



DesignWare® Cores LPDDR54 PHY ATE Firmware

Application Note

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

Copyright Notice and Proprietary Information

© 2021 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.

www.synopsys.com

Contents

Revision History	7
Preface	11
Recommended Reading	12
Web Resources	13
Synopsys Statement on Inclusivity and Diversity	14
Customer Support	15
Chapter 1	
Introduction	17
1.1 Purpose	18
Chapter 2	
Loading, Configuring and Running the Test ATE Firmware	19
2.1 Overview	20
2.2 Bring up VDD, VDDQ, and VDD2H (Step A)	22
2.3 Start Clocks and Reset the PHY (Step B)	23
2.4 Load IMEM / DMEM Memory (Step C)	24
2.5 Configure Technology Specific Registers (Step D1)	25
2.6 Configure ARC Clock Rate (Step D2)	26
2.7 Configuring the Firmware Via the Message Block (Step D3)	27
2.8 Execute the Firmware (Step E)	28
2.9 Read the Message Block Results (Step F)	29
2.10 Performing more testing	30
Chapter 3	
ATE Firmware Usage	31
3.1 ATE Firmware Use	32
3.2 Test Ordering	33
3.3 Interacting with the ATE Firmware and Determining When the Firmware is Done	34
3.3.1 ATE Firmware Run Time Equations (Half-Rate ARC)	34
3.3.2 ATE Firmware Run Time Equations (Full-Rate ARC)	38
Chapter 4	
Individual ATE Test Details	43
4.1 General Message Block Input Fields	44
4.1.1 TestsToRun and PassFailResults Message Block Fields	45

4.1.2 TestsToRun and PassFailResults Bit Mapping	46
4.1.3 DfIClkFreqRatio and Clocking Mode	46
4.1.4 UseMsgBlkPhyCfg Behaviour	47
4.2 Revision Number Check	48
4.2.1 Message Block Inputs / Outputs and legal values:	48
4.3 Impedance Calibration	49
4.3.1 Message Block Inputs / Outputs and legal values:	49
4.3.2 Impedance Calibrator Behavioural Model	50
4.4 PLL Lock / LCDL Calibration	51
4.4.1 Message Block Inputs / Outputs and legal values:	51
4.4.2 LCDL 1UI Lock Code Instance Numbers	53
4.4.3 MemClk Toggle	54
4.4.4 MemClk Toggle Infinite Mode	54
4.5 RxReplica Calibration Test	56
4.5.1 Message Block Inputs / Outputs and legal values	56
4.5.2 Pre-test Requirements	56
4.6 LCDL (Locally Calibrated Delay Line) Linearity	57
4.6.1 Message Block Inputs / Outputs and legal values:	58
4.6.2 Message Block Inputs - Illegal Argument Checking	61
4.6.3 LCDL Linearity Instance Numbers	61
4.6.4 LCDL Count Values as Message Block Outputs:	62
4.6.5 How to Interpret the LCDL Count Value Output from the LCDL Linearity Test Results, and Plot the Results	63
4.7 Address/Command Loopback	64
4.7.1 Message Block Inputs / Outputs and legal values:	66
4.7.2 Message Block Inputs - Illegal Argument Checking	69
4.7.3 AC Loopback Infinite-Traffic Mode	69
4.7.4 CK DIFF Slice Single-Ended Testing	69
4.7.5 How to Interpret the Output From the Address/Command Loopback Test Results	70
4.7.6 Pre-test Requirements	75
4.7.7 Termination on Address / Command Pins for Pad Loopback	75
4.8 Data Loopback 1D	77
4.8.1 Message Block Inputs / Outputs and legal values:	79
4.8.2 Message Block Inputs - Illegal Argument Checking	82
4.8.3 Data Loopback 1D Infinite-Traffic Mode	82
4.8.4 DQS Testing	83
4.8.5 DQS + WCK DIFF Slice Single-Ended Testing	83
4.8.6 How to Interpret the Output From the Data Loopback 1D Test Results	83
4.8.7 Pre-test requirements	87
4.8.8 Termination on Data Pins for Loopback	87
4.8.9 Optimizing Data Loopback Runtime	87
4.9 Data loopback 2D	88
4.9.1 Message Block Inputs / Outputs and legal values:	88
4.9.2 Message Block Inputs - Illegal Argument Checking	91
4.9.3 Data Loopback 2D Infinite-Traffic Mode	91
4.9.4 How to Interpret the Output From the Data Loopback 2D Test Results	92
4.9.5 Pre-test Requirements	93
4.9.6 Termination on Data Pins for Pad Loopback	93
4.9.7 Optimization of the run time for the 2D Data Loopback test	93

4.10 DCA Loopback	94
4.10.1 Message Block Inputs / Outputs and legal values:	95
4.10.2 Message Block Inputs – Illegal Argument Checking	97
4.10.3 DCA Delay Sweep Optimization	97
4.10.4 How to Interpret the Output From the DCA Loopback Test Results	99
4.10.5 Pre-test requirements	100
4.10.6 Termination on WCK Pins for DCA Testing	100
4.11 Burn-in	101
4.11.1 Message Block Inputs / Outputs and legal values:	102
4.11.2 Running the Burn-In Test	102
4.11.3 Pre-test requirements	103
Chapter 5	
Simulation Firmware	105
5.1 Simulation Requirements for Firmware	106
5.1.1 PAD Model Behavior	106
5.1.2 Hi-Z Bus Modeling	106
5.2 Disabling Behavioral X-injection in FIFOs	107
Chapter 6	
Bringing Up the ATE firmware	109
6.1 Initial running of the ATE firmware in simulation or silicon	110
6.2 Bringing-Up Individual Tests	111
6.2.1 General Failure Causes	111
6.2.2 Revision Check Test	111
6.2.3 Impedance Calibration	111
6.2.4 PLL Lock / LCDL Lock Test	112
6.2.5 RxReplica Calibrate Test	112
6.2.6 LCDL Linearity Test	112
6.2.7 Address / Command Loopback Test	113
6.2.8 Data Loopback 1D Test	114
6.2.9 Data Loopback 2D Test	114
6.2.10 DCA Loopback Test	114
6.2.11 Burn-In Test	114
Chapter 7	
ATE Message Block Definition	115
7.1 ATE Message Block definition	116

Revision History

The following tables lists the revision history of the PHY from release to release. Refer to the PHY release notes for a detailed description of PHY component updates.



Note

- Links and references to section, table, figure, and page numbers in this table are only assured to be valid for the version in which the change is made
- In some instances, documentation-only updates occur. The DesignWare IP product information (<http://www.designware.com>) has the latest documentation

Document Version	Date	Description
1.10	October 18, 2021	<p>Updated:</p> <ul style="list-style-type: none"> ■ “Configure Technology Specific Registers (Step D1)” on page 25 ■ “Message Block Inputs / Outputs and legal values:” on page 66 ■ “Message Block Inputs / Outputs and legal values:” on page 79 ■ “Message Block Inputs / Outputs and legal values:” on page 88 ■ “Message Block Inputs / Outputs and legal values:” on page 95 ■ “RxReplica Calibrate Test” on page 112 <p>Added:</p> <ul style="list-style-type: none"> ■ “AcLoopbackCoarseSec” on page 75 ■ “AcLoopbackCoarseDiff” on page 75 ■ “DatLoopbackWckCoarse” on page 86 ■ “DatLoopbackDqsCoarse” on page 86 ■ “DatLoop2dByteBitFail” on page 93 ■ “Dca_Delay” on page 100

Document Version	Date	Description
1.09	June 4, 2021	<ul style="list-style-type: none"> ■ Describe usage of the provided *_ecc.txt files. ■ Added full-rate ARC mode runtime equations. ■ Updated half-rate ARC mode runtime equations. ■ Document new PubRev message block input. ■ Recommend the use of the ZCal behavioural model from the CTB. ■ Continual ZCal calibration must be disabled if no behavioural model is used. ■ Update PLL Lock-test infinite MemClk toggle generation usage description. ■ Fix RxReplica checking bound to match firmware. ■ Elaborate on LcdlPassPercent behaviour ■ Provide suggested LcdlClksToRun value. ■ Fix recommended values for LcdlStride. ■ Add LCDL Linearity test bring-up documentation on how to tune the LcdlClksToRun value ■ Clarify that the Burn-In test must be run in isolation.
1.08	April 2, 2021	<ul style="list-style-type: none"> ■ Added documentation on selecting between full-rate and half-rate ARC clock mode. ■ Document relationship between the loopback increment message block inputs, and loopback min-eye-width message block inputs. ■ Runtime equation for data-loopback-1D/2D should be a function of DatLoopFineIncr ■ Fixed the formula for the number of transitions of DctWriteProt required to bring AC-Loopback to completion, when infinite-traffic mode is selected.
1.07	December 1, 2020	<ul style="list-style-type: none"> ■ Correction to description of when PLL is in x4 and when it is in x8 mode (section 2.5). ■ If continuous impedance calibration is enabled, the latest codes are pushed to the drivers prior to driving traffic. ■ Corrections to TestOptions[2] and TestOptions[3] descriptions, respecting the values of ClockingMode which effect their behaviour. ■ MemClkToggle and MemClkTime must both be non-zero for the PLL Lock test to drive a clock on CK. ■ RxReplica test now stores the Replica path-phases and RxReplica 1UI lock codes to the results section of the message block. ■ Improved description on how to program AC Loopback test input AcMinEyeWidthSec ■ Improve description of when the AC SE passing region wraps around from max-delay to min-delay. ■ Added examples of AC DIFF slice EYE positions, and how they are stored to the results section of the message block. ■ Document improvement to DCA loopback test, which reduced test runtime.

Document Version	Date	Description
1.06	August 17, 2020	<ul style="list-style-type: none"> ■ Added DCA Loopback test documentation. ■ Document RxReplica path-phase check error bars. ■ Added “infinite-traffic” mode support for AC loopback and Data Loopback 1D/2D. ■ Add documentation regarding DatLoopMinLoopPwr in the context of Burn-In. ■ Burn-In now supports core-side loopback. ■ Added Burn-In configuration recommendations, depending on supported protocols. ■ Update Burn-In wait-time to be more accurate. ■ Make it clear that the WCK DIFF slices also need to be terminated. ■ Added documentation for AC Loopback input AcLoopLaneMask. ■ Corrected Data Loopback 1D/2D input DatLoopMinEye-Width recommendation. ■ Added documentation for Data Loopback outputs DatLoopbackRxEnbVal and DatLoopbackDqsBitmap. ■ Fixed cross-references.
1.05	July 22, 2020	References to section 4.6.3 were using the incorrect section number.
1.04	July 16, 2020	Add description of PLL Lock test message block output LcdlResultsRxReplica
1.03	June 30, 2020	<p>Include documentation on dedicated DQS, single-ended, and RxReplica testing.</p> <p>Subsequent ATE tests can be run without reloading the IMEM/DMEM.</p> <p>Updated runtime equations.</p> <p>Fixed release binary names.</p> <p>SEC slice(s) should not be terminated for pad-side AC loopback.</p>
1.02	April 30, 2020	Preliminary runtime equations for each test included
1.01	April 17, 2020	<p>Include description of 1UI lock code LCDL encoding.</p> <p>Include preliminary runtime equations.</p>
1.00	March 27, 2020	Initial Release
0.10	March 13, 2020	Draft Release
0.04	February 20, 2020	Continued Development
0.03	December 2, 2019	Added Data Loopback 2D chapter, (test is no longer a shell).
0.02	November 12, 2019	Add “Simulating FW” chapter.
0.01	October 25, 2019	Initial Pre-Release

Preface

This Application Note describes the LPDDR54 PHY ATE Firmware.

Recommended Reading

The following documentation provides essential information about the IP to create a complete solution:

- Designware Cores LPDDR5/4/4X PHY Implementation Guide
- DesignWare Cores LPDDR5/4/4X PHY Databook

Web Resources

- DesignWare IP product information: <http://www.designware.com>
- Your custom DesignWare IP page: <http://www.mydesignware.com>
- Documentation through SolvNetPlus: <http://solvnetplus.synopsys.com> (Synopsys password required)
- Synopsys Common Licensing (SCL): <http://www.synopsys.com/keys>

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:

- First, prepare the following debug information, if applicable:
- For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

File > Build Debug Tar-file

Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <core tool startup directory>/debug.tar.gz.

- 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
- Then, contact Support Center, with a description of your question and supplying the above information, using one of the following methods:
 - For fastest response, use the SolvNet website. If you fill in your information as explained below, your issue is automatically routed to a support engineer who is experienced with your product. The Sub Product entry is critical for correct routing.

Go to <http://solvnet.synopsys.com/EnterACall> and click on the link to enter a call. Provide the requested information, including:

- Product: Designware Cores
- Sub Product: LPDDR54_PHY
- Tool Version:
- Problem Type:
- Priority:
- Title: Provide a brief summary of the issue or list the error message you have encountered
- Description: For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

After creating the case, attach any debug files you created in the previous step.

- 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 name, Sub Product name, and Tool Version number in your e-mail (as identified above) 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 in the previous step.
- 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:
<http://www.synopsys.com/Support/GlobalSupportCenters>

1

Introduction

The following sections are included in this chapter:

- [“Purpose”](#) on page 18

1.1 Purpose

The process of testing the PHY is done through ATPG vectors and functional tests. The firmware provided with the PHY can be used to perform the functional testing for the PHY on Automated Test Equipment (ATE).

This application note describes how to use the ATE firmware.

**Note**

This application note pertains to the Firmware Version: C-2021.10 of DWC DDRPHY ATE firmware.

There may be variations in operation, interface, and usage procedure, depending on the DWC DDRPHY ATE firmware version.

2

Loading, Configuring and Running the Test ATE Firmware

The following sections are included in this chapter:

- [“Overview” on page 20](#)
- [“Bring up VDD, VDDQ, and VDD2H \(Step A\)” on page 22](#)
- [“Start Clocks and Reset the PHY \(Step B\)” on page 23](#)
- [“Load IMEM / DMEM Memory \(Step C\)” on page 24](#)
- [“Configure Technology Specific Registers \(Step D1\)” on page 25](#)
- [“Configure ARC Clock Rate \(Step D2\)” on page 26](#)
- [“Configuring the Firmware Via the Message Block \(Step D3\)” on page 27](#)
- [“Execute the Firmware \(Step E\)” on page 28](#)
- [“Read the Message Block Results \(Step F\)” on page 29](#)
- [“Performing more testing” on page 30](#)

2.1 Overview

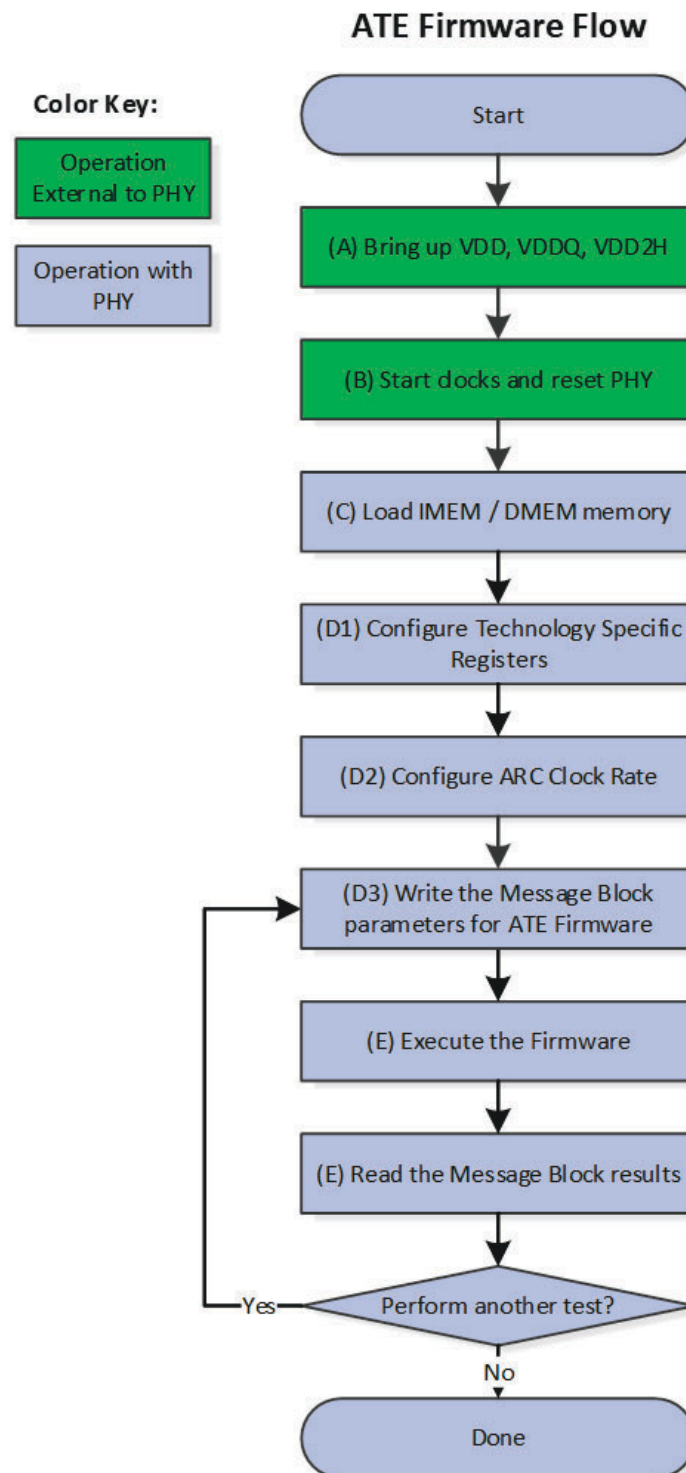
This section is meant to give an understanding of the environment and the process by which the user will run the ATE firmware.

Test ATE is a firmware image which contains several tests that are targeted primarily at automated testing. To run the ATE tests, the host system will load a binary image into the SRAMs using APB or JTAG and then start the firmware execution.

The process for loading and running the ATE firmware is very similar to the process for loading the training or diagnostic firmware:

- Power and reset the device (steps A and B)
- Load the ATE firmware (step C)
- Configure any technology specific registers (step D1)
- Configure ARC clock rate (step D2)
- Configure the ATE firmware to run the desired test(s) (step D3)
- Run the firmware (step E)
- Read back the test results (step F)

Each of these steps is detailed in the following sections.

Figure 2-1 ATE Firmware Flow

2.2 Bring up VDD, VDDQ, and VDD2H (Step A)

The first step for initializing the PHY is to apply power. The power supplies can come up and stabilize in any order. While the power supplies are coming up, all outputs will be unknown, and the values of the inputs are don't cares. Once the power supplies are stable, there are rules on how the power supplies must behave. See “Power Management” in the PUB Databook for more information on the power supply requirements.

2.3 Start Clocks and Reset the PHY (Step B)

The second step is to start the PHY clocks and apply the synchronous reset. The procedure to reset the PHY is as follows:

1. BP_PWROK, Reset, DfiClk, Reset, Reset_async, and WRSTN must be valid.
2. Drive Reset=1 and Reset_async=1
3. Wait a minimum of 10ns
4. Start DfiClk and ApbClk
5. Wait for a minimum of 64 DfiClk and 64 ApbClk cycles
6. Drive BP_PWROK=1 (if not already driven to 1).
7. APB and DFI inputs must be valid (no transaction on DFI/ APB interfaces)
8. If using the TDR interface, it must be reset using the following procedure
 - a. TDRCLK must be running (Step #4)
 - b. Drive WRSTN=0 (asynchronous or synchronous to TDCLK)
 - c. Wait a minimum of 1us
 - d. Drive WRSTN=1 (must be synchronous with TDCLK)
 - e. For more details on the TDR serial interface, refer to PUB section “TDR Serial Interface”
9. Drive Reset=0 and Reset_async=0
10. The PHY has now successfully completed the reset sequence and is ready to accept APB / TDR transactions.

2.4 Load IMEM / DMEM Memory (Step C)

Loading the firmware is accomplished by accessing the PMU SRAMs over the APB bus or JTAG. 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 ATE firmware is provided in three formats:

- Binary image (.bin files)
- Text files with APB address/data pairs (.incv files)
- Text 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 address with the associate data. To use the binary image file, write the binary contents of the file 16 bits at a time to the memory.

There is a separate file for:

- The instruction memory image (ddr_ate_imem.bin or ddr_ate_ate_imem.incv or ddr_ate_i-mem_ecc.txt)
- The data memory image (ddr_ate_dmem.bin or ddr_ate_dmem.incv or ddr_ate_dmem_ecc.txt)

Both the instruction memory and data memory images must be loaded. The instruction memory image contains the executable ATE code. The data memory image contains data structures that the instruction memory image needs to run.

Table 2-1 SRAM Address Map

Memory	Start Address	End Address	Data Width per Address Increment
IMEM	0x50000	0x57FFF (64kB)	16
DMEM	0x58000	0x5FFFF (64kB)	16



Note

While loading firmware IMEM/DMEM images, the csrMicroReset must be set so that {Reset = 0; stall =1 }.



Note

The IMEM and DMEM are physically 32bits wide. Therefore, writes to the IMEM and DMEM must be made in pairs: an even 16bit address followed by the subsequent 16bit odd address. For example, a write to 0x58000 must be followed by a write to 0x58001.



Note

The provided _ecc.txt files are provided for simulation backdoor memory initialization.

2.5 Configure Technology Specific Registers (Step D1)

There are some CSR registers that need to be set differently based on the chip technology used. These registers need to be set to the correct values before the ATE firmware is executed. Set the following registers:

1. PLLCtrl1 CSR – See section “Optimal PLL Settings” in the PHY Databook for the correct values for all fields.
2. PLLCtrl4 CSR – See section “Optimal PLL settings” in the PHY Databook for the correct values for all fields.
3. PLLUPllProg0/1/2/3 CSR – See section “Optimal PLL settings” in the PHY Databook for the correct values for all fields.



Note

The PLL configuration table to reference depends on whether the PLL is in x4 or x8 mode. The PLL is in x4 mode when ClockingMode=0 and DfiClkFreqRatio=1 (corresponding to LP4 1:2 mode). For all other modes, the PLL is in x8 mode. See section 4.1 on page 44 for more details on the message block inputs ClockingMode and DfiClkFreqRatio.

2.6 Configure ARC Clock Rate (Step D2)

The ARC micro-processor clock frequency can be configured to be either DfiClk frequency, or half-DfiClk frequency. The ARC rate must be configured prior to executing the firmware, and is selected by programming the CSR field UcclkFull in the CSR UcclkHckEnables.

Setting UcclkFull=0x0 enables half-rate mode ($\text{ArcClk} = \text{DfiClk} / 2$), while setting UcclkFull=0x1 enables full-rate mode ($\text{ArcClk} = \text{DfiClk}$). Refer to the PUB databook for CSR description of UcclkHckEnables. When programming the field UcclkFull, the other fields in the CSR UcclkHckEnables must be left at their reset values.

Half-Rate ARC	Full-Rate ARC
$\text{ArcClk} = \text{DfiClk} / 2$	$\text{ArcClk} = \text{DfiClk}$
CSR UcclkFull=0x0	CSR UcclkFull=0x1

**Note**

Only the CSR field UcclkFull in the CSR UcclkHckEnables may be programmed by the user: the other fields in UcclkHckEnables must be left at their reset values.

**Note**

Full-Rate mode is only supported if the ARC processor timing has been closed at DfiClk rates.

2.7 Configuring the Firmware Via the Message Block (Step D3)

For the ATE firmware to run, it needs some basic information about the system and customer options. Input is passed to the firmware by using a data structure called the message block. This data structure is also used to pass information back from the firmware to the user.

The message block data structure always starts at address 0 of the data memory, and must be configured over APB or JTAG prior to running the firmware. The format of the message block is provided as a C data structure in a header file for convenience. The header file (`mnPmuSramMsgBlock_ate.h`) can be found in the release package in the directory with firmware image.

The message block fields that are annotated as “input” must be written by the customer before running the firmware. The fields that are annotated as “output” provide information after firmware has completed. See the message block field descriptions in the header file for the function of each field.

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



Note

The format of the message block is provided as a C data structure in a header file for convenience. The header file (`mnPmuSramMsgBlock_ate.h`) can be found in the release package in the directory with firmware image.



Note

The IMEM and DMEM are physically 32bits wide. Therefore, writes to the IMEM and DMEM must be made in pairs: an even 16bit address followed by the subsequent 16bit odd address. For example, a write to 0x58000 must be followed by a write to 0x58001.



Note

While writing or reading the Message Block in the DMEM, the `csrMicroReset` must be set so that `{ ResetToMicro = 0; StallToMicro = 1 }`.

2.8 Execute the Firmware (Step E)

Once the test ATE firmware image has been loaded and the message block configured, the PHY Microcontroller Unit (PMU) can begin running the firmware.

