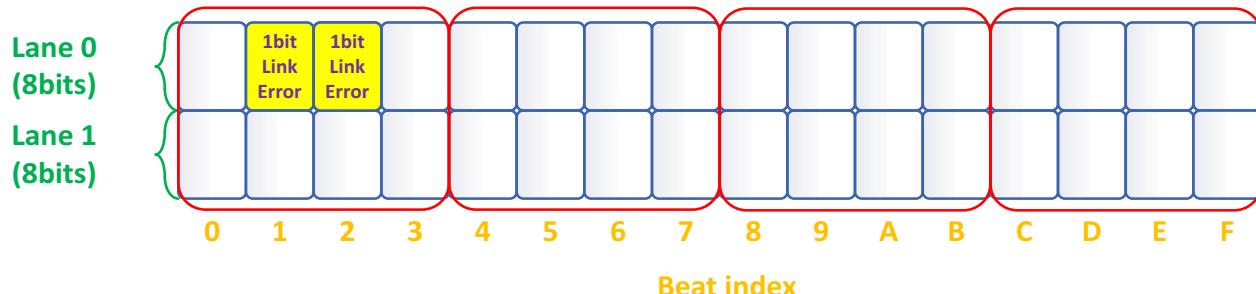
ECC judges it as 2 correctable errors. Therefore, this error is corrected by Inline ECC. Even though the Link ECC error is corrected by Inline ECC, the RRESP for this data informs as SLVERR. If this type of Link ECC error happens with the read part of RMW, the corrected data is sent to SDRAM.

Figure 12-3 2-bits Link ECC Error In The Different Inline ECC Data Chunk



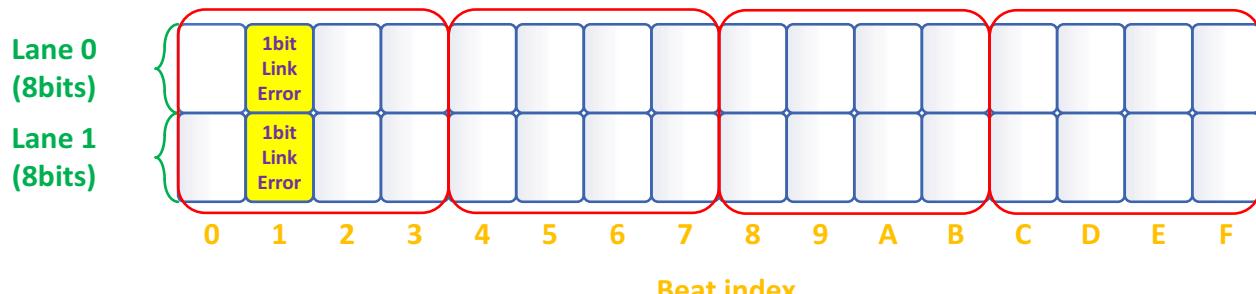
If 2-bits Link ECC error happens in the same beat as shown in [Figure 12-4](#), Inline ECC judges as uncorrected error. If this type of error happens on the read part of RMW command, the data including 2-bits error is sent to SDRAM.

Figure 12-4 2-bits Link ECC Error In The Same Inline ECC Data Chunk

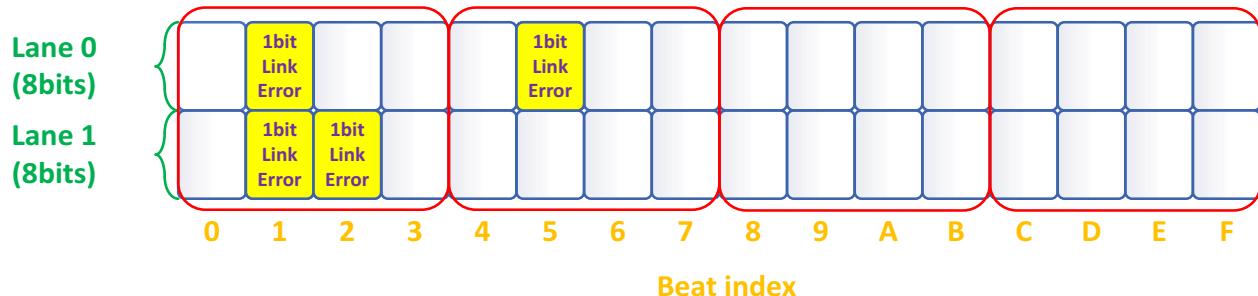


The Link ECC parity is generated for each data lane. In the following case, it is 1-bit error for each data lane from the perspective of Link ECC. This error can be fixed by Link ECC.

Figure 12-5 2-bits Link ECC Error In The Same Inline ECC Data Chunk And Different Data Lane



2-bits error for each data lane can be propagated to Inline ECC. If the link ECC 2-bits error happens on the same Inline ECC data chunk, it becomes 3-bits or 4-bits error. However, the Inline ECC feature does not detect an error greater than 2 bits. So, in this case, errors cannot be detected correctly.

Figure 12-6 2-bits Link ECC Error In The Different Data Lane**Write**

Link ECC works without distinguishing a common write data and Inline ECC data. There is no difference in the behavior depending on whether Inline ECC is enabled or not.

Poisoning

Link ECC poisoning feature poisons a DMI bit. If Link ECC poisoning feature is used, Inline ECC mechanism cannot detect any errors.

12.4.6 Link ECC Error Reporting

This section has the following sub-sections:

- “[Write Link ECC Error](#)” on page [324](#)
- “[Read Link ECC Error](#)” on page [324](#)

12.4.6.1 Write Link ECC Error

A counter of Single-Bit-Errors and a Double-Bit-Error flag will be maintained on the DRAM, both of which can be read through mode register MR43. The DRAM will also store both syndromes from the most recent single-bit ECC error in MR44 & MR45. These Mode Registers can be read by software via MRCTRL register to check Link ECC error status.

For more information about MRR operation, see the “[Mode Register Reads and Writes](#)” on page [252](#) section.



- According to JEDEC LPDDR5 specification, these registers are cleared on Power-Down exit. It is recommended to disable automatic Power-Down/Self-Refresh entry feature.
- When performing MRR command via `MRCTRL` register, it is recommended to have at least 32 pclk cycles between two consecutive writes to the `MRCTRL` register to ensure the timing constraint “tMRR + tMRW”.

12.4.6.2 Read Link ECC Error

For Link ECC configuration, there are several indirect registers as in [Table 12-4](#). The selectors of the indirect registers are defined in `LNKECCINDEX` register.



The term “indirect register” means the register is internally replicated (like a register array), and addressed through setting the register’s selector to access the required one.

Table 12-4 Link-ECC Indirect Registers

Register Name	Depth	Selector (Fields of LNKECCINDEX)
LNKECCERRCNT0.rd_link_ecc_uncorr_cnt	MEMC_NUM_RANKS* (MEMC_DRAM_DATA_WIDTH/8)	
LNKECCERRCNT0.rd_link_ecc_corr_cnt	MEMC_NUM_RANKS* (MEMC_DRAM_DATA_WIDTH/8)	LNKECCINDEX.rd_link_ecc_err_byte_sel LNKECCINDEX.rd_link_ecc_err_rank_sel
LNKECCERRCNT0.rd_link_ecc_err_syndrome	MEMC_NUM_RANKS* (MEMC_DRAM_DATA_WIDTH/8)	

LNKECCINDEX is quasi-dynamic registers. To write them, set SWCTL.sw_done to '0' at the beginning of the programming sequence and set back to '1' at the end. After that, poll SWSTAT.sw_done_ack for acknowledge.

To read LNKECCERRCNT0 register indirectly, follow the steps in [Table 12-5](#).

Table 12-5 Procedure to Read from the LNKECCERRCNT0 Register indirectly

Step	Operation	Description
1	Set SWCTL.sw_done to '0'	Enable quasi-dynamic register write
2	Write appropriate values to: ■ LNKECCINDEX.rd_link_ecc_err_byte_sel ■ LNKECCINDEX.rd_link_ecc_err_rank_sel	Select target byte(s) of Data Select target Rank
3	Set SWCTL.sw_done to '1'	Complete the quasi-dynamic register write
4	Write '1' to SWSTAT.sw_done_ack	Wait for acknowledge
5	Read ■ LNKECCERRCNT0.rd_link_ecc_uncorr_cnt ■ LNKECCERRCNT0.rd_link_ecc_corr_cnt ■ LNKECCERRCNT0.rd_link_ecc_err_syndrome	Read out the Link-ECC Syndrome, uncorrected error count and corrected error count.

12.4.7 Link ECC Data Poisoning

In Link ECC mode, the DDRCTL provides the ability to add errors in the read/write data burst. This data burst can then be used as a test of how the system handles uncorrectable, correctable Link ECC errors.

[Table 12-6](#) shows the software sequence for the Link ECC poisoning. One sequence injects one Link ECC error. Software can repeat the sequence to inject multiple Link ECC errors.

Table 12-6 Software Sequence for the Link ECC Poisoning

Step	Description	Comment
1	Write appropriate value to: ■ LNKECCPOISONCTL0.linkecc_poison_byte_sel ■ LNKECCPOISONCTL0.linkecc_poison_dmi_sel ■ LNKECCPOISONCTL0.linkecc_poison_type ■ LNKECCPOISONCTL0.linkecc_poison_rw	Select target byte(s) of Data Select target DMI(s) Single-bit error or Double bit error Select Read or Write
2	Write '1' to LNKECCPOISONCTL0.linkecc_poison_inject_en	Enable Link-ECC poisoning. A Link-ECC error is injected to the first matched data/dmi.
3	Poll LNKECCPOISONSTAT.linkecc_poison_complete=1	Once a Link-ECC error is injected, the register becomes equal to '1'. This step can be skipped if software wants to cancel the error injection.
4	Write '0' to LNKECCPOISONCTL0.linkecc_poison_inject_en	Disable Link-ECC poisoning.
5	Poll LNKECCPOISONSTAT.linkecc_poison_complete=0	Make sure the Link-ECC poisoning is completed/terminated.

12.4.8 Performance Impact

Enabling the Link ECC feature (LNKECCCTL0.wr/rd_link_ecc_enable=1) will impact the performance:

- Some timing parameters in JEDEC LPDDR5 specification will increase. Certain timing registers need to be adjusted accordingly.
- Read Latency from DFI to HIF will increase by 2 core clock cycles.

12.4.9 Registers Related to Link ECC Support

The following are the registers related to link ECC support:

- LNKECCCTL0
- LNKECCCTL1
- LNKECCPOSNCTL0
- LNKECCPOISONSTAT
- LNKECCINDEX
- LNKECCERRCNT0
- LNKECCERRSTAT

For more information about these registers, see the “Register Descriptions” chapter of the [DWC DDRCTL LPDDR5/4 Programming Guide](#).

12.5 LPDDR5 Post Package Repair

This topic contains the following sections:

- “Overview of LPDDR5 Post Package Repair” on page [327](#)
- “Enabling LPDDR5 Post Package Repair Feature (PPR)” on page [327](#)
- “Limitations of LPDDR5 PPR” on page [327](#)
- “Software Sequence for LPDDR5 PPR” on page [327](#)
- “Registers Related to LPDDR5 PPR” on page [329](#)

12.5.1 Overview of LPDDR5 Post Package Repair

LPDDR5 devices support a method of Fail Row address repair, PPR (Post Package Repair). DDRCTL supports it with a special sequence of MRW/ACT/PRE which is introduced in this section.

12.5.2 Enabling LPDDR5 Post Package Repair Feature (PPR)

To implement LPDDR5 PPR feature in the controller, the hardware parameter `DDRCTL_LPDDR5_PPR` must be enabled in coreConsultant.

12.5.3 Limitations of LPDDR5 PPR

- Supports only LPDDR5 devices with Synopsys LPDDR 5X/5/4X PHY or 5/4/4X PHY
- Supports both 1:1:2 and 1:1:4 frequency ratio
- Supports both 16B mode and BG addressing mode
- Supports DRAM clock only up to 800 MHz, that is, 3200 Mbps in 1:1:2 or 6400 Mbps in 1:1:4
- PPR should basically follow boot sequence; and once PPR is done, reset sequence must immediately follow it. DRAM, DDRCTL, and PHY should be reset and re-initialized if further operation is needed.
- Other features are unavailable if PPR is used and until reset. The following are not available:
 - Any traffic
 - Data retention
 - Low power features
 - Calibration
 - Other software sequence

Avoiding them during PPR is the software/systems responsibility.

- Even if there is more than one DFI interface, you cannot select target DRAM device in DFI interface wise. For example, if PPR is done on rank 1, all rank 1 devices connected to all DFI interfaces are targeted.

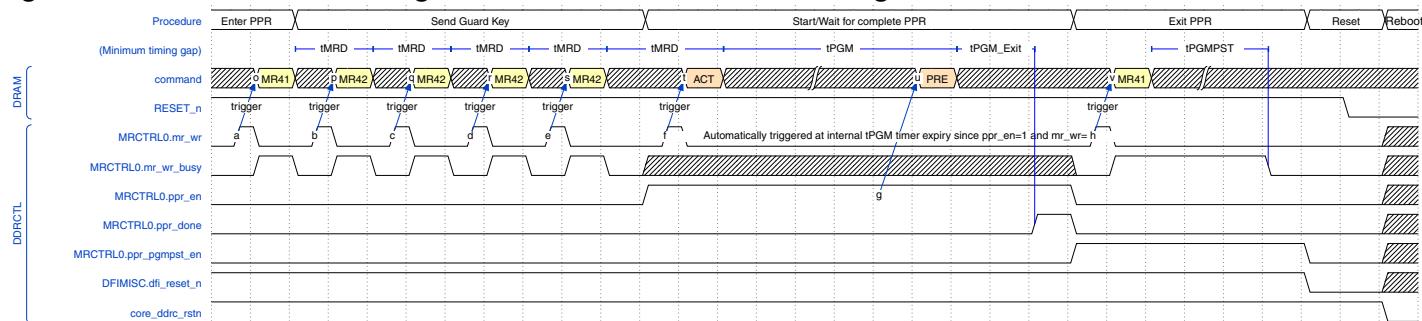
12.5.4 Software Sequence for LPDDR5 PPR

The controller supports LPDDR5 PPR with the software sequences mentioned in [Table 12-7](#).

Table 12-7 PPR Sequence

Step	Description	Comment
	Before the sequence, ensure the following registers are programmed correctly along with the other registers: 1. Set DDR4PPRTMG0/1 2. Set DFIPHYMSTR.dfi_phymstr_en=0 3. Set DFIUPD0.dfi_phyupd_en=0	
1	Disable all low power features: 1. Set PWRCTL.selfref_sw=0 2. Set PWRCTL.dsm_en=0 3. Set PWRCTL.selfref_en=0 4. Set PWRCTL.powerdown_en=0 5. Set HWLPCTL.hw_lp_en=0 6. Poll STAT.operating_mode until it is '1'	
2	Disable all periodic calibration features: 1. Set DQSOSCCTL0.dqsosc_enable=0 2. Set DBG1.dis_auto_zq=1 3. Set DFIUPDTMG2.ppt2_en=0 4. Set DFIUPD0.dis_auto_ctrlupd=1 5. Set DERATECTL0.derate_mr4_pause_fc=1 6. Poll STAT0.dqsosc_state until it is '0'	
3	Disable all traffic: 1. Set OPCTRL1.dis_dq=1 2. Set RFSHCTL0.dis_auto_refresh=1 3. Set MSTR4.wck_on=0 4. Set SBRCTL.scrub_en=0	
4	Stop WCK: 1. Set OPCTRLCMD.ctrlupd=1 2. Poll OPCTRLSTAT.ctrlupd_busy until it is '0'	
5	Enter PPR: 1. Send MRW to MR41 with OP[4]=1	See “Mode Register Reads and Writes” on page 252.
6	Send Guard Key: 1. Send MRW to MR42 four times, with values specified in JEDEC specification	For information regarding Guard Key, see JESD209-5. See “Mode Register Reads and Writes” on page 252.

Step	Description	Comment
7	Program the target rank/bank/row into the following registers: 1. Target rank: MRCTRL0.mr_rank 2. Target bank: MRCTRL0.mr_addr 3. Target row: MRCTRL0.mr_data 4. Set MRCTRL0.mr_type=0	MRCTRL0.mr_rank must be one hot. Three LSB bits of mr_addr represents the targeted bank and/or bank group, that is, {BA2, BA1, BA0} in 16B mode and {BG0, BA1, BA0} in BG mode. Other mr_addr bits should be '0'. BG1 or BA3 is not selectable as per JEDEC specification.
8	Start PPR: 1. Set MRCTRL0.ppr_en=1 2. Set MRCTRL0.mr_wr=1	
9	Wait until PPR completes: 1. Poll MRSTAT.ppr_done until it is '1' 2. Set MRCTRL0.ppr_en=0	
10	Exit PPR: 1. Set MRCTRL0.ppr_pgmpst_en=1 2. Send MRW to MR41 with OP[4]=0 3. Poll MRSTAT.mr_wr_busy until it is '0' 4. Set MRCTRL0.ppr_pgmpst_en=0	See "Mode Register Reads and Writes" on page 252.
11	Reset LPDDR DRAM: 1. Set DFIMISC.dfi_reset_n=0	After PHY receives dfi_reset_n=0, PHY asserts RESET_n at DRAM interface. If multiple MCs exist in your system, it is recommended to set dfi_reset_n to '0' for all MCs.
12	Reboot. Reset DDRCTL and PHY.	

Figure 12-7 PPR/MR Related Registers and DRAM Commands During PPR Procedure

12.5.5 Registers Related to LPDDR5 PPR

The following are the registers related to LPDDR5 PPR:

- MRCTRL0
- MRSTAT
- DDR4PPRTMG0
- DDR4PPRTMG1

They are applicable to LPDDR5 PPR regardless of their prefix "DDR4".

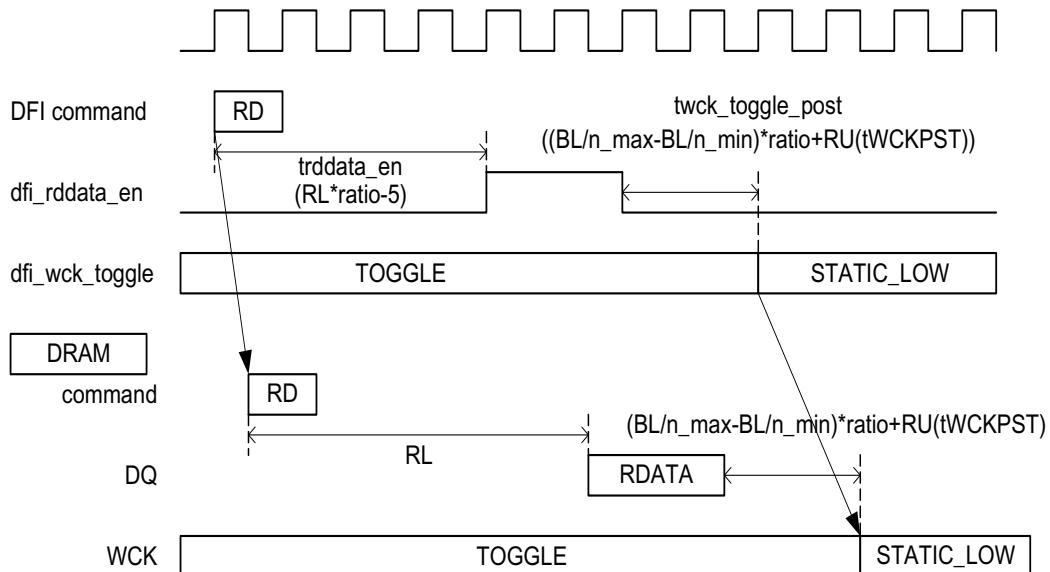


Note MRCTRL0.ppr_en/MRCTRL0.ppr_pgmpst_en must be kept '0' outside of the software sequence for LPDDR5 PPR.

12.6 LPDDR5X Support With Synopsys LPDDR5X/5/4X PHY

WCK toggle period is determined by DFITMG0.dfi_tphy_wrlat, DFITMG0.dfi_t_rddata_en and DFITMG5.dfi_twck_toggle_post, as shown in [Figure 12-8](#).

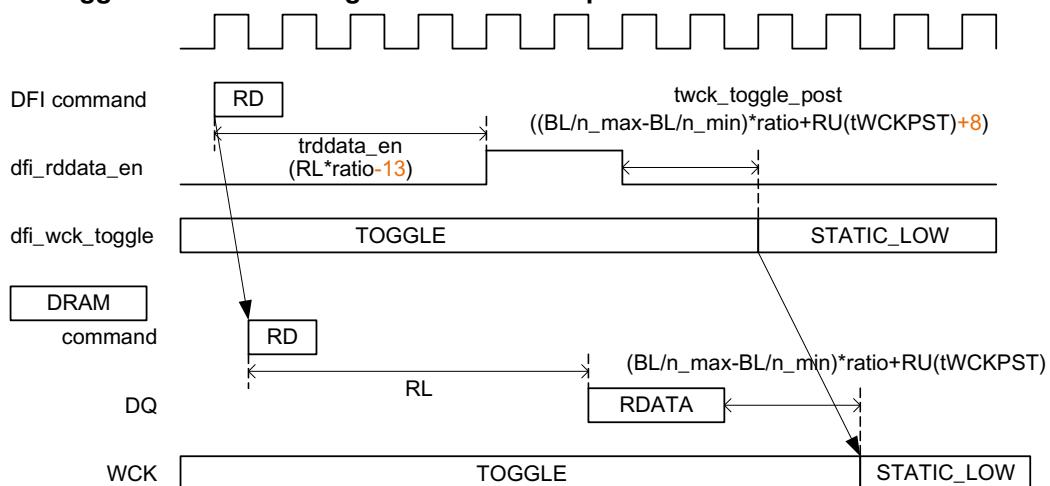
Figure 12-8 WCK Toggle Period



In Synopsys LPDDR5X/5/4X PHY, the parameters `tphy_wrlat` and `twck_toggle_post` are not changed. On the other hand, the parameter `trddata_en` is changed to "`RL * 4 -13 + (AcInPipe - DxInPipe) * 2`" when data rate is higher than 3200 Mbps in 1:4 mode. This is supported through the registers `DFITMG6.dfi_twck_toggle_post_rd` and `DFITMG6.dfi_twck_toggle_post_rd_en`.

When data rate is higher than 3200 Mbps in 1:4 mode for Synopsys LPDDR5X/5/4X PHY, `DFITMG6.dfi_twck_toggle_post_rd` must be set to "`twck_toggle_post + 8`", and `DFITMG6.dfi_twck_toggle_post_rd_en` must be set to '1', as illustrated in [Figure 12-9](#).

Figure 12-9 WCK Toggle at Data Rates Higher Than 3200 Mbps



13

RAS Features

This chapter contains the following sections:

- “[Memory ECC](#)” on page [334](#)
- “[Scrubber](#)” on page [373](#)
- “[On-Chip Parity \(OCPAR\)](#)” on page [386](#)
- “[Registers Parity Protection \(REGPAR\)](#)” on page [393](#)
- “[On-Chip Command and Address Path Protection \(OCCAP\)](#)” on page [398](#)
- “[On-Chip External SRAM Address Protection \(OCSAP\)](#)” on page [406](#)
- “[DFI Sideband Watchdog Timers](#)” on page [408](#)

13.1 Memory ECC

This section discusses the features of memory error correction code (ECC).

It contains the following subsections:

- “[Inline ECC Support](#)” on page 334

13.1.1 Inline ECC Support

This topic contains the following sections:

- “[Overview of Inline ECC Support](#)” on page 334
- “[Enabling Inline ECC](#)” on page 335
- “[Address Map](#)” on page 335
- “[Selectable Protected Regions](#)” on page 342
- “[Locking of ECC Region](#)” on page 345
- “[Error Injection Through Software \(ECC Data Poisoning\)](#)” on page 346
- “[Related Hardware Parameters](#)” on page 350
- “[Command Flow](#)” on page 351
- “[Address Protection with ECC \(ECCAP\)](#)” on page 359
- “[Scrubbing](#)” on page 362
- “[QoS](#)” on page 362
- “[Features and Limitations](#)” on page 363
- “[Interrupts Related to Inline ECC](#)” on page 365
- “[Signals Related to Inline ECC](#)” on page 365
- “[Registers Related to Inline ECC](#)” on page 367

13.1.1.1 Overview of Inline ECC Support

The following are the features of Inline ECC:

- ECC parity (code) is stored together with the data without using a dedicated sideband memory device.
- Inline ECC is a necessity for LPDDR4/5 due to device characteristics and device topology.
 - Inline ECC is supported with LPDDR4 and LPDDR5 (BG rotation address mapping) protocols.
- The supported memory data widths are 16 and 32.
- For Inline ECC, 64/8 SECDED Hamming code is used:
 - Data to ECC ratio is 8/1.
- Inline ECC requires Data Mask (DM) to be enabled.
- RMW command is required when a write access cannot fill one Hamming code (64 bits), For details, see “[Read-Modify-Write \(RMW\) Generation](#)” on page 48.

13.1.1.2 Enabling Inline ECC

To enable Inline ECC, set the hardware configuration parameter `MEMC_INLINE_ECC` to '1'.



- Inline ECC feature is supported in limited configurations. When Inline ECC is enabled in hardware configuration, sideband ECC cannot be enabled.
- `ECCCFG0.ecc_type` register is don't care in Inline ECC configurations.
- Inline ECC feature cannot be supported with `MEMC_BURST_LENGTH=32`. Therefore, `MEMC_BURST_LENGTH=32` and `MEMC_INLINE_ECC=1` are exclusive.

For details about this option, see “HW Configuration / DDRC Parameters” in Parameter Descriptions chapter.



- Inline ECC requires Data Mask, hence, `DBICTL.dm_en` must be set to '1' (and corresponding mode register in SDRAM) whenever inline ECC is enabled by software.
For devices or DIMMs which do not support Data Mask, Inline ECC cannot be used.

13.1.1.3 Restriction on Data Mask When ECC is Used

Each data lane (memory data width) in the DDRCTL is 16, 32, or 64 bits. If ECC is enabled, the controller can do a write operation if the write is aligned to memory data width. If the granularity of the write is smaller than memory data width, the controller must issue an RMW command. In this case, the DDRCTL first fetches the read data from the DRAM and merges with the write data from the controller, and then does a write with no bytes masked.

13.1.1.4 Address Map

Definitions:

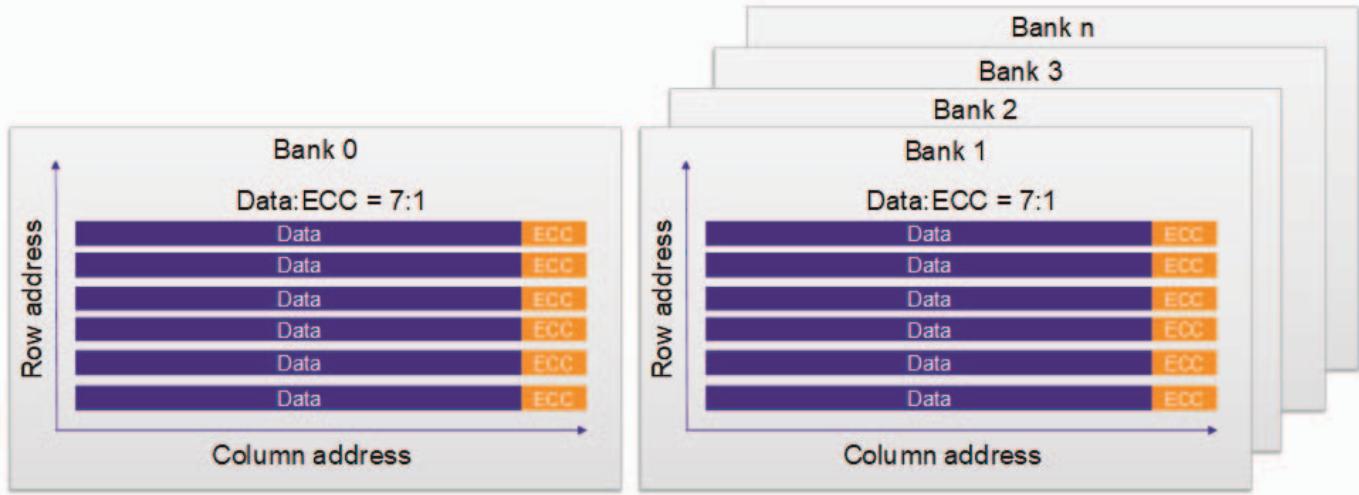
- Data Block: 8 burst accesses of data (a burst can be DRAM BL8 or BL16).
- Super-block: Block with ECC parity (9 burst accesses).
- Giant-block: Depending on address map, a transaction sequence can be divided into 8 data + 1 ECC, 16 data + 2 ECC, 32 data + 4 ECC (unit is one burst access).
- Hole: ECC parity + wasted region of the memory.
- Usable Memory Size: All the data blocks that fits.
- Usable Size Ratio: Usable memory size/Full memory size.
- Utilization Percentage: The percentage of memory used for Data and ECC.

When Inline ECC is enabled, the highest three column bits must be mapped to the highest address map position possible. The controller flexible address mapping scheme is constrained so that the highest system address space is reserved for ECC parity and waste as one region as shown in [Figure 13-1](#). For the normal data in all remaining regions, the system address is linear and continuous.

The reason for the waste is due to how the memory is divided for ease of implementation. Data to ECC ratio is 8 to 1 (divide by 9), but the memory is mapped 7 to 1 (divide by 8 which is a power of 2).

- Usable size ratio is $7/8=87.5\%$.
- Utilization percentage is $(7/8 + (1/8 * 7/8)) = 63/64 = 98.4\%$ and the remaining becomes the waste. The unprotected wasted area can still be accessible by the system, but not in linear and continuous fashion.

Figure 13-1 ECC Code Location – Upper 1/8 of Each Column Address



System to SDRAM Address Example

This is an example for LPDDR4 (16Mb x 16DQx 8 banks, single channel):

- Column width is 10bits. Number of columns=1024 (0x400).
- Row width is 14bits, Number of rows=16384 (0x4000).
- Bank width is 3bits, Number of banks=8(0x8).
- Total SDRAM address=1024 * 16384 * 8=128M words, DQ width is 16bits. Therefore, memory density per channel is 256M bytes.

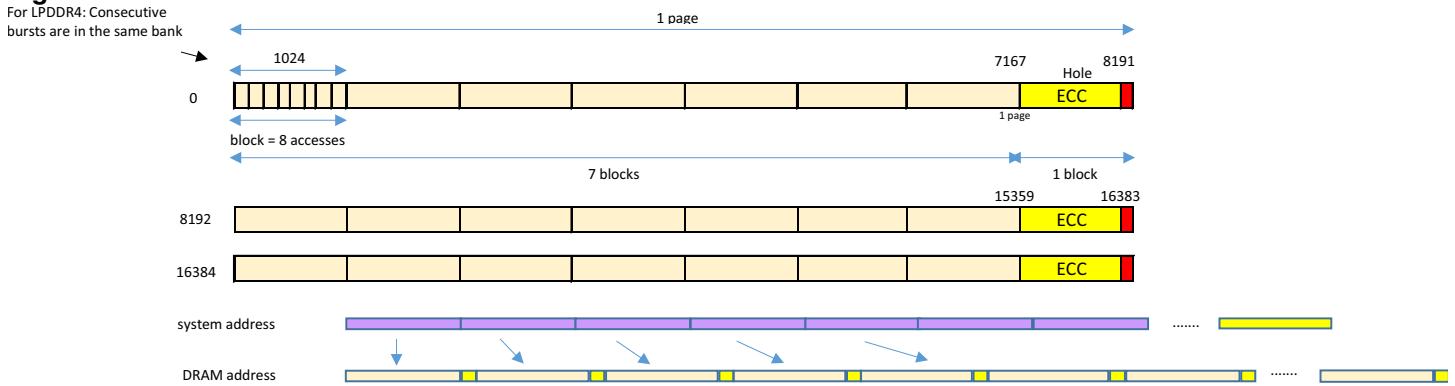
Four instances of the above LPDDR4 device can be combined to give 64 bits of DRAM data width.

- MEMC_DRAM_DATA_WIDTH=64 bits (four LPDDR4 X16 devices)¹
- MEMC_BURST_LENGTH=16
- 1 page=8192 bytes (1024 columns)
- 1 access (burst) is 128 bytes (BL16)
- 1 block=8 accesses (1024 bytes)
- For each access, 16 bytes of ECC parity is required (1 column).

All ECC parity information for a full block (that is, 8 BL16 data accesses) is tightly packed into one ECC access (that is, 1 BL16).

Figure 13-2 shows the space usage or data and ECC code, it is a high-level description to explain what the result (where is data, where is ECC from both AXI address and SDRAM address perspective). The red zones are the wasted regions.

1. MEMC_DRAM_DATA_WIDTH must be set to '16' or '32' in LPDDR5/4/4X Controller, so this cannot be supported.

Figure 13-2 Inline ECC Address Translation

In the previous example:

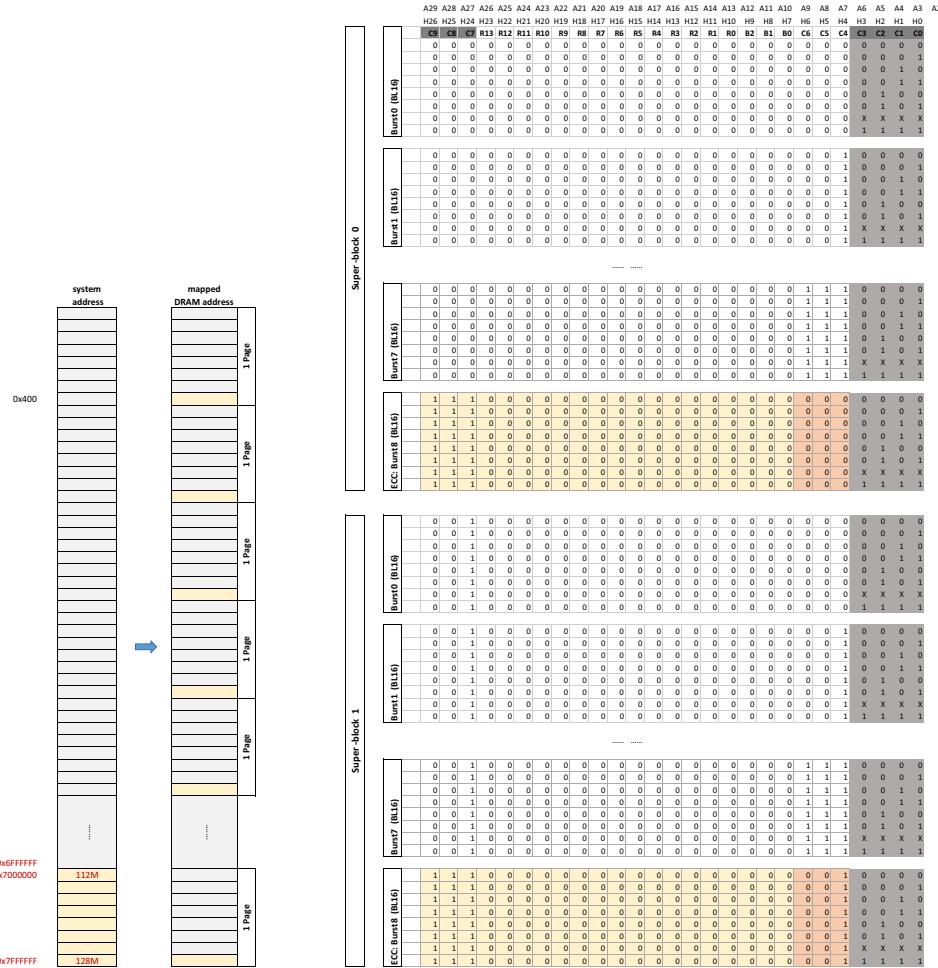
1. {C9, C8, C7} (or highest three columns) must map to the MSB of system address.
2. No special address translation for data.
3. ECC address is in term of HIF address, that is generated by replacing col[6:4] (offset address in one block) with the highest three hif_addr (C9, C8, C7), then set the highest three hif_addr (C9, C8, C7) to 3'b111. Set the lowest 4 bit to 4'b0000. An example format is 3'b111, (R13...R0), (B2...B0), ({C9...C7}), 4'b0000.
4. BG/Bank address bits can be mapped anywhere between {C9, C8, C7} and {C3,...,C0} groups. It affects Bank/Bank-Group rotation, for more information, refer to "Address Map Considerations for Inline ECC". In this example, there is no Bank/Bank-Group rotation.
5. Rank/Row/Column address bits can be mapped with the limitations described in "Address Map Limitation".



For LPDDR5, as LPDDR5 has burst address (B3 to B0), C3 to C0 is replaced by B3 to B0 and C9 to C4 is replaced by C5 to C0.

Figure 13-3 shows the detailed address mapping for system address (AXI) and SDRAM address in the example above. A* stands for system address, H* stands for HIF address. For more information, refer to “Application to HIF Address Mapping” on page 157.

Figure 13-3 Inline ECC Address Map Example (BL16)



Address Map Considerations for Inline ECC

The internal mapping of ECC blocks are such that data and corresponding ECC are always mapped to the same page (that is, same bank and row). This is required to achieve the highest scheduling efficiency for all protocols. Therefore, the address mapping decision for the banks and bank groups (the position of mapping) impact the way incoming sequential commands are divided into internal ECC blocks. This is done automatically by the controller. For example, if 8 (or 16) sequential commands are mapped to 2 banks, there are 2 partial (full) ECC blocks with 2 ECC overhead comments.

To achieve good performance, when setting address mapping, you must consider whether you need banks or bank groups to rotate within 8 HIF transaction sequence, what is the size of one “transaction sequence”, how many “transaction sequences” are interleaving. This helps to determine how many block channel resources are required.

Recommendation is to satisfy the following two conditions:

- Number of block channels \geq (Giant-block size/8) \times (number of interleaving sequences) $\times 2$
Where $\times 2$ is for read/write interleaving
- Giant-block size == (Number of ECC blocks belong to a Giant block) $\times 8$.

If bank/bank-group is rotating within ECC block size (that is, 8 HIF commands), multiple ECC blocks are interleaved internally and then a Giant block has 2 or 4 ECC blocks, otherwise a Giant block has an ECC block.

For more details, see [Figure 13-4](#), [Figure 13-5](#), and [Figure 13-6](#), and [Table 13-1](#).

Figure 13-4 1 Giant Block = 1 ECC Block = 8 HIF Commands

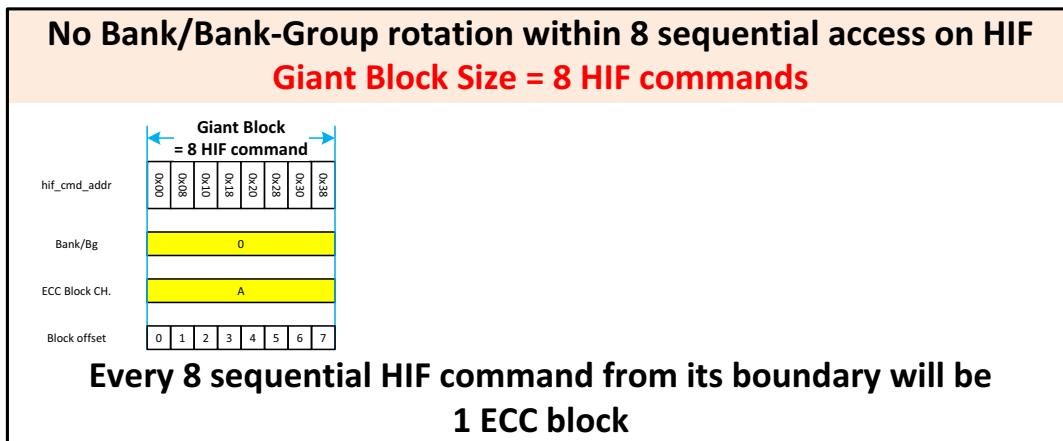


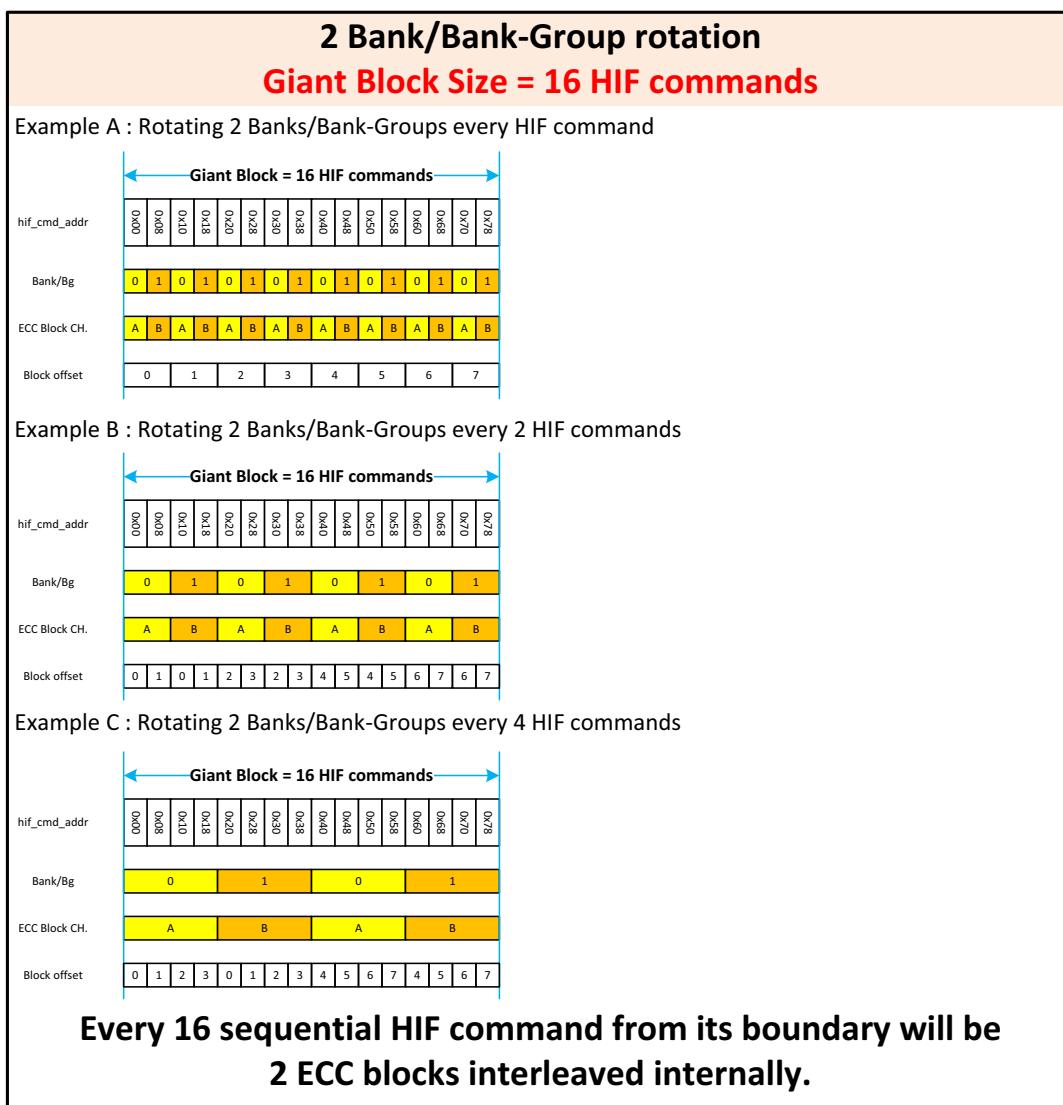
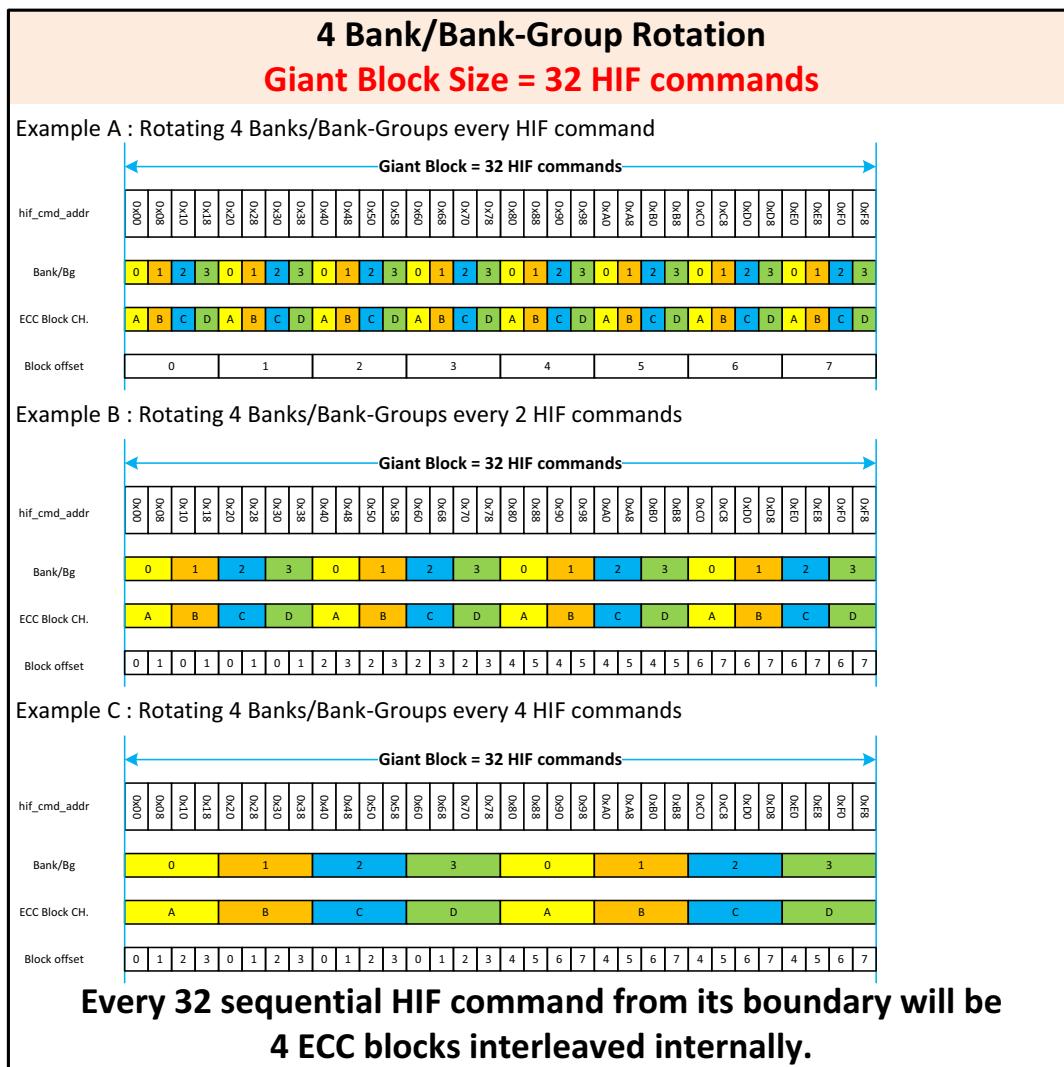
Figure 13-5 1 Giant Block = 2 ECC Blocks = 16 HIF Commands

Figure 13-6 1 Giant block = 4 ECC blocks = 32 HIF Commands

- For MEMC_BURST_LENGTH=16, burst_rdwr=BL16, LPDDR4 case.
 - For no bank rotation (within 8 HIF transaction sequence), COL[6:4] must be mapped to HIF[6:4]. In this case, one transaction sequence (size is 8 BL16) is 1 block access, which requires 1 block channel.
 - For 2 bank rotation, COL[6] must be mapped to HIF[7], COL[5:4] and BANK[0] can be mapped to anywhere within HIF[6:4] (C5, C4 relative ordering must be preserved). In this case, one transaction sequence (size is 8 BL16) is 2 interleaving block accesses, which require 2 block channels.
 - For 4 bank rotation, COL[6] must be mapped to HIF[8], COL[5:4] and BANK[1:0] can be mapped to anywhere within HIF[7:4] (C5, C4 relative ordering must be preserved). In this case, one transaction sequence (size is 8 BL16) is 4 interleaving block accesses, which require 4 block channels.
 - 8 bank rotation is not recommended. In that case, a transaction sequence (size is 8 BL16) causes 8 interleaving block accesses and consume 8 block channels. Following examples illustrate the need for many block channels which may not be available:

- Example 1: You must issue small sized transaction sequence which generates more overhead commands and needs more block channels. If you do not have enough block channels, that generates more override channel condition.
- Example 2: If you have more than one interleaving sequence, that needs more block channels. If you do not have enough block channels, that generates more override channel condition.

Table 13-1 **Inline ECC Address Mapping**

MEMC_BURST_LENGTH=16																		
NO Bank rotation																		The addresses within the highlighted boxes can be mapped in any combination with the following rules:
For LPDDR4:																		
*C6 must be the highest																		
*C6, C5, C4 relative ordering must be preserved																		
C6:4 is block offset address																		
C3:0 is start address for one burst																		
For LPDDRS:																		
*C2 must be the highest																		
*C2, C1, C0 relative ordering must be preserved																		
C2:0 is block offset address																		
B3:0 is start address for one burst																		
The others are block addresses																		
NO BG rotation																		
ECC address is calculated by DRAM address as follows:																		
For LPDDR4:																		
Col[Highest :- 3] <= 3'b111																		
Col[6:4] <= Col[Highest :- 3]																		
Col[3:0] <= 4'b0000																		
For LPDDRS:																		
CJ[Highest :- 3] <= 3'b111																		
CJ[2:0] <= C[Highest :- 3]																		
B[3:0] <= 4'b0000																		
Remaining addresses are pass-through																		
H0, H1, H2: the highest addressable HIF based on DRAM size																		
LPDDR4																		
2 Bank rotation																		
LPDDR5																		
4 Bank rotation																		
8 Bank rotation (not needed, not realistic)																		
2 BG rotation																		
LPDDRS																		
4 BG rotation																		
8 Bank rotation (not needed, not realistic)																		

13.1.1.5 Selectable Protected Regions

Enable ECC protection on the critical area of the memory. Remaining addresses can be accessed with no performance overhead.

