



DesignWare Cores LPDDR5/4/4X PHY Training Firmware Application Note

**DWC LPDDR54 PHY
Firmware Version: C-2021.10**

Copyright Notice and Proprietary Information Notice

© 2021 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

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 <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

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

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Contents.....	3
Revision History.....	6
Preface.....	8
1 Introduction	11
1.1 Purpose	11
1.2 Firmware overview	11
1.3 PHY Configurations.....	12
1.4 Supported Training Features.....	13
2 Hardware Training Architecture	15
3 Loading, Configuring and Running the Firmware	17
3.1 Overview	17
3.2 Loading the Training Firmware	18
3.3 Configuring the firmware via the message block.....	19
3.3.1 Initial Device State	19
3.3.1.1 Illegal Configurations	19
3.4 Running the firmware	20
3.4.1 Mailbox facility for firmware	20
3.4.2 Decoding Messages	23
3.4.2.1 Major Messages.....	23
3.4.2.2 Controlling the Number of Messages.....	25
3.5 Reading the training results via the message block	26
4 Training Steps.....	27
4.1 Overview	27
5 Training step details	31
5.1 Device Initialization (DevInit).....	32
5.2 Fine Write Leveling.....	33
5.3 Coarse Write Leveling for LPDDR4	36
5.4 Coarse Write Leveling for LPDDR5	37
5.4.1 Write Level Training configuration options (WrLvlTrainOpt – LPDDR5 only).....	37
5.4.1.1 LP5 Coarse Write Leveling step	38
5.4.1.2 WCK ODT Settings.....	38
5.5 Read Gate Training	38
5.6 TxDQ Training.....	41
5.7 Command Address Training.....	45
5.7.1 CA Training configuration options (CATrainOpt)	46
5.7.1.1 Optimize CA VREF	47
5.7.1.2 Delayed clock	47
5.7.1.3 ACTxDly step size	48

5.7.2	CS training (LPDDR5 only)	48
5.7.2.1	Reducing CS training error	48
5.7.3	CA/CS training implementation summary	49
5.7.4	Example debug logs output.....	50
5.7.5	CA Training Termination Settings	50
5.7.5.1	LPDDR5 MR17 settings	50
5.7.5.2	LPDDR4X MR22 settings.....	51
5.7.5.3	LPDDR4 MR22 settings	51
5.8	DRAM DCA Training (LPDDR5 only).....	51
5.9	PHY DCA Training (LPDDR5 only).....	52
5.10	Training Loops	53
5.11	TxDfE Training (LPDDR5 only).....	54
5.12	Strobeless Training.....	55
5.13	RxClk Training	56
5.13.1.1	Training Patterns	57
5.13.1.2	RxClk Training Extra Options	57
5.13.2	Coarse Bit Selection.....	58
5.14	2D Training	62
5.14.1	Rx 2D Training	63
5.14.1.1	Rx DfE Training	64
5.14.1.2	Rx Receiver Mode (DfeModeCfg)	64
5.14.2	Tx 2D Training.....	65
5.14.3	2D Training Options.....	65
5.14.3.1	2D Training Verbose Messages.....	65
5.14.3.2	VREF Step Control.....	65
5.14.4	2D training data dumps	72
5.15	Max Read Latency Training.....	73
5.16	PPT2 Initialization Training (for PUB2.x only)	75
5.17	Message Block Training Margin Reporting.....	76
5.18	End of Training.....	77
5.18.1	CDDs.....	77
5.18.2	PS RAM	78
6	Protocol Specific Requirements.....	79
6.1	Multiple PHYS.....	79
6.1.1	LPDDR4.....	79
6.1.2	LPDDR5.....	79
6.2	LPDDR Automatic Boot Clock Frequency switching	80
6.3	Pstate training order	80
6.4	DQ and CA swizzle	81
6.5	Byte Swap.....	81
6.6	Link ECC	81
7	Simulating Firmware.....	82
7.1	Simulation requirements for Firmware	82
7.1.1	DRAM model behavior	82
7.1.2	Hi-Z bus modeling.....	84
7.1.3	Disabling behavioral X-injection in FIFOs.....	84

7.1.4	DRAM refresh protocol checking	84
7.2	Simulation Environment Bring Up	85
7.3	Reducing Digital Simulation Time	85
8	Streaming Messages.....	86
9	Training Time Optimization.....	88
9.1	LPDDR54 Optimization conditions.....	88
9.2	General Procedure	88
9.3	PMU Clock Speed	88
9.4	Training Sequence.....	88
9.4.1	TX DFE (LPDDR5 Only)	88
9.4.2	PHY DCA (LPDDR5 Only).....	88
9.4.3	Training Loops (LPDDR5 only)	89
9.5	VREF Sweep Adjustment.....	89
9.5.1	Range and Step Selection Procedure	89
9.5.2	Example Settings.....	90
9.6	Trade-off of training time vs. accuracy	92

Revision History

Document Revision	Date	Description
2.40a	October 11, 2021	<p>Updated:</p> <ul style="list-style-type: none"> - Updated Section 3.5 Reading the training results via the message block - Added section 5.18.2 PS RAM - Added section 6.6 Link ECC
2.30a	June 15, 2021	<p>Updated:</p> <ul style="list-style-type: none"> - Version and Date - 5.13 RxClk Training - 9.4.3 Training Loops (LPDDR5 only) - 9.5.2 Example Settings
2.20a	April 19, 2021	<p>Updated Version and Date</p> <p>Updated Section 3.2 Loading the Training Firmware</p> <p>Updated Section 5.14.3.2.10 2DMisc Weighted Mean</p> <p>Updated Chapter 6 Protocol Specific Requirements</p>
2.10a	November 20, 2020	<p>Update supporting training features for C-2020.11 release (C- replaces both previous versions A- & B-)</p> <p>Added section 1.4 Supported Training Features</p> <p>Updates to section 5 including; TxDq Training, RxClk Training, Training loops, and Strobeless training</p> <p>Added section 5.17 Training Margin Reporting</p> <p>Added section 9 Training Time optimizations</p>
2.00	September 15, 2020	Update supporting training features for A/B-2020.09 release
1.00	June 30, 2020	Update supporting training features for A/B-2020.06 release

Document Revision	Date	Description
0.40	March 23, 2020	Update supporting training features for A-2020.03-T-53545 release
0.30	December 13, 2019	Update supporting training features for A-2019.12-T-0045591 release
0.20	May 28, 2019	Update supported training features and devinit
0.10	August 20, 2018	Initial version

Preface

This databook describes the LPDDR5/4/4X PHY Training Firmware Application Note.

Web Resources

- DesignWare IP product information: <https://www.synopsys.com/designware-ip.html>
- Your custom DesignWare 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/keys>

STAR on the Web (SotW)

- You must review all Stars on the Web (SotWs) associated with your product. SotWs are considered a part of the Synopsys documentation suite, and show critical information related to your product. To review product SotWs, refer to the DesignWare IP product information:

<https://www.synopsys.com/designware-ip.html>

Reference Documentation

The following reference documents are used in this application note:

- DesignWare Cores LPDDR5/4/4X PHY Databook, Synopsys, Inc.
- DesignWare Cores LPDDR5/4/4X PHY Utility Block (PUB) Databook, Synopsys, Inc.
- Dwc_lpddr54_phy_phyinit_overview_appnote, Synopsys, Inc.

on maximum supported board skews, refer to your PUB data book.



Not all training applies to each PHY. Specific training is based on the supported protocols and system configurations. Consult the PHY Databook for more information on training support.

For term definitions and glossary, see the PUB documentation.

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 waveforms file (such as VPD or VCD).
 - Identify the hierarchy path to the DesignWare instance.
 - 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://solvetplus.synopsys.com>
- 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: LPDDR54

- d. After creating the case, attach any debug files you created.

For more information about general usage information, refer to the following article in SolvNetPlus:

<https://solvetplus.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>

1 Introduction

1.1 Purpose

The purpose of this application note is to present an overview of the training firmware delivered with the PHY IP and instructions on how to use the firmware. The firmware is precompiled C code executed by a micro-controller (also known as the PMU) embedded inside the PHY Utility Block (PUB). To ensure proper training behavior, customers are required to interface with the firmware through the protocols explained in Section 3.3.

1.2 Firmware overview

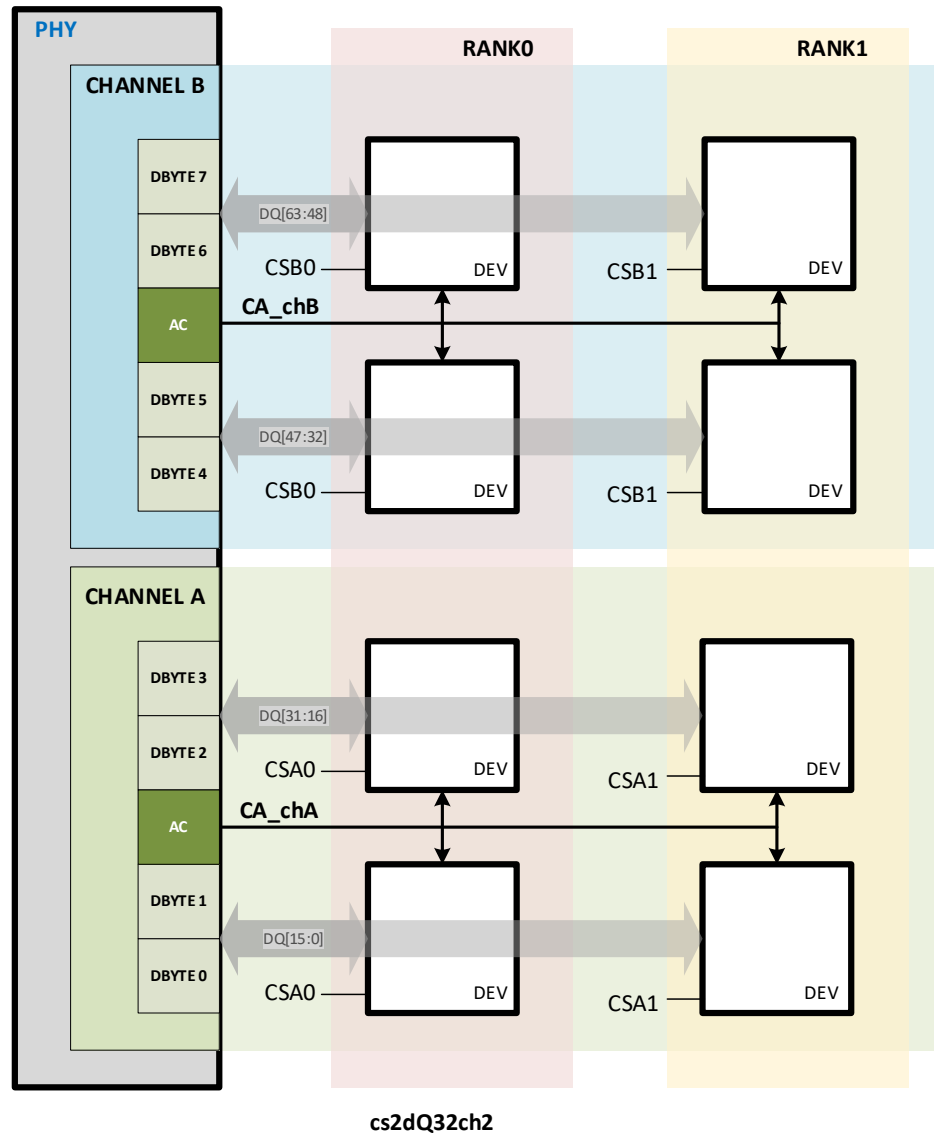
DDR memory systems have many timing relationships that must be satisfied for them to operate properly. Satisfying all these relationships is complicated by the fact that the relationships are affected by the delays between the PHY and the DDR memory. Factors such as board trace length and connectors affect these delays. The PHY firmware is designed to calculate the correct PHY configuration to ensure that these relationships are satisfied with minimum input from the user.

The PHY IP is trained through a sequence of logical training steps implemented in the firmware. Training takes an untrained system and optimizes it for both delay and voltage margin.

The individual steps that make up training are explained in section 4 Training Steps.

1.3 PHY Configurations

The PHY configuration is broken into 3 main categories: chip select (cs), data lanes (DQ), and Channels (ch). Chip select allow you to select which rank is active. The terms chip select, and timing group may be used interchangeably. The DQ number refers to number of data lanes and DBI/DMI pins added together. Channels describe the number of independent sets of data and address connections that exist.



chip selects: 2
data lanes: 32 per channel
channels: 2

Figure 1: Schematic Cs2DQ32Ch2 x16 devices

1.4 Supported Training Features

Depending on the PHY configuration and the type of memory attached, the PHY hardware and firmware support different training features. The tables below give an overview of how the supported features change for different memory standards and firmware images.

For details on maximum supported board skews, refer to your PUB data book.

Training	LPDDR5	LPDDR4/4X	Required	
			Simulation	Silicon
Devinit	- Per PHY	- Per PHY	Yes	Yes
CA Training	-Per Lane	-Per Lane	No	Yes
CS Training	-Per CS	NA	No	No
RxEn	-Per DBYTE -Per Rank	-Per DBYTE -Per Rank	Yes	Yes
WrLvl	-Per DBYTE -Per Rank	-Per DBYTE -Per Rank	Yes	Yes
DRAM DCA	-Per Device	NA	Yes	No
Read Training	-Per DBYTE -Per Rank -Per Lane -Per Differential Clock	-Per DBYTE -Per Rank -Per Lane -Per Differential Clock	Yes	Yes
PHY Read DCA Training	-Per DBYTE -Per Rank	NA	No	No
Write Training	-Per DBYTE -Per Rank -Per Lane	-Per DBYTE -Per Rank -Per Lane	Yes	Yes
PHY Write DCA Training	-Per Device	NA	No	No
Max Read Latency	-Per PHY	-Per PHY	Yes	Yes
TxDfE Training	-Per Device	NA	No	No



For silicon, the resulting minimum SequenceCtrl value is 0x121f for LPDDR4 and 0x125f for LPDDR5
For simulation, the resulting minimum SequenceCtrl value is 0x21f

2 Hardware Training Architecture

The PHY contains dedicated hardware to allow it to train a variety of memory protocols and configurations. This hardware is used during training to determine the optimal delay settings for the memory system with minimal input from the user. The hardware can also be used to optimize the PHY and DRAM settings in both time and voltage.

A high-level overview of the specialized training hardware controlled by the firmware is shown in figure 2.

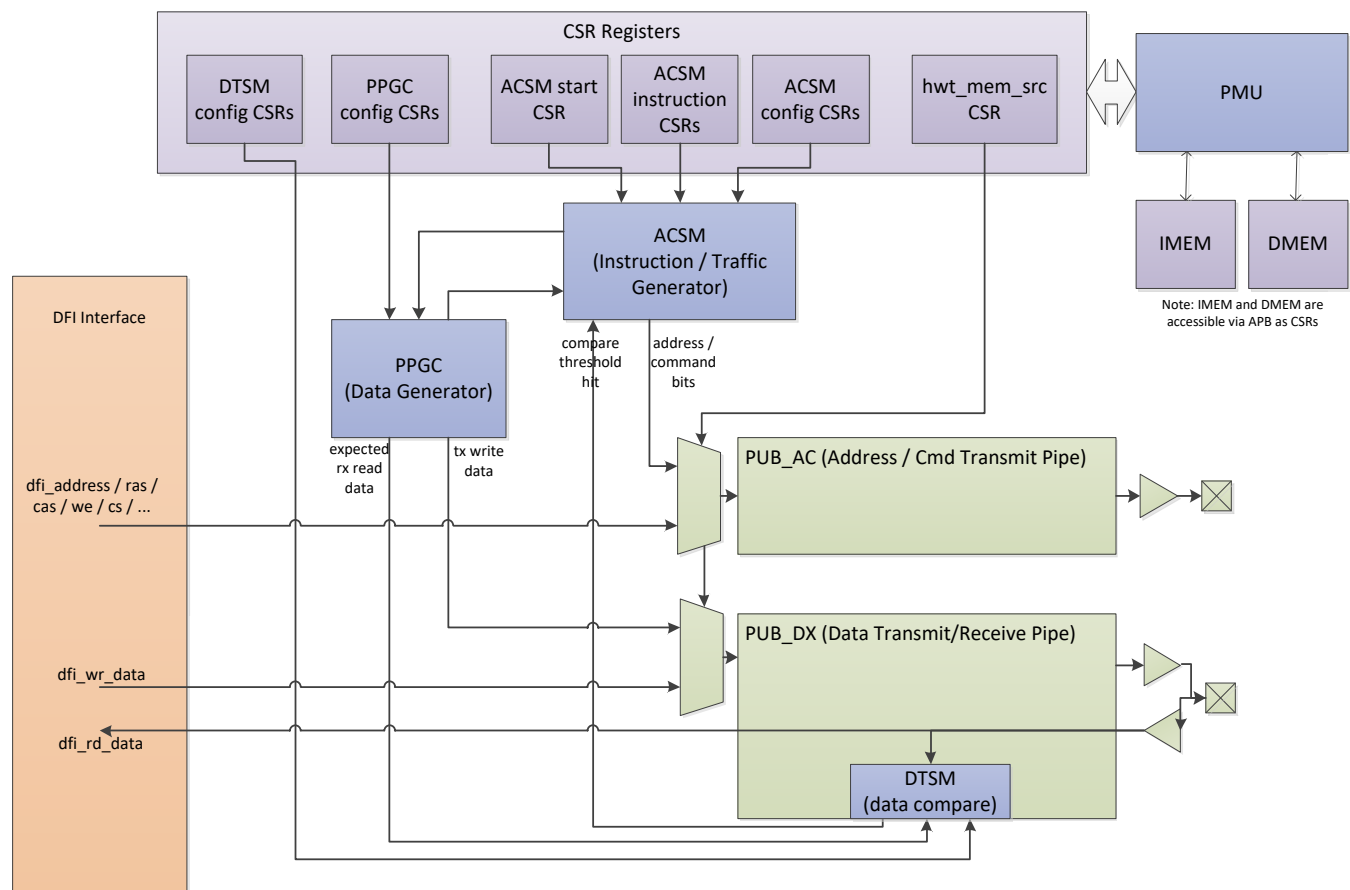


Figure 2: HW Training (HWT) architecture

The PHY's specialized hardware is spread across 3 logical blocks: address command (AC), DBYTE, and the PHY utility block (PUB). These hardware blocks are described more fully in the PUB documentation. The AC and DBYTE blocks are hard macros. The PUB is the synthesizable logic for the PHY. The PUB contains most of the dedicated training hardware designed to flexibly drive the DRAM interface at full speed.

The dedicated training hardware can be broken down into 4 major pieces:

- **PMU (Programmable Microcontroller Unit):** One instance per PHY. This is the microcontroller that runs the training firmware. It has dedicated instruction and data SRAMs that are used to store the training code and data structures during training. These memories are accessible over the APB bus, like the other PHY internal control registers. The PMU has access to the internal PHY control registers, which allows it to control the rest of the dedicated hardware and program the trained PHY state.
- **ACSM (Address-Command State Machine):** One instance per PHY. As the name suggests, the ACSM's purpose is to drive the address/command bus in sync with the DTSM.
- **DTSM (Data-Training State Machine):** Nine instances per DBYTE or one instance per DQ and DBI bit. Used to compare received data with an expected pattern. Each DTSM keeps track of how many failures and successes it has seen and can be programmed to automatically set flags when the fail or success counts pass separately programmable thresholds. The DTSM also controls automatic adjustments to trainable delays.
- **PPGC (PRBS Pattern Generator-Checker):** One instance per PHY. On write commands, the PPGC provides write data to the DTSM. On read commands, the PPGC tells the DTSM what should have previously been written into the memory location being read.