Once the firmware is running, it will run all the tests that were configured to run. It will write the results back to the message block so that once it is finished the results can be read.

To execute the ATE firmware, do the following steps:

1. Initialize the PHY:
 - a. Write the MicroContMuxSel CSR to 1 to give control of the internal CSR bus to the PMU
2. Start the firmware:
 - a. Reset MicroController {csrMicroReset: ResetToMicro = 1; StallToMicro = 1}
 - b. De-assert MicroController {csrMicroReset: ResetToMicro = 0 ; StallToMicro = 1}
 - c. Start MicroController {csrMicroReset: ResetToMicro = 0 ; StallToMicro = 0}
3. Wait for the ATE firmware to finish:
 - a. These methods are indicated in section 3.3 on page 34.
4. Force the microcontroller into stall, and enable APB accesses to all PHY registers
 - a. Set the StallToMicro bit in the MicroReset CSR to 1 to cause the PMU to stop running, leave the other fields at their current settings
 - b. Write the MicroContMuxSel CSR to 0 to give control of the internal CSR bus to the APB/JTAG. Read the results: PASS/FAIL status can be read from the PassFailResults field in the message block.
 - c. For more information, the entire message block output data structure can be read:
 - i. Read the test results from the message block (see section 2.9 on page 29)

2.9 Read the Message Block Results (Step F)

The firmware will use the fields in the message block that are marked as output to send information back to the user. These fields will contain the following information:

- Pass / Failure of the individual tests
- Result data for various tests

See the message block field descriptions in the detailed descriptions for each test for more information on the message block contents for each test in [chapter 4](#) on [page 43](#).



Note

While writing or reading the Message Block in the DMEM, the `csrMicroReset` must be set so that `{ ResetToMicro = 0; StallToMicro = 1 }`.

2.10 Performing more testing

To run another test, go back to step D2 (section [2.5](#) on page [25](#)) and repeat. The firmware IMEM and DMEM images do not need to be reloaded when running more tests.

Otherwise, the testing is complete.

3

ATE Firmware Usage

This section will explain the usage cases for the ATE firmware, and how the `dwc_ddrphy_ate_main()` function is structured.

The following sections are included in this chapter:

- [“ATE Firmware Use” on page 32](#)
- [“Test Ordering” on page 33](#)
- [“Interacting with the ATE Firmware and Determining When the Firmware is Done” on page 34](#)

3.1 ATE Firmware Use

The ATE firmware is meant to be used in conjunction with the ATPG vectors. The ATPG vectors test the majority of the digital logic, and the ATE firmware will test many of the analog functions (PLL, LCDLs, IO drivers and receivers, among others) of the PHY and some of the digital logic that is not tested as part of the ATPG patterns (for example, some of the clock crossing paths).

3.2 Test Ordering

The order the tests are executed is important. For example, the PLL lock and the impedance calibration tests are run before the loopback tests. The tests are always run in the following order:

- a. Revision Number Check
- b. LCDL Linearity
- c. Impedance Calibration
- d. PLL Lock
- e. RxReplica Calibration
- f. Address/Command Loopback
- g. Data Loopback 1D Eye
- h. Data Loopback 2D Eye
- i. DCA Loopback
- j. Burn In

3.3 Interacting with the ATE Firmware and Determining When the Firmware is Done

The ATE firmware is designed so that a minimum of interaction is required. The ATE firmware has three ways to determine when it has finished running:

1. Use the formulas in section 3.1.1 and 3.1.2 to calculate the maximum amount of time the firmware will take to run for the enabled tests.
2. Poll the UctWriteProtShadow CSR bit. The ATE firmware will set this CSR bit to 1 while it is running and set it to 0 when the ATE firmware is complete.
3. Watch the Master pin BP_MEMRESET_L. It will be set to 0 while the ATE firmware is running and will be set to 1 when the ATE firmware is finished.

Any of these methods can be used to determine when the ATE firmware is finished running.

3.3.1 ATE Firmware Run Time Equations (Half-Rate ARC)

To get the half-rate ARC mode total run time, sum up the “Test Main Loop” time plus the time for each enabled test. The following table indicates the worst case run time for each test.



Note The following equations are to be used for half-rate ARC mode (UcclkFull=0x0). Refer to section 2.6 for more details.

Test Name	Run time (Half-Rate ARC)
Test Main Loop	12,140 DFICKS
Revision Number Check	590 DFICKS
Impedance Calibration	63uS + 6,470 DFICKS
PLL / LCDL Lock	16us + 21,550 DFICKS + (7,570 * NumDbytes) + (2,470 * NumAc) + MemClkToggle * (870 + MemClkTime) DFICKS
RxReplica Calibration	Following Equation Is In Terms of DFICKS 370 + 1,350*NumDbytes
LCDL Linearity	Following Equation Is In Terms of DFICKS $18,110 + 28,430 * \text{NumAC} + 84,200 * \text{NumDbytes} + 3 * \text{LcdIClksToRun} * (((\text{LcdIEndPhase} - \text{LcdIStartPhase}) / \text{LcdIStride}) + 2) + (12,600 * \text{NumAC} + 37,520 * \text{NumDbytes} + 2720) * ((\text{LcdIEndPhase} - \text{LcdIStartPhase}) / \text{LcdIStride})$

Test Name	Run time (Half-Rate ARC)
Address / Command Loopback	<p>Following Equations Are In Terms of DFICLKs</p> <p>Note: In equation below:</p> <ul style="list-style-type: none"> ■ AcFreqRatio = 1 when AC(s) are in 1:2 mode ■ AcFreqRatio = 2 when AC(s) are in 1:4 mode ■ AcLoopPadLoopBk = ! AcLoopCoreLoopBk <p>Refer to section 4.1.3 for details.</p> <p>If AcLoopMinLoopPwr = 0 (or AcLoopMinLoopPwr = 1 and NumAc = 1):</p> $49,760 * \text{AcFreqRatio} + \text{AcLoopClksToRun} * ((127 / \text{AcLoopIncrement}) + 2) + 4,800 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 2,700 * \text{NumAC} * ((127 / \text{AcLoopIncrement}) + 1) + 4,830 * \text{NumAC} + (((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 1) * (320 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} * \text{NumAC} + 1,130 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 660 * \text{NumAC}) + 1,020 * (127 / \text{AcLoopIncrement}) + 1,830 * ((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 11,040$ <p>If AcLoopMinLoopPwr = 1 and NumAc = 2:</p> $\text{AcLoopClksToRun} * (\text{NumAC} * ((127 / \text{AcLoopIncrement}) + 1) + 1) + 5,250 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + \text{NumAC} * (49,700 * \text{AcFreqRatio} + 9,690) + \text{NumAC} * (270 * \text{NumAC} + 3,440) * ((127 / \text{AcLoopIncrement}) + 1) + \text{NumAC} * (((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 1) * (130 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} * \text{NumAC} + 1,310 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 220 * \text{NumAC} + 2,280) + 4,440$

Test Name	Run time (Half-Rate ARC)
Data Loopback 1D	<p>Following Equations Are In Terms of DFICLKs</p> <p>Note: In equation below:</p> <ul style="list-style-type: none"> ■ DbyteFreqRatio = 1 when Dbytes are in 1:2 mode ■ DbyteFreqRatio = 2 when Dbytes are in 1:4 mode ■ DatLoopPadLoopBk = !DatLoopCoreLoopBk ■ dqs_en = DatLoopDiffTestMode bit[0] DatLoopCoreLoopBk ■ t_or = DatLoopDiffTestMode bit[1] DatLoopDiffTestMode bit[3] ■ t_and = DatLoopDiffTestMode bit[1] & DatLoopDiffTestMode bit[3] ■ c_or = DatLoopDiffTestMode bit[2] DatLoopDiffTestMode bit[4] ■ c_and = DatLoopDiffTestMode bit[2] & DatLoopDiffTestMode bit[4] <p>Refer to section 4.1.3 for details.</p> <p>If DatLoopMinLoopPwr = 0:</p> $2 * \text{DatLoopPadLoopBk} * (10,890 * c_or + 10,890 * t_or + 7,250) + 2,800 * \text{NumDbytes} + (\text{DatLoopClksToRun} + 2,890 * \text{NumDbytes}) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) * (400 * \text{DatLoopPadLoopBk} * \text{NumDbytes} * (\text{DatLoopDiffTestMode} > 0) + 240 * \text{DatLoopPadLoopBk} * (c_and + dqs_en + t_and) + 92 * \text{DfiClkFreqRatio} * \text{NumDbytes}) + 1,420 * (64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 685 * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 59,520$ <p>If DatLoopMinLoopPwr = 1:</p> $55,460 * \text{DatLoopPadLoopBk} + 220 * \text{NumDbytes} * \text{NumDbytes} * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + 4 * \text{NumDbytes} * \text{NumDbytes} * (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) * (9 * \text{DatLoopPadLoopBk} * (c_and + dqs_en + t_and) + 51 * \text{DatLoopPadLoopBk} * (\text{DatLoopDiffTestMode} > 0) + 5 * \text{DfiClkFreqRatio}) + \text{NumDbytes} * (\text{DatLoopClksToRun} + 4,160) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + 21 * \text{NumDbytes} * (1,110 * \text{DatLoopPadLoopBk} * (c_or + t_or) + 1,350) + 1,140 * \text{NumDbytes} * (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) + 7,050$

Test Name	Run time (Half-Rate ARC)
Data Loopback 2D	<p>Following Equations Are In Terms of DFICLKs</p> <p>Note: In equation below:</p> <ul style="list-style-type: none"> ■ $\text{DatLoopPadLoopBk} = !\text{DatLoopCoreLoopBk}$ <p>If $\text{DatLoopMinLoopPwr} = 0$:</p> $39,820 * \text{DatLoopPadLoopBk} + 2,030 * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) + (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) * (\text{DatLoopClksToRun} + 2,510 * \text{NumDbytes} + 1,320) + 57,700$ <p>If $\text{DatLoopMinLoopPwr} = 1$:</p> $69,550 * \text{DatLoopPadLoopBk} + 109 * \text{NumDbytes} * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + \text{NumDbytes} * (\text{DatLoopClksToRun} + 3,670) * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + 3,510 * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) + 30,850$
DCA Loopback	<p>Following Equations Are In Terms of DFICLKs</p> <p>In the follow equations:</p> $\text{NumDcaCoarse} = 4 - \text{CountOnes}(\text{DcaLoopDcaCoarseSkip})$ <p>If $\text{DcaLoopMinLoopPwr} = 0$:</p> $250 * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + 6,000 * \text{NumDcaCoarse} * (12 / \text{DcaLoopDcaFineIncr}) + 2 * [\text{NumDcaCoarse} * ((12 / \text{DcaLoopDcaFineIncr}) + 1) - 1] * [140 * \text{NumDbytes} * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 820 * \text{NumDbytes} + 375 * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 860 * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr})]$ <p>If $\text{DcaLoopMinLoopPwr} = 1$:</p> $110 * \text{NumDbytes} * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + \text{NumDbytes} * [\text{NumDcaCoarse} * ((12 / \text{DcaLoopDcaFineIncr}) + 1) - 1] * [150 * \text{NumDbytes} * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 880 * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 4,860] + 1,000 * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + 2,610 * \text{NumDcaCoarse} * (12 / \text{DcaLoopDcaFineIncr})$

3.3.2 ATE Firmware Run Time Equations (Full-Rate ARC)

To get the full-rate ARC total run time, sum up the “Test Main Loop” time plus the time for each enabled test. The following table indicates the worst case run time for each test.



Note

The following equations are to be used for half-rate ARC mode (UcclkFull=0x1). Refer to section 2.6 for more details.

Test Name	Run time (Full-Rate ARC)
Test Main Loop	7,270 DFICKS
Revision Number Check	310 DFICKS
Impedance Calibration	63uS + 5,270 DFICKS
PLL / LCDL Lock	16uS + $18,050 + 1,290 * \text{NumAC} + 3,940 * \text{NumDbytes} + \text{MemClkToggle} * (\text{MemClkTime} + 510)$ DFICKS
RxReplica Calibration	Following Equation Is In Terms of DFICKS $190 + 700 * \text{NumDbytes}$
LCDL Linearity	Following Equation Is In Terms of DFICKS $9,600 + 14,280 * \text{NumAC} + 42,480 * \text{NumDbytes} + 3 * \text{LcdlClksToRun} * (((\text{LcdlEndPhase} - \text{LcdlStartPhase}) / \text{LcdlStride}) + 2) + (6,370 * \text{NumAC} + 18,930 * \text{NumDbytes} + 1,630) * ((\text{LcdlEndPhase} - \text{LcdlStartPhase}) / \text{LcdlStride})$

Test Name	Run time (Full-Rate ARC)
Address / Command Loopback	<p>Following Equations Are In Terms of DFICLKs</p> <p>Note: In equation below: AcFreqRatio = 1 when AC(s) are in 1:2 mode AcFreqRatio = 2 when AC(s) are in 1:4 mode AcLoopPadLoopBk = !AcLoopCoreLoopBk Refer to section 4.1.3 for details.</p> <p>If AcLoopMinLoopPwr = 0 (or AcLoopMinLoopPwr = 1 and NumAc = 1):</p> $24,940 * \text{AcFreqRatio} + \text{AcLoopClksToRun} * ((127 / \text{AcLoopIncrement}) + 2) + 2,560 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 1,390 * \text{NumAC} * (\text{floor}(127 / \text{AcLoopIncrement}) + 1) + 2400 * \text{NumAC} + (((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 1) * (170 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} * \text{NumAC} + 650 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 340 * \text{NumAC}) + 640 * (127 / \text{AcLoopIncrement}) + 1,050 * ((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 6,080$ <p>If AcLoopMinLoopPwr = 1 and NumAc = 2:</p> $\text{AcLoopClksToRun} * (\text{NumAC} * ((127 / \text{AcLoopIncrement}) + 1) + 1) + 2,920 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 4 * \text{NumAC} * (6,250 * \text{AcFreqRatio} + 1,290) + \text{NumAC} * (140 * \text{NumAC} + 1,890) * ((127 / \text{AcLoopIncrement}) + 1) + \text{NumAC} * (\text{floor}((256 * \text{AcFreqRatio} - 1) / \text{AcLoopIncrement}) + 1) * (70 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} * \text{NumAC} + 750 * \text{AcLoopDiffTestMode} * \text{AcLoopPadLoopBk} + 108 * \text{NumAC} + 1,290) + 2,640$

Test Name	Run time (Full-Rate ARC)
Data Loopback 1D	<p>Following Equations Are In Terms of DFICKS</p> <p>Note: In equation below:</p> <ul style="list-style-type: none"> ■ DbyteFreqRatio = 1 when Dbytes are in 1:2 mode ■ DbyteFreqRatio = 2 when Dbytes are in 1:4 mode ■ dqs_en = DatLoopDiffTestMode bit[0] DatLoopCoreLoopBk ■ t_or = DatLoopDiffTestMode bit[1] DatLoopDiffTestMode bit[3] ■ t_and = DatLoopDiffTestMode bit[1] & DatLoopDiffTestMode bit[3] ■ c_or = DatLoopDiffTestMode bit[2] DatLoopDiffTestMode bit[4] ■ c_and = DatLoopDiffTestMode bit[2] & DatLoopDiffTestMode bit[4] <p>Refer to section 4.1.3 for details.</p> <p>If DatLoopMinLoopPwr = 0:</p> $\text{DatLoopPadLoopBk} * (12,300 * c_or + 12,300 * t_or + 9,460) + 1,540 * \text{NumDbytes} + (\text{DatLoopClksToRun} + 1,540 * \text{NumDbytes}) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) * (210 * \text{DatLoopPadLoopBk} * \text{NumDbytes} * (\text{DatLoopDiffTestMode} > 0) + 120 * \text{DatLoopPadLoopBk} * (c_and + dqs_en + t_and) + 43 * \text{DfiClkFreqRatio} * \text{NumDbytes}) + 860 * (64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 460 * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 29,090$ <p>If DatLoopMinLoopPwr = 1:</p> $21,130 * \text{DatLoopPadLoopBk} + 31 * \text{NumDbytes} * \text{NumDbytes} * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + \text{NumDbytes} * \text{NumDbytes} * (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) * (60 * \text{DatLoopPadLoopBk} * (c_and + dqs_en + t_and) + 13 * \text{DatLoopPadLoopBk} * (\text{DatLoopDiffTestMode} > 0) + 40 * \text{DfiClkFreqRatio}) + \text{NumDbytes} * (\text{DatLoopClksToRun} + 2,360) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + \text{NumDbytes} * (11,740 * \text{DatLoopPadLoopBk} * (c_or + t_or) + 7,620) + 500 * \text{NumDbytes} * (((256 * \text{DfiClkFreqRatio} - 1) / \text{DatLoopFineIncr}) + 1) + 15,250$

Test Name	Run time (Full-Rate ARC)
Data Loopback 2D	<p>Following Equations Are In Terms of DFICKS</p> <p>If DatLoopMinLoopPwr = 0:</p> $37,150 * \text{DatLoopPadLoopBk} + 1,200 * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) + (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) * (\text{DatLoopClksToRun} + 1,330 * \text{NumDbytes} + 820) + 21,910$ <p>If DatLoopMinLoopPwr = 1:</p> $52,640 * \text{DatLoopPadLoopBk} + 56 * \text{NumDbytes} * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + \text{NumDbytes} * (\text{DatLoopClksToRun} + 2,090) * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) * ((64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart}) / \text{DatLoopFineIncr}) + 1) + 2,060 * \text{NumDbytes} * (((\text{DatLoop2dVrefEnd} - \text{DatLoop2dVrefStart}) / \text{DatLoop2dVrefIncr}) + 1) + 7,040$
DCA Loopback	<p>Following Equations Are In Terms of DFICKS</p> <p>In the follow equations: $\text{NumDcaCoarse} = 4 - \text{CountOnes}(\text{DcaLoopDcaCoarseSkip})$</p> <p>If DcaLoopMinLoopPwr = 0:</p> $130 * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + 3,280 * \text{NumDcaCoarse} * (12 / \text{DcaLoopDcaFineIncr}) + (\text{NumDcaCoarse} * ((12 / \text{DcaLoopDcaFineIncr}) + 1) - 1) * (130 * \text{NumDbytes} * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 850 * \text{NumDbytes} + 465 * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr})) + 520 * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr})$ <p>If DcaLoopMinLoopPwr = 1:</p> $53 * \text{NumDbytes} * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + \text{NumDbytes} * (\text{NumDcaCoarse} * ((12 / \text{DcaLoopDcaFineIncr}) + 1) - 1) * (96 * \text{NumDbytes} * ((\text{DcaLoopDelayIncr} + 191) / \text{DcaLoopDelayIncr}) + 510 * ((\text{DcaLoopDelayIncr} + 200) / \text{DcaLoopDelayIncr}) + 2,130) + 640 * \text{NumDbytes} * ((256 * \text{DfiClkFreqRatio} - 1) / \text{DcaLoopDelayIncr}) + 3,720 * \text{NumDcaCoarse} * (12 / \text{DcaLoopDcaFineIncr})$

4

Individual ATE Test Details

The following sections are included in this chapter:

- “General Message Block Input Fields” on page 44
- “Revision Number Check” on page 48
- “Impedance Calibration” on page 49
- “PLL Lock / LCDL Calibration” on page 51
- “RxReplica Calibration Test” on page 56
- “LCDL (Locally Calibrated Delay Line) Linearity” on page 57
- “Address/Command Loopback” on page 64
- “Data Loopback 1D” on page 77
- “Data loopback 2D” on page 88
- “DCA Loopback” on page 94
- “Burn-in” on page 101

The following sections give more detailed information on each of the tests that are included in the ATE firmware. The information on each will include:

- Message block fields used and legal / recommended values
- Pass / Fail conditions
- The output results for each test

For all the tests, the input fields of the message block must be configured prior to starting the execution of the firmware.

4.1 General Message Block Input Fields

There are several message block fields that are used by all the tests and must be set correctly for any test to run. These message block inputs are:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the desired test(s) to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test is run and passes. The firmware will set the bit to 0 if the test fails or is not run.
TestOptions[0]	Input	This value must be set to 0 unless directed by Synopsys.	Setting this bit to 1 sets the PllLockPhSel field of the PllCtrl0 CSR to 2'b11. This increases lock time detection and is necessary for some technologies.
TestOptions[1]	Input	This value must be set to 0 unless directed by Synopsys.	Setting this bit to 0 sets the PLL reset time to 1us. This is the default setting and must be used unless directed by Synopsys. Setting this bit to 1 increases the PLL reset time to 3us. This is necessary for some technologies.
TestOptions[2]	Input	1 = Drive AC traffic while testing the Dbyte SE slices during the data loopback test. The traffic driven is based on TestOptions[3]. 0 = AC pins are idle during data loopback test	While testing the Dbyte SE slices, traffic can be optionally driven on the AC slices and the Dbyte WCK DIFF slices. When enabled, TestOptions[3] is used to determine the traffic type to drive. Note: AC specific drive strengths must be properly configured to ensure transmitted values appear on the AC bumps. Note: WCK slice is only active if the PHY is LP5 enabled, and ClockingMode=0x1.
TestOptions[3]	Input	1 = Drive PRBS pattern on AC SE slice 0 = Drive mission mode commands on the AC slice interfaces.	If TestOptions[2]=1, TestOptions[3] is used to select the traffic pattern to drive on the AC SE slices + Dbyte WCK slices. If TestOptions[2]=0, TestOptions[3] is a don't-care.
TestOptions[15]	Input	This value must be set to 0 unless directed by Synopsys.	Setting this bit to 0 enables the CSR save/restore operation. This is the default setting and must be used unless directed by Synopsys.

Field Name	Direction	Legal / Recommended Values	Comments
PubRev	Input	This value must be set to 0 unless directed by Synopsys.	Override the value in the CSR PUBREV when programmed with a non-zero value.
DfiClkFreq	Input	Must be set to match the input DFI Clock Frequency in MHz	For example, enter 0x0320 for 800MHz.
ClockingMode	Input	LP4 Mode = 0 LP5 Mode = 1	Controls the relationship between DfiClk, CK, and DQS/WCK. Refer to section 4.1.3.
DfiClkFreqRatio	Input	1:2 Mode = 1 1:4 Mode = 2	In 1:2 mode: ■ DQS = 2xDfiClk frequency In 1:4 mode: ■ DQS = 4xDfiClk frequency Refer to section 4.1.3.
DacRefModeCtl	Input	LP5/LP4X mode = 0 (VrefDacRef=VDDq) LP4 mode = 1 (VrefDacRef=VAA)	Refer to field of same name in register csrVrefDacRefCtl.
UseMsgBlkPhyCfg	Input	Use csrPhyConfig contents = 0 Use messageBlock contents = 1	See section 4.1.4

4.1.1 TestsToRun and PassFailResults Message Block Fields

The TestsToRun message block field is used to tell the firmware which of the ATE tests to run. It is a bit vector with a bit per test. If a bit is set to (1), the corresponding test will be run. If a bit is (0), the corresponding test will not be run.

The PassFailResults message block field is used to indicate the test pass/fail results of all the tests. It is a bit vector with a bit per test. It uses the same bit to test mapping as the TestsToRun message block field. A (1) in the corresponding bit position indicates that the test has passed. A (0) in the corresponding bit position indicates that the test has either failed, or was not run.

The ATE firmware may set more bits in the PassFailResults vector than the user has indicated in the TestsToRun field. For example, if the user indicates that only the Data Loopback 1D test should be run, the firmware will automatically run the Impedance Calibration and PLL/LCDL Lock tests, because these tests are prerequisites for any of the loopback tests. The firmware will set the PassFailResults bits for both the Impedance Calibration and PLL/LCDL Lock tests in this case.

Because of this, the user can determine ATE pass/fail by using the following equation:

$$\text{ATE_Pass} = ((\text{PassFailResults} \& \text{TestsToRun}) == \text{TestsToRun})$$

This will mask off the extra tests that the firmware automatically ran, or the user indicated should not be run.

If the automatically added tests are included in the TestsToRun setting, a simple equal test can be performed:

$$\text{ATE_Pass} = (\text{PassFailResults} == \text{TestsToRun})$$

4.1.2 TestsToRun and PassFailResults Bit Mapping

The TestsToRun and the PassFailResults message block field both use the same bit mapping:

Field bit number	Test / Result
[0]	DMEM / IMEM revision check
[1]	Impedance Calibration
[2]	PLL Lock / LCDL Lock
[3]	LCDL Linearity
[4]	Address/Command Loopback
[5]	Data Loopback 1D
[6]	Data Loopback 2D
[7]	Burn-In
[8]	RxReplica Calibration
[9]	DCA Loopback

For the TestsToRun field, if the corresponding bit is (1) then the test will be run. If the corresponding bit is (0), the test will not be run. Note that the impedance calibration and PLL Lock tests will automatically be run if the Address/Command Loopback, Data Loopback 1D, Data Loopback 2D, DCA Loopback, or Burn-In tests are set to run.