- To maintain uninterrupted linear system address space, ECC region maps to the top of the system address, ECC region size is 1/8 memory size.
- System address space is divided into 8/16/32/64 regions. The granularity is determined by register `ECCCFG0.ecc_region_map_granu[1:0]`. Note, that highest 1/8 region is always ECC region.
- Lowest 7 regions are Selectable Protected Regions. The Selectable Protected Regions can be protected/non-protected selectively by `ECCCFG0.ecc_region_map[6:0]`. Other upper regions are protected/non-protected region, determined by `ECCCFG0.ecc_region_map_other`. Each bit of `ECCCFG0.ecc_region_map[6:0]` correspond to each of lowest 7 regions respectively. To protect a region, set the corresponding bit to '1', otherwise set to '0'. See [Figure 13-7](#) for more details.
- When the region is set to non-protected, ECC overhead commands are not injected and therefore, there is no performance drop.

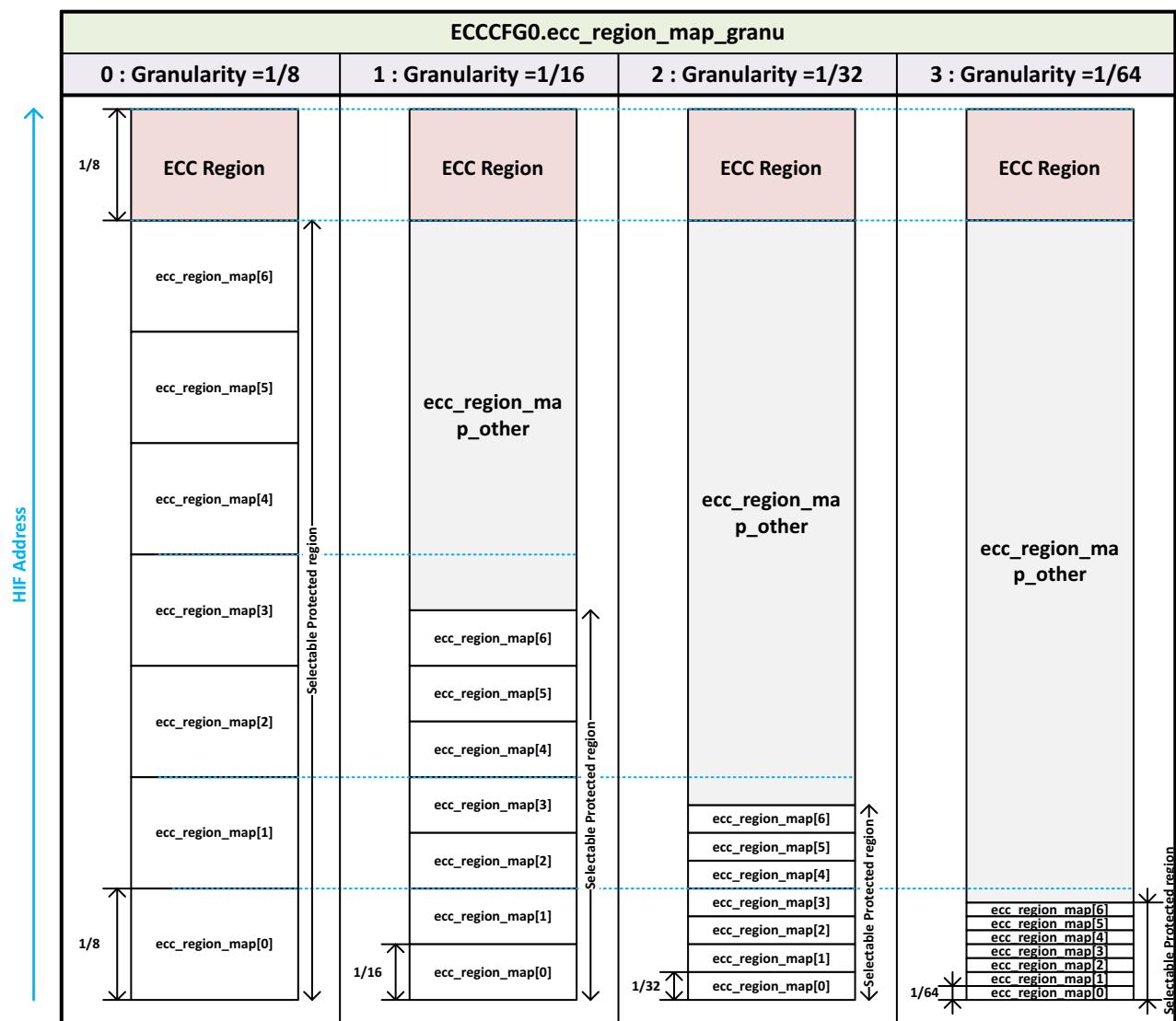
Figure 13-7 Selectable Protected Region

Figure 13-8 ECC Regions Example

1) ECCCFG0.ecc_region_map_granu=0 (1/8)

System Address bit	A32	A31	A30	A29	A28	A27	A26-A0		
SDRAM Address	C9	C8	C7	R13	R12	R11	Any		
	0	0	0	X	X	X	X	Protected	ecc_region_map[0]=1
	0	0	1	X	X	X	X	Non-Protected	ecc_region_map[1]=0
	0	1	0	X	X	X	X	Protected	ecc_region_map[2]=1
	0	1	1	X	X	X	X	Protected	ecc_region_map[3]=1
	1	0	0	X	X	X	X	Non-Protected	ecc_region_map[4]=0
	1	0	1	X	X	X	X	Non-Protected	ecc_region_map[5]=0
	1	1	0	X	X	X	X	Protected	ecc_region_map[6]=1
	1	1	1	X	X	X	X	ECC region	

2) ECCCFG0.ecc_region_map_granu=1 (1/16)

System Address	A32	A31	A30	A29	A28	A27	A26-A0		
SDRAM Address	C9	C8	C7	R13	R12	R11	Any		
	0	0	0	0	X	X	X	Protected	ecc_region_map[0]=1
	0	0	0	1	X	X	X	Non-Protected	ecc_region_map[1]=0
	0	0	1	0	X	X	X	Protected	ecc_region_map[2]=1
	0	0	1	1	X	X	X	Protected	ecc_region_map[3]=1
	0	1	0	0	X	X	X	Non-Protected	ecc_region_map[4]=0
	0	1	0	1	X	X	X	Non-Protected	ecc_region_map[5]=0
	0	1	1	0	X	X	X	Protected	ecc_region_map[6]=1
	0	1	1	1	X	X	X	Non-Protected	
	1	0	X	X	X	X	X	Non-Protected	
	1	1	0	X	X	X	X	Non-Protected	
	1	1	1	X	X	X	X	ECC region	

3) ECCCFG0.ecc_region_map_granu=3 (1/64)

System Address	A32	A31	A30	A29	A28	A27	A26-A0		
SDRAM Address	C9	C8	C7	R13	R12	R11	Any		
	0	0	0	0	0	0	X	Protected	ecc_region_map[0]=1
	0	0	0	0	0	1	X	Non-Protected	ecc_region_map[1]=0
	0	0	0	0	1	0	X	Protected	ecc_region_map[2]=1
	0	0	0	0	1	1	X	Protected	ecc_region_map[3]=1
	0	0	0	1	0	0	X	Non-Protected	ecc_region_map[4]=0
	0	0	0	1	0	1	X	Non-Protected	ecc_region_map[5]=0
	0	0	0	1	1	0	X	Protected	ecc_region_map[6]=1
	0	0	0	1	1	1	X	Non-Protected	
	0	0	1	X	X	X	X	Non-Protected	
	0	1	X	X	X	X	X	Non-Protected	
	1	0	X	X	X	X	X	Non-Protected	
	1	1	0	X	X	X	X	Non-Protected	
	1	1	1	X	X	X	X	ECC region	

In the three examples of [Figure 13-8](#), where number of system (HIF) address bits is 33, the highest column address bit is column[9], `ECCCFG0.ecc_region_map=7'b1001101` and `ECCCFG0.ecc_region_map_granu=0,1,3` respectively.

In [Figure 13-8](#), the:

- Blue regions are ECC protected,
- White regions are non-ECC protected, and
- Red region are ECC region.



Note All regions are protected with the following setting (default):

- `ECCCFG0.ecc_region_map=7'b1111111`
- `ECCCFG0.ecc_region_map_granu=0`
- `ECCCFG0.ecc_region_map_other=0`

13.1.1.6 Locking of ECC Region

The ECC region comprises two separate sub-regions:

1. Used sub-region:
 - ❑ ECC parity area for ECC protected regions.
2. Unused sub-region:
 - ❑ ECC parity area for unprotected regions (reserved but not used by inline ECC logic).
 - ❑ Waste region, which is never used by inline ECC logic.

The used sub-region is locked/unlocked by the register `ECCCFG1.ecc_region_parity_lock`.

The unused sub-region is locked/unlocked by the register `ECCCFG1.ecc_region_waste_lock`.

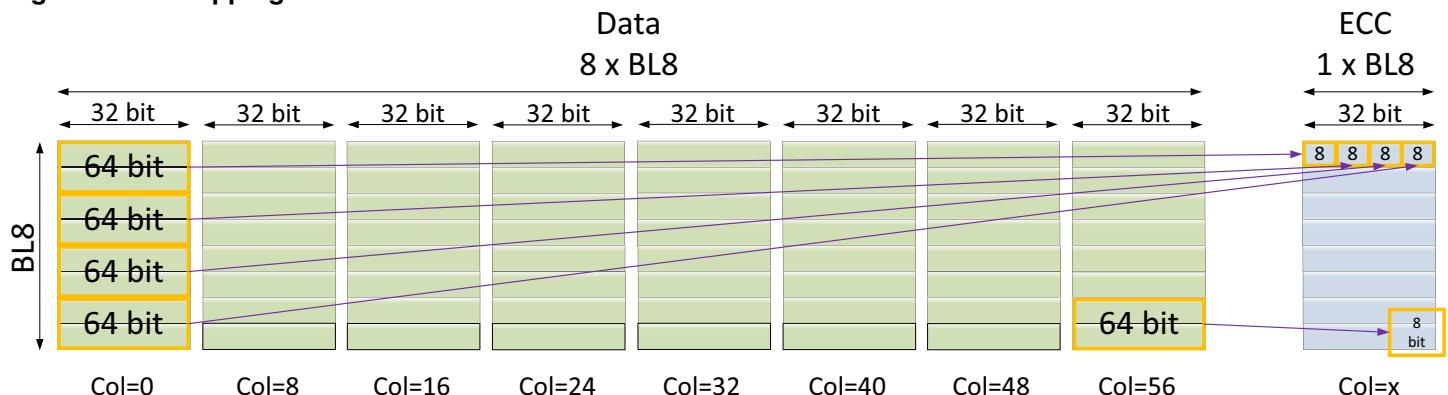


For more information on identifying Used and Unused region, see “[Formula of Used and Unused Region](#)” on page [349](#).

Used and unused regions can be identified by the HIF address.

```
// hif_addr_h3 means the highest valid 3 bits of HIF address.
assign ecc_region_map_ext={1'b0, reg_ddrc_ecc_region_map};
assign offset_addr=~MEMC_BURST_LENGTH==8 ? col[5:3] : col[6:4];
assign hif_ecc_addr_sep={offset_addr,hif_cmd_addr[highest_col_bit-3 -: 3]} >>
(3-reg_ddrc_ecc_region_map_granu);
assign ecc_region_used=reg_ddrc_ecc_mode==3'b000 ? 1'b0 : (&hif_addr_h3) & ~| hif_ecc_addr_sep[5:3] &
ecc_region_map_ext[hif_ecc_addr_sep[2:0]];
assign ecc_region_unused=reg_ddrc_ecc_mode==3'b000 ? 1'b0 : (&hif_addr_h3) &
~(~| hif_ecc_addr_sep[5:3] & ecc_region_map_ext[hif_ecc_addr_sep[2:0]]);
```

[Figure 13-9](#) shows the mapping between the Data and the ECC code. Regardless of the memory data width, 64/8 SECDED is used.

Figure 13-9 Mapping Between Data and ECC

In normal conditions, the user-software must not access the Used region of the ECC hole and therefore must be kept locked. However, for debugging or error injection purposes into ECC parity, it can be unlocked by the `ecc_region_parity_lock` register.

If required, for normal transfers, the user-software can access the Unused region of the ECC hole and therefore can be kept unlocked using the `ecc_region_waste_lock` register. However, it must be noted that the unused region is not consecutive in terms of system addresses and it is interleaved with the used region.

The unused region can be made consecutive by using the ECC region remap functionality. For more information, see “[ECC Region Remap](#)” on page [346](#).

Locking the ECC hole sub-regions from access means the following:

- Any write or RMW request to this region is discarded, while the HIF output `hif_wdata_ptr_addr_err` is asserted.
- Any read request to this region is executed to the memory, but without the ECC check and returns all zeros, while the output `hif_rdata_addr_err` is asserted.

For AXI configurations, any assertion of `hif_wdata_ptr_addr_err` or `hif_rdata_addr_err` also results in SLVERR/ERROR response.

When Inline ECC is disabled in the software, all the memory space can be accessed.

13.1.1.7 Error Injection Through Software (ECC Data Poisoning)

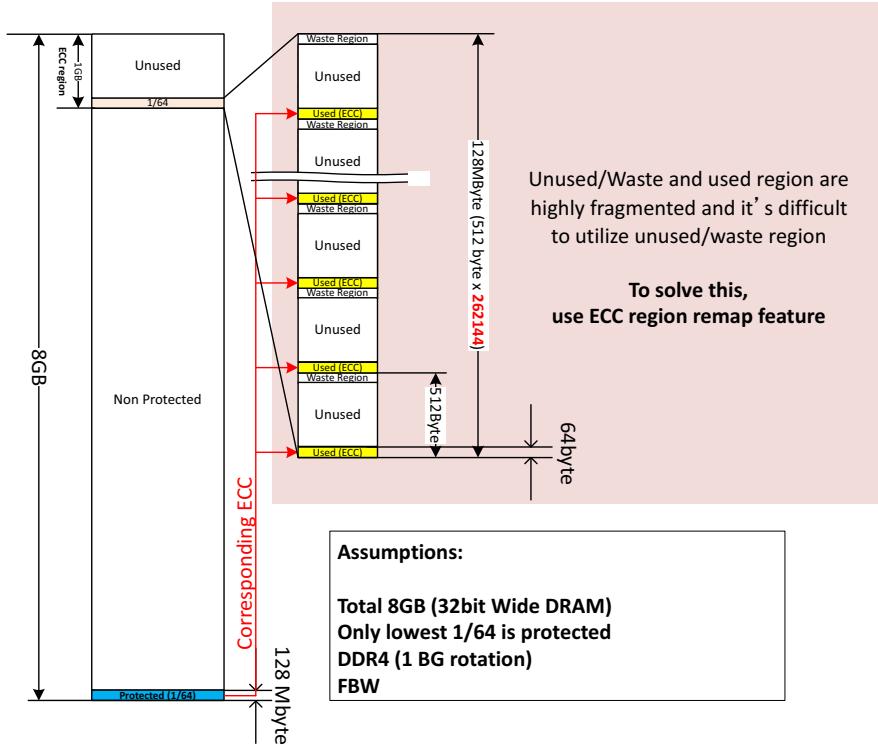
ECC error injection is useful for system-level software validation. Unlike in sideband ECC, there is no dedicated hardware support for it. However, errors can be easily injected through the software by unlocking the ECC region through the `ecc_region_parity_lock` register and overriding certain ECC parity bits. When the corresponding addresses are read from a protected memory region, ECC errors are generated as correctable or uncorrectable, depending on the type of error introduced. For more details, see section “[Locking of ECC Region](#)” on page [345](#).

13.1.1.8 ECC Region Remap

The unused sub-region can be used as unprotected region. However, it can be highly fragmented due to ECC mapping method (that is, ECC and Data are mapped to the same page), and it may be difficult to utilize the unused/wasted region.

Figure 13-10 shows an example of the fragmentation.

Figure 13-10 Example of Fragmentation



Enabling ECC region remap feature (setting `ECCCFG0.ecc_region_remap_en` to '1') can make the unused ECC region consecutive. When ECC region remap feature is enabled, ECC region is remapped as shown in Figure 13-11.

Figure 13-11 ECC Region Remapping

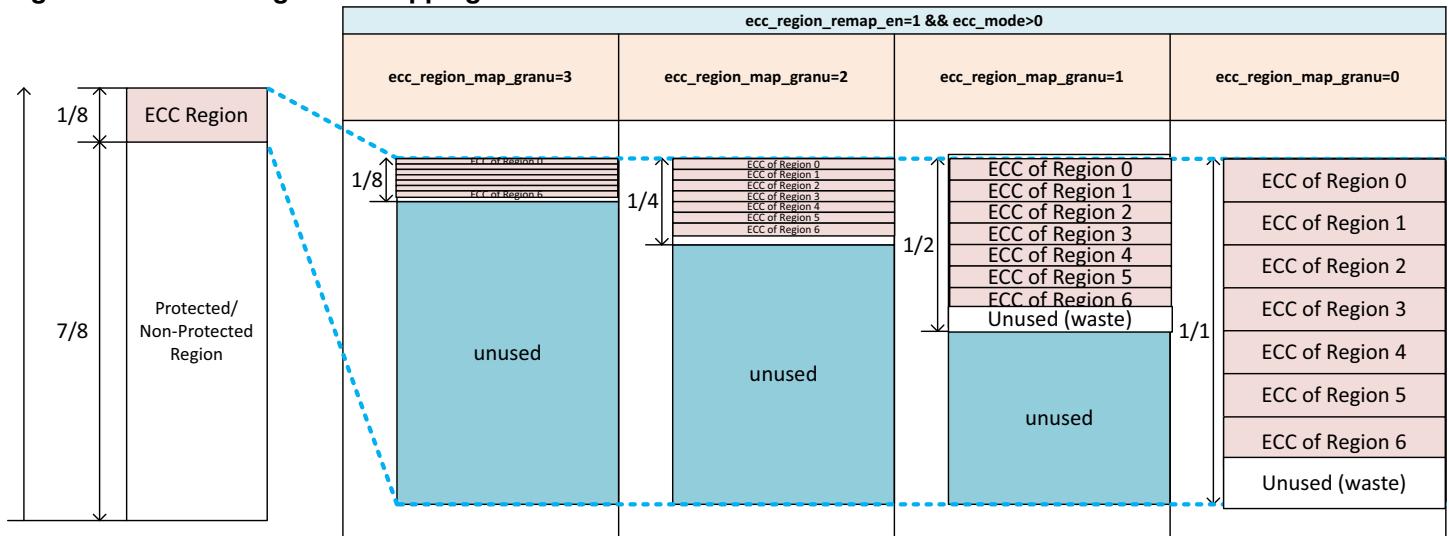


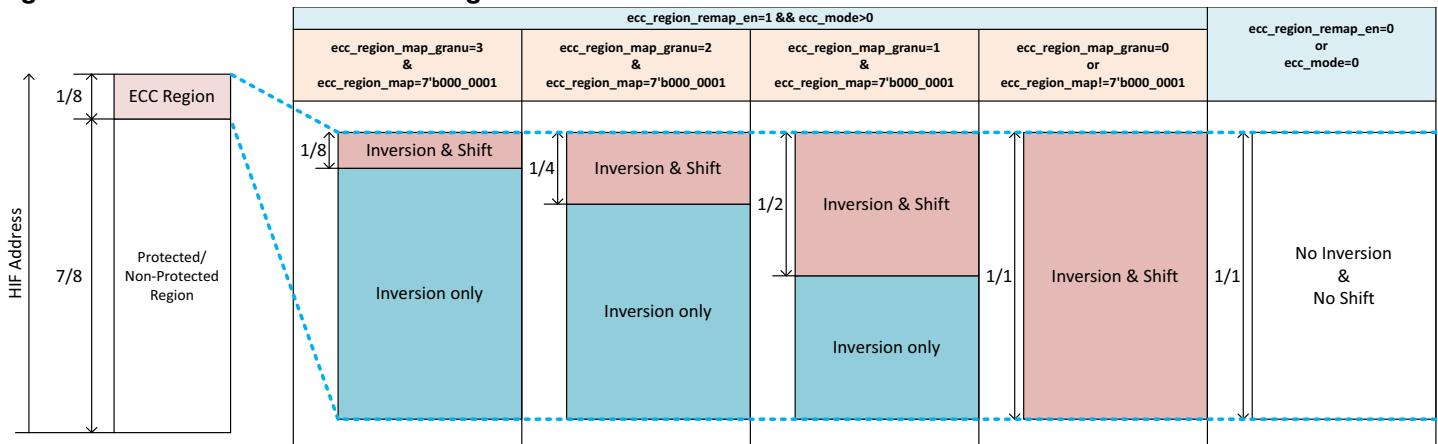
Figure 13-11 is for `ECCCFG0.ecc_region_map_other=0`. When `ECCCFG0.ecc_region_map_other=1`, “unused” in the figure is replaced with “used”.

To optimize performance LPDDR5X/5/4X Memory Controller has two types of remapping in ECC region, which are used for different parts of the ECC region.

- Inversion and Shift: Address map is not optimized for performance because C6...C4 are mapped to higher bits of HIF address with remap enabled.
- Inversion only: Performance is equivalent to other regions in terms of address map.

See [Figure 13-12](#) performance of ECC region, this shows the mapping for “Inversion & Shift” and “Inversion only” which depends on `ECCCFG0.ecc_region_map_granu`, `ECCCFG0.ecc_region_map` and `ECCCFG0.ecc_region_remap_en`.

Figure 13-12 Performance of ECC Region



When ECC region remap is enabled (`ECCCFG0.ecc_region_remap_en=1`), ECC address can be calculated from the HIF command address (`hif_cmd_addr`) by the formula of ECC address, instead of the original calculation method in [Table 13-1](#) on page 342:

Formula of ECC address when `ECC_region_remap_en=1`

If $x > y$,

```
hif_ecc_addr={3'b111, ~hif_cmd_addr[hif_highest_bit -: 6],  
hif_cmd_addr[hif_highest_bit-6 : x+3], hif_cmd_addr[x-1:y], {hif_cmd_addr[x+2:x],  
hif_cmd_addr[y-1:0]}>>3}
```

otherwise

```
hif_ecc_addr={3'b111, ~hif_cmd_addr[hif_highest_bit -: 6],  
hif_cmd_addr[hif_highest_bit-6 : x+3], {hif_cmd_addr[x+2:x],  
hif_cmd_addr[y-1:0]}>>3}
```

Where x is derived from HIF address bits that map to column 4 and $y=4$.

`hif_highest_bit` indicates the highest valid bit of HIF address.

When ECC region remap is enabled (`ECCCFG0.ecc_region_remap_en=1`), there is a limitation for the address mapping of Inline ECC.



Note Block offset address must be mapped to the consecutive HIF bits. That is, C6...C4 must be mapped to consecutive HIF bits in `MEMC_BURST_LENGTH=16`. C5...C3 must be mapped to consecutive HIF bits in `MEMC_BURST_LENGTH=8`.

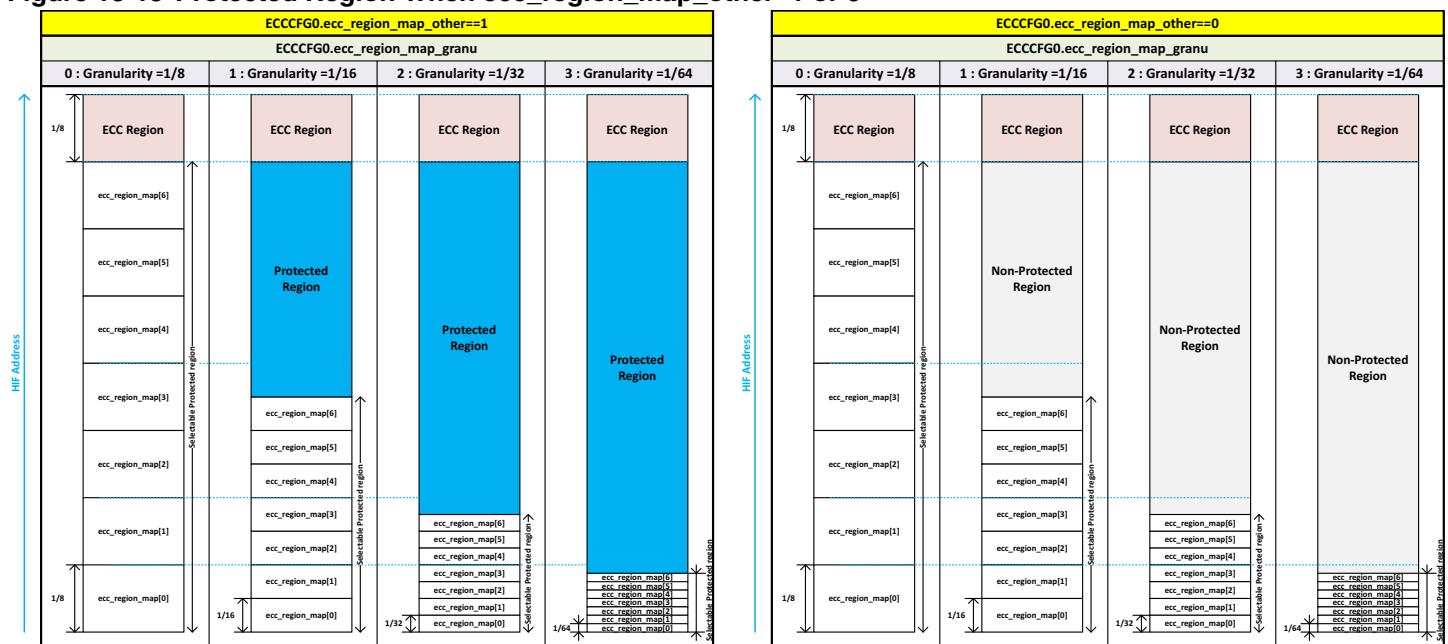
13.1.1.9 ECC Region Map Others

When `ECCCFG0.ecc_region_map_granu!=0`, there is a region that is not controlled by `ECCCFG0.ecc_region_map`. Set `ECCCFG0.ecc_region_map_other` to '1' to make this a protected region. In this way, you can choose to have a small unprotected region (such as 1/64 of the memory space), while protecting the remainder.

When `ECCCFG0.ecc_region_map_other` is '1', the ECC region remap can be enabled as well. In this case all the ECC region are remapped as "Inversion & Shift".

See [Figure 13-13](#) about protected region when `ECCCFG0.ecc_region_map_other` is '1' or '0'.

Figure 13-13 Protected Region When `ecc_region_map_other=1` or `0`



13.1.1.10 Formula of Used and Unused Region

The following equations are defined in terms of the HIF address (`hif_cmd_addr`), the used and unused regions in different cases (depending on `ECCCFG0.ecc_region_remap_en` and `ECCCFG0.ecc_region_map_other`). In each case, if `ecc_region_used` evaluates as true, the HIF address is part of the used region. If `ecc_region_unused` evaluates as true, the HIF address is part of the unused region.

When `ECCCFG0.ecc_region_remap_en=0` and `ECCCFG0.ecc_region_map_other=0`:

```
// hif_addr_h3 means the highest valid 3 bits of HIF address.
ecc_region_map_ext={1'b0, reg_ddrc_ecc_region_map}; assign
offset_addr=~MEMC_BURST_LENGTH==8 ? col[5:3] : col[6:4];
hif_ecc_addr_sep={offset_addr,hif_cmd_addr[highest_col_bit-3 -: 3]} >>
(3-reg_ddrc_ecc_region_map_granu); ecc_region_used=reg_ddrc_ecc_mode==3'b000 ?
1'b0 : (&hif_addr_h3) & ~|hif_ecc_addr_sep[6:3] &
ecc_region_map_ext[hif_ecc_addr_sep[2:0]];
ecc_region_unused=reg_ddrc_ecc_mode==3'b000 ? 1'b0 : (&hif_addr_h3) &
~(|hif_ecc_addr_sep[6:3] & ecc_region_map_ext[hif_ecc_addr_sep[2:0]]);
```

When `ECCCFG0.ecc_region_remap_en=0` and `ECCCFG0.ecc_region_map_other=1`:

```
// hif_addr_h3 means the highest valid 3 bits of HIF address.
ecc_region_map_bit7=~|reg_ddrc_ecc_region_map_granu ? 1'b0 :
reg_ddrc_ecc_region_map_other;ecc_region_map_ext={ecc_region_map_bit7,
reg_ddrc_ecc_region_map};offset_addr=~MEMC_BURST_LENGTH==8 ? col[5:3] : col[6:4];
hif_ecc_addr_sep={offset_addr,hif_cmd_addr[highest_col_bit-3 -: 3]} >>
(3-reg_ddrc_ecc_region_map_granu); ecc_region_used_sep = ~|hif_ecc_addr_sep[5:3]
& ecc_region_map_ext[hif_ecc_addr_sep[2:0]];ecc_region_used_other=
|hif_ecc_addr_sep[5:3] & ~&offset_addr &
reg_ddrc_ecc_region_map_other;ecc_region_used=reg_ddrc_ecc_mode==3'b000 ? 1'b0 :
(&hif_addr_h3) & (ecc_region_used_sep | ecc_region_used_other);ecc_region_unused
=reg_ddrc_ecc_mode==3'b000 ? 1'b0 : (&hif_addr_h3) & ~ecc_region_used_sep |
ecc_region_used_other);
```

When `ECCCFG0.ecc_region_remap_en=1` and `ECCCFG0.ecc_region_map_other=0`:

```
ecc_region_used =(&hif_cmd_addr[highest -: (3+granu)]) &&
reg_ddrc_ecc_region_map[~hif_ecc_addr[highest-(3+granu) -:3
]];ecc_region_unused=(&hif_cmd_addr[highest -: 3]) && ~ecc_region_used
```

When `ECCCFG0.ecc_region_remap_en=1` and `ECCCFG0.ecc_region_map_other=1`:

```
ecc_region_map_bit7=~|ecc_region_map_granu ? 1'b0 :
reg_ddrc_ecc_region_map_other;ecc_region_map_ext={ecc_region_map_bit7,
reg_ddrc_ecc_region_map};ecc_region_used =(~((&hif_cmd_addr[highest -: (3+granu)]) & ecc_region_map_ext[~hif_ecc_addr[highest-(3+granu) -:3]]) ||
(~reg_ddrc_ecc_region_map_other & (&hif_cmd_addr[highest -: 3])) &
(~&hif_cmd_addr[highest-3 -: granu]) & (~hif_cmd_addr[highest-3-:3])) );
```

13.1.1.11 Related Hardware Parameters

The following are the new hardware configuration parameters that are available in the GUI:

- `MEMC_INLINE_ECC`: enables Inline ECC.
- `MEMC_SIDEBAND_ECC`: enables Sideband ECC (mutually exclusive with Inline ECC).
- `MEMC_ECCAP`: enables address protection within Inline ECC.
- `MEMC_NO_OF_BLK_CHANNEL`: interleaving depth. Indicates the number of blocks that can be interleaved at DDRC input (HIF). Valid values are 4, 8, 16, and 32.

The following are the new hardware configuration parameters that are not available in the GUI (derived parameters):

- `MEMC_NO_OF_BLK_TOKEN`: indicates number of blocks that can be maintained in DDRC. Equals to `MEMC_NO_OF_ENTRY + MEMC_NO_OF_BLK_CHANNEL`. Though this number is optimized, it does not consider the read latency. There could be corner cases where block tokens could be full before CAM is full and cause `hif_cmd_stall` to be asserted, and this number links the read (BRT) and write (BWT) tokens.
- `MEMC_NO_OF_BRT`: indicates the read cache (ECC array) depth which is equal to `MEMC_NO_OF_ENTRY/2 + MEMC_NO_OF_BLK_CHANNEL`.
- `MEMC_NO_OF_BWT`: indicates the write cache (ECC accumulator) depth which is equal to `MEMC_NO_OF_ENTRY/2 + MEMC_NO_OF_BLK_CHANNEL`.

Debug Port

Table 13-2 shows the output signals for debugging.

Table 13-2 Output Signal for Debugging

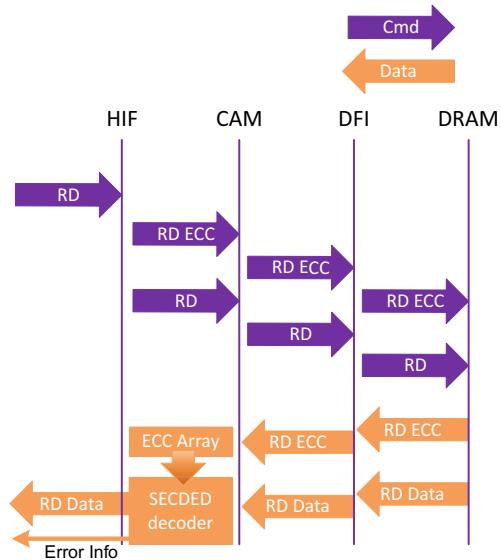
Name	Width	IO	From/To	Description
dbg_dfi_ie_cmd_type	[2:0]	O	From DFI	<p>Debug signal. This signal is valid when dfi_cs is valid and command is RD/RDA/WR/WRA, otherwise don't care</p> <p>Command == RD/RDA</p> <ul style="list-style-type: none"> ■ 000: RD_N (RD Data for non-protected region) ■ 001: RD_E (RD Data for protected region) ■ 010: RE_B (RD ECC in block read/write) <p>Command == WR/WRA</p> <ul style="list-style-type: none"> ■ 000: WD_N (WR Data for non-protected region) ■ 001: WD_E (WR Data for protected region) ■ 010: WE_BW (WR ECC in block write) ■ 111: MPR write (DDR4 only)

13.1.1.12 Command Flow

The controller fully manages the injection and scheduling of ECC commands which are also called the “overhead commands”. This section discusses the data structures and the flow of ECC commands.

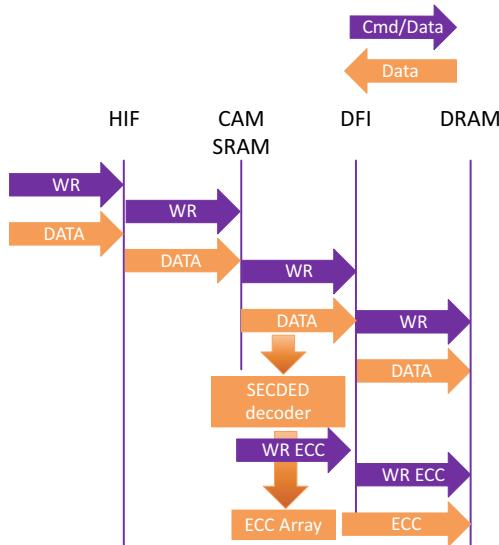
When a Read (Data) command is incoming from HIF, the Read ECC command is generated internally as necessary. For example, if the Read ECC is already stored in the ECC Array, then Read ECC command is not required. The ECC address is calculated from the HIF address. The Read ECC command must be scheduled out before the Read Data command.

Figure 13-14 Read Operation Overview



When a Write (Data) command is incoming from HIF, the ECC is calculated when it is being scheduled out, and ECC is stored into an ECC Accumulator. When Write ECC is accumulated internally, the Write ECC command becomes available in the CAM to be scheduled out.

Figure 13-15 Write Operation Overview



ECC is written to SDRAM using data mask if ECC Accumulator has partial ECC for an ECC block. As RMW is not used for writing ECC to SDRAM, data mask must be enabled if Inline ECC feature is used.

13.1.1.12.1 Write ECC CAM

The WR ECC commands are stored in a separate write ECC CAM. This improves the DDR efficiency especially in the worst case scenarios where ECC overhead ratio is very high.

The depth of the WR ECC CAM is always half of the WR CAM depth. [Table 13-3](#) provides the depths for different hardware configurations.

Table 13-3 Depths for Different Hardware Configurations

CAM Size (MEMC_NO_OF_ENTRY)	WR CAM	WR ECC CAM
16	16	8
32	32	16
64	64	32



Write Data SRAM size is not affected, since it is not required for WR ECC CAM.

13.1.1.12.2 Block Tokens / ECC Arrays

The number of BTs (Block Tokens) in DDRCTL is defined by the hardware parameter `MEMC_NO_OF_BLK_TOKEN`. A BT can handle both reads and writes. Each block must have a BT.

- When one block access only has read, the block only need BRT
- When one block access only has write, the block only need BWT
- When one block access has both read and write or has RMW, the block need both BRT and BWT

Each BWT has an internal buffer (ECC_ACC) to accumulate ECC for writes in the block. Every time CAM schedules a write command, the corresponding ECC is calculated and stored in ECC_ACC. One block may have more than one write ECC command. When write ECC command is served, the associated write data is supplied directly from ECC_ACC. ECC_ACC is cleared with the last write ECC command or when the block is terminated.

Each BRT has an internal buffer (ECC_ARRAY) to store ECC that returns from read ECC command. One block can only have one read ECC command. Once the read ECC command is served, the ECC is stored in ECC_ARRAY. ECC_ARRAY is used to decode the following read commands in the block.

If a block has both BWT and BRT, and the ECC_ARRAY is ready when write ECC command is injected, the data of write ECC command is merged from both ECC_ACC and ECC_ARRAY, otherwise the write ECC command uses ECC_ACC only. The write ECC command could be a Masked Write or a Write in LPDDR4.

If a block has both BWT and BRT, and in a Read after Write case, the read data is decoded by the generated ECC in ECC_ACC, because the ECC_ARRAY could be out of date.

When access to a data block, and its block address is not in any valid block channel (see next chapter for details) a new BT is allocated to the data block. The BT is kept allocated until the following conditions are met:

- The BT is not in any valid block channel.
- All commands associated to the BT are served from CAM.

One BT only has maximum one Read ECC command and multiple Write ECC commands. The Write ECC command is the last command of the BT if needed. Once Read ECC is executed, the ECC parity is stored and used until the BT is released. The stored ECC is updated by any Write command with this BT (without writing back to SDRAM) and finally this is written to SDRAM as last command of the BT if needed.

13.1.1.12.3 Block Channels

Block channel is a data structure to support interleaving block address accesses at HIF with saving number of overhead commands.

Number of block channels is determined by a hardware parameter `MEMC_NO_OF_BLK_CHANNEL` and this means interleaving depth.

Each block channel has the following information (not necessarily all):

- Block Address (based on HIF address)
- ECC Address (based on DRAM address)
- Block Token (BT), Block Write Token (BWT), Block Read Token (BRT)
- A flag to indicate if any read is done in the channel
- A flag to indicate if any write is done in the channel
- A flag to indicate if this channel is valid

- A vector to record which offset address is written in the block
- A vector to record which offset address is read in the block

The channel structure has a write pointer to indicate the channel to be overridden with.

After reset, all channels are invalid (valid=0) and the write pointer indicates Channel 0 (CH0).

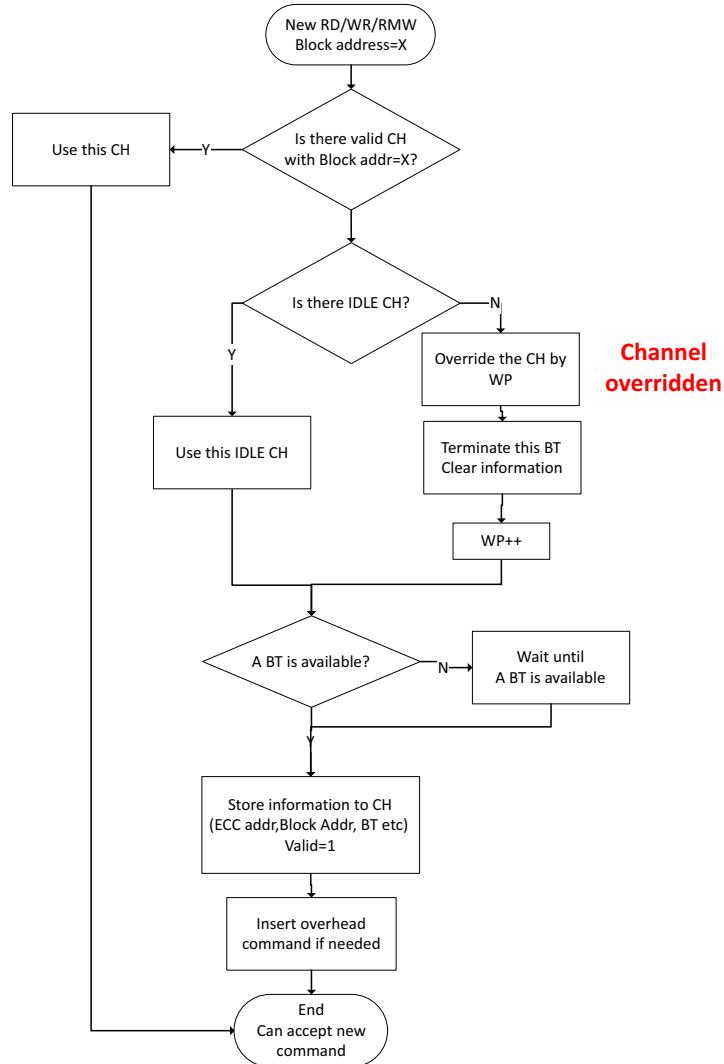
When a command (RD/WR/RMW) is incoming with block address=X, if there is a valid channel (valid=1) with block address=X, use this channel and the already allocated BT, push the incoming command and overhead commands into CAM with the BT. Otherwise if there is any IDLE channel (valid=0), use this IDLE channel, allocate a new BT, push the incoming command and overhead commands into CAM with the new BT, and then the channel becomes valid. Otherwise, update the channel indicated by the write pointer, allocate a new BT and push commands into CAM, then write pointer is increased by 1. If this channel is already valid, this is “channel overridden” condition and terminates the BT.

When a RD or RMW command is incoming, and this is first Read/RMW command in this channel, push an overhead RD ECC command into CAM prior to the command.

Every time a WR or RMW is incoming, push overhead WR ECC command to CAM following the command.

Figure 13-16 shows the flow for channels.

Figure 13-16 Block Channel Flow



Index:

- CH: Channel index
- WP: Write pointer to indicate CH to be overridden

Also, all block channels are invalidated (valid=0) and BTs associated to channels are terminated by the following events.

- Before entering Self-Refresh with CAM drain
- Any read/write access to unlocked ECC parity hole sub-region, which is represented by `ECCCFC1.ecc_region_parity_lock=0`
- `blk_channel_idle_time_x32` timer expires
- A full block read (8 access to different offset address to make all the address are read in the block)
- A full block write (8 access to different offset address to make all the address are written in the block)

When a Write follows a Read to the same address, the block is terminated before the Write command. The reason is that although CAM guarantee the order is RD and then WR, the ECC cache could be updated by the write data before read data is returned due to the read latency being bigger than the write latency. As a result, when the read data is returned, it does not match with ECC cache which can cause decoder error.



Note When one or more block channels are active, the DDRC command channel is not treated as IDLE. This affects to idleness timers PWRTMG.selfref_to_x32, RFSHSET1TMG0.refresh_to_x1_x32, PWRTMG.powerdown_to_x32, and HWLPTMG0.hw_lp_idle_x32.

13.1.1.12.4 Overhead Commands

Each data block requires overhead commands depending on command type and is shown in a table [Table 13-4](#).

Table 13-4 Overhead Commands Depending on Command Types

Command Type (HIF)	Overhead Command	Note
RD	Read ECC (RE_B)	Read ECC is served prior to Read Data command
WR	Write ECC (WE_BW)	Write ECC is served following Write Data command
RMW	Read ECC (RE_B)	Read ECC is served prior to Read part of RMW
	Write ECC (WE_BW)	Write ECC is served following Write part of RMW

For Read ECC (RE_B), if RE_B is already inserted by a RD or RMW command, no more RE_B are inserted when RD or RMW command is received again within a channel.

For Write ECC (WE_BW), WE_BW is pushed to CAM after a WR command. For LPDDR4 protocol and if the block is not full write access, it is pushed to CAM as a Masked Write. Otherwise, it is pushed to CAM as a Write command. If Read ECC (RE_B) is already inserted within a channel and its data is returned, the Masked Write is changed to Write command in CAM.

Although Write ECC (WE_BW) is pushed to CAM after every WR command, those Write ECC commands could be combined as one command if the previous WE_BW has not been served.

Figure 13-17 on page 357 gives an example sequence to explain how channel works and how overhead commands are injected (#channels=4).

Figure 13-17 Channel Sequence Example

#Seq	From HIF	Block channels	To Scheduler	Note
0	After reset			Channels are all invalid Write pointer = CH0
1	[RD] Blk A		[RD ECC] [RD] Blk A Blk A	There is no valid CH with blk=A, use CH0. This is first read for CH0, insert overhead RD command. Increment write pointer.
2	[WR] Blk B		[WR] [WR ECC] Blk B Blk B	There is no valid CH with blk=B, use CH1. This is write, Insert overhead WR command. Increment write pointer.
3	[RD] Blk A		[RD] Blk A	There is valid CH with blk=A, use CH0. This is read, but read is already issued for this channel, no overhead command is inserted. Do not increment write pointer.
4	[RD] Blk C		[RD ECC] [RD] Blk C Blk C	There is no valid CH with blk=C, use CH2. This is first read for CH2, insert overhead RD command. Increment write pointer.
5	[RMW] Blk D		[RD ECC] [RMW] [WR ECC] Blk D Blk D Blk D	There is no valid CH with blk=D, use CH3. This is first read for CH3, insert overhead RD and WR command. Increment write pointer.
6	[RD] Blk B		[RD ECC] [RD] Blk B Blk B	There is valid CH with blk=B, use CH1. This is first read in CH1. Insert overhead RD command. Do not increment write pointer.
7	[RD] Blk E		[RD ECC] [RD] Blk E Blk E	There is no valid CH with blk=E, use CH0. CH0 is already valid, so this is "channel overridden". BT0 will be released once all commands belonging to BT0 are scheduled out from the scheduler. Overhead RD command is inserted for new block as this is first read. Increment write pointer.
8	[WR] Blk F		[WR] [WR ECC] Blk F Blk F	There is no valid CH with blk=F, use CH1. CH1 is already valid, so this is "channel overridden". BT1 will be released once all commands belonging to BT1 are scheduled out from the scheduler. This is write, insert overhead WR ECC command. Increment write pointer.
9	[WR] Blk F		[WR] [WR ECC] Blk F Blk F	Assuming there are six WR commands to Blk F after seq#8, this is 8th WR command and this WR makes it full block write. The block F is terminated and CH1 becomes IDLE channel. Valid becomes 0 any rd and any wr are cleared as well. This is write, insert overhead WR ECC command. BT5 will be released once all commands belonging to BT5 are scheduled out from the scheduler. Do not increment write pointer.
10	[WR] Blk A		[WR] [WR ECC] Blk H Blk H	There is no valid CH with blk=A, use CH1 rather than override the CH2 (indicated by WP). This is write, insert overhead WR ECC command. Do not increment write pointer.

13.1.1.12.5 Block Address Collision

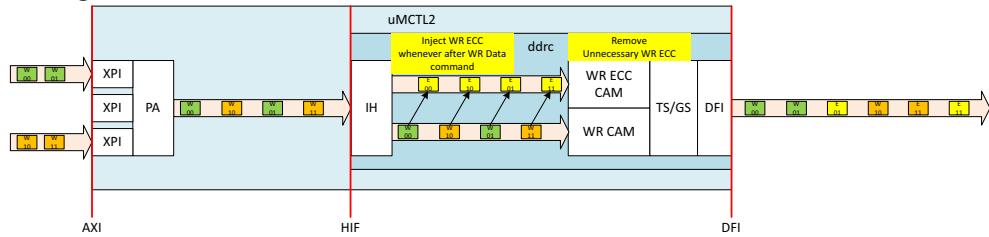
Block address collision means incoming command and existing commands (in CAM) have same block address (that is, use same ECC address) and have different BT (Block Token). This happens when a new block channel (block address=B) is initiated while there are entries with block address=B in CAM. If this happens, CAM does not accept incoming command until the collision is resolved by serving colliding commands in the CAM. At the same time, CAM prioritizes colliding commands (that is, flushes). The collision includes Write after Write (WAW), Read after Write (RAW), and Write after Read (WAR).

If incoming command and existing commands have the same BT and completely the same address, this is handled as normal address collision.

13.1.1.12.6 Scheduling Mechanism Improvement for WR

Figure 13-18 shows new mechanism to minimize performance drop by inline ECC:

Figure 13-18 Scheduling Mechanism



- Schedule WR ECC command without dedicated PRE-ACT cycle as much as possible (*):
 - Before serving any page-miss command to a bank, page-hit WR ECC command on the same bank (if it's there) is scheduled out.
 - Last WR Data burst for a block and corresponding WR ECC burst can be scheduled back to back (at least for BL16 case).
 - This is done by the following mechanism:
 - IH always generates WR ECC command immediately after WR Data command and pushes into WR ECC CAM so that the DDRCTL can schedule WR ECC command whenever needed (one WR ECC command pushed into CAM for one WR Data command).
 - WR ECC CAM combines WR ECC commands when incoming WR ECC command from IH and existing WR ECC command in WR ECC CAM have the same Block Token (BT) to minimize number of overhead WR ECC commands.
 - WR ECC command becomes candidate to be scheduled when all WR commands belonging to a block in CAM are served.

(*) The following are exceptions:

- Critical Refresh
- RD/WR switching due to starvation/q-empty
- CAM flushing due to collision
- High priority transaction: xVPR/xVPW/HPR
- PHY Master request
- If both WR ECC command and WR Data command are candidate to be scheduled for a bank, priority is the following:
 - a. Page-hit oldest WR ECC command (corresponding BT is already terminated/overridden)
 - b. Page-hit oldest WR Data command
 - c. Page-hit oldest WR ECC command (corresponding BT is not terminated/overridden)
 - d. Page-miss oldest WR ECC command (corresponding BT is already terminated/overridden)
 - e. Page-miss oldest WR Data command
 - f. Page-miss oldest WR ECC command (corresponding BT is not terminated/overridden)
- Bank Priority is the following:
 - Oldest bank for WR ECC (corresponding BT is already terminated/overridden)

- Oldest bank for WR Data
- Therefore, as long as WR Data command arrives at CAM before serving WR ECC command, the WR ECC command is not scheduled out. Typically, the WR ECC command is scheduled out when the block channel is terminated or before serving page-miss WR Data command.

13.1.1.12.7 Performance

The idle command latency of the controller increases by one controller clock cycle with inline ECC.