All training hardware is controlled by the firmware running on the PMU. The firmware running on the PMU reads and writes internal registers (called CSRs or Configuration Status Registers) that enable operating modes, set delays, or store information. The registers can also be accessed via the APB port to allow customers some visibility into the real-time status of training.



While the training firmware is running, only the registers marked as APBONLY will be accessible. All other registers cannot be accessed. For more information, see the Initialization section of the PUB documentation.

3 Loading, Configuring and Running the Firmware

3.1 Overview

During the PHY initialization process, the following high-level steps are run to train the PHY:

- The training firmware is loaded
- The firmware is configured with the information that it needs to run
- The firmware is run
- The training results are read back

This process is followed once for each frequency, or PState, that the PHY needs to train.

To run training, the following steps must be done:

1. Execute steps A-C of the flowchart.
2. Execute step (D), loading the firmware IMEM and DMEM images if not skipped.
3. Run the firmware once for each frequency, following steps E-I-Ps for each frequency.
4. Execute steps I-J to bring the PHY into mission mode.

For further details regarding the PHY execution sequence refer to the Phyinit Overview Application Note.

3.2 Loading the Training Firmware

Loading the firmware is accomplished by accessing the PMU SRAMs over the APB bus. The SRAMs are mapped into the APB address space starting at address 0x50000 for the instruction memory and 0x58000 for the data memory. The PMU firmware requires 64KB of instruction memory and 64KB of data memory.

The training firmware is provided in two formats:

- Binary image (.bin files)
- Text files with APB address/data pairs (.incv files)
- ECC files to enable backdoor loading of ECC SRAMs during simulation. (_ecc.txt files)

The two formats contain identical firmware images, and the user can use whichever format is more convenient. To use the APB address / data pairs file, write each data to the associated address. To use the binary images, load the binary contents of the file into the starting address that is appropriate for the instruction memory (for the IMEM image) and the data memory (for the DMEM image). For more information on when in the initialization process to perform the loading operation, see the “PHY Initialization” section of the PUB documentation.

There is a separate file for:

- the instruction memory image (<firmware_name>_imem.bin or <firmware_name>_imem.incv)
- the data memory image (<firmware_name>_dmem.bin or <firmware_name>_dmem.incv)

Both the instruction memory and the data memory images must be loaded. The instruction memory image contains the executable training code. The data memory image contains data structures that the instruction memory image needs to run. The data memory image must be reloaded between runs of the firmware to re-initialize the data structures. The reloading of the data memory image is during step F of the initialization process.

Separate images are provided based on the memory type. The following table details the firmware support for various memory types:

DRAM Type	Firmware file name
LPDDR4 UDIMM	Lpddr4_pmu_train_*
LPDDR4X UDIMM	Lpddr4x_pmu_train_*
LPDDR5 UDIMM	Lpddr5_pmu_train_*

Table 1 – Firmware support to Various Memory Types

3.3 Configuring the firmware via the message block

For the training firmware to run, it needs some basic information about the memory system and the training steps needed. This information is passed to the firmware by using a data structure called the message block. The message block is a reserved area in the micro-controller's data memory which is accessible over APB. The message block allows the user to configure the operation of the firmware for their system. The message block data structure is also used to pass information back from the firmware to the user. The message block is provided as a C data structure for convenience.

There is a separate C data structure definition for each of the provided data memory images. The header files can be found in the release package in the directory with firmware images. Most message block fields are annotated as "input", meaning the address must be written by the customer before the micro-controller begins execution. Some fields are annotated as "output" and provide feedback after training has completed. The last field type is the "inout"; addresses that serve both as inputs and outputs. See the message block field descriptions in the appropriate header file for the function of each field.

The structures in the file are named to correspond to the data memory image. The table below lists all the message block definitions:

DRAM Type	Header file name	Structure name
LPDDR4 UDIMM	mnPmuSramMsgBlock_lpddr4.h	_PMU_SMB_LPDDR4_t
LPDDR4X UDIMM	mnPmuSramMsgBlock_lpddr4x.h	_PMU_SMB_LPDDR4X_t
LPDDR5 UDIMM	mnPmuSramMsgBlock_lpddr5.h	_PMU_SMB_LPDDR5_t

Table 2 – Message Block Definitions

The message block should be configured with all required inputs prior to the execution of training and cannot be accessed through the APB bus while firmware training is running. Once training is done, the message block can be accessed to retrieve the training information calculated by the firmware.

3.3.1 Initial Device State

Each message block includes inputs for the MR settings to train with. Depending on the technology, memory configurations can be specified per-system, per-rank, or per-device. Most DRAM state inputs will directly match JEDEC control word definitions.

3.3.1.1 Illegal Configurations

The PHY supports many training features and options, but some configurations of the message block are not supported during training.

3.4 Running the firmware

Running the firmware is simply a process of taking the PMU out of reset and stall. The firmware image will then be run.

There is a protocol for starting the firmware and interacting with it to determine the training status. The “PHY Initialization” section of the PUB documentation shows exactly what steps need to be executed. PhyInit software provides an example C code for executing the training firmware in the `dwc_ddrphy_phyinit_G_execFW()` routine. See the PhyInit Application note and source code for more details on how to compile and run PhyInit.

Once the firmware is running, it will pass messages back to the user to indicate overall completion, completion of individual steps and additional debug messages. These messages are passed back via the mailbox facility. The number and types of messages sent are configurable. Details on the mailbox and the messaging can be found in the next section.

3.4.1 Mailbox facility for firmware

The interface for passing status messages between the PHY training firmware and a customer’s design is implemented using a 4-phase mailbox protocol over APB that allows the micro-controller to synchronously send feedback to the customer while training is being executed. The mailbox is used for general status and debug. The number of messages that the firmware passes back to the user is controllable with a message block field (`HdtCtrl`).

Interaction with the mailbox is not required during training if the firmware is configured to pass minimal messages back (`HdtCtrl` setting `0xff`).

To ensure that the messages are received by the user, the firmware implements a flow control that will stall execution until any message that is sent has been acknowledged as received. If the firmware is configured for minimal messages, then the only message sent will be the completion message (which does not need to be acknowledged).

To read and acknowledge the messages sent by the firmware, the user will access the following registers:

Register Field	Mailbox use
UctWriteProtShadow	When set to 0, the PMU has a message for the user
UctWriteOnlyShadow	Used to pass the message ID for major messages. Also used to pass the lower 16 bits for streaming messages.
UctDatWriteOnlyShadow	Used to pass the upper 16 bits for streaming messages. Not used in passing major messages.
DctWriteProt	By setting this register to 0, the user acknowledges the receipt of the message.

Table 3 – Registers Fields

The protocol for using these registers can be broken down into the following steps:

1. Poll the UctWriteProtShadow, looking for a 0
2. When a 0 is seen, read the UctWriteOnlyShadow register to get the major message number. If reading a streaming message, also read the UctDatWriteOnlyShadow register.
3. Write the DctWriteProt to 0 to acknowledge the receipt of the message
4. Poll the UctWriteProtShadow, looking for a 1
5. When a 1 is seen, write the DctWriteProt to 1 to complete the protocol
6. Go to step 1.

By following the 4-phase protocol, the customer will know when a valid message is ready to be read from the message CSRs and the firmware will know if a previous message has yet to be read. The firmware is careful to avoid overwriting unread messages and in the worst cases will even stall execution to protect an unread message. The process for decoding major messages and streaming messages is in the following sections.



The terminology of “mail” is sometimes used to describe information passed through the “mailbox”.

Below is a general pseudocode algorithm for implementing the protocol:

```
//-----
// get_mail function to handle protocol and return message
// mode sets 16 bit (major message) or 32 bit (streaming message)
//-----
Function get_mail (mode){
    While (APB_read(UctWriteProtShadow) != 0){ }
    mail = APB_read(UctWriteOnlyShadow)
    if (mode == 32bit){ mail |= (APB_read(UctDatWriteOnlyShadow) << 16) }
    APB_write(DctWriteProt, 0)
    While (APB_read(UctWriteProtShadow) == 0){ }
    APB_write(DctWriteProt, 1)
    Return mail
}

//-----
// Main
//-----
Function main(){
    APB_write(DctWriteProt, 1)           //protocol initialization
    APB_write(UctWriteProt, 1)          //protocol initialization
    start_firmware()                    // See PUB documentation
    While (completion message not seen){
        mail = get_mail(16bit)
        decode_major_message(mail)      // See section 3.4.2
        If (mail == 0x08 ): get_streaming_message() // See section 10 for Streaming Messages
    }
}
```

3.4.2 Decoding Messages

A “message” is a set of values passed through the mailbox protocol. There are two distinct types of messages: major messages and streaming messages (for streaming messages, see section 10).

3.4.2.1 Major Messages

Major messages are made up of a single value. The encoding of the major messages is held constant for all firmware images unless noted in the release notes. The pass/fail status of the firmware can be obtained by reading the message block instead of the mailbox.

Major messages are broken into 3 categories: Firmware Complete, Stage Completion, and Start Streaming messages.

The major message is obtained by reading the value of `UctWriteOnlyShadow`. The value can then be referenced to the tables below to determine what the major message is. The number and categories of messages sent by the firmware is controlled by setting the `HdtCtrl` field of the message block as described in section 3.4.2.2 (Controlling the Number of Messages).

Firmware Complete major messages indicate the end result of the firmware run. Receiving either one of these messages indicates that the firmware is complete and has stopped running.

The Firmware Complete major message values are:

UctWriteOnlyShadow Value	Message meaning
0x07	Training has run successfully (firmware complete)
0xff	Training has failed (firmware complete)

Table 4 – Firmware Complete Major Messages

The Stage Completion Major Messages indicate when each stage of the training has finished. Each of these messages must be acknowledged via the mailbox protocol or the firmware will stall and wait.

The Stage Completion major message values are:

UctWriteOnlyShadow Value	Message meaning
0x00	End of initialization
0x01	End of fine write leveling
0x02	End of read enable training
0x03	End of read delay center optimization
0x04	End of write delay center optimization
0x05	End of RxDigStrobe training
0x06	End of DRAM DCA training
0x07	PMU has completed all of its SequenceCtrl tasks and is in a power gated idle state
0x09	End of MaxRdLat training
0x0a	End of PHY RX DCA training
0x0b	End of PHY TX DCA training
0x0c	Reserved
0x0d	End of CA training
0x0e	End of TxDFE training
0xfd	End of Read Training center optimization (SI Friendly portion of 1d RdDqs training)
0xfe	End of Write leveling coarse delay

Table 5 – Stage Completion Major Messages

The Start Streaming Message indicates that the firmware is beginning a streaming message. Streaming messages contain debug information and are not required to be enabled. A streaming message consists of multiple values. Streaming messages must be processed completely to acknowledge them when they are enabled, or the firmware will stall and wait. To process streaming messages, follow the protocol in Chapter 9 (Streaming Messages).

The Start Streaming Message major message value is:


UctWriteOnlyShadow Value	Message meaning
0x08	Start streaming message mode (see Chapter 9 for details)

Table 6 – Start Streaming Message Major Message

3.4.2.2 Controlling the Number of Messages

To control the total number of debug messages, a verbosity subfield (HdtCtrl) exists in the message block. Every message has a verbosity level associated with it, and as the HdtCtrl value is increased, less important messages stop being sent through the mailboxes. The meanings of several major HdtCtrl thresholds are explained below:

HdtCtrl Value	Description	Messages the firmware will send				
		Pass/Fail Major Messages	Assertion Streaming Messages	Stage Completion Major Messages	General Debug Streaming Messages	Detailed Debug Streaming Messages
0x04	Maximum Detailed debug (e.g. intermediate training step info)	Y	Y	Y	Y	Y
0x05	Detailed debug (e.g. eye delays)	Y	Y	Y	Y	Y
0x0A	Coarse debug (e.g. rank information)	Y	Y	Y	Y	N
0xC8	Stage completion	Y	Y	Y	N	N
0xC9	Assertion messages	Y	Y	N	N	N
0xFF	Firmware Complete	Y	N	N	N	N

 Warning!	If HdtCtrl is set to a value other than 0xFF, major and streaming messages will need to be processed and the mailbox must be polled and the messages retrieved. Failure to process the messages will result in the firmware stalling while waiting for the message to be acknowledged.
-----------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Setting HdtCtrl to 0xff means that the mailbox does not need to be polled. Setting HdtCtrl to 0xff also guarantees that no streaming messages will be sent and only the final pass or fail condition will be reported via the mailbox.

3.5 Reading the training results via the message block

The training firmware will use the fields in the message block that are marked as output or inout to send information back to the user. For more information on the message block, see section 3.3. These fields will contain information such as:

- Pass / Failure of the training (CsTestFail)
- Critical delay differences for accesses (CDD_*)
- Trained VREF values (VREF*)
- Trained DRAM DFE values (DRAMDFE_*)
- Trained DRAM DCA values (DRAMDCA_*)

See the message block field descriptions in the appropriate header file for the function of each field.

4 Training Steps

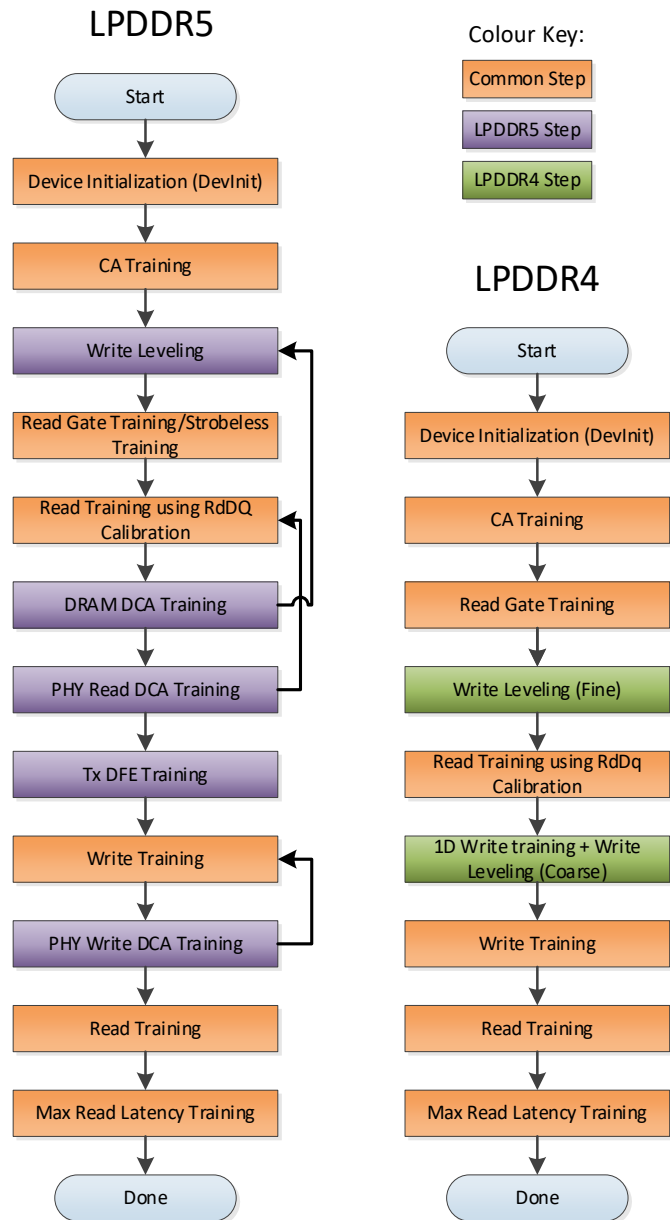
4.1 Overview

Training is a group of stages that optimize a system's delays and voltages. Each training step focuses on a specific delay or voltage, building off earlier steps to eventually result in a fully trained system.

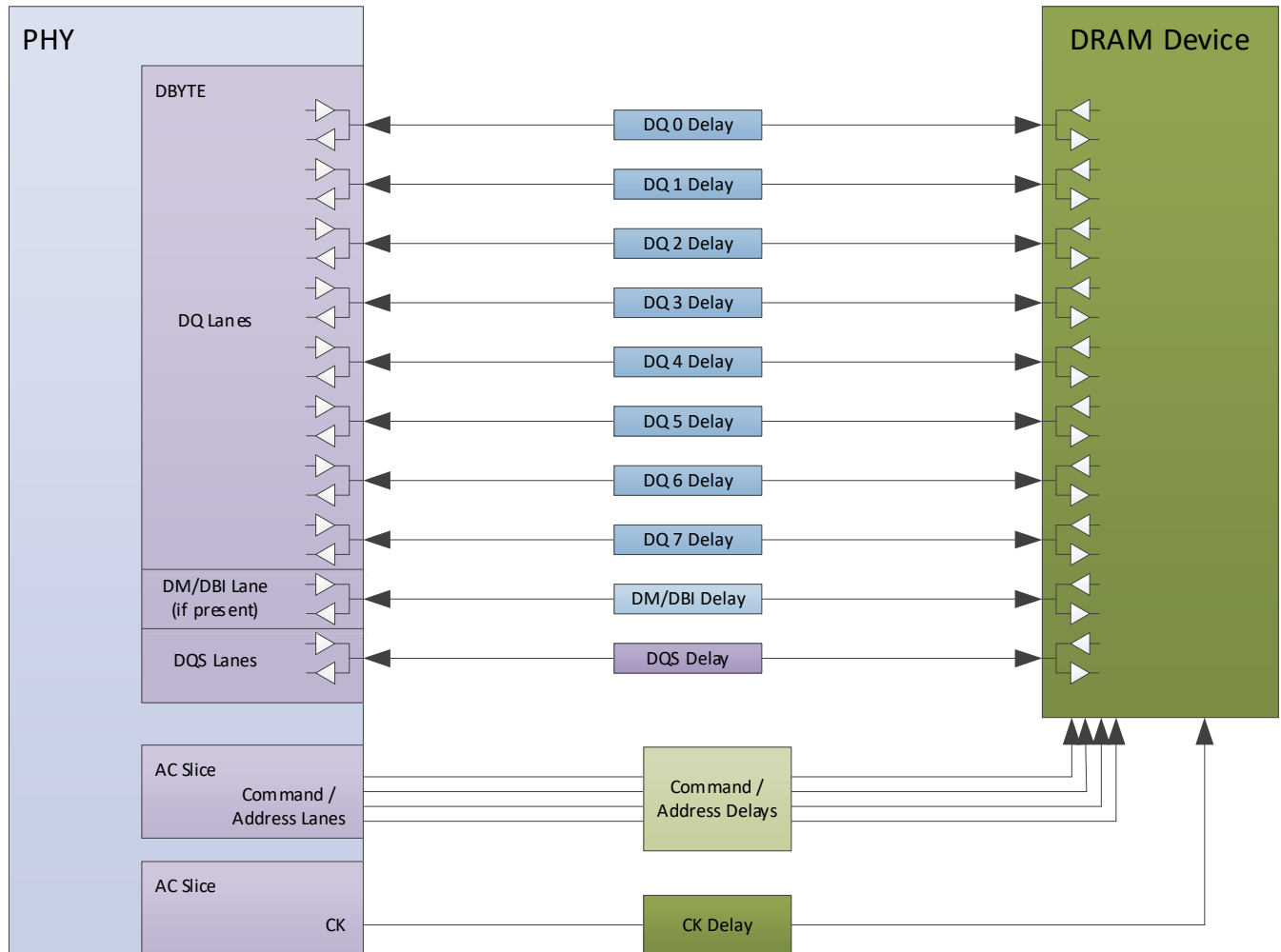
The major training stages, in order of execution, are:

- Device Initialization
- Command / Address Training
- Write Leveling
- Read Gate Training
- Read Strobeless Training (LPDDR5 only)
- Write Leveling (broken into a Fine and Coarse step)
- Read Training (into Dq Calibration and full steps)
- DRAM Duty Cycle Adjustment (LPDDR5 only)
- PHY Duty Cycle Adjustment (broken into read and write, LPDDR5 only)
- Tx DFE Training (LPDDR5 only)
- Write Training
- Max Read Latency Training

Later sections will discuss how the individual training steps work.