For the PassFailResults field, if the corresponding bit is (1) then the test ran and passed. If the corresponding bit is (0), the test failed or was not run.

4.1.3 DfiClkFreqRatio and Clocking Mode

The following table summarizes how MemClk and DQS related to DfiClk, as a function of the message block inputs ClockingMode (LP4/LP5) and DfiClkFreqRatio (1:2/1:4).

	LP4		LP5	
	1:2	1:4	1:2	1:4
MemClk	2* DfiClk	4* DfiClk	1* DfiClk	1* DfiClk
DQS/WCK	2* DfiClk	4* DfiClk	2* DfiClk	4* DfiClk

The following table summarizes how the frequency ratio of the AC and Dbyte macros are affected by the values of ClockingMode and DfiClkFreqRatio. As seen below, if the ClockingMode is set to LP5 mode, the ACs always run in 1:2 mode.

	LP4	LP5
AC Frequency Ratio	DfiClkFreqRatio	1:2
Dbyte Frequency Ratio	DfiClkFreqRatio	

4.1.4 UseMsgBlkPhyCfg Behaviour

The user has the option of manually specifying the PHY configuration, or telling ATE FW to automatically read the configuration from the PhyConfig CSR.

If UseMsgBlkPhyCfg is set to 0, ATE FW will read csrPhyConfig, and use the contents to test the PHY.

If UseMsgBlkPhyCfg is set to 1, the user must populate the following message block inputs, as they will be used to determine how to test the PHY.

Field	Valid Values	Notes
PhyCfgNumChan	0x1 = 1 channel 0x2 = 2 channels	
PhyCfgNumDbPerChan	0x2 = 2 Dbytes per channel	
PhyCfgDmiEn	0x0 = DMI is disabled / does not exist 0x1 = DMI is enabled	
PhyCfgNumRank	0x1 = 1 rank 0x2 = 2 ranks	
PhyCfgLp5En	0x0 = PHY does NOT support LP5 0x1 = PHY supports LP5	If equal to (1), LP5 specific checking will be performed.

4.2 Revision Number Check

This test is a consistency check on the ATE IMEM revision versus the DMEM revision of the firmware, to make sure they have the same version number. The test will pass if the revision numbers are the same, and the test will fail if the revision numbers are different. Customers are encouraged to run the check during ATE to ensure that the IMEM and DMEM images are consistent. The observed IMEM and DMEM revision numbers are both saved to the message block.

Test also performs a register test (write, followed by read and check) on a single PUB CSR as a simple connectivity check.

Fail Condition(s):

- The revision numbers of the IMEM and the DMEM do not match.
- The CSR write / read / check fails.



The revision number for a release can be found in the `mnPmuSramMsgBlock_ate.h` file that is included with the release in the directories that contain the IMEM and DMEM images. The revision number is indicated as two C define statements, `PMU_ATE_INTERNAL_REV1` and `PMU_ATE_INTERNAL_REV0`. `PMU_ATE_INTERNAL_REV1` indicates the upper four digits of the revision number and `PMU_ATE_INTERNAL_REV0` indicates the lower four digits.

4.2.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Revision Number Check test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes, and set the bit to 0 if the test fails.
AtelmemRevision	Output	Any	Will be set to the IMEM revision after the test is run.
AteDmemRevision	Output	Any	Will be set to the DMEM revision after the test is run.

4.3 Impedance Calibration

This test causes the impedance calibration engine to run once, and then writes the results to the message block. The calibration engine uses the externally provided resistor (value specified by ZCalRZN message block input) to perform the calibration of the driver impedances.

This tests the analog components and the digital logic associated with the calibration engine, which are not tested as part of the ATPG patterns.

The results will be dependent upon the value of the external calibration resistor and are susceptible to process, voltage, and temperature (PVT) variation. It is up to the user to fully account for variances outside the PHY's control, such as PVT variation and external resistor variance.

The user has the option to enable continuous impedance recalibration. If selected, then the CalInterval message block input must also be configured.

If continuous calibration is enabled, the firmware will push the updated codes to the drivers before traffic is driven.

See the CSR descriptions in the PUB documentation for details on how to program the ZCalRZN and ZCalCompVref fields.

Fail Condition(s):

- Impedance calibration engine does not finish within the expected time.
- ZCalCompResult = 0x00 or 0x7F
- ZCalCodePU = 0x00 or 0xFF
- ZCalCodePD = 0x00 or 0xFF



Note

The Verilog models do not model the analog behavior of the voltage comparator. If no ZCal behavioural model is used, this test is expected to indicate failure in simulation. Therefore, it is recommended to use the ZCal model from the CTB in the user's environment (see section 4.3.2).

4.3.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Impedance Calibration test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to (1) if the test passes and set the bit to (0) if the test fails.

Field Name	Direction	Legal / Recommended Values	Comments
ContinuousCal	Input	Disable = 0x0 Enable = 0x1	If enabled, the impedance calibrator will run periodically for the duration that the ATE tests run. Should be enabled if the user will also be running pad-side loopback. If enabled, the firmware will push updated calibration codes to the drivers prior to driving traffic. Note: Must be set to (0) if ZCal model is not being used. Refer to section 4.3.2 for more details.
CalInterval	Input	See ZCalRate CSR description for more details	If ContinuousCal is (1), specify how often impedance calibration should run.
ZCalRZN	Input	Must be set to match the external calibration resistor.	See CSR of the same name for description of contents.
ZCalCompVref	Input	Voltage reference used by the impedance calibration comparator.	See CSR of the same name for description of contents.
ZCalCodePU	Output	See CSR of the same name for description of contents.	Pull-up impedance code
ZCalCodePD	Output	See CSR of the same name for description of contents.	Pull-down impedance code
ZCalCompResult	Output	See CSR of the same name for description of contents.	Comparator offset calibrator code

4.3.2 Impedance Calibrator Behavioural Model

It is recommended to include the ZCal behavioral model from the CTB (`dwc_ddrphy_zcalana_bhv.v`) in the user's simulation environment. The CTB behavioral model includes a command which will auto-bind it to the ZCal, and ensures the impedance calibrator codes converge to legal values.

4.4 PLL Lock / LCDL Calibration

The PLL will be initialized based on the values of DfiClkFreqRatio and Clocking mode, as defined in the message block (see section 4.1.3). The test will then attempt to lock to the provided input clock. Once the lock process is complete, it will watch the PLL Lock signal for 512 DFICLKs, and make sure the PLL stays locked. The result of the PLL lock procedure is then reported in the message block output PLLResults.

If the PLL locks within the time indicated in the PHY Databook, and remains locked, this portion of the test will pass. If the PLL fails to lock within the time in the PHY databook, or loses lock, the entire PLL lock test will report failure.

The test will also check each LCDL lock code, to make sure that they fall within the expected range, based on the frequency that the test is being run. If any LCDL falls outside of the expected range, the PLL lock test will report failure.

Fail condition(s):

- The PLL lock bit is not set after the amount of time required by the specification.
- The PLL lock bit falls during the 512 DFICLK sample window.
- Any AC/Dbyte LCDL calibration code falls outside of the expected range.

4.4.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the PLL Lock / LCDL Calibration test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes and set the bit to 0 if the test fails.
TestOptions[0]	Input	This value must be set to 0 unless directed by Synopsys.	Setting this bit to 1 sets the PllLockPhSel field of the PllCtrl0 CSR to 2'b11. This increases lock detection time, and is necessary for some technologies.
TestOptions[1]	Input	This value must be set to 0 unless directed by Synopsys.	Setting this bit to 0 sets the PLL reset time to 1us. This is the default setting and must be used unless directed by Synopsys. Setting this bit to 1 sets the PLL reset time to 3us, which is necessary for some technologies.

Field Name	Direction	Legal / Recommended Values	Comments
MemClkToggle	Input	0x0 = CK does not toggle 0x1 = CK toggles at MemClk 0x2 = CK toggles at DfiClk / 2 0x3 = CK toggles at DfiClk / 4 0x4 = CK toggles at DfiClk / 8 0x5 = CK toggles at DfiClk / 16 0x6 = CK toggles at DfiClk / 24 Others = RESERVED	Optionally enables driving MemClk, or a divided down DfiClk, on the CK pin(s). The input MemClkTime determines for how long the clock is driven for, before ATE proceeds to the next test. The relationship between MemClk and DfiClk is determined by the message block input ClockingMode and DfiClkFreqRatio (section 4.1.3). Note: if MemClkToggle and MemClkTime are both non-zero, the user should also run the Impedance Calibration test, and program AcTxSlewDiff / AcTxImpedanceDiff.
MemClkTime	Input	0x0000 = CK does not toggle. 0xFFFF = Infinite duration. PLL lock test will enter infinite generation test mode (refer to section 4.4.4). OtherValues = test will wait the provided number of DfiClk cycles, following which it will stop the slow-clock and proceed to the next test.	Note: if MemClkToggle and MemClkTime are both non-zero, the user should also run the Impedance Calibration test, and program AcTxSlewDiff / AcTxImpedanceDiff.
AcTxSlewDiff	Input	Refer to description of CSRs 'TxSlewDIFF*' for field description	Required if MemClkToggle and MemClkTime are both non-zero.
AcTxImpedanceDiff	Input	Refer to description of CSRs 'TxImpedanceDIFF*' for field breakdown	Required if MemClkToggle and MemClkTime are both non-zero.
PLLResults	Output	Bits[3:0] – PLL Lock status 0x1 = PLL lock succeeded 0x2 = PLL lock failed Bits[15:4] - RESERVED	Indication of whether the PLL locking operation was successful or not.
LcdlResultsAc[AcNum]	Output	An array of bit vectors indexed by AC instance.	Each bit for a given AC will have the bit set to (0) if the LCDL passed the range check for the current frequency. It will be set to (1) if the LCDL failed the range check. The bit vector is indexed using different numbering from the LCDL linearity test, and is described in section 4.4.2.

Field Name	Direction	Legal / Recommended Values	Comments
LcdlResultsDb[DbyteNum]	Output	An array of bit vectors, indexed by Dbyte instance.	Each bit for a given Dbyte will have the bit set to (0) if the LCDL passed the range check for the current frequency. It will be set to (1) if the LCDL failed the range check. The bit vector is indexed using different LCDL numbering from the LCDL linearity test, and is described in section 4.4.2.
LcdlResultsRxReplica	Output	Bit vector indexed by Dbyte instance	The bit for a given Dbyte's RxReplica will be set to (0) if the LCDL passed the range check for the current frequency. It will be set to (1) if the LCDL failed the range check. The bit vector is indexed using the Dbyte instance number.

4.4.2 LCDL 1UI Lock Code Instance Numbers

The following table outlines how LCDL numbers map to physical LCDLs in AC and Dbyte instances. This mapping is utilized for selecting LCDLs in the context of the outputs LcdlResultsAc and LcdlResultsDb. The encoding is slightly different from the LCDL Linearity encoding, because the RxReplica LCDLs have been grouped together. Consequently, the RxReplica statuses are placed in the message block output LcdlResultsRxReplica, and indexed by Dbyte instance.

Value (decimal)	Dbyte LCDL	AC LCDL
0	Dbyte TxDq_r0	AC SE0
1	Dbyte TxDq_r1	AC SE1
2	Dbyte TxDq_r2	AC SE2
3	Dbyte TxDq_r3	AC SE3
4	Dbyte TxDq_r4	AC SE4
5	Dbyte TxDq_r5	AC SE5
6	Dbyte TxDq_r6	AC SE6
7	Dbyte TxDq_r7	AC SE7
8	Dbyte TxDq_r8	AC SEC0
9	Dbyte TxDqs	AC SEC1

Value (decimal)	Dbyte LCDL	AC LCDL
10	Dbyte RxEn	AC DIFF
11	Dbyte RxClkT_r0	n/a
12	Dbyte RxClkT_r1	n/a
13	Dbyte RxClkT_r2	n/a
14	Dbyte RxClkT_r3	n/a
15	Dbyte RxClkT_r4	n/a
16	Dbyte RxClkT_r5	n/a
17	Dbyte RxClkT_r6	n/a
18	Dbyte RxClkT_r7	n/a
19	Dbyte RxClkT_r8	n/a
20	Dbyte RxClkC_r0	n/a
21	Dbyte RxClkC_r1	n/a
22	Dbyte RxClkC_r2	n/a
23	Dbyte RxClkC_r3	n/a
24	Dbyte RxClkC_r4	n/a
25	Dbyte RxClkC_r5	n/a
26	Dbyte RxClkC_r6	n/a
27	Dbyte RxClkC_r7	n/a
28	Dbyte RxClkC_r8	n/a
29	Dbyte WCK (If LPDDR5_ENABLED)	n/a

4.4.3 MemClk Toggle

The MemClkToggle field of the message block allows the test to output a clock on the CK pins for measurement. The clock will toggle at one of the following frequencies (controllable by the user through the MemClkToggle message block field): MemClk, DfiClk / 2, DfiClk / 4, DfiClk / 8, DfiClk / 16, or DfiClk / 24. The message block input MemClkTime is used to control the length of time the slow clock will be generated for.

4.4.4 MemClk Toggle Infinite Mode

If MemClkToggle is non-zero, and MemClkTime has been set to 0xFFFF, the ATE firmware will enter “infinite MemClk toggle mode,” during which the selected MemClk will be continuously driven on the CK.

To exit the infinite MemClk toggle mode, the user must return to the START phase, and follow the bring-up sequence described in section 2.1.



If the user has set MemClkToggle!=0 and MemClkTime=0xFFFF, the ATE-FW will enter “infinite MemClk toggle mode.” To exit this mode, the user must return to the START phase, and follow the bring-up sequence described in section 2.1.

4.5 RxReplica Calibration Test

This test tests the RxReplica circuitry in the DQS DIFF slices. Each RxReplica is calibrated, and the resulting path-phase values are checked. Note, this test is only supported for data-rates 1600Mbps and above.

If the data-rate is 3200Mbps or above, the test will also make sure that the first two non-zero path-phases are separated by 1UI (within an error bar of $\pm 14\%$).



Note

The RxReplica calibration test is only supported for data-rates ≥ 1600 Mbps.

Fail condition(s):

- All the path-phases in any RxReplica are equal to 0x0
- All the path-phases in any RxReplica are equal to 0x1FF.
- If the data-rate is ≥ 3200 Mbps
 - The difference between the first two non-zero path phases is not 1UI in any RxReplica (with a margin of $\pm 14\%$).

4.5.1 Message Block Inputs / Outputs and legal values

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the RxReplica calibration test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes, and set the bit to 0 if the test fails.
RxReplica1UiLockCode [DbyteNum]	Output		The 1UI Lock codes for the LCDL in each Dbyte's RxReplica.
RxReplicaPathPhase[D byteNum][PathPhaseNum]	Output		The 5 path-phase values for each RxReplica.

4.5.2 Pre-test Requirements

It is required that the PLL/LCDL lock test be run prior to running the RxReplica calibration test. Because of this requirement, the firmware will automatically enable it whenever the RxReplica calibration test is enabled. The PassFailResults message block field will include the results for it, even if it isn't specified in the TestsToRun field.



Note

Because the PLL Lock test is required for the RxReplica test, this test must be run with a DfIClk within the legal range of the PLL to lock.

4.6 LCDL (Locally Calibrated Delay Line) Linearity

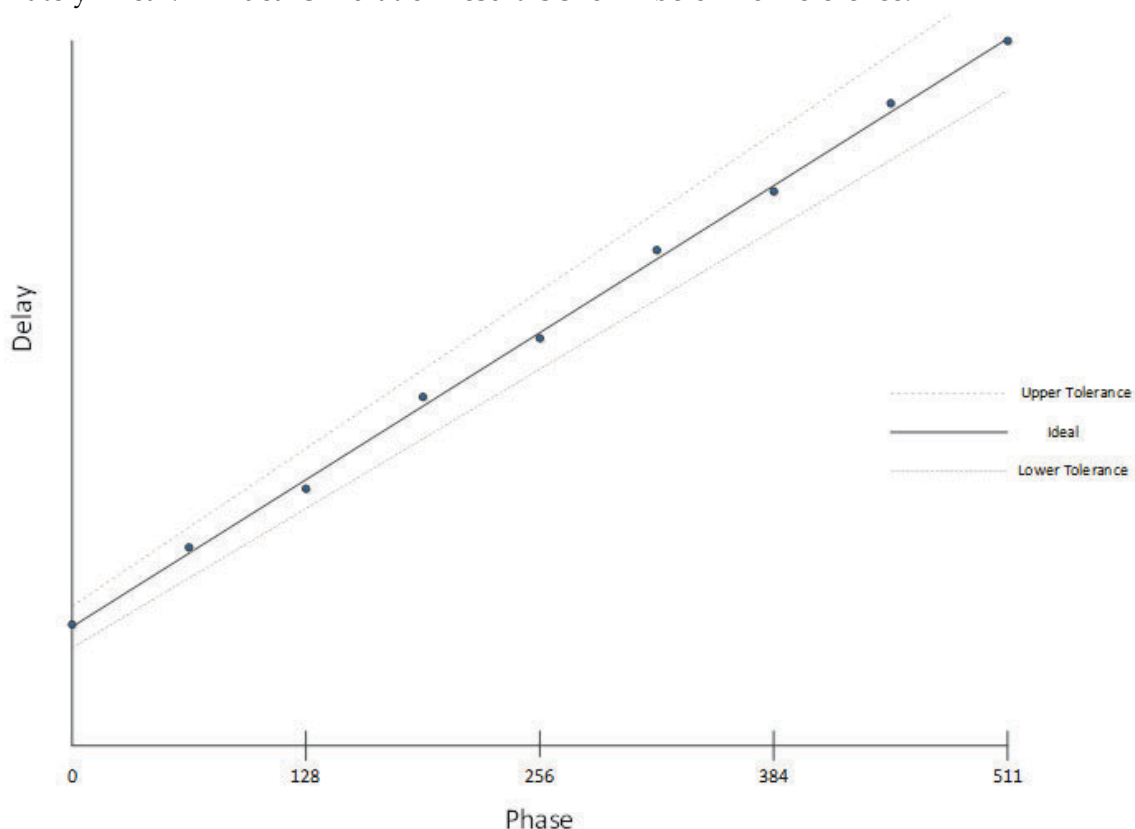


Note

During linearity testing, the delay elements are susceptible to power supply noise. The test environment should strive to provide an extremely clean and stable power supply in order to get the best possible measurements from the test. Additionally, spread spectrum clocking (SSC) on DfiClk could also be a source of error in the measurement.

This test evaluates the linearity of every Locally Controllable Delay Line (LCDL) located in both AC and Dbyte instances. The test measures the delay from the least to the greatest phase setting, and draws a line from the first point to the last point. The test checks to see if the delay at any phase setting deviates from this line more than the percentage indicated by the user. A separate result will be reported for each LCDL, as well as an overall result.

The value of LcdlClksToRun message block input for this test determines the amount of time used to measure each point. The time affects the accuracy for each measurement and with a higher count the measurement tolerance can be set tighter. The graph of delay setting for a given LCDL should be approximately linear. An ideal simulation result is shown below for reference.



To determine the accuracy of each delay element, on completion of each LCDL measurements, two phases are used to calculate a delay slope. Using this slope, the test calculates the delay of the LCDL under test. It uses this calculated delay to predict the value of the oscillator counter. It then measures the oscillator count and calculates the difference between the predicted value and the actual value. The base allowed deviation range is the greater of:

- A static difference of 3 counts
- A 5% variation in the count

The value of the message block input `LcdlPassPercent` affects the percentage variation allowed. It does not affect the static count difference value. The effect of `LcdlPassPercent` on the variation allowed by the firmware is defined by the following formula:

$$\text{Allowed Variation} = \pm \left(\frac{5 * \text{LcdlPassPercent}}{100} \right) \%$$

If every point of an LCDL is within this variation, the LCDL is marked as passing. If any point falls outside this range, the LCDL is marked as failing. Customers should set `LcdlPassPercent` to 100 (0x64).

The values of start phase, end phase, and stride control how many points the test measures, and therefore test time. Each LCDL is composed of 512 “delay elements”. The 512 delay elements are selected by a 9 bit value. The lower 3 bits of the 9 bit delay value use common circuitry. The upper 6 bits use separate circuitry. It is not required to test all 512 settings to test all parts of the circuit. The following combinations of start, end, and stride may be useful:

- Start=0, End=511, Stride=1 – This is an exhaustive test that tests all settings
- Start=0, End=511, Stride=7 – Tests all settings of upper 6 bits, and all settings of lower 3 bits
- Start=0, End=511, Stride=15 – Tests almost every other setting of the upper 6 bits, and all settings of lower 3 bits

On completion of the test, the error counts for each LCDL are written to the message block, so that the failing LCDL(s) may be located.

Fail condition(s):

- If any tested phase setting in any AC/Dbyte LCDL falls outside the allowed deviation range.

4.6.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the LCDL Linearity test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes, and set the bit to 0 if the test fails.

Field Name	Direction	Legal / Recommended Values	Comments
LcdlClksToRun	Input	<p>Range 1000-16383</p> <p>Suggested: $10 * DfiClkFreq$, where $DfiClkFreq$ is in MHz (For Example, if $DfiClkFreq=800\text{MHz}$, $LcdlClksToRun=8000$)</p> <p>For more details refer to section 6.2.6.</p>	<p>Clock cycles to run for each measurement. The higher the value, the more accurate the measurement. However, if the value is too large, the counters may reach 65,535 (refer to section 6.2.6).</p> <p>For simulation, recommend using a value of 1000 to shorten the simulation time and prevent saturation of counters when route delays are not simulated.</p>
LcdlStartPhase	Input	Recommended value of 0	Start Phase (0-511)
LcdlEndPhase	Input	Recommended value of 511	End Phase (0-511)
LcdlStride	Input	Stride 1 to 100, 7 recommended (0x01-0x64, 0x7 recommended) All other values reserved	Stride for taking measurements. Must be an odd number to ensure that all interpolated values are tested.
LcdlPassPercent	Input	<p>Recommended 100 = 0x64</p> <p>Allowed Variation = $\pm \left(\frac{5 * LcdlPassPercent}{100} \right) \%$</p>	<p>Percentage of spec values to consider pass/fail. 100 = 100% = matches spec. Numbers smaller than 100 tighten the tolerance. Numbers larger than 100 loosen the tolerance. Should be set to 100.</p>
LcdlObserveCfg[4]	Input	<p>Bit[15] - AC/Dbyte Select</p> <ul style="list-style-type: none"> ■ AC Select = 1 ■ Dbyte Select = 0 <p>Bits[14:4] - LCDL Index</p> <ul style="list-style-type: none"> ■ AC LCDL IDX = 0-10 ■ Dbyte LCDL IDX = 0-30 <p>Bits[3:0] - AC/Dbyte Instance</p> <ul style="list-style-type: none"> ■ AC = 0-(NumAc-1) Dbyte = 0-(NumDbyte-1) 	<p>The edge counts for up to 4 LCDLs are saved to the message block, for the purposes of generating linearity plots. Refer to section 4.6.4.</p>
LcdlErrCntAc[AcNum][AcLcdlNum]	Output	Any	Error count indexed by AC instance number, then by LCDL number. LCDL Number encoding is shown in section 4.6.3.

Field Name	Direction	Legal / Recommended Values	Comments
LcdlErrCntDb[DbyteNum][DbyteLcdlNum]	Output	Any	Error count indexed by Dbyte instance number, then by LCDL number. LCDL number encoding is shown in section 4.6.3.
LcdlCountValues[4][512]	Output	Any	Refer to sections 4.6.4 and 4.6.5.

4.6.2 Message Block Inputs - Illegal Argument Checking

The following checks are performed on the message block inputs:

$$LcdlStartPhase \leq 511$$

$$LcdlEndPhase \leq 511$$

$$LcdlStartPhase \leq LcdlEndPhase$$

$$LcdlStride > 0$$

If any condition is false, the test will be marked as fail, the LCDL linearity outputs will be filled with the code 0xA5, and the test will return immediately.

4.6.3 LCDL Linearity Instance Numbers

The following table outlines how LCDL numbers map to physical LCDLs in AC and Dbyte instances. This mapping is utilized for selecting LCDLs in the context of the input `LcdlObserveCfg`, and the outputs `LcdlErrCntAc` and `LcdlErrCntDb`.

