Versal ACAP Soft RLDRAM 3 Memory Controller v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG354 (v1.0) January 21, 2021

XILINX®

Table of Contents

XILINX

Chapter 1

Introduction

The Xilinx® Versal™ adaptive compute acceleration platform (ACAP) Memory IP core is a combined pre-engineered controller and physical layer (PHY) for interfacing Versal ACAP user designs to RLDRAM 3 devices.

Features

- Component support for interface widths of 18, 36, and 72 bits

  Table 1: **Supported Configurations**

Interface Width	Burst Length	Number of Device
36	BL2, BL4	1, 2
18	BL2, BL4, BL8	1, 2
36 with address multiplexing	BL2, BL4	1, 2
18 with address multiplexing	BL2, BL4, BL8	1, 2

- ODT support
- Memory device support with 576 Mb and 1.125 Gb densities
- RLDRAM 3 initialization support
- Source code delivery in Verilog
- 4:1 memory to Versal ACAP logic interface clock ratio
- Interface calibration and training information available through the Vivado® hardware manager

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal™ ACAP
Supported User Interfaces	Native
Resources	N/A
Provided with Core	
Design Files	RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation[3]	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.
3. Behavioral simulations are supported with Mixed Simulator Language. Netlist (post-synthesis and post-implementation) simulations are supported with Verilog Simulator Language and are not supported by Vivado Simulator.

XILINX

Chapter 2

Overview

Introduction to Versal ACAP

Versal™ adaptive compute acceleration platforms (ACAPs) combine Scalar Engines, Adaptable Engines, and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Most importantly, Versal ACAP hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. Versal ACAPs are enabled by a host of tools, software, libraries, IP, middleware, and frameworks to enable all industry-standard design flows.

Built on the TSMC 7 nm FinFET process technology, the Versal portfolio is the first platform to combine software programmability and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation. The portfolio includes six series of devices uniquely architected to deliver scalability and AI inference capabilities for a host of applications across different markets—from cloud—to networking—to wireless communications—to edge computing and endpoints.

The Versal architecture combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. Intelligent Engines are SIMD VLIW AI Engines for adaptive inference and advanced signal processing compute, and DSP Engines for fixed point, floating point, and complex MAC operations. Adaptable Engines are a combination of programmable logic blocks and memory, architected for high-compute density. Scalar Engines, including Arm® Cortex™-A72 and Cortex-R5F processors, allow for intensive compute tasks.

The Versal AI Core series delivers breakthrough AI inference acceleration with AI Engines that deliver over 100x greater compute performance than current server-class of CPUs. This series is designed for a breadth of applications, including cloud for dynamic workloads and network for massive bandwidth, all while delivering advanced safety and security features. AI and data scientists, as well as software and hardware developers, can all take advantage of the high-compute density to accelerate the performance of any application.

The Versal Prime series is the foundation and the mid-range of the Versal platform, serving the broadest range of uses across multiple markets. These applications include 100G to 200G networking equipment, network and storage acceleration in the Data Center, communications test equipment, broadcast, and aerospace & defense. The series integrates mainstream 58G transceivers and optimized I/O and DDR connectivity, achieving low-latency acceleration and performance across diverse workloads.

The Versal Premium series provides breakthrough heterogeneous integration, very high-performance compute, connectivity, and security in an adaptable platform with a minimized power and area footprint. The series is designed to exceed the demands of high-bandwidth, compute-intensive applications in wired communications, data center, test & measurement, and other applications. Versal Premium series ACAPs include 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

The Versal architecture documentation suite is available at: https://www.xilinx.com/versal.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Chapter 4: Core Architecture
  - Chapter 5: Designing with the Core
  - Chapter 6: Design Flow Steps
  - Chapter 7: Example Design

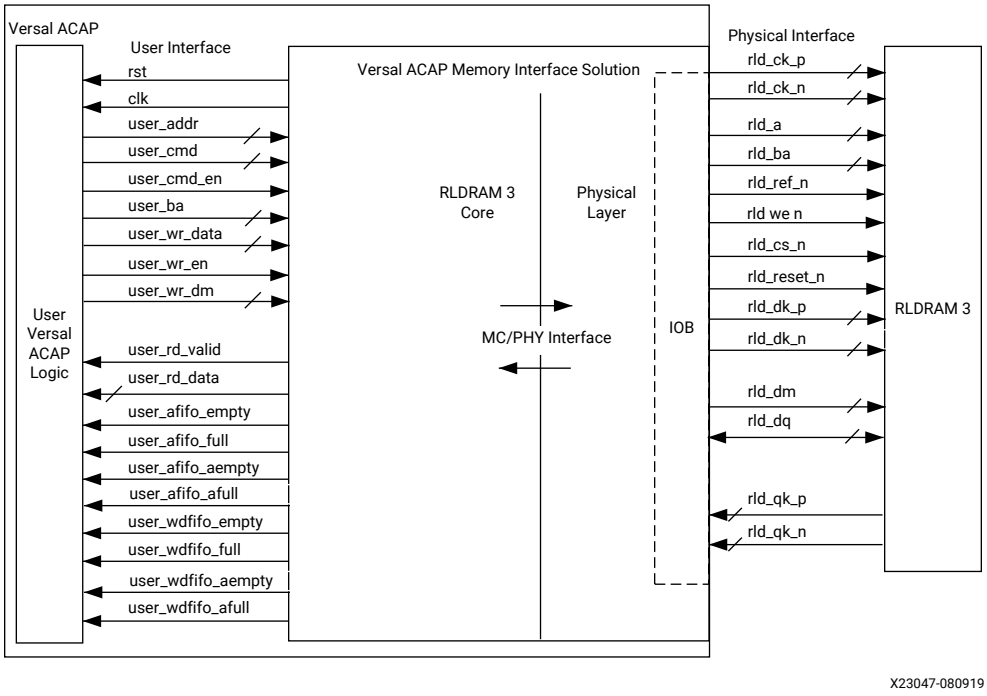
Core Overview

The RLDRAM 3 core provides solutions for interfacing with the DRAM memory type. The Versal ACAP for the RLDRAM 3 core are organized in the following high-level blocks:

- **Controller:** The controller accepts burst transactions from the user interface and generates transactions to and from the RLDRAM 3. The controller takes care of the DRAM timing parameters and refresh.
- **Physical Layer:** The physical layer provides a high-speed interface to the DRAM. This layer includes the hard blocks inside the Versal ACAP and the soft calibration logic blocks necessary to ensure optimal timing of the hard blocks interfacing to the DRAM.The new hard blocks in the Versal ACAP allow interface rates of up to 2,400 Mb/s to be achieved. These hard blocks include:
  - Data serialization and transmission
  - Data capture and deserialization
  - High-speed clock generation and synchronization
  - Fine delay elements per pin with voltage and temperature tracking

  The soft blocks include:
  - **Memory Initialization:** The calibration modules provide an initialization routine for RLDRAM 3. The delays in the initialization process are bypassed to speed up simulation time.
  - **Calibration:** The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance.Results of the calibration process are available through the Xilinx® debug tools. After completion of calibration, the PHY layer presents raw interface to the DRAM.
- **Application Interface:** The user interface layer provides a simple FIFO-like interface to the application. Data is buffered and read data is presented in request order.

Figure 1: **Versal ACAP Memory Interface Solution**



Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Note: To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the Vivado® Design Suite; Purchase means that you have to purchase a license to use the core.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

This core adheres to the Micron® RLDRAM 3 specification.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956)
- *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)

Port Descriptions

There are three port categories at the top-level of the memory interface core called the “user design.”

- The first category is the memory interface signals that directly interfaces with the RLDRAM. These are defined by the Micron® RLDRAM 3 specification.
- The second category is the application interface signals which are referred to as the "user interface." These are described in the Protocol Description section.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High `init_calib_complete` signal indicates that the initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

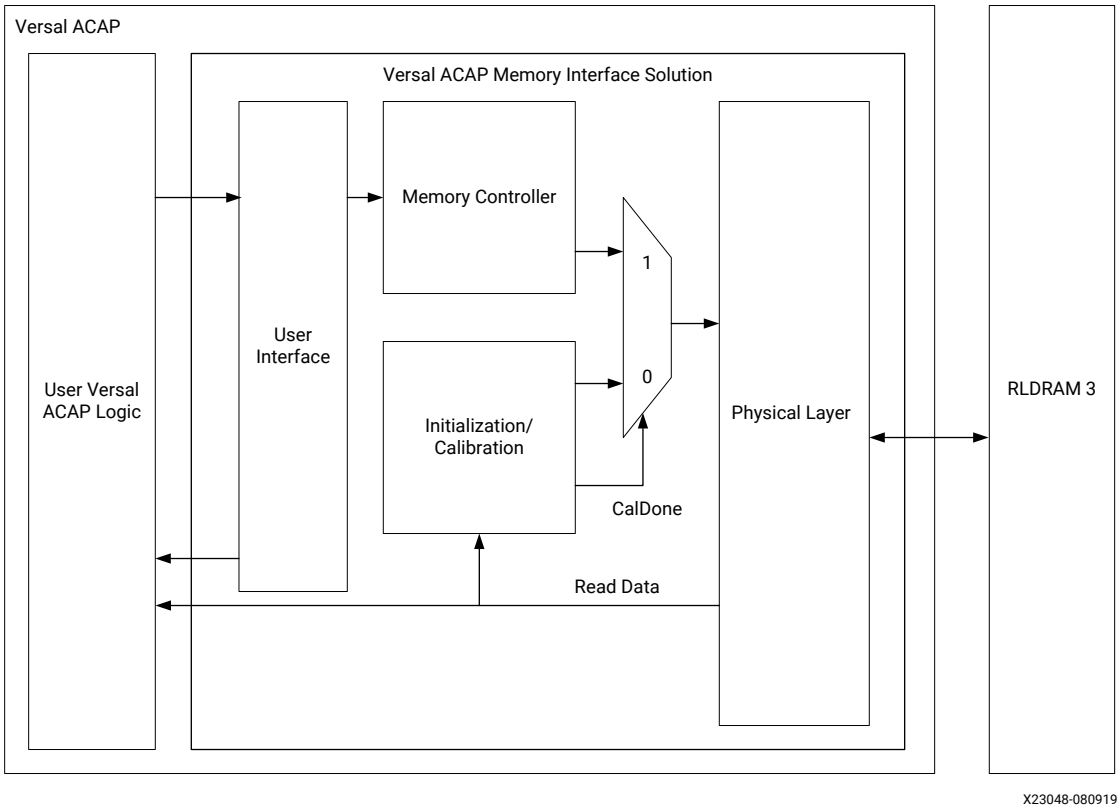
Related Information

Protocol Description

Core Architecture

This section describes the Xilinx® Versal™ adaptive compute acceleration platform (ACAP) Memory Interface Solutions core with an overview of the modules and interfaces. The core is shown below.

Figure 2: **Versal ACAP Memory Interface Solution Core**



The user interface uses a simple protocol based entirely on SDR signals to make read and write requests. See the User Interface section for more details describing this protocol.

The Memory Controller takes commands from the user interface and adheres to the protocol requirements of the RLDRAM 3 device. See the Memory Controller section for more details.

The physical interface generates the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to the RLDRAM 3 protocol and timing requirements. See the Physical Interface section for more details.

Related Information

Memory Interface
User Interface
Physical Interface

Memory Controller

The Memory Controller (MC) enforces the RLDRAM 3 access requirements like the Row Cycle time (tRC) and Write to Read time (tWTR) while keeping the throughput as high as possible. The Memory Controller also interfaces with the PHY.

Commands are reordered in the command queue by the controller based on the Row Cycle time (tRC) of the commands. For example, a command whose tRC is met is scheduled before a command whose tRC is not met thereby improving throughput of the interface. Another command reorder module reorders the position of the write and read commands in a single user cycle. This feature is mostly useful for BL2. With BL2, four commands are issued in a single user clock cycle. For example, if the four commands are Write, Read, Write, and Read with the last command of the previous user clock as Write, then the new arrangement of commands will be Read, Read, Write, and Write.

The MC features a read command bypass mode which is supported when Data Mask (DM) is disabled by setting the top-level parameter DM_EN to 0. This feature improves throughput by bypassing a read command that follows a write command to the same address. In this scenario, the write data associated with the write command can be driven to the `user_rd_data` port thereby saving tWTR (WL + BL / 2) wait time and the read command execution time. The MC command execution sequence is different from the command request order in the user interface. Therefore, the read response is reordered to match the command request order at the user interface.