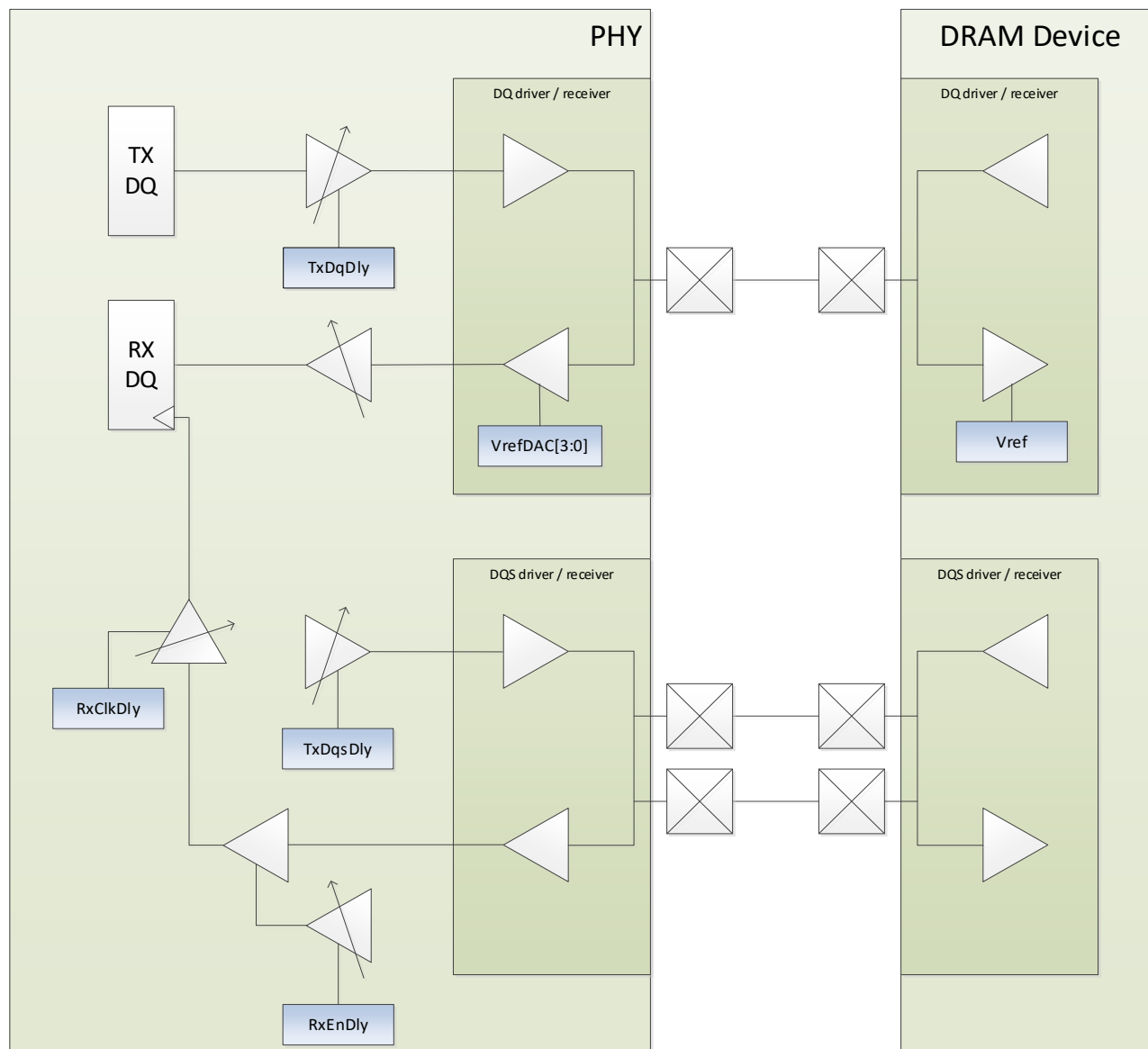


The training process is designed to compensate for delays in the system, both board and DRAM. The various delays can be seen in the diagram below:



The diagram shows the signals from a single DBYTE and a couple AC instances. Each signal that goes to the AC slices can be independently controlled. The delays in the boxes outside of the PHY and the DRAM Device represent board, connector, package, and other system related delays.

The values that are trained in the PHY and DRAM can be seen in the following diagram:




This diagram shows single DQ and single DQS lane, and the trainable delay and VREF values in both the PHY and the DRAM devices.

5 Training step details

The following sections will go into more detail for each of the training steps to provide information about how to run the step, what the training is accomplishing, a high-level description of the training process, and which CSRs are being trained in each step.

The training steps are always executed in the same order, and use the information provided in the message block to accomplish the training. The SequenceCtrl field in the message block is used to tell the firmware which steps to run. The field is divided into bits for each applicable step. Setting a bit to 1 will cause the firmware to execute that step. Setting a bit to 0 will cause the firmware to skip that step. Any bit that is marked as reserved or RFU must be set to 0.

 Warning!	The PHY does not guarantee proper operation if any training stages are skipped or executed out of sequence. Each subsequent step is dependent on CSR values, internal states, and configurations (of both the PHY and the DRAMs) that are set during prior steps. Stopping and resuming training is not supported. Reconfiguration of the PHY or DRAM while training is occurring is not supported. Violation of any of these requirements may result in an inoperable memory system.
-----------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.1 Device Initialization (DevInit)

To perform Device Initialization, set the DevInit bit of the SequenceCtrl field of the message block prior to starting the firmware.

Device initialization performs the basic initialization of the PHY and DRAM that is required before any training stage is run. Initialization includes tasks such as calibrating the internal LCDLs to the current input clock, calibrating the RxReplica LCDLs, calculating the parameters to accelerate future lock operations, resetting the DRAMS, and writing the appropriate MRs. Training may modify the initial MRs from the values provided in the message block to avoid putting the DRAMs into an untrainable state.

The CSRs affected by this step are:

CSR Name	Description	Protocol		
		LPDDR4	LPDDR4x	LPDDR5
DWC_DDRPHYA_MASTER_LP5Mode	When PHY supports LP5 Protocol	Y	Y	Y
DWC_DDRPHYA_MASTER_HwtLpCsEnA[1:0]	Enable bits for Channel A	Y	Y	Y
DWC_DDRPHYA_MASTER_HwtLpCsEnB[1:0]	Enable bits for Channel B	Y	Y	Y
DWC_DDRPHYA_DBYTE<N>_PptCtlStatic[11:0]	Enable DRAM drift Compensation	Y	Y	Y
DWC_DDRPHYA_AcDllLockParam_p[8:0]	DLL locking for AC LCDL	Y	Y	Y
DWC_DDRPHYA_DxDllLockParam_p[8:0]	DLL locking for DBYTE LCDL	Y	Y	Y
DWC_DDRPHYA_DllGainCtl_p[15:0]	DllGain for locking LCDL	Y	Y	Y
DWC_DDRPHYA_RxReplicaDllLockParam_p[8:0]	DLL locking for Rx Tracking LCDL	Y	Y	Y
DWC_DDRPHYA_RxReplicaDllGainCtl_p[15:0]	DllGain for locking Rx Tracking LCDL	Y	Y	Y

Where <N> designates the corresponding DBYTE instance.



The training flowchart indicates in what order the steps are started. DevInit is overlapped with CA Training. DevInit is started, then CA training is performed, then DevInit is finished. The messages for step completion may indicate that the steps finished in the opposite order as indicated by the flowchart. This is expected because of the organization of the firmware.

The message block settings that affect this step are:

Message Block Field Name	Default	Description	Protocol	
			LPDDR4	LPDDR5
DisableTrainingLoop	0x7	Set these fields to skip training steps that are looped [0] = 1 disable all retraining stages in loop 1: Write leveling, RxEn, Read Dq Cal [1] = 1 disable all retraining stages in loop 2: Read DQ Cal and, DRAM DCA [2] = 1 disable all retraining stages in loop 3: Write Training	N	Y

The DisableTrainingLoop feature is used to skip training loops used to improve the initial trained values after a DCA step (Rx, Tx, or DRAM) has been run. In the first loop controlled by bit 0, Write Leveling, RxEn, and Read RDC are all performed a second time. In the second loop controlled by bit 1, Read RDC and DRAM DCA are all performed an additional time. In the third Loop controlled by bit 2, Write training is performed an additional time. If the DCA training step responsible for each loop is skipped, then the loop will also be skipped.

5.2 Fine Write Leveling

To perform Write Leveling Training, set the WrLvl bit of the SequenceCtrl field of the message block prior to starting the firmware. The Write Leveling training step is broken into two parts, fine and coarse. Both the fine and coarse training parts are performed when the WrLvl bit is set.

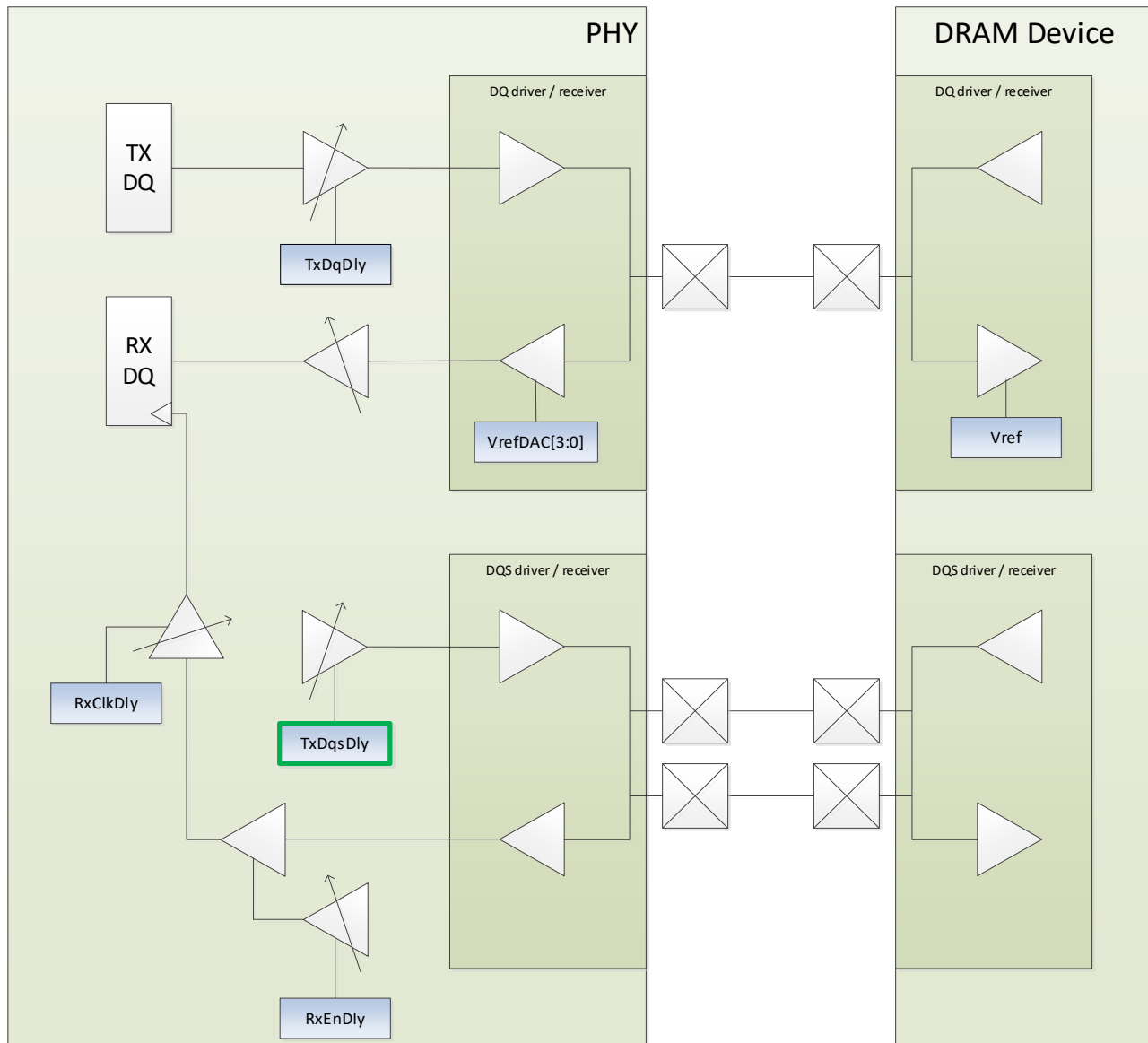
The goal of write leveling fine phase training is to align DQS rising edge with the MEMCLK rising edge at each DRAM device. Write Leveling fine phase training is required to compensate for any analog phase differences between MEMCLK and DQS distribution.

Just like read enable training, write leveling must be done serially for every cs. The training firmware will set up the DRAMs for ranks not being trained so that they will not interfere with the training of the current rank.

The firmware starts with a 2UI write leveling delay and looks for all the DQ lanes to have sampled a 1 from MEMCLK. The firmware repeatedly drives pulses on the DQS and MEMCLK buses and monitoring the DQ bus for the expected samples, automatically increasing (resp. decreasing) the DQS delays between bursts when the monitored sample is 0 (resp. 1). The firmware then sets the DQS delays to correspond to the correct edge of DQS to arrive at the DRAM aligned with MEMCLK. Training should converge with values in the range [1UI, 3UI].

This fine training ensures that the edges are aligned correctly, but not necessarily at the correct WCAS latency. The coarse part of the Write Leveling training will calculate the delay to ensure the edges align with the correct WCAS latency. See JEDEC specifications for example timing diagrams.

This step will write the values for all the TxDqsDly CSRs.



The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_TxDqsDlyTg0_p[1:0]	Timing group 0 timing for write DQS of DBYTE	Y	N
DWC_DDRPHYA_DBYTE<N>_TxDqsDlyTg1_p[1:0]	Timing group 1 timing for write DQS of DBYTE	Y	N
DWC_DDRPHYA_DBYTE<N>_TxWckDlyTg0_p[1:0]	Timing group 0 timing for WCK of DBYTE	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWckDlyTg1_p[1:0]	Timing group 1 timing for WCK of DBYTE	N	Y

Where <N> designates the corresponding DBYTE instance.

This step compensates for the phase difference between the CK signal and the DQS at each DRAM device (highlighted in green below). A delay is calculated for each DBYTE to each timing group by using the write leveling mode of the DRAM devices to place the rising edge of DQS as close as possible to the rising edge of CK.

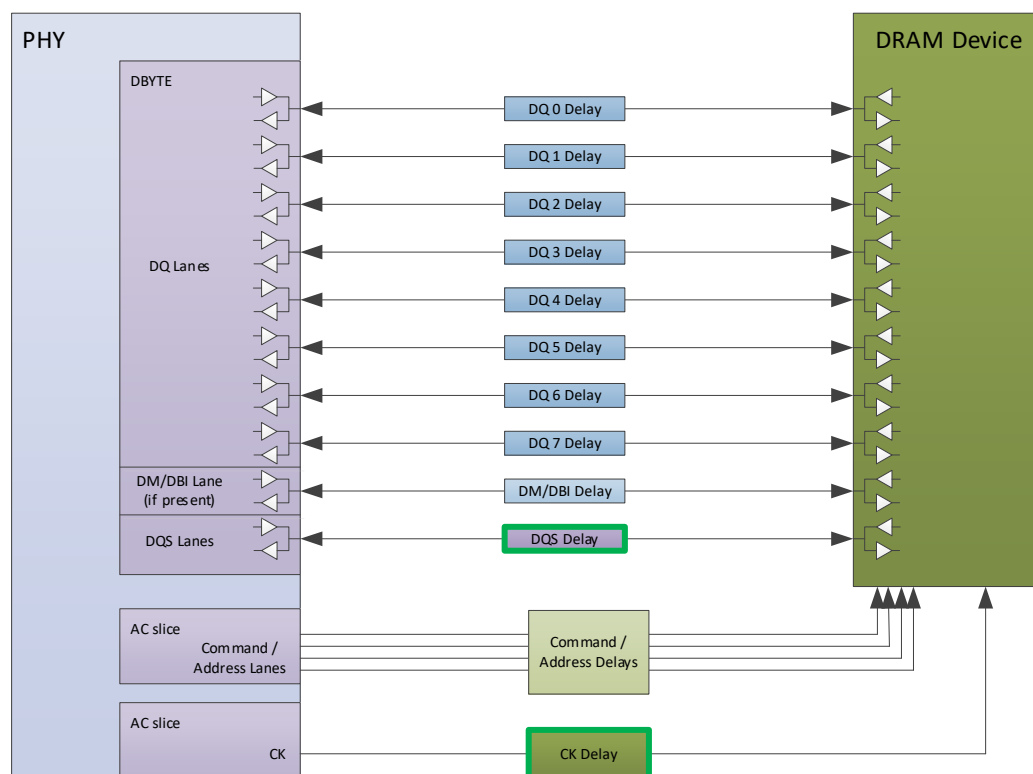


Figure 3 PHY-DRAM interface (LPDDR4)

5.3 Coarse Write Leveling for LPDDR4

Coarse write leveling is supported for LPDDR4. It is executed as a separate step after RxClk Training, where simple test transactions are sent to the DRAM to test the coarse bits of the LPDDR4 dram. To accomplish this, a preliminary Tx Dq training is done in this step. The same SequenceCtrl bit that enables fine write leveling also controls coarse write leveling. TX DQS was phase aligned to MEMCLK during write leveling fine phase training, but TX DQS latency may not match CAS write latency. As TX DQS delay is already phase aligned, it can be incorrect by some integer multiple of MEMCLK periods, as shown below.

```

LLLLllrhflrhflrhflrhflrhflrhflrhfl DQS
LLLLLLLLLLLLllrhflrhflrhflrhflrhfl Memclk

```

The initial DQ training is done with a simple 2ui repeating pattern to ensure the data read back in coarse write leveling is valid. Once the preliminary DQS-DQ timing is set, data can be written and read back to do the coarse write leveling search.

The coarse search writes a low frequency square wave to the rank being trained and then reads back the values. The PHY generates extra DQS pulses to ensure that the DRAM devices will see enough DQS pulses during the time of the write for incorrect write level timing. Any error in the comparisons between what was written to the DRAM and what was read out can be attributed to a coarse offset between the DQS and MEMCLK signals. The firmware adds to the DQS delay currently set and re-runs the square wave write/read sequence to check if the new delay fixed the errors. The firmware continues to loop, adding delay, until the data comes back without issue. If all delay settings fail to return the data as it was written, the firmware will terminate the training.

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_TxDqsDlyTg0_p[1:0]	Timing group 0 timing for write DQS of DBYTE	Y	N
DWC_DDRPHYA_DBYTE<N>_TxDqsDlyTg1_p[1:0]	Timing group 1 timing for write DQS of DBYTE	Y	N

Where <N> designates the corresponding DBYTE instance.

5.4 Coarse Write Leveling for LPDDR5

LPDDR5 Coarse write leveling (aka LPDDR5 Coarse Wck2Ck leveling) is different from LPDDR4 because of the necessity of having correct WCK timing to perform read operations. This step is executed immediately after LPDDR5 fine write leveling. In fine WCK leveling, WCK was phase aligned to CK during the fine write leveling fine training, but WCK latency may be incorrect by an integer of WCK periods.