Table 4-1 LCDL Linearity Instances Numbers (AC/Dbyte)

Value (decimal)	Dbyte LCDL	AC LCDL
0	Dbyte TxDq_r0	AC SE0
1	Dbyte TxDq_r1	AC SE1
2	Dbyte TxDq_r2	AC SE2
3	Dbyte TxDq_r3	AC SE3
4	Dbyte TxDq_r4	AC SE4
5	Dbyte TxDq_r5	AC SE5
6	Dbyte TxDq_r6	AC SE6
7	Dbyte TxDq_r7	AC SE7
8	Dbyte TxDq_r8	AC SEC0
9	Dbyte TxDqs	AC SEC1
10	Dbyte RxEn	AC DIFF
11	Dbyte RxRep	n/a
12	Dbyte RxClkT_r0	n/a
13	Dbyte RxClkT_r1	n/a
14	Dbyte RxClkT_r2	n/a
15	Dbyte RxClkT_r3	n/a
16	Dbyte RxClkT_r4	n/a

Value (decimal)	Dbyte LCDL	AC LCDL
17	Dbyte RxClkT_r5	n/a
18	Dbyte RxClkT_r6	n/a
19	Dbyte RxClkT_r7	n/a
20	Dbyte RxClkT_r8	n/a
21	Dbyte RxClkC_r0	n/a
22	Dbyte RxClkC_r1	n/a
23	Dbyte RxClkC_r2	n/a
24	Dbyte RxClkC_r3	n/a
25	Dbyte RxClkC_r4	n/a
26	Dbyte RxClkC_r5	n/a
27	Dbyte RxClkC_r6	n/a
28	Dbyte RxClkC_r7	n/a
29	Dbyte RxClkC_r8	n/a
30	Dbyte WCK (If LPDDR5_ENABLED)	n/a

4.6.4 LCDL Count Values as Message Block Outputs:

The input `LcdlObserveCfg[4]` allows the user to select 4 different LCDLs, whose edge counts will be stored as outputs in the results section of the message block. The encoding of the four 16-bit selects is shown below:

Field	Description
<code>LcdlObserveCfg[15]</code>	Qualifies whether the selected LCDL in an AC or Dbyte instance 0x1 = AC 0x0 = Dbyte
<code>LcdlObserveCfg[14:4]</code>	The instance number of the selected LCDL (see section 4.6.3)
<code>LcdlObserveCfg[3:0]</code>	The selected AC/Dbyte instance

4.6.5 How to Interpret the LCDL Count Value Output from the LCDL Linearity Test Results, and Plot the Results

The edge count data for the selected LCDLs are returned in the message block output `LcdlCountValues[4][512]`. The first index maps to the selection made in the message block input `LcdlObserveCfg[4]`.

The array is then indexed by phase, which ranges from 0 to 511. The first valid phase will be `LcdlStartPhase`. The next valid phases will be `LcdlStartPhase + LcdlStride` (and so on). `LcdlCountValues` will be populated for each valid phase.

- `LcdlCountValues[0][512]` – LCDL Ring Oscillator Edge Counts for LCDL Selection 0
- `LcdlCountValues[1][512]` – LCDL Ring Oscillator Edge Counts for LCDL Selection 1
- `LcdlCountValues[2][512]` – LCDL Ring Oscillator Edge Counts for LCDL Selection 2
- `LcdlCountValues[3][512]` – LCDL Ring Oscillator Edge Counts for LCDL Selection 3

The count values in these registers reflect the oscillator count for that phase, when the oscillator is run for the number of DFI clocks in `LcdlClksToRun`. These numbers do not need to be modified before plotting. However, to interpret the count values and see linearity, the count values need to be converted into delay values. This can be done using the following equations:

$$OscillatorTime = LcdlClksToRun * \left(\frac{1}{DfiClkFreq * 10^6} \right)$$

$$LcdlDelay[i][N] = \frac{1}{2} * \left(\frac{OscillatorTime}{LcdlCountValues[i][N]} \right)$$

The divide by 2 accounts for the fact that it takes 2 passes through the LCDL to create 1 clock to increment the oscillator counter. The `LcdlDelay` values can then be analyzed or plotted to gauge the LCDL functionality.

4.7 Address/Command Loopback

The PHY contains features to support loopback testing functionality of the address/command (AC) signals. Once placed in loopback mode, a PRBS7 pattern is generated to activate the hardware training state machines in the SE slices. To test the DIFF slices, a 1UI wide pulse looped back, and sampled using phase-detect logic. The SEC slices are tested in a similar manner to the DIFF slices, except the pulse width is the same as the DfiClk pulse width (either 2UI or 4UI).

AC loopback is a fully automated test provided in the ATE Firmware image. Signals can be looped back internally in the receiver or at the at the I/O pad. If I/O pad loopback is selected, it is important that these points are correctly externally terminated (refer to section 4.7.7)

AC lanes contain built-in loopback logic: dedicated pattern generator/checkers/samplers have been implemented.

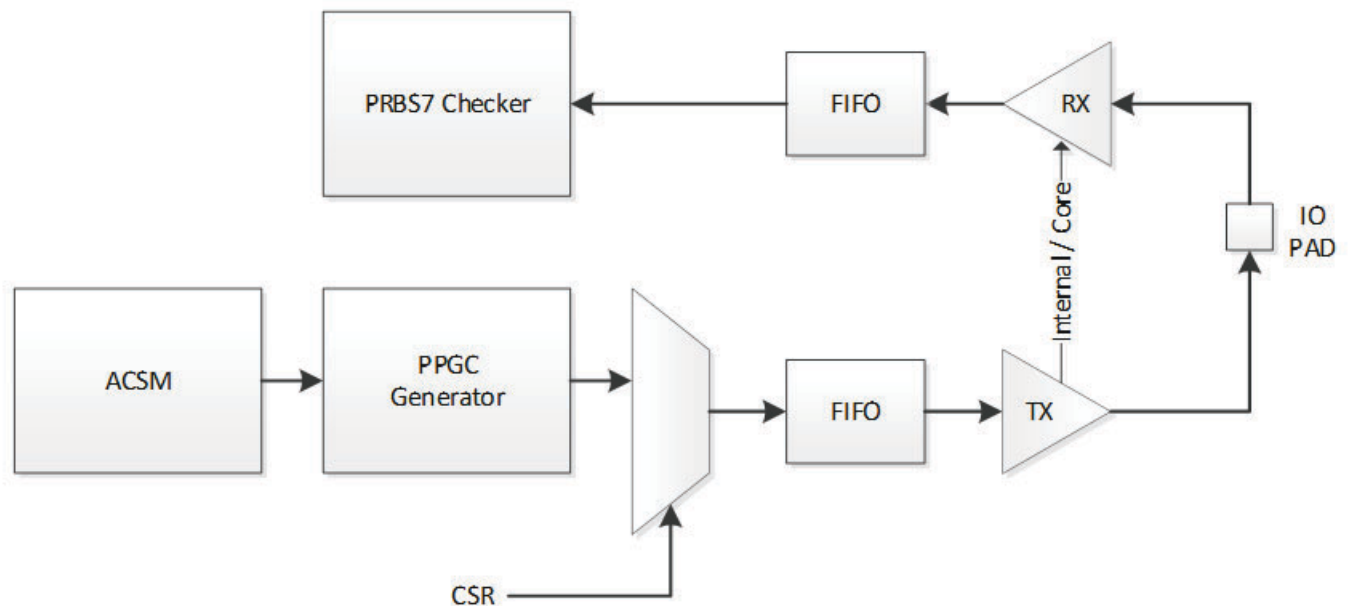
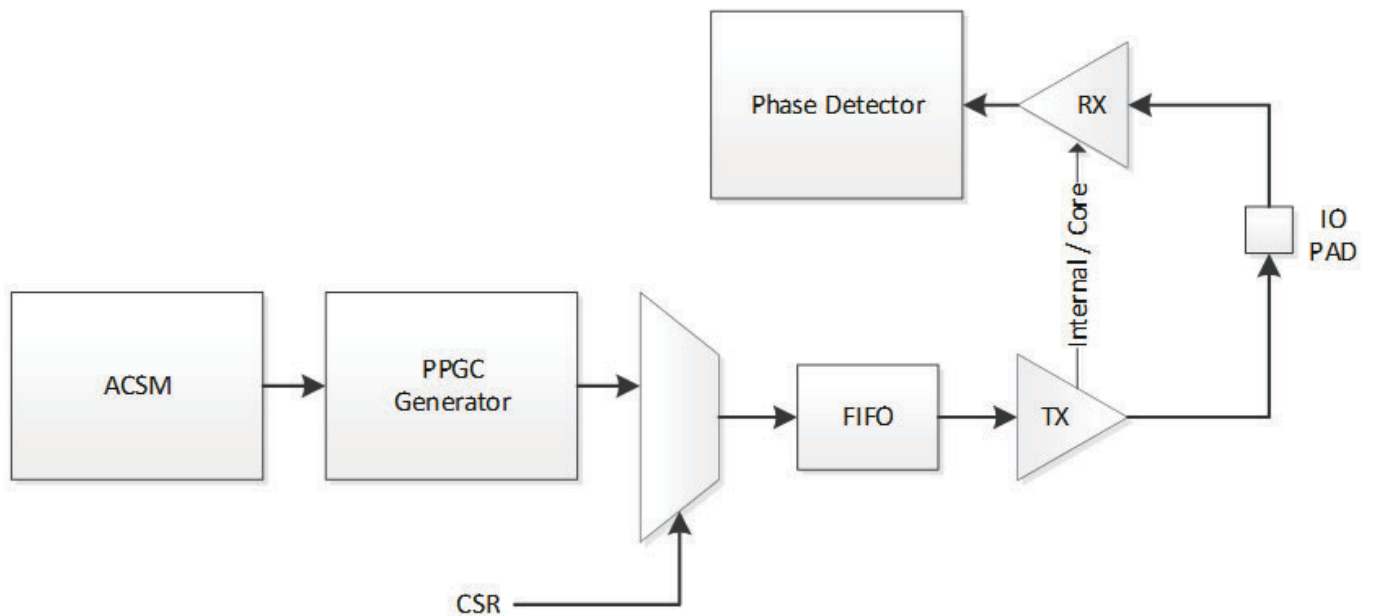
The test first makes sure impedance calibration and the PLL lock test have been run (refer to section 4.7.6). Then, the test proceeds to test the ACs.

A PRBS7 pattern (x^7+x^6+1) is transmitted through the SE slices for a user selected period, simultaneously checking the received data. Detected errors are logged and written to the message block on completion for each AC instance. The test then transmits a periodic pulse through the DIFF and SEC slices, using phase detect logic to sample the receive data.

Using the message block input AcLoopDiffTestMode the CK DIFF slice can also be tested in single-ended true mode.

Fail condition(s):

- One or more AC SE slices fail the minimum EYE width check
- One or more AC SE slices fail the stuck-at check (section)
- One or more AC DIFF slices fail the minimum EYE width check in differential mode, and (if enabled) single-ended mode
- One or more AC SEC slices fail the minimum EYE width check

Figure 4-1 AC SE Slices**Figure 4-2 AC SEC/DIFF Slices**

Note

The PRBS7 designation only indicates the equation used to generate the PRBS7 data stream. It does not imply that a specific length of the bitstream will be used during the test. The length of the PRBS pattern (before it repeats) is 127. The length of the bitstream used is controlled by the AcLoopClksToRun parameter. Small values can result in only part of the total possible pattern being used.

4.7.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Address/Command Loopback test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes, and set the bit to 0 if the test fails.
AcLoopClksToRun	Input	0-65535, recommended value 512. 1-65535: Wait the specified number of DFICLK cycles. 0: Run until user causes DctWriteProt to transition from 0x0 to 0x1.	AC SE slice PRBS transmit duration in DFICLKs. Note: Setting this field to 0 will cause the PRBS pattern to transmit until DctWriteProt CSR bit transitions from 0x0 to 0x1. Refer to section 4.7.3. Note: A value of 0 is only supported if AcLoopMinLoopPwr is 0x0.
AcLoopCoreLoopBk	Input	Pad / Core loopback 0x0 = Pad loopback 0x1 = Core loopback	Pad loopback requires proper termination (section 4.7.7).
AcLoopMinLoopPwr	Input	0x0 = All ACs tested in parallel 0x1 = Only 1 AC active at a time	Parallel testing will run faster, but results in higher power consumption. Serial testing will consume less power, but takes longer to run. Note: A value of 0x1 is only supported if AcLoopClksToRun is non-zero.
AcLoopLaneMask[2]	Input	Bit encoding: [0..7] – SE0..SE7 [8] – DIFF [9..10] – SEC0, SEC1 [11..15] - RESERVED	Bit vector array, indexed first by AC instance, then by selected slice. Setting a bit to 0x1 disables the checking of the corresponding slice in the selected AC instance.
AcMinEyeWidthSe	Input	Recommended 0x60 for core loopback and 0x40 for pad loopback. Must be divided by AcLoopIncrement if AcLoopIncrement is larger than 0x1.	The number of consecutive tested delays which must pass for AC SE slice testing to report pass. The chosen value should be greater than zero. If (0) is programmed, the SE slice EYE width checks will always pass.

Field Name	Direction	Legal / Recommended Values	Comments
AcMinEyeWidthDiff	Input	Recommended 0x30 for core loopback and 0x20 for pad loopback. Must be divided by AcLoopIncrement if AcLoopIncrement is larger than 0x1.	The number of consecutive tested delays which must pass for AC DIFF slice testing to report pass. The chosen value should be greater than zero. If (0) is programmed, the DIFF slice EYE width check will always pass.
AcMinEyeWidthSec	Input	Note: Refer to section 4.1.3 for when the ACs are in 1:2 or 1:4 mode (referred to here as "AcDfiClkFreqRatio"). "AcDfiClkFreqRatio"=1:2 Recommended 0x60 for core loopback and 0x40 for pad loopback. "AcDfiClkFreqRatio"=1:4 Recommended 0xC0 for core loopback and 0x80 for pad loopback. Must be divided by AcLoopIncrement if AcLoopIncrement is larger than 0x1.	The number of consecutive tested delays which must pass for AC SEC slice testing to report pass. The width of the pulse (in UI) depends on "AcDfiClkFreqRatio." Refer to section 4.1.3 for when the ACs are in 1:2 or 1:4 mode. The chosen value should be greater than zero. If (0) is programmed, the SEC slice EYE width check will always pass.
AcLoopDiffTestMode	Input	Specific testing enabled by setting corresponding bit to 1. Bit[0] = CK_T	Bit vector used for configuring CK DIFF slice testing. Note: only used in pad-loopback mode.
AcLoopDiffBitmapSel	Input	Bits[3:0] = CK Bitmap Select 0x0 = CK differential 0x1 = CK_T	Select which CK bitmap to save to AcLoopbackBitmapDiff. Note: If a particular CK bitmap is chosen to be saved, but the corresponding testing hasn't been enabled using AcLoopDiffTestMode, the resulting bitmap will contain all 1's.
AcLoopIncrement	Input	1-255 Recommended value 1	When constructing the EYEs, this controls the fine delay increment. This is used for testing all slice types. When using an AcLoopIncrement greater than 0x1, the AcLoopMinEyeWidth's must be scaled appropriately.

Field Name	Direction	Legal / Recommended Values	Comments
AcVrefDac	Input	Refer to description of CSRs 'AcVrefDAC*' for field description	
AcTxImpedanceSe	Input	Refer to description of CSRs 'TxImpedanceSE*' for field breakdown	
AcTxImpedanceDiff	Input	Refer to description of CSRs 'TxImpedanceDIFF*' for field breakdown	
AcTxImpedanceSec	Input	Refer to description of CSRs 'TxImpedanceCMOS*' for field breakdown	
AcTxSlewSe	Input	Refer to description of CSRs 'TxSlewSE*' for field description	
AcTxSlewDiff	Input	Refer to description of CSRs 'TxSlewDIFF*' for field description	
AcLoopbackStuckAtSe[A cNum]	Output	Refer to section 4.7.5.1.	Stuck-at results for SE slices in each AC. Each bit indicates whether that lane failed stuck-at testing. For example, if bit[4] is set to 1, it means that SE slice 4 failed stuck-at testing.
AcLoopbackBitmapSe[A cNum][SeNum][0/1]	Output	Refer to section 4.7.5.2.	A 2UI wide EYE is constructed for each SE slice in each AC instance. A (0) in a bit position indicates loopback operates correctly at that delay setting, and (1) indicates failure at that setting.
AcLoopbackBitmapSec[A cNum][SecNum][0-4]	Output	Refer to section 4.7.5.4.	Up to 5UI worth of EYE data is saved for each SEC slice in each AC instance. A (0) in a bit position indicates loopback passed at the tested delay, and (1) indicates loopback failed at the tested delay. The value of AcLoopbackNumUiSec indicates the number of UIs required to be capture the entire SEC slice EYE.
AcLoopbackNumUiSec[A cNum][SecNum]	Output	Refer to section 4.7.5.4.	The number of UIs worth of checking data that were required to capture the entire SEC slice EYE.

Field Name	Direction	Legal / Recommended Values	Comments
AcLoopbackBitmapDiff[AcNum][DiffNum][0/1/2]	Output	Refer to section 4.7.5.3.	Up to 3UI worth of EYE data is saved for each DIFF slice in each AC instance. A (0) in a bit position indicates loopback operates correctly at that setting, and (1) indicates failure at that setting. AcLoopbackNumUiDiff indicates the number of UIs required to capture the entire DIFF slice EYE.
AcLoopbackNumUiDiff[AcNum][DiffNum]	Output	Refer to section 4.7.5.3.	The number of UIs worth of checking data that were required to capture the entire DIFF slice EYE.
AcLoopbackCoarseSec[AcNum][SecNum]	Output	Any	AC SEC slice ac loopback EYE starting coarse value. Indexed by AcNum and SecSliceNum. Refer to section 4.7.5.5 for how to interpret the data.
AcLoopbackCoarseDiff[AcNum][DiffNum]	Output	Any	AC Diff slice ac loopback EYE starting coarse value. Indexed by AcNum and DiffSliceNum. Refer to section 4.7.5.6 for how to interpret the data.

4.7.2 Message Block Inputs - Illegal Argument Checking

The following check is performed on the message block inputs:

$$AcLoopIncrement > 0$$

If the condition is false, the test will be marked as fail, the AC loopback outputs will be filled with the code 0xA5, and the test will return immediately.

4.7.3 AC Loopback Infinite-Traffic Mode

AC Loopback supports an “infinite-traffic” mode, which is enabled by setting AcLoopClksToRun=0x0. During infinite-traffic mode, traffic is continually transmitted on the AC SE slices, at the current delay setting, until the user causes DctWriteProt to transition from 0x0 to 0x1. The test will then proceed to the next delay to test, based on the chosen values for AcLoopIncrement, until all delays have been tested.

In total, the following number of transitions of DctWriteProt from 0x0 to 0x1 transitions that are required is:

$$NumTransitions = \left\lceil \text{roundDown} \left(\frac{127}{AcLoopIncrement} \right) + 2 \right\rceil$$

The trivial case occurs when AcLoopIncrement >=128, as only two transitions are required. While an AcLoopIncrement this large is not recommended if AcLoopClksToRun is non-zero, it is appropriate in this case.

If the goal is the characterize the IO drivers, it is recommended to use data loopback 1D/2D in infinite-mode (sections 4.8. 3 and 4.9.3), as they support a mode where all AC and Dbyte slices are active at the same time.

4.7.4 CK DIFF Slice Single-Ended Testing

In pad-loopback mode (AcLoopCoreLoopBk=0x0), additional CK DIFF slice checking can be enabled.

By default, in pad-loopback mode:

- The CK is tested in differential mode by looping back periodic a 1UI wide pulse.

Using the message block input AcLoopDiffTestMode, the CK DIFF slices can also be tested in single-ended true (CK_T) mode.

When in single-ended mode, the message block input AcVrefDac is used to configure the VREF for the CK_C input to the differential receiver.

In core-loopback mode, single-ended testing is disabled because the loopback point is before the single-ended specific circuitry.

4.7.5 How to Interpret the Output From the Address/Command Loopback Test Results

4.7.5.1 AcLoopbackStuckatSe

This portion of the test is designed to detect stuck-at-0 errors. A stuck-at-0 fault will result in the self-seeding PRBS7 checker circuit to be seeded with zeros, and, as a result, the error count will be zero.

Each bit indicating whether the SE slice failed the stuck-at portion of the test. For example, if bit 4 is set to 1, it means that SE slice 4 failed stuck-at testing.

Stuck-at-1 type errors will not be caught in this portion of the test: instead, they will be caught in the PRBS7 loopback portion of the test.

4.7.5.2 AcLoopbackBitmapSe

The output from the AC loopback test is a set of bitmaps for all the lanes in all AC instances. These bitmaps are used to determine the passing and failing regions for each lane. For the SE slices, a 2UI window (128 fine steps) is sampled, and the EYE information is stored in two 64bit array entries. The entire EYE can be assembled concatenating all 128bits together.

AcLoopbackBitmap[curAc][curSe][0]							AcLoopbackBitmap[curAc][curSe][1]						
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]
Tested Delay 0	Tested Delay 1	Tested Delay 2	Tested Delay 3	...	Tested Delay 62	Tested Delay 63	Tested Delay 64	Tested Delay 65	Tested Delay 66	Tested Delay 67	...	Tested Delay 126	Tested Delay 127

The number of tested delays within the 2UI EYE depends on the value of AcLoopIncrement, as described by the following formula:

$$NumTestedDelays = \text{roundUp}\left(\frac{128}{AcLoopIncrement}\right)$$

The first NumTestedDelays bits will form the 2UI wide EYE test results. The remaining entries will be set to (1).

If bit[N] of the assembled EYE equals (0), it means that the loopback test passed at tested delay setting N. If bit[N] equals (1), either that the loopback test failed at tested delay N, or that the bit is unused, based on the value of AcLoopIncrement.

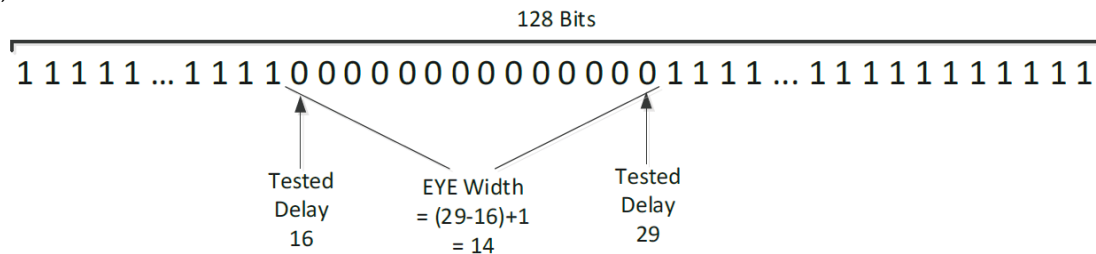
Figure 4-3 Example #1: AcLoopIncrement = 1

AcLoopbackBitmap[curAc][curSe][0]								AcLoopbackBitmap[curAc][curSe][1]							
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]		Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	
Pass/Fail Delay 0x00	Pass/Fail Delay 0x01	Pass/Fail Delay 0x02	Pass/Fail Delay 0x03	...	Pass/Fail Delay 0x3E	Pass/Fail Delay 0x3F		Pass/Fail Delay 0x40	Pass/Fail Delay 0x41	Pass/Fail Delay 0x42	Pass/Fail Delay 0x43	...	Pass/Fail Delay 0x7E	Pass/Fail Delay 0x7F	

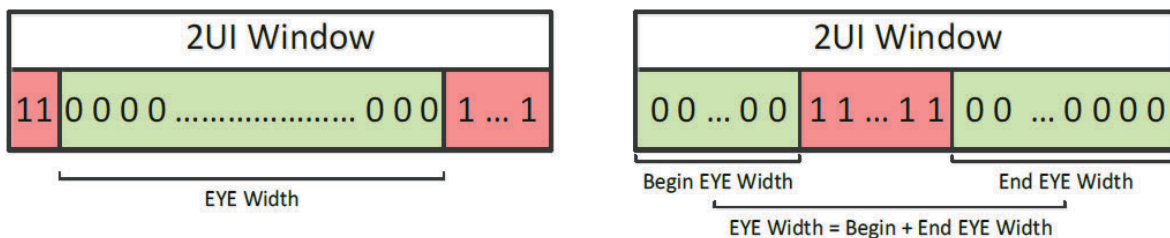
Figure 4-4 Example #2: If AcLoopIncrement = 2, the first 64bits will contain the testing information for the 2UI window, and the last 64bits will be set to all 1's.

AcLoopbackBitmap[curAc][curSe][0]								AcLoopbackBitmap[curAc][curSe][1]							
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]		Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	
Pass/Fail Delay 0x00	Pass/Fail Delay 0x02	Pass/Fail Delay 0x04	Pass/Fail Delay 0x06	...	Pass/Fail Delay 0x7D	Pass/Fail Delay 0x7F		1	1	1	1	...	1	1	

The passing region for an EYE is the longest run of contiguous zeros. For the example bitmap below, the passing region is from tested delay 16 to tested delay 29. Therefore, the EYE width is taken to be 14. If AcMinEyeWidthSe is set to 14 or lower, this SE slice would report pass. If the AcMinEyeWidthSe is set to 15 or higher, this SE slice would fail.