- Full block write: No additional latency
 - ECC burst is appended after 8 data bursts, the write data response is generated normally after each data burst.
 - Back to back full write blocks can be streamed with 88.9% application port efficiency and 100% DDR efficiency (this assumes single block channel).
- For block read: BL/2 additional latency (4 DDR cycles for BL8, eight DDR cycles for BL16)
 - First, read the ECC burst, then read the eight consecutive data bursts. The first read command of the block channel incurs the latency.
 - Back to back full read blocks can be streamed with 88.9% application port efficiency and 100% DDR efficiency.

Table 13-5 Performance Indicator

Condition					Performance			Latency (first access)			
Read		DM	No. burst	BL	No. burst overhead	Disabled Inline ECC (DDR clock cycles)	Inline ECC (DDR clock cycles)	Efficiency %	Disabled Inline ECC (DDR clock cycles)	Inline ECC (DDR clock cycles)	Additional (DDR clock cycles)
Block Read	8 burst data	X	8	8	1	32	36	88.89	RL	RL+BL/2	BL/2
	7 burst data	X	7	8	1	28	32	87.50	RL	RL+BL/2	BL/2
	6 burst data	X	6	8	1	24	28	85.71	RL	RL+BL/2	BL/2
	5 burst data	X	5	8	1	20	24	83.33	RL	RL+BL/2	BL/2
	4 burst data	X	4	8	1	16	20	80.00	RL	RL+BL/2	BL/2
	3 burst data	X	3	8	1	12	16	75.00	RL	RL+BL/2	BL/2
	2 burst data	X	2	8	1	8	12	66.67	RL	RL+BL/2	BL/2
	1 burst										

13.1.1.13 Address Protection with ECC (ECCAP)

- There is no CA parity protection in LPDDR4.
- Protect the address within the ECC scheme: when you do a read and if a read address is faulty, distinguish it from data error (when possible) and the report.
- Address flips during writes can be detected within certain delay (only when read back or if the scrubber is enabled, then within a certain time window). Address errors which may appear before the ECC are detected directly.
- SECDED on 64-bit: total of 8-bit of ECC check bits:
 - SEC part is 7-bit and there is additional one bit “overall parity” to determine 2-bit failures.
 - “The overall parity” can only detect 1, 3, 5, and 7 bit errors. Even number of errors are not detected.
- “Address Parity” is:

- Use {cs, row, ba/bg, col} addresses to calculate it.
- RAS/CAS or other command bits cannot be protected.
- “Error Threshold” is:
 - A certain number of “overall parity” failures within one memory burst.
- Use 64-bit SECDED.
- Calculate the address parity and invert two bits of ECC check bits when address parity=1.

In [Table 13-6](#), “x” indicates one-bit flip on the address. There are three error types identified and not all address failures can be detected with 100% certainty.

If there is a refresh command in between RD (ECC) and RD (DATA), then there are additional ACT (DATA) and same error types still apply.

Table 13-6 Error Types

NO	Error types	1st command		2nd command		3rd command		Result	Address Protection
		ACT	RD ECC	RD DAT	Row Ba/Bg/CS	Col Ba/Bg/CS	Col Ba/Bg/CS		
1	error type 3	x						Wrong page opened, but data and ECC are matched	ECC decoder found 2 bit uncorrectable error in all beats of a burst due to address parity. 100% address error detection using error threshold method.
2	error type 1		x					Protocol issue because RD command is to a closed bank	Unpredictable behaviour (need to confirm with DRAM vendors). If there's no DQS toggling, controller will get stuck. If there is DQS toggling, then data and ecc mismatch
3	error type 2			x				Data and ECC are not matched	ECC decoder does not have the capability to handle more than 2 errors. We rely on the overall parity. The overall parity cannot identify an address error by itself. Random number of errors are detected within a memory burst. Use an error threshold (say 4 errors out of 8 within a burst) to identify as address error - therefore address error
4	error type 2			x				Data and ECC are not matched if the wrong bank is already opened	
5	error type 1			x				Protocol issue because RD command is to a closed bank.	
6	error type 2				x			Data and ECC are not matched	
7	error type 2					x		Data and ECC are not matched if the wrong bank is already opened	
8	error type 1					x		Protocol issue because RD command is to a closed bank	
9	error type 3		x		x			Same column bit failing together and result data and ECC are matched	
10	error type 2		x		x			Column bits failing differently, Data and ECC are not matched	
11	error type 3			x		x		Bank bits failure together. Data and ECC are matched	
12	error type 2			x		x		Bank bits failing differently and both reads go to open banks. Data and ECC are not matched	
13	error type 1			x		x		Bank bits failing differently and protocol error if any RD command is to a closed page	



Note When there are read after write on same address within a block channel, the ECC check is done using ECC which was generated by the write in ECC cache. Therefore, error type 3 can become error type 2 as the controller might use correct ECC regardless of address error on reading ECC from SDRAM.

13.1.1.13.1 ECCAP of Inline ECC

The aim of ECC Address Protection (ECCAP) is to detect an address flip between DDRCTL and DRAM.

The original ECC Address Protection (ECCAP) feature can detect odd-bit address flip(s) on a row in limited situations. Other types of address flips can cause ECC and Data mismatch or protocol violation, but it tries to detect address flip(s) by counting the number of CE/UE in the burst with a programmable threshold to

detect it as far as possible. In such case with address error, multiple CE or UE are likely there within a burst, which also can cause mis-correction.

This enhanced ECCAP feature is to improve the following based on Multi-bit Error Detection (MED) rather than Single Error Correction Double Error Detection (SECDED).

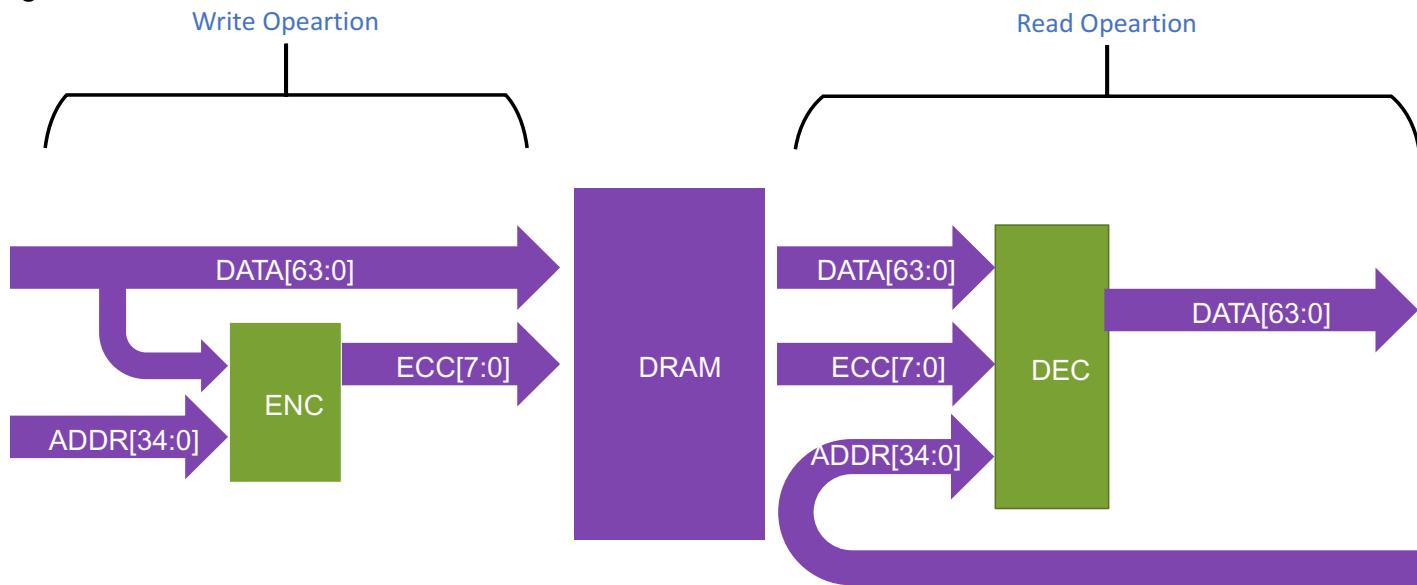
- Detect double-bit address flip (or even-bit address flip by chance) on a row apart from the odd-bit error, to protect DRAM read or write address.
- Improve detection possibility when ECC and Data mismatch due to MED. No concern about mis-correction.

To enable the enhanced ECCAP feature, the following registers must be set:

- ECCAP mode is enabled (`ECCCFG0.ecc_ap_en==1'b1`).
- MED mode is enabled (`ECCCFG1.med_ecc_en==1'b1`).
- Embedded ECCAP mode is selected (`ECCCFG1.ecc_ap_mode==1'b1`).

By applying the enhanced ECCAP, address information of DRAM traffic command is encoded into 8-bit ECC check bits together with 64-bit data using SEDDED algorithm. As a result, uncorrectable error(s) can be detected if address flip occurs in limited situations.

Figure 13-19 Enhanced ECCAP codec Data Flow Overview



The following are the related interrupts for inline ECC:

- `ecc_uncorrected_err_intr`
- `ecc_corrected_err_intr`
- `ecc_uncorrected_err_intr_fault`
- `ecc_corrected_err_intr_fault`
- `ecc_ap_err_intr_fault`
- `ecc_ap_err_intr`

In enhanced ECCAP mode, UE-related interrupts reflect if DDRCTL detects inline ECC error(s) on data and/or address. ECCAP-related interrupts reflect if the potential address flip(s) is/are detected in

correspondence to a pre-defined register `ECCCFG0.ecc_ap_err_threshold`. This register must be set to the threshold of the number of errors within a burst as many uncorrectable errors are expected to be detected within a burst if the error is from the address flip(s). The interrupt `ecc_ap_err_intr` asserts if the error number exceeds `ECCCFG0.ecc_ap_err_threshold` to indicate the potential address flip(s).

13.1.1.13.2 SEDDED of Inline ECC

Previous DDRCTL inline ECC design uses SECDED (single error correction double error detection) algorithm to protect data. 8-bit ECC is added for 64-bit data. This codec has the ability of 1-bit error correction or 2-bit error detection within the 64-bit data + 8-bit ECC.

The purpose of introducing the design of SEDDED (single error detection double error detection), that is, MED (multiple-bit error detection), is to increase the error detection ability in the systems valuing multiple error detection while not demanding error correction.

To enable SEDDED:

- Set the register field `ECCCFG1.med_ecc_en` to '1'.

The difference between SECDED and SEDDED is shown in the following table.

	1-bit error	2-bit error	3-bit error	Odd-bit error (> 3-bit)	Even-bit error (> 2-bit)
SECDED	Correct	Detect	Mis-correct	Mis-correct	Unpredictable
SEDDED	Detect	Detect	Detect	Detect	Unpredictable

The following are the related interrupts for inline ECC:

- `ecc_uncorrected_err_intr`
- `ecc_corrected_err_intr`
- `ecc_uncorrected_err_intr_fault`
- `ecc_corrected_err_intr_fault`

In SEDDED, UE-related interrupts reflect if DDRCTL detects inline ECC error(s); a 1-bit ECC error is also detected as an uncorrectable error.

13.1.1.14 Scrubbing

For more information on scrubbing functionality in Inline ECC configurations, see “[Scrubber](#)” on page [373](#).

13.1.1.15 QoS

The DDRCTL supports all priorities with Inline ECC feature:

- VPW/VPR
- NPW/LPR/HPR

One block must have the same priority for reads because read command cannot be served until corresponding read ECC command is served. This is managed in DDRCTL internally as:

- If the first read in a block is HPR, read ECC command is also generated as HPR and call this HPR block so that read ECC command is executed as same priority.
- If the first read/RMW in a block is LPR/VPR, read ECC command is also generated as LPR/VPR and calls this LPR/VPR block so that read ECC command is executed as same priority.
- If HPR is incoming to an existing HPR block, nothing can be done.
- If VPR read/RMW is incoming to an existing LPR/VPR block and read ECC command is still in the CAM, the read ECC command's priority is updated to be the same priority as incoming VPR command only when the incoming VPR command is higher priority (that is, smaller VPR timeout) than current read ECC command.
 - Read ECC is LPR and incoming read is LPR, nothing can be done.
 - Read ECC is LPR and incoming read is VPR with timeout=X, the read ECC is updated to VPR with timeout X, including X=0 case which is expired VPR.
 - Read ECC is VPR with timeout=Y and incoming read is LPR, nothing special can be done.
 - Read ECC is VPR with timeout=Y, incoming read is VPR with timeout=X and X<Y, the read ECC is updated to VPR with timeout X.
 - Read ECC is VPR with timeout=Y, incoming read is VPR with timeout=X and X >= Y, nothing special can be done.
- If HPR read is incoming to an existing LPR/VPR block channel (meaning same block address) and read ECC command is still in the CAM, this LPR/VPR block channel is terminated and new HPR block is started.



- If the read ECC command is already served when the HPR read is incoming, the block is not terminated.
- If any write command is in the existing LPR/VPR block channel, the block is not terminated to avoid block address collision.

- If LPR/VPR read is incoming to an existing HPR block channel (meaning same block address) and read ECC command is still in the CAM, this HPR block channel is terminated and new VLPR/VPR block is started.



- If the read ECC command is already served when the VPR/LPR read is incoming, the block is not terminated.
- If any write command is in the existing HPR block channel, the block is not terminated to avoid block address collision.

13.1.1.16 Features and Limitations

The following are the features and limitations of Inline ECC:

- Data Mask (DM) must always be enabled
- MEMC_DRAM_DATA_WIDTH=16, 32, 64¹
- MEMC_BURST_LENGTH=16

- Write combine is not supported for the block access (that is, no block write combine).
- MEMC_USE_RMW=1 is only.
- Scrubber block is enhanced to issue RMW scrub command when there is a correctable error (UMCTL2_SBR_EN can be set to '1'). The scrubber is not sensitive to other error types and they gets ignored.
- ECC Data Poisoning is not supported (ECCCFG1.data_poison_en must be set to '0'). There is a software method to inject errors to ECC parity bits with inline ECC. For more details, see section "[Error Injection Through Software \(ECC Data Poisoning\)](#)" on page [346](#).
- OCPAR is supported only in limited set of configurations, contact Synopsys for more details.
- Dual HIF is not supported (UMCTL2_DUAL_HIF must be set to '0').
- Supported QoS features:
 - HPR/LPR is supported.
 - VPRW is supported (UMCTL2_VPRW_EN can be set to '1').
- Unaligned reads to memory burst is not supported at the HIF interface. Unaligned writes at HIF are supported.
 - XPI in arbiter configuration always generates aligned read bursts to HIF.
- MEMC_OPT_TIMING=1 is only supported.
- Inline ECC is supported in AXI configuration only (that is, DDRCTL_SYS_INTF = 1).
- HIF only configuration (UMCTL2_INCL_ARB==0) is not supported.
- MEMC_ECC_SUPPORT must be set to '1' - Advanced ECC is not supported.
- Quarter Bus Width (MEMC_QBUS_SUPPORT) is supported as hardware parameter, but when Quarter Bus Width is used, Inline ECC must be disabled by software.
- Unused sub-regions in the ECC region can be utilized as non-protected region. The entire size of the unused sub-region depends on setting of ECCCFG0.ecc_region_map and ECCCFG0.ecc_region_map_other. However, by default, the unused sub-region are fragmented in terms of HIF/AXI address. Enabling ECC region remap feature (ECCCFG0.ecc_region_remap_en=1) makes the unused sub-regions consecutive, however the performance of sequential access for the unused region can be sub-optimal compare to normal unprotected regions. For more details, see "[ECC Region Remap](#)" on page [346](#).



Note Inline ECC cannot be enabled in software, if QBW is selected.

13.1.1.16.1 Address Map Limitation

In case of MEMC_BURST_LENGTH=16, hif_cmd_addr[6:0] can be mapped to CS or ROW from address map perspective, but this is NOT supported when Inline ECC is enabled from performance consideration.

1. MEMC_DRAM_DATA_WIDTH must be set to '16' or '32' in LPDDR5/4/4X Controller.

13.1.1.16.2 Prohibit RMW Command to Unprotected Regions and ECC Region



Note Only for configurations with UMCTL2_INCL_ARB==0, which are not supported in current version of the controller.

When inline ECC is enabled (`ecc_mode=3'b100`), data mask must be enabled. There is no reason to use RMW for non-protected regions and ECC region (both used and unused sub-regions).

If you use RMW to non-protected region, DDRC could have wrong behaviors as described:

There are two commands who have same address excepting of column address bit3 and are going to unprotected region. The first command is RMW command whose column bit3 0. The second command is a write command which column bit3 is 1 and is a masked write.

The second write command is combined with write part of the first RMW, and then the combined write command becomes write command instead of Mask write.

But when inline ECC is enabled, read part of RMW is partial read. Only the first half of data are read (column 0~7), the second half of data is still masked. As a result, data mask is used for a write command in LPDDR4 protocol that violates LPDDR4 specification.

13.1.1.17 Interrupts Related to Inline ECC

The following are the interrupts for Inline ECC:

- `ecc_uncorrected_err_intr`
- `ecc_corrected_err_intr`
- `ecc_uncorrected_err_intr_fault`
- `ecc_corrected_err_intr_fault`
- `ecc_ap_err_intr_fault`
- `ecc_ap_err_intr`

13.1.1.18 Signals Related to Inline ECC

This section contains the following subsections:

- “[HIF Signals](#)” on page [365](#)
- “[Credit Counters Signals](#)” on page [367](#)

13.1.1.18.1 HIF Signals

The following signals are generated internally by the XPI (AXI Port Interface) in AXI configurations.

- `hif_cmd_ecc_region`: this signal indicates the command belongs to the ECC protected region.
- `hif_lpr_credit[1:0]`: this signal is expanded to 2 bits. Bit 0 indicates LPR credit increment pulse from the CAM as before when the oldest entry is scheduled; Bit 1 indicates LPR credit increment pulse from the IH whenever the command does not need to inject an overhead command. Both bits can be asserted at the same time, and in that case the HIF manager needs to increment credits by 2.

- **hif_hpr_credit[1:0]**: this signal is expanded to 2 bits. Bit 0 indicates HPR credit increment pulse from the CAM as before when the oldest entry is scheduled; Bit 1 indicates HPR credit increment pulse from the IH whenever the command does not need to inject an overhead command. Both bits can be asserted at the same time, and in that case the HIF master needs to increment credits by 2.
- **hif_wr_credit[0:0]**: this signal indicates that a write request (except WR ECC command generated internally for inline ECC) and write data have been scheduled to the DDR, or that a Write Combine has occurred, and the SoC must increment the credit value associated with writes.
- **hif_wrecc_credit[1:0]**: [Inline ECC only] this signal indicates that a write ECC request and write data generated internally have been scheduled to the DDR, or that a Write Combine has occurred, and the SoC must increment the credit value associated with writes ECC. Bit[1] is dedicated for Address error (LPDDR4) request to protected region.
- **hif_rdata_corr_ecc_err**: signal to tell scrubber that read data has corrected error. Scrubber issues a masked RMW to correct it.
- **hif_cmd_wdata_mask_full**: width of this signal is equal to the maximum number of beats allowed for that configuration, which is $\text{MEMC_BURST_LENGTH}/(2*\text{MEMC_FREQ_RATIO})$. Each bit identifies a data beat for that write command. If the bit is set to '1', all the bytes for that beat are meant to be written, so hif_wdata_mask for that beat must all be set to 1s. If a bit is 0, bytes are masked.
- **hif_cmd_length**: this signal indicates for both writes and reads if the access is full or partial (can be half or quarter). This information determines the number of HIF data beats required for one HIF command.

When the HIF master or PA sends a HIF command to ECC protected region, it needs to check and consume corresponding credits according to the [Table 13-7](#).

Table 13-7 Credits Consumed When the HIF Master or PA Sends a HIF Command

HIF Command	Credit			
	LPR	HPR	WR Data	WR ECC
LPR/VPR	2			
HPR		2		
WR			1	1
RMW	2		1	1

If you send LPR or RMW to protected region, you must program `SCHED.lpr_num_entries > 0`; If you send HPR to protected region, you must program `SCHED.lpr_num_entries < MEMC_NO_OF_ENTRY-2`.



Note For non-Inline-ECC configuration, PA asserts `hif_go2critical_wr` when WR CAM is full and PA has urgent request in its pipeline. For Inline ECC configuration, PA asserts `hif_go2critical_wr` when "WR Data CAM" or "WR ECC CAM" are full and has urgent request in its pipeline. Then DDRCTL moves to write mode and WR ECC entry and WR Data entry are served.

13.1.1.18.2 Credit Counters Signals

[Table 13-8](#) shows the credit counters signals.

Table 13-8 Credit Counters Signals

Name	Width	IO	From/To	Description
wrecc_credit_cnt	7	O	From PA	<p>Number of available write ECC WR CAM slots (free positions). Each slot holds a DRAM burst Synchronous to controller clock (core_ddrc_core_clk). Value is decremented/incremented as the commands flow in out of the write ECC CAM.</p> <p>Exists: UMCTL2_INCL_ARB & MEMC_INLINE_ECC</p>

13.1.1.19 Registers Related to Inline ECC

The following are the new registers added for Inline ECC.

- ECCCFG0:
 - ecc_region_map_granu
 - ecc_ap_err_threshold
 - blk_channel_idle_time_x32
 - ecc_region_map
 - ecc_ap_en
 - ecc_mode
 - ecc_region_remap_en
 - ecc_region_map_other
- ECCCFG1:
 - active_blk_channel
 - blk_channel_active_term
 - ecc_region_waste_lock
 - ecc_region_parity_lock
- ECCCTL:
 - ecc_ap_err_intr_force
 - ecc_uncorrected_err_force
 - ecc_corrected_err_intr_force
 - ecc_ap_err_intr_en
 - ecc_uncorrected_err_intr_en
 - ecc_corrected_err_intr_en
 - ecc_ap_err_intr_clr
 - ecc_uncorr_err_cnt_clr
 - ecc_corr_err_cnt_clr

- ❑ ecc_uncorrected_err_intr_clr
- ❑ ecc_corr_err_intr_clr
- ECCAPSTAT:
 - ❑ ecc_ap_err
- OPCTRLCAM1
 - ❑ dbg_wrecc_q_depth
- SCHED5
 - ❑ wrecc_cam_highthresh
 - ❑ wrecc_cam_lowthresh

The following registers are not used for Inline ECC:

- ECCCFG1
 - ❑ data_poison_bit
 - ❑ data_poison_en
- ECCPOISONADDR0/1
- ECCPOISONPAT0/1/2
- ADVECCINDEX

13.1.1.19.1 Differences in ECC Log Registers

The widths of `ECCSTAT.ecc_corrected_err` and `ECCSTAT.ecc_uncorrected_err` are different between Sideband ECC and Inline ECC mode.

- In Inline ECC mode, `ECCSTAT.ecc_corrected_err` and `ECCSTAT.ecc_uncorrected_err` are 1 bit.
- In Sideband ECC mode, `ECCSTAT.ecc_corrected_err` and `ECCSTAT.ecc_uncorrected_err` are 2 or 4 bits according to `MEMC_FREQ_RATIO`.

The column address definition is different between Sideband ECC and Inline ECC.

- In Sideband ECC, `ecc_corr_col` is based on DFI words and not on DRAM words. Following are the details:
 - ❑ `ecc_corr_col[0]` is always 1'b0. This means it cannot identify that error is ODD column or EVEN column. It is described by "lowest bit not assigned here".
 - ❑ `ecc_corr_col[1]` is always 1'b0 also in 1:2 mode. This means it cannot identify that error is in the lowest two column bits. It is not described in the databook.
 - ❑ In other words, current `ecc_corr_col` is based on DFI word. To identify the exact DRAM column address, use `ECCSTAT.ecc_corrected_err` which is 2 or 4 bits.
- In Inline ECC, the ECC code is always 64 bit, and could consist of one or two or four DRAM addresses. As it is not possible to comply with previous DFI words, the new definition of `ecc_corr_col[11:0]` is:
 - ❑ If the `MEMC_DRAM_DATA_WIDTH=641`, `ecc_corr_col` identifies the exact DDR column address regardless of frequency ratio (1:1 or 1:2).

1. `MEMC_DRAM_DATA_WIDTH` must be set to '16' or '32' in LPDDR5/4/4X Controller.

- ❑ If the `MEMC_DRAM_DATA_WIDTH=32`, `ecc_corr_col` identifies the first DDR column address in the ECC code (where ECC is comprised of 2 column addresses), regardless of frequency ratio (1:1 or 1:2).
- ❑ If the `MEMC_DRAM_DATA_WIDTH=16`, `ecc_corr_col` identifies the first DDR column address in the ECC code (where ECC is comprised of 4 column addresses), regardless of frequency ratio (1:1 or 1:2).

`ECCBITMASK` is always 72-bit wide and is accumulated at the ECC decoder.

13.1.1.19.2 Differences in the Programming Flow

1. For multiport configurations, `PCTRL.port_en` is used to enable or disable the input traffic per port.
2. The controller idleness can be polled first from the `PSTAT` register (`wr_port_busy_n` and `rd_port_busy_n` bit fields) and must read as `PSTAT==32'b0` (not busy).
3. Write 0 to `SBRCTL.scrub_en` to disable the scrubber, SBR, (required only if SBR instantiated). Poll `SBRSTAT.scrub_busy=0`, indicates that there are no outstanding SBR read commands (required only if SBR instantiated).
4. `OPCTRL1.dis_hif` is used to enable or disable the input traffic to the DDRC part of the controller. The change here is the requirement of asserting `dis_hif` even in AXI configurations. This ensures flushing of all ECC block channels.

DDRC CAM/pipeline empty status must be polled ((`OPCTRLCAM.dbg_wr_q_empty==1'b1`) && (`OPCTRLCAM.dbg_rd_q_empty==1'b1`) && (`OPCTRLCAM.wr_data_pipeline_empty==1'b1`) && (`OPCTRLCAM.rd_data_pipeline_empty==1'b1`)).

13.1.1.20 Controller Behavior During ECC Errors

When the DDRCTL detects a correctable ECC error, it performs the following:

- Sends the corrected data to the SoC as part of the read data.
- Sends the ECC error information to the APB register module.

When the DDRCTL detects an uncorrectable error, it does the following:

- Sends the data with error to the SoC as part of the read data.
- Sends the ECC error information to the APB register module.
- Generates a SLVERR response on the AXI interface.

In addition to the previous uncorrectable errors, when the DDRCTL detects an uncorrectable error during the read part of a normal RMW command, it does the following:

- During the subsequent write part of the normal RMW command, it performs a 2-bit corruption to the recalculated ECC word.
- This corruption ensures that there is no accidental ECC auto-correction, of any data with uncorrectable errors.

13.1.1.20.1 ECC Error Reporting Registers

A wide range of information about the detected errors can be obtained by reading the ECC error reporting registers. Full details of these are provided in the Register Descriptions chapter of the Reference Manual. There are also two interrupts, `ecc_corrected_err_intr` and `ecc_uncorrected_err_intr`, which are asserted when corrected or uncorrected errors are detected.

The ECC logging registers capture the first ECC correctable error and first ECC uncorrectable error detected. They store the log information related to those errors including the ECC lanes at which the errors are reported (even if additional errors arrive), until the associated interrupt is cleared by writing to `ECCCTL.ecc_corrected_err_clr` or `ECCCTL.ecc_uncorrected_err_clr`, respectively.

However, for multi-beat ECC, odd ECC lanes are not used for reporting. Any error on odd ECC lane #N (odd SDRAM beat) is reported as #(N-1) correctable or uncorrectable error on `ECCSTAT.ecc_corrected_err` or `ECCSTAT.ecc_uncorrected_err`.

Multibeat ECC error report is shown in [Figure 13-20](#).

Figure 13-20 Multibeat ECC Error Report

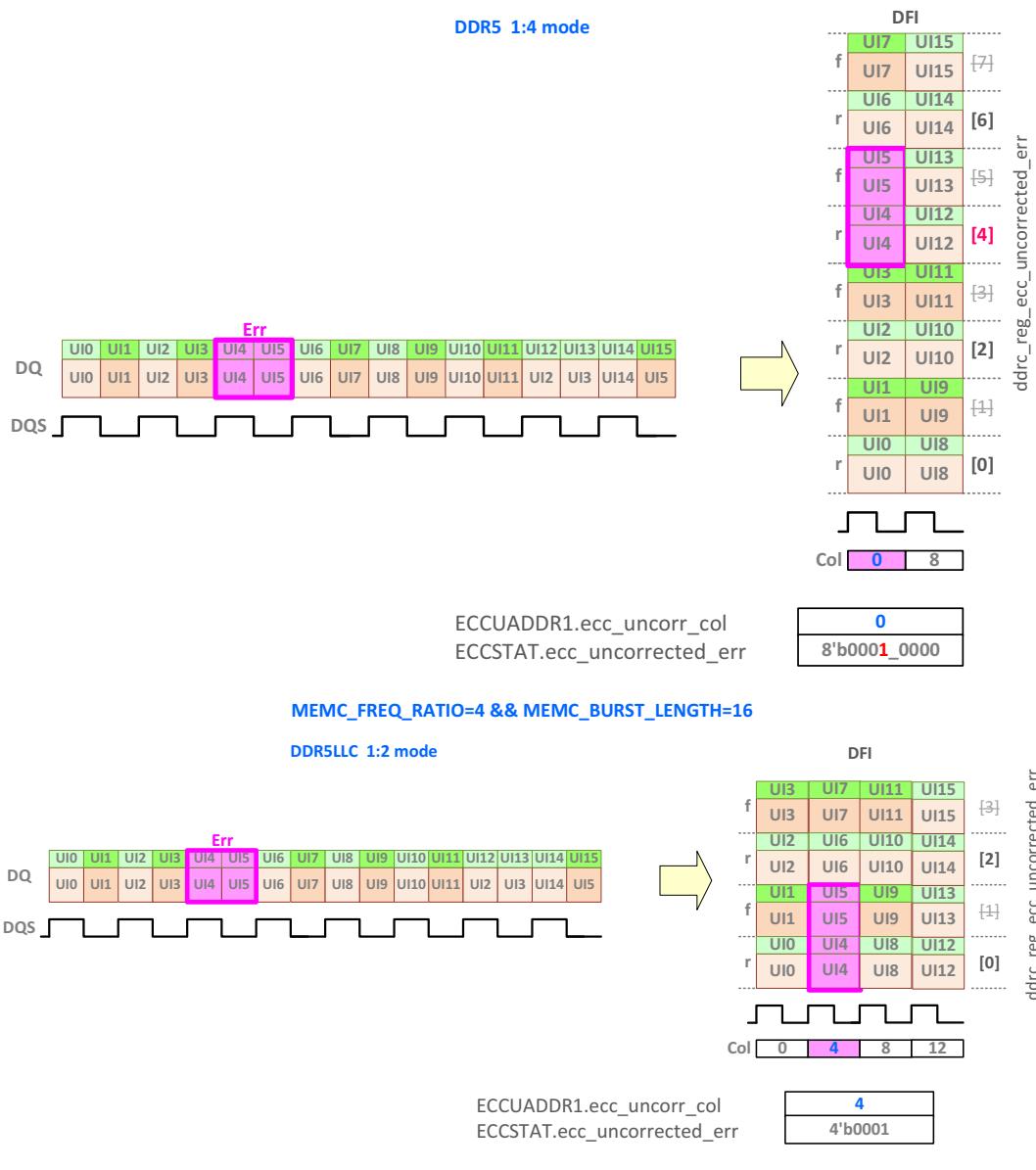


Table 13-9 specifies the encoding used for the status register field `ECCSTAT.corrected_bit_num`, which indicates the bit that is corrected (see “`REGB_DDRC_CH0 Registers`” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

Table 13-9 Encoding for `ECCSTAT.corrected_bit_num / ECCCFG2.flip_bit_pos`*

Value on <code>ECCSTAT.corrected_bit_num /</code> <code>ECCCFG2.flip_bit_pos*</code>	Bit That Has Error or is Flipped		
	<code>MEMC_DRAM_DATA_W</code> IDTH==64	<code>MEMC_DRAM_DATA_W</code> IDTH==32	<code>MEMC_DRAM_DATA_W</code> IDTH==16
0	64 (ecc[0])	32 (ecc[0])	16 (ecc[0])
1	65 (ecc[1])	33 (ecc[1])	17 (ecc[1])
2	66 (ecc[2])	34 (ecc[2])	18 (ecc[2])
3	0	0	0
4	67 (ecc[3])	35 (ecc[3])	19 (ecc[3])
5	1	1	1
6	2	2	2
7	3	3	3
8	68 (ecc[4])	36 (ecc[4])	20 (ecc[4])
9	4	4	4
10	5	5	5
...
15	10	10	10
16	69 (ecc[5])	37 (ecc[5])	21 (ecc[5])
17	11	11	11
18	12	12	12
...
21	15	15	15
22	16	16	KBD bit *
...	N/A
31	25	25	N/A
32	70 (ecc[6])	38 (ecc[6])	N/A
33	26	26	N/A
34	27	27	N/A

Value on ECCSTAT.corrected_bit_num / ECCCFS2.flip_bit_pos*	Bit That Has Error or is Flipped		
	MEMC_DRAM_DATA_W IDTH==64	MEMC_DRAM_DATA_W IDTH==32	MEMC_DRAM_DATA_W IDTH==16
	N/A
38	31	31	N/A
39	32	KBD bit *	N/A
...	...	N/A	N/A
63	56	N/A	N/A
64	71 (ecc[7])	N/A	N/A
65	57	N/A	N/A
66	58	N/A	N/A
...
71	63	N/A	N/A
72	KBD bit */ EAPAR_bit0**	N/A	N/A
73	EAPAR_bit1**	N/A	N/A



Note KBD bit* is applicable only when End2End Data Error Tracking using KBD is enabled.

EAPAR_bit is applicable only for poisoning using ECCCFS2.flip_bit_pos* when Encoded Address Parity is enabled.

For multi-beat ECC, while the effective SDRAM data width is 32, consecutive even and odd SDRAM data beats are combined to form 64-bit data and 8-bit ECC. So, the encoding used for the status register field ECCSTAT.corrected_bit_num follows the same as depicted in MEMC_DRAM_DATA_WIDTH==64 in Table 13-9 on page 371. So, the value ranging from 39 to 71 represents odd SDRAM data beat (ECC lane).

13.2 Scrubber

This section contains the following subsections:

- “[Scrubber Overview](#)” on page [373](#)
- “[Scrub Period](#)” on page [374](#)
- “[Restrictions on Scrub Interval With Auto Power-down/Auto Self-refresh Idle Time](#)” on page [375](#)
- “[Scrubber Normal Operation](#)” on page [375](#)
- “[Scrubber Address Configuration](#)” on page [375](#)
- “[Address Generation](#)” on page [376](#)
- “[Additional Notes for the ECC Scrubber](#)” on page [377](#)
- “[Status](#)” on page [378](#)
- “[Hardware Controlled Low-Power Operation](#)” on page [378](#)
- “[Software Controlled Low-Power Operation](#)” on page [379](#)
- “[Initialization Writes](#)” on page [379](#)
- “[Scrubber Credit Interface](#)” on page [381](#)

13.2.1 Scrubber Overview

The DDRCTL provides a comprehensive solution to automatically correct single-bit errors in SECDED ECC or up to 2 symbol errors in Advanced ECC in the DRAM. The ECC algorithm is based on the configuration. ECC Scrubber (SBR) is a block capable of generating initialization write commands, periodic read commands, periodic RMW commands, and correction RMW commands. In Normal mode, ECC Scrubber issues periodic background read commands to the DDRC. Periodic scrubbing is performed by the ECC Scrubber block to increase the reliability of the system by correcting correctable ECC errors in time, before they turn into uncorrectable errors. Each scrub command is a read of one memory burst (for example, BL8 or BL16) which is sent to the DDRC periodically with the lowest priority. The data read sent back is terminated inside scrubber and is not returned to the system. The clock and reset to the SBR block (`sbr_clk` and `sbr_resetn`) is separated from the controller clock and reset (`core_ddrc_core_clk` and `core_ddrc_rstn`) but must be synchronous with each other. In AXI configurations, the SBR can continue to function, such as counting and requesting an exit from low-power even if the controller clock is gated.

The Scrubber can support three types of commands using the `SBRCTL.scrub_cmd_type` register field. The commands are periodic reads, initialization writes, and periodic RMW transactions:

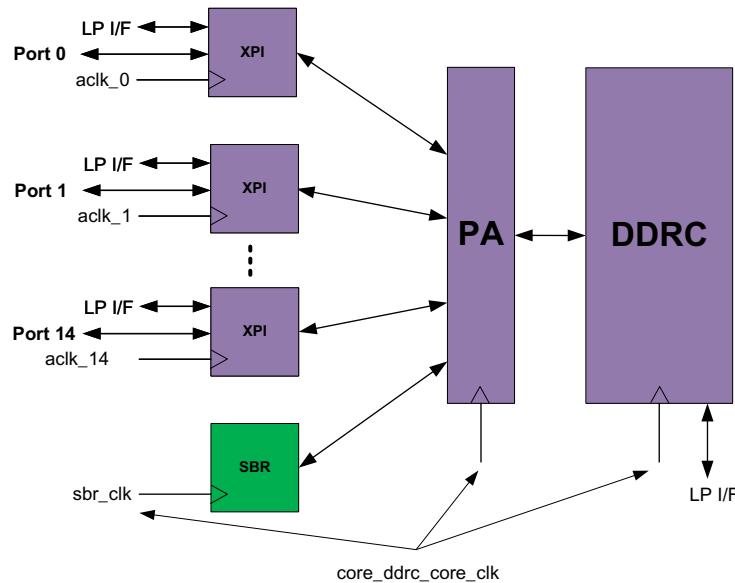
- Periodic Reads are only for scrubbing. In these reads, if a correctable error is detected by the DDRCTL, then a new correction Read Modify Write is generated.
- Initialization back-to-back writes are only for memory initialization.
- Periodic RMWs can be used for scrubbing or for external encryption schemes.

To enable the ECC Scrubber, the following hardware parameters must be set to '1':

- `UMCTL2_SBR_EN`
- `DDRCTL_SYS_INTF > 0`
- `MEMC_USE_RMW`
- `MEMC_ECC_SUPPORT > 0`

Figure 13-21 shows the SBR block with respect to other controller functions. As shown in the figure, the SBR consumes one of the ports of the Port Arbiter (PA) as a requester. If the SBR is enabled, up to 15 application ports can be supported instead of 16.

Figure 13-21 ECC Scrubber With Respect to Other Functional Blocks



In Inline ECC configurations, the SBR issues the periodic scrub read command and automatic RMW command upon detection of a single-bit ECC error. The scrubber initiates the RMW command for every single-bit of the ECC error detected. The SBR block is enhanced to issue fully masked (all bytes disabled) single read-modify-write command after detecting a correctable read error (a new HIF signal `hif_rdata_corr_ecc_err` is provided).

13.2.2 Scrub Period

The Scrub interval (tSCRUBI) is defined as the number of clock cycles between two scrub read commands. The Scrub period is defined as the time needed to read every memory location of the entire SDRAM address range.

In Inline ECC configurations, the register `SBRCTL.scrub_interval[12:0]` counts in multiples of 512 `sbr_clk` cycles (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). Value of ‘1’ counts 512 * `sbr_clk` cycles, value of 2 counts 1024 * `sbr_clk` cycles in between each scrub command, and so on.

A value of ‘0’ is a special case where the scrub read commands are issued back-to-back.

In Inline-ECC configurations, whenever `SBRCTL.scrub_interval` is programmed to ‘0’, the scrubber does not issue any RMW command to DDRC if single bit ECC errors are detected.

In Inline ECC configurations, the internal tSCRUBI counter width is 22 bits due to the interval programming granularity being 9 bits (x512). This counter width (22 bits) is derived from the highest controller clock frequency of 1066 MHz, the longest scrub period of 12 hours, and the lowest memory size. The counter granularity (9 bits) is derived from the lowest controller clock frequency of 200 MHz, the shortest scrub period of 2 hours and the highest memory size. The scrub period ranges from 2 hours to 12 hours depending on the tSCRUBI setting, memory size, and clock frequency.



Note In dual-channel configurations, all register fields of SBRCTL except for SBRCTL.scrub_en and SBRCTL.scrub_en_dch1 are shared between both channels. SBRWDATA0, SBRWDATA1 are also shared.

13.2.3 Restrictions on Scrub Interval With Auto Power-down/Auto Self-refresh Idle Time

When Auto Power-down and Auto Self-refresh is enabled, make sure the scrub interval (SBRCTL.scrub_interval) is greater than (actual time value must be greater) the power-down idle time (PWRTMG.powerdown_to_x32) and self-refresh idle time (PWRTMG.selfref_to_x32). Otherwise, the Scrubber may prevent the DDR memory from going to power-down and self-refresh by initiating scrubber periodic transactions before the idle-time expiry.

- Power-down timer max=1024 DRAM clock cycles - 256/512 Core clocks (based on DFI Frequency ratio 1:4/1:2)
- Self-refresh max idle time=8192 DRAM clock cycles - 2048/4096 Core clocks (based on DFI Frequency ratio 1:4/1:2)
- Scrub interval granularity is 512 sbr_clk cycles in Inline ECC mode. Scrub interval=(SBRCTL.scrub_interval * 512) sbr_clk cycles. sbr_clk is the same as Core clock.
- Based on the system's requirements, either PWRTMG.powerdown_to_x32 or PWRTMG.selfref_to_x32 can be fixed and SBRCTL.scrub_interval can be picked, or vice versa.

13.2.4 Scrubber Normal Operation

The SBR is enabled by the SBRCTL.scrub_en register (see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). For Inline ECC a minimum of 8 bursts (1 block) is issued to the DDRC each time; the amount of burst sent is set by programming the SBRCTL.scrub_burst_length_nm[2:0] register. Each time tSCRUBI counter times out, a scrub command (or a burst of commands for Inline ECC) is pushed to a scrub queue (8-deep), which is then immediately issued to the DDRC. The SBR counters halt if more than 8 scrub commands are collected and pending inside the scrub queue due to DDRC being busy.

13.2.5 Scrubber Address Configuration

When the ECC Scrubber is enabled, the start and maximum addresses for which SBR generates commands must be programmed through the scrubber address registers.

The registers SBRSTART1.sbr_address_start_mask_1 and SBRSTART0.sbr_address_start_mask_0 have been provided to program the start address. The SBR block internally uses the SBRSTART1.sbr_address_start_mask_1 and SBRSTART0.sbr_address_start_mask_0 to determine the start address.

The registers SBRRANGE1.sbr_address_range_mask_1 and SBRRANGE0.sbr_address_range_mask_0 have been provided to program the range mask. The SBR block internally uses the SBRRANGE1.sbr_address_range_mask_1 and SBRRANGE0.sbr_address_range_mask_0 registers to determine the maximum address.

A second set of four registers is defined for dual-channel configurations for the CH1 SBR block. The registers are:

- SBRSTART0DCH1.sbr_address_start_mask_dch1_0
- SBRSTART1DCH1.sbr_address_start_mask_dch1_1
- SBRRANGE0DCH1.sbr_address_range_mask_dch1_0
- SBRRANGE1DCH1.sbr_address_range_mask_dch1_1

The Scrubber address registers are changed only when the scrubber is disabled (SBRCTL.scrub_en=0) and there are no scrubber commands in progress (SBRSTAT.scrub_busy=0). For more information, see “REGB_ARB_PORTp” in “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

In Inline ECC configurations, make note of the following settings:

- The registers SBRSTART0.sbr_address_start_mask_0, SBRSTART1.sbr_address_start_mask_1, SBRRANGE0.sbr_address_range_mask_0, and SBRRANGE1.sbr_address_range_mask_1 must always be set in the protected region based on the ECCCFG0.ecc_region_map, ECCCFG0.ecc_region_map_granu, and ECCCFG0.ecc_region_map_other.
- The minimum value for {SBRSTART1.sbr_address_start_mask_1, SBRSTART0.sbr_address_start_mask_0} can be '0', provided this address is in the protected region.
- The maximum value for {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} can be 'ECC region start address - 1', provided this address is in the protected region.
- For more information on the protected regions, see “[Selectable Protected Regions](#)” on page [342](#).



Note The registers SBRSTART0, SBRSTART1, SBRRANGE0, SBRRANGE1, SBRAADDRRESTORE0, SBRAADDRRESTORE1 and their DCH1 counter parts are all HIF addresses.

The programming guidelines for SBRSTART0.sbr_address_start_mask_0, SBRSTART1.sbr_address_start_mask_1, SBRRANGE0.sbr_address_range_mask_0, and SBRRANGE1.sbr_address_range_mask_1 are also applicable to SBRSTART0DCH1.sbr_address_start_mask_dch1_0, SBRSTART1DCH1.sbr_address_start_mask_dch1_1, SBRRANGE0DCH1.sbr_address_range_mask_dch1_0, and SBRRANGE1DCH1.sbr_address_range_mask_dch1_1 respectively. When these registers are being programmed, the memory capacity of CH1 SDRAM is taken into consideration.

13.2.6 Address Generation

In Inline ECC configurations, ECC Scrubber (SBR) generates addresses only within protected regions, it automatically skips unprotected regions and ECC region.

- If there are no protected regions within start address {SBRSTART1.sbr_address_start_mask_1, SBRSTART0.sbr_address_start_mask_0} and end address {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0}, SBR does not generate any command.
- If SBRRANGE0.sbr_address_range_mask_0 or SBRRANGE1.sbr_address_range_mask_1 or SBRSTART0.sbr_address_start_mask_0, or SBRSTART1.sbr_address_start_mask_1 is

programmed to values in the unprotected region or in the ECC region, then during reads or writes, ECC Scrubber continuously loops through valid addresses.

For example, if {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} is set to $2^n - 1$ where n is MEMC_HIF_ADDR_WIDTH, this address is in ECC region, for back to back scrubber reads or initialization writes (tSCRUBI=0), ECC Scrubber loops continuously through valid addresses.

13.2.7 Additional Notes for the ECC Scrubber

- When ADDRMAP12.nonbinary_device_density is set to 3'b001 or 3'b010 or 3'b011 or 3'b100 or 3'b101, any write or RMW request with row [MSB: MSB-1]==2'b11 is discarded, while the HIF output hif_wdata_ptr_addr_err is asserted. Also, any read request with row [MSB: MSB-1]==2'b11 is executed, by changing the row [MSB: MSB-1] from 2'b11 to 2'b10. The return data for such reads are masked to zeros, while the HIF output hif_rdata_addr_err is asserted.
 - The ECC Scrubber does not send transactions to the invalid addresses, but skips to the next valid address in the next cycle.
 - The signals SBRRANGE0.sbr_address_range_mask_0, SBRRANGE1.sbr_address_range_mask_1, SBRSTART0.sbr_address_start_mask_0, and SBRSTART1.sbr_address_start_mask_1 must be set only to the valid addresses.
- In Inline ECC configurations, if a Scrubber periodic read transaction has both correctable and uncorrectable errors on different lanes, the controller generates a correction RMW issued to that address. This RMW only corrects the correctable error in the data. The uncorrectable error is retained by corrupting the ECC of that data.
- In dual channel configurations with UMCTL2_DUAL_CHANNEL=1 and UMCTL2_DATA_CHANNEL_INTERLEAVE_EN=1, there are two instances of Scrubber. Both Scrubber instances can be enabled and run in parallel. At the Port Arbiter, both Scrubber modules can request a grant at the same time.
 - The registers SBRSTART0, SBRSTART1 and SBRSTART0DCH1, SBRSTART1DCH1 can have the same or different values respectively. Similarly, the registers SBRRANGE0, SBRRANGE1 and SBRRANGE0DCH1, SBRRANGE1DCH1 can have the same or different values.
 - To cover the entire address space, the start (SBRSTART0, SBRSTART1 and SBRSTART0DCH1, SBRSTART1DCH1) and range (SBRRANGE0, SBRRANGE1 and SBRRANGE0DCH1, SBRRANGE1DCH1) registers must be programmed to start and end of the HIF address space.
 - Internally each Scrubber instance manipulates the HIF address to issue transactions only to the respective DRAM channel. Each Scrubber instance uses the HIF address bit mapped to the DCH interleave bit ADDRMAP0.addrmap_dch_bit0 and then decides if it will issue the transaction to a particular address.
- In dual data channel interleaving configurations (UMCTL2_DATA_CHANNEL_INTERLEAVE_EN==1):
 - When CHCTL.dual_channel_en is '0': ADDRMAP0.addrmap_dch_bit0 is programmed to '63', that is, set as unused. In this case, only scrubber channel 0 can be enabled through SBRCTL.scrub_en.
 - When CHCTL.dual_channel_en is '1': ADDRMAP0.addrmap_dch_bit0 value is in valid range of 0 to 35. In this case, scrubber channel 0 and scrubber channel 1 can be enabled through SBRCTL.scrub_en and SBRCTL.scrub_en_dch1, respectively.

13.2.8 Status

If SBRCTL.scrub_interval is set to non-zero value and there are outstanding read commands on flight issued by the SBR, SBRSTAT.scrub_busy register is asserted. SBRSTAT.scrub_done is not asserted.

If SBRCTL.scrub_interval is set to '0',

- For reads, the ECC Scrubber continues to issue read commands and does not stop automatically after the programmed address range is covered. After the first round, the scrubber continues, and the controller status reads as SBRSTAT.scrub_done=1 and SBRSTAT.scrub_busy=1.
- For writes, the ECC Scrubber stops issuing write commands after the programmed address range is covered. After the first round, the scrubber stops, and the controller status reads as SBRSTAT.scrub_done=1 and SBRSTAT.scrub_busy=0.



In Inline ECC configurations:

- If {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} is set in the unprotected region or in the ECC region and SBRCTL.scrub_interval is 0, then during reads or writes, ECC Scrubber does not generate SBRSTAT.scrub_done status.
- If {SBRSTART1.sbr_address_start_mask_1, SBRSTART0.sbr_address_start_mask_0} or {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} is set in the unprotected regions, the DDRCTL behavior is not predictable.
- For example, if {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} is set to $2^n - 1$ where n is MEMC_HIF_ADDR_WIDTH, this address is in ECC region, for back to back scrubber reads or initialization writes, the ECC scrubber does not generate SBRSTAT.scrub_done.

13.2.9 Hardware Controlled Low-Power Operation

If the DDRCTL is in one of the hardware controlled low-power modes, the SBR continues to operate automatically without any software intervention. This behavior can be disabled if SBRCTL.scrub_during_lowpower is set to '0' (see "REGB_ARB_PORTp" in "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). There are two such modes – automatically initiated by idleness and initiated by the hardware low-power interface.