For LPDDR5 coarse WCK leveling, the same write leveling mode that is used for fine write leveling is performed for coarse write leveling, at a reduced CK frequency to increase the write leveling range.

The divided clock is used to send a boot frequency MCLK, and a special ACSM sequence is used to synthesize a WCK that matches the CK speed. The coarse delay for WCK is incremented at this slow speed, which permits the PHY to infer the coarse bits of the WCK delay.

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_TxWckDlyTg0_p[1:0]	Timing group 0 timing for WCK of DBYTE	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWckDlyTg1_p[1:0]	Timing group 0 timing for WCK of DBYTE	N	Y

Where <N> designates the corresponding DBYTE instance.

5.4.1 Write Level Training configuration options (WrLvlTrainOpt – LPDDR5 only)

There are configuration options for LP5 write leveling training provided by the message block parameter WrLvlTrainOpt. The bitwise description is as follows:

Bit number	Field Name	Description
0	LP5 Coarse Write Leveling step	0 – Execute LP5 Coarse Write Leveling 1 – Skip LP5 Coarse Write Leveling
1	WCK ODT settings	0 – PHY's WCK ODT is disabled (recommended) 1 – PHY's WCK ODT is enabled

5.4.1.1 LP5 Coarse Write Leveling step

It is possible to skip LP5 Coarse Write Leveling step by setting bit[0] in message block parameter WrLvlTrainOpt.

5.4.1.2 WCK ODT Settings

Write leveling should be performed with DRAM's WCK-ODT ON and PHY's WCK-ODT OFF in actual use case. Hence, disable PHY WCK ODT by default. If required, it can be enabled by setting WrLvlTrainOpt[1]

5.5 Read Gate Training

To perform Read Gate Training, set the RxEn bit of the SequenceCtrl field of the message block prior to starting the firmware.

Read gate training (RxEn training) is used to determine the round-trip latency, between the PHY and the DRAM. The round-trip latency must be known so that the PHY's receivers are activated by the read gate at the right time to capture the full burst of read data. All the DBYTE instances can be trained in parallel, but each rank and channel are trained independently.

Read enable training lines the DQS gate, also called RX enable, up with the first rising edge of DQS for the read. After this alignment, a required setup time is subtracted from the aligned RX enable, placing the RX enable delay just before the first rising edge of DQS.

The PHY starts searching at the minimum possible RX enable delay; a value calculated by assuming an ideal 0-delay system (no board delay and minimal tDQSCK). The first step is to find any falling edge of DQS. The internal hardware is set to lock to the falling edge of DQS using the PHY internal DQS sample flop. The sequence issues several back to back reads, with the middle read activating the sample flop.

The training hardware increases delay when it samples 1 and decreases when it reads a 0, thereby locking to a falling edge.

```
Ttadgtttadgtttadgttttttttttttttttttttttt CMD (Read)
Ttttttttttttttttttttttaddgttttttttttttttttttttt CMD (Read Resp)
||||||rhflrhflrhflrhflrhflrhflrhflrhflrhfl|||| RDQS_t
L||||||rhflrhflrhflrhflrhflrhflrhflrhflrhfl|||| DQSGATE
```

To simplify the algorithm and to support memory types that do not universally support infinite preamble mode a special algorithm is used to position the read gate at the beginning of the read burst. The PHY will perform a series of reads with a specific spacing where the static postamble and the preamble to merge and provide a trainable pattern:

```
TtttadddgTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT CMD (Read)
TTTTTTTTTTTTTTTTTadddgTTTTTTTTTTTTTTTTT CMD (Read Resp)
||||rhflrhflrhflrhfl||||rhflrhflrhflrhfl|||| RDQS_t
LLLLLLLLLLLLLLLLLLLLrhrrrrrrrrrrrrrrrrrrrr DQSGATE
```

This step will write the values for all the RxEnDly CSRs. If any byte fails to find its RX enable delay, then the firmware aborts execution of all future steps and signals failure to the memory controller.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_RxEnDlyTg0_p[1:0]	Timing group 0 timing for read gate	Y	Y
DWC_DDRPHYA_DBYTE<N>_RxEnDlyTg1_p[1:0]	Timing group 1 timing for read gate	Y	Y

Where <N> designates the corresponding DBYTE instance.

This step compensates for the delay for the read command to reach each DRAM device and have the DQS pulse for the read return to the PHY (highlighted in green below). This delay does include all DRAM DQS related parameters such as tDQSK. This delay does not include the read CAS latency, which is known ahead of time.

A delay is calculated for each DBYTE to each timing group by finding the time of the first rising edge of DQS for each timing group and changing the delay of the RxEn to place it as close to the middle of the DQS preamble as possible.

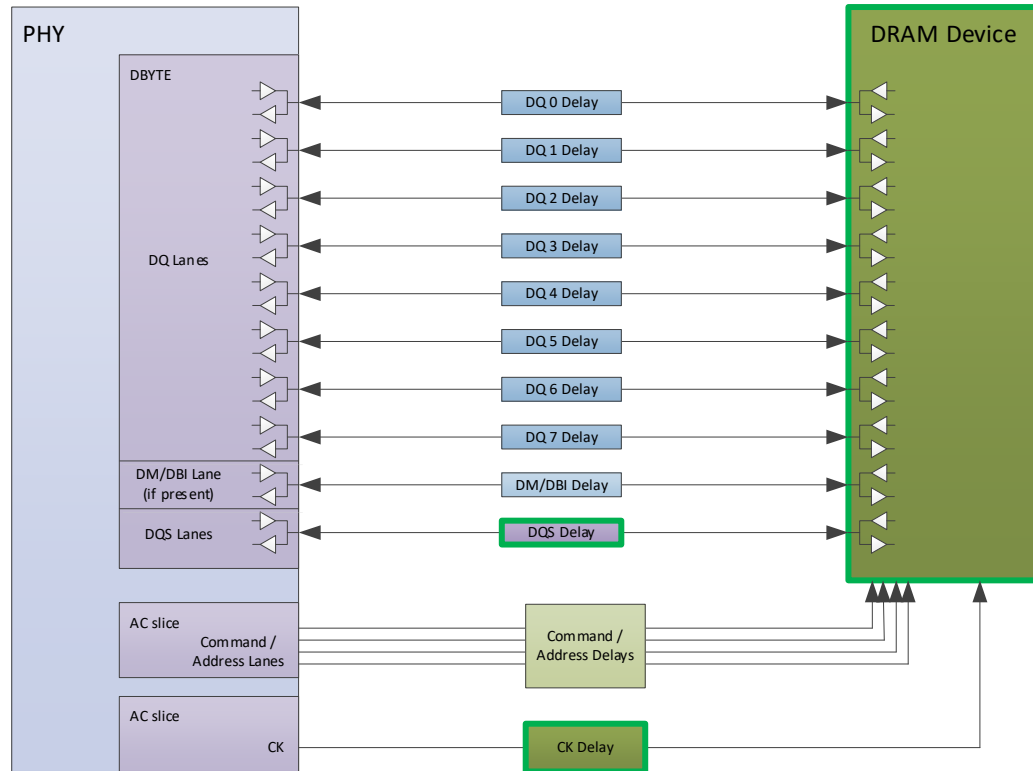


Figure 4 PHY-DRAM Interface (LPDDR4)

5.6 TxDQ Training

To perform Write training (TxDq), set the WrDq bit of the SequenceCtrl field of the message block prior to starting the firmware.

Write training is used to determine the PHY to DRAM latency and the DRAM VREF. This latency must be known so that the DRAM's receivers are activated by the time PHY to DRAM latency plus write latency has elapsed. All the DBYTE instances can be trained in parallel, but each rank is trained independently.

Write training is done by sending bursts of write FIFO transactions to the DRAM and then reading back that data with bursts of read FIFO transactions. The FIFO transactions use PRBS23 data and can produce DBI or non DBI like patterns. The first step is to find the estimated tDRAMDQS2DQ by reading the DQS Oscillator and use this with TxDQS delay to estimate the TxDQ delay. We scan 1UI around our estimated TxDQ delay value for a total of a 2UI data scan. Our data scan stores the error rate at each delay and voltage setting. If 2D training is enabled, then we repeat this scan for all valid DRAM VREF settings.

When 2D is disabled we pick the center point of the passing region in the 2UI scan to provide maximum margin and keep the default VREF.

In the case of full 2D, the data is turned into an eye contour as the data is collected. This contour is saved for all lanes, DBYTEs, ranks, channels, and DFE taps. Then the data is compounded across the shared resources, for write training the DRAM has one VREF per DBYTE. Then the data is sent to the 2D center finding Algorithm to select the VREF and delay with the optimal settings.

This step cannot be combined with Tx DFE training as the amount of memory to store all the eyes for each setting in Tx DFE exceeds the DMEM size.

The CSRs and MRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_TxDqDlyTg0_r[8:0]_p[1:0]	Timing group 0 timing for write DQ delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqDlyTg1_r[8:0]_p[1:0]	Timing group 1 timing for write DQ delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqLeftEyeOffsetTg0_r[8:0]_p[1:0]	Timing group 0 timing for write DQ left offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqLeftEyeOffsetTg1_r[8:0]_p[1:0]	Timing group 1 timing for write DQ left offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqRightEyeOffsetTg0_r[8:0]_p[1:0]	Timing group 0 timing for write DQ right offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqRightEyeOffsetTg1_r[8:0]_p[1:0]	Timing group 1 timing for write DQ right offset	Y	Y
MR14	VREF(DQ) settings	Y	Y
MR15	VREF(DQ)[15:8] settings	N	Y

Where <N> designates the corresponding DBYTE instance.

This step compensates for the delay for the write command to reach each DRAM device and have the data stored correctly. This delay includes parameters such as tDRAMDQS2DQ and related parameters. This delay does not include the write CAS latency, which is known ahead of time.

A delay and VREF is calculated for each lane to each timing group by finding the eye of settings and placing the trained result to maximize the margins.

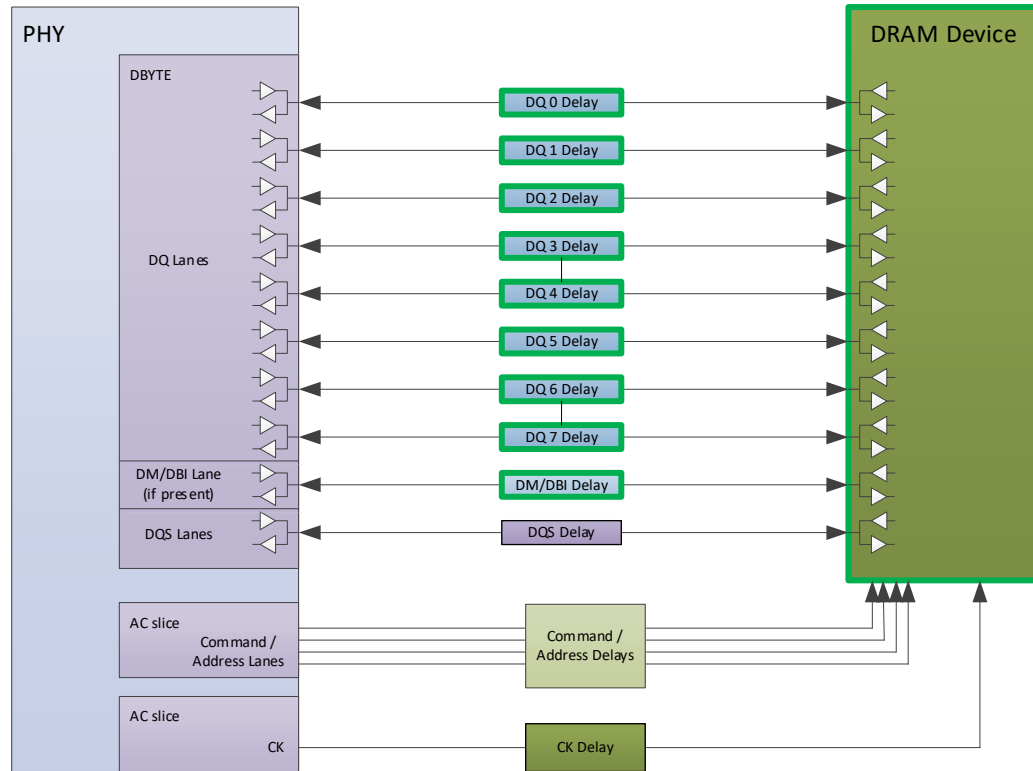


Figure 5 PHY-DRAM Interface (LPDDR4)


The message block settings that affect this step are:


Message Block Field Name	Default	Description	Protocol	
			LPDDR4	LPDDR5
DisableTrainingLoop	0x7	Set these fields to skip training steps that are looped [0] = 1 disable all retraining stages in loop 1: Write leveling, RxEn, Read Dq Cal [1] = 1 disable all retraining stages in loop 2: Read DQ Cal and, DRAM DCA [2] = 1 disable all retraining stages in loop 3: Write Training	N	Y
Train2DMisc	0x0	2D Training Miscellaneous Debug Control Train2DMisc[0]: Print Verbose 2D Eye Contour 0 = Do Not Print Verbose Eye Contour 1 = Print Verbose Eye Contour Train2DMisc[1]: Print Verbose Eye Optimization Output 0 = Do Not Print Verbose Eye Optimization Output 1 = Print Verbose Eye Optimization Output	Y	Y
TX2D_Delay_Weight	0x0	[0-7] 0 ... 255 During TX 2D training when finding an eye center the delay and voltage components are weighed such that the combined margin is delay margin * TX_Delay_Weight2D + voltage margin * TX_Voltage_Weight2D. Either weight may be zero but if both are zero each weight is taken to have a value of one.	Y	Y
TX2D_Voltage_Weight	0x0	[0-7] 0 ... 255 During TX 2D training when finding an eye center the delay and voltage components are weighed such that the combined margin is delay margin * TX_Delay_Weight2D + voltage margin * TX_Voltage_Weight2D. Either weight may be zero but if both are zero each weight is taken to have a value of one.	Y	Y
BitTimeControl	0x0	BitTimeControl [0-2]: Input for the amount of data bits per DQ before deciding if any specific voltage and delay setting passes or fails. Every time this input increases by 1, the number of data comparisons is doubled. The run time will increase proportionally to the number of bit times requested per point. 0 = 2 ⁰ times of basic amount (default behavior) 1 = 2 ¹ times of basic amount 2 = 2 ² times of basic amount ... 7 = 2 ⁷ times of basic amount Reserved09[3-7]: RFU, must be zero	Y	Y
TX2D_TrainOpt	0x0	Reserved	Y	Y
RdWrPatternA	0x0	Lower-byte read and write pattern for training. When set to 0 uses default patterns.	Y	Y

RdWrPatternB	0x0	Upper-byte read and write pattern for training. When set to 0 uses default patterns.	Y	Y
RdWrInvert	0x0	Per-byte per bit invert for read and write pattern for training. When RdWrPatternA and RdWrPatternB = 0 this is unused	Y	Y
LdffMode	0x0	In LDFF mode raw PATN/PRBS sequences driven on DBI & EDC lanes. If this is set to 0 pattern follows MR settings. 0 = 1 Force DBI patterns on all lanes 1 = 1 Force non DBI patterns on all lanes	Y	Y
VrefInc	0x1	The size of the VREFStep. should always be 1 in silicon	Y	Y

5.7 Command Address Training

In order to remove skew from the CA lanes a CA training step is provided.

 Note	CA training is not supported when boot frequency clock is disabled while initializing DRAM i.e. when MsgMisc[3] is 1.
-----------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------

 Note	LPDDR5 CA training uses mode 2. Therefore, it is required that the customer has DM/DBI pins present on their PHY and connected to the DRAM device in order to perform the CA training step.
-------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

To perform Command/ Address Training, set the LPCA bit (bit [12]) of the SequenceCtrl field in the message block prior to starting the firmware.

During CA training, the DRAM is put into Command Bus Training Mode, a mode where the Command/ Address lines are sampled and fed back asynchronously on the DQ lines. A series of bit patterns are sent with varying timing (in ACTxDly CSR) to determine the valid delay range for each bit. The timing is then set to the optimal value for each bit.

This step compensates for the differences between the CK signals and the Command and Address signals to the DRAM devices.

The PHY has per lane control of delay. In the case of multiple devices or multiple ranks, the results of each device or rank must be compounded before computing the result.

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_AC[1:0]_ACTxDly_r[5:0]_p[1:0]	AC instance <N> timing for bits [5:0] in instance	Y	Y
DWC_DDRPHYA_AC[1:0]_ACTxDly_r[6]_p[1:0]	AC instance <N> timing for bit 6 in instance	N	Y

5.7.1 CA Training configuration options (CATrainOpt)

There are a couple of configuration options for CA training provided by the message block parameter CATrainOpt. The bitwise description is as follows:

Bit number	Field Name	Description
0	CAVref	0 - No CA VREF Optimization 1 - Optimize CA Vref
1	CS Training (LPDDR5 only)	0 - Disable CS Training 1 - Enable CS Training
3	Delayed clock	0 - Use delayed clock for better margins on CA lanes 1 - Use normal clock
[7:4]	ACTxDly step size	Value by which ACTxDly is incremented during CA/CS training: <ul style="list-style-type: none"> • If bit 7 is set, delay is incremented by 8, • If bit 6 is set, delay is incremented by 4, • if bit 5 is set, delay is incremented by 2 • else delay is incremented by 1

5.7.1.1 Optimize CA VREF

When CATrainOpt[0] is set, the firmware optimizes the CA VREF (MR12) for the DRAM after it has determined per lane ACTxDly delays.

To do this, the DRAM is again put into Command Bus Training Mode. A series of bit patterns are sent with varying VREF in the valid MR12 range keeping the lane delays constant for each lane. The VREF in MR12 is then set to the optimal value for each rank.



For LPDDR5 x8 mode devices, the training firmware assigns same CA Vref values to the upper and lower devices.

5.7.1.2 Delayed clock

Using delayed clock feature shall give better training margins on the CA and CS lanes. This is achieved by delaying the CK by a delay value based on the protocol as shown below:

Protocol	CK Delay
LPDDR4	1 memclk
LPDDR5	1/4 memclk

For example, delayed clock helps to cover $-1/4 t_{CK}$ to $+3/4 t_{CK}$ of total skew on CS.

The CSR affected by delayed clock feature is:

CSR Name	Description	CATrainOpt[3] = 0		CATrainOpt[3] = 1
		LPDDR4	LPDDR5	LPDDR5/LPDDR4
DWC_DDRPHYA_AC[1:0]_CKTxDly_r[7:0]_p[1:0]	AC instance <N> timing for bits [7:0] in instance	128	64	0

5.7.1.3 ACTxDly step size

This option is useful to speed up the simulations as the entire delay range can be covered using increments of multiple steps. This field should not be used when executing the firmware in silicon.

5.7.2 CS training (LPDDR5 only)

LPDDR5 uses a different driver for CS than the other lanes, so a CS training step is provided.

In order to enable CS training, set the LPCA bit of the SequenceCtrl field in the message block prior to starting the firmware and additionally, set bit[1] in message block parameter CATrainOpt as stated above.