Note that, due to the nature of the AC SE slice PRBS checkers, the passing region in the generate bitmap can appear to wrap around from the max tested delay to the min tested delay. When this occurs, the width of the EYE is the sum of the two passing regions.

**Note**

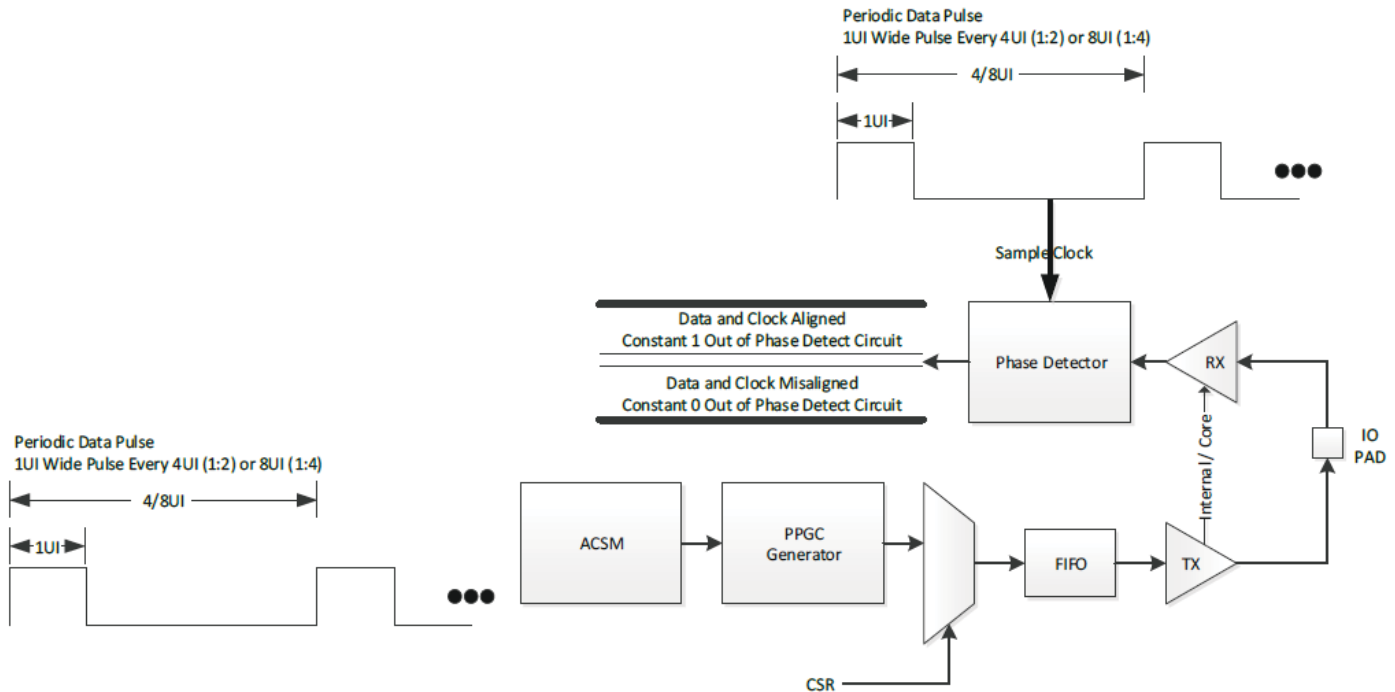
Because of the nature of the SE slice circuitry used to perform the testing, the passing region wraps from the end of the bitmap to the beginning of the bitmap. A passing region with 7 passing at the end of the bitmap and 6 passing at the beginning of the bitmap with all other values failing would also have a width of 13. [This does not apply to the data loopback test results.]

4.7.5.3 AcLoopbackBitmapDiff

The DIFF slices are tested by transmitting a periodic, 1UI wide pulse whose duty cycle is either 25% (1:2 mode) or 12.5% (1:4 mode). The periodic pulse has the delay swept prior to transmission. The signal is

sampled in a phase detect circuit, using a clock with the same duty cycle as the transmitted pulse. Whenever the received pulse and the clock are aligned, the output of the phase detector will be a constant (1) (EYE is open). If the clock and data are misaligned, the output is (0) (EYE is closed).

It is important to note that the output polarity of phase detector is the opposite of the bitmap polarity. If a (0) is reported by the phase detector, the corresponding bitmap position will be set to (1). If a (1) is reported by the phase detector, the corresponding bitmap position will be set to (0).



There is inherent metastability in the phase-detect hardware. Therefore, in real silicon one will likely observe fuzzy EYEs, whose width is greater than 1UI. However, the important observation is whether an EYE exists, and that it fulfills a given minimum EYE width criteria, as controlled by the message block input `AcMinEyeWidthDiff`.

Because the measured EYE width can be greater than 1UI, the output bitmap has enough capacity for 3UI worth of sample data, to ensure the entire EYE is captured. The output `AcLoopbackBitmapDiff` is used to store the UIs (at most 3) where the EYE is observed to be open. The message block output `AcLoopbackNumUiDiff` indicates the number of UIs that were required to capture the EYE. The following formula can be used to determine how many bits in the bitmap are required to assemble the EYE.

$$NumUsedBits = roundUp\left(\frac{64 * AcLoopbackNumUiDiff}{AcLoopIncrement}\right)$$

For example, if `AcLoopIncrement` is 1, and `AcLoopbackNumUiDiff` is 2, then the first 128 bits concatenated together for the EYE. All unused bitmap locations will be 1 at the end of the test.

The following diagrams illustrate how the swept delay range is captured into the output message block. In this example, we're considering the AC DIFF slice bitmap, with an AC frequency ratio of 1:4 (8UI of delay is swept). Also, a fine increment of (1) is assumed (a larger fine increment will result in the EYE being compacted, based on the description above).

In general, two considerations affect the stored EYE bitmap: are 2UI or 3UI required to capture the entirety of the passing region, and does the EYE wrap around from the max delay to the minimum delay? These two considerations result in 4 general forms for the final stored bitmap.



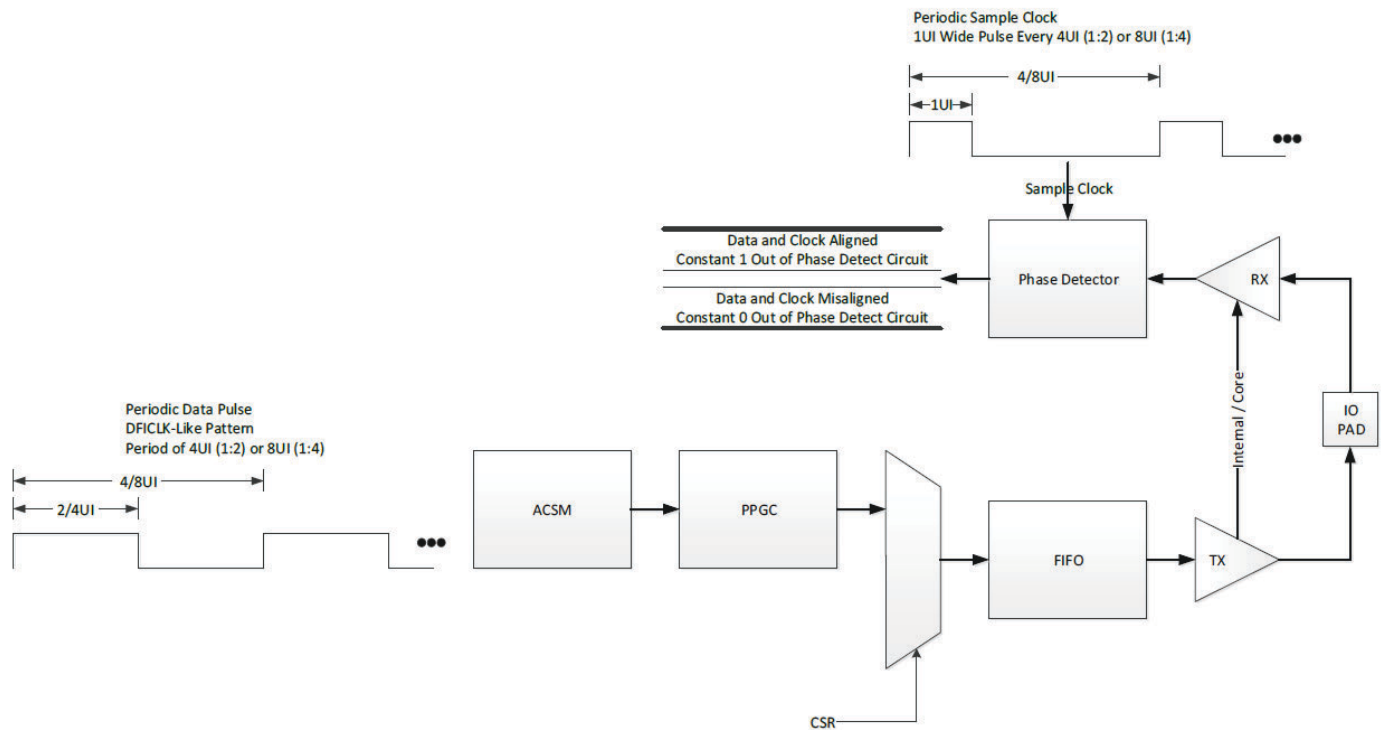
Like the AC SE slices (section 4.7.5.2), the EYE width is the size of the largest contiguous passing region. If the passing region wraps around, the EYE width is the sum of the width of the passing regions at the minimum and maximum delays.

These general EYE capture principles also apply to 1:2 mode (4UI delay range is swept), and the AC SEC slices (except, up to 5UI of passing region is stored instead of 3UI).

4.7.5.4 AcLoopbackBitmapSec

The SEC slice is tested by sending a periodic 50% duty cycle pulse, whose period is the same as the DFICLK (making it a “DFICLK-like” pattern). When the AC(s) are in 1:2 mode the pulses are 2UI wide, and the period of the pattern is 4UI. When the AC(s) are in 1:4 mode, the pulses are 4UI wide, and the period of the

pulses is 8UI wide. The signal is sampled using similar approach as the DIFF slices (section 4.7.5.3). Refer to section for how to determine when the AC(s) are in 1:2 mode, and when it is in 1:4 mode.



Again, because of metastability in the phase detect sampler, measured EYE widths can be greater than 2/4UI. Hence, the SEC slice bitmaps contain enough capacity for 5UI worth of sample data. The number of UI required to capture the entire EYE are stored in the message block output `AcLoopbackNumUiSec`. The following formula can be used to determine how many bits in the bitmap are required to assemble the EYE.

$$NumUsedBits = roundUp\left(\frac{64 * AcLoopbackNumUiSec}{AcLoopIncrement}\right)$$

For example, if `AcLoopIncrement` is 1, and `AcLoopbackNumUiSec` is 3, then the first 192 bits concatenated together for the EYE. All unused bitmap locations will be 1 at the end of the test.

The methodology for storing the SEC slice bitmap to the message block is analogous to the methodology used by for the AC DIFF slices (section 4.7.5.3).



Unlike the DIFF slice testing, the width of the generated pattern varies between 1:2 (2UI) and 1:4 mode (4UI), depending on the frequency ratio of the ACs ("AcDfiClkFreqRatio"). The value of "AcDfiClkFreqRatio," and consequently `AcMinEyeWidthSec`, depends the value of `ClockingMode` and `DfiClkFreqRatio` (refer to section 4.1.3).

**Note**

Because of the nature of the DIFF/SEC slice phase-detect phase detect circuitry used to perform the test, the passing region can wrap from the end of the bitmap to the beginning of the bitmap. A passing region with 10 passing at the end of the bitmap and 12 passing at the beginning of the bitmap with all other values failing would also have a width of 22.

[This does not apply to the data loopback test results.]

4.7.5.5 AcLoopbackCoarseSec

These values are the coarse delay bits for the corresponding AcLoopbackBitmapSec value. Meaning that the value of AcLoopbackCoarseSec indicates the coarse value for where the 5UI bitmap starts. The starting coarse values can be different per Ac and should be used to normalize the data when comparing lanes.

For example, if AcLoopbackCoarseSec [0] is 4 and AcLoopbackCoarseSec [1] is 5 then the bitmap for AcLoopbackBitmapSec [0] starts in UI 4 and the AcLoopbackBitmapSec [1] starts in UI 5. To plot the data for both bits, the plot should contain data for UI 4 to UI 6. AcLoopbackBitmapSec [1] should be padded to contain all failing values for UI 4 and AcLoopbackBitmapSec [0] should be padded to contain all failing values for UI 6.

4.7.5.6 AcLoopbackCoarseDiff

These values are the coarse delay bits for the corresponding AcLoopbackBitmapDiff value. Meaning that the value of AcLoopbackCoarseDiff indicates the coarse value for where the 3UI bitmap starts. The starting coarse values can be different per Ac and should be used to normalize the data when comparing lanes.

For example, if AcLoopbackCoarseDiff [0] is 4 and AcLoopbackCoarseDiff [1] is 5 then the bitmap for AcLoopbackBitmapDiff [0] starts in UI 4 and the AcLoopbackBitmapDiff [1] starts in UI 5. To plot the data for both bits, the plot should contain data for UI 4 to UI 6. AcLoopbackBitmapDiff [1] should be padded to contain all failing values for UI 4 and AcLoopbackBitmapDiff [0] should be padded to contain all failing values for UI 6.

4.7.6 Pre-test Requirements

It is required that the Impedance Calibration and PLL/LCDL Lock test be run prior to running the AC loopback test. Because of this requirement, the firmware will automatically enable these tests whenever the AC loopback test is enabled. The PassFailResults message block field will include the results for these tests, even if they are not specified in the TestsToRun field.

**Note**

Because the PLL Lock test is required for the Address/Command Loopback test, this test must be run with a DfIClk within the legal range of the PLL to lock.

4.7.7 Termination on Address / Command Pins for Pad Loopback

Address / Command SE and DIFF slice pins must be terminated to VSS for pad loopback testing. Termination strength of 50 Ohms is suggested. The pads do not need to be terminated for core loopback.

The Vref needs to be set appropriately for the termination used. The Vref for the AC loopback test is controlled by the AcVrefDac message block parameter.

**Note**

The SEC slice pins should not be terminated.

4.8 Data Loopback 1D

The PHY contains features to support loopback testing functionality. Once placed in loopback mode, stimulus patterns are generated by the firmware image to activate the hardware training state machines. Data byte DQ/DM and DQS/WCK lanes are looped back to their respective receivers and utilize the training hardware to perform the test.

The data loopback is a fully automated test. Signals can be looped back internally in the receiver or at the I/O pad. If I/O pad loopback is selected, it is important that these points are correctly externally terminated (section 4.8.8).

If not previously initialized, a PHY impedance calibration is initiated, followed by PLL configuration and locking sequence. Once completed, if the loopback is performed at the I/O pad, a receiver enable calibration is performed. This aligns each of the receivers enable signal with the center of the received DQS preamble. This allows the test to automatically compensate for different technologies and packages.

The loopback test can now proceed. The PPGC is configured to generate a PRBS16 test pattern ($x^{16} + x^{15} + x^{13} + x^4 + 1$), which will be used to test the SE slices. The ACSM then generates write data bus transactions. Each lane's DTSM then compares the looped back data at each receiver, to the check data pattern also supplied by the PPGC. As part of this process the delays are varied to allow measurement of the size of the EYE for the loopback. The widest passing regions is then compared against the `DatLoopMinEyeWidth` for each lane to determine pass or fail for the test.

If the PHY is LP5 enabled, the Dbyte WCK DIFF slice is tested by transmitting a periodic 1UI wide pulse. The pulse is sampled using phase-detect logic in the WCK DIFF slice receiver. By sweeping the transmit delay, an EYE can be constructed. The passing region is compared against `DatLoopMinEyeWidth` for each instance to determine pass or fail for the test.

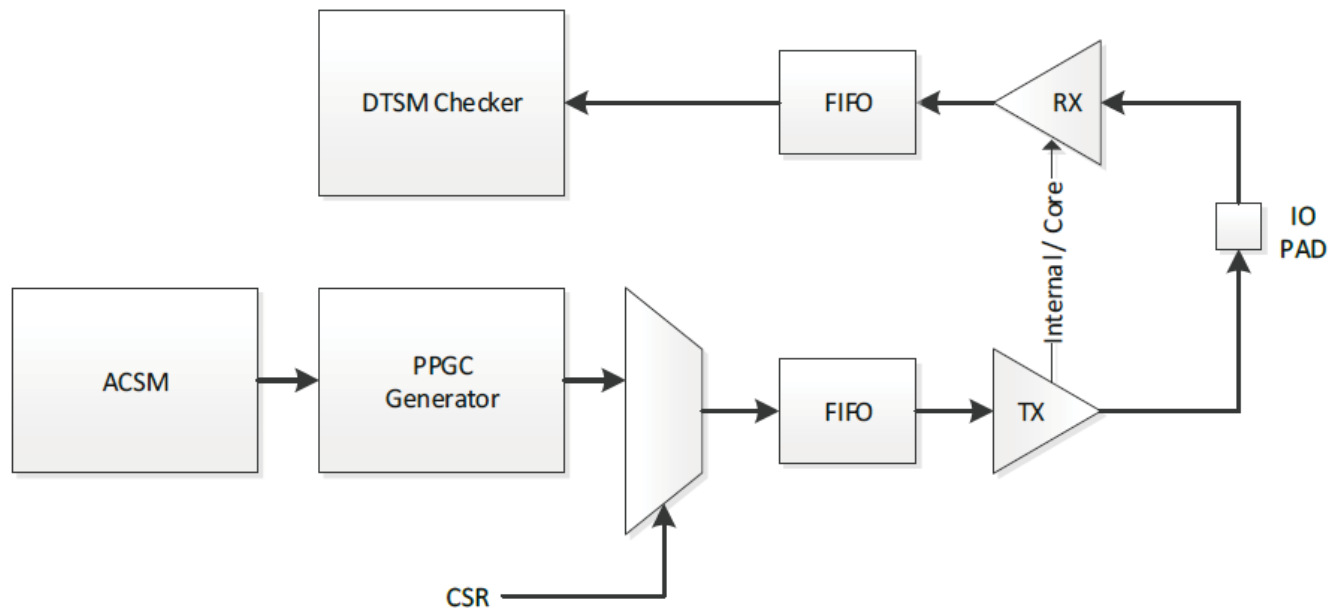
In core-loopback mode, the DQS DIFF slice is tested in the same way as the WCK DIFF slice. In pad-loopback mode, the looped-back DQS is used to clock in the DQ data, and hence dedicated loopback testing is disabled by default. However, in pad-loopback mode, the message block input `DatLoopDiffTestMode` provides the user with the option to run dedicated DQS slice testing.

Additionally, in pad-loopback mode, the WCK and DQS DIFF slices can be tested in single-ended mode (true or complement). The message block input `DatLoopDiffTestMode` is used to optionally enable WCK and DQS single-ended testing modes.

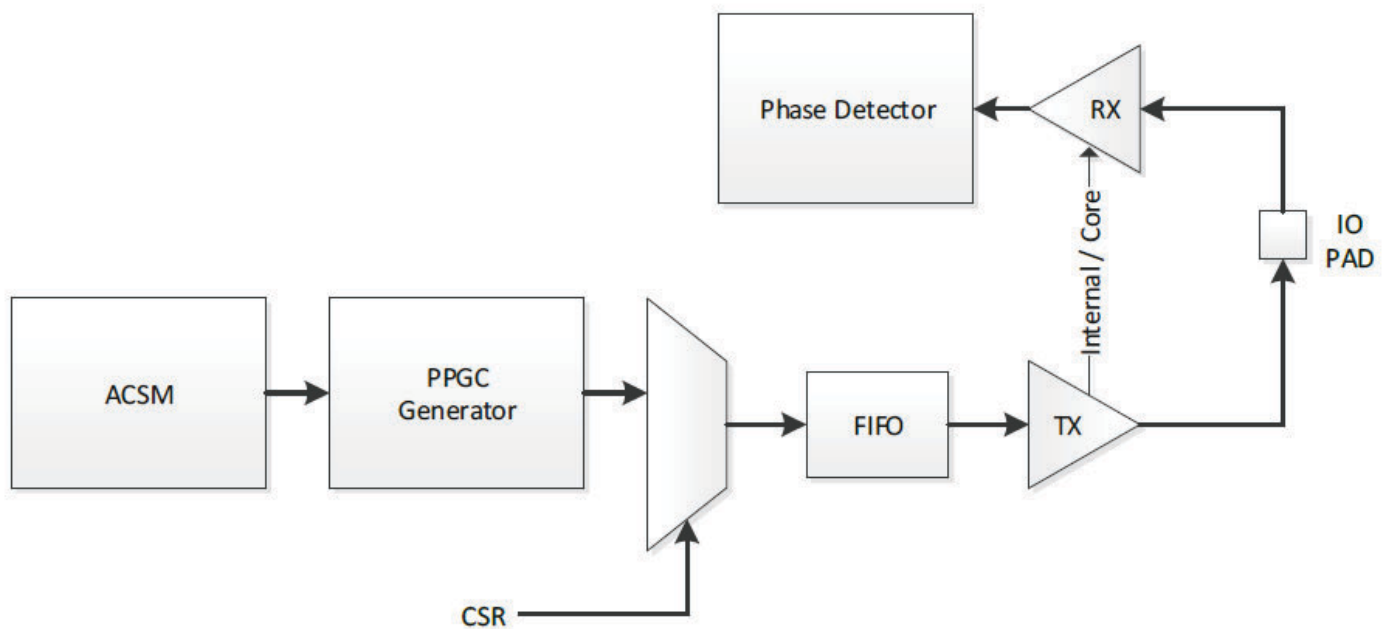
Fail condition(s):

- One or more Dbyte SE slices fail the `MinEyeWidth` check
- One or more Dbyte WCK DIFF slices fail the `MinEyeWidth` check in differential mode, and all enabled single-ended modes in pad-loopback mode
- One or more Dbyte DQS DIFF slices fail the `MinEyeWidth` check in core-loopback mode, or all enabled differential and single-ended test modes in pad-loopback mode.

DBYTE SE Slices



DBYTE DIFF Slices



**Note**

The PRBS16 designation only indicates the equation used to generate the PRBS16 data stream. It does not imply that a specific length of the bitstream will be used during the test. The length of the PRBS sequence (before it repeats) is 65,535. The length of the bitstream used is controlled by the DatLoopClksToRun parameter. Small values can result in only part of the total possible pattern being used.

4.8.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Data Loopback 1D test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes and set the bit to 0 if the test fails.
DatLoopClksToRun	Input	Range: 0,100-65535, Recommended: 512 100-65535: Wait the specified number of DFCLK cycles. 0: Run until user causes DctWriteProt to transition from 0x0 to 0x1.	Dbyte SE slice PRBS transmit duration in DFCLKs. Note: Setting this field to 0 will cause the PRBS pattern to transmit until DctWriteProt CSR bit transitions from 0x0 to 0x1. Refer to section 4.8.3. Note: A value of 0 is only supported if DatLoopMinLoopPwr is also 0x0.
DatLoopCoreLoopBk	Input	Pad / Core loopback select 0x0 = Pad loopback 0x1 = Core loopback	In pad-loopback mode, the looped-back DQS strobe is used to clock in the looped-back DQ data. In core-loopback mode, the internally generated strobe is used to clock in the looped-back DQ data (section 4.8.4). Pad loopback requires proper termination (section 4.8.8).
DatLoopMinLoopPwr	Input	0x0 = All Dbytes tested in parallel 0x1 = Only 1 Dbyte active at a time	Parallel testing will run faster, but results in higher power consumption. Serial testing will consume less power, but takes longer to run. Note: A value of 0x1 is only supported if DatLoopClksToRun is non-zero.
TxDqsDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
TxWckDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxEnDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.