Auto-refresh commands are inserted into the command flow by the MC to meet the memory device refresh requirements. CMD_PER_CLK is a top-level parameter used to determine how many memory commands are provided to the MC per Versal ACAP logic clock cycle. The number of memory commands per Versal ACAP logic clock cycle depends on the burst length. For example, the CMD_PER_CLK is set to 1 for burst length = 8, CMD_PER_CLK is set to 2 for burst length = 4, and CMD_PER_CLK is set to 4 for burst length = 2.

PHY

The PHY is considered the low-level physical interface to an external RLDRAM 3 device as well as all calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

Overall PHY Architecture

The Versal ACAP PHY is composed of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high performance physical layers.

The MC and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is divided by 4. A more detailed block diagram of the PHY design is shown in the Core Architecture.

The MC is designed to separate out the command processing from the low-level PHY requirements to ensure a clean separation between the controller and physical layer. The command processing can be replaced with custom logic if desired, while the logic for interacting with the PHY stays the same and can still be used by the calibration logic.

Table 2: **PHY Modules**

Module Name	Description
rld3_pl_0.sv	RLDRAM 3 top module
rld3_pl_0_rld3_mem_intfc.sv	This is the module where the PHY, controller, and calibration modules are instantiated. The modules which correct the bitslips and address pipe modules are also instantiated here.
rld3_pl_0_phy.sv	This is the top module of the XPHY where the XPLL and bank wrapper modules are instantiated.
rld3_pl_phy_v1_0_pll.sv	PLL module where the XPLLs are instantiated.
rld3_pl_phy_v1_0_xphy_bank_wrapper.sv	Bank wrapper module which contains the instantiation of the XPHY banks.
rld3_pl_phy_v1_0_xphy_bank.sv	XPHY bank module
rld3_pl_v1_0_cmd_addr_pi.sv	This module adds the necessary delay on to the address and control signals that is going to the memory interface to match the latency and calibration requirements.

Table 2: **PHY Modules** *(cont'd)*

Module Name	Description
rld3_pl_v1_0_data_pi.sv	This module adds the necessary delay on the data and dm that is going to the memory interface to match the latency and calibration requirements.
rld3_pl_v1_0_cal_top.sv	Calibration top module
rld3_pl_v1_0_cal_addr_decode.sv	This module interacts with C code to do the necessary calibration related operations. This contains the necessary register space for calibration.

The PHY architecture encompasses all of the logic contained in `rld_xphy.sv`. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. For more information on the hard silicon physical layer architecture, see the *Versal ACAP SelectIO Resources Architecture Manual* (AM010).

Related Information

Core Architecture

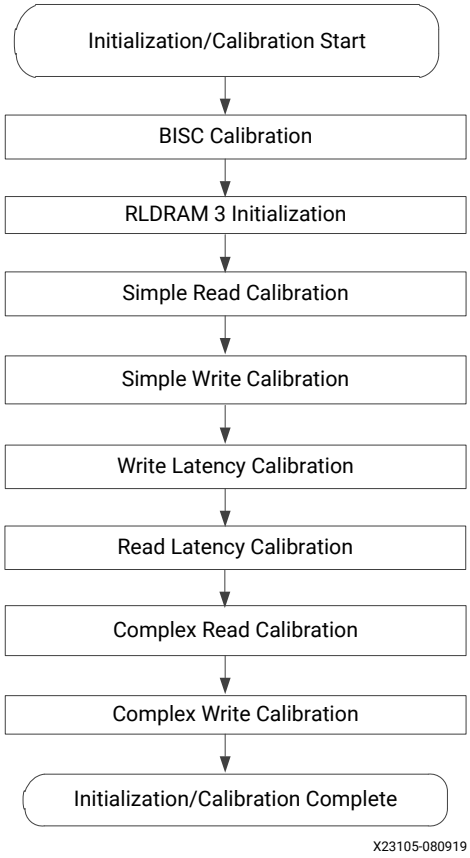
Memory Initialization and Calibration Sequence

After deassertion of the system reset, the PHY performs some required internal calibration steps first.

1. The built-in self-check (BISC) of the PHY is run. BISC is used in the PHY to compute internal skews for use in voltage and temperature tracking after calibration is completed.
2. After BISC is completed, calibration logic performs the required power-on initialization sequence for the memory.
3. When both routines are finished, the control is transferred to MicroBlaze™, which is a soft processor that calibrates the timing of the write and read data paths.
4. After calibration is completed, the PHY calculates internal offsets to be used in voltage and temperature tracking.

The following figure shows the overall flow of memory initialization and the different stages of calibration.

Figure 3: **PHY Overall Initialization and Calibration Sequence**



When simulating the RLDRAM 3 example design, the calibration process is bypassed to allow for quick traffic generation to and from the RLDRAM 3 device. Calibration is always enabled when running the example design in hardware. The hardware manager GUI provides information on the status of each calibration step or description of error in case of calibration failure.

If the hardware manager GUI is not used, the first step in determining the calibration status is to check the status of `init_calib_complete` and `calib_error` signals. The `init_calib_complete` only asserts if calibration passes successfully, otherwise `calib_error` is asserted. Calibration halts on the very first error encountered. There are three status registers, cal_passed_stages, cal_failed_stages, and cal_skipped_stages that provide information on the successfully completed calibration stages, the failed stages, and the skipped stages, respectively.

Calibration Status Register Signal Descriptions

Table 3: **Calibration Status Register Signal Descriptions**

Status Register	Bits	Description
cal_passed_stages[31:0]	0	PHY ready asserted
	1	RLDRAM 3 Initialization completed
	2	Simple Read Per bit De-skew/Calibration completed
	3	Simple Write Per bit De-skew/Calibration completed
	4	Write Latency Calibration completed
	5	Read Latency Calibration completed
	6	Complex Read Calibration completed
	7	Complex Write Calibration completed
	[31:8]	Reserved
cal_failed_stages[31:0]	0	PHY ready failed to assert
	1	RLDRAM 3 Initialization failed to complete
	2	Simple Read Per bit De-skew/Calibration failed to complete
	3	Simple Write Per bit De-skew/Calibration failed to complete
	4	Write Latency Calibration failed to complete
	5	Read Latency Calibration failed to complete
	6	Complex Read Calibration failed to complete
	7	Complex Write Calibration failed to complete
	[31:8]	Reserved
cal_skipped_stages[31:0]	0	Always set to 0 because PHY ready is never skipped
	1	Always set to 0 because RLDRAM 3 Initialization is never skipped
	2	Simple Read Per bit De-skew/Calibration skipped (set to 1) below 800 MHz
	3	Simple Write Per bit De-skew/Calibration skipped (set to 1) below 800 MHz
	4	Always set to 0 because Write Latency Calibration is never skipped
	5	Always set to 0 because Read Latency Calibration is never skipped
	6	Complex Read Calibration skipped (set to 1) below 800 MHz
	7	Complex Write Calibration skipped (set to 1) below 800 MHz
	[31:8]	Reserved

Simple Read Calibration

To maximize the data eye and center the internal read sampling clock in the read DQ window for robust sampling, the following steps are performed:

1. Per bit deskew of the DQ bus to maximize the data eye by removing skew and On-Chip Variation effects.

2. Sweep internal read clock and its inverse across all DQ bits to find the center of the rise and fall data eyes.
3. The read clock, QK is delayed using the PQTR and NQTR delay taps to generate the internal read clock and its inverse, respectively.
4. For this stage of calibration, a pre-defined `1010...` pattern is read from Mode Register 2 in the RLDRAM 3 device because the write path has not yet been calibrated. The per bit deskew step begins with simultaneously incrementing PQTR and NQTR until the internal read clocks and their inverse are in the valid data region.
5. IDELAY taps for all DQ bits are incremented until invalid data is sampled on each bit thereby deskewing DQ bits with its associated read clock and its inverse.
6. Read clocks and their inverse are moved using PQTR and NQTR delay taps to detect the valid window edges and then centered in the valid window.

Simple Write Calibration

To maximize the data eye and center the write clock in the write DQ window for robust sampling, the following sequential steps are performed:

1. Move DK using ODELAY taps until it samples valid data on all corresponding DQ bits for 15 consecutive taps. Restore DK to its original position by removing the 15 ODELAY taps.

   Note: DK has a 90° phase offset with respect to DQ and DM bits.
2. Move DQ bits using ODELAY taps until DK samples valid data.
3. Move DQ bits until DK samples invalid data thereby deskewing all DQ bits with respect to its associated DK.
4. Continue incrementing DQ ODELAY taps to detect the second edge with inverted data pattern.
5. Set DQ taps to average of both edges to center align DQ with respect to DK

Write Latency Calibration

1. In this stage, a single write command with a write latency of 0 is issued to different banks with a data pattern: `0xffffffff`, `0x00000000`, `0x00000000`, `0x00000000`, and `0xffffffff`.
2. Reads are continuously issued to locations where zeros are written. Because the read latency calibration has not been performed, read latency is set to a large value so that data is properly read.
3. An incorrect write latency results in 1s in the read data. Write latency is incremented by 1 until the read data is all 0s.
4. Repeat the same steps with the write pattern: `0x00000000`, `0xffffffff`, `0xffffffff`, `0xffffffff`, and `0x00000000`.
5. When read data is all 1s, this stage is complete.

Read Latency Calibration

Read latency calibration involves two steps. The first step determines the read bitslip value and the second step determines the read latency value.

1. The read bitslip value is set to zero and the read latency is set to a large value so that read data is properly read. A unique write data pattern is written to different bank addresses. These addresses are read back continuously. Adjust the bitslip of each nibble until the read data matches the expected pattern.
2. Once the correct read bitslip is determined, set the read latency value to zero. Read once from the same address and compare with expected pattern. If not matched, then increase the read latency by 1. Repeat until read data matches expected data thereby completing read latency calibration.

Complex Read Calibration

In this stage, bursts of complex pattern are written to the memory and read back. The comparison results of the read data with the expected data is recorded in a cal comparison status register. The rise and fall nibble wise comparison results are available to the calibration algorithm. Based on these comparison results the algorithm decides on which taps to manipulate.

The complex read calibration steps are listed as follows:

1. Revert PQTR and NQTR delays to 0 for all read clocks and its inverse clocks.
2. Check whether the read clock and its inverse clock samples the noise region of the data.
3. If either of the clocks samples the valid region of the data, then DQ and DM IDELAY taps are incremented together until both clocks start sampling the noise region.
4. The read clock and its inverse are moved using PQTR and NQTR taps to traverse the data window from noise to valid region and then valid region to noise. The delays taps difference between the noise to valid crossing point and the valid to noise crossing point is called the valid window. The read clock and its inverse is centered in their respective valid windows thereby completing complex read calibration.

Complex Write Calibration

DM mode must be enabled for DM calibration. The DM mode is assumed to be enabled for this description. During this stage, DK does not move, only DQ bits and DM associated with a DK are incremented simultaneously.

1. Revert DQ and DM ODELAY taps to deskew taps so all the DQ bits and DM are edge-aligned. This might result in DK ending up near or in the right noise region.
2. Increment DQ/DM ODELAY taps one tap at a time until valid data detected for 15 fine taps. The right edge is recorded as current tap value – 15.

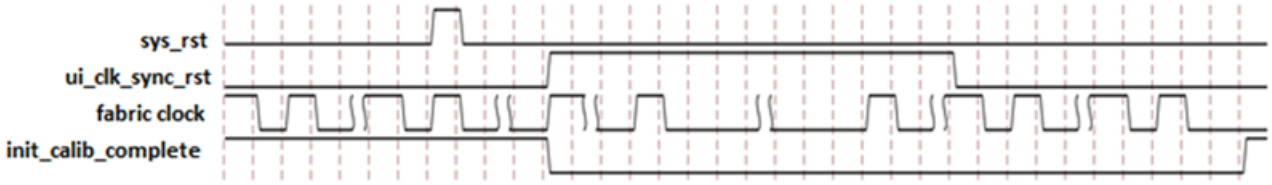
3. Increment DQ/DM ODELAY taps 10 taps at a time until left noise region is detected. Then decrement one fine tap at a time until valid data detected for 15 fine taps. Left edge is recorded as current tap – 15.
4. Set DQ/DM ODELAY tap to average of right and left edges.