The procedure is similar to CA training, but in this case a special pattern is sent to measure the transition between clock edges with CS. When this delay is found, the CS delay is $\frac{1}{2}$ TCK earlier than this transition.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_AC[1:0]_ACTxDly_r[7:6]_p[1:0]	AC instance <N> timing for CS bits in instance	Y	N
DWC_DDRPHYA_AC[1:0]_ACTxDly_r[9:8]_p[1:0]	AC instance <N> timing for CS bits in instance	N	Y

5.7.2.1 Reducing CS training error

It is quite possible that the CS pulse width is not ideal (1tCK) but narrow (min is 0.3tCK depending on SoC/PKG/PCB/DRAM PKG, refer to tCSIVW1). In such a case, there will be CS training error if CS pulse is assumed to be perfect (0.5tCK).

In order to overcome this timing error, a message block field CSBACKOFF is provided which realizes programmable CS delay adjustments as shown below:

$$cs_dly = dly - 2 * ONEUI(64) * CSBACKOFF / 4$$

$$CSBACKOFF = 3 : -0.375tCK$$

$$CSBACKOFF = 2 : -0.25tCK$$

$$CSBACKOFF = 1 : -0.125tCK$$

$$CSBACKOFF = \text{default} : -0.5tCK$$

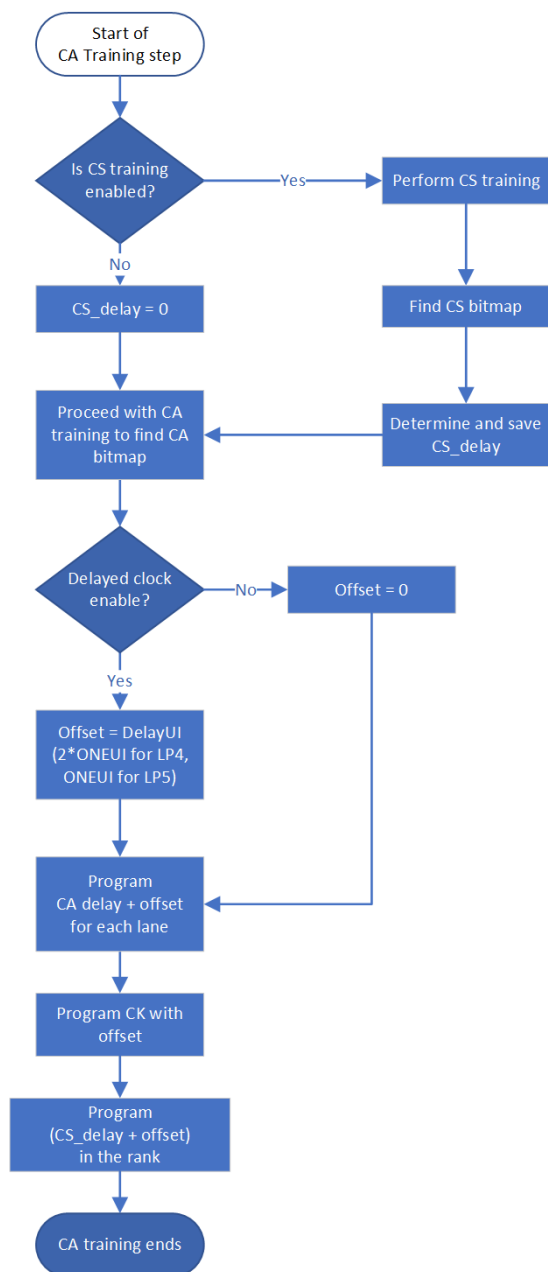
dly = the delay at which transition on CA lane is determined by CS training algorithm

cs_dly = the actual delay value to be programmed in ACTxDly for LP5 CS lanes

The user needs to program CSBACKOFF with appropriate value in case the CS pulse width is not ideal.

5.7.3 CA/CS training implementation summary

The detailed flowchart for the implementation of CA training, including CS training, is shown below. For LPDDR4, CS training is always considered as disabled.



5.7.5.2 LPDDR4X MR22 settings

For rank0, termination should be enabled by setting MR22[5:3] to 0. In case of X8 device, one the upper byte ODT should be disabled by setting MR22[7] to 1 on rank 0 and rank 1. For rank1 in the two-rank case termination should be disabled on CA and CK by setting MR22[5] to 1 and MR22[3] to 1. In all cases MR17[2:0] should be set based on PHY ODT settings.

Rank	MR22 (x16)	MR22 (x8)
0	0x06	0x86
1	0x2E	0xAE

Table 5 Example LPDDR4X MR22 settings with SOCODT=6

5.7.5.3 LPDDR4 MR22 settings

For both ranks, termination should be enabled by setting MR22[4:3] to 0. In case of X8 device, one the upper byte ODT should be disabled by setting MR22[7] to 1 on rank 0 and rank 1. The second rank's termination will be disabled by the bond pad.

Rank	MR22 (x16)	MR22 (x8)
0	0x1E	0x9E
1	0x1E	0x9E

Table 6 Example LPDDR4 MR22 settings with SOCODT=6

5.8 DRAM DCA Training (LPDDR5 only)

LPDDR5 has an internal duty cycle adjustment (DCA) setting and a duty cycle monitor (DCM). The DRAM DCA training adjusts the DCA setting based on the DCM output.

The DRAM DCA training is performed when bit[6] of the SequenceCtrl field in the message block is set prior to starting the firmware. The firmware records flip0 and flip1 results for each DCA setting. Refer to JEDEC section 4.2.7 for details on DCM measurement procedure.

Once flip0 and flip1 results for each DCA setting are available, averaging is done for each byte in every rank.

An example

PMU10: DCA val: -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7

```
-----
PMU10: DB0 fip0      0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
PMU10: DB0 fip1      0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
PMU10: DB1 fip0      0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
PMU10: DB1 fip1      0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
-----
```

0 or **1** : is a transition

0 or 1 : is not a transition

PMU10: Rank 0, DCAUpper calculated as = 4

PMU10: Rank 0, DCALower calculated as = 3

There is no PHY CSR trained by this step but MR30 as below:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
MR30[3:0]	Duty Cycle Adjustment Lower	N	Y
MR30[7:4]	Duty Cycle Adjustment Upper	N	Y

5.9 PHY DCA Training (LPDDR5 only)

The PHY has internal duty cycle adjustment settings that are designed to be a fine tuning to the DRAM duty cycle adjustment provided by the DRAM. To train this, an eye with measurement is performed after initial training and the duty cycle adjustment is made to maximize the eye width.

The PHY Read DCA training is performed when bit [7] of the SequenceCtrl field in the message block is set. PHY Write DCA training is performed when bit [8] of the SequenceCtrl field in the message block is set prior to starting the firmware.

During PHY Read DCA training, 1D eye scan of 2UI width is performed for one PHY Read DCA control setting. From the Rx eye data generated, minimum eye width is determined and recorded for each Dbyte. The same the same procedure is done for the next fine PHY Read DCA control setting. Once all the DcaCtrl settings are done, the DCACtrl setting that gives the minimum eye width is programmed in the RxWCKDcaCtrlT/C CSRs.

Similarly, during PHY Write DCA training, 1D eye scan of 2UI width is performed for one PHY Write DCA control setting. From the Tx eye data generated, minimum eye width is determined and recorded for each Dbyte. Once all the fine PHY Write DCA control settings are done, the DCACtrl setting that gives the minimum eye width is programmed in the TxWCKDcaCtrlT/C CSRs

A simplified training pattern is used in these steps as we don't want signal integrity effects distorting the error caused by duty cycle. In Read DCA we use fixed patterns using Read DQ Calibration commands as

write are not trained at that point in time. In Write DCA we use fixed patterns using Write FIFO and Read FIFO commands. The fixed pattern is the same one used in Read SI Friendly training.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_RxWCKDcaCtrlTTg0_p[1:0]	Rx Timing group 0 Duty Cycle Distortion correction for true WCK (PHY Rd DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_RxWCKDcaCtrlTTg1_p[1:0]	Rx Timing group 1 Duty Cycle Distortion correction for true WCK (PHY Rd DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_RxWCKDcaCtrlCTg0_p[1:0]	Rx Timing group 0 Duty Cycle Distortion correction for compliment WCK (PHY Rd DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_RxWCKDcaCtrlCTg1_p[1:0]	Rx Timing group 1 Duty Cycle Distortion correction for compliment WCK (PHY Rd DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWCKDcaCtrlTTg0_p[1:0]	Tx Timing group 0 Duty Cycle Distortion correction for true WCK (PHY Write DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWCKDcaCtrlTTg1_p[1:0]	Tx Timing group 1 Duty Cycle Distortion correction for true WCK (PHY Write DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWCKDcaCtrlCTg0_p[1:0]	Tx Timing group 0 Duty Cycle Distortion correction for compliment WCK (PHY Write DCA)	N	Y
DWC_DDRPHYA_DBYTE<N>_TxWCKDcaCtrlCTg1_p[1:0]	Tx Timing group 1 Duty Cycle Distortion correction for compliment WCK (PHY Write DCA)	N	Y

Where <N> designates the corresponding DBYTE instance.

5.10 Training Loops

The purpose of the training loops is to improve the system results when the differential signals have their duty cycle compensated. After duty cycle compensation it is likely that the optimal trained point will change from the previously trained point. In the default training flow there are retrained steps after each duty cycle adjustment (DCA) training. Each retraining loop repeats all previously run affected training steps. The training loops are enabled by default if the corresponding DCA training step is run. You can disable the loop while having the DCA training enabled by setting the appropriate disable bit in the DisableTrainingLoop field in the message block.

Retraining Loop 1 is run after DRAM DCA. In Loop 1 the following steps are retrained; Write Leveling, RxEn, and Read Training with DQ Calibration.

Retraining Loop 2 is run after PHY Read DCA. In Loop 2 the following step is retrained; Read Training with DQ Calibration.

Retraining Loop 3 is run after PHY Write DCA. In Loop 3 the following step is retrained; Write Training.

Field Name	Description	Protocol	
		LPDDR4	LPDDR5
DisableTrainingLoop	Set these fields to skip training steps in their respective loops	N	Y

By default, we run all training in each enabled step. Each retraining step can be selectively disabled based on system requirements for performance or run time improvements. The following is a description of each bit of DisableTrainingLoop[5:0] when that bit is set to one.

0. Disable all retraining stages in loop 1: Write Leveling, RxEn, and Read Training with DQ Calibration.
1. Disable all retraining stages in loop 2: Read Training with DQ Calibration.
2. Disable all retraining stages in loop 3: Write Training.
3. Disable Write leveling training step in loop 1.
4. Disable RxEn training step in loop 1.
5. Disable Read Training with DQ Calibration training step in loop 1.

5.11 TxDFE Training (LPDDR5 only)

LPDDR5 supports a vendor optional TxDFE setting that improves the eye size. The TxDFE training tries all of the supported TxDFE settings and finds the optimal setting. See JEDEC specification for more details about the DRAM DFE setting

To perform TxDFE training, set the TxDFE bit of the SequenceCtrl field of the message block prior to starting the firmware.

As the DQ training, TxDFE training is done by performing a full 2D scan, as performed in 2D write training. Training firmware repeats the 2D scan for all valid DRAM VREF setting and calculate the area of eye. The smallest area of all lanes in each byte is retained. This process is repeated for each DFE setting, and the DFE setting that results in the largest minimum eye area is used.

This step affects MR24 as below:

Trained MR	Description	Protocol	
		LPDDR4	LPDDR5
MR24[2:0]	DFE Quantity for Lower Byte	N	Y
MR24[6:4]	DFE Quantity for Upper Byte	N	Y


TxDfE configuration options in message block:

Field Name	Description	Protocol	
		LPDDR4	LPDDR5
VrefInc	This controls the VREF Increment size for 2D training	N	Y
BitTimeControl	Input for the amount of data bits per DQ before deciding if any specific voltage and delay setting passes or fails. Every time this input increases by 1, the number of 1D/2D data comparisons is doubled	N	Y

5.12 Strobeless Training

To perform Strobeless Training, set the RxEn bit of the SequenceCtrl field of the message block and configure MR20 [1:0] to have RDQS_T and RDQS_C disabled prior to starting the firmware. It is required that the data rate be equal to or less than 1600 Mbps in order to use this mode.

Strobeless training is used to determine the round-trip latency, between the PHY and the DRAM. This step is run as a replacement to both RxEn and Read training so neither of those steps are executed. The PHY internally generates a digital strobe that is the same frequency as DQS would have been. The timing of this strobe is then adjusted to have the strobes rising edge in the center of the incoming data. All DBYTE instances and lanes can be trained in parallel, but each rank and channel are trained independently.

 Note	When RDQS Strobeless mode is enabled no RxDqs training including VREF training is performed.
-------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_RxDigStrbDlyTg0_r[8:0]_p[1:0]	Timing group 0 Rx Digital Strobe Coarse Delay	N	Y
DWC_DDRPHYA_DBYTE<N>_RxDigStrbDlyTg1_r[8:0]_p[1:0]	Timing group 1 Rx Digital Strobe Coarse Delay	N	Y

Where <N> designates the corresponding DBYTE instance.

5.13 RxClk Training

To perform Read training (RxClk), set the RdDqs bit of the SequenceCtrl field of the message block prior to starting the firmware.

Read training is used to determine the DRAM DQ to PHY RX latency. This latency must be known so that the PHY's receivers are activated by the time DRAM to PHY latency plus read latency has elapsed. All the DBYTE instances can be trained in parallel, but each rank, channel, and clock(true and compliment) are trained sequentially. We also train the PHY VREFDAC when 2D training is not disabled.

Read training is done two times using different methods of receiving transactions. Read DQS SI friendly uses the DRAM DQ calibration registers and DQ calibration commands to generate fixed pattern read traffic. Full read training is done by sending bursts of write FIFO transactions to the DRAM and then reading back that data with bursts of read FIFO transactions. The FIFO transactions use PRBS23 data and can produce DBI or non DBI like patterns. The first step is to find the estimated tPHYDQS2DQ by reading the RxReplica circuit and use this with known preamble length to estimate the Rx DQS delay. In SI friendly the firmware scans 2UI skipping every other point around our estimated Rx DQs delay value for a total scan size of 2UI and a range of 4UI. In full read training the firmware uses the median values of all the lanes in each DByte to generate the estimated starting location. Then we scan 1UI around our estimated value for a total scan range and width of 2UI. Our data scan stores the error rate at each delay and voltage setting. If 2D training is enabled, then we repeat this scan for all valid VREFDACs and VREF settings.

In the case of 2D disabled we pick the center point of our eye within the 2UI scan to provide maximum margin.

In the case of full 2D the data is turned into an eye contour as the data is collected. This contour is saved for all lanes, DBYTEs, ranks, channels, clocks (true and compliment), and VREFDACs. Then the data is compounded across shared resources, for read training there are shared VREFDACs for each rank and shared VREFDacs per DFE setting. Then the data is sent to the 2D center finding Algorithm to select the VREF and delay with the optimal settings.

5.13.1.1 Training Patterns

We provide the ability to set a custom pattern for the DQ Calibration and modify the PRBS pattern. The read traffic pattern is controllable using 4 message block fields RdWrPatternA, RdWrPatternB, RdWrInvert, and LdffMode. Together RdWrPatternA and RdWrPatternB allow the user to set a 16bit fixed pattern to be used for DQ Calibration read training. The default pattern of DQ calibration is 0xA536. The RdWrInvert pattern is a mask applied to DQ Calibration pattern when DBI is enabled. The default pattern of the invert mask is 0xAA. LdffMode allows the user to force the pattern to conform to DBI or non DBI patterns where applicable. By default, if DBI is enabled for TX or RX then all read and write training patterns are DBI. Setting LdffMode[0]=1 will force DBI patterns regardless of MR settings. Setting LdffMode[1]=1 will for non DBI patterns on all lanes regardless of MR settings.

Field Name	Description	Protocol	
		LPDDR4	LPDDR5
RdWrPatternA	Lower-byte read and write pattern for training	Y	Y
RdWrPatternB	Upper-byte read and write pattern for training	Y	Y
RdWrInvert	Per-byte per bit invert for read and write pattern for training	Y	Y
LdffMode	Allows user to force DBI or non DBI on read and write patterns.	Y	Y

5.13.1.2 RxClk Training Extra Options

There are options that can be used to accelerate RxClk training in the Misc field. They are described in the following sections:

5.13.1.2.1 2UI SI Friendly Scan Mode Misc[3]

The Misc[3] setting controls the SI Friendly scan size. When set to zero, SI Friendly scans increase delay step size to compensate for larger skews with somewhat less accuracy and scan 4ui of data. When set to 1, Si Friendly scans reduce the step size to 1 and improves accuracy for SI Friendly results.

5.13.1.2.2 Pre-Compute RxClk Coarse bit Misc[5]

By default, the final RxClk training tests the final trained result with even and odd coarse bits. This can provide better drift range at the expense of training time. Setting Misc[5]=1 will precompute the best coarse based on the SI Friendly RxClk results. This feature should only be used at or above 3200 Mbps.

5.13.1.2.3 Single RxClk scan in SI Friendly Read Misc[6]

By default, the SI Friendly RxClk training tests both true and complement RxClk delay in order to get optimal strobe position for subsequent steps. Setting Misc[6]=1 will use the RxClk_t scan result for RxClk_c delay for SI friendly training.

5.13.2 Coarse Bit Selection

The implementation of RxClk2UIDlyTg allows for two representations of the same amount of delay. The subfields of RxClk2UI are RxClkDlyTgFine[6:0] where one unit of delay is one-sixtyfourth of a UI = UI/64 and RxClkT2UIDlyTgCoarse[9:7] where one unit of delay is one UI. RxClk fine can represent 0/64 to 127/64 steps worth of delay. This means that a setting of 1 coarse 32 fine is equivalent to 0 coarse 96 fine, both representing 96/64 of a UI of delay.

To ensure the optimal trained point and flexibility in PPT we perform read training with PRBS twice to cover ranges with both sets of coarse values. The first scan is performed with the coarse bits set to odd values and the second scan is performed with even values. After both scans have been performed we apply a cost function to each lane. We pick the run that has the set of fine values closest to 64.

$$c = |fine\ bits - 64|$$

The set of values to be considered is a function of the Rx DFE settings. We take all the settings and find the maximum of the cost values for each set of even values and set of odd values. The final decision is which set of even or odd has lowest cost.

$$set = \min \left(\begin{matrix} \max(\in c_{odd}) \\ \max(\in c_{even}) \end{matrix} \right)$$

DFE settings	RxClk values considered per lane, per dbyte
Static 0	RxClkC Rank0
Static 1	RxClkC Rank1
DFE On	RxClkT Rank0 RxClkT Rank1
Per Rank Static	RxClkC specified rank RxClkT specified rank

The order of scans for RxClk are both ranks, then both coarse, and then both true and complement clocks;

1. RxClkC, Odd coarse, Rank 0
2. RxClkC, Odd coarse, Rank 1
3. RxClkC, Even coarse, Rank 0
4. RxClkC, Even coarse, Rank 1
5. RxClkT, Odd coarse, Rank 0
6. RxClkT, Odd coarse, Rank 1
7. RxClkT, Even coarse, Rank 0
8. RxClkT, Even coarse, Rank 1

At the end of all RxClk runs we run the above algorithm for all lanes and dbytes to decide which RxClk setting to use.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_RxClkT2UIDlyTg0_r[8:0]_p[1:0]	Timing group 0 timing for Receive clock true delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_RxClkT2UIDlyTg1_r[8:0]_p[1:0]	Timing group 1 timing for Receive clock true delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_RxClkC2UIDlyTg0_r[8:0]_p[1:0]	Timing group 0 timing for Receive clock compliment delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_RxClkC2UIDlyTg1_r[8:0]_p[1:0]	Timing group 1 timing for Receive clock compliment delay	Y	Y
DWC_DDRPHYA_DBYTE<N>_VREFDAC[3:0]_r[8:0]_p[1:0]	PHY RX VREF DAC control for RxDQ	Y	Y

Where <N> designates the corresponding DBYTE instance.