Field Name	Direction	Legal / Recommended Values	Comments
RxDigStrbDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxCkT2UIDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxCkC2UIDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
DatLoopCoarseStart	Input	Range: 0-7 Recommended to set this to 0.	This is the start of the coarse range. The test will start at the beginning of this coarse value.
DatLoopCoarseEnd	Input	Range: 0-7 Recommended to set this to 7.	This is the end of the coarse range. The test will stop once the delay exceeds Coarse= DatLoopCoarseEnd, Fine = 0.
DatLoopFineIncr	Input	Range: 1-7 Recommended to set this to 1.	This defines the value which the delay is incremented for each measurement (in terms of fine 1/64UI steps). Setting this to a value greater than 1 will reduce runtime, at the expense of measurement resolution. When using a DatLoopFineIncr greater than 0x1, the value of DatLoopMinEyeWidth must be scaled appropriately.
DatLoopMinEyeWidth	Input	Recommended 0x30 for core loopback and 0x20 for pad loopback. Must be divided by DatLoopFineIncr if DatLoopFineIncr is larger than 0x1.	The number of consecutive tested delays which must pass for Dbyte SE and DIFF slice testing to report pass. The chosen value should be greater than zero. If (0) is programmed the EYE width checks will always pass.
DatLoopDiffTestMode	Input	Specific testing enabled by setting corresponding bit to 1. Bit[0] - DQS differential Bit[1] - DQS_T single-ended Bit[2] - DQS_C single-ended Bit[3] - WCK_T single-ended Bit[4] - WCK_C single-ended	Bit vector used for configuring DQS + WCK DIFF slice testing. Note: In core-loopback mode, this field is ignored, and DQS differential mode is enabled by default (refer to sections 4.8.3 and 4.8.5).

Field Name	Direction	Legal / Recommended Values	Comments
DatLoopDiffBitmapSel	Input	Bits[3:0] = DQS Bitmap Select 0x0 = DQS differential bitmap 0x1 = DQS_T bitmap 0x2 = DQS_C bitmap Bits[7:4] = WCK Bitmap Select 0x0 = WCK differential bitmap 0x1 = WCK_T bitmap 0x2 = WCK_C bitmap	Select which DQS+WCK bitmaps to save to DatLoopbackDqsBitmap and DatLoopbackWckBitmap. Note: If a particular DQS/WCK bitmap is chosen to be saved, but the corresponding testing hasn't been enabled using DatLoopDiffTestMode, the resulting bitmap will contain all 1's.
DbVrefDac	Input	Refer to description of CSRs 'VrefDAC*' for field description	
DbRxVrefCtl	Input	Refer to description of CSR 'RxVrefCtl' for field description	
DbRxDfeModeCfg	Input	Refer to description of CSR 'RxDfeModeCfg' for field description	
DbTxImpedanceSe	Input	Refer to description of CSRs 'TxImpedanceSE*' for field breakdown	
DbTxImpedanceDiff	Input	Refer to description of CSRs 'TxImpedanceDIFF*' for field breakdown	
DbTxSlewSe	Input	Refer to description of CSRs 'TxSlewSE*' for field description	
DbTxSlewDiff	Input	Refer to description of CSRs 'TxSlewDIFF*' for field description	
DatLoopbackRxEnbVal[DbyteNum]	Output	Any	Data loopback RxEnb delay value, indexed by Dbyte. In pad-side loopback, this will store the trained RxEn delays. In core-side loopback this will contain all 1's (section 4.8.4).
DatLoopbackCoarse[DbyteNum][SeNum]	Output	Any	SE slice data loopback EYE starting coarse value. Indexed by Dbyte and SE slice. Refer to section for how to interpret the data.
DatLoopbackBitmap[DbyteNum][SeNum][0/1]	Output	Any	SE slice data loopback eye bitmap indexed by Dbyte, SE slice, and UI0/1. Refer to section 4.8.6.1 for how to interpret the data.

Field Name	Direction	Legal / Recommended Values	Comments
DatLoopbackWckBitmap [DbyteNum][0/1/2]	Output	Any	Up to 3UI worth of EYE data is saved for the WCK DIFF slice in each Dbyte instance (if it exists). A (0) in a bit position indicates loopback operates correctly at that setting, and (1) indicates failure at that setting. See section 4.8.6.3 .
DatLoopbackNumUiWck [DbyteNum]	Output	Any	The number of UIs worth of data that were required to capture the entire WCK EYE (section 4.8.6.3).
DatLoopbackDqsBitmap [DbyteNum][0/1]	Output	Any	When running in core-loopback mode, or if enabled in pad-loopback mode, a 2UI wide EYE for each DQS DIFF slice is constructed. Refer to section 4.8.4 for more details on when the DQS bitmap is generated, and section 4.8.6.4 for how to interpret the outputted DQS bitmaps.
DatLoopbackWckCoarse [DbyteNum]	Output	Any	WCK Diff slice data loopback EYE starting coarse value. Indexed by Dbyte. Refer to section 4.8.6.5 for how to interpret the data.
DatLoopbackDqsCoarse [DbyteNum]	Output	Any	DQS Diff slice data loopback EYE starting coarse value. Indexed by Dbyte. Refer to section 4.8.6.6 for how to interpret the data.

4.8.2 Message Block Inputs - Illegal Argument Checking

The following checks are performed on the message block inputs

$$DatLoopFineIncr > 0$$

$$DatLoopCoarseStart \leq DatLoopCoarseEnd$$

$$DatLoopCoarseStart \leq 7$$

$$DatLoopCoarseEnd \leq 7$$

If any condition is false, the test will be marked as fail, the outputs associated with data loopback 1D will be filled with the code 0xA5, and the test will return immediately.

4.8.3 Data Loopback 1D Infinite-Traffic Mode

Data Loopback 1D supports an “infinite-traffic” mode, which is enabled by setting DatLoopClksToRun to be 0. In this mode, traffic is continually transmitted on the DQ SE slices, at the current delay setting, until the user causes DctWriteProt to transition from 0x0 to 0x1. The test will then proceed to the next delay to test, until all delays have been tested.

In total, the number of required DctWriteProt transition from 0x0 to 0x1 is:

$$NumTransitions = \left\lceil \text{roundDown} \left(\frac{64 * (\text{DatLoopCoarseEnd} - \text{DatLoopCoarseStart})}{\text{DatLoopFineIncr}} \right) + 1 \right\rceil$$

The trivial case occurs when DatLoopCoarseEnd equals DatLoopCoarseStart, as only one transition is required.

This mode can be used to characterize the drivers. Though, in this case, it is recommended to set TestOptions[2]=0x1 (PWR_SAVE_DISABLE=0x1), as this will tell the data loopback 1D to toggle all the AC and Dbyte slices.

Infinite-mode is only supported if DatLoopMinPwr = 0x0.

4.8.4 DQS Testing

In pad-side loopback mode, the looped back DQS signal is used to clock in the looped back DQ data. In core-loopback, an internally generated strobe is used to clock in the looped-back DQ data. For this reason, in core-loopback mode, dedicated DQS loopback testing is enabled by default. This dedicated DQS testing can also be enabled in pad-loopback mode via the message block input DatLoopDiffTestMode.

4.8.5 DQS + WCK DIFF Slice Single-Ended Testing

In pad-loopback mode (DatLoopCoreLoopBk=0x0), additional DQS and WCK DIFF slice single-ended checking can be enabled. Using the message block input DatLoopDiffTestMode, the WCK and DQS DIFF slices can also be tested in single-ended mode (either true or complement). This message block input is also used to enable dedicated DQS pad-loopback testing (section 4.8.4).

While in single-ended mode, the message block input DbVrefDac is used to configure the VREF for the other input to the differential receiver.

In core-loopback mode, single-ended testing is disabled, because the core-loopback point is before the single-ended circuitry.

4.8.6 How to Interpret the Output From the Data Loopback 1D Test Results

4.8.6.1 DatLoopbackCoarse

These values are the coarse delay bits for the corresponding DatLoopbackBitmap value. Meaning that the DatLoopbackCoarse value indicates the coarse value for where the 2UI bitmap starts. The starting coarse values can be different per Dbyte and Lane and should be used to normalize the data when comparing lanes.

For example, if DatLoopbackCoarse[0] is 4 and DatLoopbackCoarse[1] is 5 then the bitmap for DatLoopbackBitmap[0] starts in UI 4 and the DatLoopbackBitmap[1] starts in UI 5. To plot the data for both bits, the plot should contain data for UI 4 to UI 6. DatLoopbackBitmap[1] should be padded to contain all failing values for UI 4 and DatLoopbackBitmap[0] should be padded to contain all failing values for UI 6.

4.8.6.2 DatLoopbackBitmap

These values are a set of bitmaps for all the lanes of all the Dbyte instances. The bitmaps can be used to see the bitmap for a given lane. It is not necessary to read these locations to determine pass/fail. They are provided to be informational and to qualify failing results. `DatLoopbackBitmap[DbyteNum][7:0]` are DQ[7:0] for DbyteNum. `DatLoopbackBitmap[DbyteNum][8]` is DM/DBI for DbyteNum.

Interpreting the bitmap is analogous to the AC loopback SE slice bitmaps (see section): two 64bit array entries are used to create a 128bit wide EYE. The bits which correspond to tested delays are a function of the selected fine increment value.

DbLoopbackBitmap[CurDbyte][curSe][0]							DbLoopbackBitmap[CurDbyte][curSe][1]						
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]
Tested Delay 0	Tested Delay 1	Tested Delay 2	Tested Delay 3	...	Tested Delay 62	Tested Delay 63	Tested Delay 64	Tested Delay 65	Tested Delay 66	Tested Delay 67	...	Tested Delay 126	Tested Delay 127

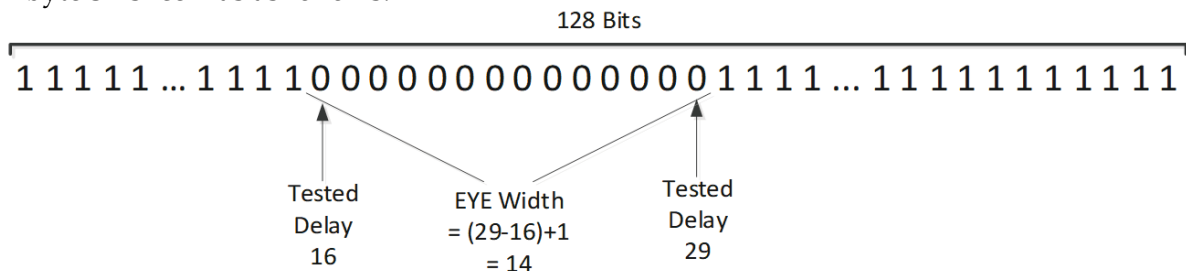
Example #1: `DatLoopFineIncr = 1:`

DbLoopbackBitmap[CurDbyte][curSe][0]							DbLoopbackBitmap[CurDbyte][curSe][1]						
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]
Pass/Fail Delay 0x00	Pass/Fail Delay 0x01	Pass/Fail Delay 0x02	Pass/Fail Delay 0x03	...	Pass/Fail Delay 0x3E	Pass/Fail Delay 0x3F	Pass/Fail Delay 0x40	Pass/Fail Delay 0x41	Pass/Fail Delay 0x42	Pass/Fail Delay 0x43	...	Pass/Fail Delay 0x7E	Pass/Fail Delay 0x7F

Example #2 `DatLoopFineIncr = 2:`

DbLoopbackBitmap[CurDbyte][curSe][0]							DbLoopbackBitmap[CurDbyte][curSe][1]						
Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]	Bit[0]	Bit[1]	Bit[2]	Bit[3]	...	Bit[62]	Bit[63]
Delay 0x00	Delay 0x02	Delay 0x04	Delay 0x06	...	Delay 0x7D	Delay 0x7F	1	1	1	1	...	1	1

The passing region for an EYE is the longest run of contiguous zeros. For example, if the extracted EYE for a given Dbyte SE slice was as follows:



The passing region for this bit is from delay 16 to 29, so a size of 14. If the `DatLoopMinEyeWidth` is set to 14 or lower, this lane would pass. If the `DatLoopMinEyeWidth` is set to 15 or higher, this lane would fail.

A consequence of the Dbyte PRBS checking hardware is that the Dbyte SE EYEs will not wrap around from the max delay to the min delay (unlike the AC SE slices).

**Note**

Note that when comparing bitmaps for different lanes, the differences in `DatLoopbackCoarse` values must be accounted for to align the eyes relative to each other correctly.

4.8.6.3 `DatLoopbackWckBitmap`

Interpreting the Dbyte WCK bitmap is analogous to the AC loopback DIFF slice bitmaps (see section 4.7.5.3): up to three 64bit array entries are available for storing the EYE in the message block output `DatLoopbackWckBitmap`. The exact number of entries required will be returned in the message block output `DbLoopbackNumUiWck`. The bits which correspond to tested delays are a function of the selected fine increment value.

There is inherent metastability in the phase-detect hardware. Therefore, in real silicon one will likely observe fuzzy EYEs, whose width is greater than 1UI. However, the important observation is whether an EYE exists, and that it fulfills a given minimum EYE width criteria, as controlled by the message block input `DatLoopMinEyeWidth`.

Because the measured EYE width can be greater than 1UI, the output bitmap has enough capacity for 3UI worth of sample data, to ensure the entire EYE is captured. The message block output `DbLoopbackNumUiWck` indicates the number of UIs that were required to capture the entire EYE. The following formula can be used to determine how many bits in the bitmap are required to assemble the EYE.

$$\text{NumUsedBits} = \text{roundUp} \left(\frac{64 * \text{DbLoopbackNumUiWck}}{\text{DatLoopFineIncr}} \right)$$

All unused bitmap locations will be 1 at the end of the test.

The algorithm used to store the WCK bitmap is analogous to the AC DIFF slice algorithm (section 4.7.5.3). Also, like the AC DIFF slices, the passing region can wrap around from the max delay to the min delay.

**Note**

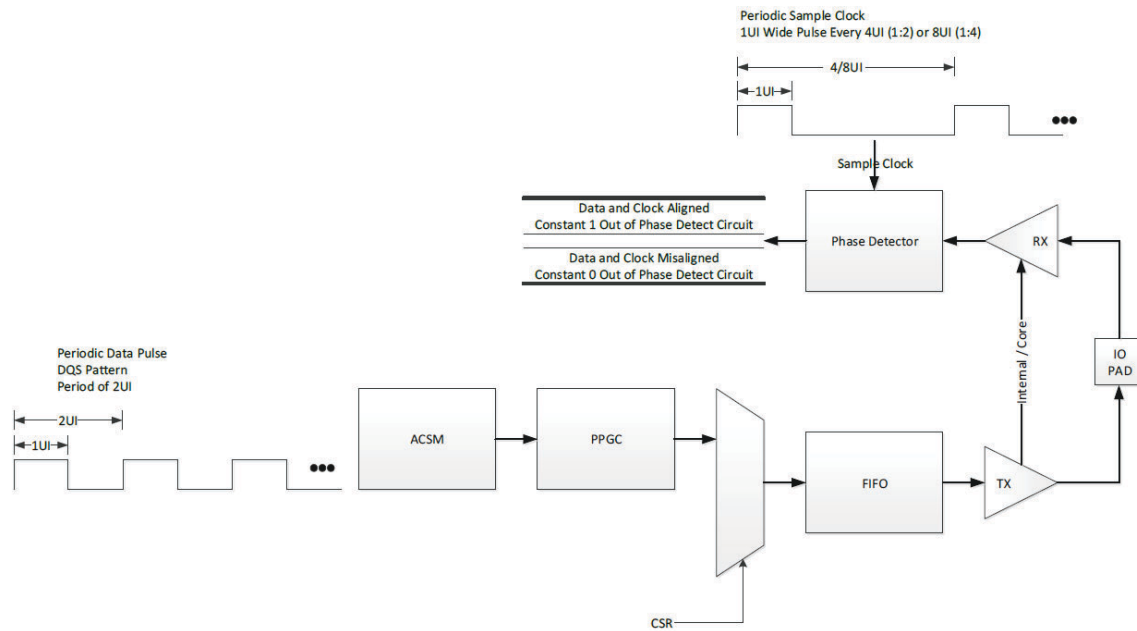
Because of the nature of the DIFF slice phase-detect phase detect circuitry used to perform the test, the passing region wraps from the end of the bitmap to the beginning of the bitmap. A passing region with 10 passing at the end of the bitmap and 12 passing at the beginning of the bitmap with all other values failing would also have a width of 22.
[This does not apply to the data loopback test results.]

4.8.6.4 `DatLoopbackDqsBitmap`

The DQS DIFF slice is tested by driving a 1UI wide pulse with a period of 2UI. Two 64bit array entries are used for storing the EYE in the message block output `DatLoopbackDqsBitmap`.

Again, because of metastability in the phase detect sampler, measured EYE widths can be greater than 1UI. However, the key observation is that the EYE exists, and that it fulfills a given minimum EYE width criteria, as controlled by the message block input `DatLoopMinEyeWidth`.

Because the period of the pattern is 2UI, and the output bitmap is 2UI wide, the entire swept region can be held in the output message block field. Therefore, there is no 'NumUi' qualifier for the DQS bitmap. Though, like the WCK bitmap, the passing region can wrap-around from the max-delay to the min-delay.



4.8.6.5 DatLoopbackWckCoarse

These values are the coarse delay bits for the corresponding DatLoopbackWckBitmap value. Meaning that the value of DatLoopbackWckCoarse indicates the coarse value for where the 3UI bitmap starts. The starting coarse values can be different per Dbyte and should be used to normalize the data when comparing lanes.

For example, if DatLoopbackWckCoarse[0] is 4 and DatLoopbackWckCoarse[1] is 5 then the bitmap for DatLoopbackWckBitmap[0] starts in UI 4 and the DatLoopbackWckBitmap[1] starts in UI 5. To plot the data for both bits, the plot should contain data for UI 4 to UI 6. DatLoopbackWckBitmap[1] should be padded to contain all failing values for UI 4 and DatLoopbackWckBitmap[0] should be padded to contain all failing values for UI 6.

4.8.6.6 DatLoopbackDqsCoarse

These values are the coarse delay bits for the corresponding DatLoopbackDqsBitmap value. Meaning that the value of DatLoopbackDqsCoarse indicates the coarse value for where the 2UI bitmap starts. The starting coarse values can be different per Dbyte and should be used to normalize the data when comparing lanes.

For example, if DatLoopbackDqsCoarse[0] is 4 and DatLoopbackDqsCoarse[1] is 5 then the bitmap for DatLoopbackDqsBitmap[0] starts in UI 4 and the DatLoopbackDqsBitmap[1] starts in UI 5. To plot the data for both bits, the plot should contain data for UI 4 to UI 6. DatLoopbackDqsBitmap[1] should be padded to contain all failing values for UI 4 and DatLoopbackDqsBitmap[0] should be padded to contain all failing values for UI 6.

4.8.7 Pre-test requirements

It is required that the Impedance Calibration and PLL/LCDL Lock tests be run prior to running the Data Loopback 1D test. Because of this requirement, the firmware will automatically enable these tests whenever the Data Loopback 1D test is enabled. The PassFailResults message block field will include the results for these tests, even if they are not specified in the TestsToRun field.

**Note**

Because the PLL Lock test is required for the Data Loopback 1D test, this test must be run with a DfiClk within the legal range of the PLL to lock. Termination on Data Pins for Loopback

4.8.8 Termination on Data Pins for Loopback

For Pad-side loopback, DQS, DQ, and (if applicable) WCK pins must be terminated. Termination strength of 50 Ohms is suggested. Termination voltage is VSS. The pads do not need to be terminated for core loopback.

4.8.9 Optimizing Data Loopback Runtime

The data loopback test can take significant time to run if the entire delay range is scanned. For most designs, it is not necessary to scan the full range. If the delay range is reduced, it will cause a corresponding decrease in run time. To optimize the run time, perform the following:

1. Run the data loopback test specifying the full range for DatLoopCoarseStart (use 0) and DatLoopCoarseEnd (use 7).
2. Read out all the DataLoopbackCoarse values that the test reports.
3. For the optimized values, use the following:
 - a. Use the minimum DataLoopbackCoarse[] as the DatLoopCoarseStart value. Might need to subtract 1 from this value if the eye is very close to the beginning of the range or if chip variation is large.
 - b. Use the maximum DataLoopbackCoarse[] plus one as the DatLoopCoarseEnd value. Might need to add two from this value instead of one if the eye is very close to the end of the range or if chip variation is large.

Note, that the EYE position will vary, depending on whether the test is running in:

- Core or pad-loopback mode
- 1:2 or 1:4 mode
- LP4 or LP5 mode

Therefore, an optimized DatLoopCoarseStart and DatLoopCoarseEnd pair should be measured for each group of settings.

4.9 Data loopback 2D

This test measures and returns the EYE width and height for a targeted DQ bit that the user specifies. This test is like Data Loopback 1D (section 4.8), but now the Vref is varied as well. The user can select the starting and ending Vrefs, and the Vref increment. A bitmap is constructed for the chosen DQ lane at each Vref. Combining the bitmaps together results in a 2D mapping of the EYE.

During execution, every DQ bit will be tested across the user specified delays and Vrefs. However, the message block inputs DatLoopByte and DatLoopBit select the DQ bit whose EYE will be stored in the message block.

Fail condition(s):

- One or more SE slices fail the MinEyeWidth check at any tested Vref

4.9.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Data Loopback 2D test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes and set the bit to 0 if the test fails.
DatLoopClksToRun	Input	Range: 0,100-65535 Recommended: 512 100-65535: Wait the specified number of DFCLK cycles. 0: Run until user tells the test to stop. The user must cause DctWriteProt to transition from 0x0 to 0x1. Only supported if DatLoopMinLoopPwr is also 0x0.	Dbyte SE slice PRBS transmit duration in DFCLKs. Note: Setting this field to 0x0 will cause the PRBS pattern to transmit until DctWriteProt CSR bit transitions from 0x0 to 0x1. Refer to section 4.9.3. Note: A value of 0x0 is only supported if DatLoopMinLoopPwr is also 0x0.
DatLoopMinLoopPwr	Input	0x0 = All Dbytes tested in parallel 0x1 = Only 1 Dbyte active at a time	Parallel testing will run faster, but results in higher power consumption. Serial testing will consume less power, but takes longer to run. Note: A value of 0x1 is only supported if DatLoopClksToRun is non-zero.
DatLoopCoreLoopBk	Input	Pad / Core loopback 0x0 = Pad loopback (recommended) 0x1 = Core loopback	Pad loopback requires proper termination (section 4.9.3). Pad-side loopback is recommended, as core-side loopback will bypass Vref.

Field Name	Direction	Legal / Recommended Values	Comments
TxDqsDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
TxWckDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxEnDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxDigStrbDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxCkT2UIDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
RxCkC2UIDly	Input	Set to 0xFFFF.	Must be set to the legal / recommended value.
DatLoopCoarseStart	Input	Range: 0-7 Recommended to set this to 0.	This is the start of the coarse range. The test will start at the beginning of this coarse value.
DatLoopCoarseEnd	Input	Range: 0-7 Recommended to set this to 7.	This is the end of the coarse range. The test will stop at the end of this coarse value.
DatLoopFinelIncr	Input	Range: 1-7 Recommended to set this to 1.	This defines the value which the delay is incremented for each measurement (in terms of fine 1/64UI steps). Setting this to a value of other than 0x1 will limit the resolution of the measurements. When using a DatLoopFinelIncr greater than 0x1, the value of DatLoopMinEyeWidth must be scaled appropriately.
DatLoopMinEyeWidth	Input	Recommended 0x0030 for core loopback and 0x0020 for pad loopback. Must be divided by DatLoopFinelIncr if DatLoopFinelIncr is larger than 0x1.	The number of consecutive tested delays which must pass for Dbyte SE slice testing at each tested Vref setting. The chosen value should be greater than zero. If (0) is programmed the EYE width checks will always pass.
DatLoop2dVrefStart	Input	0-127	Used to set the starting Vref for the 2D scan. This also defines the minimum Vref that must pass the DatLoopMinEyeWidth criteria.
DatLoop2dVrefEnd	Input	0-127	Used to set the ending Vref for the 2D scan. This also defines the maximum Vref that must pass the DatLoopMinEyeWidth criteria.

Field Name	Direction	Legal / Recommended Values	Comments
DatLoop2dVrefIncr	Input	1-127	This defines the value which the VREF is incremented for each measurement.
DatLoopByte	Input	0-(NumDbytes-1)	DatLoopByte and DatLoopBit together select the DQ bit (SE slice) whose 2D eye will be saved in the output field DatLoop2dVrefBitmap.
DatLoopBit	Input	0-(NumSeSlices-1)	DatLoopByte and DatLoopBit together select the DQ bit (SE slice) whose 2D eye will be saved in the output field DatLoop2dVrefBitmap.
DbRxVrefCtl	Input	Refer to description of CSR 'RxVrefCtl' for field description	
DbRxDfeModeCfg	Input	Refer to description of CSR 'RxDfeModeCfg' for field description	
DbTxImpedanceSe	Input	Refer to description of CSRs 'TxImpedanceSE*' for field breakdown	
DbTxImpedanceDiff	Input	Refer to description of CSRs 'TxImpedanceDIFF*' for field breakdown	
DbTxSlewSe	Input	Refer to description of CSRs 'TxSlewSE*' for field description	
DbTxSlewDiff	Input	Refer to description of CSRs 'TxSlewDIFF*' for field description	
DatLoopbackRxEnbVal[DbyteNum]	Output	Any	Data loopback RxEnb delay value, indexed by Dbyte. In pad-side loopback, this will store the trained RxEn delays. In core-side loopback this will contain all 1's (section 4.8.4).
DatLoop2dVrefCoarse[VrefIdx]	Output	Any	SE slice data loopback 2D eye bitmap starting coarse value indexed by Vref. See section 4.9.4.1 to understand how to interpret the data.
DatLoop2dVrefBitmap[VrefIdx][UI0/1]	Output	Any	SE slice data loopback 2D eye bitmap indexed by Vref. See 4.9.4.2 section to understand how to interpret the data.