The SBRCTL.scrub_burst_length_lp[2:0] register is specified to determine the number of back-to-back scrub commands, up to 1024, that can be issued together during low-power operation. In Sideband ECC/Advance ECC configurations during normal operation mode of the controller (not power-down or self-refresh modes), scrub_burst_length_nm must be programmed to '1' and only one scrub command is generated at every scrub interval.

- Automatically Initiated Low-Power Mode: if SBRCTL.scrub_during_lowpower is set to '1', scrub_burst_length_lp number of scrub reads are issued every (scrub_burst_length_lp * scrub_interval) cycles when the controller is in any of the automated low-power modes (STAT.operating_mode=10 (power-down) or 11 (self-refresh)). If in self-refresh, STAT.selfref_type must also be set to 11; then issuing of the scrub read automatically wakes up the controller. Note that the SBR cannot put the SDRAMs back to low-power; if the idleness condition continues, the automated low-power entry is automatically initiated again.
- Low-Power Interface Initiated Low-Power Mode: the DDRCTL can be put into self-refresh through the hardware low-power interfaces. If the low-power mode is entered through this interface, it has to be exited by the same way. If scrub_during_lowpower is set to '1', every

(scrub_burst_length_lp * scrub_interval) cycles scrub_burst_length_lp number of scrub reads are attempted by asserting cactive_in_ddrc signal which in turn asserts cactive_ddrc output signal. The SBR is one of the several possible reasons for the DDRCTL to request exit from this low-power state by asserting cactive_ddrc. It is assumed that the external low-power controller initiates the exit upon receiving cactive_ddrc high.

13.2.10 Software Controlled Low-Power Operation

If the DDRCTL is in one of the software controlled low-power modes such as self-refresh power-down, it is recommended to follow the following procedure:

- Before entering into any of the software controlled low-power modes, disable the SBR by setting SBRCTL.scrub_en to '0' (see "REGB_ARB_PORTp" in "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).
- If scrubbing is required in this low-power mode, the whole address range can be quickly scrubbed by issuing back-to-back scrub read commands.
 - Software must wake up the DDRCTL and initialize the SDRAM/PHY as necessary.
 - To start the SBR in an accelerated rate, set scrub_interval to '0' and enable the SBR.
 - Once the SBR scrubs the entire SDRAM range, sbr_done_intr interrupt signal is asserted. In addition, SBRSTAT.scrub_done software status bit is set which can be polled by the software. The scrubbing operation continues until SBR is disabled. Note that sbr_done_intr signal and scrub_done register are never set if the SBR is operating in its normal mode (scrub_interval > 0). System can monitor sbr_done_intr interrupt to be set or poll the SBRSTAT.scrub_done bit to be 1.
 - If the software does not want to wait for the complete scrubbing operation to finish, scrub_interval can be set to a non-zero value anytime. The SBR re-starts the scrub commands from the start address.
 - Disabling the SBR or setting the scrub_interval back to its normal operating value (tSCRUBI) which is greater than 0 clears both the interrupt signal and the status bit.
- After software controlled self-refresh is disabled, software must poll the STAT.operating_mode. Once STAT.operating_mode changes from self-refresh to normal, then Scrubber can be enabled.

13.2.11 Initialization Writes

The previous sections refer to scrub read commands. The SBR block can be used to initialize the memories with a defined pattern. SBRCTL.scrub_cmd_type must be programmed to '1' (see "REGB_ARB_PORTp" in "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

SBRWDATA0 and SBRWDATA1 registers (exists if MEMC_DRAM_DATA_WIDTH==64¹) can be programmed with the desired pattern.

Scrubber gets its write data pattern from these SBRWDATA0 and SBRWDATA1 registers.

Since the data in these registers represents the DRAM Width data, software must ensure that in HBW only the lower half of the data has valid bytes and disabled upper half is zero padded.

Similarly in quarter bus width mode, software must ensure only the lower quarter bytes have valid data and the disabled bytes are zero padded.

1. MEMC_DRAM_DATA_WIDTH must be set to '16' or '32' in LPDDR5/4/4X Controller.

In Inline ECC configurations, the registers {SBRSTART1.sbr_address_start_mask_1, SBRSTART0.sbr_address_start_mask_0} and {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} must be programmed in the protected region based on the ECCCFG0.ecc_region_map, ECCCFG0.ecc_region_map_granu, and ECCCFG0.ecc_region_map_other.

- The minimum value for {SBRSTART1.sbr_address_start_mask_1, SBRSTART0.sbr_address_start_mask_0} can be 0, provided this address is in the protected region.
- The maximum value for {SBRRANGE1.sbr_address_range_mask_1, SBRRANGE0.sbr_address_range_mask_0} can be 'ECC region start address - 1', provided this address is in protected region.
- For more information on the definition of protected regions, see “[Selectable Protected Regions](#)” on page [342](#).

After PHY and SDRAM initialization, perform the following steps:

1. If MEMC_INLINE_ECC is defined and ECC is enabled, set ECCCFG1.ecc_parity_region_lock to '1'.
2. Ensure that PCTRL.port_en are set to '0', block the AXI ports from taking the transaction.
3. If UMCTL2_OCECC_EN is defined and On-chip ECC is enabled (OCECCCFG0.ocecc_en=1), set OCECCCFG0.ocecc_en to '0'. This is required to avoid On-chip ECC error interrupt.
4. Disable Auto power-down and Auto self-refresh if it is enabled (PWRCTL.powerdown_en=0 and PWRCTL.selfref_en=0)
5. Set SBRCTL.scrub_cmd_type=1.
6. Set SBRCTL.scrub_interval=0.
7. Set the desired pattern through SBRWDATA0 and SBRWDATA1 registers.
8. Enable the SBR by programming SBRCTL.scrub_en=1.
9. System can monitor sbr_done_intr interrupt to be set or poll the SBRSTAT.scrub_done bit to be '1'.
10. Poll SBRSTAT.scrub_busy=0 (all scrub writes data have been sent).
11. Disable SBR by programming SBRCTL.scrub_en=0.
12. Enable Auto power-down and Auto self-refresh if required (PWRCTL.powerdown_en=1 and PWRCTL.selfref_en=1).
13. If UMCTL2_OCECC_EN is defined, enable On-chip ECC by programming OCECCCFG0.ocecc_en to '1'.
14. Prepare for normal scrub operation (Reads) by programming SBRCTL.scrub_cmd_type=0 and SBRCTL.scrub_interval to tSCRUBI.
15. Enable the SBR by programming SBRCTL.scrub_en=1.
16. Enable AXI ports by programming PCTRL.port_en=1 .



Note You must have valid data and ECC data in the SDRAM before putting the scrubber in Read mode.



`SBRSTAT.scrub_done/sbr_done_intr` is generated when the initialization write is completed to the scrubber end address (default end address or end address as defined by `SBRRANGE1.sbr_address_range_mask_1`, and `SBRRANGE0.sbr_address_range_mask_0`). If scrubber cannot write to this end address, `scrub_done` is not asserted. You need to make sure that the end address is a valid address where scrubber can execute a write.

13.2.12 Scrubber Credit Interface

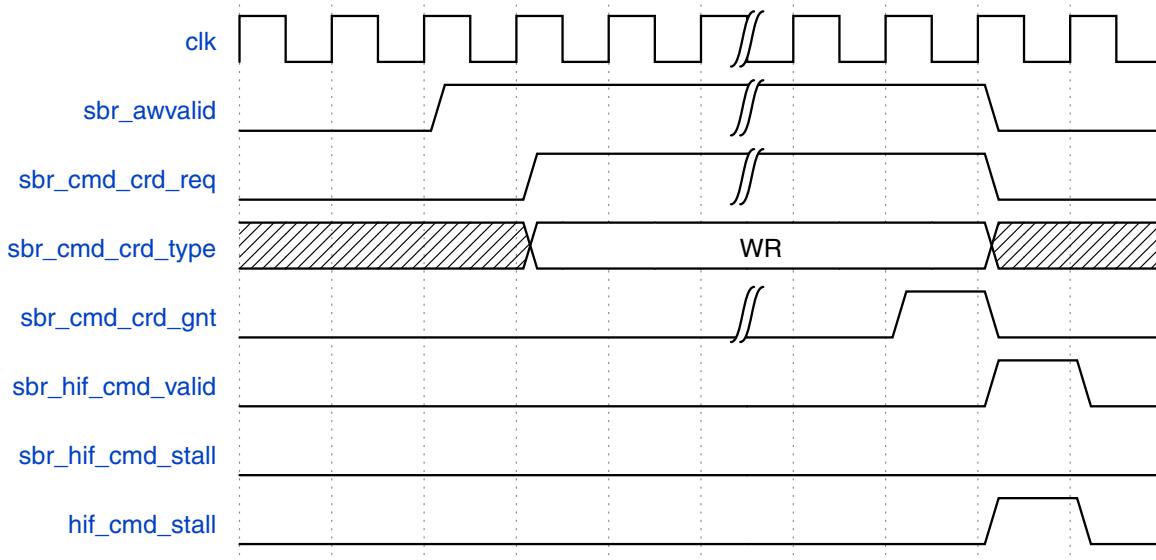
This interface exists in HIF configurations (`UMCTL2_INCL_ARB==0`) with ECC scrubber enabled.



Limitation: the combination of `UMCTL2_SBR_EN==1` and `UMCTL2_INCL_ARB==0` has only been tested in configurations where `UMCTL2_DUAL_HIF` is 1.

When the scrubber initiates a read or write request, a credit is first requested on the scrubber credit interface. For instance, `sbr_cmd_crd_req` is asserted while `sbr_cmd_crd_type` indicate type of credit. Credit grant is indicated by asserting `sbr_cmd_crd_gnt` by the external credit manager. Once granted, a valid command is driven on the scrubber HIF interface, while the external HIF interface is stalled.

Figure 13-22 Scrubber Credit Interface



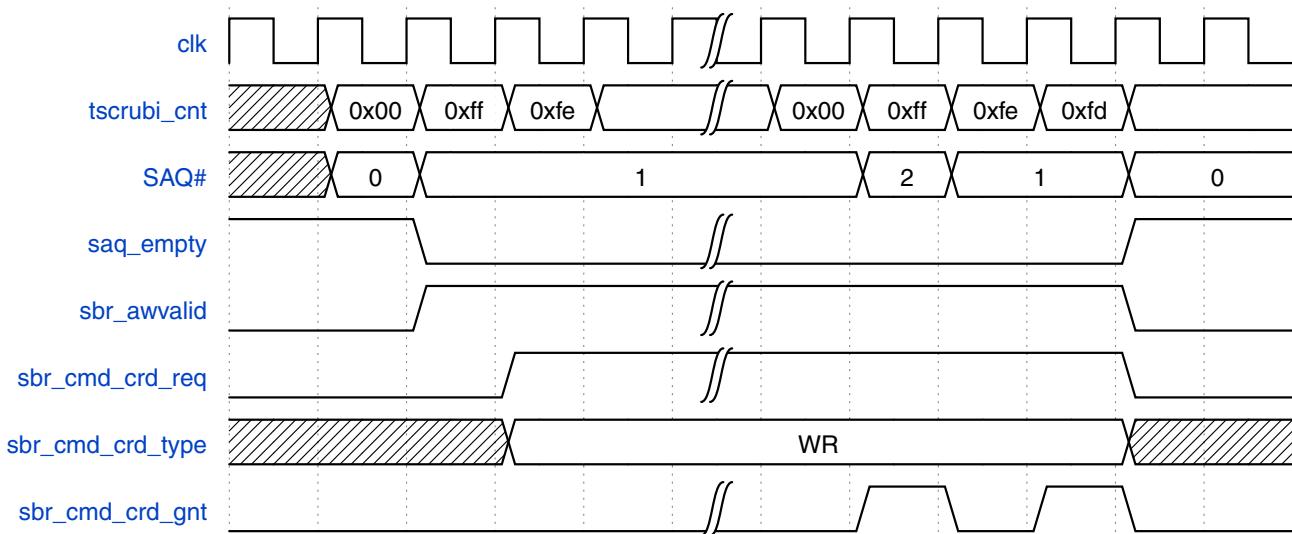
The delay inserted before granting a request is system dependent. The scrubber has an internal FIFO, Scrubber Address Queue (SAQ), which can store up to 8 pending requests and allows the scrubber to handle delays on the scrubber credit interface. A separate FIFO is used for read-modify-write requests (RMW FIFO).

As shown in [Figure 13-23](#), every time the scrub interval elapses (`tscrubi_cnt=0x0`), the scrubber pushes a request into SAQ. While the queue is not empty, the scrubber initiates requests. After one clock cycle, the request is propagated to the scrubber credit interface (`sbr_cmd_crd_req` asserted). If long delays occur at the scrubber credit interface, and scrubber makes a new request before the first one is granted, the request is not lost and is pushed to the queue (SAQ#=2).

When the first request is granted, the corresponding entry is removed from the queue (SAQ#=1), but sbr_awvalid (Scrubber write command valid) remains asserted, since SAQ is still not empty.

If the scrubber makes 8 consecutive requests without any of these being granted, the SAQ FIFO is filled, the scrubber is stalled and stops generating any additional requests.

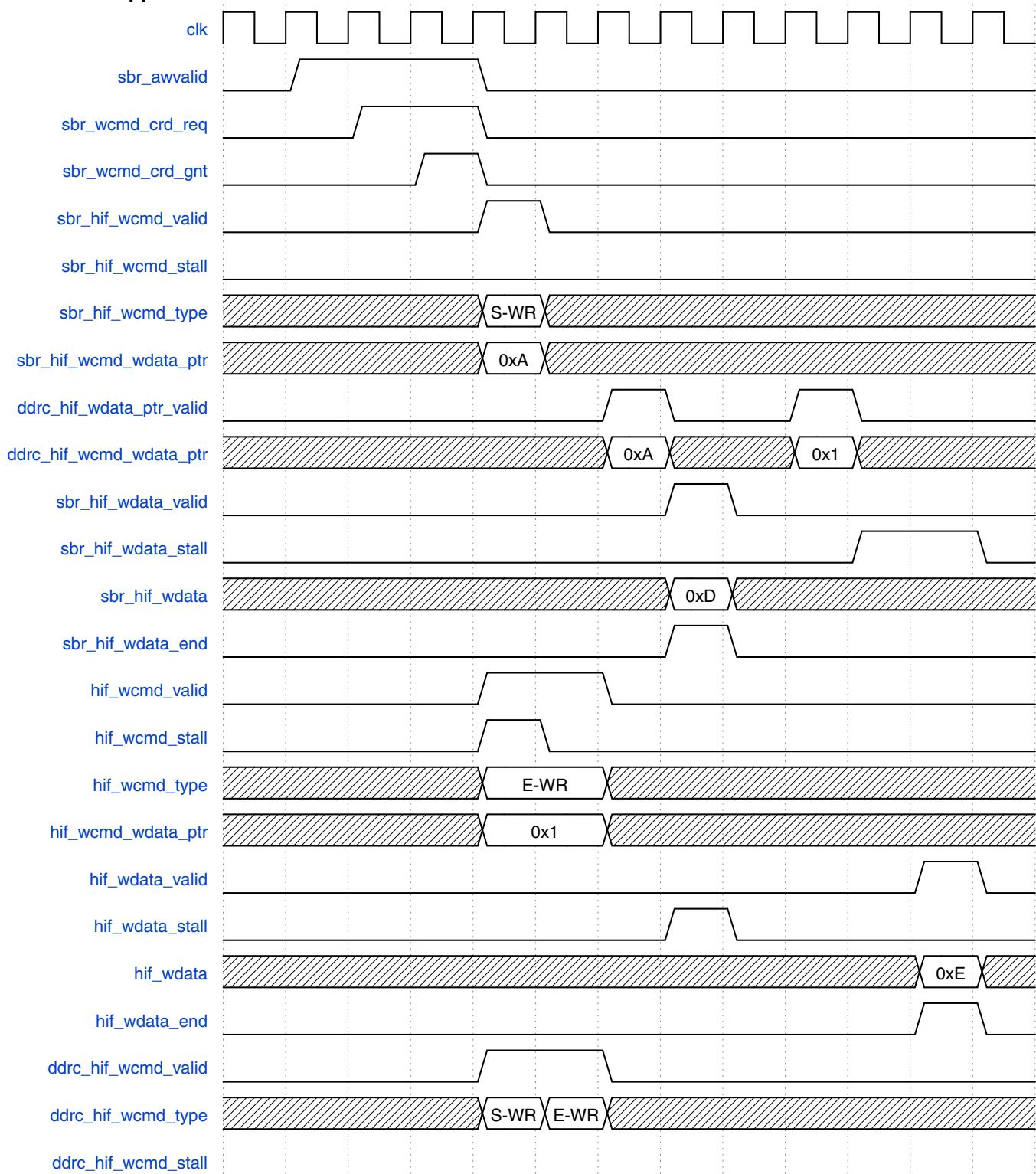
Figure 13-23 Scrubber Credit Interface Delay



When an external HIF request is issued at the same time with a scrubber HIF request (made by scrubber after a credit was granted), scrubber request takes priority, as shown in [Figure 13-24](#).

[Figure 13-24](#) illustrates a scenario in which both the scrubber HIF and the external HIF interfaces attempt a write request at the same time. The scrubber asserts `sbr_hif_wcmd_valid` after it was granted a credit. The external HIF interface asserts `hif_wcmd_valid` at the same time, but `hif_wcmd_stall` was also asserted (internal FIFO is not empty), delaying the command acceptance. Once the scrubber command was accepted, the external HIF command request can proceed. The diagram shows how the valid signal from the HIF interface is taking priority and is propagated to DDRC (`ddrc_hif_wcmd_valid`): scrubber (S-WR), followed by external HIF (E-WR). Note that S-WR and E-WR represent the same write type, and different names were given only for illustration purposes.

Overlapping read commands are handled similarly.

Figure 13-24 Overlapped SBR HIF / External HIF Write

13.2.12.1 External Credit Granting

This section provides recommendations on credit granting for the credit agent.

From the scrubber perspective, the DDRC can be either in low-power, or in normal mode of operation. This is reflected on the `sbr_state` output:

- 3'b001: Disabled
- 3'b010: Normal operating mode
- 3'b100: Hardware Controlled Low-Power mode

For more information about `sbr_state`, refer to the "Signal Descriptions" chapter.

When the DDR controller is in Hardware Controlled Low-Power mode (`sbr_state=3'b100`), the scrubber can still make requests if the `SBRCTL.scrub_during_lowpower` bit is set to '1'. In this case, the external credit agent should immediately grant any requests made by the scrubber.

When the DDR controller is in normal operating mode (`sbr_state=3'b010`), the external credit agent can consider the scrubber as a low-priority consumer.

The scrubber does not generate any new requests in disabled state. However, any outstanding requests in scrubber SAQ or RMW FIFOs, that were generated prior to disabling the scrubber, may continue to request credits.

The external credit manager should decrement a credit for every cycle when a grant is asserted. It should also make sure that the grant is asserted only for valid requests. Any excessive grants can lead to credits getting lost. The scrubber waits until the completion of all outstanding requests before deasserting `SBRSTAT.scrub_busy` to '0'. Therefore, the external agent must grants credits for each request, even when the scrubber is disabled.

13.2.12.2 Signals Related To Scrubber Credit Interface

The following are the signals related to the SBR Credit Interface for single HIF, dual channel configurations:

- `sbr_cmd_crd_req`
- `sbr_cmd_crd_type`
- `sbr_cmd_crd_gnt`
- `sbr_cmd_crd_req_dch1`
- `sbr_cmd_crd_type_dch1`
- `sbr_cmd_crd_gnt_dch1`
- `sbr_state`
- `sbr_state_dch1`

In case of dual HIF interface, two scrubber credit interfaces are connected to accommodate simultaneous read and write requests issued by the scrubber. The following signals are used in those configurations:

- `sbr_rcmd_crd_req`
- `sbr_rcmd_crd_type`
- `sbr_rcmd_crd_gnt`
- `sbr_wcmd_crd_req`

- sbr_wcmd_crd_type
- sbr_wcmd_crd_gnt
- sbr_rcmd_crd_req_dch1
- sbr_rcmd_crd_type_dch1
- sbr_rcmd_crd_gnt_dch1
- sbr_wcmd_crd_req_dch1
- sbr_wcmd_crd_type_dch1
- sbr_wcmd_crd_gnt_dch1
- sbr_state
- sbr_state_dch1

For more information about these signals, refer to the "Signal Descriptions" chapter.

13.3 On-Chip Parity (OCPAR)

This topic contains the following sections:

- “Overview of On-Chip Parity” on page 386
- “Enabling On-Chip Parity” on page 387
- “Write Data Parity” on page 387
- “Read Modify Write Operation” on page 390
- “Signals Related to On-Chip Parity” on page 391
- “Registers Related to On-Chip Parity” on page 391

13.3.1 Overview of On-Chip Parity

In the DDRCTL, all data paths from end-to-end are protected using parity. Address path protection is also provided at the AXI inputs. Parity has the capability to detect an odd number of errors. Once error is detected, the mechanism can neither locate nor correct the errors.

Though ECC and parity are connected to create an overlapped protection, they can be enabled independently. However, if parity is enabled and ECC is disabled, some of the parity functionality may not be available.

Parity functionality is supported only for AXI configurations.

Figure 13-25 On-Chip Parity Overview in Sideband ECC or ECC Disabled Configurations

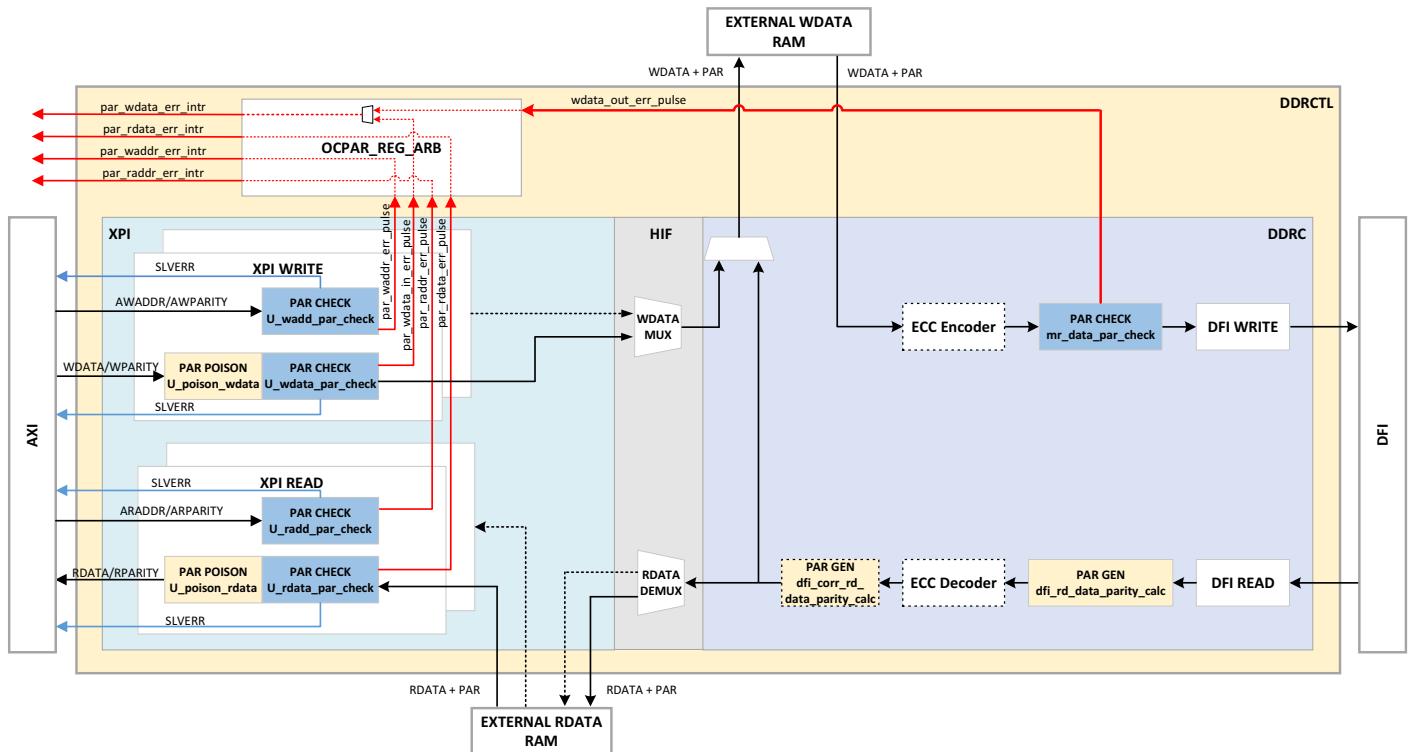
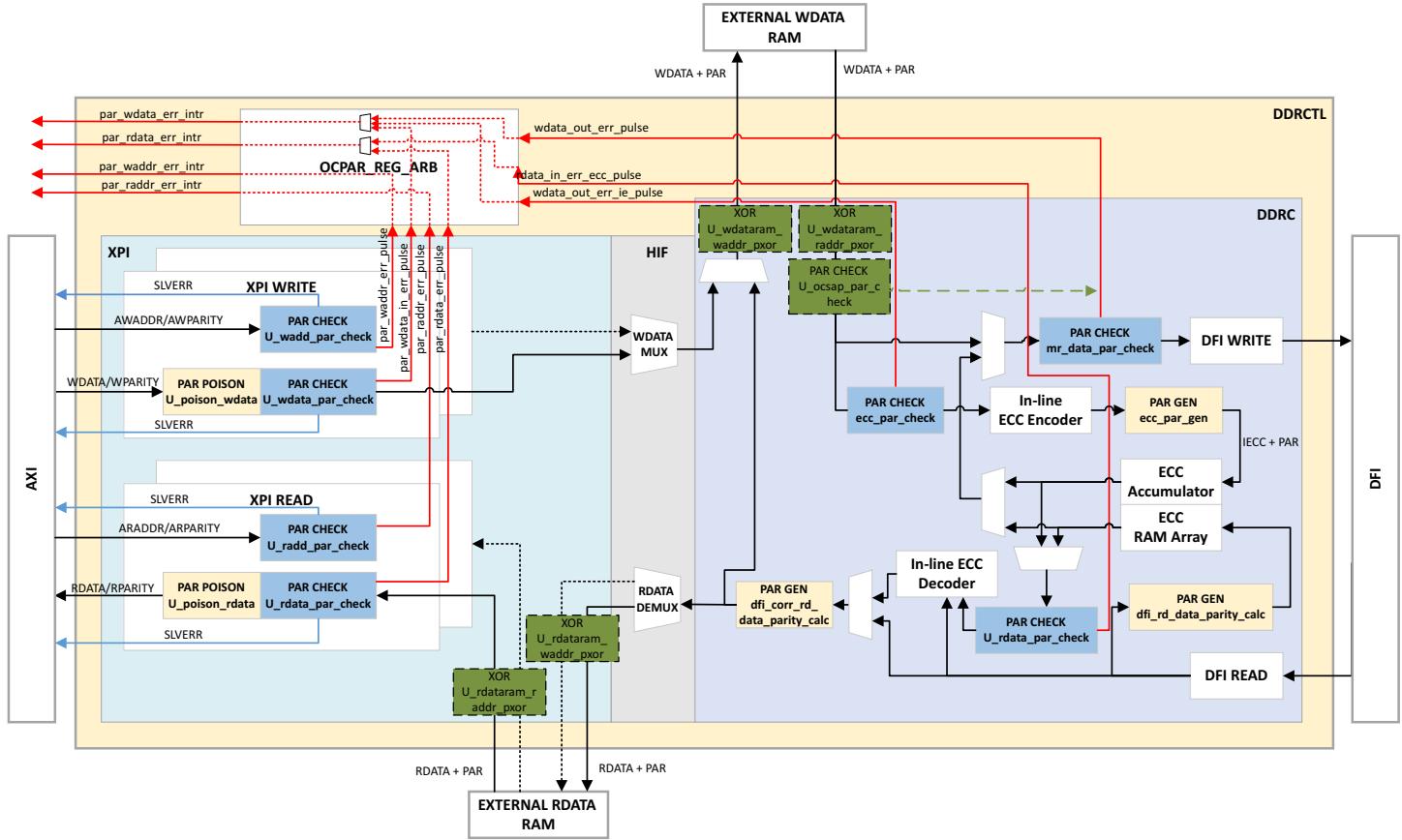


Figure 13-25 shows the on-chip parity architecture in sideband ECC configurations (MEMC_ECC_SUPPORT>0 and MEMC_SIDEBAND_ECC=1) or configurations without ECC support (MEMC_ECC_SUPPORT=0).

Figure 13-26 shows the on-chip parity architecture in Inline ECC configurations (MEMC_INLINE_ECC=1).

Figure 13-26 On-Chip Parity Overview in Inline ECC Configurations



- The blocks in dotted line are only present in ECC configurations (that is, MEMC_ECC_SUPPORT > 0).
- In Figure 13-25 and Figure 13-26, Parity Poisoning inside DDRC block is present after every PAR GEN blocks.
- The highlighted blocks in green are present when DDRCTL_OCSAP_EN=1.

13.3.2 Enabling On-Chip Parity

To enable on-chip parity, set the hardware configuration parameter UMCTL2_OCPAR_EN to '1'.

13.3.3 Write Data Parity

For every write data beat, the DDRCTL receives one bit of parity per byte. Write data parity on AXI interface is carried through a sideband signal called wparity_n[UMCTL2_PORT_NBYTES_n-1:0].

When write data parity errors are detected, they generate maskable interrupt signal par_wdata_err_intr, that can be disabled with OCPARCFG0.par_wdata_err_intr_en. OCPARSTAT1.par_wdata_err_intr[15:0] register logs AXI port/ports on which parity error is detected. OCPARSTAT2.par_wdata_out_err_intr[0] register logs if error detected on DFI. Additional check at the DFI is present in case Inline ECC is used (MEMC_INLINE_ECC=1), this protects the ECC path

before the encoder; in case a parity error is detected, interrupt `par_wdata_err_intr_n` is asserted. `OCPARSTAT2.par_wdata_out_err_intr[1:0]` register logs whether the error came from the data path (bit 0) and/or the ECC path (bit 1).

Write transactions with parity errors on the AXI interface are returned with a SLVERR response on AXI write response channel. Note that the DDRCTL treats all AXI transactions as bufferable regardless of the transaction attribute. In other words, if parity checker after the AXI interface is not bypassed (that is, if `OCPARCFG0.par_wdata_axi_check_bypass_en=0`), the write response is generated when the last write data beat is accepted inside the DDRC CAM. Therefore, AXI SLVERR write response is generated only based on parity checking on the AXI interface. Any parity check failure detected on the DFI interface is not signaled through SLVERR but signaled through the `par_wdata_err_intr` interrupt signal that can be disabled through the register `OCPARCFG0.par_wdata_err_intr_en`. The SLVERR response can be disabled through `OCPARCFG0.par_wdata_slverr_en`.

If ECC is enabled and there is a write data parity error on the DFI interface, the ECC check bits are corrupted for that write data beat. Note that write address is committed to the memory and write data is not masked if there is a parity error. It is up to the system through software to retry and write the correct data.

13.3.3.1 Output Check

The protection schemes, ECC and parity overlap, in other words, linked together. ECC is first calculated, and then the parity is terminated. To ensure this overlap, ECC encoder and the parity check are separated by a pipeline.

13.3.3.2 Input Check

The input parity check is done at the AXI interface, when both `wvalid_n` and `wready_n` is active. This provides data protection for all the upstream logic outside the controller. The status of the check is sent along with the write pointer and used for the response generation. As parity is not terminated, one bit per byte of data is appended for each beat.

13.3.4 Read Data Parity

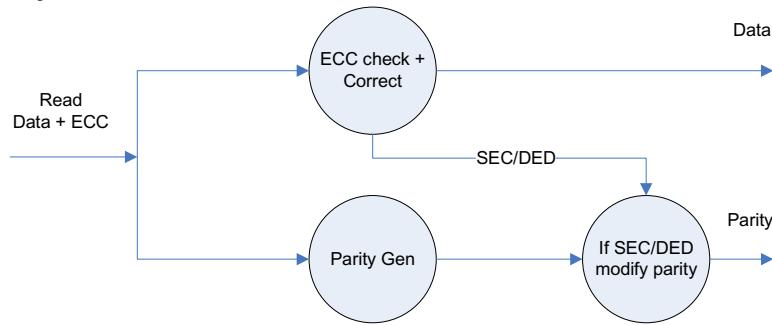
For every read data beat, the DDRCTL generates one bit of parity per byte. Read data parity on AXI interface is carried through a sideband signal called `rparity_n[UMCTL2_PORT_NBYTES_n-1:0]`.

Parity is checked at the AXI interface. Read transactions with read data parity errors are returned with a SLVERR response which can be disabled using `OCPARCFG0.par_rdata_slverr_en`.

In addition to the error response, read data parity errors when detected, generate a maskable interrupt (`par_rdata_err_intr`), that can be disabled through the register `OCPARCFG0.par_rdata_err_intr_en`. The `OCPARSTAT1.par_rdata_err_intr_n` registers log the AXI ports on which parity error is detected.

13.3.4.1 Parity Generation

Read data parity is generated from the same pipeline stage as the ECC decoder. As shown in [Figure 13-27](#), the paths are protected by overlapping ECC and parity scheme.

Figure 13-27 Read Data Parity Generation

13.3.4.2 Parity Check

Parity check is done at the AXI interface when `rvalid_n` is active.

If inline ECC is used, additional check is done at the DFI, this is done before the ECC decoder. This is to protect the ECC word which comes separately from the data. In case an error is detected, interrupt `par_rdata_err_intr` is asserted and `OCPARSTAT2.par_rdata_in_err_ecc_intr` register is updated.

13.3.4.3 Address Parity

The AXI address (`araddr`, `awaddr`) is protected by one bit of parity for the entire address or by one bit of parity for each address byte, depending on hardware parameter `UMCTL2_OCPAR_ADDR_PARITY_WIDTH` setting. Address parity is carried by a sideband signal `arparity`/`awparity`. The address parity is checked and terminated within the XPI. Write address parity check is performed when `awready_n==1` and `awvalid==1`. Likewise, read address parity check is performed when `arready_n==1` and `arvalid_n==1`. Address parity errors are signaled to the master through AXI SLVERR response which can be disabled by `OCPARCFG0.par_addr_slverr_en` register.

In addition to the error response, address parity errors generate maskable interrupts `par_waddr_err_intr` and `par_raddr_err_intr` respectively, for write address and read address errors. Interrupts can be disabled through registers `OCPARCFG0.par_waddr_err_intr_end` and `OCPARCFG0.par_raddr_err_intr_en`.

This feature uses the same infrastructure as “[Transaction Poisoning](#)” on page 57. Therefore, addresses with parity errors are effectively poisoned leading to similar operation.

Address parity errors are signaled through maskable interrupts (`par_wdata_err_intr`, `par_raddr_err_intr`), that can be disabled with `OCPARCFG0.par_waddr_err_intr_en` / `OCPARCFG0.par_raddr_err_intr_en`. `OCPARSTAT0.par_waddr_err_intr[15:0]` and `OCPARSTAT0.par_raddr_err_intr[15:0]` register logs AXI port/ports on which address parity error is detected.

13.3.4.4 Embedded SRAM Protection

External SRAMs interfaces have data and byte parity.

Output parity is driven by the controller along with the data going to the external SRAMs (there is no checker inside the controller for these outputs):

- `wdataram_din_par`: parity for `wdataram_din` (write data RAM input, valid bytes according to `wdataram_mask`)
- `rdataram_din_par_n`: parity for `rdataram_din_n` (port n read data RAM input, all bytes valid)

You must drive the input parities to the controller based on the data read from each SRAM (controller expects the correct parity for each data byte whenever the data read is valid):

- `wdataram_dout_par`: parity for `wdataram_dout` (write data RAM output)
- `rdataram_dout_par_n`: parity for `rdataram_dout_n` (port n read data RAM output)

Note that `rdataram_din_par_n` and `rdataram_dout_par_n` are available only if the external RAM for port n is enabled; otherwise the parity is buffered inside the RRB (internal) along with the read data. In this case, you need not to perform any action.

13.3.5 Read Modify Write Operation

Read modify write operations make use of the read path and write path. As the read and write paths are properly protected, RMW operations are properly protected. In case there is an uncorrectable ECC error on the read part, the parity is corrupted (based on register setting) on the read data path. This leads to automatic ECC corruption on the write.

If parity is not enabled, controller still corrupts the write ECC for RMW directly upon detection of uncorrectable error on the read part of the transaction.

13.3.6 Parity Poisoning

The DDRCTL provides the ability to inject parity errors in the data path. This on-chip parity poisoning can be used to test the feature and system response to parity errors.

This functionality is enabled by setting the register `OCPARCFG1.par_poison_en`. The parity error can be injected in the following data interfaces with any combination:

- Read Data – Parity error is injected after the generation at the DFI interface enabled by `OCPARCFG1.par_poison_loc_rd_dfi` register. If Inline ECC is enabled, in the read data path, after the parity generation, data can go through two different paths (data or ECC). Setting the register `OCPARCFG1.par_poison_loc_rd_iecc_type` gives the possibility to poison either one or the other.
- Read Data – Parity error is injected at the AXI interface after the check, only one port at a time, enabled by `OCPARCFG1.par_poison_loc_rd_port` register. An error injected here is not captured by the controller.
- Write Data – Parity error is injected at the AXI interface before the check, only one port at a time, enabled by `OCPARCFG1.par_poison_loc_wr_port` register.

One-shot triggering is supported, which means that error is injected for the first valid data either after `OCPARCFG1.par_poison_en` is set to '1' or after the interrupt clear for that path is asserted. This means that only the parity corresponding to the first beat of the transaction on DFI/AXI is poisoned.

- Write '1' to `OCPARCFG0.par_wdata_intr_clr` of both channels CH0 and CH1 (applicable only for dual channel configuration) to clear the interrupt. This allows to re-enable poisoning over another transaction at the Write Data AXI interface.
- Write '1' to `OCPARCFG0.par_rdata_intr_clr` of both channels CH0 and CH1 (applicable only for dual channel configuration) to clear the interrupt. This allows to re-enable poisoning over another transaction at the Read Data DFI and AXI interfaces.

AXI traffic restrictions for on-chip parity poisoning:

- AXI burst type must be INCR.
- AXI address must be aligned to SDRAM burst (For more information on AXI to SDRAM address translation, see “[Address Mapping](#)” on page [153](#)).
- AXI address must not access invalid LPDDR4 row address (For more information, see “[Non-binary Device Densities](#)” on page [162](#)).
- AXI Write burst must not cause RMW on to HIF. To avoid RMW generation for AXI Write burst, see “[Read-Modify-Write \(RMW\) Generation](#)” on page [46](#).
- AXI address must be aligned to INLINE_ECC block boundary (When `MEMC_INLINE_ECC==1`. For more details on Inline ECC block boundary, see “[Inline ECC Support](#)” on page [334](#)).
- One AXI transaction must not access multiple ECC regions/blocks (when `MEMC_INLINE_ECC==1`. For more details about Inline ECC regions/block, see “[Inline ECC Support](#)” on page [334](#)).
- In Inline ECC configurations, for poisoning inside DDRC block, the AXI must perform read accesses to all the locations that are written by AXI from the time poisoning has been enabled.



Note Note that all the AXI traffic restrictions apply only if the controller works in the full bus width mode, done by appropriately setting `MSTR0.data_bus_width` (see “[REGB_DDRC_CH0 Registers](#)” in the “[Register Descriptions](#)” chapter of the [Synopsys LPDDR5X/5/4X Memory Controller Reference Manual](#)).

All automatic MRR and read retry commands shall be disabled before enabling for on-chip parity poisoning as follows:

- Disable DQS oscillator by setting `DQSOSCCTL0.dqsosc_enable=0` (see “[Controller Assisted Drift Tracking by MRR Snooping \(LPDDR5\)](#)” on page [235](#)).
- Disable or Pause Automatic Temperature Derating by setting `DERATECTL0.derate_enable=0` or `DERATECTL0.derate_mr4_pause_fc=1` (see “[Automatic Temperature Derating](#)” on page [219](#)).

13.3.7 Signals Related to On-Chip Parity

For more information on signals related to the on-chip parity, see:

- “[AXI Port n Write Address On-Chip Parity Signals](#) (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)”
- “[AXI Port n Write Data On-Chip Parity Signals](#) (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)”
- “[AXI Port n Read Address On-Chip Parity Signals](#) (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)”
- “[AXI Port n Read Data On-Chip Parity Signals](#) (for $n = 0; n \leq \text{UMCTL2_A_NPORTS}-1$)”

13.3.8 Registers Related to On-Chip Parity

The following are the registers related to the on-chip parity:

- `OCPARCFG0`
- `OCPARCFG1`
- `OCPARSTAT0`
- `OCPARSTAT1`

- OCPARSTAT2

For more information about these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.4 Registers Parity Protection (REGPAR)

This topic contains the following sections:

- “Overview of Registers Parity Protection (REGPAR)” on page 393
- “Enabling Registers Parity Protection” on page 393
- “Register Parity Generation” on page 393
- “Register Parity Checking” on page 395
- “Register Parity Poisoning” on page 396
- “Registers Related to REGPAR” on page 397

13.4.1 Overview of Registers Parity Protection (REGPAR)

This feature is available for you only with the DWC-AC-LPDDR54-CONTROLLER Automotive Product (AC) license.

Register parity protection consists of the following two actions:

- Register Parity generation (on APB clock domain for configuration registers and on DDRCTL/application port clock domains for status registers).
- Register Parity checking (on APB clock, DDRCTL clock, Application port clock for configuration registers and on APB clock domain for status registers).



`pclk_rp` and `presetn_rp` are required for this REGPAR functionality. `presetn_rp` must have the same source as `presetn` (must be asserted at the same moment). `pclk_rp` must be driven synchronously to `pclk` and must be free-running.

13.4.2 Enabling Registers Parity Protection

To enable Registers Parity Protection, set the hardware configuration parameter `UMCTL2_REGPAR_EN` to ‘1’.

Support for Register parity can be enabled/disabled from software using the register field `REGPARCFG.reg_par_en`.

13.4.3 Register Parity Generation

This section contains the following subsections:

- “Configuration Registers” on page 393
- “Status (Read-only) Registers” on page 394
- “Register Parity Checking” on page 395
- “Register Parity Poisoning” on page 396
- “Registers Related to REGPAR” on page 397

13.4.3.1 Configuration Registers

In the following situations the parity information is generated on APB clock domain:

- At reset, based on default reset value.

- Whenever you update the register.
- Whenever you enable the register parity protection through REGPARCFG.reg_parity_en (see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).
- Whenever you disable the register parity poisoning through REGPARCFG.reg_par_poison_en (see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

Parity information is generated based on 32-bit register value. The number of parity bits is controlled by the hardware parameter UMCTL2_REGPAR_TYPE, as following:

- UMCTL2_REGPAR_TYPE=0 - for every 32-bit register, one-bit parity is calculated.
- UMCTL2_REGPAR_TYPE=1 - for every 32-bit register, 4-bit parity is calculated, one bit for every byte of the register.

This information is appended to the payload and synchronized together with the DDRCTL clock or application port clock domain.

R/W1C and R/W1S register fields are masked when calculating parity information for the entire configuration register (for more details about register field types see the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). This is needed to avoid false error detection caused by the fact that these register fields are automatically cleared. A separate mechanism is used for protecting these register fields. Flip-flops in R/W1S and R/W1C logic are duplicated and error is detected by comparing the outputs of original and duplicated flip-flops.

13.4.3.2 Status (Read-only) Registers

There are two types of read-only registers:

- Synchronized to the APB clock as a whole (using one CDC for all 32 bits, like configuration registers)
- Synchronized to the APB clock field by field (using separate CDC for each field)

Parity information is calculated and propagated differently for the two types of status registers.

For registers synchronized to the APB clock as a whole, the calculation mechanism is similar to the configuration register parity calculation. Parity information is generated based on 32-bit register value. The number of parity bits is controlled by the hardware parameter UMCTL2_REGPAR_TYPE, as following:

- UMCTL2_REGPAR_TYPE=0 - for every 32-bit register, one-bit parity is calculated.
- UMCTL2_REGPAR_TYPE=1 - for every 32-bit register, 4-bit parity is calculated, one bit for every byte of the register.

This information is appended to the payload and synchronized together with the APB clock domain.

For registers synchronized field by field to APB clock, register parity information is calculated differently. One parity bit is calculated for each field, regardless of the hardware parameter UMCTL2_REGPAR_TYPE. This information is appended to the field’s payload and synchronized together with the APB clock domain. After synchronization, a cumulative 1-bit error is calculated (OR between parity checking result of each). This cumulative error information is appended to the register value on APB clock domain (1 if any of the fields has parity error, 0 if none of the fields have parity error).

After synchronization, parity is checked for each field, however an error is not flagged yet. A cumulative 1-bit error is calculated based on error information of all fields (cumulative error is 1 if at least one field has

parity error, 0 if none of the fields has parity error). This 1-bit error indication is appended to the status register value.

13.4.4 Register Parity Checking

Parity checking is performed independently and executed concurrently in all the clock domains.

A parity error is flagged by the:

- Interrupt signal `reg_par_err_intr`, if enabled by `REGPARCFG.reg_par_err_intr_en`
- Interrupt signal `reg_par_err_intr_fault`, regardless of `REGPARCFG.reg_par_err_intr_en` setting
- `REGPARSTAT.reg_par_err_intr` register field, regardless of `REGPARCFG.reg_par_err_intr_en` setting

You can force the REGPAR error flagging by setting `REGPARCFG.reg_par_err_intr_force` to '1'. In this situation, a REGPAR error is flagged by:

- Interrupt signal `reg_par_err_intr`, if enabled by `REGPARCFG.reg_par_err_intr_en`
- `REGPARSTAT.reg_par_err_intr` register field, regardless of `REGPARCFG.reg_par_err_intr_en` setting

The interrupt is cleared by writing a '1' to `REGPARCFG.reg_par_err_intr_clr`. This field is automatically cleared.

13.4.4.1 Register Parity Checking on APB Interface Clock

Parity checking on the APB clock domain (excluding R/W1S and R/W1C register fields of configuration registers) is performed:

- Periodically for all configuration registers in round-robin order, one register being checked every APB clock cycle.
- Whenever a configuration register is read.

For periodic checking the existing register read multiplexing path is re-used. When APB read and register parity checking coincide, the read has a higher priority. This means that the round-robin mechanism is stopped and parity checking address is not incremented at this point. Register parity checking is instead performed on the register which is currently read.

If using this feature, before writing to a register, it is recommended to perform a spare APB read of this register prior to the APB write. This triggers a check on the register contents before being updated.

Depending on register type, parity checking is performed as follows:

- For configuration registers and status registers synchronized as a whole: When register is selected, parity is re-calculated and compared against the parity information appended to register payload.
- For status registers synchronized field by field: When the register is selected, the cumulative error information is directly used as an error indication.

Register parity error detection is flagged by the interrupt signals `reg_par_err_intr`, `reg_par_err_intr_fault`, and by the register `REGPARSTAT.reg_par_err_intr`.

Since register parity checking is performed one register at a time, errors are detected with latency. The worst case error detection latency can be calculated as:

```
tregpar_err_latency=MAX_RANGE / 4 * clock_period
```

where MAX_RANGE is the biggest address range among all existing register blocks. Currently all register blocks have an address range of 0x1000 (which translates in decimal to 4096), hence the worst case error detection latency is

```
tregpar_err_latency=4096/4*clock_period=1024*clock_period.
```

Register parity error on R/W1S or R/W1C register field is immediately flagged. Note, that register parity checking of these register fields is only performed on APB clock domain.

13.4.4.2 Register Parity Checking on DDRCTL Controller Clock and Application Port Clock

Register parity checking on DDRCTL controller and application port clock domains is performed for dynamic registers if the clocks are asynchronous to the APB clock. When clocks are asynchronous, CDC logic is instantiated and mirror registers are placed on destination clocks. Parity checking is needed to detect errors in CDC logic and mirror registers.

Parity information is synchronized to core_ddrc_core_clk and aclk_N together with the register value. Parity checking is performed every clock cycle on destination clock domain in parallel for all registers being crossed. If an error is detected, error information is synchronized back to the APB clock domain, where it is flagged by interrupt signals reg_par_err_intr, reg_par_err_intr_fault, and by the register REGPARSTAT.reg_par_err_intr (see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

Error detection latency in this case is determined by the amount of time it takes for error information to be synchronized to the APB clock domain.

Note, that parity checking of R/W1S and R/W1C register fields is not performed on controller clock or application port domains, because of the self-clear behavior of these fields.

13.4.5 Register Parity Poisoning

The DDRCTL provides the ability to inject register parity errors. This can be used to test the feature and system response to register parity errors.

Register parity poisoning is enabled by setting the register REGPARCFG.reg_par_poison_en (see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual). When enabled, a register parity error is injected upon accessing configuration registers. There is no poisoning capability for read-only registers. You must perform the following register accesses to trigger register parity poisoning:

- Write/read for dynamic registers
- Read for quasi-dynamic registers
- Read for static registers

For details about the different register types, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

Later, the poisoned address is observed as an error only if checking has occurred in the APB Clock domain on that register (For more information, see the “[Register Parity Checking on APB Interface Clock](#)” on page 395).

Therefore, a poisoned address needs two APB accesses, one to poison it, and then a subsequent APB Read to the same address, to guarantee triggering of register parity error.

Poisoning is removed when REGPARCFG.reg_par_poison_en is set to '0' and register parity is recalculated for all configuration registers to revert false parity error introduced with poisoning.

In the case of 4-bit parity (UMCTL2_REGPAR_TYPE=1) only the most significant parity bit is poisoned.



Note You must configure all the fields of the REGPARCFG register with a single write operation. It is also recommended to have at least 128 pclk cycled between two consecutive writes to the REGPARCFG.

13.4.6 Registers Related to REGPAR

The following are the registers related to REGPAR:

- REGPARCFG
- REGPARSTAT

For more information about these registers, refer to the "Register Descriptions" chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.5 On-Chip Command and Address Path Protection (OCCAP)

This topic contains the following sections:

- “Overview of On-Chip Command and Address Path Protection (OCCAP)” on page 398
- “Enabling On-Chip Command and Address Path Protection” on page 398
- “Protection Mechanisms” on page 398
- “Arbiter Protection” on page 401
- “DDRC Protection” on page 401
- “Poisoning” on page 403
- “Signals Related to OCCAP” on page 405
- “Registers Related to OCCAP” on page 405

13.5.1 Overview of On-Chip Command and Address Path Protection (OCCAP)

OCCAP supplements the OCPAR feature which is a similar protection mechanism for data path only.