This step compensates for the delay for the Read data to reach each DBYTE and have the data read correctly. This delay includes parameters such as tPHYDQS2DQ and related parameters. This delay does not include the read CAS latency, which is known ahead of time.

A delay and VREF is calculated for each lane from each timing group by finding the eye of passing settings and placing the trained result to maximize the margins.

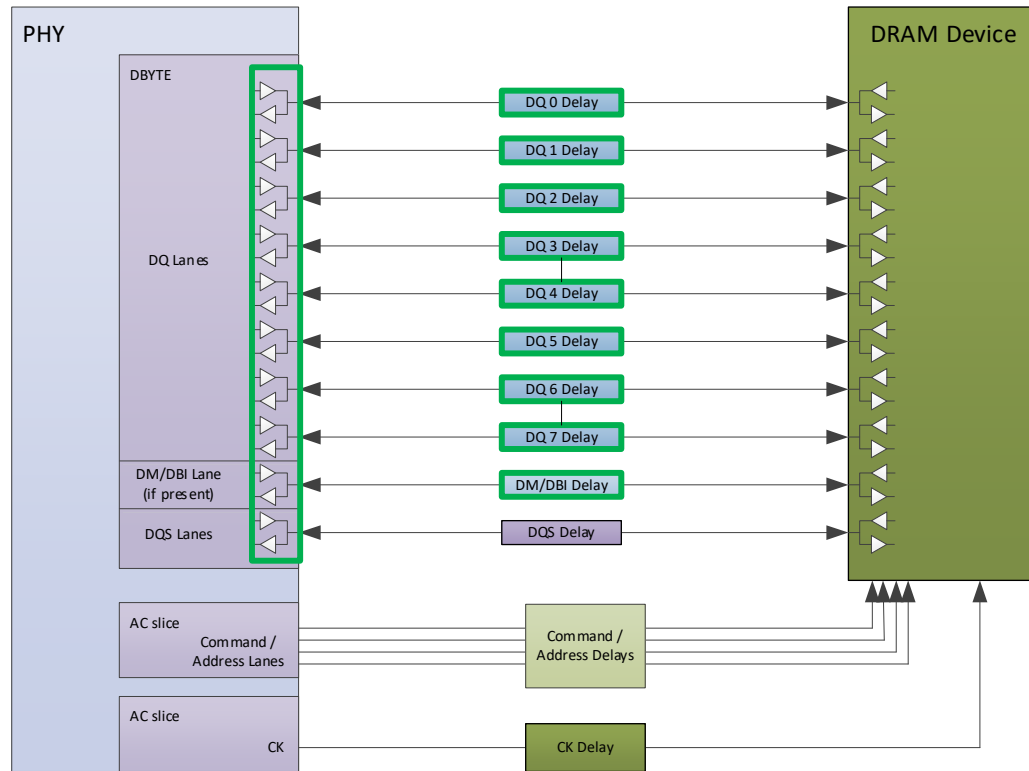


Figure 6 PHY-DRAM Interface (LPDDR4)

The message block settings that affect this step are:

Message Block Field Name	Default	Description	Protocol	
			LPDDR4	LPDDR5
DisableTrainingLoop	0x7	Set these fields to skip training steps that are looped [0] = 1 disable all retraining stages in loop 1: Write leveling, RxEn, Read Dq Cal [1] = 1 disable all retraining stages in loop 2: Read DQ Cal and, DRAM DCA [2] = 1 disable all retraining stages in loop 3: Write Training	N	Y
Train2DMisc	0x0	2D Training Miscellaneous Debug Control Train2DMisc[0]: Print Verbose 2D Eye Contour 0 = Do Not Print Verbose Eye Contour 1 = Print Verbose Eye Contour Train2DMisc[1]: Print Verbose Eye Optimization Output 0 = Do Not Print Verbose Eye Optimization Output 1 = Print Verbose Eye Optimization Output	Y	Y
RX2D_Delay_Weight	0x0	[0-7] 0 ... 255 During RX 2D training when finding an eye center the delay and voltage components are weighed such that the combined margin is delay margin * RX_Delay_Weight2D + voltage margin * RX_Voltage_Weight2D. Either weight may be zero but if both are zero each weight is taken to have a value of one.	Y	Y
RX2D_Voltage_Weight	0x0	[0-7] 0 ... 255 During RX 2D training when finding an eye center the delay and voltage components are weighed such that the combined margin is delay margin * RX_Delay_Weight2D + voltage margin * RX_Voltage_Weight2D. Either weight may be zero but if both are zero each weight is taken to have a value of one.	Y	Y
Reserved09	0x0	Reserved09[0-2]: bitTimeControl Input for the amount of data bits 1D/2D WFF/RFF per DQ before deciding if any specific voltage and delay setting passes or fails. Every time this input increases by 1, the number of 1D/2D data comparisons is doubled. The 1D/2D run time will increase proportionally to the number of bit times requested per point. 0 = 2 ⁰ times of basic amount (default behavior) 1 = 2 ¹ times of basic amount 2 = 2 ² times of basic amount ... 7 = 2 ⁷ times of basic amount Reserved09[3-7]: RFU, must be zero	Y	Y
RX2D_TrainOpt	0x0	Reserved	Y	Y
RdWrPatternA	0x0	Lower-byte read and write pattern for training. When set to 0 uses default patterns.	Y	Y

RdWrPatternB	0x0	Upper-byte read and write pattern for training. When set to 0 uses default patterns.	Y	Y
RdWrInvert	0x0	Per-byte per bit invert for read and write pattern for training. When RdWrPatternA and RdWrPatternB = 0 this is unused	Y	Y
LdffMode	0x0	In LDFF mode raw PATN/PRBS sequences driven on DBI & EDC lanes. If this is set to 0 pattern follows MR settings. 0 = 1 Force DBI patterns on all lanes 1 = 1 Force non DBI patterns on all lanes	Y	Y
VrefInc	0x1	The size of the VREF Step.	Y	Y

5.14 2D Training

Optimal RxDqDly, RxVREF, TxDqDly and TxVREF values are found during 2D training. In this phase patterns are written and read to memory devices from the PHY and checked for errors.

Data is collected separately on each lane of each DBYTE of each rank and channel. The collected eye data is then analyzed, and delay and voltage thresholds calculated.

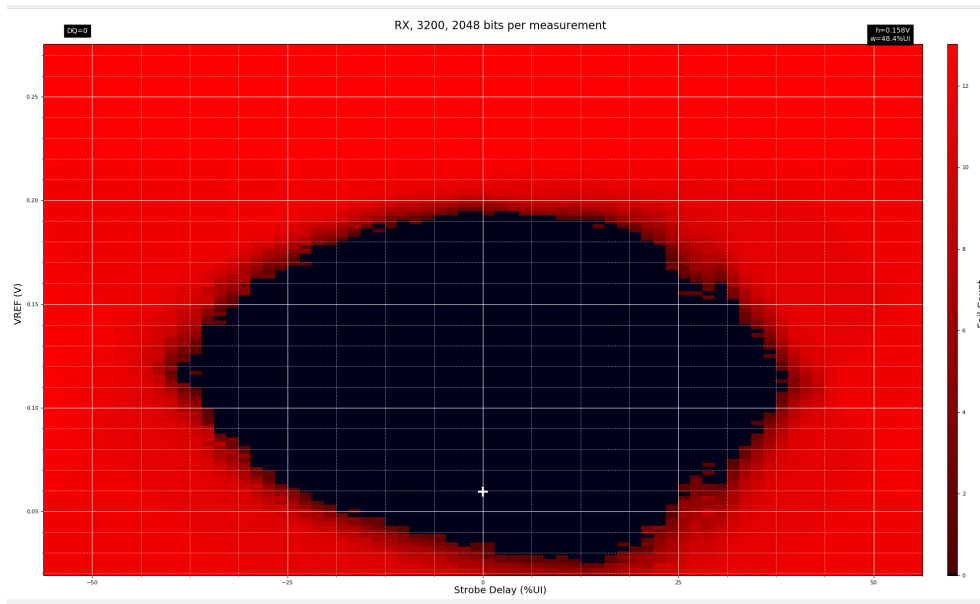


Figure 7 Signal EYE for a single lane of a single rank. Each point corresponds to a sampling delay and threshold voltage. Red are failing points and black are passing points. The horizontal axis represents the sampling delay and the vertical axis the threshold voltage.

The optimal training point is found by calculating the largest empty circle and using the point at the center of the circle.

The two dimensions are of different physical properties and to accommodate differing margin requirements the two dimensions can be given different weights. Using different weights corresponds to finding the center of the largest empty oval rather than circle. (Figure 9) Increasing the weight of a parameter will force the training algorithm to proportionately amplify the consideration of that parameter.

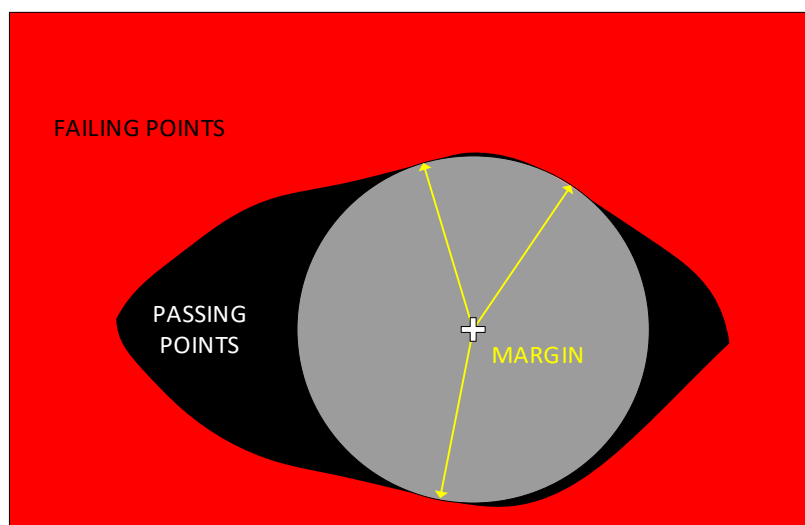


Figure 8 Trained EYE center. The optimal sampling delay and threshold voltage are found by maximizing the margin (largest empty circle). The margin is the distance to the nearest failing point. In this diagram 2D training weights are the same for voltage and delay.

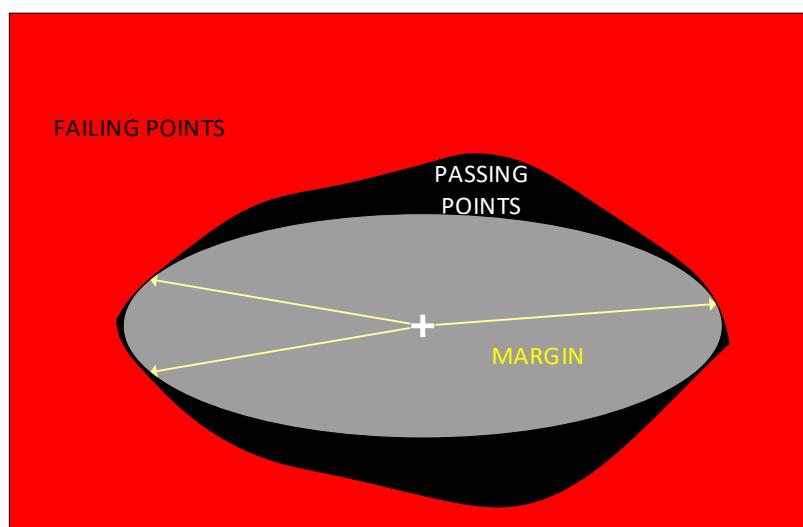


Figure 9 Trained EYE center using different weights for delay and voltage threshold.

5.14.1 Rx 2D Training

Rx training margins can be adjusted by modifying `RX2D_Delay_Weight` and `RX2D_Voltage_Weight` in the message block. Setting them both to the same value will weigh both dimensions equally. The units for delay are 64ths of a UI. See PHY Databook to determine the voltage units.

In certain Rx DFE modes separate eye data is gathered for each rank and previous bit history of a lane. An optimal training point is then calculated considering shared resources across all involved eyes. RX2D training weights are used in the optimization calculations.

5.14.1.1 Rx DFE Training

The LPDDR54 PHY supports an unrolled, 1 tap DFE receiver mode. The implementation of this receiver requires training two VREFDACs separately using history filtering. Including the separate receiver for the true and complement strobes, the number of receivers that need to be trained is 4 per lane. In order to enable DFE, the RxDfeMode field in PHYInit needs to be set to 4. See section 5.1.6.2 for details on this setting.

The order that the DACS are trained in are:

1. VrefDac1; Which receives RxClk complement previous bit history 0.
2. VrefDac3; Which receives RxClk complement previous bit history 1.
3. VrefDac0; Which receives RxClk true previous bit history 0.
4. VrefDac2; Which receives RxClk true previous bit history 1.

Due to the data eye shape specific to DFE, the hill climb optimization method may not be suitable for picking the correct VREF and delay. It is recommended that weighted mean optimization be used for RxDFE. See section 5.14.3.2.10

5.14.1.2 Rx Receiver Mode (DfeModeCfg)

A PHYInit setting RxDfeMode exists that corresponds to the PHY CSR DfeModeCfg. DFE is controlled by this setting, but not all settings enable bit history. RxDfeMode=4 enables DFE (bit history). RxDfeMode=2 enables VREF that is independent for each rank, but there is no DFE. Table 1 explains the different settings.

Table 7 RxDfeMode Settings

DfeMode	Description
0	VREF shared between ranks
1	Not Supported
2	Independent VREF per rank
4	DFE mode, VREF shared between ranks

5.14.2 Tx 2D Training

Tx training margins can be adjusted by modifying TX2D_Delay_Weight and TX2D_Voltage_Weight in the message block. Setting them both to the same value will weigh both dimensions equally. The units for delay are 64^{ths} of a UI. See Relevant JEDEC Specifications to determine the voltage units.

Depending on protocol and memory device architecture multiple lanes will share voltage thresholds. In these cases, the eyes of each lane are compounded and a single shared optimal threshold and individual per lane delays are computed. TX2D training weights are used in the optimization calculations.

5.14.3 2D Training Options

5.14.3.1 2D Training Verbose Messages

There are two controls which independently can be used to enable verbose 2D training messages. **Enabling any of the verbose modes will significantly slow down training because of the messaging overhead.**

Setting message block field Train2DMisc bit 0 turns on verbose output messages for the training eyes in different stages of processing.

Setting message block field Train2DMisc bit 1 turns on verbose output for the various optimization routines.

5.14.3.2 VREF Step Control

By default, 2D training scans the entire valid range for VREF DAC values. For receive DACs this is 0 to 127. For Tx this is protocol dependent, but covers the entire range supported by JEDEC.

There are two ways to control the granularity of VREF during 2D training. The following subsections covers both.

5.14.3.2.1 VrefInc

VrefInc is a message block field that increases the step size for all 2D steps. The entire range is scanned in steps of VrefInc for Tx and Rx. If VrefInc is 0, and 2D training is enabled the step size is assumed to be 1.

5.14.3.2.2 Advanced VREF Step Control

In order to give finer grained control over the size and resolution of VREF scans for 2D training, a more detailed collections of setting is provided. A list of message block fields and descriptions for this feature are included in Table 1 Message Block settings control the VREF step size on a finer grained basis than the one global setting. Rx Step size is broken in to two sets of settings, one for pattern mode steps that use read DQ calibration patterns, and Prbs for ones that use Prbs patterns (such as final RxClk training).

To use a particular set of settings, the “Step” field for that setting should be nonzero. Otherwise the VrefInc setting is used across the entire VREF range. If all settings are zero, including VrefInc, the

maximum supported range and step size of 1 is assumed. Details of the settings for various steps for Rx outlined in subsequent sections

5.14.3.2.3 Advanced VREF Step Control: Rx Prbs and Pattern Mode

There are two settings for Rx VREF one controls Rx steps that have DQ Calibration Patterns such as RxClk Si Friendly and PHY DCA training. The other controls Rx steps that use PRBS patterns such as final RxClk training.

5.14.3.2.4 Advanced VREF Control: Rx DFE Settings

For Rx step control settings there are two different settings one for Dfe0 and one for Dfe1.

The following table explains which setting should be used to control sweeps in various situations

RxDfeMode	Dfe0 Settings	Dfe1 Settings
0	Applies to all Rx Sweeps	Not Used
1	Not Used	Applies to all Rx Sweeps
2	Applies to Rank0 Rx Sweeps	Applies to Rank1 Rx Sweeps
4	DFE Previous History = 0 DAC0, DAC1	DFE Previous History = 1 DAC2, DAC3

When using RxDfeMode=2, the DFE0 setting should be used to control the VREF step of rank0 and the DFE1 settings should be used to control the VREF step for rank1.

5.14.3.2.5 Advanced VREF Control: Tx Settings

There is one set of step control settings for Tx VREF control that are activated when TxVrefStep is nonzero. This controls the VREF range and step size for Tx steps such as Tx DQ training and Tx DFE training.

Table 8 Variable Step Size Message Block Fields.

Message Block Field Name	Description
RxVrefStartPatDfe0	Starting VREF Value for Rx Training for DFE0 for Pattern Mode
RxVrefStartPatDfe1	Starting VREF Value for Rx Training for DFE1 for Pattern Mode
RxVrefStartPrbsDfe0	Starting VREF Value for Rx Training for DFE0 for Prbs Mode
RxVrefStartPrbsDfe1	Starting VREF Value for Rx Training for DFE0 for Prbs Mode
TxVrefStart	Starting VREF Value for Tx Training for Prbs Mode
RxVrefEndPatDfe0	Ending VREF Value for Rx Training for DFE0 for Pattern Mode
RxVrefEndPatDfe1	Ending VREF Value for Rx Training for DFE1 for Pattern Mode
RxVrefEndPrbsDfe0	Ending VREF Value for Rx Training for DFE0 for Prbs Mode
RxVrefEndPrbsDfe1	Ending VREF Value for Rx Training for DFE0 for Prbs Mode
TxVrefEnd	Ending VREF Value for Tx Training for Prbs Mode
RxVrefStepPatDfe0	VREF Step Value for Rx Training for DFE0 for Pattern Mode
RxVrefStepPatDfe1	VREF Step Value for Rx Training for DFE1 for Pattern Mode
RxVrefStepPrbsDfe0	VREF Step Value for Rx Training for DFE0 for Prbs Mode
RxVrefStepPrbsDfe1	VREF Step Value for Rx Training for DFE0 for Prbs Mode
TxVrefStep	VREF Step Value for Tx Training for Prbs Mode

5.14.3.2.6 SI Friendly Offset Control

In some cases, the SI friendly training can choose a delay setting that doesn't work for Tx PRBS training. To alleviate this situation there is a setting that will place the RxClk training at a given offset from the left edge of the eye, instead of at the trained center. SIFriendlyDlyOffset[7:1] determines the number of 64ths of a UI to offset the RxClk training from the left edge of the eye. If the resulting trained point is not in the passing region of the eye training will fail.

5.14.3.2.7 Bit Time Control

The number of read and write transactions when scanning each delay and voltage setting can be altered for the Read training, Write training, and TxDFE steps. The base number of transactions is a multiplier applied to the maximum fifo depth and the burst length. The nominal value of the multiplier is described in Table 2. This multiplier can be optional replaced with the BitTimeControl message block field. If the BitTimeControl field is set to a non-zero value, then the multiplier will be replaced with $2^{BitTimeControl} - 1$.