Reset Sequence

The `sys_rst` signal resets the entire memory interface design which includes general interconnect (fabric) logic, RIU interface logic, MicroBlaze, and calibration logic. The `sys_rst` input signal is synchronized internally to create the `ui_clk_sync_rst` signal. The `ui_clk_sync_rst` reset signal is synchronously asserted and synchronously deasserted.

The following figure shows the `ui_clk_sync_rst` signal (fabric reset) is synchronously asserted with a few clock delays after the `sys_rst` signal is asserted. When the `ui_clk_sync_rst` signal is asserted, there are a few clocks before the clocks are shut off.

Figure 4: **Reset Sequence Waveform**



The following are the reset sequencing steps:

1. Reset to design is initiated after the `ui_clk_sync_rst` signal goes High.
2. The `init_calib_complete` signal goes Low when the `ui_clk_sync_rst` signal is High.
3. Reset to design is deactivated after the `ui_clk_sync_rst` signal is Low.
4. After the `ui_clk_sync_rst` signal is deactivated, the `init_calib_complete` signal is asserted after calibration is completed.

MicroBlaze MCS ECC

The MicroBlaze MCS local memory provides an option to enable Error Correcting Code (ECC). Error correction corrects single bit errors and detects double bit errors. Two additional ports are added to indicate single bit errors (LMB_CE) and double bit errors (LMB_UE).

The MicroBlaze MCS ECC can be selected from the **MicroBlaze MCS ECC option** section in the **Advanced Options** tab. The block RAM size increases if the ECC option for MicroBlaze MCS is selected.

Related Information

Advanced Options Tab

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one XPLL per I/O bank used by the memory interface and BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two XPLLs per bank. If a bank is shared by two memory interfaces, both XPLLs in that bank are used.

Note: RLDRAM 3 generates the appropriate clocking structure and no modifications to the RTL are supported.

The RLDRAM 3 IP generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to XPLL (located in center bank of memory interface)
- XPLL to BUFG (located at center bank of memory interface) driving the Xilinx® Versal™ adaptive compute acceleration platform (ACAP) logic and all XPLLs
- XPLL to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate Versal ACAP logic

Requirements

GCIO

- Must use a differential I/O standard
- Must be placed anywhere within the two or three banks of the interface
- The I/O standard and termination scheme are system dependent. For more information, consult the *Versal ACAP SelectIO Resources Architecture Manual* (AM010).

Input Clock Requirement

- Clock input period jitter must be ≤ 3 ps RMS.
- The input clock should always be clean and stable. The IP functionality is not guaranteed if this input system clock has a glitch, discontinuous, etc.

BUFGs and Clock Roots

BUFGs and clock roots must be located in one of the banks of the memory interface.

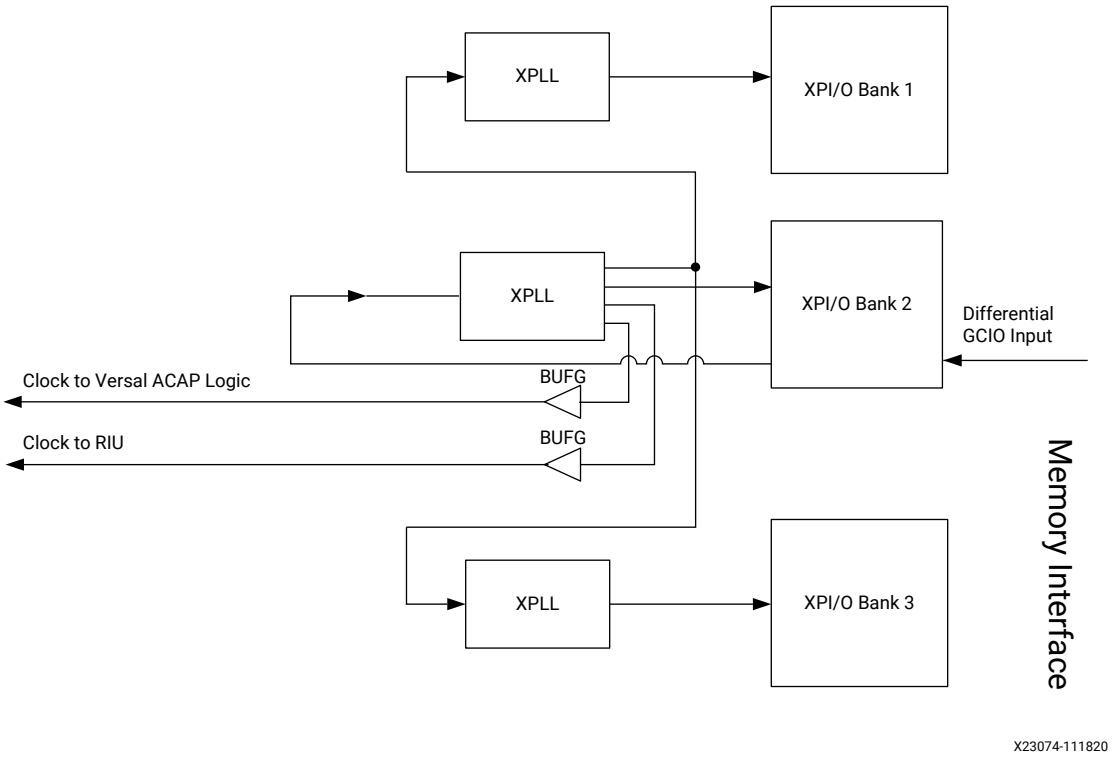
XPLL

- XPLL is used to generate the Versal ACAP logic system clock (1/4 of the memory clock)
- Must be located in the center bank of memory interface
- CLKOUTPHY from XPLL drives XPHY within its bank
- Must use internal feedback

Note: XPLLs do not have fractional clock generation.

The following figure shows an example of the clocking structure for a three bank memory interface. The GCIO drives the XPLL located at the center bank of the memory interface. The XPLL in the center bank drives the inputs of the XPLLs in each of the adjacent banks.

Figure 5: **Clocking Structure for Three Bank Memory Interface**



XPLL Usage

There are two XPLLs per bank. One XPLL per bank is needed for the soft memory interface and the total number of XPLLs depends on number of banks required for the interface. The following shows the XPLL allocation/sharing rules:

- Any of the two XPLLs on a bank can be used for soft memory interface if that bank is not used by the integrated DDR MC.
- If the same bank is shared by the integrated DDR MC and the soft memory interface, XPLL-0 must be used for the integrated DDR MC and XPLL-1 must be used for the soft memory interface.
- When the same bank is shared by the soft memory interface and any IP other than the integrated DDR MC, any of the two XPLLs can be used for soft memory interface.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-High reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

For more information on reset, see the Reset Sequence and Core Architecture sections.

Related Information

Reset Sequence
Core Architecture

PCB Guidelines

Strict adherence to all documented RLDRAM 3 PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *Versal ACAP PCB Design User Guide* (UG863).

Pin and Bank Rules

The rules are for single-rank memory interfaces.

IMPORTANT! *The Versal ACAP soft memory IP can only use XPIO pins that are fully fabric accessible. Refer to the package file to determine XPIO pins that are fully fabric accessible. The XPIO pins/banks in a Versal device that are not fabric accessible are called shadow pins/banks.*

- Address/control means `cs_n`, `ref_n`, `we_n`, `ba`, `ck`, `reset_n`, and `a`.
- Each bank has nine nibbles numbered 0 to 8. Two consecutive nibbles in a bank are paired to form a nibble pair. For example, 0-1, 2-3, 4-5, and 6-7 are the four nibble pairs in a bank.
- Pins in a nibble are numbered 0 to 5. For example in the package pin name IO_L0N_XCC_N0P1_M0P1_700, N0P1 indicates pin 1 of nibble 0 in bank 700.

Note: There are two XPLLs per bank and one XPLL is required in every bank that is being used for the memory interface.

1. For 1x18 pin rules:
   a. All signals for this interface must be placed in two consecutive banks.
   b. All data associated with a Write clock pair (`dk_p/n`) must be placed in the same bank.
   c. A data group comprising of nine `dq` bits, one Read clock pair (`qk_p/n`), and one DM must be placed in two consecutive nibbles. The Read clock pair (`qk_p/n`) can be placed in any one of the two nibbles.
   d. DQ bits must only be placed in nibbles 2, 3, 4, and 5.
   e. `qk_p/n` must be assigned to pin pair 0/1 in a nibble.

f. Write clock pairs `dk_p/n[0]` and `dk_p/n[1]` and memory clock pair (`ck_p/n`) must be placed in nibble 8.

g. Memory clock pair (`ck_p/n`) must be placed in pin pair 2/3 of nibble 8.

h. Write clock pairs `dk_p/n[*]` must be placed in pin pairs closest to the adjacent nibble with its associated data group as shown in the 1x18 pinout example.

i. Address and command pin rules:

- Skipping nibbles between address and data groups is not allowed.
- Free nibbles in the data bank and nibbles 0 or 7 of the adjacent bank can be used for Address/Control. If the adjacent bank is on right side of the data bank, nibble 0 must be used else nibble 7 must be used for address/control as shown in the 1x18 pinout example.

2. For 2x18 pin rules:

a. All signals for this interface must be placed in two consecutive banks.

b. All data associated with a Read clock pair (`qk_p/n`) must be placed in the same bank.

c. A data group comprising of nine `dq` bits, one Read clock pair (`qk_p/n`), and one DM should be placed in two consecutive nibbles.

d. The Read clock pair (`qk_p/n`) must be assigned to pin pair 0/1 in any one of the two nibbles.

e. Write clock pairs `dk_p/n[*]` and memory clock pair (`ck_p/n`) must be placed in three contiguous nibbles in the Address/control bank as shown in the 2x18 example pinout.

f. Memory clock pair (`ck_p/n`) must be placed in pin pair 2/3 of nibble 8.

g. No other signals or free pins allowed between `ck_p/n` and `dk_p/n` pairs.

h. All Address/control signals must be in a single bank.

3. For 1x36 pin rules:

a. All signals for this interface must be placed in two consecutive banks.

b. A data group comprising of nine DQ bits, and one Read clock pair (`qk_p/n`) must be placed in two consecutive nibbles. The Read clock pair (`qk_p/n`) must be placed on pin pairs 0/1 in any one of the two nibbles.

c. All DQ bits must be placed in a single bank.

d. 18 bits data group comprising of 18 DQ bits and one DM must be placed in four consecutive nibbles. DM can be placed in any of the four nibbles.

e. Write clock pairs `dk_p/n[0]` and `dk_p/n[1]` and memory clock pair (`ck_p/n`) must be placed in nibble 8.

f. Memory clock pair (`ck_p/n`) must be placed in pin pair 2/3 of nibble 8.

g. Write clock pairs `dk_p/n[*]` must be placed in pin pairs closest to the adjacent nibble with its associated data group as shown in the 1x18 pinout example.

h. Address and command pin rules:
   - Skipping nibbles between address and data groups is not allowed.
   - Address/Control must be placed in five consecutive nibbles in a bank adjacent to the data bank. If the adjacent bank is on right side of the data bank, nibbles 0, 1, 2, 3, and 8 must be used else nibbles 8, 4, 5, 6, and 7 must be used for address/control as shown in the 1x36 example pinout.

4. For 2x36 pin rules:
   a. All signals for this interface must fit in three consecutive banks.
   b. A data group comprising of nine DQ bits, and one Read clock pair (`qk_p/n`) must be placed in two consecutive nibbles. The Read clock pair (`qk_p/n`) must be placed on pin pairs 0/1 in any one of the two nibbles.
   c. All DQ bits must only be placed in the first and third banks, not the middle bank.
   d. 18 bits data group comprising of 18 DQ bits and one DM must be placed in four consecutive nibbles. DM can be placed in any of the four nibbles.
   e. Memory clock pair (`ck_p/n`) must be placed in pin pair 2/3 of nibble 8 in the middle bank.
   f. Write clock pairs `dk_p/n[*]` must be placed in pin pairs closest to the adjacent nibble with its associated data group as shown in the 2x36 example pinout.
   g. No other signals or free pins allowed between `ck_p/n` and `dk_p/n` pairs.
   h. All Address/control signals must be in the middle bank.
5. The IO_VP pin is an additional bank pin that is used as a reference to calibrate internal on-die termination (DCI). This pin must be externally connected to a 240Ω resistor on the PCB and pulled up to the bank VCCO voltage. DCI is required for this interface. All rules for the DCI in the *Versal ACAP SelectIO Resources Architecture Manual* (AM010) must be followed.
6. The `reset_n` can be placed on any available pin within the banks used by the interface.
7. Banks can be shared between two controllers.
   - Each byte lane is dedicated to a specific controller (except for `reset_n`).
   - Byte lanes from one controller cannot be placed inside the other. For example, with controllers A and B, “AABB” is allowed, while “ABAB” is not.
8. All I/O banks used by the memory interface must be in the same SLR of the column for the SSI technology devices.
9. Maximum height of interface is three contiguous banks for 72-bit wide interface.
10. Bank skipping is not allowed.
11. The input clock for the XPLL in the interface must come from the a GCIO pair within the banks used for the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements should be considered). For more information, see the Clocking section.