This feature is available for you only with the following Automotive Product (AC) license:

- DWC-AC-LPDDR54-CONTROLLER

For automotive applications, reliability is important. The design needs to be robust against:

- Permanent faults (affecting both sequential and combinatorial logic)
- Transient faults (affecting sequential logic only)

The detection of single bit errors is important for both the fault types.

13.5.2 Enabling On-Chip Command and Address Path Protection

To enable On-Chip Command and Address Path Protection, set the hardware configuration parameter UMCTL2_OCCAP_EN to ‘1’.

13.5.3 Protection Mechanisms

The command and address path protection employs four mechanisms:

1. Parity protection: Parity is generated internally at the input of a block, propagated to the data, and checked at the output. This is used mainly for big registers, FIFOs, or memory arrays. An error is raised if the parity check at the output fails.
2. Duplication with comparison: The entire module is duplicated and outputs are checked every cycle, one by one, with XOR gates. An error is raised if any of the comparators fails.
3. Automotive FIFO Controllers. Enhanced versions of DesignWare BCMS of FIFO Controllers.
4. Triple Module Replication (TMR) of Registers.

The OCCAP feature is enabled by the OCCAPCFG.occup_en register (see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual).

13.5.3.1 Parity Protection

Parity calculation and check are based on a standard even parity with XOR gates.

For every 8 bits, data is XOR reduced to generate the parity bit. Input to the calculation/check module is always a multiple of 8 bits. The resulting errors from the different checkers are OR'ed to generate an interrupt and the corresponding fault signal.

The parity generators/checkers can be found throughout the hierarchy in various modules. The naming convention for these are:

- *OCCAP_en*U_pgen and *OCCAP_en*U_pcheck
- *PAR_check*U_pgen and *PAR_check*U_pcheck

If UMCTL2_OCCAP_PIPELINE=1 is set, additional pipelining stages are added:

- Outputs generated by pchecks throughout the design for error reporting.

This may help with timing issues if needed.

If UMCTL2_OCCAP_PIPELINE=1 is set, includes an additional pipeline for pcheck in rd_ie_rdata_ctl module when OCPAR is enabled with Inline ECC.

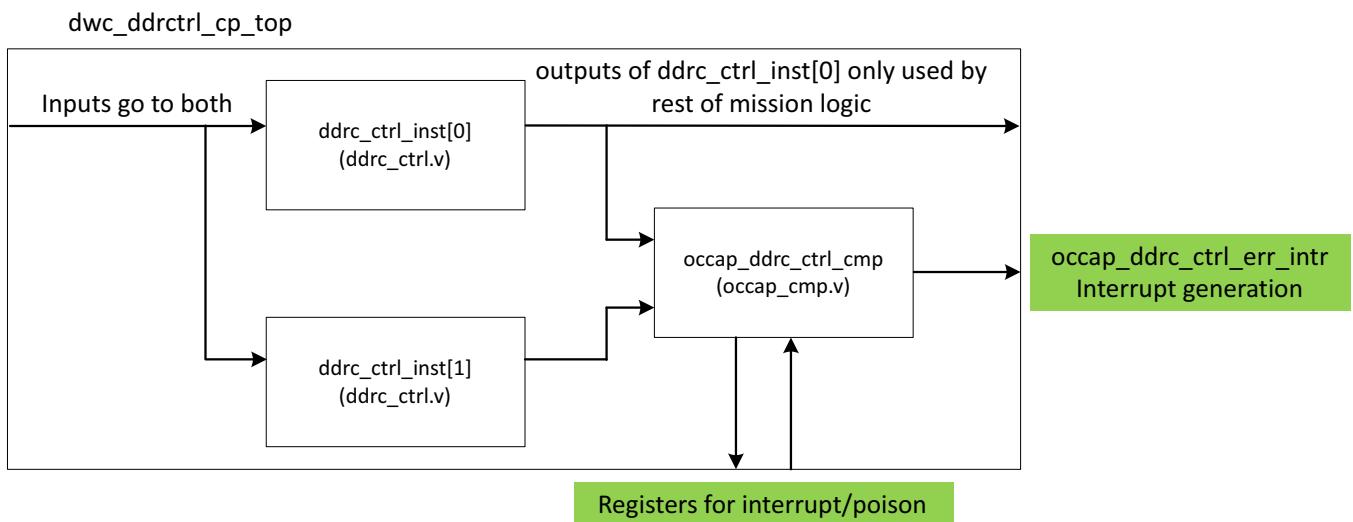
13.5.3.2 Duplication With Comparison

A replication of the module is instantiated together with the original block and a comparator, inside a *_wrapper module.

Same inputs are fed to both modules (original and replication), while both outputs are fed to the comparators. Standard comparators with XOR gates check every single output bit, and flag an error whenever there is a mismatch.

[Figure 13-28](#) shows an example for the DDRC control (dwc_ddrctrl_cp_top) block in the DDRC.

Figure 13-28 DDRC Control Wrapper



The duplicated modules are configuration dependent and are in hierarchy at:

- *U_rp/RP_inst[1:0] (for Arbiter)
- U_wp/WP_inst[1:0] (for Arbiter)
- U_ws/WS_inst[1:0] (for Arbiter)
- U_au/AU_inst[1:0] (for Arbiter)
- *ddrc_ctrl_wrapper/ddrc_ctrl_inst[1:0] (for DDRC Control)
- *mr_wrapper/mr_inst[1:0] (for DDRC Data)
- *rd_wrapper/rd_inst[1:0] (for DDRC Data)

Note that the comparator checking of ddrc_ctrl_inst[1:0]/mr_inst[1:0]/rd_inst[1:0] is disabled outside of reset when SWCTL.sw_done=0. This means that the other register settings are being changed.

For special requirements on these duplicated module, refer to the “OCCAP Synthesis Constraints chapter in the Synopsys LPDDR5X/5/4X Memory Controller User Guide.

If UMCTL2_OCCAP_PIPELINE=1 is set, additional pipelining stages are added:

- Inputs to occap_cmp from the duplicated modules.
- Outputs generated by occap_cmp for error reporting.

This may help with timing issues if needed.

13.5.3.3 Automotive FIFO Controllers

The FIFO controllers within the DDRCTL use standard products from the Synopsys library – see “Synchronizers Used in DWC_ddrctl” in Synopsys LPDDR5X/5/4X Memory Controller User Guide. These are in following modules:

- Synchronous (Dual Clock) FIFO Controller: DWC_ddrctl_bcm07
- Submodule of bcm07: DWC_ddrctl_bcm05
- Synchronous (Single Clock) FIFO: DWC_ddrctl_bcm65
- Synchronous (Single Clock) FIFO Controller (and submodule of bcm65): bcm06

When UMCTL2_OCCAP_EN=1, these are replaced with automotive versions of these existing BCM modules:

- DWC_ddrctl_bcm07 -> DWC_ddrctl_bcm07_atv
- DWC_ddrctl_bcm05 -> DWC_ddrctl_bcm05_atv
- DWC_ddrctl_bcm65 -> DWC_ddrctl_bcm65_atv
- DWC_ddrctl_bcm06 -> DWC_ddrctl_bcm06_atv

These Automotive BCMs use Triple Module Replication (TMR) techniques to be robust to transient errors in Automotive applications. In turn, some of these uses following sub-modules:

- bcm05_atv - Uses DWC_ddrctl_bcm95_i and DWC_ddrctl_bcm21_atv
- bcm06_atv - Uses DWC_ddrctl_bcm95_i

DWC_ddrctl_bcm21_atv is an automotive version of the standard DWC_ddrctl_bcm21, which is used for CDC. It is based on a TMR of the standard DWC_ddrctl_bcm21, with majority voting logic. When instantiated, each register inside the DWC_ddrctl_bcm21_atv has the name ‘sample_meta*’ or

'sample_sync'. It is only instantiated in the following instances of bcm05_atv inside bcm07_atv and can be identified by:

- *U_bcm07_atv/U_PUSH_FIFOCTL*
- *U_bcm07_atv/U_POP_FIFOCTL*

DWC_ddrctl_bcm95_i is a triple module replication of a registers, with majority voting logic between the three versions. When TMR occurs, each register has the name "dw_so_reg*".

13.5.3.4 Triple Module Replication (TMR) of Registers

The Synopsys library provides a method to replace a register with a triple module replication version of the same register, with majority voting logic between the three versions of the same register. This is provided by the DWC_ddrctl_bcm95_i module. An instance of DWC_ddrctl_bcm95_i module explicitly implements a small number of important registers, and these registers can be identified by the name "dw_so_reg*".

13.5.4 Arbiter Protection

The arbiter specifically refers to the AXI Port Interface (XPI) and the Port Arbiter (PA).

The arbiter protection mechanism uses:

- Parity
- Duplication with Comparison
- Automotive FIFO Controller
- TMR of registers

An error in any of the Arbiter blocks is flagged by:

- Interrupt signal occap_arb_err_intr, if enabled by OCCAPCFG.occap_arb_intr_en
- Fault signal occap_arb_err_intr_fault, regardless of OCCAPCFG.occap_arb_intr_en setting
- OCCAPSTAT.occap_arb_err_intr register field, regardless of OCCAPCFG.occap_arb_intr_en setting

For testing or debug purposes, you can force an Arbiter OCCAP interrupt by setting OCCAPCFG.occap_arb_intr_force to '1'. In this situation, the error is flagged by:

- Interrupt signal occap_arb_err_intr, if enabled by OCCAPCFG.occap_arb_intr_en
- OCCAPSTAT.occap_arb_err_intr register field, regardless of OCCAPCFG.occap_arb_intr_en setting

The interrupt is cleared by writing "1" to OCCAPCFG.occap_arb_intr_clr. For more information, see "REGB_DDRC_CH0 Registers" in the "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.5.5 DDRC Protection

This topic contains the following subsections:

- ["DDRC Control Protection" on page 402](#)

- “[DDRC Data Protection](#)” on page 402

13.5.5.1 DDRC Control Protection

The DDRC Control block (`ddrc_ctrl`) contains the address and control path of the DDRC. The DDRC Control protection uses ‘Duplication with Comparison’ mechanism.

An error in the DDRC Control block is flagged by:

- Interrupt signal `occup_ddrc_ctrl_err_intr`, if enabled by `OCCAPCFG1.occup_ddrc_ctrl_intr_en`
- Fault signal `occup_ddrc_ctrl_err_intr_fault`, regardless of `OCCAPCFG1.occup_ddrc_ctrl_intr_en` setting
- `OCCAPSTAT1.occup_ddrc_ctrl_err_intr` register field, regardless of `OCCAPCFG1.occup_ddrc_ctrl_intr_en` setting

For testing or debug purposes, you can force a DDRC Control OCCAP interrupt by setting `OCCAPCFG1.occup_ddrc_ctrl_intr_force` to ‘1’. In this situation, the error is flagged by:

- Interrupt signal `occup_ddrc_ctrl_err_intr`, if enabled by `OCCAPCFG1.occup_ddrc_ctrl_intr_en`
- `OCCAPSTAT1.occup_ddrc_ctrl_err_intr` register field, regardless of `OCCAPCFG1.occup_ddrc_ctrl_intr_en` setting

The interrupt is cleared by writing “1” to `OCCAPCFG1.occup_ddrc_ctrl_intr_clr`. For more information, see “[REGB_DDRC_CH0 Registers](#)” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.5.5.2 DDRC Data Protection

The DDRC Data refers to several modules within DDRC data path, for example, `memc_wu`, `mr`, `rd`, `rt`, and `dfi_data`. The control/address logic of the DDRC data related modules are protected.

The DDRC data protection uses:

- Parity
- Duplication with Comparison
- Automotive FIFO Controller

An error in the DDRC Data block is flagged by:

- Interrupt signal `occup_ddrc_data_err_intr`, if enabled by `OCCAPCFG1.occup_ddrc_data_intr_en`
- Fault signal `occup_ddrc_data_err_intr_fault`, regardless of `OCCAPCFG1.occup_ddrc_data_intr_en` setting
- `OCCAPSTAT1.occup_ddrc_data_err_intr` register field, regardless of `OCCAPCFG1.occup_ddrc_data_intr_en` setting

For testing or debug purposes, you can force a DDRC Data OCCAP interrupt by setting `OCCAPCFG1.occup_ddrc_data_intr_force` to ‘1’. In this situation, the error is flagged by:

- Interrupt signal `occup_ddrc_data_err_intr`, if enabled by `OCCAPCFG1.occup_ddrc_data_intr_en`
- `OCCAPSTAT1.occup_ddrc_data_err_intr` register field, regardless of `OCCAPCFG1.occup_ddrc_data_intr_en` setting

The interrupt is cleared by writing “1” to `OCCAPCFG1.occup_ddrc_data_intr_clr`. For more information, see “`REGB_DDRC_CH0 Registers`” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.5.6 Poisoning

The DDRCTL provides the ability to inject errors into the protection mechanisms. This can be used to test the feature and system response to OCCAP errors.

13.5.6.1 OCCAP Comparator Poisoning

All the OCCAP comparators that are used in the Duplication with Comparison mechanism can be poisoned by the software. The poisoning can be parallel (all bits of all outputs of the duplicate modules are inverted) or sequential (each bit is inverted one at a time). The sequential poisoning is performed in parallel for all involved comparators. The duration of the sequential poisoning depends on the size of the biggest comparator.

You cannot perform parallel poisoning and sequential poisoning at the same time.

Errors can be injected into the poisoning logic (either parallel or sequential) such that the XOR logic for one signal is not poisoned when expected. This is flagged by the corresponding `OCCAPSTAT*.occup_*_poison_parallel_err` or `OCCAPSTAT*.occup_*_poison_seq_err` being set.

The Arbiter comparators can be tested through the following registers:

- `OCCAPCFG.occup_arb_cmp_poison_seq`
- `OCCAPCFG.occup_arb_cmp_poison_parallel`
- `OCCAPCFG.occup_arb_cmp_poison_err_inj`
- `OCCAPSTAT.occup_arb_cmp_poison_seq_err`
- `OCCAPSTAT.occup_arb_cmp_poison_parallel_err`

The DDRC Control comparators can be tested through the following registers:

- `OCCAPCFG1.occup_ddrc_ctrl1_poison_seq`
- `OCCAPCFG1.occup_ddrc_ctrl1_poison_parallel`
- `OCCAPCFG1.occup_ddrc_ctrl1_poison_err_inj`
- `OCCAPSTAT1.occup_ddrc_ctrl1_poison_seq_err`
- `OCCAPSTAT1.occup_ddrc_ctrl1_poison_parallel_err`

The DDRC Data comparators can be tested through the following registers:

- `OCCAPCFG1.occup_ddrc_data_poison_seq`
- `OCCAPCFG1.occup_ddrc_data_poison_parallel`
- `OCCAPCFG1.occup_ddrc_data_poison_err_inj`

- OCCAPSTAT1.occap_ddrc_data_poison_seq_err
- OCCAPSTAT1.occap_ddrc_data_poison_parallel_err

For more information, see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.



Note This sequence should only be performed if OCCAPCFG.occap_en=1.

[Table 13-10](#) shows the software poisoning sequence example for DDRC control.

Table 13-10 Software Poisoning Sequence Example for DDRC Control

Step	Description	Comment
1	Set OCCAPCFG1.occap_ddrc_ctrl_poison_err_inj=1. This field must not be set in the same APB access as Step 2.	Enables error injecting into the poisoning procedure. This step is optional.
2	Set OCCAPCFG1.occap_ddrc_ctrl_poison_seq=1 Or OCCAPCFG1.occap_ddrc_ctrl_poison_parallel=1 Do not set both at the same time.	Enables selected poisoning procedure, sequential or parallel.
3	Poll for OCCAPSTAT1.occap_ddrc_ctrl_poison_complete=1	Checks for the completion of the poisoning procedure.
4	Depending on Step 2, check corresponding: OCCAPSTAT1.occap_ddrc_ctrl_poison_seq_err Or OCCAPSTAT1.occap_ddrc_ctrl_poison_parallel_err If OCCAPCFG1.occap_ddrc_ctrl_poison_err_inj=1, the error status must be 1, otherwise it must be 0.	Checks if the corresponding poisoning operated as expected.
5	Set OCCAPCFG1.occap_ddrc_ctrl_intr_clr	Clears the following registers: OCCAPSTAT1.occap_ddrc_ctrl_err_intr and OCCAPSTAT1.occap_ddrc_ctrl_poison_seq_err Or OCCAPSTAT1.occap_ddrc_ctrl_poison_parallel_err
6	Poll for OCCAPSTAT1.occap_ddrc_ctrl_poison_complete=0	Makes sure poisoning has terminated.

13.5.6.2 RAQ Poisoning

Due to the complexity of the design and the unpredictability of behavior and latencies, parity poisoning is possible only for the Read Address Queues (RAQ) in each port’s XPI block.

The number of RAQs in each XPI is configuration dependent: UMCTL2_XPI_USE2RAQ_n. Either 1 or 2 RAQ exist and poisoning occurs for all RAQs within a port.

When poisoning is enabled, every time the RAQ is written, all parity bits are flipped, basically an odd parity is calculated instead. An error is then flagged every time the RAQ is read.

RAQ poisoning is enabled through the register OCCAPCFG.occap_arb_raq_poison_en. For more information, see “REGB_DDRC_CH0 Registers” in the “Register Descriptions” chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.5.7 Signals Related to OCCAP

For information on signals related to OCCAP, refer to “On-Chip Command/Address Path Protection Signals” section in the Signal Descriptions chapter.

13.5.8 Registers Related to OCCAP

The following are the registers related to the OCCAP:

- OCCAPCFG
- OCCAPSTAT
- OCCAPCFG1
- OCCAPSTAT1

For more information on these registers, refer to the “Register Descriptions” chapter in the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.6 On-Chip External SRAM Address Protection (OCSAP)

This topic contains the following sections:

- “Overview of On-Chip External SRAM Address Protection” on page 406
- “Enabling On-Chip External SRAM Address Protection (OCSAP)” on page 406
- “On-Chip External SRAM Address Parity” on page 406
- “On-Chip External SRAM Address Parity Poisoning” on page 406
- “Signals Related to On-Chip External SRAM Address Protection” on page 407
- “Registers Related to On-Chip External SRAM Address Protection” on page 407

13.6.1 Overview of On-Chip External SRAM Address Protection



Note This feature is available only with the DWC-AC-LPDDR54-CONTROLLER Automotive Product (AC) license.

On-Chip external SRAM Address Protection (OCSAP) works in conjunction with OCPAR or OCECC support with Inline ECC. This is primarily to provide address protection for the external SRAM while end-to-end data path protection is achieved either using OCPAR or OCECC.

SRAM address has 2-bit even parity which is embedded within the data parity/ECC before writing to SRAM. The actual data parity/ECC is retrieved during the read operation of SRAM. The SRAM address parity is checked and terminated; only the retrieved data parity/ECC is further propagated along with the data. It uses the existing error reporting mechanism as followed in OCPAR or OCECC configuration. See [Figure 13-26](#) on page 387 where OCSAP related blocks are highlighted in green.

13.6.2 Enabling On-Chip External SRAM Address Protection (OCSAP)

To enable On-Chip External SRAM Address Protection, set the hardware configuration parameter DDRCTL_OCSAP_EN to ‘1’.

This feature can then be enabled/disabled by software using the OCSAPCFG0 . ocsap_par_en register.

13.6.3 On-Chip External SRAM Address Parity

For every write or read to external SRAM, 2-bit SRAM address even parity is generated. This 2-bit address parity is unique for four consecutive SRAM address locations and it is calculated as follows:

$$\begin{aligned}\text{Parity [0]} &= A_0 \wedge A_2 \wedge A_4 \wedge A_6 \\ \text{Parity [1]} &= A_1 \wedge A_3 \wedge A_5 \wedge A_7\end{aligned}$$

SRAM address parity, which is 2-bits, is combined with either the data path parity (in case of OCPAR) or the data path ECC (in case of OCECC) by XOR operation.

13.6.4 On-Chip External SRAM Address Parity Poisoning

The DDRCTL provides the ability to inject SRAM address parity errors. This poisoning can be used to test the feature and system’s responses to incorrect SRAM addresses. This functionality is enabled by setting the register OCSAPCFG0 . ocsap_poison_en.

The following SRAM address parity can be poisoned:

- WDATARAM address - based on OCSAPCFG0.wdataram_addr_poison_ctl, one of the WDATARAM address bits is corrupted during the computation of WDATARAM address parity. This eventually injects one of the WRDATARAM address parity errors. Based on OCSAPCFG0.wdataram_addr_poison_loc, parity is poisoned in either WDATARAM read address or write address.
- RDATARAM address - based on OCSAPCFG0.rdataram_addr_poison_ctl, one of the RDATARAM address bits is corrupted during the computation of RDATARAM address parity. This eventually injects one of the RRDATARAM address parity errors. Based on OCSAPCFG0.rdataram_addr_poison_loc, parity is poisoned in either RDATARAM read address or write address. Only one port at a time can be poisoned based on OCSAPCFG0.rdataram_addr_poison_port.

One-shot triggering is supported, which means that error is injected for the first valid address either after OCSAPCFG0.ocsap_poison_en is set to '1' or after the interrupt clear for that path is asserted. This means that only the parity corresponding to the first valid access of the transaction on SRAM is poisoned.

AXI traffic restrictions for on-chip external SRAM address parity poisoning:

- All the AXI traffic restrictions apply for on-chip external SRAM address parity poisoning with OCECC enable.
- All the AXI traffic restrictions as defined in section “[Parity Poisoning](#)” on page 390 apply for on-chip external SRAM address parity poisoning with OCPAR enable.



Note Only one (OCSAPCFG0.ocsap_poison_en, OCPARCFG1.par_poison_en, or OCECCCGF1.ocecc_poison_en) is effective at one time.

13.6.5 Signals Related to On-Chip External SRAM Address Protection

OCSAP works in conjunction with either OCPAR or OCECC support. It used the existing error reporting mechanism as followed in OCPAR or OCECC configuration.

13.6.6 Registers Related to On-Chip External SRAM Address Protection

OCSAP works in conjunction with either OCPAR or OCECC support. In addition to OCPAR/OCECC related registers, the following is the only register related to the On-Chip external SRAM Address Protection:

- OCSAPCFG0

For more information on these registers, see the "Register Descriptions" chapter of the Synopsys LPDDR5X/5/4X Memory Controller Reference Manual.

13.7 DFI Sideband Watchdog Timers

13.7.1 Overview of DFI Sideband Watchdog Timers

There are watchdog timers present in the design for monitoring the DFI Sideband handshake completion. If any controller-initiated DFI Sideband handshake does not complete within the expected duration, then the corresponding error status registers are set, and interrupt is flagged.

The purpose of these watchdog timers is to detect and report any lock-up conditions in case of misbehavior of the associated logic inside Controller or PHY. The misbehavior could happen due to permanent faults (affecting both sequential and combinational logic) or transient faults (affecting sequential logic only).

Watchdog timers are present for the following controller-initiated DFI Sideband requests:

- Controller-initiated DFI Update request
- DFI Status interface
- DFI Low-power interface

The following two DFI sideband requests initiated by PHY are not monitored by the Controller.

- PHY-initiated DFI Update request
- DFI PHY Master Interface (PMI) request

Synopsys PHY has watchdog timers to monitor these two requests. Any error found on these two interfaces are notified to the Controller via the DFI Error Interface (see the section “[DFI Error Interface](#)” on page 108).

Software controlled poisoning feature is available to test the DFI Sideband watchdog timers before the start of normal operation.



DFI Sideband Watchdog timer feature is supported in limited configurations. For more information, contact Synopsys.

13.7.2 Hardware Parameter Related to DFI Sideband Watchdog Timers

To enable DFI Sideband Watchdog timer feature, set the hardware configuration parameter DDRCTL_DFI_SB_WDT to '1'.

13.7.3 Registers Related to DFI Sideband Watchdog Timers

The following registers are related to the DFI Sideband watchdog timer feature:

- DFISBINTRPTCFG
- DFISBPOISONCFG
- DFISBTIMERSTAT
- DFISBTIMERSTAT1

13.7.4 Interrupts Related to DFI Sideband Watchdog Timers

The following interrupts are associated with the DFI Sideband Watchdog timer feature:

- dfi_sideband_timer_err_intr
- dfi_sideband_timer_err_intr_fault

13.7.5 Controller-initiated DFI Update Watchdog Timers

The operational details of the DFI Update Watchdog timers are given in the following table.

Monitored DFI Interface Signal	Associated Register	Register Description	Watchdog Timer Behavior	Meaning of the Error
dfi_ctrlupd_req	DFIUPDTMG0.dfi_t_ctrlup_min	Specifies the minimum number of DFI clock cycles that the d _i dfi_ctrlupd_req must be asserted.	Timer starts down-counting from the programmed register value when d _i dfi_ctrlupd_req is asserted HIGH. Flags error if d _i dfi_ctrlupd_req is detected LOW when the timer is non-zero.	Controller did not keep d _i dfi_ctrlupd_req asserted for the minimum required duration.
dfi_ctrlupd_req dfi_ctrlupd_ack	DFIUPDTMG0.dfi_t_ctrlup_max	Specifies the maximum number of DFI clock cycles that the d _i dfi_ctrlupd_req can assert.	Timer starts down-counting from the programmed register value when d _i dfi_ctrlupd_req is asserted HIGH. Flags error if d _i dfi_ctrlupd_req is detected HIGH when the timer is zero.	Controller did not de-assert d _i dfi_ctrlupd_req even after the max period has expired. This could happen due to error in Controller or because PHY didn't de-assert d _i dfi_ctrlupd_ack signal within the required time period.

13.7.6 DFI Status Interface Watchdog Timers

The operational details of the DFI Status Interface Watchdog timers are given in the following table.

Monitored DFI Interface Signal	Associated Register	Register Description	Watchdog Timer Behavior	Meaning of the Error
dfi_init_start	DFITMG7.dfi_t_init_start	Specifies the minimum time for which d _i dfi_init_start should stay HIGH once asserted.	Timer starts down-counting from the programmed register value when d _i dfi_init_start is asserted HIGH. Flags error if d _i dfi_init_start is detected LOW when the timer is non-zero.	Controller did not keep d _i dfi_init_start asserted for the minimum required duration.

Monitored DFI Interface Signal	Associated Register	Register Description	Watchdog Timer Behavior	Meaning of the Error
dfi_init_complete	DFITMG7.dfi_t_init_complete	During a frequency change operation, specifies the maximum number of DFI clock cycles after the de-assertion of the dfi_init_start signal to the reassertion of the dfi_init_complete signal.	Timer starts down-counting from the programmed register value when dfi_init_start moves from HIGH to LOW. Flags error if dfi_init_complete is detected LOW when the timer is zero.	PHY did not re-assert dfi_init_complete signal within the required time period.

13.7.7 DFI Low Power Interface Watchdog Timers

The operational details of the DFI Low Power Interface Watchdog timers are given in the following table.

Monitored DFI Interface Signal	Associated Register	Register Description	Watchdog Timer Behavior	Meaning of the Error
dfi_lp_ctrl_req dfi_lp_data_req	DFILPTMG1.dfi_tlp_resp	Specifies the maximum number of DFI clock cycles after the assertion of the dfi_lp_ctrl_req/dfi_lp_data_req signal to the assertion of the associated dfi_lp_ctrl_ack/dfi_lp_data_ack signal.	Timer starts down-counting from the programmed register value when dfi_lp_ctrl_req/dfi_lp_data_req is detected HIGH. Flags error if dfi_lp_ctrl_req/dfi_lp_data_req is detected LOW when the timer is non-zero.	Controller did not keep dfi_lp_ctrl_req/dfi_lp_data_req asserted for the minimum required duration.
dfi_lp_ctrl_wakeup	DFILPTMG0.dfi_lp_wakeup_dsm DFILPTMG0.dfi_lp_wakeup_sr DFILPTMG0.dfi_lp_wakeup_pd DFILPTMG0.dfi_lp_wakeup_data	Indicates the value in DFI clock cycles to drive on dfi_lp_ctrl_wakeup signal when different powerdown states are entered: deep sleep mode, self-refresh, power-down, data bus idle.	Timer starts down-counting from the programmed register value when dfi_lp_ctrl_req goes from HIGH to LOW and dfi_lp_ctrl_ack is HIGH. Flags error if dfi_lp_ctrl_ack is detected HIGH when the timer is zero.	PHY did not de-assert dfi_lp_ctrl_ack within the required time period.

Monitored DFI Interface Signal	Associated Register	Register Description	Watchdog Timer Behavior	Meaning of the Error
dfi_lp_data_wakeup	DFILPTMG0.dfi_lp_wakeup_dsm DFILPTMG0.dfi_lp_wakeup_sr DFILPTMG0.dfi_lp_wakeup_pd DFILPTMG0.dfi_lp_wakeup_data	Indicates the value in DFI clock cycles to drive on dfi_lp_data_wakeup signal when different powerdown states are entered: deep sleep mode, self-refresh, power-down, data bus idle.	Timer starts down-counting from the programmed register value when dfi_lp_data_req goes from HIGH to LOW and dfi_lp_data_ack is HIGH. Flags error if dfi_lp_data_ack is detected HIGH when the timer is zero.	PHY did not de-assert dfi_lp_data_ack within the required time period.

13.7.8 Poison Logic

Error injection poison logic is provided for testing the functionality of the DFI Sideband Watchdog timers. Each timer has its own APB register bit that enables the poisoning functionality. Refer to DFISBPOISONCFG register for the register programming details.

The watchdog timers check if the DFI sideband signals assert or de-assert within the required time intervals specified in the DFI specification. When the poison feature is enabled, the internally used DFI signals are forced to misbehave, and this causes the watchdog timer logic to flag errors.

14

Programming the Controller

This chapter includes the following sections:

- “Initialization” on page [414](#)
- “Registers” on page [418](#)
- “Modes of Operation” on page [423](#)
- “Software Sequences” on page [424](#)



Note Only documented sequences are supported.

14.1 Initialization

This section describes the programming sequence that must be executed at power-up to bring the controller, the PHY, and the memories into a state where HIF reads and writes can be performed.

This section includes the following subsections:

- “[Register Groups](#)” on page [414](#)
- “[Controller Initialization](#)” on page [415](#)
- “[PHY Initialization](#)” on page [415](#)
- “[SDRAM Initialization](#)” on page [415](#)
- “[LPDDR5X/5/4X Initialization Summary Tables](#)” on page [416](#)

14.1.1 Register Groups

See “Clock And Reset Requirements” in the Synopsys LPDDR5X/5/4X Memory Controller Databook for a summary of clocking in DDRCTL.

Some APB registers can be changed at any time. They are defined as dynamic registers. For more information about this, see “[Dynamic Registers](#)” on page [418](#). Other APB registers must not be changed after the `core_ddrc_rstn` is de-asserted. “[Group 1: Registers that can be Written when No Read/Write Traffic is Present at the DFI](#)”, “[Group 2: Registers that can be Written in Self-refresh and MPSM Modes](#)”, “[Refresh Related Registers](#)” are exceptions to this. These are indicated as Static registers.

The APB registers are reset by the signal `presetn`, which is de-asserted synchronously with `pclk`.

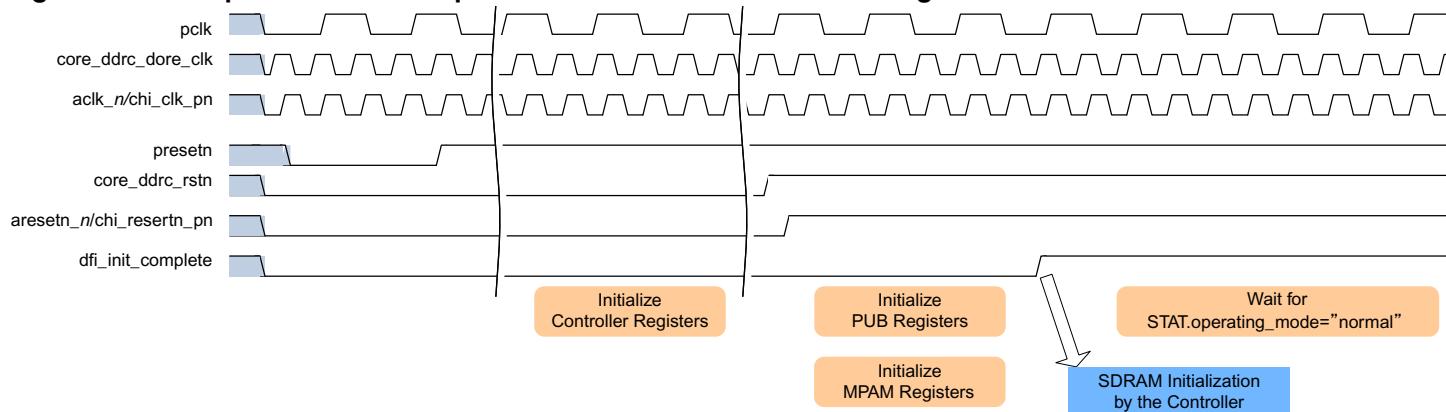
Static registers are sampled directly in the destination domains (`core_ddrc_core_clk/aclk` domains) without synchronizing circuits. The de-assertion of the signal `presetn` is synchronized to the destination domain clocks and used to de-assert the reset on the static registers in their destination domains. Therefore, the static registers must be programmed when `core_ddrc_core_clk/aclk_n` and `pclk` are stable, `presetn` is de-asserted (at least for 4 cycles of clock in the destination domain) and `core_ddrc_rstn/aresetn_n` are asserted.

Synchronizing circuits are used to transfer the signals for dynamic registers to their destination domains. While these registers can be initialized with `core_ddrc_rstn` asserted, they can be changed thereafter with `core_ddrc_rstn` de-asserted.

14.1.2 Controller Initialization

Figure 14-1 shows the relationship between clocks and resets for initialization of registers. Power and clocks must be in accordance with the requirements of full ASIC/SoC design requirements, and the SDRAM power and clocks must in accordance with the SDRAM specifications.

Figure 14-1 Simplified Relationship Between Clocks and Resets for Register Initialization



After power-up:

1. Assert the resets `core_ddrc_rstn`, `ras_log_rstn`, `sbr_resetn`, and `aresetn/chi_reseretn_pn`
2. Assert `presetn`
3. Start the clocks (`pclk`, `core_ddrc_core_clk`, `sbr_clk`, and `aclk_n/chi_clk_pn`)
4. De-assert `presetn` once the clocks are active and stable
5. Allow 128 APB clock cycles for synchronization of `presetn` to `core_ddrc_core_clk` and `aclk/chi_clk_pn` domains and to permit initialization of end logic
6. Initialize the registers
7. De-assert the resets `core_ddrc_rstn`, `ras_log_rstn`, `sbr_resetn`, and `aresetn/chi_reseretn_pn`

Any subsequent access to the programming interface of the registers which may be accessed outside of reset must happen when both `pclk` and `core_ddrc_core_clk` are active and stable (see the “Register Descriptions” chapter).

14.1.3 PHY Initialization

Before the memory can be accessed by the DDRCTL controller, the PHY must be initialized.

When using the Synopsys LPDDR5X/5/4X PHY, the PHY Utility Block (PUB) can be used for initialization. The PUB and details on the PHY initialization using the PUB are documented in the Synopsys LPDDR5X/5/4X PHY Utility Block (PUB) Databook. For an outline of the full initialization sequence for a Synopsys LPDDR5X/5/4X, refer to [Table 14-1](#) on page [416](#).

14.1.4 SDRAM Initialization

Synopsys PHY initializes the SDRAM as part of PHY initialization procedure.

14.1.5 LPDDR5X/5/4X Initialization Summary Tables

The following tables summarize the initialization procedure when using the Synopsys LPDDR5X/5/4X PHY ([Table 14-1](#)). For more information about this, see the Synopsys LPDDR5X/5/4X PHY Utility Block (PUB) Databook.



Note The following registers are quasi-dynamic registers and have to be programmed by using a special sequence described in “[Quasi Dynamic Registers](#)” on page [419](#):

- DFIMISC.dfi_init_start
- DFIMISC.dfi_reset_n
- DFIMISC.dfi_init_complete_en

For example, if you need to update DFIMISC.dfi_init_start, you need to follow “[Group 3: Registers that can be Written When Controller is Empty](#)” on page [421](#).

Table 14-1 DDRCTL and Memory Initialization with LPDDR5X/5/4X PHY

Step	Application	SVTB Task	Notes
1	Follow the PHYs power up procedure		See PUB databook for details
2	Program the DWC_ddrctl registers		Note 1
3	Disable auto-refreshes, self-refresh, powerdown and assertion of dfi_dram_clk_disable by setting RFSHCTL0.dis_auto_refresh = 1, PWRCTL.powerdown_en = 0 and PWRCTL.selfref_en = 0 PWRTL.en_dfi_dram_clk_disable = 0	reset_dut	
4	De-assert reset signal core_ddrc_rstn		
5	Set DFIMISC.dfi_init_complete_en to '0' Set DFIMISC.dfi_reset_n to '1'	phy_init	
6	Start PHY initialization and training by accessing relevant PUB registers	phy_init	See PUB databook for details
7	Poll the PUB register APBONLY.UctShadowRegs[0] =1'b0	phy_init	See PUB databook for details
8	Read the PUB Register APBONLY.UctWriteOnlyShadow for training status	phy_init	See PUB databook for details
9	Write the PUB Register APBONLY.DctWriteProt = 0	phy_init	See PUB databook for details
10	Poll the PUB register APBONLY.UctShadowRegs[0] =1'b1	phy_init	See PUB databook for details

Step	Application	SVTB Task	Notes
11	Write the PUB Register APBONLY.DctWriteProt = 1	phy_init	See PUB databook for details
12	Poll the PUB register MASTER.CalBusy=0	phy_init	See PUB databook for details
13	Set DFIMISC.dfi_init_start to '1'	phy_init	
14	Poll DFISTAT.dfi_init_complete=1	phy_init	
15	Set DFIMISC.dfi_init_start to '0'	phy_init	
16	The following registers may need to be updated after training has completed: <ul style="list-style-type: none"> ■ RANKTMG0.diff_rank_wr_gap ■ RANKTMG0.diff_rank_rd_gap ■ DRAMSET1TMG2.rd2wr ■ DRAMSET1TMG2.wr2rd ■ DRAMSET1TMG24.rd2wr_s ■ DRAMSET1TMG9.wr2rd_s ■ DFITMG0.dfi_t_ctrl_delay ■ DFITMG1.dfi_t_wrdata_delay ■ DFITMG2.dfi_twck_delay (LPDDR5 only) 	phy_init	Refer to relevant PHY documentation
17	Set DFIMISC.dfi_init_complete_en to '1'	phy_init	
18	Set PWRCTL.selfref_sw to '0'	phy_init	
19	Wait for DWC_ddrctl to move to normal operating mode by monitoring STAT.operating_mode signal	reset_dut	
20	Set back registers in step 3 to the original values if desired		



Note When running training with the PHY. The following controller registers must be programmed as follows at this stage:

- INITTMG0.skip_dram_init=2'b11
- PWRCTL.selfref_sw= 'b1

Programming them as follows is only allowed for simulation purposes when skipping training.

- For LPDDR4 only:
 - INITTMG0.skip_dram_init=0 (that is, SDRAM INIT through the controller).
 - PWRCTL.selfref_sw=0

14.2 Registers

Registers are accessed through the APB interface. The full list of registers is provided in the "Register Descriptions" chapter of the LPDDR5X/5/4X Memory Controller Reference Manual.

The following sections outline under what circumstances the APB registers can be written. Most registers are intended to be initialized while the DDRCTL controller is in reset (`core_ddrc_rstn = 0`), and should not need to be changed afterwards. The exceptions to this rule are listed in the following sections.

`core_ddrc_core_clk` must be brought up and running before DDRCTL controller is brought out of reset (`core_ddrc_rstn` is de-asserted).

The memory map is a hierarchical structure consisting of different blocks. Certain registers may be instantiated within several modules. The registers are uniquely identified by the block and register name. For example the register STAT exists within both blocks UMCTL2_REGS and UMCTL2_REGS_DCH1 and the individual instances are uniquely identified as `UMCTL2_REGS . STAT` and `UMCTL2_REGS_DCH1 . STAT`. The information in the following sections applies to all instances of any given register.

The "Programming Mode" field in the description column of the register tables specifies the type of register, whether static, dynamic, or quasi dynamic. See the "Register Descriptions" chapter of the LPDDR5X/5/4X Memory Controller Reference Manual.

This section contains the following sub-sections:

- ["Dynamic Registers" on page 418](#)
- ["Quasi Dynamic Registers" on page 419](#)
- ["Interrupt Signals and Corresponding Interrupt Status Registers" on page 422](#)

14.2.1 Dynamic Registers

The dynamic registers can be written at any time during the operation of the DDRCTL. As the APB block is typically in a different clock domain from the DDRCTL controller, these signals are synchronized to the DDRCTL controller clock domain. For the remaining registers which are considered static, synchronization is not required. To know if a particular register is of the type dynamic, see the "Programming Mode" field in the description column of the register tables in the "Register Descriptions" chapter of the LPDDR5X/5/4X Memory Controller Reference Manual.

14.2.1.1 Refresh Related Registers

The refresh related registers are dynamic but to update them the following must be done:

- Change the refresh associated register as desired.
- After the changed register is known stable, toggle the `RFSHCTL0.refresh_update_level` signal.

The SDRAM controller notes the `refresh_update_level` signal change and updates all refresh-related register values accordingly. This mechanism is needed to avoid sampling errors in the target clock domain as well as to allow the controller to provide special handling (such as issuing an additional refresh and resetting the refresh timer if needed) when a refresh-related timing register has changed.

Refresh related registers that can be updated are denoted as "Dynamic - Refresh Related" in the "Programming Mode" field in the "Register Descriptions" chapter of the LPDDR5X/5/4X Memory Controller Reference Manual.

Most of the refresh related registers are dynamic.

However, `RFSHSET1TMG4.refresh_timer*_start_value_x32` can only be programmed during initialization. For details, see section “Constraints on refresh_timerX_start_value_x32” section.

14.2.2 Quasi Dynamic Registers

In addition to the dynamic registers, the following categories of registers can be written after reset:

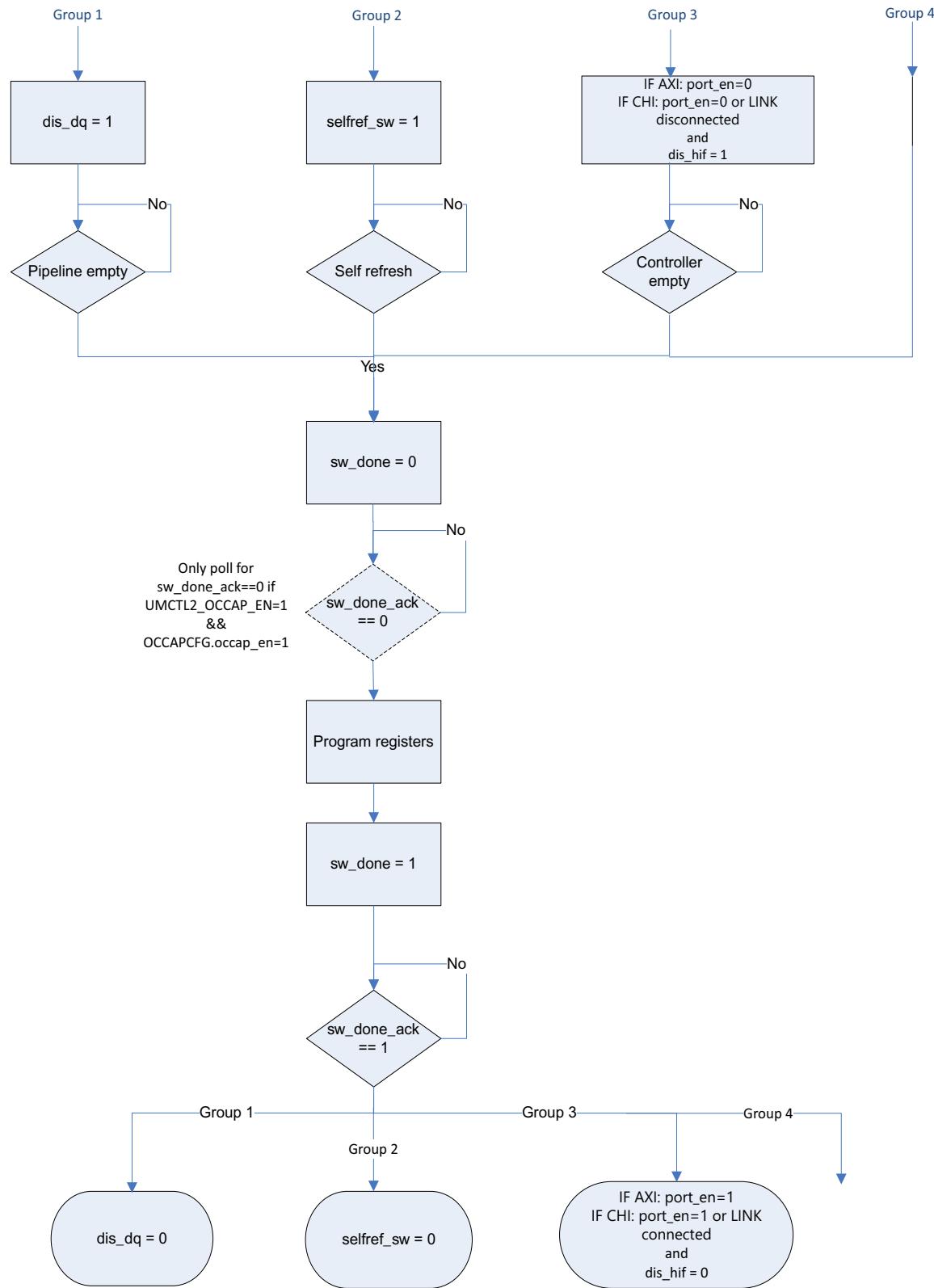
- “[Group 1: Registers that can be Written when No Read/Write Traffic is Present at the DFI](#)”
- “[Group 2: Registers that can be Written in Self-refresh and MPSM Modes](#)”
- “[Group 3: Registers that can be Written When Controller is Empty](#)”
- “[Group 4: Registers that can be Written Depending on MSTR2.target_frequency](#)”

Each category requires specific conditions for the registers to be programmed. Once the programming conditions are met, `SWCTL.sw_done` register must be programmed to 1'b0 to enable the software programming. If `UMCTL2_OCCAP_EN=1 && OCCAPCFG.occap_en=1`, the `SWSTAT.sw_done_ack` must be read as 1'b0.

Once the programming is completed, `SWCTL.sw_done` must be set to 1'b1 and `SWSTAT.sw_done_ack` must be read as 1'b1. This is to ensure that quasi dynamic registers are propagated correctly to the destination clocks.

The protocol flowchart is shown in [Figure 14-2](#).

Figure 14-2 Software Handshake Protocol



The "Programming Mode" field in the description column of the register tables in the "DWC_ddrctl Registers" chapter of the databook specifies the quasi dynamic type registers along with the group.

Traffic has to be enabled again depending on the register category as described in the following sections.

14.2.2.1 Group 1: Registers that can be Written when No Read/Write Traffic is Present at the DFI

By setting OPCTRL1.dis_dq register and polling OPCTRLCAM.wr_data_pipeline_empty and OPCTRLCAM.rd_data_pipeline_empty¹, it is possible to prevent any read or write traffic from being sent on the DFI. In this mode, it is safe to write to the group 1 registers.

Re-enable the traffic by writing OPCTRL1.dis_dq to 1'b0.

14.2.2.2 Group 2: Registers that can be Written in Self-refresh and MPSM Modes

When the DDRCTL has entered self-refresh mode through software (PWRCTL.selfref_sw), the DFI bus is idle until software exits self-refresh.



- For self-refresh, ensure that self-refresh is not caused by "Automatic Self-refresh only" by checking the STAT.operating_mode=3'b011 and STAT.selfref_type=2'b10.

In this section, references to self-refresh mean SR-Powerdown.

In this mode, it is safe to write the group 2 registers.

Re-enable the traffic by writing PWRCTL.selfref_sw to 1'b0 and polling STAT.operating_mode.

14.2.2.3 Group 3: Registers that can be Written When Controller is Empty

For multi-port AXI configurations, PCTRL.port_en is used to enable or disable the input traffic per port.

The Controller idleness can be polled first from PSTAT register (wr_port_busy_n and rd_port_busy_n bit fields) and should read as PSTAT==32'b0 (not busy). Write 0 to SBRCTL.scrub_en to disable SBR (required only if SBR instantiated). Poll SBRSTAT.scrub_busy=0, indicates that there are no outstanding SBR read commands (required only if SBR instantiated). OPCTRL1.dis_hif is used to enable or disable the input traffic to the controller. Then, DDRC CAM/pipeline empty status must be polled

((OPCTRLCAM.dbg_wr_q_empty== 1'b1) && (OPCTRLCAM.dbg_rd_q_empty== 1'b1) && (OPCTRLCAM.wr_data_pipeline_empty== 1'b1) && (OPCTRLCAM.rd_data_pipeline_empty== 1'b1)).

In this mode, it is safe to write the group 3 registers.

Enable the traffic by writing 1'b1 to PCTRL.port_en or to OPCTRL1.dis_hif if using HIF only. Enable SBR if desired by writing 1'b1 to SBRCTL.scrub_en (only required if SBR instantiated).

1. To make sure the correct value is propagated, registers OPCTRLCAM.wr_data_pipeline_empty and OPCTRLCAM.rd_data_pipeline_empty must be polled at least twice after OPCTRL1.dis_dq is set to '1'.

14.2.2.4 Group 4: Registers that can be Written Depending on MSTR2.target_frequency

When UMCTL2_FREQUENCY_NUM is larger than 1, it is safe to write the registers in the blocks REGB_FREQf_CHc:

- the value of f is between 0 and (UMCTL2_FREQUENCY_NUM - 1) excluding the register block related to the current target frequency
- the value of c is between 0 and the number of channel minus 1

For example in dual channel configurations, if UMCTL2_FREQUENCY_NUM is set to 3 and MSTR2.target_frequency is equal to '1', it is safe to write registers in REGB_FREQ0_CH0, REGB_FREQ0_CH1, REGB_FREQ2_CH0 and REGB_FREQ2_CH1.