Table 9 Training step multipliers with BitTimeControl = 0.

Training Step	Conditions	Multiplier
Read Training	Silicon	1
	Simulation	5
	Silicon with DFE enabled	5
	Simulation with DFE enabled	25
Write Training	Silicon	1
	Simulation	5
TxDFE	N/A	1

Table 10 Base number of bits per point

Protocol	Bust Length	Max Fifo Depth	Bits per point
LPDDR4	16	5	80
LPDDR5	16	8	128

Equations for total number of bits per point.

With BitTimeControl = 0:

$$TotalBits = Multiplier * BaseBitsPerPoint$$

With BitTimeControl != 0:

$$TotalBits = (2^{BitTimeControl} - 1) * BaseBitsPerPoint$$

5.14.3.2.8 2DMisc Iteration Count

During 2D training, an optimization algorithm is used to search for the point in the signal eye that has the greatest noise margin. The Iteration Count field of the Train2DMisc can be used to control the number of iterations that the algorithm uses. This may affect the speed and quality of the training.

Fewer iterations will make the training faster at the possible cost of a better solution than would otherwise be found. (A better solution is one that has higher noise margins.) More iterations will take longer but may be more likely to find a better solution. Early termination is a feature in which the algorithm terminates when forward progress ceases or a cap on the number of iterations is reached, whichever happens first. The iteration count can only be set to even values.

Train2DMisc[5:2]: Iteration Count for Optimization Algorithm

Iteration count = Train2DMisc[5:2] << 1

```

b0000 - Iteration count == 0 is default, count = 16
b0001 - Iteration count == 2 early termination with cap at 16
b0010 - Iteration count == 4
b0011 - Iteration count == 6
b0100 - Iteration count == 8
...
b1101 - Iteration count == 26
b1110 - Iteration count == 28
b1111 - Iteration count == 30

```

5.14.3.2.9 Number of Seeds to Use for 2D Optimization Algorithm

If largest empty circle (LEC) is used as the 2D training optimization algorithm the number and location of the seeds for the optimization search can be controlled using message block field Train2DMisc[7:6]. This control can be used to avoid training to sub-optimal points.

In general, the more seeds used the greater the possibility of a better training point. Using more seeds will slow the training times.

```
Train2DMisc[7:6]: Number of Seeds for Optimization Algorithm
    b00 = 2 seeds, left and right of center, default behavior
    b01 = 1 seed, center seed
    b10 = 2 seeds, left and right of center
    b11 = 3 seeds, left, center and right
```

If a training eye is pinched at the center in the voltage dimension, as may occur with a signal reflection, using seeds to the left and right of center may produce a better training point.

5.14.3.2.10 2DMisc Weighted Mean

There may be times when the training results of the default margin maximization algorithm (Largest Empty Circle) is not the solution that is desired. An alternative is to use a weighted mean algorithm by setting Train2DMisc[10] to 1. This will train to a solution that is the weighted mean (2D center of mass) of the open area of the eye.

The weighted mean algorithm is faster than the LEC algorithm but may produce poor results in some situations. One situation where results of weighted mean may be less than desirable is in the case of an eye pinched at its center in the voltage dimension. This may occur due to reflections on a signal line.

The weighted mean algorithm may improve the training with DFE enabled.

The weighted mean algorithm may be modified by setting Train2DMisc[12:11].

```
Train2DMisc[12:11]: Weighted Mean Mode
    b00 = Regular weighted mean; trained_delay = f(Vref); default
    b01 = Square of voltage; trained_delay = f(Vref * Vref)
    b10 = Log of voltage; trained_delay = f( ln2(Vref) )
    b11 = RESERVED
```

Using the SQUARE mode will weigh larger eye openings over smaller openings. In an asymmetrical eye this will tend to move the trained delay toward the larger opening.

Using the LOG mode will reduce the weight relative weight of larger eye openings. This will tend to move the trained delay toward the center of the eye on the delay axis.

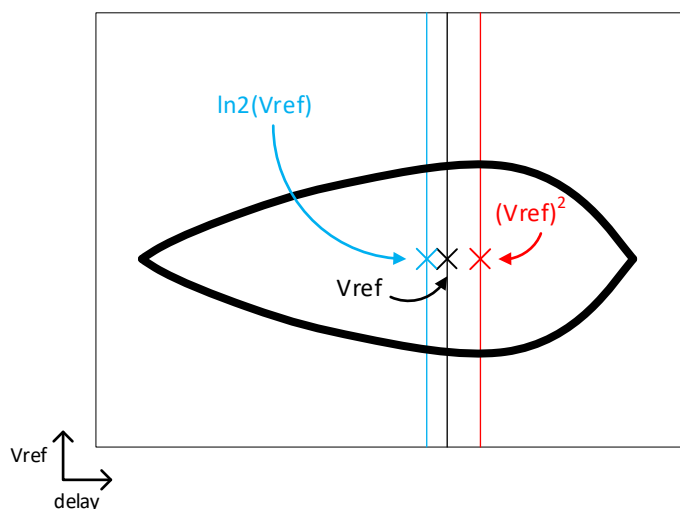


Figure 10 Illustration of possible effect of weighted mean modes on eye training.

5.14.3.2.11 TruncV

The TruncV feature is enabled by setting SIFriendlyDlyOffset[0] to 1 in the message block. This feature affects 2D training when the training eye is truncated at lower voltages. Under normal circumstances only delay and voltage points that have been measured are used to calculate the optimal training point. When this feature is enabled the lower boundary of measured voltages is not considered in voltage margin calculations and training. The trained point is always in the measured and passing part of the eye.

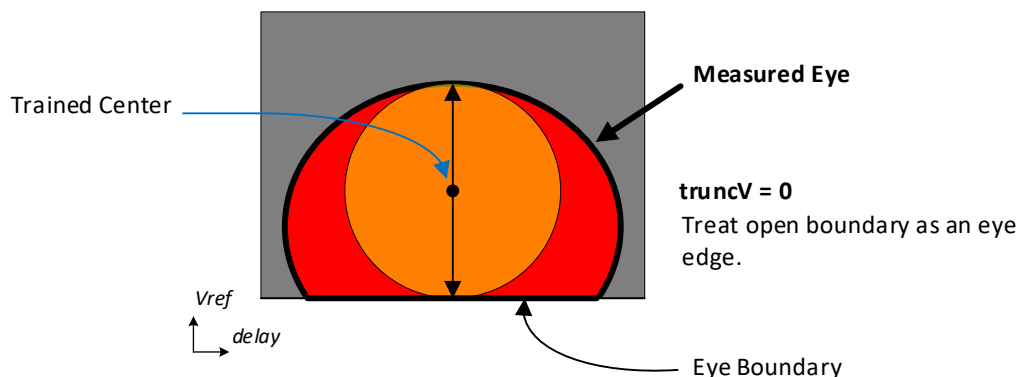


Figure 11 Training of truncated eye with TruncV feature disabled.

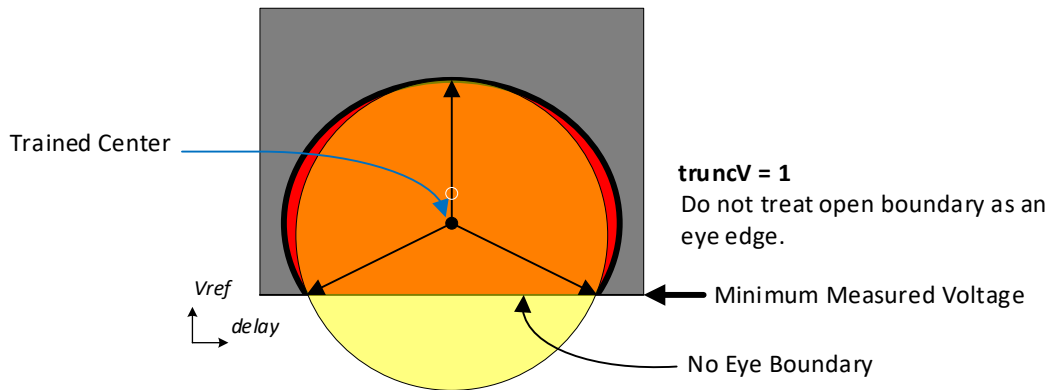


Figure 12 Training of truncated eye with TruncV feature enabled.

5.14.4 2D training data dumps

When operating in full debug mode (HdtCtrl = 0x4), the training firmware will print summarized eye contours as it is processing them. Here is an example of a typical printed eye contour:

```
PMU4: pmu_2Dtrain() - EXPORT PROCESSED EYES
PMU4: ----- 2D Read Scanning TG 0 (CS 0x0) Lanes 0x0ff -----
PMU4: -- DB0 L0 -- centers: anchor delay = 24, optimal delay offset = 21 voltage = 61
PMU4: 78 79 84 89 94 97 100 103 106 108 105 102 100 94 90 84 84 78 71 0 0 0 0 0 0 0 0 0 0 0 0
PMU4: 66 56 51 45 41 36 35 32 26 24 24 28 30 30 34 40 42 50 56 127 127 127 127 127 127 127 127 127 127 127
```

Here is a line-by-line breakdown of how to interpret the output:

```
PMU4: pmu_2Dtrain() - EXPORT PROCESSED EYES
```

This indicates that the output is for eyes that have been processed to find the optimal trained centers. Setting message block field Train2DMisc bit 8 will also print eye contours prior to processing. This will take additional time, but can be useful when debugging an unrecoverable errors that prevent the post-processed contours from being printed.

```
PMU4: ----- 2D Read Scanning TG 0 (CS 0x0) Lanes 0x0ff -----
```

This message indicates that these are RX (“2D Read Scanning”) eyes for timing group 0 (“TG 0”) of the device. For TX data, the message would start with (“2D Write Scanning”). For timing group 1, the message would read “TG 1”.

```
PMU4: -- DB0 L0 -- centers: anchor delay = 24, optimal delay offset = 21 voltage = 61
```

This message specifies the dbyte (“DB0”) and lane (“L0”) for the eye contour. The anchor delay is the decimal starting value of the first points in the contour. The “optimal delay offset” is the trained delay center relative to the anchor, and the “voltage” value is the trained voltage center.

```
PMU4: 78 79 84 89 94 97 100 103 106 108 105 102 100 94 90 84 84 78 71 0 0 0 0 0 0 0 0 0 0 0 0
PMU4: 66 56 51 45 41 36 35 32 26 24 24 28 30 30 34 40 42 50 56 127 127 127 127 127 127 127 127 127 127 127
```

These two rows of values are the top and bottom of the eye contour. Each point indicates the maximum (top) or minimum (bottom) voltage where there were no bit errors at the given delay offset. Internally in the firmware, the delay scans are 64 values wide. To save time only 31 values are printed in the contour messages. So, the leftmost value pair (78, 66 above) corresponds to delay offset 0 from the anchor, and the

next value pair (79, 56 above) corresponds to delay offset +2, and so on. Setting message block field Train2DMisc bit 9 will print all 64 values instead of the abbreviated 31 value format.

The eye contours are dumped in the following order:

1. Rx – SI friendly pattern
 - a. RxClk Complement
 - b. RxClk True
2. Tx
3. Rx – PRBS
 - a. RxClk Complement (first run)
 - b. RxClk Complement (second run)
 - c. RxClk True (first run)
 - d. RxClk True (second run)

If DFE is enabled, each of RxClk Complement and RxClk True dumps will be further split into Previous Bit 0 and Previous Bit 1 eye contours.

5.15 Max Read Latency Training

To perform Max Read Latency Training, set the MxRdLat bit of the SequenceCtrl field of the message block prior to starting the firmware.

When reading data from the drams, the received data is temporarily buffered in an internal receive data FIFO. After a short period, called the read latency, the data in the FIFO is guaranteed to be valid. The read latency training stage finds the smallest read latency that works for all FIFOs in the PHY, limiting system memory latency.

The training is done by setting the read latency very low, then incrementing it while repeatedly performing reads. Most of these reads will fail because the read latency is too small. Eventually, as the read latency reaches the minimum correct value, reads on some bytes will begin to pass. Once all bytes are passing, the firmware will stop searching and add a user-programmable offset passed in through the message block field “DFIMRLMargin”. The PHY will train all max read latency CSR’s in the PHY to the same value, ensuring all data is aligned on the DFI interface.

The final delay written to the DFIMRL CSR is the optimized read latency training.

The margin value is meant to cover for any temperature and voltage variations in the system. If a value of 0 is used for the margin, then the FIFO may not be valid under all possible conditions. It is recommended that the user program the DFIMRLMargin to 2 or 3 for LPDDR4 and LPDDR5 systems. The additional latency on LPDDR systems helps account for the expected drift in the DRAM’s tDQSCK.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
DWC_DDRPHYA_DBYTE<N>_DFIMRL[4:0]_p[1:0]	Timing group 0 timing for write DQ delay	Y	Y

Max Read Latency Training configuration options in message block:

Field Name	Description	Protocol	
		LPDDR4	LPDDR5
DFIMRLMargin	Margin added to smallest passing trained DFI Max Read Latency value, in units of DFI clocks. Recommended to be ≥ 1 .	Y	Y

5.16 PPT2 Initialization Training (for PUB2.x only)

To perform PPT2 (Periodic Phase Training) Initialization training you need to set `pUserInputAdvanced->RetrainMode` set to 0x4 in `phyinit`. PPT2 is only function on PHYs that support it.

PPT2 retraining finds the edges of the RxClk, TxDQ, and TxDQS (Write Link ECC) eyes during training. During mode due to thermal and voltage drift the optimal delays and VREFs may change. In order to detect if they have drifted we do a search to find the left and right edges of the eyes for RxClk, TxDQ, and TxDQS. After the scan we compare the edges found to the trained value plus the eye offsets. The delta between these values is applied as a compensation term to the corresponding trained value.

The training is done by scanning the delay corresponding to the eye offset to generate an eye. For example, scanning `RxClkT2UIDlyTg` to find `RxClkTLeftEyeOffsetTg` and `RxClkTRightEyeOffsetTg`. To perform this scan, we use a fixed pattern (0xCCCC) that is the same as the retraining step. This eye is then compared to our trained value to determine the left and right offset for that eye.

The CSRs affected by this step are:

CSR Name	Description	Protocol	
		LPDDR4	LPDDR5
<code>DWC_DDRPHYA_DBYTE<N>_RxClkTLeftEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for RxClk True Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkTLeftEyeOffsetTg1_r[8:0]_p[1:0]</code>	Timing group 1 timing for RxClk True Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkTRightEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for RxClk True Right Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkTRightEyeOffsetTg1_r[8:0]_p[1:0]</code>	Timing group 1 timing for RxClk True Right Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkCLeftEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for RxClk Complement Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkCLeftEyeOffsetTg1_r[8:0]_p[1:0]</code>	Timing group 1 timing for RxClk Complement Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkCRightEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for RxClk complement Right Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_RxClkCRightEyeOffsetTg1_r[8:0]_p[1:0]</code>	Timing group 1 timing for RxClk Complement Right Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_TxDqLeftEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for Tx DQ Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_TxDqLeftEyeOffsetTg1_r[8:0]_p[1:0]</code>	Timing group 1 timing for Tx DQ Left Eye Offset	Y	Y
<code>DWC_DDRPHYA_DBYTE<N>_TxDqRightEyeOffsetTg0_r[8:0]_p[1:0]</code>	Timing group 0 timing for Tx DQ Right Eye Offset	Y	Y

DWC_DDRPHYA_DBYTE<N>_TxDqRightEyeOffsetTg1_r[8:0]_p[1:0]	Timing group 1 timing for Tx DQ Right Eye Offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqsLeftEyeOffsetTg0_r[8:0]_p[1:0]	Timing group 0 timing for Tx DQS Left Eye Offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqsLeftEyeOffsetTg1_r[8:0]_p[1:0]	Timing group 1 timing for Tx DQS Left Eye Offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqsRightEyeOffsetTg0_r[8:0]_p[1:0]	Timing group 0 timing for Tx DQS Right Eye Offset	Y	Y
DWC_DDRPHYA_DBYTE<N>_TxDqsRightEyeOffsetTg1_r[8:0]_p[1:0]	Timing group 1 timing for Tx DQS Right Eye Offset	Y	Y

Where <N> designates the corresponding DBYTE instance.

5.17 Message Block Training Margin Reporting


Following the completion of 2D training the worst-case margins are written to the message block. Each margin is the worst-case delay or voltage margin for each direction (RX/TX), channel and rank that have been trained.

Delay margins are an approximate measure of the worst-case margin for the trained dbytes and lanes in the specified direction, channel and rank in units of 64^{ths} of a UI.

Voltage margins are an approximate measure of the worst-case margin for the trained dbytes and lanes in the specified direction, channel and rank in units that are specified in the *PHY Databook*.

RxClkDly_Margin_<ch><rank>[7:0]	Worst-case delay margin for RX, channel <ch>, rank <rank> (64ths of UI)
VrefDac_Margin_<ch><rank>[7:0]	Worst-case Vref margin for RX, channel <ch>, rank <rank> (voltage units ¹)
TxDqDly_Margin_<ch><rank>[7:0]	Worst-case delay margin for TX, channel <ch>, rank <rank> (64ths of UI)
DeviceVref_Margin_<ch><rank>[7:0]	Worst-case Vref margin for TX, channel <ch>, rank <rank> (voltage units ¹)

¹See PHY Databook to determine the voltage units.

 Note	When Disable2D = 1 neither delay, nor voltage margins will be populated.
-------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------

5.18 End of Training

The firmware does not end when the last SequenceCtrl stage completes, it would calculate the critical command delays (CDDs), and write the training results to PS RAM if 15 Pstate Mode enabled.

5.18.1 CDDs

First the microcontroller must calculate the critical command delays (CDDs) between the timing groups and guarantee the state of the DRAMs. This step cannot be skipped.

The critical command delays define how routing differences between timing groups impacts the minimum command spacing between reads or writes when switching between timing groups. For example, when switching timing groups to access a rank with shorter delays, the memory controller must wait slightly longer before issuing the first command to the “faster” timing group to avoid its shorter latencies causing the command to catch up with, and corrupt, the command already in flight to the “slower” timing group. The opposite is also true; when switching from a fast to a slow timing group, the command spacing’s can be reduced. The PHY reports all CDD’s to the memory controller through the message block. The PHY calculates the CDDs by analyzing the delay differences between timing groups for when read-data arrives at the PHY (RxEnDly) or when write data leaves the PHY (TxDqsDly). The dbyte with the largest delay difference between two timing groups bounds the minimum CDD between the timing groups. The training firmware rounds the CDD to the closest MEMCLK boundary before writing the result to the message block for ease of use by the memory controller. CDD’s are discussed in more detail in the PUB databook.



The CDDs only calculate flight time differences between ranks. To find true legal command spacings, the user must also consider DRAM protocol requirements and bus termination change times (where applicable). The CDD delays are applied as adders to the legal command spacings.

After calculating the CDD’s, the firmware ends by guaranteeing devices are left in the state the user requested, writing the exact MRs settings in the message block to the devices. At the end of training the DRAM will be left in the following state:

DRAM Type	Condition
LPDDR4	The device will be left in Self-Refresh+Powerdown with FSP-OP=1 and FSP-WR=0. FSP-OP1 mode registers will be programmed as per the user settings provided in the message block
LPDDR4X	The device will be left in Self-Refresh+Powerdown with FSP-OP=1 and FSP-WR=0. FSP-OP1 mode registers will be programmed as per the user settings provided in the message block
LPDDR5	The device will be left in Self-Refresh+Powerdown with FSP-OP=1 and FSP-WR=0. FSP-OP1 mode registers will be programmed as per the user settings provided in the message block

5.18.2 PS RAM

When enable 15 Pstate Mode, the training results will be written to PS RAM at the end of each training, see PUB Databook section 7.24.3 for details on PS RAM.