Field Name	Direction	Legal / Recommended Values	Comments
DatLoop2dDbyteBitFail [DbyteNum]	Output	Any	Data Loopback array of bit vectors indicating which Dbyte lanes failed 2D eye checks. See section 4.9.4.3 to understand how to interpret the data.

4.9.2 Message Block Inputs - Illegal Argument Checking

The following checks are performed on the message block inputs

$$DatLoopFineIncr > 0$$

$$DatLoopCoarseStart \leq DatLoopCoarseEnd$$

$$DatLoopCoarseStart \leq 7$$

$$DatLoopCoarseEnd \leq 7$$

$$DatLoop2dVrefFineIncr > 0$$

$$DatLoop2dVrefStart \leq DatLoop2dVrefEnd$$

$$DatLoop2dVrefStart \leq 127$$

$$DatLoop2dVrefEnd \leq 127$$

If any condition is false, the test will be marked as fail, the outputs associated data loopback 2D will be initialized with the code 0xA5, and the test will return immediately.

4.9.3 Data Loopback 2D Infinite-Traffic Mode

Data Loopback 2D supports an “infinite-traffic” mode, which is enabled by setting `DatLoopClksToRun` to be 0. In this mode, traffic is transmitted on the SE slices, at the current Vref and delay setting, until the user causes `DctWriteProt` to transition from 0x0 to 0x1. The test will then proceed to the next delay to test, based on the values of `DatLoopCoarseStart`, `DatLoopCoarseEnd`, and `DatLoopFineIncr`. Once the delay range has been swept from beginning to end, the test will proceed to the next Vref setting, based on the values of `DatLoop2dVrefStart`, `DatLoop2dVrefEnd`, and `DatLoop2dVrefIncr`. Once all the chosen Vrefs have been swept across all chosen delays (with a `DctWriteProt` transition from 0x0 to 0x1 at each position) the test will check the collected results, and return all the corresponding message block outputs.

In total, the following number of transitions of `DctWriteProt` from 0x0 to 0x1 is required:

$$\begin{aligned}
 &NumTransitions \\
 &= \left[\text{roundDown} \left(\frac{64 * (DatLoopCoarseEnd - DatLoopCoarseStart)}{DatLoopFineIncr} \right) + 1 \right] \\
 &\quad * \left[\text{roundDown} \left(\frac{DatLoop2dVrefEnd - DatLoop2dVrefStart}{DatLoop2dVrefIncr} \right) + 1 \right]
 \end{aligned}$$

The trivial case occurs when `DatLoopCoarseEnd` equals `DatLoopCoarseStart` and `DatLoop2dVrefEnd` equals `DatLoop2dVrefStart`, as only one transition will be required.

This mode can be used to characterize the drivers. Though, in this case, it is recommended to set `TestOptions[2] = 1`, as this will tell the data loopback 2D to enable all the AC and Dbyte slices.

Infinite-mode is only supported if $\text{DatLoopMinPwr} = 0 \times 0$.

4.9.4 How to Interpret the Output From the Data Loopback 2D Test Results

4.9.4.1 DatLoop2dVrefCoarse

These values are the coarse delays for the corresponding $\text{DatLoop2dVrefBitmap}$ value indexed by Vref. Meaning that the $\text{DatLoop2dVrefCoarse}$ value indicates the coarse value for where the 2UI bitmap starts for a given Vref. The starting coarse values can differ between Vrefs, and should be used to normalize the data when comparing Vrefs.

For example, if $\text{DatLoop2dVrefCoarse}[0]$ is 4 and $\text{DatLoop2dVrefCoarse}[1]$ is 5 then the bitmap for $\text{DatLoop2dVrefBitmap}[0]$ starts in UI 4 and the $\text{DatLoop2dVrefBitmap}[1]$ starts in UI 5. The plot for the full bitmap should contain data for UI 4 to UI 6. $\text{DatLoop2dVrefBitmap}[1]$ should be padded with failing values for UI 4 and $\text{DatLoop2dVrefBitmap}[0]$ should be padded with failing values for UI 6.

4.9.4.2 DatLoop2dVrefBitmap

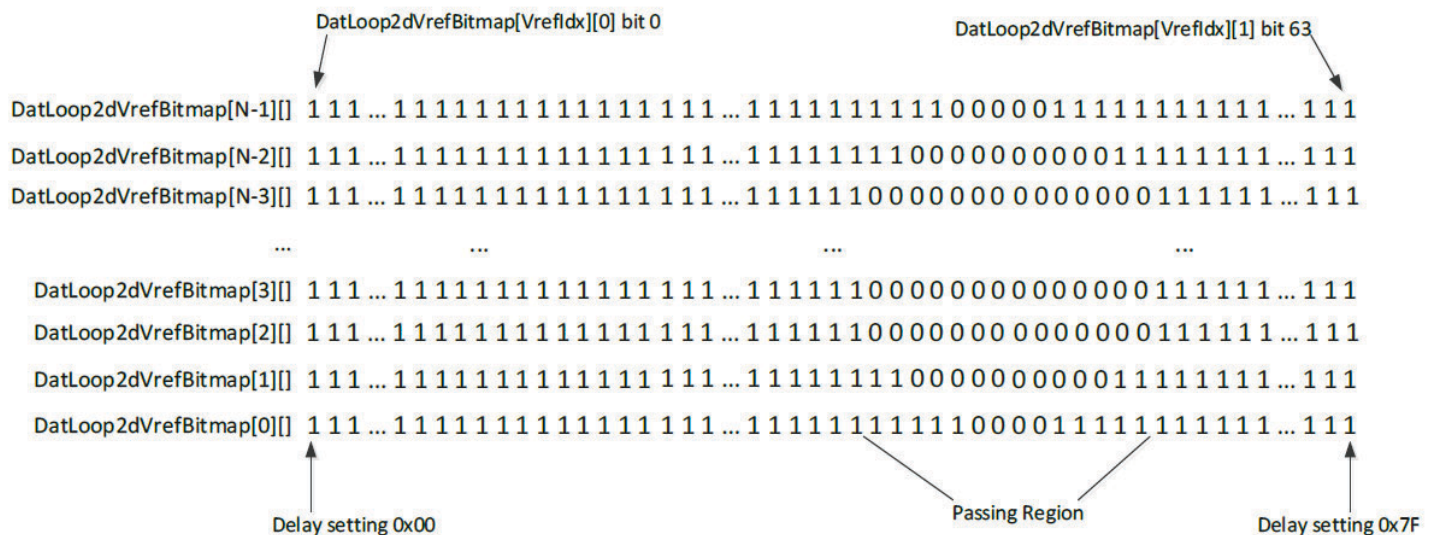
The output from the 2D Data loopback test is a set of bitmaps for the lane of the Dbyte instance under test. The bitmaps are used to determine the passing and failing regions for the pin. To see the bitmap for a given lane, concatenate the UI0 and UI1 fields for each array value. The array data will be valid from entry [0] through [N-1], where N can be found using the following formula:

$$N = \text{ceiling} \left(\frac{128}{\text{DatLoop2dVrefIncr}} \right)$$

For example:

- $\text{DatLoop2dVrefIncr} = 1 \rightarrow N = 128$ (0:127)
- $\text{DatLoop2dVrefIncr} = 3 \rightarrow N = 43$ (0:42)

In general, the overall bitmap will appear as follows:



**Note**

The DatLoop2dVrefCoarse values must be used to align the individual eyes for each Vref. If the values differ, the eyes must be padded to align corresponding coarse values. Padded eyes should be filled with failing delays, as any eye can only span at most 2UI, being 1UI in size with unknown alignment to the coarse values.

4.9.4.3 DatLoop2dDbyteBitFail

The 2D data loopback test fails if any DBYTE lane fails the minimum eye width check at any VREF. If the test fails, this vector can be used to ascertain which DBYTE lane(s) were the source of the failure.

Index of array is the DBYTE, and bits [8:0] correspond to the bits of a particular DBYTE. Bits [15:9] are RESERVED.

To check the status of (DBYTE#, LANE#), extract the data as follows:

If (DatLoop2dDbyteBitFail[DBYTE] & (1 << bit)) equals:

- 0 - The corresponding DBYTE bit passed all eye width checks across all tested VREFs
- 1 - The corresponding DBYTE bit failed an eye width check on one or more tested VREFs

4.9.5 Pre-test Requirements

It is required that the Impedance Calibration and PLL/LCDL Lock tests be run prior to running the Data Loopback 2D test. Because of this requirement, the firmware will automatically enable these tests whenever the Data Loopback 2D test is enabled. The PassFailResults message block field will include the results for these tests, even if they are not specified in the TestsToRun field.

**Note**

Because the PLL Lock test is required for the Data Loopback 2D test, this test must be run with a DfIClk within the legal range of the PLL to lock.

4.9.6 Termination on Data Pins for Pad Loopback

Refer to section 4.8.8 for details on how to terminate the Dbyte IOs.

4.9.7 Optimization of the run time for the 2D Data Loopback test

See section 4.8.9 for information on how to optimize the Data Loopback run time.

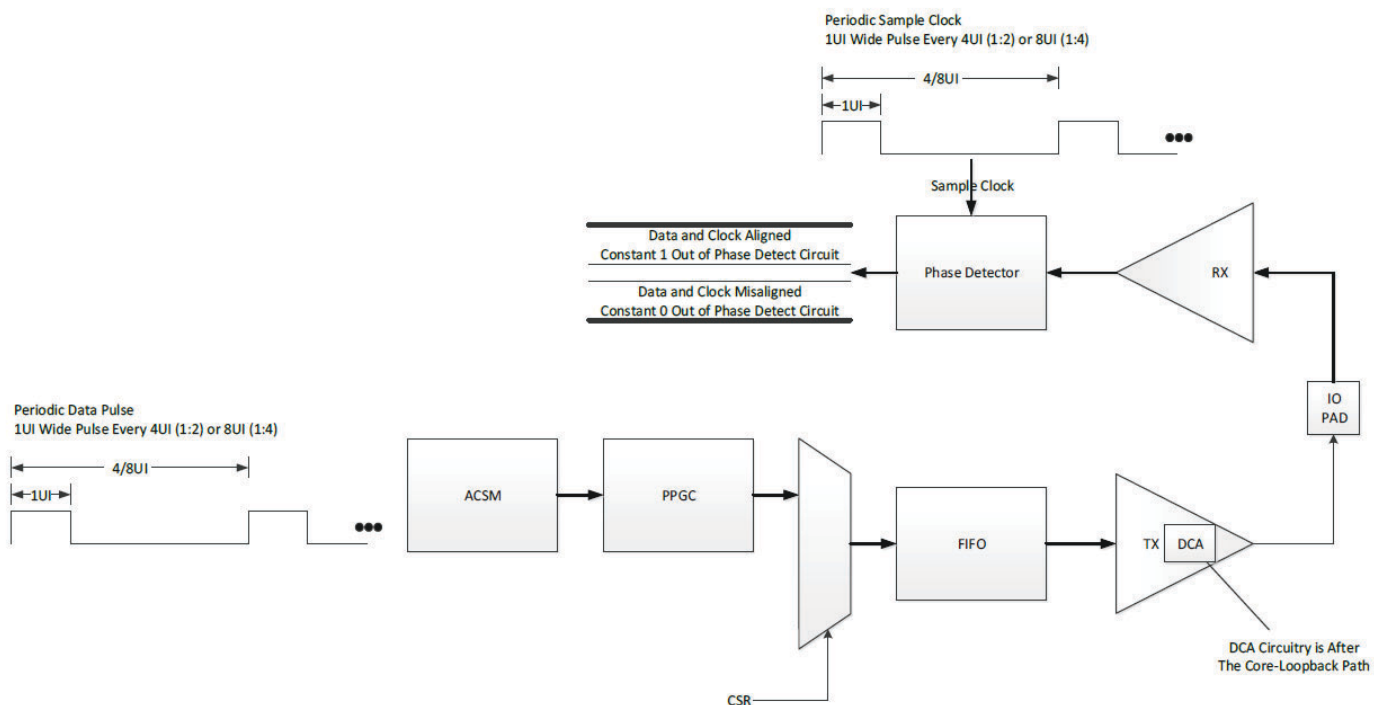
4.10 DCA Loopback

The purpose of this test is to exercise the WCK DIFF slice DCA (duty cycle adjustment) circuit, which is supported for the WCK frequency range 2.5GHz to 3.2GHz (refer to section 4.1.3). The DCA supports 4 coarse settings (0...3), and 13 fine settings (0...12). The message block inputs `DcaLoopDcaCoarseSkip` and `DcaLoopDcaFineIncr` are used to control which coarse and fine settings are tested.

In order to reduce the runtime, the full delay range is only swept for the first tested DCA coarse + fine setting: 4UI in 1:2 mode, 8UI in 1:4. For each subsequent iteration, a 3UI window surrounding the passing region is swept. The size of the window is always 3UI, regardless of the frequency ratio.

The DCA is tested at each selected fine and coarse setting by transmitting a periodic pulse. The pulse is then looped back at the pad, and sampled by a phase-detector on the receive path. An EYE is constructed by sweeping the delay, and examining the sampled receive data: using this information, the EYE width for a given DCA setting is measured. The test then repeats this process for each DCA coarse and fine setting to be tested (controlled via the message block). Once all the EYE width data has been gathered, the test verifies that as the DCA fine setting increases the EYE width in general becomes wider.

Additional information to qualify the DCA test pass / fail results are provided in the message block: the measured EYE width at each tested DCA coarse + fine setting, as well as the raw bitmaps for one WCK DIFF slice (selected using the message block input `DcaLoopBitmapSel`).



The DCA circuit is only available in PUB version 1.02a and above.

The DCA circuit only exists in LP5 enabled PHYs. If the DCA loopback test is run on an LP4-only PHY, the test will initialize the DCA specific outputs to all 0xA5's, and return FAIL.

Because the DCA circuit is after the core-loopback point, the DCA loopback test always runs in pad-loopback mode. Consequently, the Dbyte WCK pins must be properly terminated (see section 4.10.6).

Fail condition(s):

- The test is run on an LP4-only PHY.
- In any WCK, at any DCA setting, the width of the EYE is 0.
- In any WCK, at any DCA coarse setting:
 - The sum of EYE width differences across tested DCA fine settings is ≤ 0 .



Note

The DCA loopback test is not supported for LP4-only PHYs.



Note

DCA is only available in PUB versions 1.02a and above.



Note

The DCA loopback test always runs in pad-side loopback mode. Therefore, it requires the WCK pads to be properly terminated (section 4.10.6).

4.10.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the DCA Loopback test to run.
PassFailResults	Output	See section 4.1.1.	The firmware will set corresponding bit to 1 if the test passes and set the bit to 0 if the test fails.
DbTxImpedanceDiff	Input	Refer to description of CSRs 'TxImpedanceDIFF*' for field breakdown	
DbTxSlewDiff	Input	Refer to description of CSRs 'TxSlewDIFF*' for field description	
DcaLoopDcaCoarseSkip	Input	Setting Bit[i]=0x1 skips testing DCA Coarse=i Bit[0]=0x1 – Skip DCA Coarse=0x0 Bit[1]=0x1 – Skip DCA Coarse=0x1 Bit[2]=0x1 – Skip DCA Coarse=0x2 Bit[3]=0x1 – Skip DCA Coarse=0x3	By default, all 4 DCA coarse values are tested. Setting individual bits skips testing for the corresponding DCA coarse setting. The chosen value should not be 0xF, as this will cause the test to always return PASS.

Field Name	Direction	Legal / Recommended Values	Comments
DcaLoopDcaFineIncr	Input	Valid Values: 1,2,3,6	Increment used to sweep the 13 DCA fine settings (0 to 12). Chosen value must be a multiple of 6.
DcaLoopDelayIncr	Input	Recommended to set this to 1.	When constructing the EYE for a DCA setting, this is the delay fine-step used to sweep the delay range. It is recommended to use a value of 1, as using a larger value could result in the test failing to find changes to the WCK EYE width, and returning a failure.
DcaLoopMinLoopPwr	Input	0x0 = All WCKs tested in parallel 0x1 = Only 1 WCK active at a time	Parallel testing will run faster, but results in higher power consumption. Serial testing will consume less power, but takes longer to run.
DcaLoopBitmapSel	Input	Range: 0-(NumDbytes-1)	The generated bitmaps for one WCK slice are saved to the message block output 'DcaLoopBitmap'. This field selects the WCK slice.
DcaLoopMaxEyeWidth[D byteNum] [DcaCoarse][DcaFine]	Output	Any	For each Dbyte WCK and tested DCA setting, the size of the largest contiguous passing region is returned. Index by Dbyte number, DCA coarse setting, and DCA fine setting. Refer to section 4.10.4.1 for more details.
DcaLoopBitmap[DcaCoar se] [DcaFine][0/1/2]	Output	Any	The bitmaps for the Dbyte WCK slice selected using the input DcaLoopBitmapSel. Up to 3UI worth of data is saved for each tested DCA setting (the exact number of UIs is qualified using DcaLoopNumUi). Refer to section 4.10.4.2 for more details.
DcaLoopNumUi[DcaCoar se][DcaFine]	Output	Any	The number of UIs that were required to capture the selected WCK EYE at each DCA setting.
Dca_Delay [DbyteNum][0/1]	Output	Any	Starting delay of the DcaLoopBitmap, Refer to section 4.10.4.3 for more details.

4.10.2 Message Block Inputs – Illegal Argument Checking

The following checks are performed on the message block inputs:

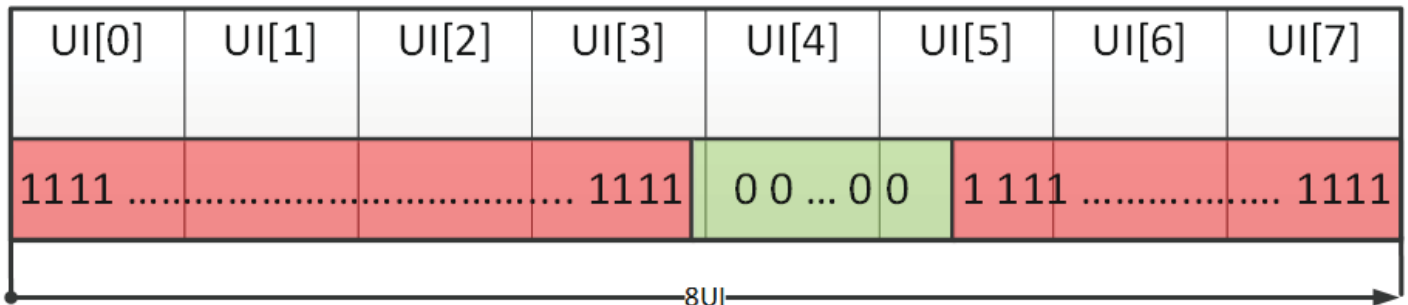
$$DcaLoopDcaFineIncr=\{1,2,3,6\}$$

$$DcaLoopDelayIncr>0$$

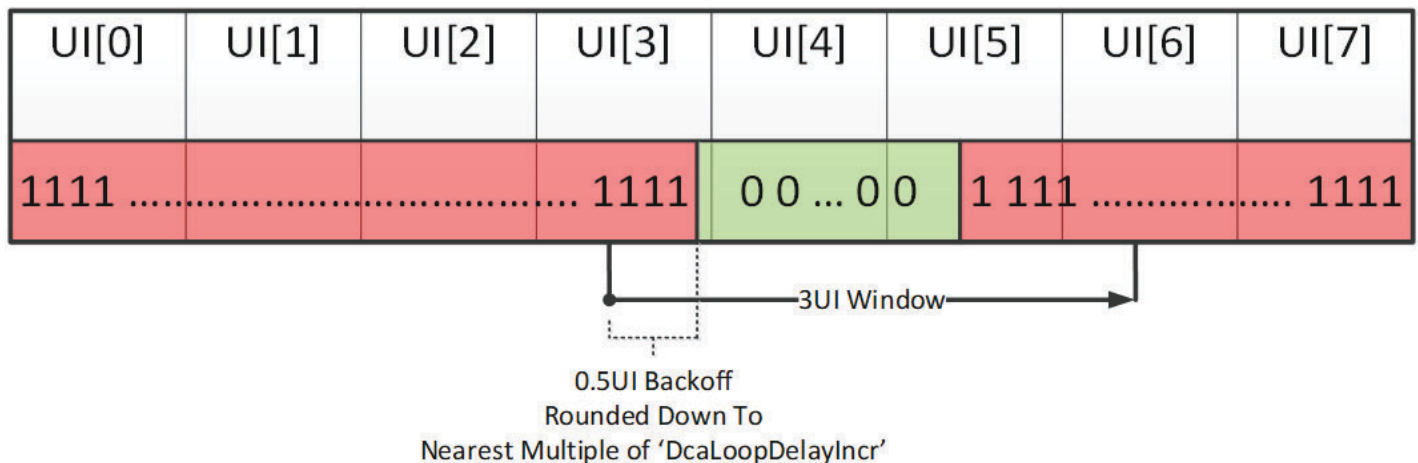
If any condition is false, the test will be marked as fail, the outputs associated DCA Loopback will be initialized with the code 0xA5, and the test will return immediately.

4.10.3 DCA Delay Sweep Optimization

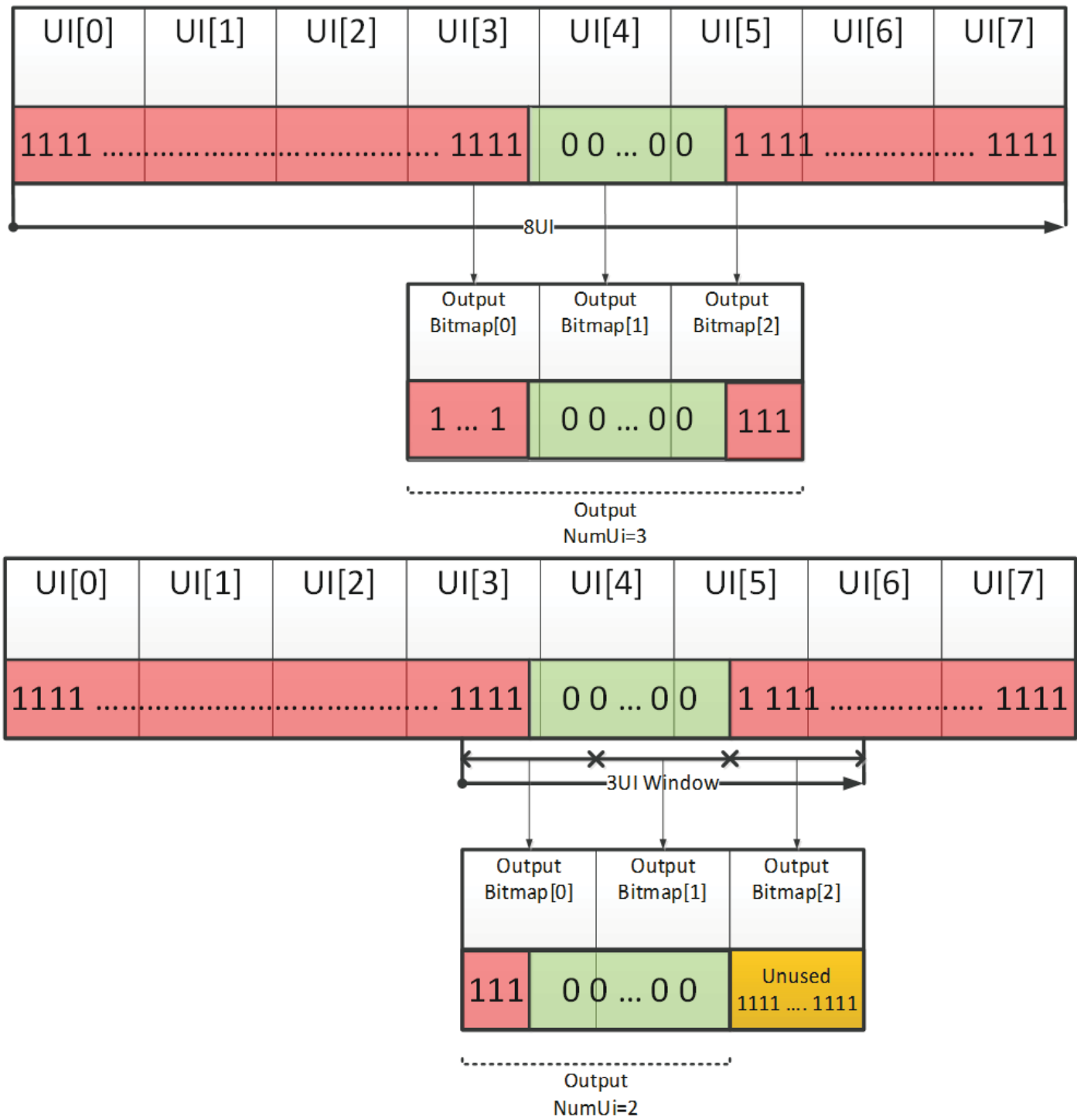
For the first tested DCA setting, the entire delay range is swept (4UI in 1:2 mode, 8UI in 1:4 mode). The example below uses 1:4 mode.