14.2.3 Interrupt Signals and Corresponding Interrupt Status Registers

Interrupt signals and corresponding interrupt status registers are listed in [Table 14-2](#).

Table 14-2 Interrupt Signals and Corresponding Interrupt Status Registers

Interrupt Signal	Interrupt Status Register
derate_temp_limit_intr	DERATESTAT0.derate_temp_limit_intr
ecc_ap_err_intr	ECCAPSTAT.ecc_ap_err
ecc_corrected_err_intr	ECCSTAT.ecc_corrected_err
ecc_uncorrected_err_intr	ECCSTAT.ecc_uncorrected_err
rd_linkecc_corr_err_intr	LNKECCERRSTAT.rd_link_ecc_corr_err_int
rd_linkecc_uncorr_err_intr	LNKECCERRSTAT.rd_link_ecc_uncorr_err_int
sbr_done_intr	SBRSTAT.scrub_done
par_waddr_err_intr	OCPARSTAT0.par_waddr_err_intr_<0-15>
par_raddr_err_intr	OCPARSTAT0.par_raddr_err_intr_<0-15>
par_wdata_err_intr	OCPARSTAT1.par_wdata_err_intr_<0-15>
par_rdata_err_intr	OCPARSTAT1.par_rdata_err_intr_<0-15>
reg_par_err_intr	REGPARSTAT.reg_par_err_intr
occup_arb_err_intr	OCCAPSTAT.occup_arb_err_intr
occup_ddrc_ctrl_err_intr	OCCAPSTAT1.occup_ddrc_ctrl_err_intr
occup_ddrc_data_err_intr	OCCAPSTAT1.occup_ddrc_data_err_intr
occup_dfiic_err_intr	OCCAPSTAT2.occup_dfiic_err_intr
dfi_sideband_timer_err_intr	DFISBTIMERSTAT1.dfi_sideband_timer_err_intr
dfi_error_intr	DFIERRORSTAT.dfi_error_intr

14.3 Modes of Operation

This section describes following topics:

- “[SDRAM Selection](#)” on page [423](#)
- “[Low-Power Modes](#)” on page [423](#)

14.3.1 SDRAM Selection

SDRAM type is selected by setting the MSTR0 register fields as indicated in [Table 14-3](#).

Table 14-3 SDRAM Selection

MSTR0.lpddr4	MSTR0.lpddr5	MSTR0.lpddr5x	Selected DRAM
1	0	0	LPDDR4
0	1	0	LPDDR5
0	1	1	LPDDR5X
All other values are not supported.			

MSTR0.lpddr4 – Present only in designs that support LPDDR4

MSTR0.lpddr5 – Present only in designs that support LPDDR5

MSTR0.lpddr5x - Present only in designs that support LPDDR5X

14.3.2 Low-Power Modes

The operating mode register bits can be polled to determine the current mode of operation of the DWC_ddrctl.

[Table 14-4](#) shows the modes of operation.

Table 14-4 Modes of Operation

Operating Mode Register Bits	Mode of Operation of the DWC_ddrctl
000	Uninitialized. The DWC_ddrctl may be in reset, or it may be out of reset, but SDRAM initialization sequence has not yet completed.
001	Normal operating mode. The DWC_ddrctl is ready to accept read and write requests and the DWC_ddrctl may issue reads and writes to SDRAM.
010	SDRAM is in power-down mode.
011	SDRAM is in self-refresh or self-refresh power down mode.

14.4 Software Sequences

This section consists of the following subsections:

- “[Changing Clock Frequencies](#)” on page [424](#)
- “[Software Sequence for Removal of Clocks](#)” on page [432](#)
- “[Power Removal Flow](#)” on page [433](#)
- “[Special Software Self-Refresh Sequence With Controller Initiated Retraining Followed by Burst PPT2](#)” on page [436](#)
- “[Changing RFM Level](#)” on page [438](#)
- “[Sequence of Burst DFI Control Update for Core VDD Change](#)” on page [439](#)
- “[Stopping DFI Sideband Signals During MMFW](#)” on page [440](#)



- Note** 1. The following registers are static registers and therefore are normally set while the controller is in reset. However, it may be set outside of reset as part of the software sequence.

- DFIPHYMSTR.dfi_phymstr_en
- DFIUPD0.dfi_phyupd_en
- DFILPCFG0.dfi_lp_en_sr
- DFILPCFG0.dfi_lp_data_req_en

2. The following registers are quasi-dynamic registers and have to be programmed by using a special sequence described in “[Quasi Dynamic Registers](#)” on page [419](#):

- DFIUPD0.dis_auto_ctrlupd
- HWLPCTL.hw_lp_en
- DFIMISC.dfi_init_complete_en
- DFIMISC.dfi_frequency
- DFIMISC.dfi_init_start
- MSTR2.target_frequency
- ZQCTL2.dis_srx_zqcl

For example, if you need to update DFIUPD0.dis_auto_ctrlupd, you need to follow “[Group 3: Registers that can be Written When Controller is Empty](#)” on page [421](#).

14.4.1 Changing Clock Frequencies

This section consists of the following subsections. They are applicable only when UMCTL2_FREQUENCY_NUM>1. For UMCTL2_FREQUENCY_NUM, changing clock frequency is not supported.

- “[Dynamic Frequency Ratio Change Support](#)” on page [425](#)
- “[Changing Clock Frequencies Without Frequency Set Point](#)” on page [425](#)
- “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#)
- “[Changing Clock Frequencies With Enabling/Disabling DVFS](#)” on page [430](#)
- “[Changing Clock Frequencies With Enabling/Disabling DVFSQ](#)” on page [430](#)



When switching from higher frequency to lower frequency, the controller may violate the JEDEC requirement that no more than 16 refreshes must be issued within 2*tREFI. These extra refreshes are not expected to cause a problem in the SDRAM.

14.4.1.1 Dynamic Frequency Ratio Change Support

The DDRCTL can dynamically change frequency ratio while changing clock frequency.

- Frequency ratio aligns with TMGCFG.frequency_ratio in the REGB_FREQf_CH0 block, where f is the target frequency.
- The value of TMGCFG.frequency_ratio cannot be changed when core_ddrc_rstn=1. Either 1:1:4 or 1:1:2 frequency ratio is applicable for single target frequency.

14.4.1.2 Changing Clock Frequencies Without Frequency Set Point



Note This sequence is only applicable for LPDDR4. For LPDDR5, refer to “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#).

The operations for frequency change flow involve the following steps:

1. During initialization, program the timing register-set REGB_FREQ(1/2/3)_CH, whichever you prefer, with timing settings required for the alternative clock frequency.
2. Disable low-power activities by programming PWRCTL.selfref_en, PWRCTL.powerdown_en and HWLPCCTL.hw_lp_en to '0'.
3. Ensure that STAT.operating_mode=1, which indicates that the controller is in normal mode. Continue to read this until it is read as '1' three times in a row.
4. Set PCTRL.port_en to '0' to block the AXI ports from taking transactions. Poll PSTAT.rd_port_busy_n=0 and PSTAT.wr_port_busy_n=0. Wait until all AXI ports are idle.
5. If Scrubber is instantiated, set SBRCTL.scrub_en to '0', to disable the scrubber and poll SBRSTAT.scrub_busy=0. This indicates that, there are no outstanding SBR read commands.
6. Set OPCTRL1.dis_hif to '1', so that no new commands are accepted by the controller and poll OPCTRLCAM.dbg_wr_q_empty=1, OPCTRLCAM.wr_data_pipeline_empty=1, OPCTRLCAM.dbg_rd_q_empty=1, OPCTRLCAM.rd_data_pipeline_empty=1.
7. If DERATECTL0.derate_eanble is '1', then set DERATECTL0.derate_mr4_pause_fc to '1' to pause automatic MRR to MR4.
8. If DFIUPD0.dis_auto_ctrlupd=0, then set DFIUPD0.dis_auto_ctrlupd to '1' to disable periodic DFI update request temporarily.
9. If DFI PHY Master interface is active in controller (DFIPHYMSTR.dfi_phymstr_en=)¹ then disable it by programming DFIPHYMSTR.dfi_phymstr_en to '0'.
10. Disable DQS Oscillator if it is enabled, by setting DQSOSCCTL0.dqsosc_enable to '0'. Poll DQSOSC-STAT0.dqsosc_state=0.
11. If DFILPCFG0.dfi_lp_en_sr² is '1', set DFILPCFG0.dfi_lp_en_sr to '0' and poll DFISTAT.dfi_lp_ctrl_ack_stat=0.

1. The register DFIPHYMSTR.phymstr_en is a static register and therefore is normally set while the controller is in reset. However, it may be set outside of reset as part of this sequence.
2. The register DFILPCFG0.dfi_lp_en_sr is a static register and therefore is normally set while the controller is in reset. However, it may be set outside of reset as part of this sequence.



Note DFILPCFG0.dfi_lp_en_sr can be disabled only in normal mode or power down with PWRCTL.selfref_en=0 to avoid both entering self refresh power down and internal DFI low power happen at the same time.

12. If DFILPCFG0.dfi_lp_data_req_en is '1', set DFILPCFG0.dfi_lp_data_req_en to '0' and poll DFISTAT.dfi_lp_data_ack_stat=0.
13. Set PWRCTL.en_dfi_dram_clk_disable to '0'.
14. Enter self-refresh mode by setting PWRCTL.selfref_sw to '1'. Poll operating_mode=3, which indicates controller entered into self refresh.
15. Ensure that self-refresh is due to software by checking that STAT.selfref_type[1:0]=2'b10.
16. Set OPCTRL1.dis_dq to '1' so that no CAM commands are de-queued. Poll OPCTRLCAM.wr_data_pipeline_empty=1, OPCTRLCAM.rd_data_pipeline_empty=1.
17. Set DFIMISC.dfi_init_complete_en to '0' to ensure that the controller initialization state machine is not reset if the PHY needs to perform some initialization after the frequency change.
18. If DFI PHY update interface is active in the controller (DFIUPD0.dfi_phyupd_en=1), disable it by programming DFIUPD0.dfi_phyupd_en to '0'.
19. Program DFIMISC.dfi_frequency to the target_frequency value. (Refer to PHY databook for more details.)
20. Program MSTR2.target_frequency to the target_frequency value to select REGB_FREQ* registers.
21. Set DFIMISC.dfi_init_start to '1'. PHY performs internal sequences to cleanly stop pipelines and prepare for clock frequency change.
22. The PHY de-asserts dfi_init_complete. Poll DFISTAT.dfi_init_complete=0.
23. Change the clock frequency to the controller ensuring there are no glitches.
24. Toggle RFSHCTL0.refresh_update_level to allow the new refresh-related register values to propagate to the refresh logic.
25. Set DFIMISC.dfi_init_start to '0'. The PHY performs internal sequences to relock PLLs and calibrate ZQ/Delay-Lines.
26. If the DFI PHY update interface was active prior to frequency change, then enable PHY update Interface by setting DFIUPD0.dfi_phyupd_en to '1'.
27. The PHY asserts dfi_init_complete. Poll DFISTAT.dfi_init_complete=1.
28. Set DFIMISC.dfi_init_complete_en to '1', to indicate to the controller that the PHY has finished frequency change.
29. Exit self-refresh by setting PWRCTL.selfref_sw to '0'. And wait until STAT.operating_mode!=2'b11, indicating that the controller exited from self-refresh state.
30. Restore the values of DFILPCF0.dfi_lp_en_sr and DFILPCFG0.dfi_lp_data_req_en after to self-refresh exit.
31. Update Mode Registers of the DRAM (through MRCTRL0.mr_* / MRCTRL1.mr_*) if necessary due to timing values change because of frequency change.
32. Set OPCTRL1.dis_dq back to '0', to allow read and write traffic to be sent to SDRAM.
33. Program the controller registers back to their default value, which were disabled prior to frequency change.

34. Enable HIF commands by setting `OPCTRL1.dis_hif` to '0'.
35. Reset `DERATECTL0.derate_mr4_pause_fc` to '0' if `DERATECTL0.derate_mr4_pause_fc` has been set to '1' prior to frequency change.
36. Reset `DFIUPD0.dis_auto_ctrlupd` to '0' if `DFIUPD0.dis_auto_ctrlupd` has been set to '1' prior to frequency change.
37. If the DFI PHY Master interface was active prior to frequency change, then enable PHY Master interface by setting `DFIPHYSMSTR.dfi_phymstr_en` to '1'.
38. Enable DQS Oscillator if it was disabled prior to frequency change, by setting `DQSOSCCTL0.dqsosc_enable` to '1'.
39. Enable `PWRCTL.selfref_en`, `PWRCTL.powerdown_en`, `HWLPCTL.hw_lp_en` if it was disabled prior to frequency change.
40. Set `PWRCTL.en_dfi_dram_clk_disable` to '1' if it was programmed to '0' prior to frequency change.
41. Enable AXI ports by programming `PCTRL.port_en` to '1' and scrubber by programming `SBRCTL.scrub_en` to '1', if AXI ports and scrubber were disabled prior to frequency change.

14.4.1.3 Changing Clock Frequencies With Frequency Set Point



- When `MEMC_NUM_RANKS>1` and `MSTR4.wck_on=1`, `MSTR4.ws_off_en` must be equal to 1 if you change the frequency by this sequence.
- When `MEMC_NUM_RANKS=1`, `MSTR4.wck_on` should be equal to 0, as guided in the Synopsys DDRCTL LPDDR5/4 Databook (refer to section 12.3.1 WCK Behavior). If you need to set `MSTR4.wck_on` to '1', disable and re-enable `MSTR4.wck_on` before and after this sequence.

The operations for frequency change flow involve the following steps:

1. During initialization, program the timing register-set `REGB_FREQ(1/2/3)_CH`, whichever you prefer, with the timing settings required for the alternative clock frequency.
2. Disable low-power activities by programming `PWRCTL.selfref_en`, `PWRCTL.powerdown_en` and `HWLPCTL.hw_lp_en` to '0'.
3. Ensure that `STAT.operating_mode=1`, which indicates that the controller is in normal mode. Continue to read this until it is read as '1' three times in a row.
4. Set `PCTRL.port_en` to '0' to block the AXI ports from taking the transactions. Poll `PSTAT.rd_port_busy_n=0` and `PSTAT.wr_port_busy_n=0`. Wait until all AXI ports are idle.
5. If Scrubber is instantiated, set `SBRCTL.scrub_en` to '0', to disable the scrubber and poll `SBRSTAT.scrub_busy=0`. This indicates that there are no outstanding SBR read commands.
6. Set `OPCTRL1.dis_hif` to '1', so that no new commands are accepted by the controller and poll `OPCTRLCAM.dbg_wr_q_empty=1`, `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.dbg_rd_q_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`.
7. If `DERATECTL0.derate_eanble` is '1', then set `DERATECTL0.derate_mr4_pause_fc` to '1' to pause automatic MRR to MR4.
8. If `DFIUPD0.dis_auto_ctrlupd=0`, then set `DFIUPD0.dis_auto_ctrlupd` to '1' to disable periodic DFI update request temporarily.

9. If DFI PHY Master interface is active in the controller (DFIPHYMSTR.dfi_phymstr_en=1), then disable it by programming DFIPHYMSTR.dfi_phymstr_en to '0'.
10. Disable DQS Oscillator if it is enabled, by setting DQSOSCCCTL0.dqsosc_enable to '0'. Poll DQSOSC-STAT0.dqsosc_state=0.
11. Disable automatic ZQ Calibration if it is enabled, by setting ZQCTL0.dis_auto_zq to '1'.
12. If DFILPCFG0.dfi_lp_en_sr is '1', set DFILPCFG0.dfi_lp_en_sr to '0' and poll DFISTAT.dfi_lp_ctrl_ack_stat=0.



Note DFILPCFG0.dfi_lp_en_sr can be disabled only in normal mode or power down with PWRCTL.selfref_en=0 to avoid both entering self refresh power down and internal DFI low power happen at the same time.

13. If DFILPCFG0.dfi_lp_data_req_en is '1', set DFILPCFG0.dfi_lp_data_req_en to '0' and poll DFISTAT.dfi_lp_data_ack_stat=0.
14. Set PWRCTL.en_dfi_dram_clk_disable to '0'.
15. Set PWRCTL.stay_in_selfref to '1' and then set PWRCTL.selfref_sw to '1'.
16. Poll STAT.selfref_state!=3'b000, indicating that the controller has entered into self-refresh state.
17. Read STAT.selfref_state and STAT.selfref_type. If STAT.selfref_state=3'b001 and STAT.selfref_type!=3'b001 ('Self Refresh 1' state caused solely by the software, not by DFI PHY Master hardware state machine), then jump to the next step, else program PWRCTL.selfref_sw to '0' and program PWRCTL.stay_in_selfref to '0', poll STAT.selfref_state=3'b000, and return to step 16.
18. Set OPCTRL1.dis_dq to '1' so that no CAM commands are de-queued. Poll OPCTRLCAM.wr_data_pipeline_empty=1, OPCTRLCAM.rd_data_pipeline_empty=1.
19. In LPDDR4, set PWRCTL.stay_in_selfref to '0' to enter 'Self-refresh power-down' and poll STAT.selfref_state=3'b010. This indicates the controller entered into self-refresh power-down state.
20. Set DFIMISC.dfi_init_complete_en to '0' to ensure that the controller initialization state machine is not reset if the PHY needs to perform some initialization after the frequency change.
21. If DFI PHY update interface is active in the controller (DFIUPD0.dfi_phyupd_en=1), disable it by programming DFIUPD0.dfi_phyupd_en to '0'.
22. Program DFIMISC.dfi_frequency to the target_frequency value (refer to PHY databook for more details).
23. Toggle DFIMISC.dfi_freq_fsp[0] to select a different DRAM FSP for the target PState.
24. Program MSTR2.target_frequency to the target_frequency value, to select REGB_FREQ* registers.
25. Set DFIMISC.dfi_init_start to '1'. PHY performs internal sequences to cleanly stop pipelines and prepare for clock frequency change.
26. The PHY de-asserts dfi_init_complete. Poll DFISTAT.dfi_init_complete=0.
27. Change the clock frequency to the controller ensuring there are no glitches.
28. Toggle RFSHCTL0.refresh_update_level to allow the new refresh-related register values to propagate to the refresh logic.

29. Set `DFIMISC.dfi_init_start` to '0'. The PHY performs internal sequences to relock PLLs and calibrate ZQ/Delay-Lines.
30. If the DFI PHY update interface was active prior to frequency change, then enable PHY update Interface by setting `DFIUPD0.dfi_phyupd_en` to '1'.
31. The PHY asserts `dfi_init_complete`. Poll `DFISTAT.dfi_init_complete=1`.
32. Set `DFIMISC.dfi_init_complete_en` to '1', to indicate to the controller that PHY finished frequency change.
33. Set `ZQCTL2.dis_srx_zqcl` to '1', if `ZQCTL2.dis_srx_zqcl=0`.
34. In LPDDR4, set `PWRCTL.stay_in_selfref` to '1', and then set `PWRCTL.selfref_sw` to '0'. Poll `STAT.selfref_state=3'b011`. This indicates that the controller has entered into 'Self refresh 2' state.
35. If DVFSQ¹ is not active (`MR19.OP[3:2]=0`), issue `zq_calib_short` command by programming `OPCTRLCMD.zq_calib_short` to '1' and poll `OPCTRLSTAT.zq_calib_short_busy=0`.
36. If DVFSQ² is not active (`MR19.OP[3:2]=0`), set `ZQCTL2.dis_srx_zqcl` to '0', if it was set to '1' prior to frequency change.
37. In LPDDR5, set `PWRCTL.selfref_sw` to '0'.
38. Set `PWRCTL.stay_in_selfref` to '0'. Poll `STAT.selfref_state=0`. This indicates that the controller is not in self-refresh state.
39. After self-refresh exit, restore the values of registers `DFILPCF0.dfi_lp_en_sr` and `DFILPCFG0.dfi_lp_data_req_en` to its original value.
40. Set `OPCTRL1.dis_dq` back to '0', to allow read and write traffic to be sent to SDRAM.
41. Program the controller registers back to their default values, which were disabled prior to frequency change.
42. Enable HIF interface by setting `OPCTRL1.dis_hif` to '0'.
43. Reset `DERATECTL0.derate_mr4_pause_fc` to '0' if `DERATECTL0.derate_mr4_pause_fc` has been set to '1' prior to frequency change.
44. Reset `DFIUPD0.dis_auto_ctrlupd` to '0' if `DFIUPD0.dis_auto_ctrlupd` has been set to '1' prior to frequency change.
45. If the DFI PHY Master interface was active prior to frequency change, then enable PHY Master interface by setting `DFIPHYMSTR.dfi_phymstr_en` to '1'.
46. Enable DQS Oscillator if it was disabled prior to frequency change, by setting `DQSOSCCTL0.dqsosc_enable` to '1'.
47. If DVFSQ³ is not active (`MR19.OP[3:2]=0`), enable automatic ZQ Calibration if it was disabled prior to frequency change, by setting `ZQCTL0.dis_auto_zq` to '0'.
48. Enable `PWRCTL.selfref_en`, `PWRCTL.powerdown_en`, `HWLPCTL.hw_lp_en` if it was disabled prior to frequency change.
49. Set `PWRCTL.en_dfi_dram_clk_disable` to '1' if it was programmed to '0' prior to frequency change.
50. Enable AXI ports by programming `PCTRL.port_en` to '1', and scrubber by programming `SBRCTL.scrub_en` to '1', if AXI ports and scrubber were disabled prior to frequency change.

1. DVFSQ is applicable only for LPDDR5. Execute this step for LPDDR4 as DVFSQ is not applicable and LPDDR5 if DVFSQ is not active.
2. DVFSQ is applicable only for LPDDR5. Execute this step for LPDDR4 as DVFSQ is not applicable and LPDDR5 if DVFSQ is not active.
3. DVFSQ is applicable only for LPDDR5. Execute this step for LPDDR4 as DVFSQ is not applicable and LPDDR5 if DVFSQ is not active.



- Ensure to satisfy the timing parameters `tCKFSPE`, `tFC`, `tCKFSPX`.
- When using `dfi_freq_fsp` pin to switch the FSP-OP, Synopsys LPDDR5/4/4X PHY will leave the SDRAM at FSP-1 at the end of initialization (training). So, the following settings will be present.
During initialization, `MSTR2.target_frequency(0) = DFIMISC.dfi_frequency(0) = DFIMISC.dfi_freq_fsp(1) = SDRAM.FSP_OP(1)`. And SDRAM MR values will be programmed as per the controller's FREQ-0 register values.
During frequency change, `MSTR2.target_frequency(1) = DFIMISC.dfi_frequency(1) = DFIMISC.dfi_freq_fsp(0) = SDRAM.FSP_OP(0)`. SDRAM MR values will be programmed as per the controller's FREQ-1 register values.
- After setting `dfi0_init_start=1`, LPDDR5/4/4X PHY sets `VRCG=1` in MR13 (LPDDR4) or MR16 (LPDDR5) before `dfi0_init_complete` becomes equal to '0'.
- After setting `dfi0_init_start=0`, LPDDR5/4/4X PHY sets `VRCG=0` in MR13 (LPDDR4) or MR16 (LPDDR5) before `dfi0_init_complete` becomes equal to '1'.

14.4.1.4 Changing Clock Frequencies With Enabling/Disabling DVFSC



According to JESD209-5C, all banks must be in the IDLE state when DVFSC mode changes as part of FSP-OP switching operation. This section assumes PHY switches FSP-OP during frequency change sequence and device is in IDLE state.

During initialization, set LPDDR5 MR19 mode register to the desired value to enable or disable DVFSC. Then perform the steps described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#). Synopsys LPDDR5/4/4X PHY will program the SDRAM mode registers during initialization and frequency change.

For example, changing the `data_rate` from 6400Mbps (FREQ_0: DVFSC disable) to 1067Mbps (FREQ_1: DVFSC enable).

`FREQ_0: MR19.OP[1:0] = 0` (to use VDD2H: 1.05V rail - {DVFSC disable}).

`FREQ_1: MR19.OP[1:0] = 1` (to use VDD2L: 0.9V rail - {DVFSC enable}).

14.4.1.5 Changing Clock Frequencies With Enabling/Disabling DVFSQ

The operations for frequency change flow involve the following steps:

DVFSQ High-to-Low Transition

1. During initialization, set LPDDR5 MR19 mode register to the desired value to enable or disable DVFSQ. In this case, set `MR19.OP[3:2]=1` (to use `VDDQ = 0.3V` - {DVFSQ LOW}). Synopsys LPDDR5/4/4X PHY will program the SDRAM mode registers during initialization and frequency change.
2. Execute the frequency change steps till step 34 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 34: Set `ZQCTL2.dis_srx_zqcl` to '1', if `ZQCTL2.dis_srx_zqcl = 0`).
3. Poll `ZQSTAT.zq_reset_busy=0`. The SoC can initiate a ZQ Reset operation only if `ZQSTAT.zq_reset_busy` is low.

4. Set `ZQCTL1.zq_reset` to '1' and poll `ZQSTAT.zq_reset_busy=0`. This sets `MR28.OP[0] = 1` (ZQ RESET) at SDRAM, to set default ZQ strength to meet $VDDQ = 0.3V$ operation.
5. Set `MR28.OP[1]`(ZQ Stop) to '1', through Mode Register Read/Write signals (`MRCTRL0.mr_*/MRCTRL1.mr_*`), to disable background calibration.
6. Wait for t_{ZQSTOP} time.
7. Enter DVFSQ (reduce $VddQ$ below 0.5v nominal).
8. Continue operation with reduced $VddQ$.
9. Continue frequency change steps from step 38 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 38: In LPDDR5, set `PWRCTL.selfref_sw` to '0').



`VRCG=0` in `MR16` is set by LPDDR5/4/4X PHY as part of the frequency change procedure while `dfi_init_start=0` and `dfi_init_complete=0`. This is aligned with section 7.7.1.2.1 of JEDEC209-5C.

DVFSQ Low-to-High Transition Without VRCG During $VddQ$ Ramp

1. During initialization, set LPDDR5 MR19 mode register to the desired value to enable or disable DVFSQ. In this case, set `MR19.OP[3:2] = 0` (to use $VDDQ = 0.5V - \{DVFSQ HIGH\}$). Synopsys LPDDR5/4/4X PHY will program the SDRAM mode registers during initialization and frequency change.
2. Execute the frequency change steps till step 19 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 19: Set `OPCTRL1.dis_dq` to '1' so that no CAM commands are de-queued. Poll `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`).
3. SDRAM will be operating at low speed with $VddQ < 0.5v$ nominal when DVFSQ is LOW.
4. Ramp $VddQ$ up to 0.5v nominal.
5. Set `MR28.OP[1]`(ZQ Stop) to '0', through Mode Register Read/Write signals (`MRCTRL0.mr_*/MRCTRL1.mr_*`), to enable background calibration.
6. Wait t_{ZQCALx} .
7. Continue frequency change steps from step 21 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 21: Set `DFIMISC.dfi_init_complete_en` to '0' to ensure that the controller initialization state machine is not reset if the PHY needs to perform some initialization after the frequency change).

DVFSQ Low-to-High Transition with VRCG

1. During initialization, set LPDDR5 MR19 mode register to the desired value to enable or disable DVFSQ. In this case, set `MR19.OP[3:2] = 0` (to use $VDDQ = 0.5V - \{DVFSQ HIGH\}$). Synopsys LPDDR5/4/4X PHY will program the SDRAM mode registers during initialization and frequency change.
2. Execute the frequency change steps till step 19 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 19: Set `OPCTRL1.dis_dq` to '1' so that no CAM commands are de-queued. Poll `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`).
3. SDRAM will be operating at low speed with $VddQ < 0.5v$ nominal when DVFSQ is LOW.
4. Enable VRCG (`MR16.OP[6]=1`), through Mode Register Read/Write signals (`MRCTRL0.mr_*/MRCTRL1.mr_*`).
5. Wait t_{VRCG_enable} (stall traffic).

6. Ramp VddQ up to 0.5v nominal.
7. Set MR28.OP[1](ZQ Stop) to '0', through Mode Register Read/Write signals (MRCTRL0.mr_* / MRCTRL1.mr_*), to enable background calibration.
8. Wait tZQCALx.
9. Continue frequency change steps from step 21 described in section “[Changing Clock Frequencies With Frequency Set Point](#)” on page [427](#) (step 21: Set DFIMISC.dfi_init_complete_en to '0' to ensure that the controller initialization state machine is not reset if the PHY needs to perform some initialization after the frequency change).



Note For more details on ZQRESET and ZQSTOP, refer to section “ZQ Calibration” in LPDDR5X/5/4X Memory Controller Databook.

14.4.2 Software Sequence for Removal of Clocks

Software can be used to keep the SDRAM in self-refresh. AXI and DDRC clocks can be removed when in self-refresh by following the sequence described in [Table 14-5](#).



Note Dynamic and quasi dynamic registers cannot be programmed if any of the clocks has been removed. Clocks must be turned back on before starting the programming sequence. Also the clock gating logic must ensure that there are no glitches on the clocks when they are removed/enabled.

In the following software sequences, ignore the registers that are not available for your configuration. You can generate a report of registers for your configuration after you configure the core in the coreConsultant GUI. See section “Creating Optional Reports and Views” in the user guide.

In DDR5, you must ensure auto self-refresh and auto power-down mode are disabled through PWRCTL.selfref_en and PWRCTL.powerdown_en before the first step in [Table 14-5](#), and recover the auto self-refresh and auto power-down mode setting to original values after the last step in [Table 14-6](#).

Table 14-5 Software Clock Removal Sequence

Step	Description	Comment
1	Write '0' to PCTRL.port_en	Blocks AXI port from taking anymore transactions.
2	Poll PSTAT.rd_port_busy_n = 0 Poll PSTAT.wr_port_busy_n = 0	Waits until all AXI ports are idle.
3	Write '0' to SBRCTL.scrub_en	Disables SBR, only required if SBR is instantiated.
4	Poll SBRSTAT.scrub_busy = 0	Indicates that there are no outstanding SBR read commands, only required if SBR instantiated.
5	Write '1' to PWRCTL.selfref_sw	Causes the system to move to Self Refresh state.
6	Poll STAT.selfref_type= 2'b10 Poll STAT.selfref_state = 3'b010	Waits until Self Refresh state is entered. Waits until Self Refresh Power Down state is entered.

Step	Description	Comment
7	Remove AXI clock	
8	Remove DDRC controller clock	

The clocks must be re-enabled by following the sequence described in [Table 14-6](#).

Table 14-6 Re-Enabling the Clocks

Step	Description	Comment
1	Enable AXI clock	
2 ^a	Enable DDRC controller clock	
3	Write '0' to PWRCTL.selfref_sw	Cause system to exit from self-refresh state.
4	Poll STAT.selfref_type = 2'b00 In DDR5, poll STAT.operating_mode until normal state instead.	Wait until self-refresh state is exited.
5	Write '1' to PCTRL.port_en	AXI ports are no longer blocked from taking transactions.
6	Write '1' to SBRCTL.scrub_en	Enable SBR if desired, only required if SBR instantiated.

a. After this step, it is recommended for DDR4 to re-enable generation of PHY initiated update requests (dfi_phyupd_req). For Synopsys PHY this can be done through the PUB's DSGCR.PUREN.

14.4.3 Power Removal Flow

Table 14-7 Power Removal

Step	Description	Comment
1	Disable low-power activities by programming PWRCTL.selfref_en, PWRCTL.powerdown_en, and HWLPCCTL.hw_lp_en to '0'.	
2	Ensure that STAT.operating_mode=1, which indicates that the controller is in normal mode.	Continue to read this until it is read as '1' three times in a row.
3	Write '0' to PCTRL_n.port_en	Blocks AXI port from taking anymore transactions.
4	Poll PSTAT.rd_port_busy_n = 0 Poll PSTAT.wr_port_busy_n = 0	Waits until all AXI ports are idle.
5	Write '0' to SBRCTL.scrub_en	Disables SBR, only required if SBR instantiated.
6	Poll SBRSTAT.scrub_busy = 0	Indicates that there are no outstanding SBR read commands, only required if SBR instantiated.

Step	Description	Comment
7	If DFIUPD0.dis_auto_ctrlupd=0, then set DFIUPD0.dis_auto_ctrlupd to '1' to disable periodic DFI update request temporarily. It can be enabled again at registers reprogramming during re-enabling the power.	
8	Set DFIPHYMSTR.dfi_phymstr_en=0 to prevent dfi_phymstr_ack from being asserted.	
9	Polling STAT.selfref_type=NOT "01" to confirm whether the current situation is during the retraining of PHY master.	
10	Set ZQ Stop(MR28 OP[1]) to '1'. Send MRW command using MRCTRL0 and MRCTRL1. (LPDDR5 only)	For LPDDR5, ZQ Stop needs to be set to '1', if VDDQ needs to be turned off during self-refresh power down state.
11	If DFILPCFG0.dfi_lp_en_sr is '1', set DFILPCFG0.dfi_lp_en_sr to '0' and poll DFISTAT.dfi_lp_ctrl_ack_stat=0.	
12	If DFILPCFG0.dfi_lp_data_req_en is '1', set DFILPCFG0.dfi_lp_data_req_en to '0' and poll DFISTAT.dfi_lp_data_ack_stat=0.	
13	Write '1' to PWRCTL.selfref_sw	Causes system to move to self-refresh state.
14	Poll STAT.selfref_type= 2'b10 Poll STAT.selfref_state = 3'b010	Waits until self-refresh power down state is entered.
15	Program DFIMISC.dfi_frequency as required. Refer to PHY databook for more details.	0x1f for PHY deep-sleep/retention.
16	Set DFIMISC.dfi_init_start to '1'	
17	The PHY de-asserts dfi_init_complete. The controller polls DFISTAT.dfi_init_complete=0.	
18	Set DFIMISC.dfi_init_start to '0'.	
19	The PHY asserts dfi_init_complete. The controller polls DFISTAT.dfi_init_complete=1.	

Step	Description	Comment
20	Place IOs in retention mode	Refer to relevant PHY databook for more information.
21	Remove power	

Table 14-8 Re-Enabling the Power

Step	Description	Comment
1	Enable Power	
2	Reset controller/PHY by driving core_ddrc_rstn = 1'b0, aresetn_n = 1'b0, presetn = 'b0	
3	Remove APB reset, presetn = 1'b1, and reprogram the registers to pre-power removal values	
4	Program RFSHCTL0.dis_auto_refresh = 1'b1	
5	Program RFMMD0.init_raa_cnt = RAAMMT (= 8*RFMMD0.raaimt*2*(RFMMD0. raamult + 1))	
6	Program INITTMG0.skip_dram_init = 2'b11	Skips the DRAM initialization routine and starts up in self-refresh mode.
7	Program PWRCTL.selfref_sw = 1'b1	Keeps the controller in self-refresh mode.
8	Remove the controller reset core_ddrc_rstn = 1'b1 aresetn_n = 1'b1	
9	Program DFIMISC.dfi_init_complete_en to 1'b0	PHY initialization needs to be rerun so set to '0' until initialization complete.
10	Program DFIMISC.dfi_reset = 1'b1	
11	Program RFSHCTL0.dis_auto_refresh = 1'b0	
12	Run PHY initialization/training as required, including removing the IOs from retention mode including steps below	Refer to the relevant PHY databook for more information.

Step	Description	Comment
13	Program DFIMISC.dfi_frequency and DFIMISC.dfi_freq_fsp as required. See PHY databook for more details.	If HWFFC was enabled before removing power, set registers as following: 1. Set DFIMISC.dfi_frequency and MSTR2.target_frequency to HWFFCSTAT.current_frequency before removing power. 2. Set DFIMISC.dfi_freq_fsp and HWFFCCTL.init_fsp to HWFFCSTAT.current_fsp before removing power. 3. In case of UMCTL2_FREQUENCY_NUM=14, set dfi_frequency = target_frequency - 7, if target_frequency is greater than 7.
14	Set DFIMISC.dfi_init_start to '1'	
15	The PHY asserts dfi_init_complete. The controller polls DFISTAT.dfi_init_complete = 1.	
16	Set DFIMISC.dfi_init_start to '0'.	
17	Program DFIMISC.dfi_init_complete_en to 1'b1	Indicates to controller that PHY has completed re-training/initialization.
18	Program PWRCTL.selfref_sw = 1'b0	Trigger self-refresh exit.
19	Poll STAT.selfref_type = 2'b00	Wait until self-refresh state is exited.
20	Poll STAT.operating_mode for Normal Mode entry	
21	Set ZQ Stop(MR28 OP[1]) to '0'. Send MRW command using MRCTRL0 and MRCTRL1. (LPDDR5 only)	For LPDDR5, ZQ Stop needs to be set to '0', if ZQ Stop is set to '1' before power removal.
22	Write PCTRL.port_en = 1	AXI ports are no longer blocked from taking transactions.
23	Write '1' to SBRCTL.scrub_en	Enable SBR if desired, only required if SBR instantiated.

14.4.4 Special Software Self-Refresh Sequence With Controller Initiated Retraining Followed by Burst PPT2



- When MEMC_NUM_RANKS>1 and MSTR4.wck_on=1, MSTR4.ws_off_en must be equal to 1 if you change the frequency by this sequence.
- When MEMC_NUM_RANKS=1, MSTR4.wck_on should be equal to 0, as guided in the Synopsys DDRCTL LPDDR5/4 Databook (refer to section 12.3.1 WCK Behavior). If you need to set MSTR4.wck_on to '1', disable and re-enable MSTR4.wck_on before and after this sequence.

The operations for software self-refresh along with retraining (PHY retraining and Burst PPT2) on SWSR exit involve the following steps:

1. During initialization, PPT2 related registers are assumed to be set, including `DFIUPDTMG2.ppt2_en=1`.
2. Disable low-power activities by programming `PWRCTL.selfref_en`, `PWRCTL.powerdown_en` and `HWPCTL.hw_lp_en` to '0'.
3. Ensure that `STAT.operating_mode=1`, which indicates that the controller is in normal mode. Continue to read this until it is read as '1' three times in a row.
4. Set `PCTRL.port_en` to '0' to block the AXI ports from taking transactions. Poll `PSTAT.rd_port_busy_n=0` and `PSTAT.wr_port_busy_n=0`. Wait until all AXI ports are idle.
5. If Scrubber is instantiated, set `SBRCTL.scrub_en` to '0', to disable the scrubber and poll `SBRSTAT.scrub_busy=0`. This indicates that there are no outstanding SBR read commands.
6. Set `OPCTRL1.dis_hif` to '1', so that no new commands are accepted by the controller and poll `OPCTRLCAM.debug_wr_q_empty=1`, `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.debug_rd_q_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`.
7. If `DERATECTL0.derate_eanble` is '1', then set `DERATECTL0.derate_mr4_pause_fc` to '1' to pause automatic MRR to MR4.
8. If `DFIUPD0.dis_auto_ctrlupd=0`, then set `DFIUPD0.dis_auto_ctrlupd` to '1' to disable periodic DFI update request temporarily.
9. If `DFILPCFG0.dfi_lp_en_sr` is '1', set `DFILPCFG0.dfi_lp_en_sr` to '0' and poll `DFISTAT.dfi_lp_ctrl_ack_stat=0`.



`DFILPCFG0.dfi_lp_en_sr` can be disabled only in normal mode or power-down with `PWRCTL.selfref_en=0` to avoid both entering self-refresh power-down and internal DFI low-power happen at the same time.

10. If `DFILPCFG0.dfi_lp_data_req_en` is '1', set `DFILPCFG0.dfi_lp_data_req_en` to '0' and poll `DFISTAT.dfi_lp_data_ack_stat=0`.
11. Set `PWRCTL.en_dfi_dram_clk_disable` to '0'.
12. Enter self-refresh mode by setting `PWRCTL.selfref_sw` to '1'. Poll `operating_mode=3`, which indicates that the controller has entered into self-refresh.
13. Ensure that self-refresh is due to software by checking that `STAT.selfref_type[1:0]=2'b10`.
14. Set `OPCTRL1.dis_dq` to '1' so that no CAM commands are de-queued. Poll `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`.
15. Set `DFIMISC.dfi_init_complete_en` to '0' to ensure that the controller initialization state machine is not reset if the PHY needs to perform some initialization while retraining.
16. Program `DFIMISC.dfi_frequency` to 5'h0c, that is Retrain only.
17. Long term Idle.
18. Set `DFIMISC.dfi_init_start` to '1'. PHY performs internal sequences to cleanly stop pipelines.
19. The PHY de-asserts `dfi_init_complete`. Poll `DFISTAT.dfi_init_complete=0`.
20. Set `DFIMISC.dfi_init_start` to '0'. The PHY performs internal sequences to re-lock PLLs and calibrate ZQ/Delay-Lines.
21. The PHY asserts `dfi_init_complete`. Poll `DFISTAT.dfi_init_complete=1`.

22. Set DFIMISC.dfi_init_complete_en to '1', to indicate to the controller that the PHY has finished retraining.
23. Start the Burst PPT2 process. If ZQCTL0.dis_auto_zq is '0', set ZQCTL0.dis_auto_zq to '1'.
24. Program PPT2CTRL0.ppt2_burst to '1'.
25. Exit self-refresh by setting PWRCTL.selfref_sw to '0'. And wait until STAT.operating_mode!=2'b11, indicating that the controller has exited from the self-refresh state.
26. Once the selfref_sw is made, the Burst PPT2 process starts. Wait for Burst PPT2 to complete by polling PPT2STAT0.ppt2_burst_busy to '0'.
27. Reset ZQCTL0.dis_auto_zq to '0' if ZQCTL0.dis_auto_zq has been set to '1' prior to Burst PPT2.
28. Restore the values of DFILPCF0.dfi_lp_en_sr and DFILPCFG0.dfi_lp_data_req_en after the self-refresh exit.
29. Set OPCTRL1.dis_dq back to '0', to allow read and write traffic to be sent to SDRAM.
30. Program the controller registers back to their default values, which were disabled prior to retraining.
31. Enable HIF commands by setting OPCTRL1.dis_hif to '0'.
32. Reset DERATECTL0.derate_mr4_pause_fc to '0' if DERATECTL0.derate_mr4_pause_fc has been set to '1' prior to retraining.
33. Reset DFIUPD0.dis_auto_ctrlupd to '0' if DFIUPD0.dis_auto_ctrlupd has been set to '1' prior to retraining.
34. Enable PWRCTL.selfref_en, PWRCTL.powerdown_en, HWLPCCTL.hw_lp_en if they were disabled prior to retraining.
35. Set PWRCTL.en_dfi_dram_clk_disable to '1' if it was programmed to '0' prior to retraining.
36. Enable AXI ports by programming PCTRL.port_en to '1' and scrubber by programming SBRCTL.scrub_en to '1', if AXI ports and scrubber were disabled prior to retraining.

14.4.5 Changing RFM Level

The operations for changing the RFM level in normal operation involve the following steps:

1. Disable low-power activities by programming PWRCTL.selfref_en and HWLPCCTL.hw_lp_en to '0'.
2. Ensure that STAT.operating_mode=1, which indicates that the controller is in normal mode. Continue to read this until it is read as '1' three times in a row.
3. Set OPCTRL1.dis_hif to '1' so that no new commands are accepted by the controller, and poll OPCTRLCAM.dbg_wr_q_empty =1, OPCTRLCAM.wr_data_pipeline_empty =1, OPCTRLCAM.dbg_rd_q_empty =1, OPCTRLCAM.rd_data_pipeline_empty =1.
4. Poll RFMSTAT.rank_raa_cnt_gt0 to '0'. RAA counter is decremented by automatic Refresh command. It would take 8 * tREFI to complete this step in the worst case. Toggling RFSHCTL0.refresh_update_level can shorten the period.
5. Set MRCTRL0.dis_mrrw_trfc to '1'.
6. Send MRW commands using MRCTRL0 and MRCTRL1 to change the RFM level (MR57:OP[7:6]).
7. Set MRCTRL0.dis_mrrw_trfc back to '1'.
8. Send MRR commands using MRCTRL0 and MRCTRL1 to read RAAIMT (MR27:OP[5:1]), RAAMULT (MR27:OP[7:6]) and RAADEC (MR57:OP[1:0]).
9. Program RFMMOD0.raaimt, RFMMOD0.raadec, RFMMOD0.raamult, and RFMMOD0.rfmth_rm_thr for selected RFM level.

10. Enable HIF commands by setting `OPCTRL1.dis_hif` to '0'.
11. Program the controller registers back to their default values, which were disabled prior to changing RFM level sequence.

14.4.6 Sequence of Burst DFI Control Update for Core VDD Change

1. Disable DQS Oscillator if it is enabled, by setting `DQSOSCCTL0.dqsosc_enable` to '0'.
2. If `DERATECTL0.derate_enable` is '1', then set `DERATECTL0.derate_mr4_pause_fc` to '1' to pause automatic MRR to MR4.
3. Disable low-power activities by programming `PWRCTL.selfref_en` and `PWRCTL.powerdown_en` to '0'.
4. Ensure that `STAT.operating_mode` is equal to '1', which indicates that the controller is in normal mode.

Continue to read this until it is read as '1' three times in a row.

5. Disable automatic ZQ Calibration if it is enabled, by setting `ZQCTL0.dis_auto_zq` to '1'.
6. If `DFIUPD0.dis_auto_ctrlupd` is '0', then set `DFIUPD0.dis_auto_ctrlupd` to '1' to disable periodic DFI update request temporarily.
7. If `DFIUPDTMG2.ppt2_en` is '1', then set `DFIUPDTMG2.ppt2_en` to '0' to disable PPT2 feature.
8. Poll `DQSOSCSTAT0.dqsosc_state` is '0'.
9. Program `DFIUPDTMG1.dfi_t_ctrlupd_burst_interval_x8` based on current frequency.
10. Set `OPCTRLCMD.ctrlupd_burst` to '1' to start burst DFI control update
11. After core VDD change duration, Set `OPCTRLCMD.ctrlupd_burst` to '0' and poll `OPCTRL-STAT.ctrlupd_burst_busy`.
12. If `DFIUPDTMG2.ppt2_en` was active prior to core VDD change, then enable PPT2 by setting `DFIUPDTMG2.ppt2_en` to '1'.
13. If `DFIUPD0.dis_auto_ctrlupd` was active prior to core VDD change, then enable auto control update by setting `DFIUPD0.dis_auto_ctrlupd` to '0'.
14. Enable automatic ZQ Calibration if it was disabled prior to core VDD change, by setting `ZQCTL0.dis_auto_zq` to '0'.
15. Enable `PWRCTL.selfref_en` and `PWRCTL.powerdown_en` if they were disabled prior to core VDD change.
16. Reset `DERATECTL0.derate_mr4_pause_fc` to '0' if `DERATECTL0.derate_mr4_pause_fc` has been set to '1' prior to frequency change.
17. Enable DQS Oscillator if it was disabled prior to core VDD change, by setting `DQSOSCCTL0.dqsosc_enable` to '1'.

14.4.7 Stopping DFI Sideband Signals During MMFW

14.4.7.1 Stop DFI Sideband Signals



- `BlockDfiInterfaceEn` and `MicroContMuxSel` are the registers of Synopsys LPDDR5X/5/4X PHY. Refer to the *Synopsys LPDDR5X/5/4X PHY Utility Block (PUB) Databook* for more details.
- LPDDR5X/5/4X PHY has a signal `dwc_1pddr5xphy_pmu_busy`, which indicates executing MMFW. Asserting `dwc_1pddr5xphy_pmu_busy` means that you must stop asserting DFI sideband signals during it. So, you have a way to start the sequence for stopping DFI sideband signals after `dwc_1pddr5xphy_pmu_busy` is asserted. However, the controller does not use `dwc_1pddr5xphy_pmu_busy` because it is too late to stop asserting DFI sideband signals if you use `dwc_1pddr5xphy_pmu_busy`.

The operations for stop DFI sideband signals flow involve the following steps:

1. Disable DQS Oscillator to stop `dwc_ddrphy_snoop`, by setting `DQSOSCCTL0.dqsosc_enable` to '0' if it is enabled.
2. Disable all the low power features for the purpose of disabling `dfi_lp_ctrl_req` by programming `PWRCTL.selfref_en`, `PWRCTL.dsm_en`, and `PWRCTL.powerdown_en` to '0'.
3. Disable `dfi_lp_data_req` by programming `DFILPCFG0.dfi_lp_data_req_en` to '0'.
4. Ensure that `STAT.operating_mode=1`, which indicates that the controller is in normal mode.
Continue to read this until it is read as '1' three times in a row.
5. Disable `dfi_ctrlupd_req` by programming `DFIUPDTMG2.ppt2_en`.
6. Set `PCTRL.port_en` to '0' to block the AXI ports from taking transactions. Poll `PSTAT.rd_port_busy_n=0` and `PSTAT.wr_port_busy_n=0`. Wait until all AXI ports are idle.
7. If Scrubber is instantiated, set `SBRCTL.scrub_en` to '0', to disable the scrubber and poll `SBRSTAT.scrub_busy=0`. This indicates that there are no outstanding SBR read commands.
8. Set `OPCTRL1.dis_hif` to '1', so that no new commands are accepted by the controller and poll `OPCTRLCAM.dbg_wr_q_empty=1`, `OPCTRLCAM.wr_data_pipeline_empty=1`, `OPCTRLCAM.dbg_rd_q_empty=1`, `OPCTRLCAM.rd_data_pipeline_empty=1`.
9. Disable `dfi_ctrlupd_req` by programming `DFIUPD0.dis_auto_ctrlupd` to '1'.
10. If `BlockDfiInterfaceEn` of PHY Registers is equal to '0', set `MicroContMuxSel` to '0' and `BlockDfiInterfaceEn` to '1' to stop `dfi_phyupd_ack` and `dfi_phymstr_ack`.
11. Set `MicroContMuxSel` to '1' not to allow to access any non-APBONLY CSRs.
12. Enable HIF interface by setting `OPCTRL1.dis_hif` to '0'.
13. Enable AXI ports by programming `PCTRL.port_en` to '1' and scrubber by programming `SBRCTL.scrub_en` to '1', if AXI ports and scrubber were disabled prior to stop DFI sideband signals sequence.
14. Poll `DQSOSCSTAT0.dqsosc_state` equal to '0'.



Step 9 can be skipped when `DDRCTL_OVERRIDE_PPT2` is defined.