All the CSRs and MRs related to training results list below:

CSRs:

TxCKDcaCtrlCKC, TxACDcaCtrl, CKTxDly, TxCKDcaCtrlCKT, ACTxDly,

TxDQSDcaCtrlTTg*, TxWckDlyTg*, TxWCKDcaCtrlCTg*, TxWCKDcaCtrlTTg*, TxDqsDlyTg*,
TxDQSDcaCtrlCTg*, RxWCKDcaCtrlTTg*, RxEnDlyTg*, RxWCKDcaCtrlCTg*, TxDQDcaCtrlTg*,
PptDqsCntInvTrnTg*,

RxReplicaUICalWait, RxReplicaCtl03, RxReplicaCtl01, DFIMRL, PptWck2DqoCntInvTrnTg0,
PptWck2DqoCntInvTrnTg1, RxReplicaPathPhase0, RxReplicaPathPhase1, RxReplicaPathPhase2,
RxReplicaPathPhase3, RxReplicaPathPhase4, RxDigStrbEn, DxDigStrobeMode, TxDcaMode,

RxDigStrbDlyTg*, RxClkT2UIDlyTg*, TxDqDlyTg*, RxClkC2UIDlyTg*, VrefDAC0, VrefDAC1, VrefDAC2,
VrefDAC3,

DllGainCtl, RxReplicaDllGainCtl, RxReplicaDllLockParam, AcDllLockParam, DxDllLockParam,
SingleEndedMode.

MRs:

LPDDR4: MR12, MR14.

LPDDR5: MR12, MR14, MR15, MR24, MR30.

6 Protocol Specific Requirements

6.1 Multiple PHYs

6.1.1 LPDDR4

When connecting multiple PHYs to the same LPDDR4 device (for example two 16 bit PHYs on a Package-On-Package [POP] system). The following restrictions must be met:

1. The training hardware on the PHY controlling the DRAM RESET pin must be initialized first.
2. The PHY not controlling RESET must start within 500ns of the PHY controlling reset.
 - a. If PU-Cal is set to the default value, a larger start delay can be tolerated.
3. The MR3 settings for PU-Cal and PDDS are the same for both PHYs
4. When training multiple PStates, the PHY controlling reset must be initialized first for each PState.
5. When training multiple PStates, all PHYs must complete training of the previous PState before the first PHY starts training for the next PState.

This is to ensure that the RESET timing is met on both PHYs and that ZQCal Start Occurs after both PHYs have initialized their pull up settings to the same value as required by the DRAM device.

6.1.2 LPDDR5

When connecting multiple PHYs to the same LPDDR4 device (for example two 16-bit PHYs on a Package-On-Package [POP] system). The following restriction must be met:

1. The training hardware on the PHY controlling the DRAM RESET pin must be initialized first.
2. When training multiple PStates, the PHY controlling reset must be initialized first for each PState.
3. When training multiple PStates, all PHYs must complete training of the previous PState before the first PHY starts training for the next PState.

6.2 LPDDR Automatic Boot Clock Frequency switching

Per JEDEC specification, an untrained/uninitialized LPDDR device does not guarantee correct operation of the CA bus at frequencies beyond the boot clock (tCKb). The boot frequency range is

- 10MHz (max tCKb=100ns described in several DRAM vendor datasheets)
- 55MHz (min tCKb=18ns described in the LPDDR4 JEDEC spec)

The PHY contains hardware to synthesize MEMCLK frequencies within the boot-range without modifying the DFICLK or the PLL frequency.

The PHY firmware will automatically set the MEMCLK frequency as required by the JEDEC specs for the dram initialization, command bus training, and 1D training steps. See the table below:

Training Major Step	Training Sub-Step	MEMCLK Frequency
DevInit()	Set Reset_n=0	n/a
	Wait tInit1	n/a
	Set Reset_n=1	n/a
	Wait tInit3	n/a
	Enable Boot Frequency MEMCLK	{10:55Mhz}
	Assert CKE	{10:55Mhz}
	Send MRW required for ZQCAL	{10:55Mhz}
	Send ZQCAL Start	{10:55Mhz}
	Set ZQCAL Latch	{10:55Mhz}
	Set FSP-WR=1	{10:55Mhz}
	Set FSP-OP[1] registers to high speed settings	{10:55Mhz}
FinishTraining()	Place all ranks in {FSP_OP=1 , VRCG=0}	Application Frequency
	Place all ranks in {Powerdown+Self-Refresh}	Application Frequency

6.3 Pstate training order

For LPDDR, it is required that the last Pstate trained be the same as the first mission mode Pstate that will be specified in the DFI initialization request. This requirement is because the firmware must be used to set up the state of the DRAM state prior to transition to mission mode.

6.4 DQ and CA swizzle

The hardware supports routing Dq bits within the PHY to different Dq pins on the DRAM device as long as the Dq bits are contained in the same byte. For Command Address training and MRR commands to function properly, the Dq<N>LnSel bits need to be programmed correctly. Additionally, the PHY supports re-assigning address pins on the Command Address bus for pins CAA<0-5> as well as CAB<0-5>. These re-assignments are controlled by the MapCAA<N>toDfi csr. See the PUB databook for more information on these CSRs.

If the CAA re-assignment is incorrect, training will typically fail completely on early steps. Incorrect DQ mapping may cause a variety of symptoms, including but not limited to CA training failures and coarse write leveling failures.

6.5 Byte Swap

LPDDR firmware supports swapping of bytes of x16 devices as long as the swaps are legal as defined in the PUB databook. The LPDDR4/LPDDR5 firmware does not detect these swaps. The swap data should be programmed in phyinit. This allows the firmware to distinguish between upper and lower bytes for MRR commands and CA training. DQ bits swizzles are not detected and need to be programmed.

6.6 Link ECC

For LPDDR5, it is required that RECC should be enabled as well if WECC is enabled. This requirement is because both WECC and RECC must be enabled when using WCK-RDQS_t/Parity training mode.

Ideal / Acceptable ttttttdddd0ddddddd0ddddddd0ddddddd0dddddgttttt

Not Acceptable ttttttdddd0dddjddd0ddddjddd0dddjddd0dddddgttttt

Examples of when X's on the DQ lines are expected and reasonable:

- X's are expected and reasonable if a DRAM location is read before it is written.
- X's are expected and reasonable if a DRAM protocol error has occurred. An example would be if reading from a closed page.

7.1.2 Hi-Z bus modeling

There are times during the training process that the PHY may sample the DQ or DQS values when the bus is Hi-Z. This is normal and an expected behavior in the firmware. In a real system, this value would either be a static 0, a static 1, or a pseudo-random 0/1. The firmware is designed to accommodate this. But if the Z is converted to an X, this affects the firmware as if the value is both 0 and 1 simultaneously and can cause the system to train improperly or malfunction. To simulate the actual analog behavior, the PHY Verilog code has a simulation only function that will turn Z's into non-X's. This option must be enabled for the training firmware to operate properly. To enable this, the user must set a VCS plusarg:

```
VCS +PLUSARG :: ""+ddr_squashz_to_0""
```

This option will cause the PHY behavioral model for the receiver to convert the Z's on the bus to 0's instead of X's.

7.1.3 Disabling behavioral X-injection in FIFOs

Inside the PHY, there are simulation only X-injection models in the receive FIFO that enforce proper read and write pointer separation during mission mode operation.

In the Max Read Latency training step, the pointer separation is being trained, and the X-injection must be disabled so that the behavioral RTL model of the data-comparison doesn't cause X-propagation.

When performing digital simulations of PHY Training, the X-injection must be disabled with the following VDEFINE.:

```
EXTRA VDEFINE :: "+VDEFINE DWC_DDRPHY_Tech__CDCBUF__DISABLE_BEHAVIORAL_VERILOG"
```

7.1.4 DRAM refresh protocol checking


The training firmware will perform activate and precharge commands to open and close banks to use to test transactions. These transactions are done only on the open banks and not from the underlying array. The firmware doesn't attempt to send refresh commands to preserve the contents of the array. Many memory models will detect refresh protocol violations. For example, max refresh interval and tRAS max limits can be violated during training. These errors need to be disabled in the memory model.

7.2 Simulation Environment Bring Up

When developing a simulation environment for the firmware, it is recommended that the initial simulations only set the devinit bit of sequenceCtrl. DevInit only simulations will not train any of the PHY's delays but will allow basic behavior to be checked with the fastest turn-around time possible. Features Dev-init only simulation can be used to verify include firmware image loading, basic message block input requirements, and optionally the mailbox protocol. If the mailbox protocol is not being verified, set HdtCtrl to 0xff to disable message passing.

7.3 Reducing Digital Simulation Time

Running all training stages in simulation can take a significant amount of time. To help reduce simulation times, the firmware has several message block inputs especially for digital simulations.

 Warning!	These options are only to be used for digital simulations. These options should never be used in a real system. Use of these options in silicon system may result in an inoperable or improperly trained memory system.
-----------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Message block Field MsgMisc contains:

- ShortReset – Setting this bit field to 1 will shorten the DRAM reset to 1ns instead of the required JEDEC time. Since digital simulations don't generally require extended reset lengths, this will speed up the DRAM initialization process in simulation. This should only be set to 1 for simulations.

For short simulations which run the DevInit training step only, use the “devinit_skiptrain” sequence of PhyInit. This sequence will pre-compute known optimal delay values for a zero-delay system. See the PhyInit sequence for further details.

8 Streaming Messages

Streaming messages are used to pass more information back than just a static message number. They are used for debug purposes and are not necessary for normal operation. Streaming messages are not guaranteed to have the same encoding from one firmware revision to the next.

The start of a streaming message is signaled by the major message "0x8". Streaming messages are made up of several pieces of mail and provide more verbose debug information to customers as the firmware is executing. After the customer has acknowledged receiving a major message of 0x8, the firmware will immediately send a 32-bit message, called a string index, through the mailbox. String indices are decoded by referencing the appropriate DRAM image specific strings file that comes with each release.

Strings files are located in the same directory as the instruction and data memory images and have the suffix ".strings".

An excerpt from an example strings file is included below for reference.

```
0x00820000 "PMU5: Resetting DRAM\n"
```

```
0x007d0001 "PMU3: MREP Delay = %d\n"
```

Each line is composed of the string index, followed by the string it represents. The strings themselves follow the same formatting rules as C's printf function.

The lower 16 bits of the string index are particularly useful as they say exactly how many additional 32-bit pieces of mail need to be read before the streaming message can be fully formatted. For example, for the value of 0x007d0001, the lower 16 bits indicate that one 32-bit value will need to be read after the string index to complete the streaming message. Once all of the arguments have been received, the streaming message has ended. And the next mailbox message will go back to being a major message.

Pseudocode for polling and printing the full string value is given below:

```
//-----  
// Decode_Streaming_Message Pseudocode  
//-----  
Function decode_streaming_message(){  
    string_index = get_mail(32bit)  
    debug_string = lookup_string_index(string_index)  
    while (i <= (string_index & 0xffff)){  
        args[i] = get_mail(32bit)  
        i++  
    }  
    Printf(debug_string, arg[0], arg[1], ... , arg[i-1])  
}
```

9 Training Time Optimization

9.1 LPDDR54 Optimization conditions

The LPDDR54 PHY has a robust data collection and training solution designed to work at high speed. Some customers may need faster training times for their applications. Synopsys provides some options to reduce training time for customers who have systems that can achieve consistent results.

9.2 General Procedure

Once the PHY is functioning and performing well, the customer can disable some steps to decrease link training time. This should be done with a combination of SI simulation and lab data.

9.3 PMU Clock Speed

The PHY microcontroller supports a clock divider to easy static timing closure. If microcontroller static timing is closed at the DFI clock frequency, the PmuClockDiv setting can be set to 0 to improve training time.

9.4 Training Sequence

Some steps can be skipped provided depending on the customer system. The following sections describe steps that can be optionally skipped to decrease training time

9.4.1 TX DFE (LPDDR5 Only)

LPDDR5 has an optional DFE feature. The process of evaluating the best DFE setting requires collecting data eyes for all lanes and finding the best setting out of the legal values. Since the improvement of DRAM DFE depends on actual DRAM and channel characteristics, a default setting can be used.

9.4.2 PHY DCA (LPDDR5 Only)

LPDDR54 training firmware supports optimizing built in PHY DCA settings. This is in addition to the DRAM DCA settings. When adjusting PHY DCA settings it may be necessary to retrain the DQ link to compensate for shifts caused by duty cycle adjustment. Since the improvement of PHY DCA over DRAM DCA training depends on actual DRAM and channel characteristics, a default setting can be used.

9.4.3 Training Loops (LPDDR5 only)

When adjusting PHY or DRAM DCA, the firmware retrains the DQ bus of the corresponding step. For DRAM DCA training, the retraining is done to recenter WCK/CK write leveling and RxEn Placement as well as to improve RxClk Eye margin for TX DQ training. If there is sufficient margin, it is possible to skip the retraining. The final RxClk step that runs after TX DQ training will optimize the RxClk margin before entering mission mode.

Category	Parameter	Recommended values	Note
PMU	PmuClockDiv	0x0	Operate PMU with high frequency : UcClk = full speed of DFICLK (800MHz for LP5-6400, 533MHz for LP4X-4266, 1:4 ratio) This requires customer's timing closure with flag_ucclk_full = 1
Training sequence	SequenceCtrl	0x125f (LPDDR5) 0x121f (LPDDR4X)	Disable DFE, PHY DCA trainings
	DisableTraining Loop	0x7 (LPDDR5)	Disable re-training loop of RxEn, Write-leveling, SI Friendly Read trainings after DRAM DCA training
Vref	See Table 11-13		Optimized Vref sweep range, step settings for Write/Read training
Delay/Vref optimization	Train2DMisc	0x400	Weighted mean algorithm
Read training option	Misc	0x60	Bit[5] = 1 : Pre Compute RxClk Coarse bit Bit[6] = 1 : Do single RxClk scan

9.5 VREF Sweep Adjustment

The LPDDR54 training firmware provides settings to reduce the number of VREF scans needed to perform training. These controls are described in detail in section 5.14.3.2.

9.5.1 Range and Step Selection Procedure

To select optimal VREF step parameters, the range of variation needs to be covered, and enough data needs to be collected for the 2D algorithm to determine delay and VREF training values. It is recommended to execute enough VREF sweeps to contain passing regions in the eye after accounting for the variation.

The eye height and expected center depends on ODT and drive strength settings. The total variation in the eye height is expected to contain the following components.

- 1) ODT variation
- 2) Drive strength variation
- 3) Receiver VREF offset

The size of these variation depends on the DRAM device and the PHY process. The next section has some example settings taking some of these parameters into account. These parameters depend on direction (TX vs RX) also.

9.5.2 Example Settings

The recommended Vref sweep range is estimated to cover both driver impedance and ODT variation of +/-10% with opposite direction in addition to receiver offset, where it's assumed that $VOH = VDDQ/2$ and thus ideal $Vref = VDDQ/4$. For some applications which uses different VOH , it's also recommended to adjust VrefStart and VrefEnd values depending on actual channel characteristic by simulation and/or Si validation.

Mode	Parameter	Value	Vref [mV]	% of VDDQ	Note
LPDDR5 (VDDQ = 0.5V) Assumption: Ideal Vref ($VOH/2$) = 125mV (25% VDDQ), $VOH = VDDQ/2$	RxVrefStartPatDfe0,1	0x17	66.2	13.2%	For SI Friendly Read training, PHY Vref sweep range of $VOH/2 \pm 58mV$ with 12mV step, and Vref sweep count = 11
	RxVrefEndPatDfe0,1	0x53	186.7	37.3%	
	RxVrefStepPatDfe0,1	0x6	12.0	2.4%	
	RxVrefStartPrbsDfe0,1	0x17	66.2	13.2%	For Read training, PHY Vref sweep range of $VOH/2 \pm 58mV$ with 12mV step, and Vref sweep count = 11
	RxVrefEndPrbsDfe0,1	0x53	186.7	37.3%	
	RxVrefStepPrbsDfe0,1	0x6	12.0	2.4%	
	TxVrefStart	0x10	90.0	18.0%	For Write training, DRAM Vref sweep range of $VOH/2 \pm 35mV$ with 10mV step, and Vref sweep count = 8
	TxVrefEnd	0x2c	160.0	32.0%	
	TxVrefStep	0x4	10.0	2.0%	

Table 11 Example LPDDR5 VREF Range and Step Settings

Mode	Parameter	Value	Vref [mV]	% of VDDQ	Note
LPDDR4X (VDDQ = 0.6V) Assumption: Ideal Vref (VOH/2) = 150mV (25% VDDQ), VOH = VDDQ/2	RxVrefStartPatDfe0,1	0x17	79.4	13.2%	For SI Friendly Read training, PHY Vref sweep range of VOH/2 +/- 70mV with 14.5mV step, and Vref sweep count = 11
	RxVrefEndPatDfe0,1	0x53	224.0	37.3%	
	RxVrefStepPatDfe0,1	0x6	14.5	2.4%	
	RxVrefStartPrbsDfe0,1	0x17	79.4	13.2%	For Read training, PHY Vref sweep range of VOH/2 +/- 70mV with 14.5mV step, and Vref sweep count = 11
	RxVrefEndPrbsDfe0,1	0x53	224.0	37.3%	
	RxVrefStepPrbsDfe0,1	0x6	14.5	2.4%	
	TxVrefStart	0x7	115.2	19.2%	For Write training, DRAM Vref sweep range of VOH/2 +/- 35mV with 14.4mV step, and Vref sweep count = 6
	TxVrefEnd	0x1b	186.6	31.1%	
	TxVrefStep	0x4	14.4	2.4%	

Table 12 LPDDR4X VREF Range and Step Settings

Mode	Parameter	Value	Vref [mV]	% of VDDQ	Note
LPDDR4 (VDDQ = 1.1V) Assumption : Ideal Vref (VOH/2) = 183mV (16.6% VDDQ), VOH = VDDQ/3	RxVrefStartPatDfe0,1	0xb	92.6	8.4%	For SI Friendly Read training, PHY Vref sweep range of VOH/2 +/- 90mV with 26.5mV step, and Vref sweep count = 8
	RxVrefEndPatDfe0,1	0x35	278.1	25.3%	
	RxVrefStepPatDfe0,1	0x6	26.5	2.4%	
	RxVrefStartPrbsDfe0,1	0xb	92.6	8.4%	For Read training, PHY Vref sweep range of VOH/2 +/- 90mV with 26.5mV step, and Vref sweep count = 8
	RxVrefEndPrbsDfe0,1	0x35	278.1	25.3%	
	RxVrefStepPrbsDfe0,1	0x6	26.5	2.4%	
	TxVrefStart	0x6	136.4	12.4%	For Write training, DRAM Vref sweep range of VOH/2 +/- 40mV with 17.6mV step, and Vref sweep count = 6
	TxVrefEnd	0x1a	224.4	20.4%	
	TxVrefStep	0x4	17.6	1.6%	

Table 13 LPDDR4 VREF Range and Step Settings

9.6 Trade-off of training time vs. accuracy

Since training time reduction approach is prepared for well-designed system, some applications especially with signal/power integrity challenging systems are required to evaluate trade-off of training time and training accuracy.

There are basically two ways to validate accuracy of training results. It's recommended to compare difference in delay and Vref margin values between default settings and recommended settings.

- 1) Delay and Vref margin evaluation from training log
- 2) Mission mode write and read eye margin evaluation by using diagnostic firmware