12. Versal ACAP XPIO only supports internally generated V_{REF}. See the *Versal ACAP SelectIO Resources Architecture Manual* (AM010) for details.
13. RLDRAM 3 pins not mentioned in the cited pin rules (JTAG, MF, etc.) or ones that you choose not to use in your design must be connected as per Micron® RLDRAM 3 data sheet specification.
14. The system reset pin (`sys_rst_n`) can be placed on any pin within the banks used for the memory interface.

Related Information

Clocking

Pin Swapping

- Pins can swap freely within each nibble pairs (data and address/control) (for more information, see the Pin and Bank Rules section).
- Nibble pairs (data and address/control) can swap easily with each other.
- Pins in the address/control nibble pairs can swap freely within and between their byte groups.
- No other pin swapping is permitted.

Related Information

Pin and Bank Rules

RLDRAM 3 Pinout Examples

IMPORTANT! *Due to the calibration stage, `set_input_delay`/`set_output_delay` on the RLDRAM 3 is not necessary. Ignore the unconstrained inputs and outputs for RLDRAM 3 and the signals which are calibrated.*

1x18 Pinout

The following table shows an example of a 1x18 RLDRAM 3 interface contained within two banks.

Table 4: **1x18 Pinout**

x18 Bank 1			x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP0	IO_L0P_XCC_N0P0	A0	MxP54	IO_L0P_XCC_N0P0	REFn
MxP1	IO_L0N_XCC_N0P1	A1	MxP55	IO_L0N_XCC_N0P1	RESETn

Table 4: **1x18 Pinout** *(cont'd)*

x18 Bank 1			x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP2	IO_L1P_N0P2	A2	MxP56	IO_L1P_N0P2	BA0
MxP3	IO_L1N_N0P3	A3	MxP57	IO_L1N_N0P3	BA1
MxP4	IO_L2P_N0P4	A4	MxP58	IO_L2P_N0P4	BA2
MxP5	IO_L2N_N0P5	A5	MxP59	IO_L2N_N0P5	BA3
MxP6	IO_L3P_XCC_N1P0	A6	MxP60	IO_L3P_XCC_N1P0	–
MxP7	IO_L3N_XCC_N1P1	A7	MxP61	IO_L3N_XCC_N1P1	–
MxP8	IO_L4P_N1P2	A8	MxP62	IO_L4P_N1P2	–
MxP9	IO_L4N_N1P3	A9	MxP63	IO_L4N_N1P3	–
MxP10	IO_L5P_N1P4	A10	MxP64	IO_L5P_N1P4	–
MxP11	IO_L5N_N1P5	A11	MxP65	IO_L5N_N1P5	–
MxP12	IO_L6P_GC_XCC_N2P0	DQ0	MxP66	IO_L6P_GC_XCC_N2P0	–
MxP13	IO_L6N_GC_XCC_N2P1	DQ1	MxP67	IO_L6N_GC_XCC_N2P1	–
MxP14	IO_L7P_N2P2	DQ2	MxP68	IO_L7P_N2P2	–
MxP15	IO_L7N_N2P3	DQ3	MxP69	IO_L7N_N2P3	–
MxP16	IO_L8P_N2P4	DQ4	MxP70	IO_L8P_N2P4	–
MxP17	IO_L8N_N2P5	DQ5	MxP71	IO_L8N_N2P5	–
MxP18	IO_L9P_GC_XCC_N3P0	QK0p	MxP72	IO_L9P_GC_XCC_N3P0	–
MxP19	IO_L9N_GC_XCC_N3P1	QK0n	MxP73	IO_L9N_GC_XCC_N3P1	–
MxP20	IO_L10P_N3P2	DQ6	MxP74	IO_L10P_N3P2	–
MxP21	IO_L10N_N3P3	DQ7	MxP75	IO_L10N_N3P3	–
MxP22	IO_L11P_N3P4	DQ8	MxP76	IO_L11P_N3P4	–
MxP23	IO_L11N_N3P5	DM0	MxP77	IO_L11N_N3P5	–
MxP24	IO_L12P_GC_XCC_N4P0	DQ9	MxP78	IO_L12P_GC_XCC_N4P0	–
MxP25	IO_L12N_GC_XCC_N4P1	DQ10	MxP79	IO_L12N_GC_XCC_N4P1	–
MxP26	IO_L13P_N4P2	DQ11	MxP80	IO_L13P_N4P2	–
MxP27	IO_L13N_N4P3	DQ12	MxP81	IO_L13N_N4P3	–
MxP28	IO_L14P_N4P4	DQ13	MxP82	IO_L14P_N4P4	–
MxP29	IO_L14N_N4P5	DQ14	MxP83	IO_L14N_N4P5	–
MxP30	IO_L15P_XCC_N5P0	QK1p	MxP84	IO_L15P_XCC_N5P0	–
MxP31	IO_L15N_XCC_N5P1	QK1n	MxP85	IO_L15N_XCC_N5P1	–
MxP32	IO_L16P_N5P2	DQ15	MxP86	IO_L16P_N5P2	–
MxP33	IO_L16N_N5P3	DQ16	MxP87	IO_L16N_N5P3	–
MxP34	IO_L17P_N5P4	DQ17	MxP88	IO_L17P_N5P4	–
MxP35	IO_L17P_N5P5	DM1	MxP89	IO_L17P_N5P5	–

Table 4: **1x18 Pinout** *(cont'd)*

x18 Bank 1			x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP36	IO_L18P_XCC_N6P0	A12	MxP90	IO_L18P_XCC_N6P0	–
MxP37	IO_L18N_XCC_N6P1	A13	MxP91	IO_L18N_XCC_N6P1	–
MxP38	IO_L19P_N6P2	A14	MxP92	IO_L19P_N6P2	–
MxP39	IO_L19N_N6P3	A15	MxP93	IO_L19N_N6P3	–
MxP40	IO_L20P_N6P4	A16	MxP94	IO_L20P_N6P4	–
MxP41	IO_L20N_N6P5	A17	MxP95	IO_L20N_N6P5	–
MxP42	IO_L21P_XCC_N7P0	A18	MxP96	IO_L21P_XCC_N7P0	–
MxP43	IO_L21N_XCC_N7P1	A19	MxP97	IO_L21N_XCC_N7P1	–
MxP44	IO_L22P_N7P2	A20	MxP98	IO_L22P_N7P2	–
MxP45	IO_L22N_N7P3	A21	MxP99	IO_L22N_N7P3	–
MxP46	IO_L23P_N7P4	CSn	MxP100	IO_L23P_N7P4	–
MxP47	IO_L23N_N7P5	Wen	MxP101	IO_L23N_N7P5	–
MxP48	IO_L24P_GC_XCC_N8P0	DK0p	MxP102	IO_L24P_GC_XCC_N8P0	–
MxP49	IO_L24N_GC_XCC_N8P1	DK0n	MxP103	IO_L24N_GC_XCC_N8P1	–
MxP50	IO_L25P_N8P2	CKp	MxP104	IO_L25P_N8P2	–
MxP51	IO_L25N_N8P3	CKn	MxP105	IO_L25N_N8P3	–
MxP52	IO_L26P_N8P4	DK1p	MxP106	IO_L26P_N8P4	–
MxP53	IO_L26N_N8P5	DK1n	MxP107	IO_L26N_N8P5	–

2x18 Pinout

The following table shows an example of a 2x18 RLDRAM 3 interface contained within two banks.

Table 5: **2x18 Pinout**

2x18 Bank 1			2x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP0	IO_L0P_XCC_N0P0	DQ0	MxP54	IO_L0P_XCC_N0P0	A0
MxP1	IO_L0N_XCC_N0P1	DQ1	MxP55	IO_L0N_XCC_N0P1	A1
MxP2	IO_L1P_N0P2	DQ2	MxP56	IO_L1P_N0P2	A2
MxP3	IO_L1N_N0P3	DQ3	MxP57	IO_L1N_N0P3	A3
MxP4	IO_L2P_N0P4	DQ4	MxP58	IO_L2P_N0P4	A4
MxP5	IO_L2N_N0P5	DQ5	MxP59	IO_L2N_N0P5	A5

Table 5: **2x18 Pinout** *(cont'd)*

2x18 Bank 1			2x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP6	IO_L3P_XCC_N1P0	QK0p	MxP60	IO_L3P_XCC_N1P0	A6
MxP7	IO_L3N_XCC_N1P1	QK0n	MxP61	IO_L3N_XCC_N1P1	A7
MxP8	IO_L4P_N1P2	DQ6	MxP62	IO_L4P_N1P2	A8
MxP9	IO_L4N_N1P3	DQ7	MxP63	IO_L4N_N1P3	A9
MxP10	IO_L5P_N1P4	DQ8	MxP64	IO_L5P_N1P4	A10
MxP11	IO_L5N_N1P5	DM0	MxP65	IO_L5N_N1P5	A11
MxP12	IO_L6P_GC_XCC_N2P0	DQ18	MxP66	IO_L6P_GC_XCC_N2P0	A12
MxP13	IO_L6N_GC_XCC_N2P1	DQ19	MxP67	IO_L6N_GC_XCC_N2P1	A13
MxP14	IO_L7P_N2P2	DQ20	MxP68	IO_L7P_N2P2	A14
MxP15	IO_L7N_N2P3	DQ21	MxP69	IO_L7N_N2P3	A15
MxP16	IO_L8P_N2P4	DQ22	MxP70	IO_L8P_N2P4	A16
MxP17	IO_L8N_N2P5	DQ23	MxP71	IO_L8N_N2P5	A17
MxP18	IO_L9P_GC_XCC_N3P0	QK2p	MxP72	IO_L9P_GC_XCC_N3P0	A18
MxP19	IO_L9N_GC_XCC_N3P1	QK2n	MxP73	IO_L9N_GC_XCC_N3P1	A19
MxP20	IO_L10P_N3P2	DQ24	MxP74	IO_L10P_N3P2	A20
MxP21	IO_L10N_N3P3	DQ25	MxP75	IO_L10N_N3P3	A21
MxP22	IO_L11P_N3P4	DQ26	MxP76	IO_L11P_N3P4	DK0p
MxP23	IO_L11N_N3P5	DM2	MxP77	IO_L11N_N3P5	DK0n
MxP24	IO_L12P_GC_XCC_N4P0	DQ9	MxP78	IO_L12P_GC_XCC_N4P0	DK3p
MxP25	IO_L12N_GC_XCC_N4P1	DQ10	MxP79	IO_L12N_GC_XCC_N4P1	DK3n
MxP26	IO_L13P_N4P2	DQ11	MxP80	IO_L13P_N4P2	–
MxP27	IO_L13N_N4P3	DQ12	MxP81	IO_L13N_N4P3	–
MxP28	IO_L14P_N4P4	DQ13	MxP82	IO_L14P_N4P4	CSn
MxP29	IO_L14N_N4P5	DQ14	MxP83	IO_L14N_N4P5	Wen
MxP30	IO_L15P_XCC_N5P0	QK1p	MxP84	IO_L15P_XCC_N5P0	REFn
MxP31	IO_L15N_XCC_N5P1	QK1n	MxP85	IO_L15N_XCC_N5P1	RESETn
MxP32	IO_L16P_N5P2	DQ15	MxP86	IO_L16P_N5P2	BA0
MxP33	IO_L16N_N5P3	DQ16	MxP87	IO_L16N_N5P3	BA1
MxP34	IO_L17P_N5P4	DQ17	MxP88	IO_L17P_N5P4	BA2
MxP35	IO_L17P_N5P5	DM1	MxP89	IO_L17P_N5P5	BA3
MxP36	IO_L18P_XCC_N6P0	DQ27	MxP90	IO_L18P_XCC_N6P0	–
MxP37	IO_L18N_XCC_N6P1	DQ28	MxP91	IO_L18N_XCC_N6P1	–
MxP38	IO_L19P_N6P2	DQ29	MxP92	IO_L19P_N6P2	–
MxP39	IO_L19N_N6P3	DQ30	MxP93	IO_L19N_N6P3	–