14.4.7.2 Resume DFI Sideband Signals

The operations for resume DFI sideband signals flow involve the following steps:

1. If BlockDfiInterfaceEn of PHY Registers is equal to '0' before “[Stop DFI Sideband Signals](#)”, set MicroContMuxSel to '0' and BlockDfiInterfaceEn to '0' to resume dfi_phyupd_req and dfi_phymstr_req.
2. If BlockDfiInterfaceEn of PHY Registers is equal to '1' before “[Stop DFI Sideband Signals](#)”, set MicroContMuxSel to '1' to restore it.
3. Set PCTRL.port_en to '0' to block the AXI ports from taking transactions. Poll PSTAT.rd_port_busy_n=0 and PSTAT.wr_port_busy_n=0. Wait until all AXI ports are idle.
4. If Scrubber is instantiated, set SBRCTL.scrub_en to '0', to disable the scrubber and poll SBRSTAT.scrub_busy=0. This indicates that there are no outstanding SBR read commands.
5. Set OPCTRL1.dis_hif to '1', so that no new commands are accepted by the controller and poll OPCTRLCAM.debug_wr_q_empty=1, OPCTRLCAM.wr_data_pipeline_empty=1, OPCTRLCAM.debug_rd_q_empty=1, OPCTRLCAM.rd_data_pipeline_empty=1.
6. Enable dfi_ctrlupd_req by programming DFIUPD0.dis_auto_ctrlupd to '0' if this register is equal to '0' before “[Stop DFI Sideband Signals](#)”.
7. Enable HIF interface by setting OPCTRL1.dis_hif to '0'.
8. Enable AXI ports by programming PCTRL.port_en to '1' and scrubber by programming SBRCTL.scrub_en to '1', if AXI ports and scrubber were disabled prior to resume dfi sideband signals sequence.
9. Enable DQS Oscillator to resume dwc_ddrphy_snoop by setting DQSOSCCTL0.dqsosc_enable to '1' if it is enabled before “[Stop DFI Sideband Signals](#)”.
10. Resume dfi_ctrlupd_req by programming DFIUPDTMG2.ppt2_en to '1' if these registers are equal to '1' before “[Stop DFI Sideband Signals](#)”.
11. Restore PWRCTL.selfref_en, PWRCTL.dsm_en, and PWRCTL.powerdown_en to '1' if these registers are equal to '1' before “[Stop DFI Sideband Signals](#)”.
12. Enable dfi_lp_data_req by programming DFILPCFG0.dfi_lp_data_req_en to '1' if this register is equal to '1' before “[Stop DFI Sideband Signals](#)”.



Steps 3,4,5,7,8 can be skipped when DDRCTL_OVERRIDE_PPT2 is defined.

15

Automotive

DWC_ddrctl is ISO 26262 Compliant (ASIL B Random Hardware Failures / ASIL D Systematic). This section describes the automotive features supported by the controller and enabled by the DDRCTL_PRODUCT_NAME==19 parameter when you have the corresponding automotive license. The following topics are discussed:

- “[Overview of Automotive Safety Feature](#)” on page [444](#)
- “[Automotive Safety Package Documents](#)” on page [446](#)
- “[Automotive Specific Features](#)” on page [447](#)

15.1 Overview of Automotive Safety Feature

To make an IP better suited for applications in the automotive space, functional safety needs to be considered throughout the IP development process. ISO 26262 addresses functional safety and requires the hardware components to be analyzed with respect to their ability to detect and/or control the effects of random hardware faults. Random hardware faults can occur at any point in time during the life cycle of the hardware component. They can take the form of permanent faults, or transient faults, and require safety mechanisms to be in place to ensure that severe consequences do not occur because of those faults.

The DDRCTL includes a broad range of internal safety mechanisms which assist in handling random hardware faults. These complement other mechanism already defined in the DDRCTL.

The DDRCTL IP has been analyzed in the scope of ISO 26262 targeting ASIL B for a particular reference configuration.

Reference Configuration

Users of the DDRCTL must understand that the ASIL B analysis uses a reference configuration with major features such as, REGPAR, OCCAP, OCPAR, OCSAP, and Inline ECC. The exact reference configuration is described in the accompanying Safety Manual.

The reference configuration internal safety mechanisms monitor the IP logic and report any detected errors to the application. The application should take an action to handle the error conditions and ensure safe operation of the hardware component.

FMEDA

As part of the ASIL B Random Hardware Failures analysis, a Failure Mode, Effects, and Diagnostic Analysis (FMEDA) analysis has been performed for the DDRCTL IP. The analysis utilizes a per-transistor failure rate derived in accordance with IEC TR -62380. This, in conjunction with transistor count information allows one to determine the failure rate for the IP which can then be distributed between the modules of the design according to their relative area.

Safety Goals are defined for the IP. A list of failure modes and their effects is derived for each module and each of the failure modes is assigned a portion of the failure rate of the module they belong to. Failure modes are reviewed in the context of which Safety Goals they may cause to be violated, and if the violation is of type single-point or multi-point. If there is a diagnostic capability for each failure mode a Diagnostic Coverage number is assigned. The process is repeated for all failure modes and finally results are accumulated, and IP safety metrics are collected in the FMEDA.

DFMEA

A Design Failure Mode and Effects Analysis has been performed. This analyses the nominal and safety functions of each of the major blocks in the design and identifies the potential failure modes, their potential effects impacting safety goals at the top-level and the potential causes at the top-level within each of the blocks. It assigns Severity, Observability, and Detection ratings to each failure mode and identifies controls for prevention and detection of the failure modes, which reduce the risk of the failure mode occurring. Where appropriate, it recommends actions for SoC implementers to reduce risk of design errors during integration of the IP.

DFA

A Dependent Failure Analysis has been performed. The results describe the design risk factors leading to dependent failures and identifies mitigation actions to prevent or control the dependent failures. Identified actions for integration, verification, or validation are defined or referenced in the "Assumptions of Use" section of the Safety Manual. The DFA document covers:

- Common Cause Failure (CCF) analysis, including mission circuits and their corresponding safety mechanisms, corresponding safety requirements, dependent failure initiators, coupling factors, measures for fault avoidance, measures for fault control, verification method, and verification evidence. Both internal and external safety mechanisms are considered.
- Cascading Failure (CF) analysis, including pairs of parts, subparts or elementary subparts affected by cascading failures, affected safety requirements, analysis scope, cascading failure mechanism, measures for fault avoidance, measures for fault control, verification method, and verification evidence.

15.2 Automotive Safety Package Documents

The automotive safety package contains several automotive specific documents as outlined:

- FMEDA
- Safety Manual
- DFMEA
- DFA
- Safety Case Report
- Functional Safety Assessment Report
- SG Quality Manual

You can enable the safety package by setting the `DDRCTL_PRODUCT_NAME==19`. You can access the automotive documents in your workspace directory at `workspace/automotive`.

15.3 Automotive Specific Features

DDRCTL_PRODUCT_NAME==19 is required to access automotive safety package and automotive specific features. If you do not have a valid license, this setting is not allowed in coreConsultant for you. The valid license is DWC-AC-LPDDR5X-CONTROLLER. Only with this license, and configured with DDRCTL_PRODUCT_NAME==19 are the following automotive specific features available:

- “[Registers Parity Protection \(REGPAR\)](#)” on page [393](#)
- “[On-Chip Command and Address Path Protection \(OCCAP\)](#)” on page [398](#)
- “[On-Chip Parity \(OCPAR\)](#)” on page [386](#)
- “[On-Chip External SRAM Address Protection \(OCSAP\)](#)” on page [406](#)

A

Area, Speed, and Power

This appendix provides power consumption, gate count, and latency information. It contains the following sections:

- “[Timing](#)” on page [450](#)
- “[Area and Power](#)” on page [451](#)
- “[Latency Analysis](#)” on page [454](#)

A.1 Timing

The maximum achievable system clock (`core_ddrc_core_clk`) frequency depends on the following:

- Complexity/Size of DDRCTL configuration
- Frequency ratio selected
- Process technology
- Library PVT corner used for timing closure

Therefore, the supported SDRAM speed grades depend on the timing closure frequency.

For example, in a slow technology, where the DDRCTL can reach a clock frequency of 400 MHz, data rates up to 1600 Mbps can be supported in DFI 1:2 frequency ratio mode, but 3200 Mbps can only be supported in DFI 1:4 frequency ratio mode.

If you need 4267 Mbps with LPDDR4 SDRAMs in DFI 1:2 frequency ratio mode, clock frequency must be 1067 MHz. If you need 6400 Mbps with LPDDR5 SDRAMs, clock frequency must be 800 MHz, and LPDDR5-6400 can only be supported in DFI 1:4 frequency ratio mode.

A.2 Area and Power

The reference gate counts are generated for LPDDR54 and LPDDR5X controller configurations. These reference gate counts, which are NAND-equivalent gates, are generated based on area using Synopsys Fusion Compiler using the industry standard 5 nm high-speed library with a combination of SVT, LVT and uLVT cells. These results include scan chains logic. The "Design for Test" options "Test Ready Compile" and "Insert Dft" are enabled in the coreConsultant "Synthesis" activity. A synthesis clock frequency of 1066 MHz is used for core_ddrc_core_clk, aclk, sbr_clk, and bsm_clk. The frequency of pclk is 533 MHz. Note that the same synthesis results as in this example may not be obtained. There are always fluctuations in the synthesis results.



Note Contact Synopsys if power estimate is required.

Table A-1 Area and Gate Count for Each LPDDR5X Configuration

Cfg	Parameter Settings										Gate Count [ND2]
01	UMCTL2_A_NPORTS = 1 MEMC_DRAM_DATA_WIDTH = 16	AXI	1	64	0	0	0	16	2	16	1047K
02	UMCTL2_A_NPORTS = 1	AXI	1	64	0	0	0	32	2	16	1296K
03		AXI	2	64	0	0	0	32	2	16	1835K
04	MEMC_DRAM_DATA_WIDTH = 16	AXI	2	64	0	0	0	64	2	16	2704K
05	UMCTL2_OCCAP_EN = 1	AXI	2	64	1	0	0	32	2	16	2733K
06	MEMC_ECC_SUPPORT = 1 MEMC_INLINE_ECC = 1	AXI	2	64	0	1	0	32	2	16	2583K
07	MEMC_BURST_LENGTH = 32	AXI	2	64	0	0	0	32	2	32	2459K
08	MEMC_NUM_RANKS = 4	AXI	2	64	0	0	0	32	4	16	2116K
09	See 'Configurations' table	AXI	3	64	0	0	0	32	2	16	4436K
10	See 'Configurations' table	AXI	3	64	0	0	1	32	4	32	4940K

Table A-2 Configurations

Parameter	Configuration									
	01	02	03	04	05	06	07	08	09	10
DDRCTL_SYS_INTF	1	1	1	1	1	1	1	1	1	1
UMCTL2_A_NPORTS	1	1	2	2	2	2	2	2	3	3
MEMC_NO_OF_ENTRY	64	64	64	64	64	64	64	64	64	64
UMCTL2_OCCAP_EN	0	0	0	0	1	0	0	0	0	0
MEMC_INLINE_ECC	0	0	0	0	0	1	0	0	0	0
MEMC_ECC_SUPPORT	0	0	0	0	0	1	0	0	0	0
MEMC_DRAM_DATA_WIDTH	16	32	32	64	32	32	32	32	32	32
MEMC_NUM_RANKS	2	2	2	2	2	2	2	4	2	4
MEMC_BURST_LENGTH	16	16	16	16	16	16	32	16	16	32
MEMC_LINK_ECC	0	0	0	0	0	0	0	0	0	1
UMCTL2_HWFFC_EN	0	0	0	0	0	0	0	0	0	1
UMCTL2_EXCL_ACCESS	0	0	0	0	0	0	0	0	0	16
UMCTL2_FREQUENCY_NUM	1	1	1	1	1	1	1	1	4	4
UMCTL2_VPRW_EN	0	0	0	0	0	0	0	0	1	1
UMCTL2_EXCL_ACCESS	0	0	0	0	0	0	0	0	16	16
UMCTL2_EXT_PORTPRIO	0	0	0	0	0	0	0	0	1	1
UMCTL2_AXI_LOWPWR_NOPX_CNT	0	0	0	0	0	0	0	0	16	16
UMCTL2_ASYNC_REG_N_SYNC	2	2	2	2	2	2	2	2	4	4
UMCTL2_ASYNC_DDRC_N_SYNC	2	2	2	2	2	2	2	2	4	4
UMCTL2_ASYNC_LP4DCI_N_SYNC	2	2	2	2	2	2	2	2	4	4
DDRCTL_BANK_HASH	0	0	0	0	0	0	0	0	1	1
UMCTL2_A_IDW	8	8	8	8	8	8	8	8	32	32
UMCTL2_A_NSAR	0	0	0	0	0	0	0	0	4	4
UMCTL2_XPI_USE2RAQ_0	0	0	0	0	0	0	0	0	1	1
UMCTL2_XPI_USE2RAQ_1	0	0	0	0	0	0	0	0	1	1
UMCTL2_XPI_USE2RAQ_2	0	0	0	0	0	0	0	0	1	1
UMCTL2_RRB_EXTRAM_0	0	0	0	0	0	0	0	0	1	1

Parameter	Configuration									
	01	02	03	04	05	06	07	08	09	10
UMCTL2_RRB_EXTRAM_1	0	0	0	0	0	0	0	0	1	1
UMCTL2_RRB_EXTRAM_2	0	0	0	0	0	0	0	0	1	1
UMCTL2_AXI_RAQD_0	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_RAQD_1	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_RAQD_2	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_WAQD_0	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_WAQD_1	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_WAQD_2	4	4	4	4	4	4	4	4	32	32
UMCTL2_AXI_RDQD_0	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_RDQD_1	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_RDQD_2	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_WDQD_0	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_WDQD_1	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_WDQD_2	10	10	10	10	10	10	10	10	128	128
UMCTL2_AXI_WRQD_0	10	10	10	10	10	10	10	10	32	64
UMCTL2_AXI_WRQD_1	10	10	10	10	10	10	10	10	32	64
UMCTL2_AXI_WRQD_2	10	10	10	10	10	10	10	10	32	64

A.3 Latency Analysis

This section shows a breakdown of the best-case latencies of the LPDDR5X/5/4X Memory Controller for open pages and is discussed in the following sections:

- “[AXI - HIF - AXI Write Latency With Controller Idle](#)” on page [454](#)
- “[HIF - DFI - HIF Latency With Controller Idle](#)” on page [458](#)

Both write and read latencies along with parameters which affect the latency for each system interface configuration are shown in Figures A-1 to A-2:

Figure A-1 Read and Write Latencies for AXI Interface Configuration

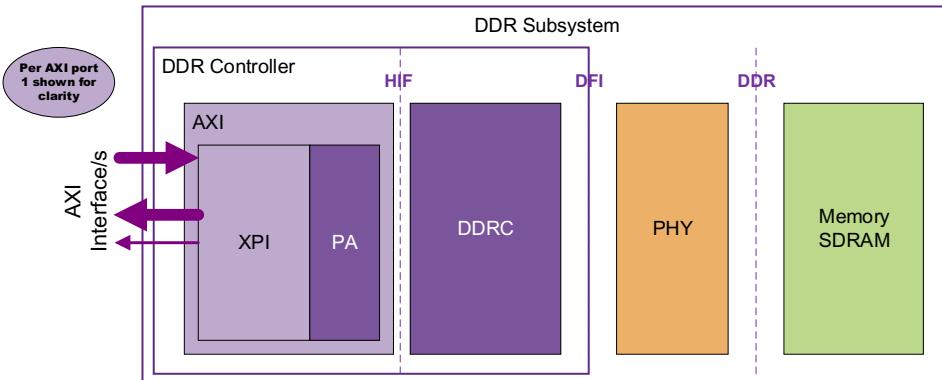
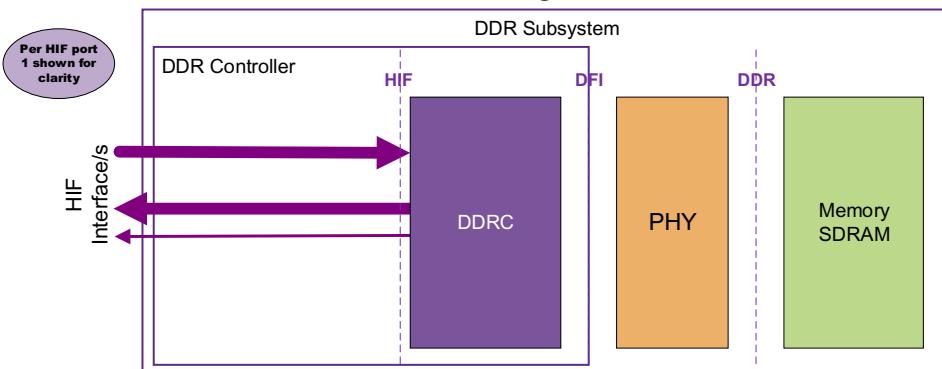


Figure A-2 Read and Write Latencies for HIF Interface Configuration



A.3.1 AXI - HIF - AXI Write Latency With Controller Idle

Various elements of the AXI interface write latency which are under the control of the DDRCTL are shown in [Figure A-1](#) on page [454](#). It shows the components of write latency for which DDRCTL is responsible.

A.3.1.1 AXI-HIF-AXI Writes Interface

The AXI write interface block diagram in [Figure A-3](#) shows the paths for the latencies described in [Figure A-4](#).

Figure A-3 AXI-HIF Write Interface Block Diagram

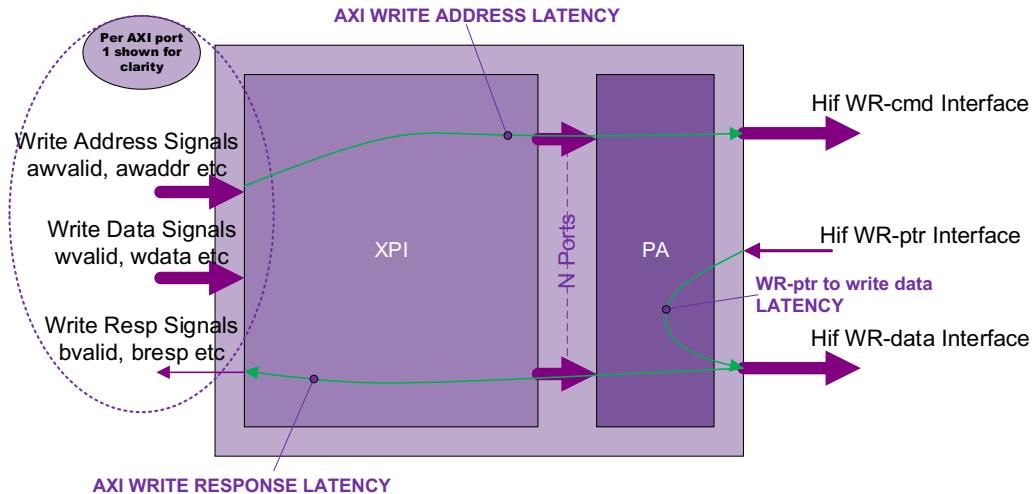


Figure A-4 AXI-HIF Write Latency

Path	From	To	Hardware Parameter Settings					Latency	Comment
			UMCTL2_PA _OPT_TYPE 1= two-cycle arbitration (1 cycle of idle latency) 2 = combinatorial (0 cycle of idle latency)	UMCTL2_A_SYNC_n	MEMC_USE _RMW or MEMC_ECC _SUPPORT	UMCTL2_XPI_USE_WAR	Reg. ECCCFG0. ecc_mode	Note! measurements are best case assuming hif wr has available write credits and write buffer space available	<i>Note! that all the parameter settings are not mutually exclusive; latency accumulates if more than one parameter is enabled.</i> <i>Recommended values are shown in yellow</i>
AXI WRITE ADDRESS LATENCY	Write address phase accepted awvalid =1 && awready =1	Command presented on HIF: hif_cmd_valid = 1	2	1	0	0	3'b000	TMIN = 1 core clk cycle	
			2	1	0	1	3'b000	TMIN + 1 core clk cycle	
			2	1	1	0	3'b000	TMIN + 1 core clk cycle	
			2	1	1	0	3'b100	MAX((TMIN + 1), (MSTR.burst_rdwr /MEMC_F_REQ_RATIO))	When ECC is enabled in software, the additional latency is due to store and forward operation on the write data with the assumption that (DDR_bytes==AXI_bytes).
			2	0	0	0	3'b000	TMIN + 1 ackl cycle + MCTL2_ASYNC_FIFO_N_SYNC_n core cycles	Default MCTL2_ASYNC_FIFO_N_SYNC_n=2
			1	1	0	0	3'b000	TMIN + 1 core clk cycle	
WR_ptr to write data LATENCY	hif_wdata_ptr_valid =1	Write data valid presented to HIF: hif_wdata_valid = 1						1 core clk cycle	
AXI WRITE RESPONSE LATENCY	hif_wdata_valid = 1 && hif_wdata_end = 1 && hif_wdata_stall = 0	bvalid=1	UMCTL2_DATA_CHANNEL_INTERLEAVE_EN				UMCTL2_A_SYNC_n		
			0				1	TMIN = 2 core clk cycle	Last hif beat, if multiple hif beats are required to transfer the number of axi bytes sent
			1				1	TMIN + 1 core clk cycle	
							0	TMIN + 1 core clk cycle + MCTL2_ASYNC_FIFO_N_SYNC_n ackl cycles	

A.3.1.2 AXI-HIF-AXI Reads Interface

The AXI read interface block diagram [Figure A-5](#) shows the paths for the latencies shown in [Figure A-6](#).

Figure A-5 AXI Read Interface Block Diagram

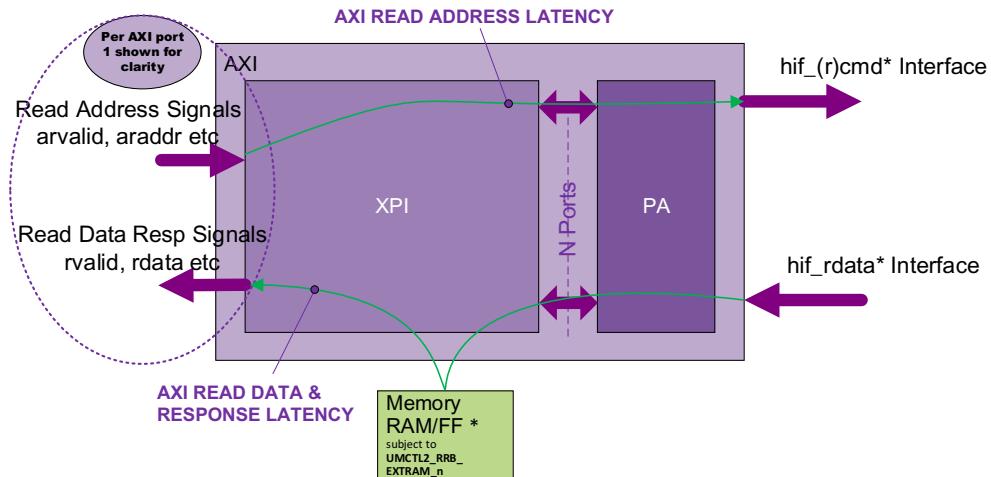


Figure A-6 AXI-HIF Read Latency

Path	From	To	Hardware Parameter Settings						Latency	Comment
AXI Read ADDRESS LATENCY	Read address phase accepted arvalid =1 && arready =1	Command presented on HIF: hif_(r)cmd_valid = 1	UMCTL2_PA_OPT_TYPE	UMCTL2_A_SYNC_n	UMCTL2_XPI_USE_INPUT_RAR	UMCTL2_XPI_USE_RAR			Note! measurements are best case assuming hif rd has available read credits and read buffer space available	Note! that all the parameter settings are not mutually exclusive; latency accumulates if more than one parameter is enabled. Recommended values are shown in yellow
			2	1	0	0				
			2	1	0	1				
			2	1	1	0				
			2	0	0	0				
			1	1	0	0				
AXI READ DATA RESPONSE LATENCY	hif_rdata_valid =1	Read data valid presented to AXI read port: rvalid = 1	UMCTL2_RRB_EXTRAM_n	UMCTL2_A_SYNC_n	UMCTL2_XPI_USE_RPR	UMCTL2_XPI_USE_RDR	UMCTL2_RRB_EXTRAM_n	UMCTL2_RRB_THRESHOLD_PPL_n		
			0	1	0	0	N/A	0	TMIN = 2 core clk cycles	
			0	1	0	1	N/A	0	TMIN + 1 core clk cycle	
			0	1	1	0	N/A	0	TMIN + 1 core clk cycle	
			0	0	0	0	N/A	0	TMIN + (1+UMCTL2_ASYNC_FIFO_N_SYNC_n) ackl cycles	
			1	1	0	0	0	0	TMIN + 1 core clk cycle	
			1	1	0	0	1	0	TMIN + 2 core clk cycle	
			0	1	0	0	N/A	1	TMIN + 1 core clk cycle	
			UMCTL2_RRB_THRESHOLD_EN_n			UMCTL2_RRB_THRESHOLD_PPL_n				
			1			1			TMIN + 1 core clk cycle	

A.3.2 HIF - DFI - HIF Latency With Controller Idle

The simplified HIF-DFI-HIF interface block diagram (Figure A-7) shows the various elements of the HIF interface write and read latency which are under the control of the DDRCTL.

Figure A-7 HIF-DFI-HIF Interface Simplified Block Diagram

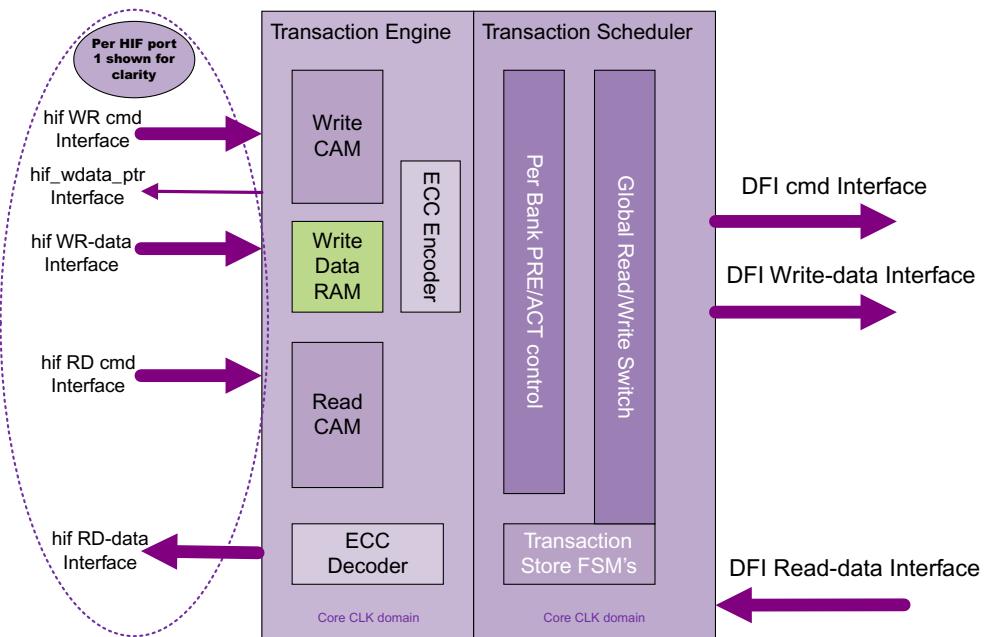


Figure A-8 HIF-DFI Write Latency Table

Path	From	To	Hardware Parameter Settings				Latency	Comment
			MEMC_OPT_TIMING	MEMC_REG_DFI_OUT	MEMC_INLINE_ECC	Reg._wrdata_delay	core_clk	Note that all the parameter settings are not mutually exclusive; latency accumulates if more than one parameter is enabled.
								Note measurements are best case assuming page is open and this command is the scheduled transaction <i>Recommended values are shown in yellow</i>
Address latency through - DDRC	Last write data word on HIF: hif_wdata_end=1 && hif_wdata_valid=1	First DFI cycle of Write command	0	0	0	>2	TMIN = 3 core clk cycles	Write column command encoding on DFI is applicable to DDRx. For other protocols, refer to the DFI specification.
			0	0	0	2	TMIN + 1 core clk cycles	Further command latency (for both write and read) can be incurred if DDR4 CAL mode is enabled. In that case, DFITMG1.dfi_t_cmd_lat specifies the number of DFI PHY clocks between the dfi_cs signal is asserted and when the associated address/ command is driven.
			0	0	0	1	TMIN + 2 core clk cycles	
			0	0	0	0	TMIN + 3 core clk cycles	
			0	1	0	>2	TMIN + 1 core clk cycles	
			0	1	0	2	TMIN + 2 core clk cycles	
			0	1	0	1	TMIN + 3 core clk cycles	
			0	1	0	0	TMIN + 4 core clk cycles	
			1	N/A	N/A	X	+ 1 core clk cycles	
			N/A	N/A	1	X	+ 1 core clk cycles	
DFI write data enable latency	Write column command on DFI: dfi_cs=0 dfi_cas_n=0 dfi_ras_n=1	dfi_wrdata_en=1					DFITMG0.dfi_tphy_wrlat	Determined by the PHY
DFI write data latency	dfi_wrdata_en=1	Write data sent on DFI interface					DFITMG0.dfi_tphy_wrdata	Determined by the PHY

Figure A-9 HIF-DFI Read Latency Table

Path	From	To	Hardware Parameter Settings		Latency	Comment
			MEMC _REG _DFI _OUT	MEMC _INLINE _ECC	core_clk	Note that all the parameter settings are not mutually exclusive; latency accumulates if more than one parameter is enabled.
Address latency through - DDRC	Command presented on HIF: hif_(r)cmd_valid = 1	First DFI cycle of Read command	0	0	TMIN = 4 core clk cycles	Recommended values are shown in yellow
			1	0	TMIN + 1 core clk cycles	
			1	1	TMIN + 2 core clk cycles	
DFI read data enable latency	Read column command on DFI	dfi_rddata_en=1			DFITMG0.dfi_t_rddata_en The unit of DFITMG0.dfi_t_rddata_en is DRAM data clock cycles.	Determined by the PHY
Read Data latency through - DDRC	dfi_rddata_valid	hif_rdata_valid = 1			TMIN = 1 core clk cycles	Note that all the parameter/register settings are not mutually exclusive; latency accumulates if more than one is enabled.
			MEMC_REG_DFI_IN_RD_DATA==1		+ 1 core clk cycle	
			In MEMC_FREQ_RATIO=2 cases, where upper and lower half of dfi_rddata_valid are not equal (dfi_rddata_valid_w0!=dfi_rddata_valid_w1)		+ 1 core clk cycle	
			In DDR4 usage and MEMC_FREQ_RATIO=4		+ 1 core clk cycle	
			DDRCTL_RETRY_MAX_ADD_RD_LAT	Reg.RETRYTMG1.retry_add_rd_lat_en		
			>0	1	+ Reg.RETRYTMG1.retry_add_rd_lat+1	
			ECC_SUPPORT > 0		+ 1 core clk cycle	
			Reg.CRCPARCTL1.rd_crc_enable			
			1		+ 1 core clk cycle	
			((DDRCTL_RSD_PIPELINE_EN==0) & ((Reg.RETRYCTL0.rd_ue_retry_enable==1 Reg.CRCPARCTL1.rd_crc_enable==1)))		+ 1 core clk cycle	
			((DDRCTL_RSD_PIPELINE_EN==1) & (Reg.ECCCFG0.ecc_mode==5))		+ 1 core clk cycle	
			((DDRCTL_RSD_PIPELINE_EN==1) & (Reg.ECCCFG0.ecc_mode==5) & (Reg.RETRYCTL0.rd_ue_retry_enable==1))		+ 1 core clk cycle	
			((DDRCTL_RSD_PIPELINE_EN==1) & (Reg.ECCCFG0.ecc_mode==5) & ((Reg.CRCPARCTL1.rd_crc_enable==1 Reg.RETRYCTL0.rd_ue_retry_enable==1)))		+ 1 core clk cycle	
			(MEMC_LINK_ECC==1) & (Reg.LNKECCCTL0.rd_link_ecc_enable==1)		+ 2 core clk cycle	
			The following are examples of settings with total latency			
			MEMC_FREQ_RATIO=2 & MEMC_DDRS=1 & MEMC_ECC_SUPPORT=0 & DDRCTL_RSD_PIPELINE_EN=0 & DDRCTL_RD_UE_RETRY=0		Total Latency = 1 core clk cycle	
			REGB_DDRC_CH0.CRCPARCTL1.rd_crc_enable=0			
			MEMC_FREQ_RATIO=4 & MEMC_DDRS=1 & MEMC_ECC_SUPPORT=3 & DDRCTL_RSD_PIPELINE_EN=1 & DDRCTL_RD_UE_RETRY=1		Total Latency = 4 core clk cycles	
			REGB_DDRC_CH0.ECCCFG0.ecc_mode=4 & RETRYTMG1.retry_add_rd_lat_en=0 & RETRYCTL0.rd_ue_retry_enable=1 & REGB_DDRC_CH0.CRCPARCTL1.rd_crc_enable=1			
			MEMC_FREQ_RATIO=4 & MEMC_DDRS=1 & MEMC_ECC_SUPPORT=3 & DDRCTL_RSD_PIPELINE_EN=1 & DDRCTL_RD_UE_RETRY=1		Total Latency = 5 core clk cycles	
			REGB_DDRC_CH0.ECCCFG0.ecc_mode=5 & RETRYTMG1.retry_add_rd_lat_en=0 & RETRYCTL0.rd_ue_retry_enable=1 & REGB_DDRC_CH0.CRCPARCTL1.rd_crc_enable=1			

B

Data Lane Mapping Examples



Note All the data lane mapping examples in this appendix are applicable for both read and write operations.

In figures:

1:4 Frequency Ratio means `MEMC_FREQ_RATIO=4`

1:2 Frequency Ratio means `MEMC_FREQ_RATIO=2`

BL16 means register field `MSTR0.burst_rdwr=5'b01000`

Figure B-1 32 Bit Memory Interface, BL16, FBW, 1:4 Frequency Ratio, 16 Byte AXI Data Bus Width and AXI Addresses are Memory-aligned

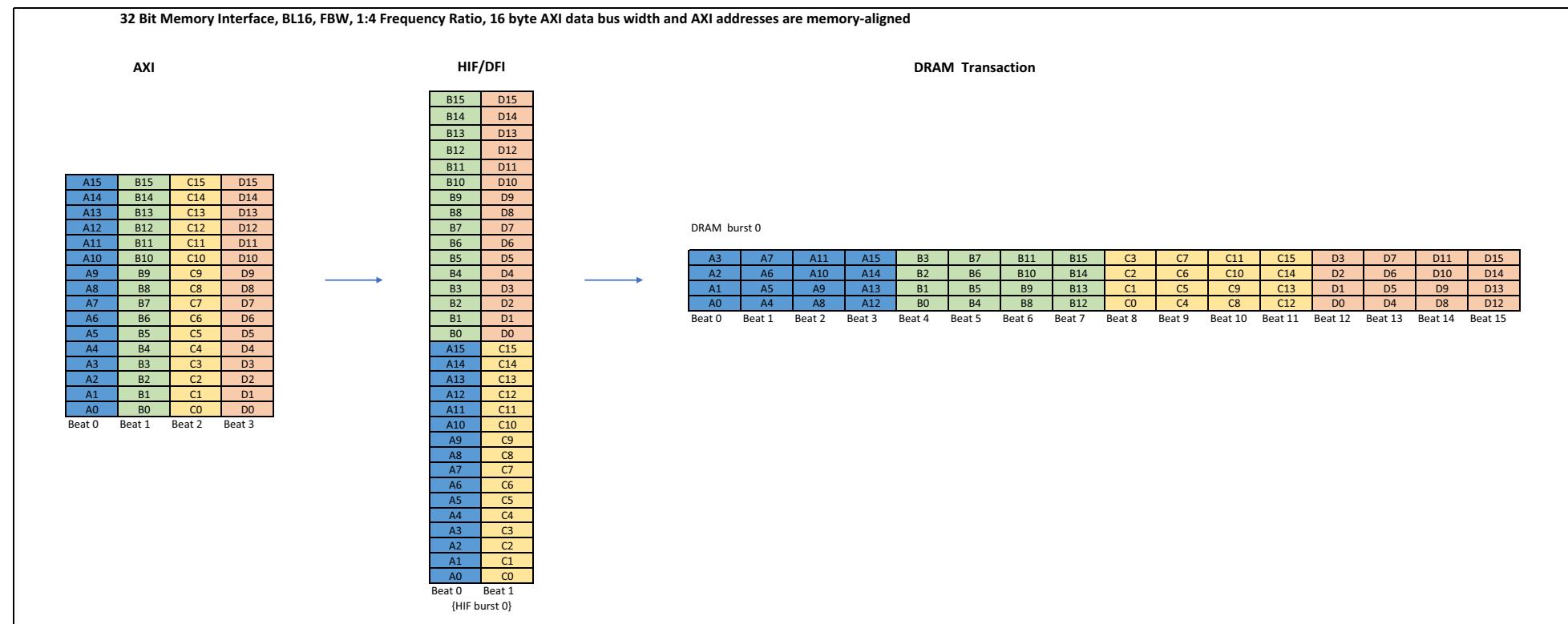


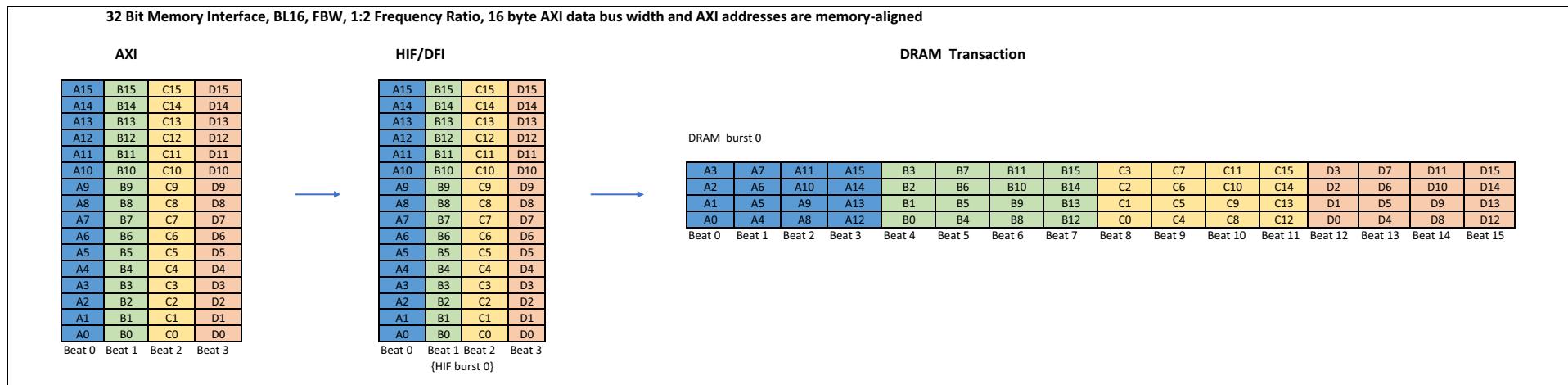
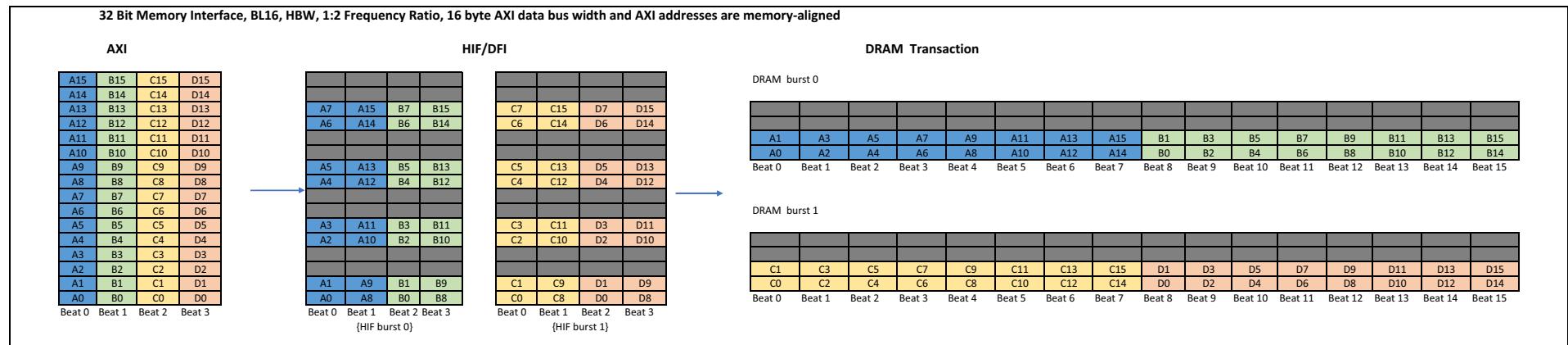
Figure B-2 32 Bit Memory Interface, BL16, FBW, 1:2 Frequency Ratio, 16 Byte AXI Data Bus Width and AXI Addresses are Memory-aligned**Figure B-3 32 Bit Memory Interface, BL16, HBW, 1:2 Frequency Ratio, 16 Byte AXI Data Bus Width and AXI Addresses are Memory-aligned**

Figure B-4 32 Bit Memory Interface, BL16, FBW, 1:2 Frequency Ratio, 16 byte HIF data bus width and HIF addresses are memory-aligned

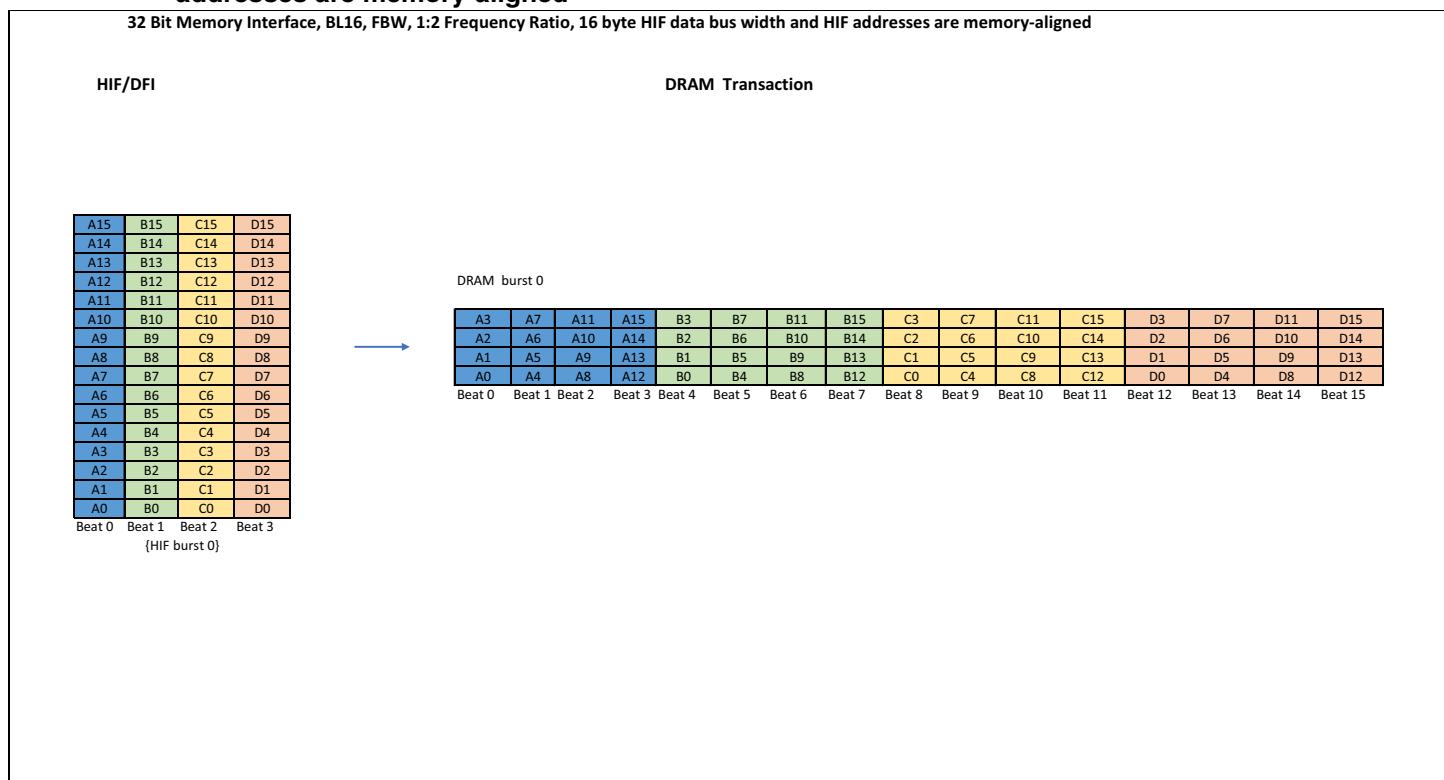


Figure B-5 32 Bit Memory Interface, BL16, FBW, 1:4 Frequency Ratio, 32 byte HIF data bus width and HIF addresses are memory-aligned

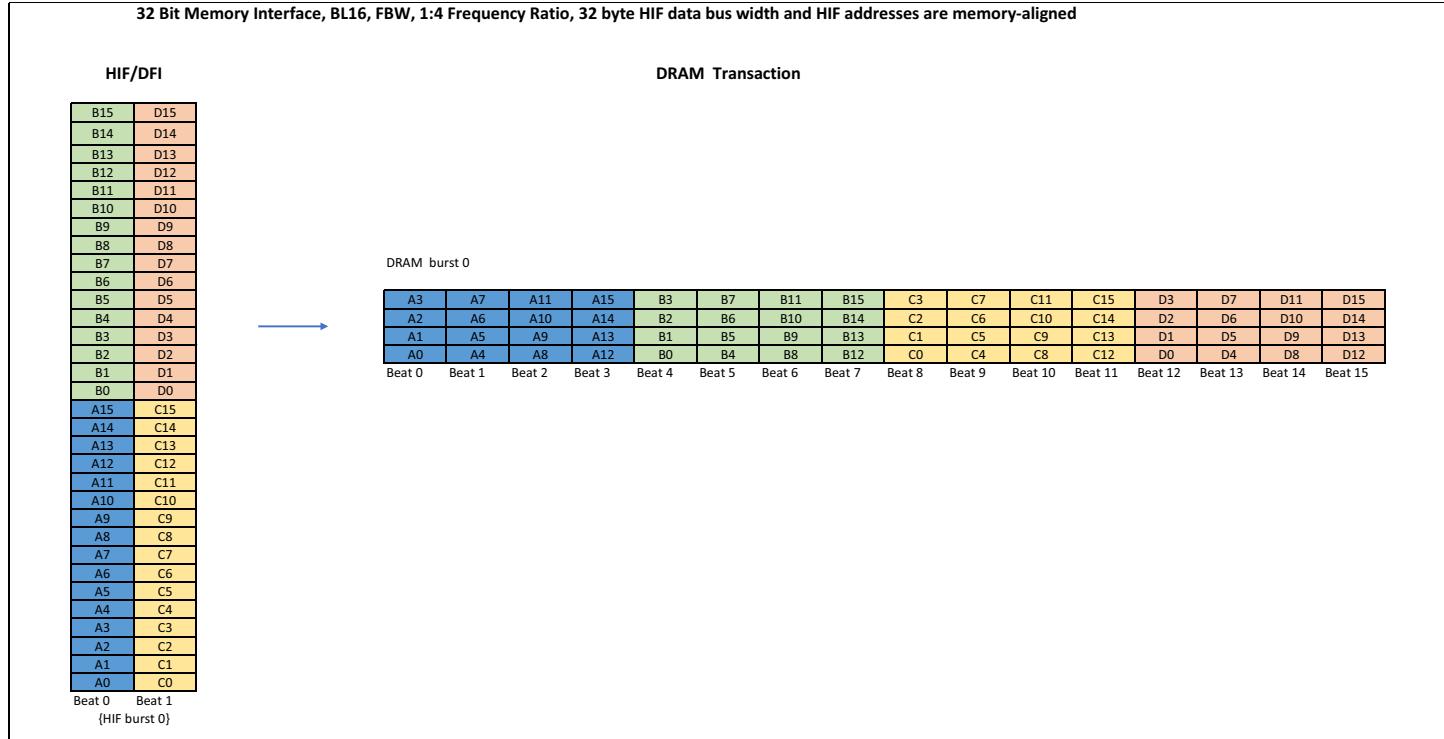


Figure B-6 32 Bit Memory Interface, BL16, HBW, 1:2 Frequency Ratio, 16 byte HIF data bus width and HIF addresses are memory-aligned

32 Bit Memory Interface, BL16, HBW, 1:2 Frequency Ratio, 16 byte HIF data bus width and HIF addresses are memory-aligned

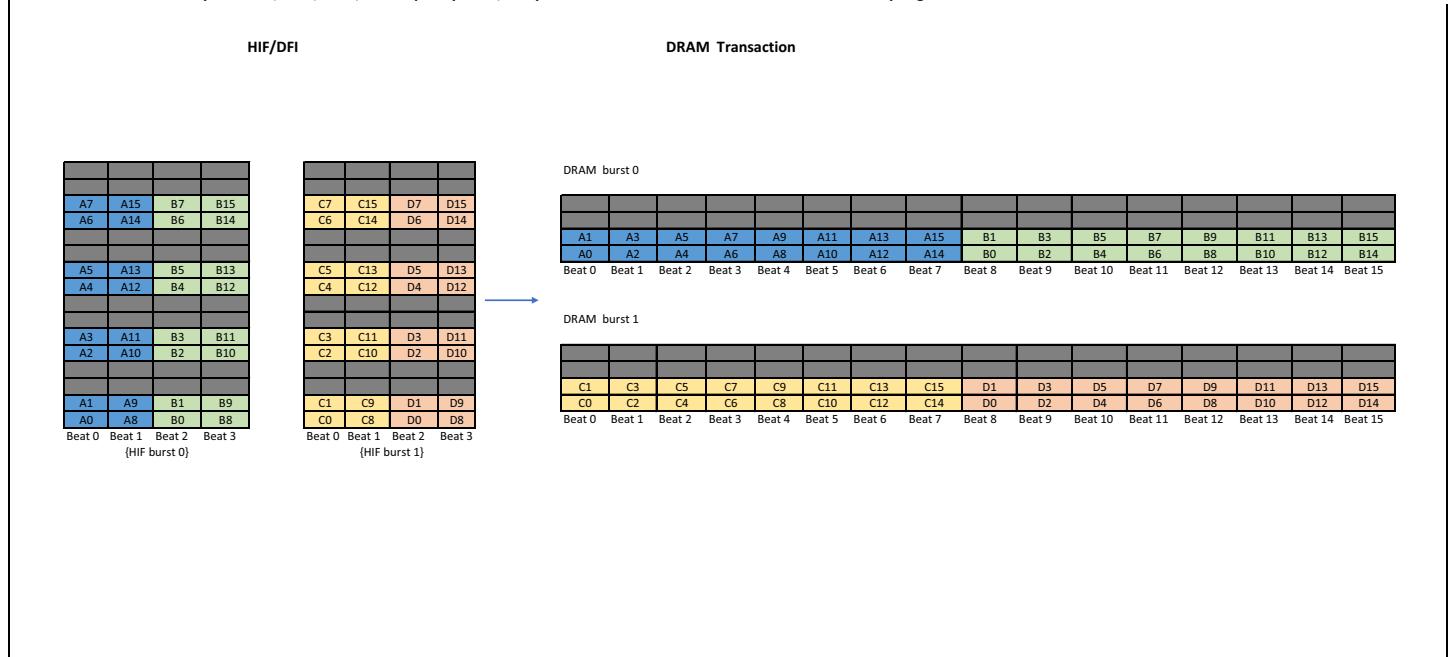


Figure B-7 32 Bit Memory Interface, BL16, HBW, 1:4 Frequency Ratio, 32 byte HIF data bus width and HIF addresses are memory-aligned