Based on the location of the passing region, the ATE-FW will sweep a 3UI window, which covers the passing region. The 3UI window starts 0.5UI before the start of the passing region (rounding down to the nearest multiple of $DcaLoopDelayIncr$). The 3UI window illustrated below is used for all subsequent DCA settings. A 3UI window is used for both 1:2 and 1:4 mode.



The effect of using two window types means some bitmap interpretation is required. Continuing for the current example.



4.10.4 How to Interpret the Output From the DCA Loopback Test Results

4.10.4.1 DcaLoopMaxEyeWidth

At each tested DCA coarse and fine setting, the measured EYE width for each WCK DIFF slice is stored in the message block output `DcaLoopMaxEyeWidth`. Due to metastability on the sampling interface, the edges of the EYE can be fuzzy, so the “max EYE width” is defined as the widest contiguous passing region.

In the following example, the EYE width would be measured to be 14.

1 1 1 1 1 ... 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 ... 1 1 1
└──┘
 Max EYE Width = 14

As a further consequence of the fuzzy edges to the EYE, it is difficult to precisely measure the effect of the DCA at every coarse and fine setting. Therefore, the expected behavior is taken to be that as the DCA fine setting increases, the max EYE width in general becomes wider.

4.10.4.2 DcaLoopBitmap

The DCA loopback test message block input `DcaLoopBitmapSel` selects one Dbyte WCK DIFF slice. For this WCK slice, a bitmap is saved for each tested DCA coarse and fine setting. The array is first indexed by DCA coarse and then by DCA fine setting. Up-to 3UI of bitmap are stored at each location, where the message block output `DcaLoopNumUi` indicates the number of UIs that have been captured. Only tested DCA settings (based on `DcaLoopDcaCoarseSkip` and `DcaLoopDcaFineIncr`) will be populated with a bitmap. Delay increments greater than 1 are supported in the DCA loopback test, but not recommended, as using a fine-step larger than 1 will reduce measurement resolution. However, if a fine-step larger than 1 is used, the bitmap will be compacted in a manner analogous to “`DatLoopbackWckBitmap`”. Refer to sections 4.7.5.3 and 4.8.6.3 for more details on how to unpack DIFF slice / WCK bitmaps. Also, refer to section 4.10.3 for the effect of the delay-optimization has on bitmap storing.

The formula for the number of bits to concatenate to form a given DCA bitmap can be found as follows:

$$\text{NumUsedBits} = \text{roundUp} \left(\frac{64 * \text{DcaLoopNumUi}}{\text{DcaLoopDelayIncr}} \right)$$

4.10.4.3 Dca_Delay

These values are the starting delay of the DcaLoopBitmap. Meaning that the value of Dca_Delay indicates the starting delay where the 3UI bitmap starts. The starting coarse values can be different per Dbyte and should be used to normalize the data when comparing lanes.

Dca_Delay[DbyteNum][Dca_Delay0] is for the smallest tested DcaCoarse setting (based on DcaCoarseSkip) and DcaFine=12.

For example:

- If DcaCoarseSkip=0, Dca_Delay0 -> DcaCoarse=0
- If DcaCoarseSkip=1, Dca_Delay0 -> DcaCoarse=1
- If DcaCoarseSkip=3, Dca_Delay0 -> DcaCoarse=2
- If DcaCoarseSkip=7, Dca_Delay0 -> DcaCoarse=3

Dca_Delay[DbyteNum][Dca_Delay1] is for all subsequent ones.

4.10.5 Pre-test requirements

It is required that the Impedance Calibration and PLL/LCDL Lock tests be run prior to running the DCA Loopback test. Because of this requirement, the firmware will automatically enable these tests whenever the DCA Loopback test is enabled. The PassFailResults message block field will include the results for these tests, even if they are not specified in the TestsToRun field.

4.10.6 Termination on WCK Pins for DCA Testing

The DCA loopback test requires that the WCK pins be properly terminated. Refer to section for details on how to terminate the Dbyte IOs.

4.11 Burn-in

The Burn-in test is designed for accelerated life testing, it is not run as part of the production ATE testing. For this reason, customers running the Burn-In test must not select any other tests in the TestsToRun message block input (except for the Impedance Calibration and PLL-Lock tests, which will be automatically run if the Burn-In test is selected).



Note

The Burn-In test must be run in isolation. If the Burn-In test has been selected in TestsToRun, the only other tests which may be selected are the Impedance Calibration and PLL-Lock tests.



Note

The Burn-In test PassFailResults bit does not indicate that any test criteria passed. It only indicates that the test ran to completion. The bit will only be set to 0 if the test was not exited properly or failed to run to completion.

Burn-in will generate traffic until it observes that DctWriteProtShadow has transitioned from 0 to 1. Burn-in will return PASS once the test finishes.

The Burn-In test does the following:

- Set up the PHY to perform both Addr/Cmd and Data loopback.
- Set up the PHY LCDLs for 63/64UI of delay
- Set up the PHY to not check the data in either the AC or the Dbyte instances
- Set up the drivers for maximum drive strength
- Set up the AC and Dbyte receiver VREFs based on the corresponding message block fields.
- Run patterns until the user stops the test by stalling the microcontroller / resetting the PHY.

While Burn-in is running, a 100Hz “heartbeat” signal is driven on the DTO pin.

The Burn-in test exercises those AC slices which are active during mission-mode: unused slices will remain inactive. For this reason, it is recommended to use LP4 clocking mode for PHYs which will be used in LP4 context, and LP5 clocking mode for PHYs which will be used in an LP5 context.

If a PHY will be used for both LP4 and LP5 contexts, it is recommended to allocate silicon for running Burn-In in LP4 and LP5 clocking-modes. It is possible for all AC slices to be active by setting TestOptions[3]=0x1 (AC_PINS_PRBS), as this will cause ALL AC pins to drive a PRBS pattern. However, it also results in the SEC slices being driven at data-rates above spec.



Note

It is recommended to run Burn-In using the clocking-mode corresponding to the operational mode the PHY will run in.

LP4-only --> Run Burn-In with clocking-mode set to LP4

LP5-only --> Run Burn-In with clocking-mode set to LP5

LP4+LP5 --> Allocate silicon for running Burn-In with clocking mode set to LP4, and allocate silicon for running Burn-In with clocking mode set to LP5.

4.11.1 Message Block Inputs / Outputs and legal values:

Field Name	Direction	Legal / Recommended Values	Comments
TestsToRun	Input	See section 4.1.1.	Set the corresponding bit to cause the Burn-In test to run.
PassFailResults	Output	See section 4.1.1.	The corresponding bit will be (0) if the test was not exited properly or failed to run to completion. The corresponding bit will be set to (1) if the test is exited properly and runs to completion.
AcVrefDac	Input	Refer to description of CSRs 'AcVrefDAC*' for field description	
DbVrefDac	Input	Refer to description of CSRs 'VrefDAC*' for field description	
DbRxVrefCtl	Input	Refer to description of CSR 'RxVrefCtl' for field description	
DbRxDfeModeCfg	Input	Refer to description of CSR 'RxDfeModeCfg' for field description	
DatLoopCoreLoopBk	Input	Pad / Core loopback 0x0 = Pad loopback (recommended) 0x1 = Core loopback	It is recommended to run Burn-In in pad loopback mode, as this maximizes analog circuitry coverage. Pad loopback requires proper termination.
DatLoopMinLoopPwr	Input	0x0 = All Dbytes active at once 0x1 = Only 1 Dbyte active at a time	If enabled, traffic will only be sent on Dbyte[i]. Once DctWriteProt transitions from 0x0 to 0x1, traffic will only continually transmit on Dbyte[i+1]. DctWriteProt must transition from 0x0 to 0x1 'NumDbytes' times to move through all the Dbytes.

4.11.2 Running the Burn-In Test

To run the Burn-In test, use the following procedure:

1. Use the normal process to load and start the ATE firmware, writing the TestsToRun field to indicate the Burn-In test should run.
2. Wait the required number of cycles for main loop overhead.
3. Wait the amount of Burn-In time desired.

4. Cause an appropriate number of 0x0 to 0x1 transitions of DctWriteProt
 - a. If DatLoopMinLoopPwr=0x0, only 1 transition is required
 - b. If DatLoopMinLoopPwr=0x1, 'NumDbyte' transitions are required
 - i. User must wait at least 10,000*DfiClkFreq DFI clocks between subsequent transitions
5. Wait 10,000*DfiClkFreq DFI Clocks after last DctWriteProt transition for the test to finish.
6. Follow the normal exit procedure to stop the firmware and read the results.

4.11.3 Pre-test requirements

It is required that the Impedance Calibration and PLL/LCDL Lock tests be run prior to running the Burn-In test. Because of this requirement, the firmware will automatically enable these tests whenever the Burn-In test is enabled. The PassFailResults message block field will include the results for these tests, even if they are not specified in the TestsToRun field.



Because the PLL Lock test is required for the Burn-In test, this test must be run with a DfiClk within the legal range of the PLL to lock.

5

Simulation Firmware

The following sections are included in this chapter:

- [“Simulation Requirements for Firmware”](#) on page 106

5.1 Simulation Requirements for Firmware



Caution

Verilog digital simulation has requirements for the model behavior and the simulation options. Failure to follow these requirements will result in failures in the simulation of the ATE firmware.

The use of the PHY in digital simulation requires that certain analog behaviors be accounted for in the simulation. These behaviors are related to the conversion of Z's and X's in the Verilog simulation into 0's and 1's as would happen in real silicon.

5.1.1 PAD Model Behavior

For the PAD behavior, the simulation cannot force X's on the IO to simulate timing errors or glitches during the testing process. Some examples of behavior that is not allowed is:

- Forcing X's on the DQ lines when transitioning from one value to another to "simulate" the analog transition of one value to another.
 - Modeling glitches between bit time transitions when the data of the consecutive bits is different with extra 0-1-0 transitions is acceptable.

Ideal llllllllllllllrhhhhhhhhhhhhhhhhhhhh

Acceptable llllllllllrfrfrfrhhhhhhhhhhhhhhhhhh

Not acceptable lllllllllljjjjjhhhhhhhhhhhhhhhhhh

- Forcing X's on the DQ/DQS lines when timing parameters are not met to indicate a timing violation.
- Forcing X's or glitches on the DQ lines between data beats of reads when the data value is not changing between the beats. For example, when the read would return 0 on consecutive beats of data the simulation is not allowed to drive an X or 1 for a short time between the beats.

Ideal / Acceptable tttttadddd0dddddd0dddddd0dddddd0ddddddgtttt

Not Acceptable tttttadddd0ddjd0dddddjd0ddjd0dddddgtttt

5.1.2 Hi-Z Bus Modeling

There are times during the ATE firmware 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 test 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 ATE 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.

5.2 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.

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

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


6

Bringing Up the ATE firmware

The following sections are included in this chapter:

- [“Initial running of the ATE firmware in simulation or silicon”](#) on page 110
- [“Bringing-Up Individual Tests”](#) on page 111

Figuring out what is happening when the firmware doesn't pass is a process of understanding how the firmware works. From a general point of view, the firmware uses the following process:

1. Write the UctWriteProt/UctWriteProtShadow CSR to 0x1 to indicate the firmware is running.
2. Assert the MemReset_L pin to hold the memory in reset (if present)
3. Save the CSR values that the firmware will use so they can be restored at the end.
4. Initialize the message block PassFailResults field to an all failing condition.
5. Read the message block to see which tests the user wants to run.
6. Run the tests the user requests. For each test, the general process is:
 - a. Read the message block to retrieve the parameters for this test.
 - b. Clear any previous test results from the message block.
 - c. Use the parameters to set up the conditions for the test.
 - d. Perform the test using the values specified in the message block.
 - e. Determine Pass/Fail.
 - f. Write the Pass/Fail result and any data to the message block. The write to the PassFailResults field only affects the bit for this test.
7. Repeat step 6 for each test in the documented test order.
8. Restore the CSRs to the values previously read in Step 3.
9. Write the UctWriteProt/UctWriteProtShadow CSR to 0x0 to indicate the firmware is finished.
10. De-assert the MemReset_L pin to let the memory come out of reset (if present)
11. Halt/stall the CPU to stop the firmware from running.

6.1 Initial running of the ATE firmware in simulation or silicon

To be sure that the firmware Instruction and Data images are loaded correctly, and the message block is being accessed correctly, the Revision Number check test can be used. This test runs very quickly in simulation and silicon and successfully completing this test will indicate that the firmware is being loaded and started correctly.

**Note**

Passing the Revision Check test does not guarantee that the entire firmware image has been loaded correctly, but that the process successfully loaded the parts of the code and data that this test needs.

To run the Revision Number check test only:

- Follow the firmware loading and running process in Chapter 2, setting the TestsToRun message block field to 0x1 in Step D2. None of the other message block fields are read if only the Revision Check test is run. The other fields can be left at default, or written to the correct values.
- When the firmware indicates that it has finished, read the PassFailResults, AteImemRevision, and AteDmemRevision message block fields.
 - The PassFailResults field should be 0x1
 - The AteImemRevision and AteDmemRevision fields should correspond to the revision used.

**Note**

The revision number for a release can be found in the mnPmuSramMsgBlock_ate.h file that is included with the release in the directories that contain the IMEM and DMEM images. The revision number is indicated as two C define statements, PMU_ATE_INTERNAL_REV1 and PMU_ATE_INTERNAL_REV0. PMU_ATE_INTERNAL_REV1 indicates the upper four digits of the revision number and PMU_ATE_INTERNAL_REV0 indicates the lower four digits.

For issues during bring-up, see the General and Revision Check sections that follow.

6.2 Bringing-Up Individual Tests

The following sections detail the most common issues which may cause individual tests to fail and provides guidance as to how to determine the reason for the failure.

6.2.1 General Failure Causes

If no tests are passing, including the Revision Check test, the most likely causes are:

- Not loading the entire images for the Instruction and Data memories. The entire images must be loaded, including all leading or trailing zeros. This can be checked by reading back the contents of the memories and comparing it to the values in the original images. All bytes must compare correctly.
- Not writing the message block value(s) correctly. This can be checked by reading back the message block locations and comparing against the data that should be there.
- Not following the process elaborated in Chapter 2 to run the firmware correctly. If any of the operations are not done, it can prevent the memory or the CSR registers from being set up correctly or corrupt the register writes that the processor that is running the firmware is doing. Double check the process, including all wait times.
- Not waiting long enough for the firmware to complete successfully. If the UctWriteProt/UctWrite-ProtShadow registers read as 0x1 at the end of the firmware, this is one of the possibilities. (Note: This register may also be a 1 if the firmware never runs and the register was 0x1 already.)

6.2.2 Revision Check Test

The Revision Check test simply compares the revision number stored in the IMEM image with the revision number stored in the DMEM image. If they are equal, the test will pass. If they are different, the test will fail.



Note

The revision number for a release can be found in the `mnPmuSramMsgBlock_ate.h` file that is included with the release in the directories that contain the IMEM and DMEM images. The revision number is indicated as two C define statements, `PMU_ATE_INTERNAL_REV1` and `PMU_ATE_INTERNAL_REV0`. `PMU_ATE_INTERNAL_REV1` indicates the upper four digits of the revision number and `PMU_ATE_INTERNAL_REV0` indicates the lower four digits.

If this test fails, the most common reason is:

- Not loading the correct firmware images. If the IMEM image is used with a different revision DMEM image, that will cause the Revision Check test to fail. Compare the `AteImemRevision` and `AteDmemRevision` values. They should be equal. If they are not equal, the test failed. To correct the issue, use the IMEM and DMEM images from the same release.

6.2.3 Impedance Calibration

The Impedance Calibration runs the impedance calibration process and then reads the calibration results to look for anomalies.

If this test fails, the most common reasons are:

- In simulation, this test is expected to fail unless something is done to the simulation test bench. This test exercises the analog comparator for the calibration resistor. The simulation Verilog model for the resistor and comparator do not fully model the analog behavior. The circuit will run and will return a

value that is the minimum or maximum value for calibration. This is failure condition for the test. If the calibration results in a min or max code, the test will be marked as failing.

- In silicon, the value should settle on a non-minimum or non-maximum value. The test will be marked as failing if any of the calibration results come back as minimum or maximum. Any other values are considered passing. The values for the calibration can be read back from the message block to determine which of the calibration results caused the failure. The values to read in the message block are:
 - ZCalCompResult (minimum of 0x0000, maximum of 0x007F)
 - ZCalCodePU (minimum of 0x0000, maximum of 0x00FF)
 - ZCalCodePD (minimum of 0x0000, maximum of 0x00FF)

6.2.4 PLL Lock / LCDL Lock Test

The PLL Lock test runs the PLL locking process and then checks a set of sticky bits to make sure the PLL lock has asserted, and hasn't deasserted. If the PLL never indicates lock, or lost lock, then the test will be marked as failing.

The LCDL calibration codes checking requires the value of DfiClkFreq to be configured correctly.

The most common reasons that the PLL test fails are:

- The general failure modes elaborated earlier. Double check the firmware process and the Message Block entries, especially the DfiClkFreq value.
- Excessive jitter and phase or frequency variation on the incoming clock can also cause the PLL to not lock correctly.

However, the message blocks outputs PLLResults, LcdlResultsAc, LcdlResultsDb, and LcdlResultsRxReplica must be used to qualify why the test failed.

6.2.5 RxReplica Calibrate Test

The RxReplica Calibrate test checks the RxReplica path-phase values, to make sure they have not saturated to all 1's/0's. However, this test is only supported for data-rates 1600Mbps and above. If the data-rate is 3200Mbps or above, the test also ensures RxReplica correctly calibrates to the 1UI + 2UI boundaries.

The LCDL 1UI lock code and 5 path-phases for each RxReplica are saved to the message block, and can be used to qualify a failure of this test.

6.2.6 LCDL Linearity Test

The LCDL Linearity test runs each LCDL at the minimum delay setting and a mid-delay setting, and gets the oscillator counts for both settings. It uses these values to create a linear line to predict the oscillator counts at each other delay setting that the user has requested. If the value seen for any delay setting for any LCDL is outside the range allowed by the user setting, the test will be marked as failing.

The process to debug a failure is broken into two parts:

1. Determine which LCDLs are failing.
 - a. This is done by reading all the LcdlErrCntAc* and LcdlErrCntDb* Message Block fields. These values will all be 0 for a passing test. Any non-zero value indicates that delay values for that LCDL

did not fall within the linearity bounds. The value indicates how many points fell outside the bounds.

2. Use the LCDL observation capability of the ATE firmware to read out all the count values for the failing LCDLs.
 - a. For each LCDL that had a non-zero count, set up one of the `LcdlObserveCfg[0-3]` message block inputs to get the ATE firmware to save all the count values for that LCDL. Up to 4 LCDLs can be saved per run of the ATE firmware.
 - b. Run the ATE firmware to get all the count values for the failing LCDLs.
 - c. Read out the oscillator count values, and process the values using the equations from the test description section to see which delay values fall outside of the required test ranges.

6.2.6.1 Optimizing LcdlClksToRun

Users may wish to use a `LcdlClksToRun` value which is larger than the suggested value (`LcdlClksToRun=10*DfiClkFreq`), because this will improve the accuracy of the linearity measurements. To find the upper limit on `LcdlClksToRun` for the user's silicon, the user will need to characterize their LCDLs in their design. This is accomplished by fixing the value for `LcdlObserveCfg[4]` while varying the `LcdlClksToRun` value, and analyzing the resulting LCDL edge counts returned in the message block output `LcdlCountValues[4][512]`. A value for `LcdlClksToRun` must be chosen to ensure the edge counts are less than the max ring-oscillator count value of 65,535. Margin is required to account for run-to-run ring-oscillator count variations, which are a result of ring-oscillator control signal synchronization. The variation will be a function of PVT and the selected `DfiClkFreq`.

Once a value for `LcdlClksToRun` has been chosen, it is recommended to examine the edge-count values for every LCDL, to ensure they are all within the required range and include margin. This is accomplished by fixing the value of `LcdlClksToRun`, and sweeping `LcdlObserveCfg` is to cover each LCDL in the PHY. Refer to section 4.6.4 on how to configure `LcdlObserveCfg`.

Users wishing to use a `LcdlClksToRun` less than the suggested value can apply the same procedure described above to find the lower limit on `LcdlClksToRun` for their silicon. During this procedure, all edge counts must be kept above 50.

If one or more LCDL edge counts reach 65535 (0xFFFF) or goes below 50 (0x0032) the LCDL Linearity test may false-fail.

6.2.7 Address / Command Loopback Test

The Address / Command Loopback test places all the AC IO into loopback (either core or pad loopback path based on the Message Block input). It runs PRBS7 data through the SE slice IOs, and a periodic pulse through the DIFF and SEC slice IOs. It increments the delay used for sending the data and determines which delay settings allow the data to be sent and received successfully. It then processes the results to find the size of the EYE (most successive passing values). If the width of the widest region is larger or equal to the required width, the test is marked as passing. If no set of consecutive values is wide enough, the test is marked as failing.

The values of passing and failing delays are saved for each SE slice IO in the `AcLoopbackBitmapSe` fields, `AcLoopbackBitmapDiff` for each DIFF slice IO, and `AcLoopbackBitmapSec` for the SEC slice IO.

To debug why the Address / Command Loopback test is failing, read out all the values and find the lanes that don't meet the minimum width criteria using the explanation of how to interpret the values can be found in the test description section.

Stuck-at testing is also performed on the SE slices, and the results of the testing are stored in `AcLoopbackStuckAtSe`. A value of 0 indicates that an AC SE slice passed stuck-at testing.

6.2.8 Data Loopback 1D Test

The Data Loopback 1D test puts all the Dbyte IO into loopback (either core or pad loopback path based on the Message Block input) and runs PRBS16 data through the SE slice IO, and a periodic 1UI wide pulse through the WCK slice IOs. It increments the delay used for sending the data and determines which delay settings allow the data to be sent and received successfully. It then processes the results to determine the widest passing EYE (most successive passing values). If the width of the widest region is larger or equal to the required width, the test is marked as passing. If no set of consecutive values is wide enough, the test is marked as failing.

The values of passing and failing delays are saved for each SE slice IO in the `DataLoopbackRxEnbVal`, `DataLoopbackCoarse` and `DataLoopbackBitmap` fields, and `DatLoopbackWckBitmap` for each WCK slice IO.

Note, the Dbyte WCK slice is only tested if the PHY is LP5 mode enabled.

To debug why the Data Loopback 1D test is failing, read out all the values and find the lanes that don't meet the minimum width criteria using the explanation of how to interpret the values can be found in the test description section.

6.2.9 Data Loopback 2D Test

Data loopback 2D tests only the SE slices for a range of selected Vrefs. Refer to the Data Loopback 1D test description above.

6.2.10 DCA Loopback Test

The DCA loopback test exercises the DCA (duty cycle adjustment) circuitry in the WCK DIFF slices. It accomplishes this by constructing 1D bitmaps for each tested DCA setting (as specified by the user via the message block). The test is only supported for WCK frequencies of 2.5GHz to 3.2GHz. For each DCA coarse setting, the DCA fine setting is swept from 0 to 12 using the message block input `DcaLoopDcaFineIncr` as the increment. Individual DCA coarse settings can be skipped using the message block input `DcaLoopDcaCoarseSkip`. For the test to report PASS, the sum of changes in the EYE width between tested DCA fine settings for a given DCA coarse setting must be greater than 0.

This test is only supported for PHYs which include LP5 support, and whose PUB version is 1.02a or above.

6.2.11 Burn-In Test

This test is only to exercise the design for Burn-In and will always pass when run.

7

ATE Message Block Definition

7.1 ATE Message Block definition

**Note**

The message block can be seen in the release in the “mnPmuSramMsgBlock_ate.h” file that can be found in the release directory at:
`/firmware/<version>/ate/ mnPmuSramMsgBlock_ate.h`