Table 5: **2x18 Pinout** *(cont'd)*

2x18 Bank 1			2x18 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP40	IO_L20P_N6P4	DQ31	MxP94	IO_L20P_N6P4	–
MxP41	IO_L20N_N6P5	DQ32	MxP95	IO_L20N_N6P5	–
MxP42	IO_L21P_XCC_N7P0	QK3p	MxP96	IO_L21P_XCC_N7P0	–
MxP43	IO_L21N_XCC_N7P1	QK3n	MxP97	IO_L21N_XCC_N7P1	–
MxP44	IO_L22P_N7P2	DQ33	MxP98	IO_L22P_N7P2	–
MxP45	IO_L22N_N7P3	DQ34	MxP99	IO_L22N_N7P3	–
MxP46	IO_L23P_N7P4	DQ35	MxP100	IO_L23P_N7P4	–
MxP47	IO_L23N_N7P5	DM3	MxP101	IO_L23N_N7P5	–
MxP48	IO_L24P_GC_XCC_N8P0	–	MxP102	IO_L24P_GC_XCC_N8P0	DK1p
MxP49	IO_L24N_GC_XCC_N8P1	–	MxP103	IO_L24N_GC_XCC_N8P1	DK1n
MxP50	IO_L25P_N8P2	–	MxP104	IO_L25P_N8P2	CKp
MxP51	IO_L25N_N8P3	–	MxP105	IO_L25N_N8P3	CKn
MxP52	IO_L26P_N8P4	–	MxP106	IO_L26P_N8P4	DK2p
MxP53	IO_L26N_N8P5	–	MxP107	IO_L26N_N8P5	DK2n

1x36 Pinout

The following table shows an example of a 1x36 RLDRAM 3 interface contained within two banks.

Table 6: **1x36 Pinout**

1x36 Bank 1			1x36 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP0	IO_L0P_XCC_N0P0	DQ0	MxP54	IO_L0P_XCC_N0P0	A0
MxP1	IO_L0N_XCC_N0P1	DQ1	MxP55	IO_L0N_XCC_N0P1	A1
MxP2	IO_L1P_N0P2	DQ2	MxP56	IO_L1P_N0P2	A2
MxP3	IO_L1N_N0P3	DQ3	MxP57	IO_L1N_N0P3	A3
MxP4	IO_L2P_N0P4	DQ4	MxP58	IO_L2P_N0P4	A4
MxP5	IO_L2N_N0P5	DQ5	MxP59	IO_L2N_N0P5	A5
MxP6	IO_L3P_XCC_N1P0	QK0p	MxP60	IO_L3P_XCC_N1P0	A6
MxP7	IO_L3N_XCC_N1P1	QK0n	MxP61	IO_L3N_XCC_N1P1	A7
MxP8	IO_L4P_N1P2	DQ6	MxP62	IO_L4P_N1P2	A8
MxP9	IO_L4N_N1P3	DQ7	MxP63	IO_L4N_N1P3	A9

Table 6: **1x36 Pinout** *(cont'd)*

1x36 Bank 1			1x36 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP10	IO_L5P_N1P4	DQ8	MxP64	IO_L5P_N1P4	A10
MxP11	IO_L5N_N1P5	DM0	MxP65	IO_L5N_N1P5	A11
MxP12	IO_L6P_GC_XCC_N2P0	DQ18	MxP66	IO_L6P_GC_XCC_N2P0	A12
MxP13	IO_L6N_GC_XCC_N2P1	DQ19	MxP67	IO_L6N_GC_XCC_N2P1	A13
MxP14	IO_L7P_N2P2	DQ20	MxP68	IO_L7P_N2P2	A14
MxP15	IO_L7N_N2P3	DQ21	MxP69	IO_L7N_N2P3	A15
MxP16	IO_L8P_N2P4	DQ22	MxP70	IO_L8P_N2P4	A16
MxP17	IO_L8N_N2P5	DQ23	MxP71	IO_L8N_N2P5	A17
MxP18	IO_L9P_GC_XCC_N3P0	QK2p	MxP72	IO_L9P_GC_XCC_N3P0	A18
MxP19	IO_L9N_GC_XCC_N3P1	QK2n	MxP73	IO_L9N_GC_XCC_N3P1	A19
MxP20	IO_L10P_N3P2	DQ24	MxP74	IO_L10P_N3P2	A20
MxP21	IO_L10N_N3P3	DQ25	MxP75	IO_L10N_N3P3	A21
MxP22	IO_L11P_N3P4	DQ26	MxP76	IO_L11P_N3P4	CSn
MxP23	IO_L11N_N3P5		MxP77	IO_L11N_N3P5	Wen
MxP24	IO_L12P_GC_XCC_N4P0	DQ9	MxP78	IO_L12P_GC_XCC_N4P0	–
MxP25	IO_L12N_GC_XCC_N4P1	DQ10	MxP79	IO_L12N_GC_XCC_N4P1	–
MxP26	IO_L13P_N4P2	DQ11	MxP80	IO_L13P_N4P2	–
MxP27	IO_L13N_N4P3	DQ12	MxP81	IO_L13N_N4P3	–
MxP28	IO_L14P_N4P4	DQ13	MxP82	IO_L14P_N4P4	–
MxP29	IO_L14N_N4P5	DQ14	MxP83	IO_L14N_N4P5	–
MxP30	IO_L15P_XCC_N5P0	QK1p	MxP84	IO_L15P_XCC_N5P0	–
MxP31	IO_L15N_XCC_N5P1	QK1n	MxP85	IO_L15N_XCC_N5P1	–
MxP32	IO_L16P_N5P2	DQ15	MxP86	IO_L16P_N5P2	–
MxP33	IO_L16N_N5P3	DQ16	MxP87	IO_L16N_N5P3	–
MxP34	IO_L17P_N5P4	DQ17	MxP88	IO_L17P_N5P4	–
MxP35	IO_L17P_N5P5	DM1	MxP89	IO_L17P_N5P5	–
MxP36	IO_L18P_XCC_N6P0	DQ27	MxP90	IO_L18P_XCC_N6P0	–
MxP37	IO_L18N_XCC_N6P1	DQ28	MxP91	IO_L18N_XCC_N6P1	–
MxP38	IO_L19P_N6P2	DQ29	MxP92	IO_L19P_N6P2	–
MxP39	IO_L19N_N6P3	DQ30	MxP93	IO_L19N_N6P3	–
MxP40	IO_L20P_N6P4	DQ31	MxP94	IO_L20P_N6P4	–
MxP41	IO_L20N_N6P5	DQ32	MxP95	IO_L20N_N6P5	–
MxP42	IO_L21P_XCC_N7P0	QK3p	MxP96	IO_L21P_XCC_N7P0	–
MxP43	IO_L21N_XCC_N7P1	QK3n	MxP97	IO_L21N_XCC_N7P1	–

Table 6: **1x36 Pinout** *(cont'd)*

1x36 Bank 1			1x36 Bank 2		
Triplet#Pin #	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port A Signals**	**Triplet#Pin #**	**Package Pin Name Nibble#NibblePin# (_NxPy)**	**Port B Signals**
MxP44	IO_L22P_N7P2	DQ33	MxP98	IO_L22P_N7P2	–
MxP45	IO_L22N_N7P3	DQ34	MxP99	IO_L22N_N7P3	–
MxP46	IO_L23P_N7P4	DQ35	MxP100	IO_L23P_N7P4	–
MxP47	IO_L23N_N7P5		MxP101	IO_L23N_N7P5	–
MxP48	IO_L24P_GC_XCC_N8P0	DK0p	MxP102	IO_L24P_GC_XCC_N8P0	REFn
MxP49	IO_L24N_GC_XCC_N8P1	DK0n	MxP103	IO_L24N_GC_XCC_N8P1	RESETn
MxP50	IO_L25P_N8P2	CKp	MxP104	IO_L25P_N8P2	BA0
MxP51	IO_L25N_N8P3	CKn	MxP105	IO_L25N_N8P3	BA1
MxP52	IO_L26P_N8P4	DK1p	MxP106	IO_L26P_N8P4	BA2
MxP53	IO_L26N_N8P5	DK1n	MxP107	IO_L26N_N8P5	BA3

2x36 Pinout

The following table shows an example of a 2x36 RLDRAM 3 interface contained within three banks.

Table 7: **2x36 Pinout**

2x36 Bank 1			2x36 Bank 2			2x36 Bank 3		
Triplet #Pin#	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port A Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Address/ Control Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port B Signals**
MxP0	IO_L0P_XCC_N0P0	DQ0	MxP54	IO_L0P_XCC_N0P0	A0	MxP108	IO_L0P_XCC_N0P0	DQ36
MxP1	IO_L0N_XCC_N0P1	DQ1	MxP55	IO_L0N_XCC_N0P1	A1	MxP109	IO_L0N_XCC_N0P1	DQ37
MxP2	IO_L1P_N0P2	DQ2	MxP56	IO_L1P_N0P2	A2	MxP110	IO_L1P_N0P2	DQ38
MxP3	IO_L1N_N0P3	DQ3	MxP57	IO_L1N_N0P3	A3	MxP111	IO_L1N_N0P3	DQ39
MxP4	IO_L2P_N0P4	DQ4	MxP58	IO_L2P_N0P4	A4	MxP112	IO_L2P_N0P4	DQ40
MxP5	IO_L2N_N0P5	DQ5	MxP59	IO_L2N_N0P5	A5	MxP113	IO_L2N_N0P5	DQ41
MxP6	IO_L3P_XCC_N1P0	QK0p	MxP60	IO_L3P_XCC_N1P0	A6	MxP114	IO_L3P_XCC_N1P0	QK0p
MxP7	IO_L3N_XCC_N1P1	QK0n	MxP61	IO_L3N_XCC_N1P1	A7	MxP115	IO_L3N_XCC_N1P1	QK0n

Table 7: **2x36 Pinout** *(cont'd)*

2x36 Bank 1			2x36 Bank 2			2x36 Bank 3		
Triplet #Pin#	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port A Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Address/ Control Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port B Signals**
MxP8	IO_L4P_N1P2	DQ6	MxP62	IO_L4P_N1P2	A8	MxP116	IO_L4P_N1P2	DQ42
MxP9	IO_L4N_N1P3	DQ7	MxP63	IO_L4N_N1P3	A9	MxP117	IO_L4N_N1P3	DQ43
MxP10	IO_L5P_N1P4	DQ8	MxP64	IO_L5P_N1P4	A10	MxP118	IO_L5P_N1P4	DQ44
MxP11	IO_L5N_N1P5	DM0	MxP65	IO_L5N_N1P5	A11	MxP119	IO_L5N_N1P5	DM2
MxP12	IO_L6P_GC_XCC_N2P0	DQ18	MxP66	IO_L6P_GC_XCC_N2P0	A12	MxP120	IO_L6P_GC_XCC_N2P0	DQ54
MxP13	IO_L6N_GC_XCC_N2P1	DQ19	MxP67	IO_L6N_GC_XCC_N2P1	A13	MxP121	IO_L6N_GC_XCC_N2P1	DQ55
MxP14	IO_L7P_N2P2	DQ20	MxP68	IO_L7P_N2P2	A14	MxP122	IO_L7P_N2P2	DQ56
MxP15	IO_L7N_N2P3	DQ21	MxP69	IO_L7N_N2P3	A15	MxP123	IO_L7N_N2P3	DQ57
MxP16	IO_L8P_N2P4	DQ22	MxP70	IO_L8P_N2P4	A16	MxP124	IO_L8P_N2P4	DQ58
MxP17	IO_L8N_N2P5	DQ23	MxP71	IO_L8N_N2P5	A17	MxP125	IO_L8N_N2P5	DQ59
MxP18	IO_L9P_GC_XCC_N3P0	QK2p	MxP72	IO_L9P_GC_XCC_N3P0	A18	MxP126	IO_L9P_GC_XCC_N3P0	QK2p
MxP19	IO_L9N_GC_XCC_N3P1	QK2n	MxP73	IO_L9N_GC_XCC_N3P1	A19	MxP127	IO_L9N_GC_XCC_N3P1	QK2n
MxP20	IO_L10P_N3P2	DQ24	MxP74	IO_L10P_N3P2	A20	MxP128	IO_L10P_N3P2	DQ60
MxP21	IO_L10N_N3P3	DQ25	MxP75	IO_L10N_N3P3	A21	MxP129	IO_L10N_N3P3	DQ61
MxP22	IO_L11P_N3P4	DQ26	MxP76	IO_L11P_N3P4	DK0p	MxP130	IO_L11P_N3P4	DQ62
MxP23	IO_L11N_N3P5		MxP77	IO_L11N_N3P5	DK0n	MxP131	IO_L11N_N3P5	
MxP24	IO_L12P_GC_XCC_N4P0	DQ9	MxP78	IO_L12P_GC_XCC_N4P0	DK3p	MxP132	IO_L12P_GC_XCC_N4P0	DQ45
MxP25	IO_L12N_GC_XCC_N4P1	DQ10	MxP79	IO_L12N_GC_XCC_N4P1	DK3n	MxP133	IO_L12N_GC_XCC_N4P1	DQ46
MxP26	IO_L13P_N4P2	DQ11	MxP80	IO_L13P_N4P2	–	MxP134	IO_L13P_N4P2	DQ47