32 Bit Memory Interface, BL16, HBW, 1:4 Frequency Ratio, 32 byte HIF data bus width and HIF addresses are memory-aligned

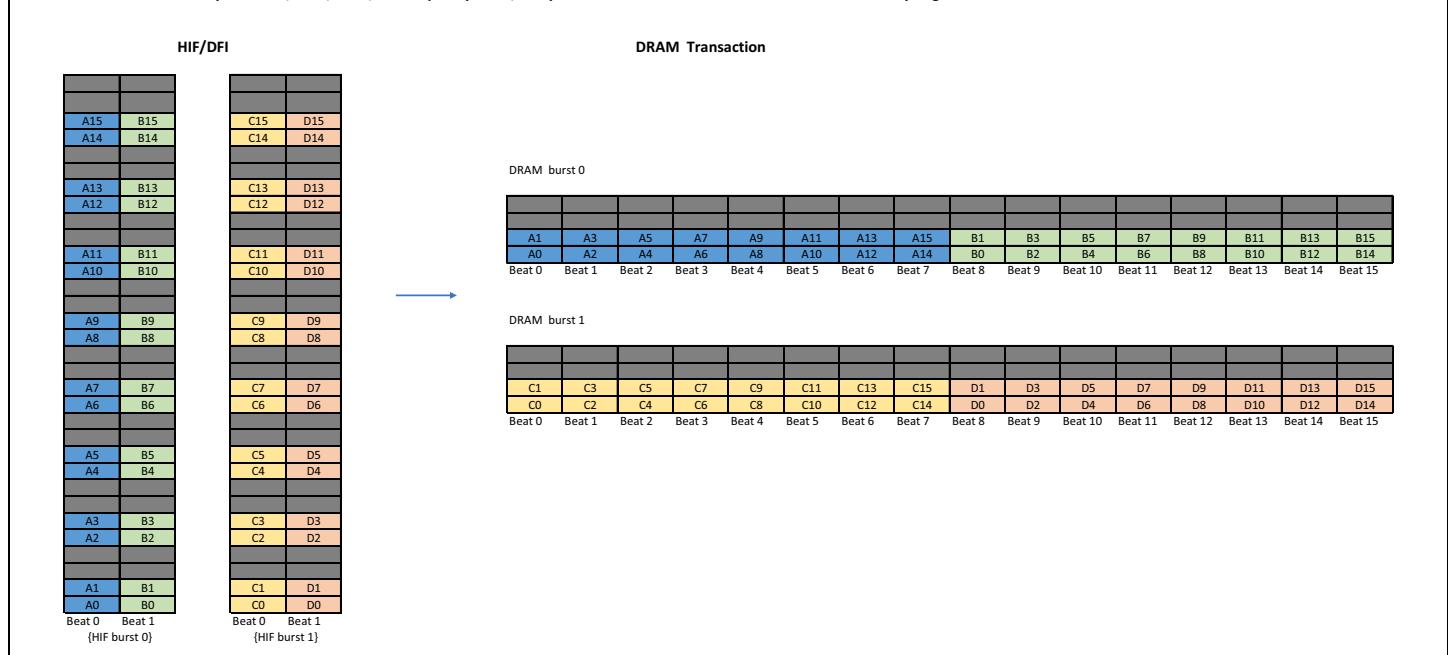


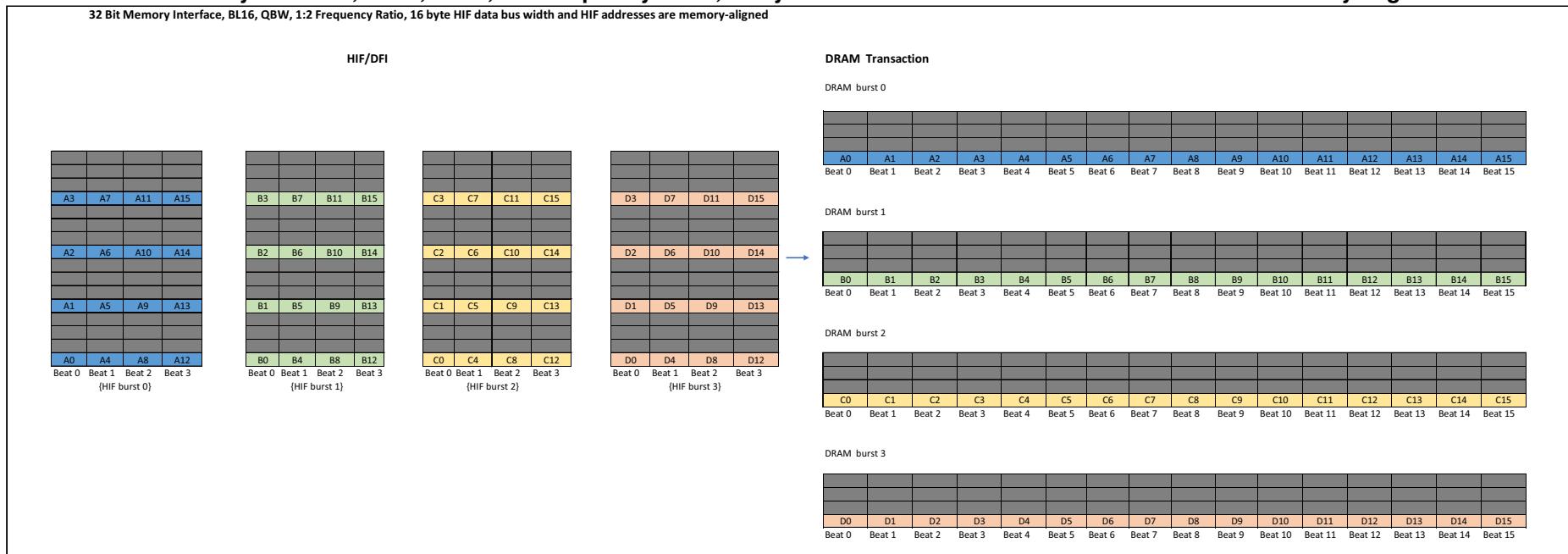
Figure B-8 32 Bit Memory Interface, BL16, QBW, 1:2 Frequency Ratio, 16 byte HIF data bus width and HIF addresses are memory-aligned

Figure B-9 32 Bit Memory Interface, BL16, QBW, 1:4 Frequency Ratio, 32 byte HIF data bus width and HIF addresses are memory-aligned

32 Bit Memory Interface, BL16, QBW, 1:4 Frequency Ratio, 32 byte HIF data bus width and HIF addresses are memory-aligned

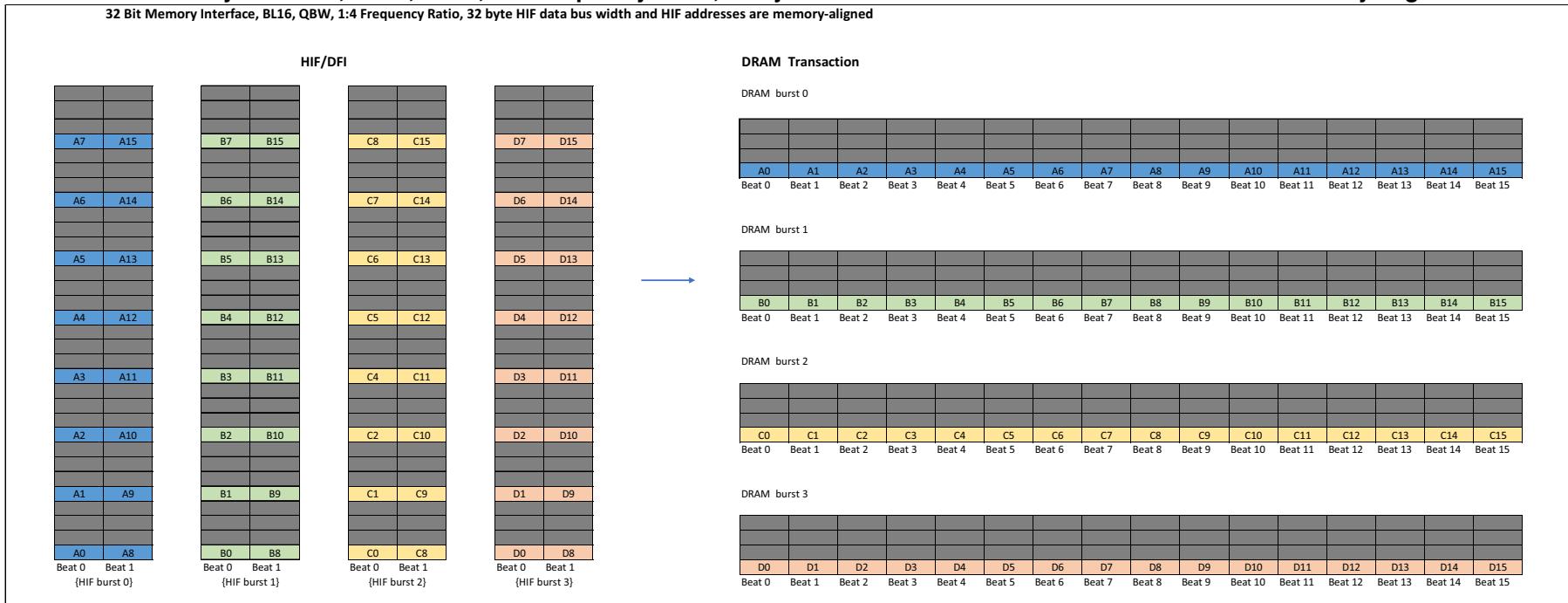


Figure B-10 64 Bit Memory Interface, BL8, FBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and addresses memory-aligned

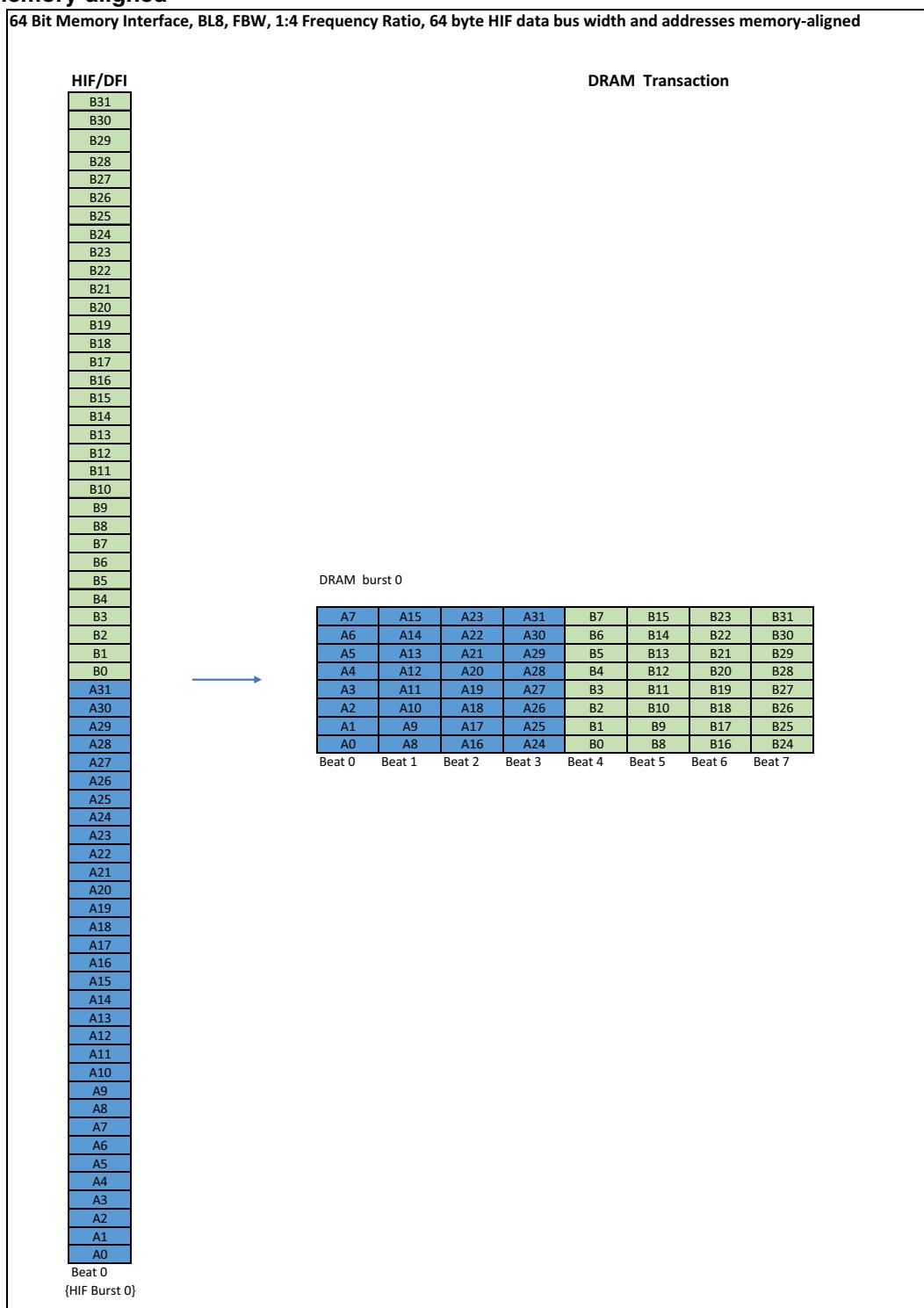


Figure B-11 64 Bit Memory Interface, BL8, HBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and addresses are memory-aligned

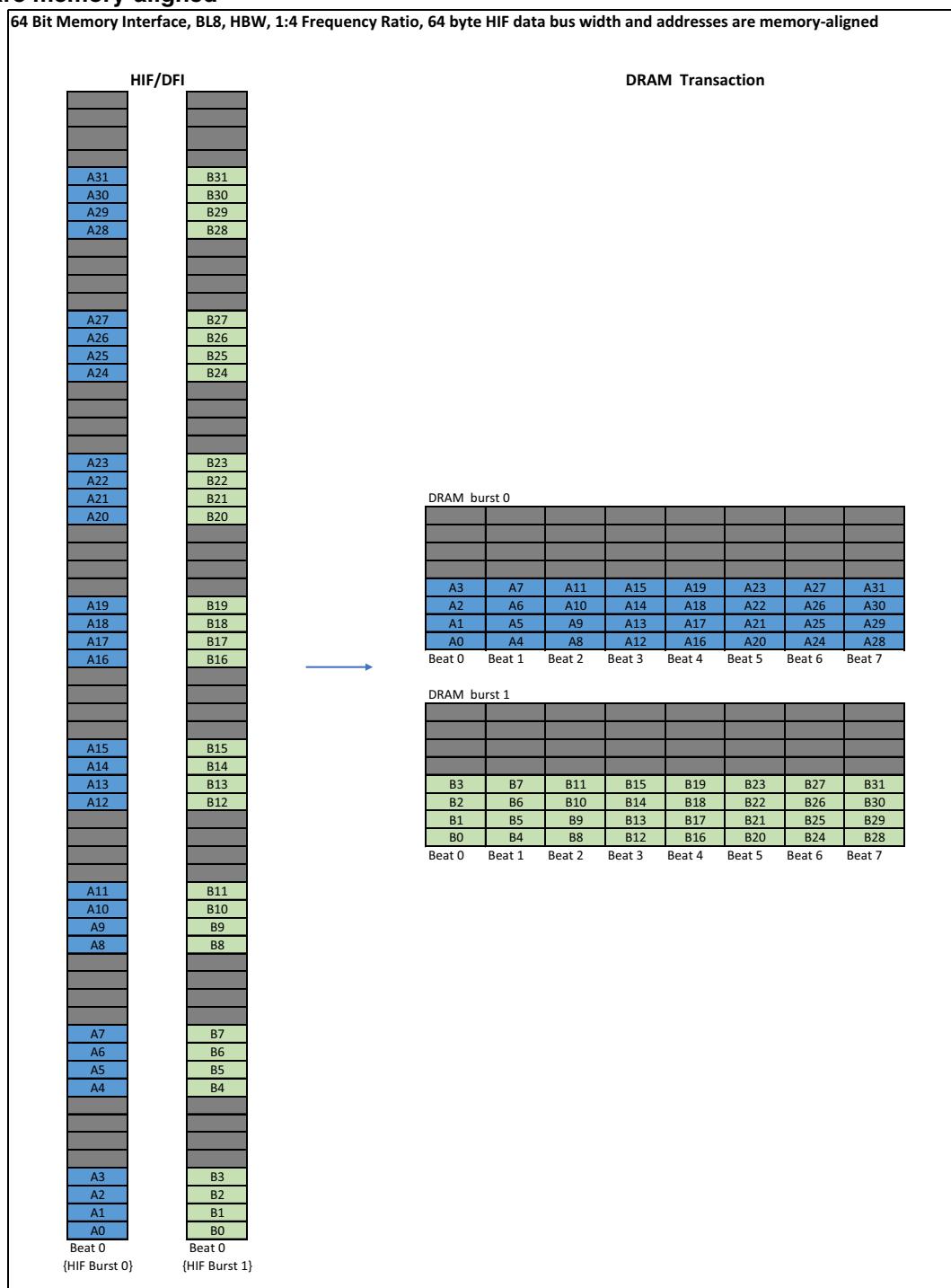


Figure B-12 64 Bit Memory Interface, BL8, QBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and HIF addresses are memory-aligned

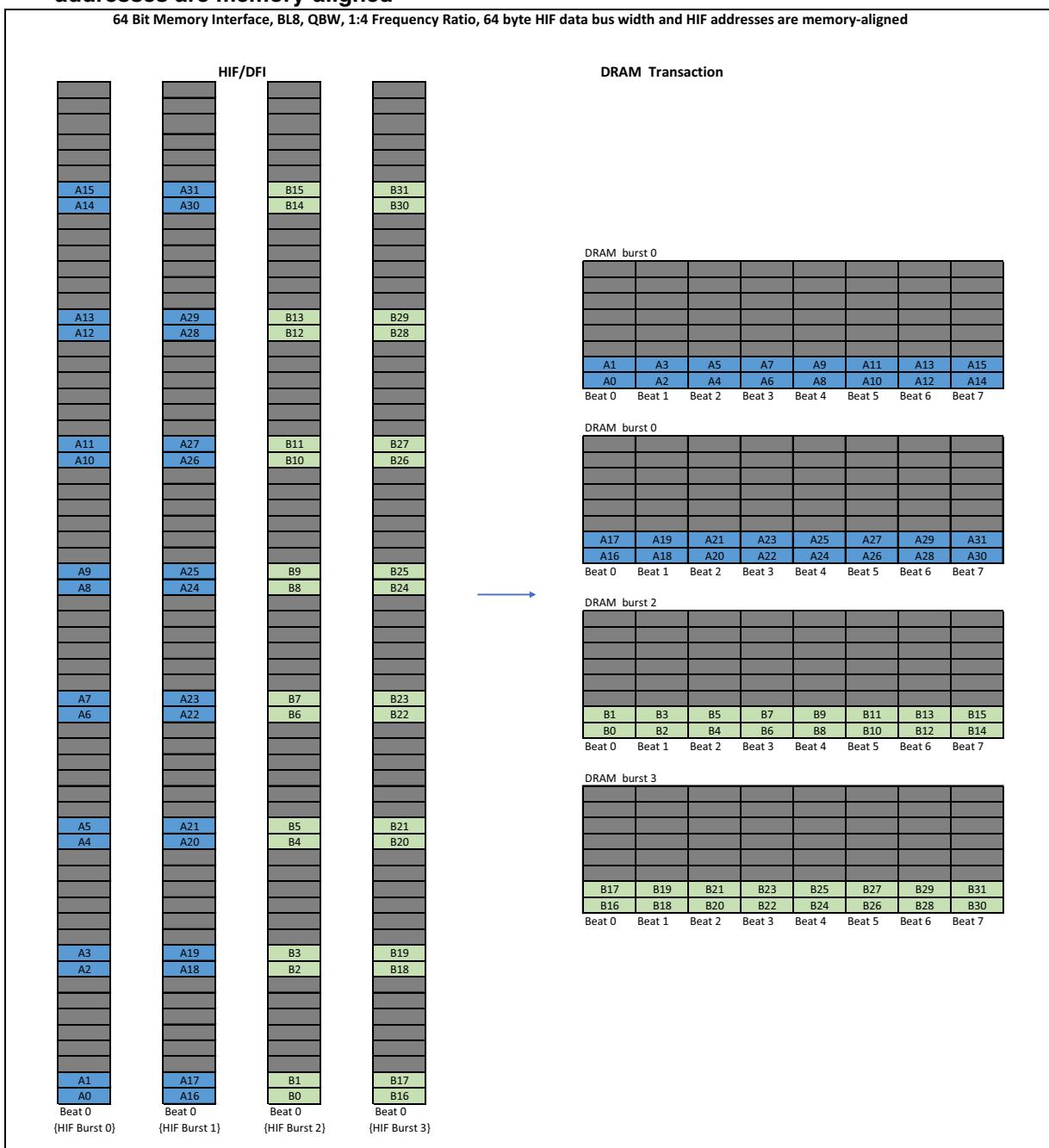


Figure B-13 64 Bit Memory Interface, BL16, FBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and HIF addresses are memory-aligned

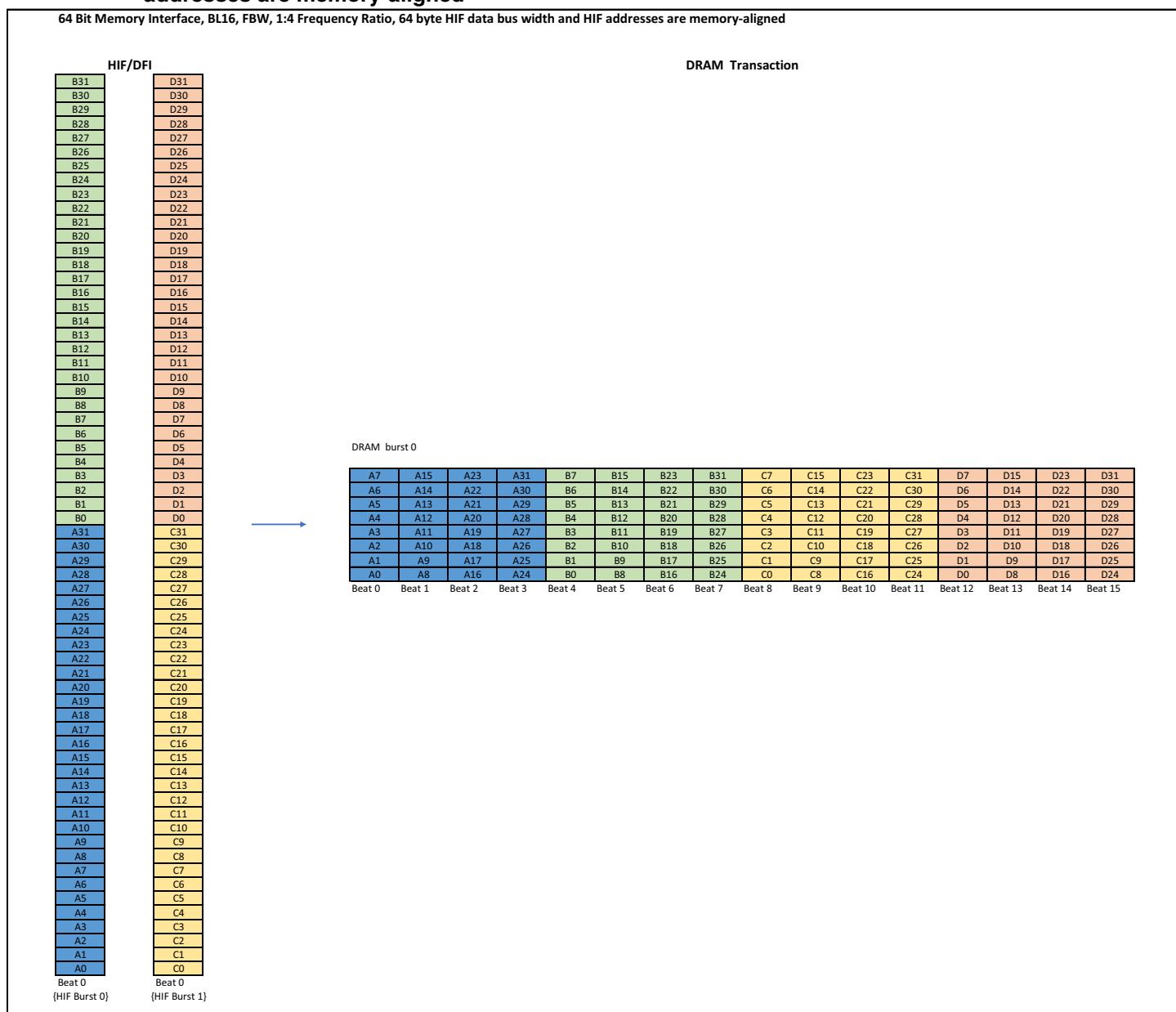


Figure B-14 64 Bit Memory Interface, BL16, HBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and HIF addresses are memory-aligned

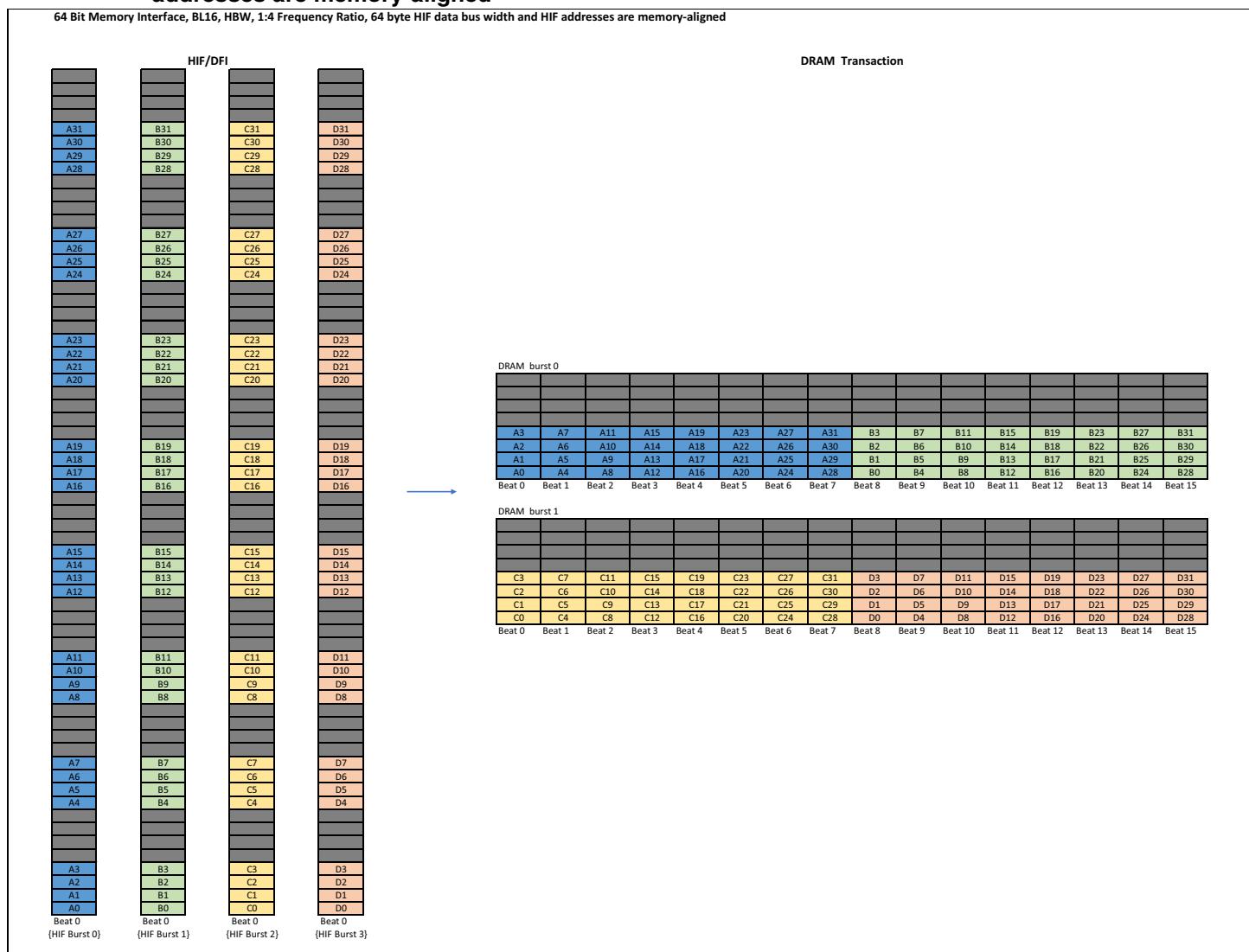


Figure B-15 64 Bit Memory Interface, BL16, QBW, 1:4 Frequency Ratio, 64 byte HIF data bus width and HIF addresses are memory-aligned

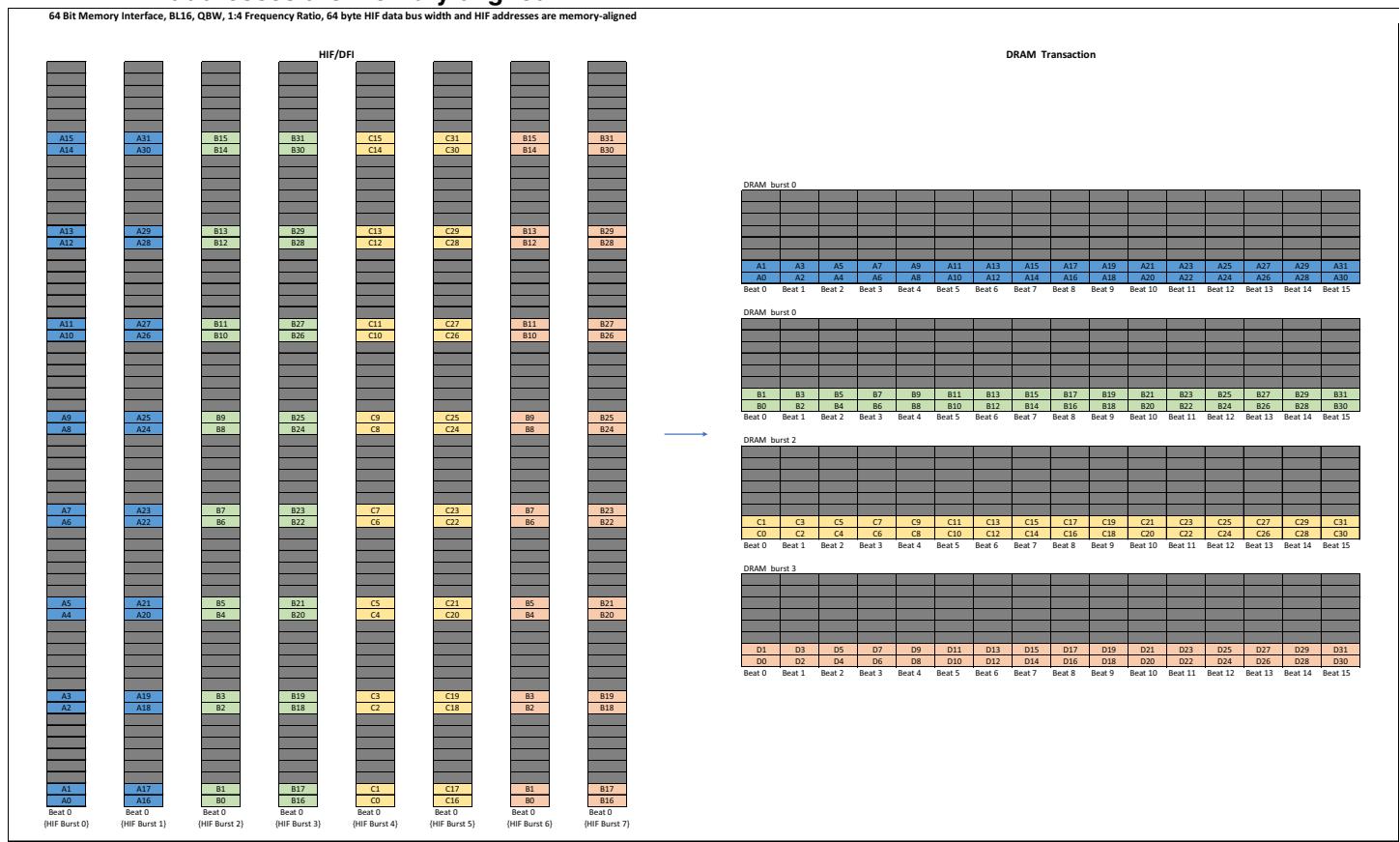


Figure B-16 64 Bit Memory Interface, BL16, FBW, 1:4 Frequency Ratio, 64 byte AXI data bus width and AXI addresses are memory-aligned

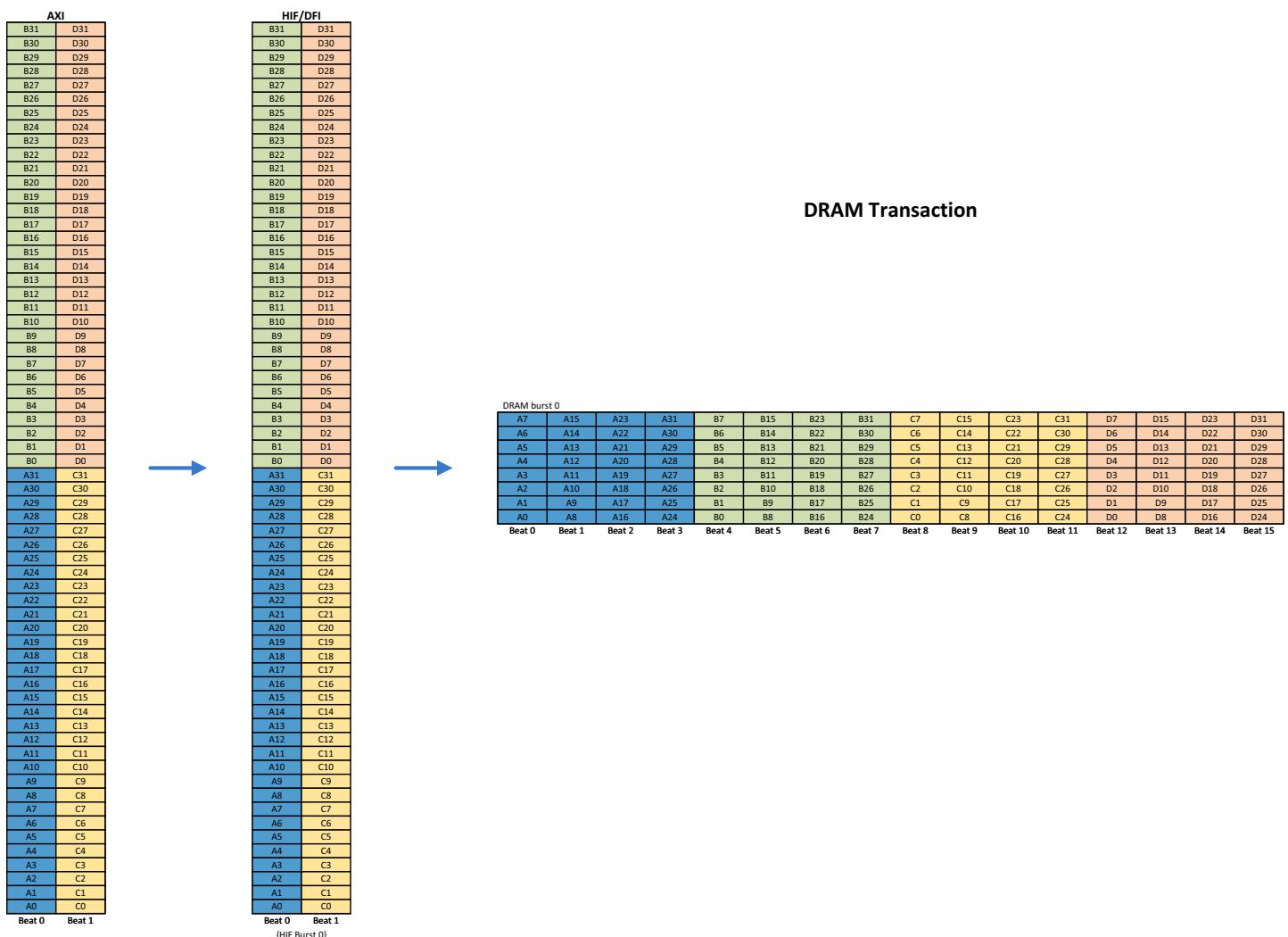


Figure B-17 64 Bit Memory Interface, BL16, HBW, 1:4 Frequency Ratio, 64 byte AXI data bus width and AXI addresses are memory-aligned

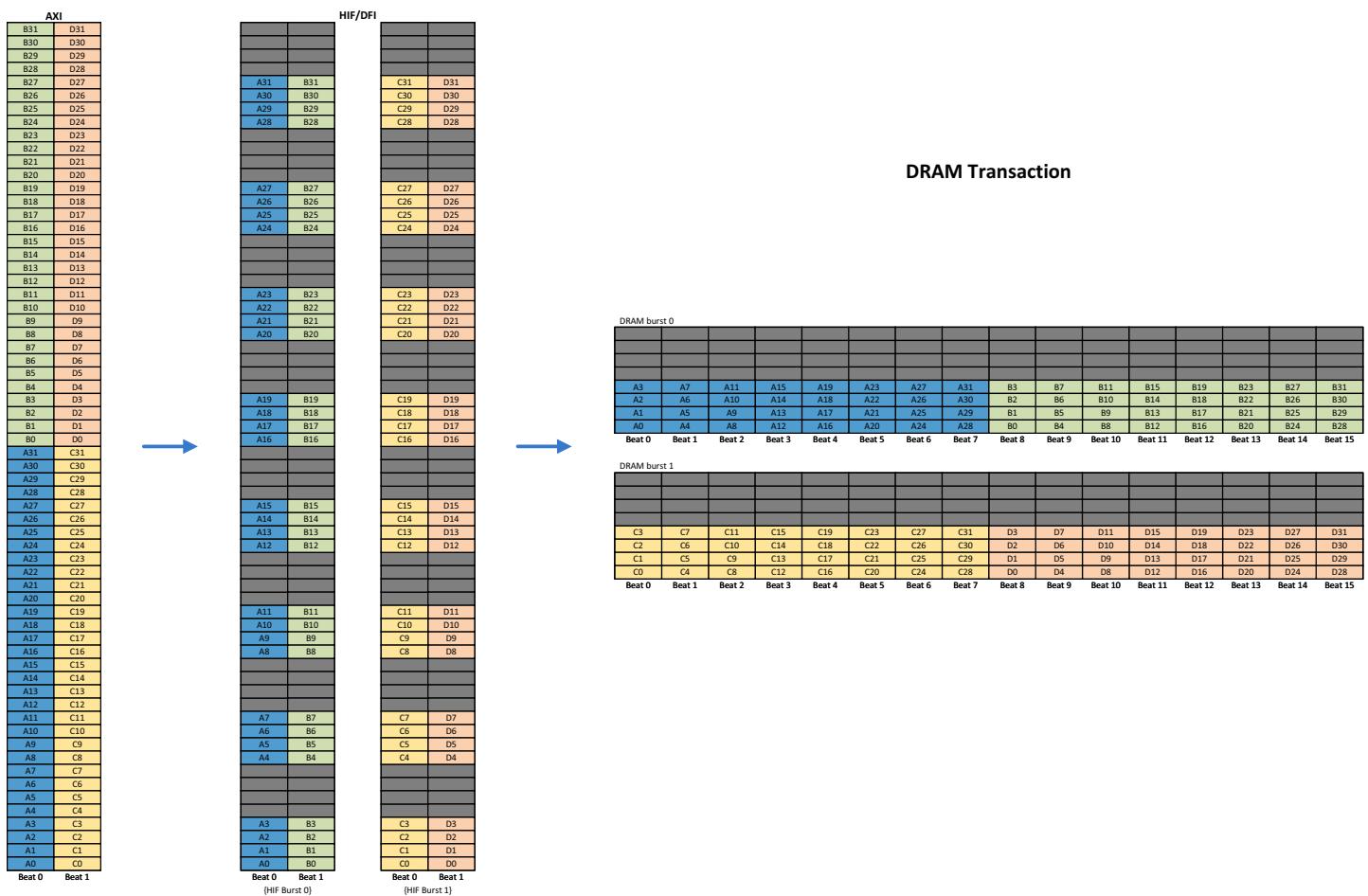
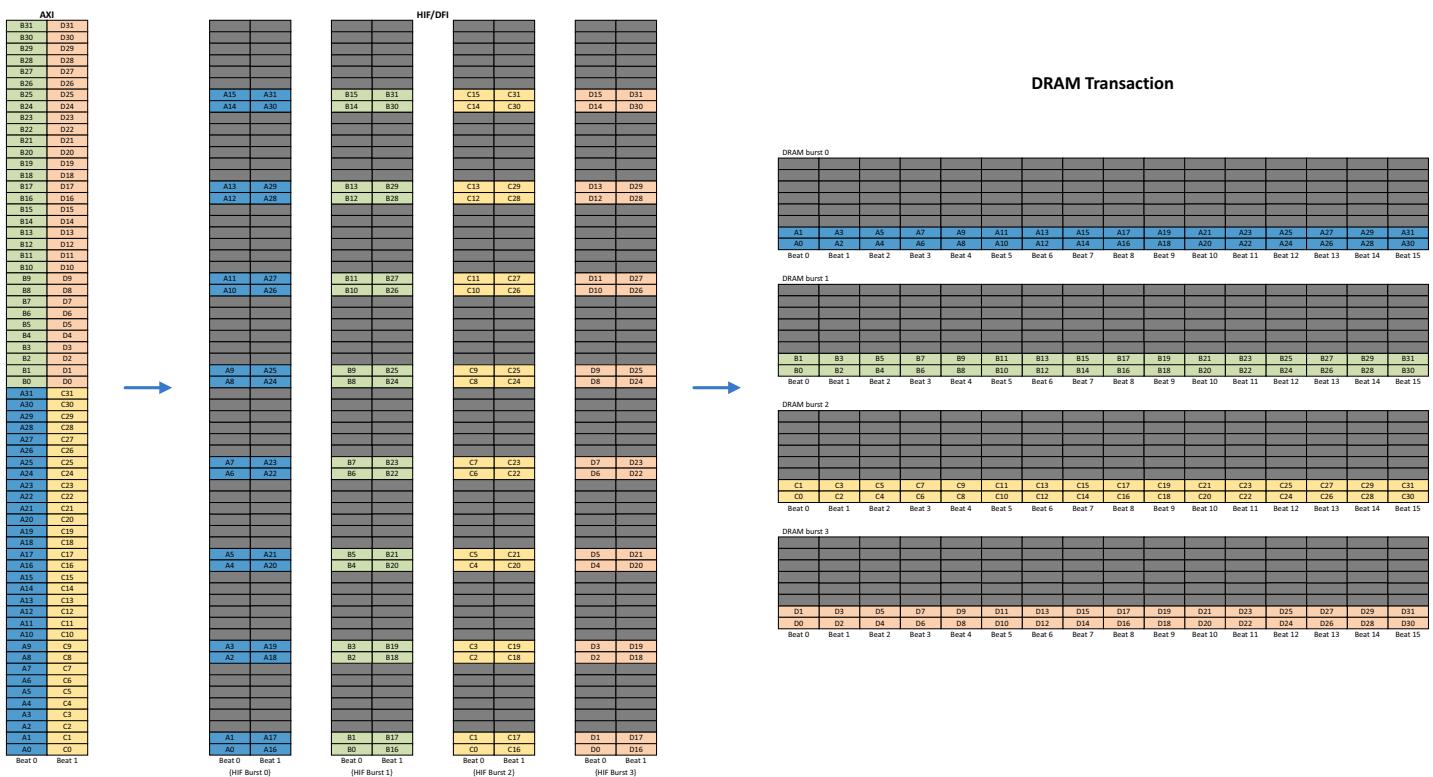


Figure B-18 64 Bit Memory Interface, BL16, QBW, 1:4 Frequency Ratio, 64 byte AXI data bus width and AXI addresses are memory-aligned



C

Password Protected Features

This chapter describes the features supported by the controller that can be enabled when you have the corresponding license.

Table C-1 Features Protected by Password

Feature	Hardware Parameter Enabling the Feature	License Required for the Feature
ECC Supported	MEMC_ECC_SUPPORT	DWC-AP-LPDDR54-CONTROLLER DWC-LPDDR54-CONTROLLER-AFP DWC-AP-LPDDR5X-CONTROLLER DWC-LPDDR5X-CONTROLLER-AFP
Enable Inline ECC	MEMC_INLINE_ECC	DWC-AP-LPDDR54-CONTROLLER DWC-LPDDR54-CONTROLLER-AFP DWC-AP-LPDDR5X-CONTROLLER DWC-LPDDR5X-CONTROLLER-AFP
Enable Link ECC	MEMC_LINK_ECC	DWC-AP-LPDDR54-CONTROLLER DWC-LPDDR54-CONTROLLER-AFP DWC-AP-LPDDR5X-CONTROLLER DWC-LPDDR5X-CONTROLLER-AFP
Enable On-Chip Parity feature	UMCTL2_OCPAR_EN	DWC-AP-LPDDR54-CONTROLLER DWC-LPDDR54-CONTROLLER-AFP DWC-AP-LPDDR5X-CONTROLLER DWC-LPDDR5X-CONTROLLER-AFP
Enable On-Chip ECC feature	UMCTL2_OCECC_EN	DWC-AP-LPDDR54-CONTROLLER DWC-AP-LPDDR5X-CONTROLLER
Enable On-Chip Command/Address Protection feature	UMCTL2_OCCAP_EN	DWC-AP-LPDDR54-CONTROLLER DWC-AP-LPDDR5X-CONTROLLER
Enable On-Chip SRAM Address Protection	DDRCTL_OCSAP_EN	DWC-AP-LPDDR54-CONTROLLER DWC-AP-LPDDR5X-CONTROLLER

Feature	Hardware Parameter Enabling the Feature	License Required for the Feature
Enable Register Parity Feature	UMCTL2_REGPAR_EN	DWC-AP-LPDDR54-CONTROLLER DWC-AP-LPDDR5X-CONTROLLER
Enable Hardware Fast Frequency Change	UMCTL2_HWFFC_EN	DWC-LPDDR54-CONTROLLER-AFP DWC-LPDDR5X-CONTROLLER-AFP



- If you want to use these features, contact [Synopsys Customer Support](#).
- These features have dependencies on other hardware parameters which must be also enabled. Check the databook and cC GUI.

D

Prime Profiles

A prime profile is a means which Synopsys can use to deliver predefined configurations of the IP. These predefined configurations are pre-verified to tape-out quality level. Prime profiles will be applied for new Synopsys products and new features. These are not applicable to previously released products.

A prime profile can be selected through coreConsultant. When a DDR controller product with prime profiles is selected, coreConsultant will be triggered to enter a special prime profile mode (once entered you cannot exit from this mode). A window will pop-up prompting to select the required prime profile configuration from a drop-down menu.

In order to be able to make changes in the prime profile hardware configuration you must check the "Analysis mode" checkbox. If the Analysis mode is enabled, coreConsultant can only generate a coreConsultant batch script, and will not complete SpecifyConfiguration. The batch script will be written to the following location in your workspace (example):

```
<workspace>/export/prime_profile_ProfileA_analysis_script.tcl
```

This batch script must be sent to Synopsys via a Solvnet case. After validation (by Synopsys R&D) of the batch file a new profile will be sent to you along with the instructions on how to install it locally.

D.1 Supported Prime Profiles

For the list of supported prime profiles, see the following document:

`<workspace>/doc/html/pve_doc/pp_page.html`

D.2 Selecting Prime Profiles

Most supported LPDDR product Prime Profiles are unlocked, that is, after selecting a Prime Profile, the hardware parameters can be overridden by sourcing a new configuration file (for same LPDDR product) or updating value for the specific parameter.

Follow the sample instructions to select a prime profile:

1. Use coreConsultant to create a workspace.
2. Start the coreConsultant GUI:

```
% coreConsultant &
```

3. Setup the prime profile trigger:

```
coreConsultant> set_configuration_parameter DDRCTL_PRODUCT_NAME 14
```

4. Select the profile (use drop-down menu, or enter the following command):

```
coreConsultant> set_prime_profile -tag <PRIME_PROFILE_NAME>
```

5. Source target hardware configuration file (enter the following command):

```
coreConsultant> source <target_hardware_configuration_file>
```

6. Update value for specific hardware parameters (use check box/list, or enter the following command):

```
coreConsultant> set_configuration_parameter <target_hardware_parameter> <value>
```

7. Write out the RTL:

```
coreConsultant> autocomplete_activity SpecifyConfiguration
```

The RTL written into the workspace will now match the prime profile with the name <PRIME_PROFILE_NAME>.