Table 7: **2x36 Pinout** *(cont'd)*

2x36 Bank 1			2x36 Bank 2			2x36 Bank 3		
Triplet #Pin#	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port A Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Address/ Control Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port B Signals**
MxP27	IO_L13N_N4P3	DQ12	MxP81	IO_L13N_N4P3	–	MxP135	IO_L13N_N4P3	DQ48
MxP28	IO_L14P_N4P4	DQ13	MxP82	IO_L14P_N4P4	CSn	MxP136	IO_L14P_N4P4	DQ49
MxP29	IO_L14N_N4P5	DQ14	MxP83	IO_L14N_N4P5	Wen	MxP137	IO_L14N_N4P5	DQ50
MxP30	IO_L15P_XCC_N5P0	QK1p	MxP84	IO_L15P_XCC_N5P0	REFn	MxP138	IO_L15P_XCC_N5P0	QK1p
MxP31	IO_L15N_XCC_N5P1	QK1n	MxP85	IO_L15N_XCC_N5P1	RESETn	MxP139	IO_L15N_XCC_N5P1	QK1n
MxP32	IO_L16P_N5P2	DQ15	MxP86	IO_L16P_N5P2	BA0	MxP140	IO_L16P_N5P2	DQ51
MxP33	IO_L16N_N5P3	DQ16	MxP87	IO_L16N_N5P3	BA1	MxP141	IO_L16N_N5P3	DQ52
MxP34	IO_L17P_N5P4	DQ17	MxP88	IO_L17P_N5P4	BA2	MxP142	IO_L17P_N5P4	DQ53
MxP35	IO_L17P_N5P5	DM1	MxP89	IO_L17P_N5P5	BA3	MxP143	IO_L17P_N5P5	DM3
MxP36	IO_L18P_XCC_N6P0	DQ27	MxP90	IO_L18P_XCC_N6P0	–	MxP144	IO_L18P_XCC_N6P0	DQ63
MxP37	IO_L18N_XCC_N6P1	DQ28	MxP91	IO_L18N_XCC_N6P1	–	MxP145	IO_L18N_XCC_N6P1	DQ64
MxP38	IO_L19P_N6P2	DQ29	MxP92	IO_L19P_N6P2	–	MxP146	IO_L19P_N6P2	DQ65
MxP39	IO_L19N_N6P3	DQ30	MxP93	IO_L19N_N6P3	–	MxP147	IO_L19N_N6P3	DQ66
MxP40	IO_L20P_N6P4	DQ31	MxP94	IO_L20P_N6P4	–	MxP148	IO_L20P_N6P4	DQ67
MxP41	IO_L20N_N6P5	DQ32	MxP95	IO_L20N_N6P5	–	MxP149	IO_L20N_N6P5	DQ68
MxP42	IO_L21P_XCC_N7P0	QK3p	MxP96	IO_L21P_XCC_N7P0	–	MxP150	IO_L21P_XCC_N7P0	QK3p
MxP43	IO_L21N_XCC_N7P1	QK3n	MxP97	IO_L21N_XCC_N7P1	–	MxP151	IO_L21N_XCC_N7P1	QK3n
MxP44	IO_L22P_N7P2	DQ33	MxP98	IO_L22P_N7P2	–	MxP152	IO_L22P_N7P2	DQ69
MxP45	IO_L22N_N7P3	DQ34	MxP99	IO_L22N_N7P3	–	MxP153	IO_L22N_N7P3	DQ70

Table 7: **2x36 Pinout** *(cont'd)*

2x36 Bank 1			2x36 Bank 2			2x36 Bank 3		
Triplet #Pin#	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port A Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Address/ Control Signals**	**Triplet #Pin#**	**Package Pin Name Nibble#Nibble Pin# (_NxPy)**	**Port B Signals**
MxP46	IO_L23P_N7P4	DQ35	MxP100	IO_L23P_N7P4	–	MxP154	IO_L23P_N7P4	DQ71
MxP47	IO_L23N_N7P5	–	MxP101	IO_L23N_N7P5	–	MxP155	IO_L23N_N7P5	–
MxP48	IO_L24P_GC_XCC_N8P0	–	MxP102	IO_L24P_GC_XCC_N8P0	DK1p	MxP156	IO_L24P_GC_XCC_N8P0	–
MxP49	IO_L24N_GC_XCC_N8P1	–	MxP103	IO_L24N_GC_XCC_N8P1	DK1n	MxP157	IO_L24N_GC_XCC_N8P1	–
MxP50	IO_L25P_N8P2	–	MxP104	IO_L25P_N8P2	CKp	MxP158	IO_L25P_N8P2	–
MxP51	IO_L25N_N8P3	–	MxP105	IO_L25N_N8P3	CKn	MxP159	IO_L25N_N8P3	–
MxP52	IO_L26P_N8P4	–	MxP106	IO_L26P_N8P4	DK2p	MxP160	IO_L26P_N8P4	–
MxP53	IO_L26N_N8P5	–	MxP107	IO_L26N_N8P5	DK2n	MxP161	IO_L26N_N8P5	–

Protocol Description

The core has the following interfaces:

- Memory Interface
- User Interface
- Physical Interface

Memory Interface

The RLDRAM 3 core is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top-level of the core.

User Interface

The user interface connects to a Versal ACAP user design to the RLDRAM 3 core to simplify interactions between the user design and the external memory device.

***Note*:** IP design flow is not supported for RLDRAM 3 Memory IP core.

User Interface Signals

The user interface provides an interface for you to send commands to the controller and receive the response for the same. The number of commands per user clock accepted by the interface depends on Burst Length (BL) of the memory device. For BL2 four commands per user clock is supported because the user clock to the memory clock ratio is 1:4. Therefore in one user clock, four memory clock transactions are possible. Similarly for BL4 two commands per user clock, and for BL8 one command per user clock is supported. The commands per user clock for the BL2 and BL4 can be mix of write and read commands. With a clock ration of 1:4 there can be four independent command channels for BL2 and two independent command channels for BL4. Each channel can accept any command to any address location. Each channel processes and responds to its commands in the order it was received.

The user interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in the table.

Table 8: **User Interface Signals**

Signal	I/O	Width	Description
sys_clk_p/n	I	1	Primary clock to IP
sys_rst	I	1	Primary Active-High reset to IP
user_cmd_en	I	1	Indicates commands, address, and bank address are valid at the input.
user_cmd	I	If AMUX ON or BL = 2: width = 3x2 Else: width = 3xCMD_PER_CLK	Command input • 3'b001 - Write • 3'b010 - Read • 3'b000 - NOP
user_addr	I	ADDR_WIDTH x CMD_PER_CLK	Memory Address corresponding to user_cmd.
user_ba	I	BANK_WIDTH x CMD_PER_CLK	Memory Bank Address corresponding to user_cmd.
user_wr_en	I	CMD_PER_CLK	Per command channel indication of valid user_wr_data and user_wr_dm when asserted.
user_wr_data	I	MEM_DQ_WIDTH x BL x CMD_PER_CLK	Write data corresponding to write command on the respective channel.
user_wr_dm	I	MEM_DM_WIDTH x BL x CMD_PER_CLK	Write data mask corresponding to write command on the respective channel. Top-level parameter DM_EN must be set to 1 to enable Data Mask support.
user_afifo_empty	O	CMD_PER_CLK	When asserted indicates CMD/Address FIFO is empty.
user_afifo_aempty	O	CMD_PER_CLK	When asserted indicates CMD/Address FIFO is almost empty.
user_afifo_full	O	CMD_PER_CLK	When asserted indicates CMD/Address FIFO is full.
user_afifo_afull	O	CMD_PER_CLK	When asserted indicates CMD/Address FIFO is almost full.
user_wdfifo_empty	O	CMD_PER_CLK	When asserted indicates Write Data FIFO is empty.

Table 8: **User Interface Signals** *(cont'd)*

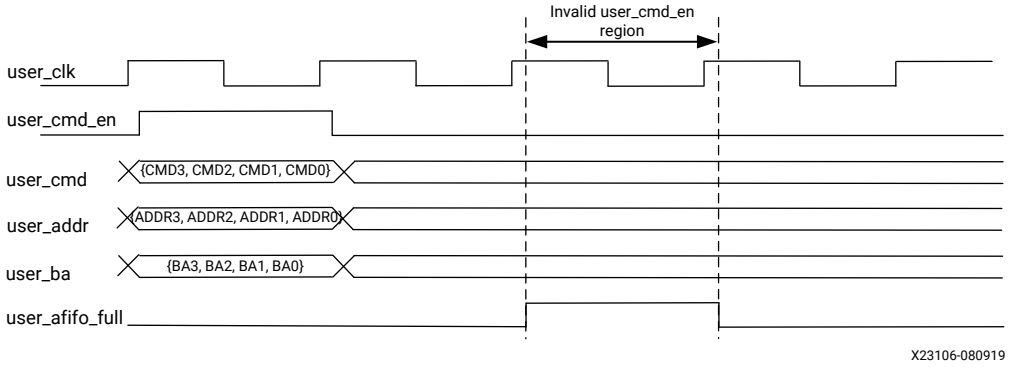
Signal	I/O	Width	Description
user_wdfifo_aempty	O	CMD_PER_CLK	When asserted indicates Write Data FIFO is almost empty.
user_wdfifo_full	O	CMD_PER_CLK	When asserted indicates Write Data FIFO is full.
user_wdfifo_afull	O	CMD_PER_CLK	When asserted indicates Write Data FIFO is almost full.
user_rd_valid	O	CMD_PER_CLK	When asserted indicates user_rd_data is valid on respective channels.
user_rd_data	O	MEM_DQ_WIDTH x BL x CMD_PER_CLK	Read data corresponding to read command issued on respective channels.
init_calib_complete	O	1	When asserted indicates completion of memory initialization and memory interface calibration.
ui_clk	O	1	User interface clock that is 1/4 of RLDRAM 3 clock.
ui_clk_sync_rst	O	1	Active-High user interface reset
calib_error	O	1	When asserted indicates error during calibration.
dbg_clk	O	1	Debug clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.

Interfacing with the Core through the User Interface

The width of certain user interface signals is dependent on the burst length. This allows the client to send multiple commands per Versal ACAP logic clock cycle as might be required for certain configurations.

The user interface protocol for the RLDRAM 3 four-word burst architecture is shown in the following figure.

Figure 6: **User Interface Command Protocol**



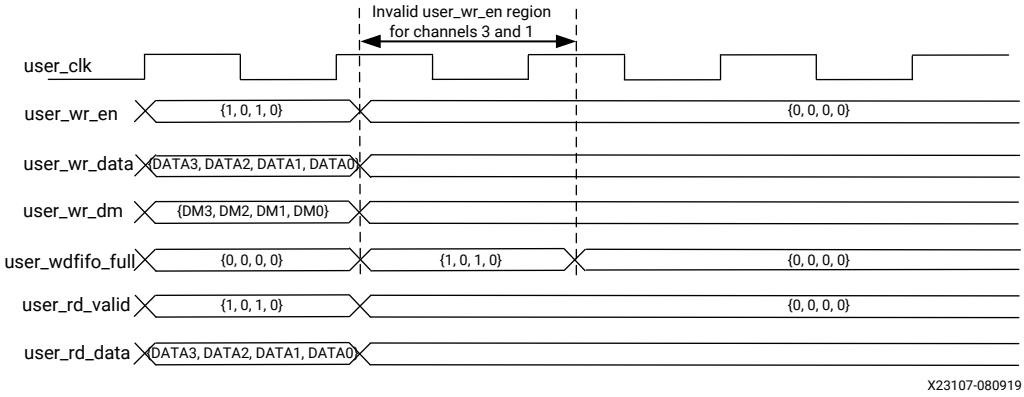
Before any requests can be accepted, the `ui_clk_sync_rst` signal must be deasserted Low. After the `ui_clk_sync_rst` signal is deasserted, the user interface FIFOs can accept commands and data for storage. The `init_calib_complete` signal is asserted after the memory initialization procedure and PHY calibration are complete, and the core can begin to service client requests.

The user interface provides an interface for you to communicate with the RLDRAM 3 controller. To assert any command, you must assert `user_cmd_en` as a single cycle pulse. At this time, `user_cmd`, `user_addr`, and `user_ba` must be valid. If the output signal, `user_afifo_full` is asserted then `user_cmd_en` is ignored and no commands are accepted at that clock edge. The assertion of `user_afifo_full` signal indicates to you that the command/address FIFO is full and no further commands can be accepted. The `user_wr_en` signal, when asserted indicates that the `user_wr_data` and `user_wr_dm` signals must be valid at this time. If the `user_wdiffo_full` signal is 1, the `user_wr_en` signal is ignored. The `user_rd_valid` signal indicates that `user_rd_data` is valid on the respective channel.

The controller performs reordering of the commands before sending them out to the memory device to improve efficiently. Therefore, commands at the memory device might not be in the same order as it was received. However, the read response is in the same order as the input read commands requested by you.

The previous figure shows the protocol for the BL2 interface. With BL2, you are allowed to send four commands at the rising edge of the user clock. When the `user_cmd_en` signal is 1, the `user_cmd`, `user_addr`, and `user_ba` signals are accepted by the user interface. However when the `user_afifo_full` signal is asserted shown as the shaded region, all commands are ignored. The commands at the `user_cmd` signal input can be mix of write, read, NOP, or refresh commands. There are no restrictions which commands can be sent on a particular channel. As shown in the following figure, CMD3 and CMD1 could be write commands, CMD0 could be a read command, and CMD2 could be a NOP command.

Figure 7: **User Interface Write Read Protocol**



The write data (`user_wr_data`) and write data mask (`user_wr_mask`) signals for a write command in a channel must be valid when the `user_wr_en` signal for that channel is asserted. If the corresponding `user_wdfifo_full` signal is 1, then the `user_wr_en` signal of that channel is ignored. As shown in the previous figure, when the `user_wdfifo_full` signal of channels 1 and 3 are asserted write data and mask signals are ignored.

The `user_rd_valid` signal when asserted indicates valid read data (`user_rd_data`) from the corresponding channel. As shown in the previous figure, valid user read data is available on channels 3 and 1, as indicated by `user_rd_valid[3]` and `user_rd_valid[1]` signals.

Also, the same protocol is applicable for BL4 and BL8 interfaces. One difference is that the number of commands per clock or the number of channels will be reduced to 2 and 1 for BL4 and BL8, respectively. The write and read data width per channel will increase by 2 and 4 times for BL4 and BL8, respectively.

Physical Interface

The physical interface is the connection from the Versal ACAP core to an external RLDRAM 3 device. The I/O signals for this interface are defined in the table. These signals can be directly connected to the corresponding signals on the RLDRAM 3 device.

Table 9: **Physical Interface Signals**

Signal	I/O	Description
rld_ck_p	O	System Clock CK. This is the address/command clock to the memory device.
rld_ck_n	O	System Clock CK#. This is the inverted system clock to the memory device.
rld_dk_p	O	Write Clock DK. This is the write clock to the memory device.
rld_dk_n	O	Write Clock DK#. This is the inverted write clock to the memory device.
rld_a	O	Address. This is the address supplied for memory operations.
rld_ba	O	Bank Address. This is the bank address supplied for memory operations.
rld_cs_n	O	Chip Select CS#. This is the active-Low chip select control signal for the memory.
rld_we_n	O	Write Enable WE#. This is the active-Low write enable control signal for the memory.
rld_ref_n	O	Refresh REF#. This is the active-Low refresh control signal for the memory.
rld_dm	O	Data Mask DM. This is the active-High mask signal, driven by the core ACAP to mask data that a user does not want written to the memory during a write command.
rld_dq	I/O	Data DQ. This is a bidirectional data port, driven by the core ACAP for writes and by the memory for reads.
rld_qk_p	I	Read Clock QK. This is the read clock returned from the memory edge aligned with read data on rld_dq. This clock (in conjunction with QK#) is used by the PHY to sample the read data on rld_dq.
rld_qk_n	I	Read Clock QK#. This is the inverted read clock returned from the memory. This clock (in conjunction with QK) is used by the PHY to sample the read data on rld_dq.
rld_reset_n	O	RLDRAM 3 reset pin. This is the active-Low reset to the RLDRAM 3 device.

Table 9: **Physical Interface Signals** *(cont'd)*

Signal	I/O	Description
rld_qvld	I	This active-High data valid port indicates that the valid input data is available on the subsequent rising clock edge.

M and D Support for Reference Input Clock Speed

Memory IPs provide a possibility to select the Reference Input Clock Speed. The value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps). Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).

The required Reference Input Clock Speed is calculated from the M, D, and O values entered in the GUI using the following formulas:

- XPLL_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.
- CLKIN (MHz) = (XPLL_CLKOUT (MHz) × D × O) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.
- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

Calculated Reference Input Clock Speed from M, D, and O values are validated as per clocking guidelines. For more information on clocking rules, see the Clocking section.

Apart from the memory specific clocking rules, validation of the possible XPLL input frequency range and XPLL VCO frequency range values are completed for M, D, and O in the GUI.

For Versal Prime series, see the *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956) for XPLL Input frequency range and XPLL VCO frequency range values.

For Versal AI Core series, see the *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957) for XPLL Input frequency range and XPLL VCO frequency range values.

For possible M, D, and O values and detailed information on clocking and the XPLL, see the *Versal ACAP Clocking Resources Architecture Manual* (AM003).

Related Information

Basic Tab
Clocking

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

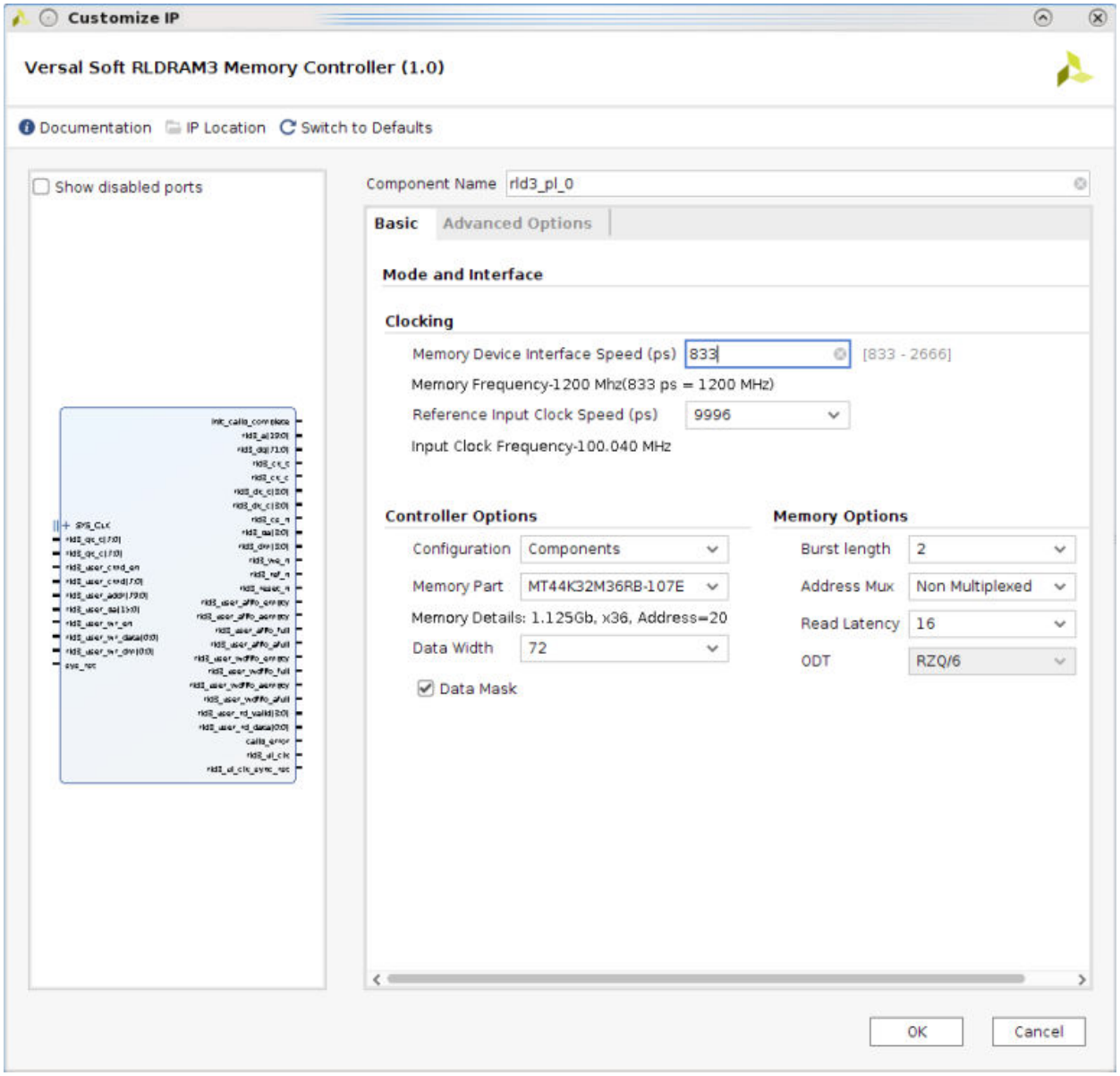
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Basic Tab

The following figure shows the Basic tab.

Figure 8: **Basic Tab**



Select the settings in the **Clocking**, **Controller Options**, and **Memory Options** sections.

- **Clocking:** Memory Device Interface Speed sets the speed of the interface. The speed entered drives the available Reference Input Clock Speeds. For more information on the clocking structure, see the Clocking section.
- **Controller Options:** Selects the Configuration, Memory Part, and Data Width.

- **Memory Options:** Sets the Burst length, Address Mux, Read Latency, and ODT.

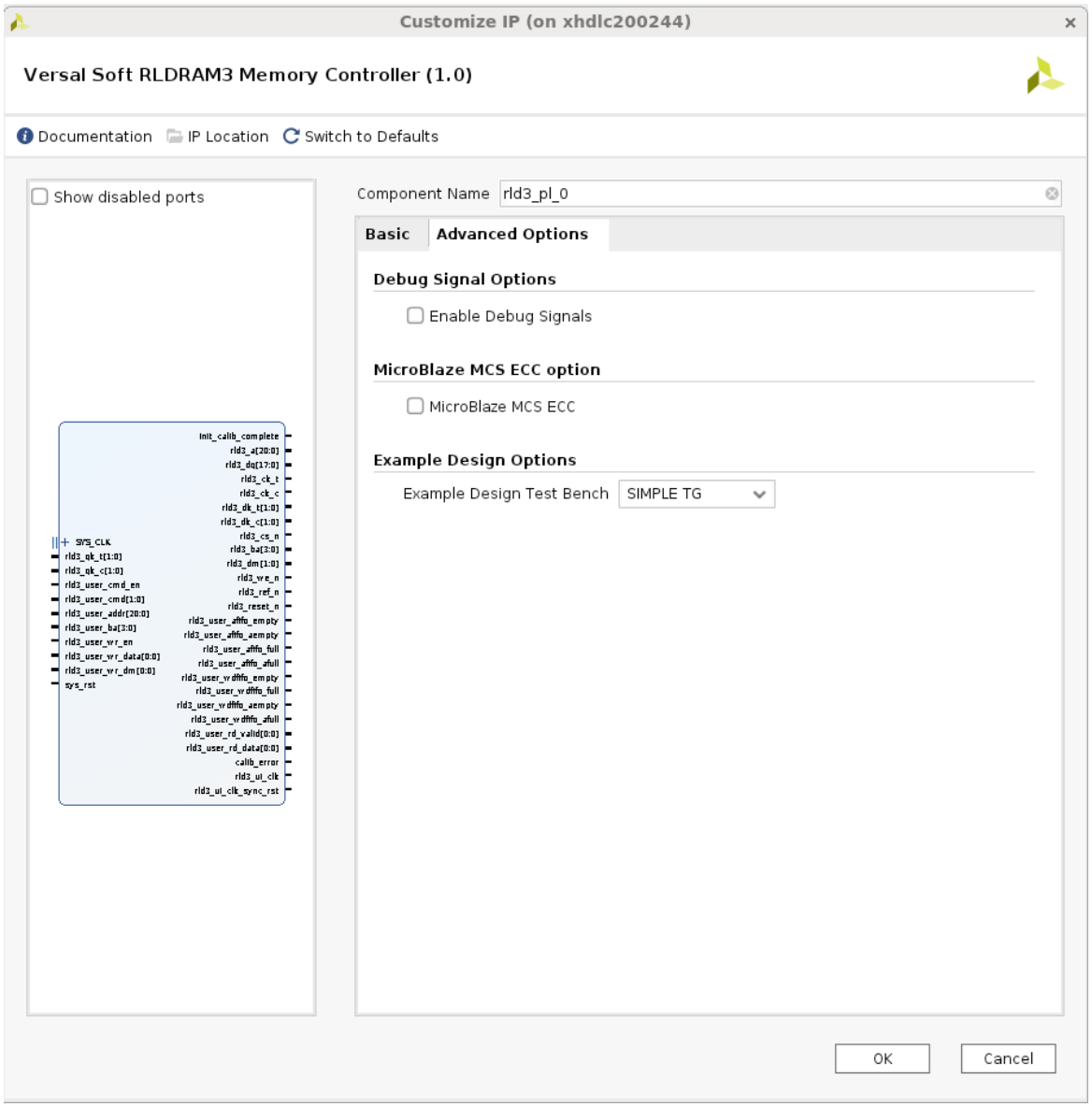
Related Information

Clocking

Advanced Options Tab

The following figure shows the Advanced Options tab. This displays the advanced memory options for the specific controller.

Figure 9: **Advanced Options Tab**



This tab has the following settings:

- **Debug Signal Options:** Enables the debug signals.
- **MicroBlaze MCS ECC option:** Enables the MicroBlaze MCS ECC.
- **Example Design Options:** Allows you to select Traffic Generator Options between Simple and Advanced Traffic Generators.

User Parameters

The following table shows the relationship between the fields in the Vivado® IDE and the user parameters (which can be viewed in the Tcl Console).

Table 10: **User Parameters**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE
Reference Clock	Reference_Clock	FALSE
Enable System Ports	Enable_SysPorts	TRUE
Clock Period (ps)	C0.RLD3_TimePeriod	1,071
Input Clock Period (ps)	C0.RLD3_InputClockPeriod	13,947
General Interconnect to Memory Clock Ratio	C0.RLD3_PhyClockRatio	4:1
Configuration	C0.RLD3_MemoryType	Components
Memory Part	C0.RLD3_MemoryPart	MT44K16M36RB-093
Data Width	C0.RLD3_DataWidth	36
Data Mask	C0.RLD3_DataMask	TRUE
Burst Length	C0.RLD3_BurstLength	8
Memory Voltage	C0.RLD3_MemoryVoltage	1.2

Setting TWTR Check Parameter OFF for RLDRAM 3 Designs

This TWTR_CHECK_OFF switch provides the ability to turn OFF TWTR timing check inside the RLDRAM 3 controller. The default value of TWTR_CHECK parameter for RLDRAM 3 is set to ON. In many cases, it has been observed that some user traffic patterns never execute the TWTR timing. If the TWTR_CHECK_OFF switch is set to OFF, then the whole logic is bypassed. This can potentially help improve timing as well as improved bus efficiency. This can be changed through the Tcl command using the user parameter TWTR_CHECK_OFF for any RLDRAM 3 designs. The table shows details of the TWTR_CHECK_OFF user parameter.

***Note*:** Do not turn this timing check off unless the access pattern will never cause a TWTR failure.

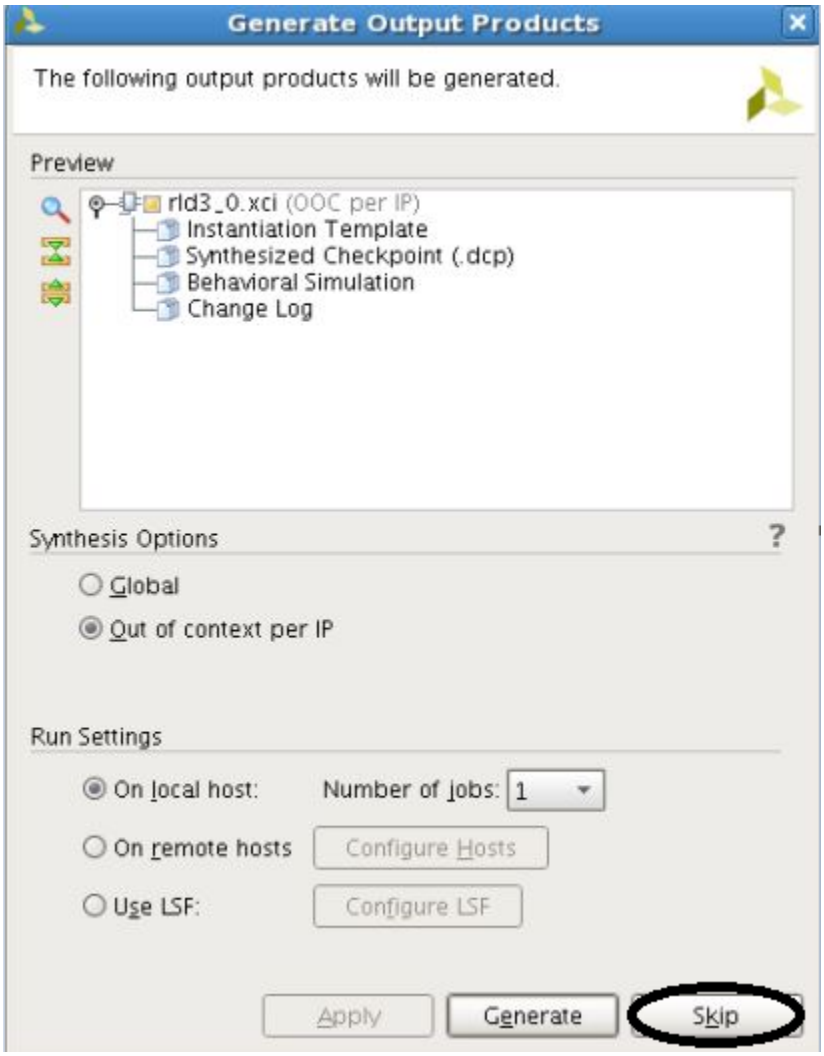
Table 11: **TWTR_CHECK_OFF User Parameter**

User Parameter	Value Format	Default Value	Possible Values
TWTR_CHECK_OFF	String	FALSE	FALSE – TWTR_CHECK parameter set to ON TRUE – TWTR_CHECK parameter set to OFF

Follow these steps to change the TWTR check parameter value.

1. Generate RLDRAM 3 IP.

2. In the **Generate Output Products** option, do not select **Generate** instead select **Skip**.

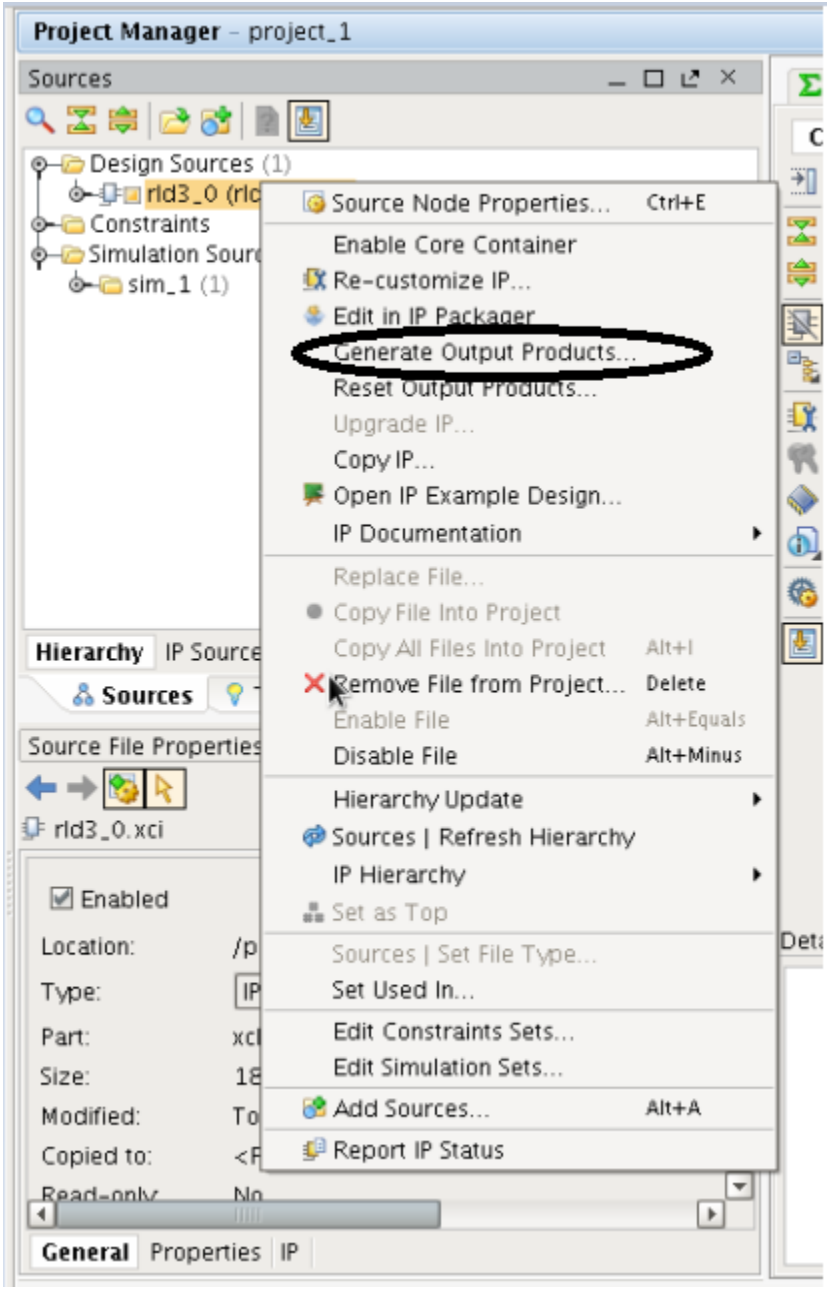


3.
```
set_property -dict [list config.TWTR_CHECK_OFF <value_to_be_set>]
[get_ips <ip_name>]
```

   For example:

```
set_property -dict [list config.TWTR_CHECK_OFF {true}] [get_ips rld3_0]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP. Set the TWTR_CHECK_OFF value by running the following command on the Tcl console:



Set the TWTR_CHECK_OFF value by running the following command on the TclThe generated output files have the TWTR_CHECK parameter value set as per the selected value.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

I/O Planning

RLDRAM 3 I/O pin planning is completed with the full design pin planning using the Vivado I/O Pin Planner. RLDRAM 3 I/O pins can be selected through several Vivado I/O Pin Planner features including assignments using I/O Ports view, Package view, or Memory Bank/Byte Planner. Pin assignments can additionally be made through importing an XDC or modifying the existing XDC file.

These options are available for all RLDRAM 3 designs and multiple RLDRAM 3 IP instances can be completed in one setting. To learn more about the available Memory IP pin planning options, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899).

Constraining the Core

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The RLDRAM 3 tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT! *The* `set_input_delay` *and* `set_output_delay` *constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

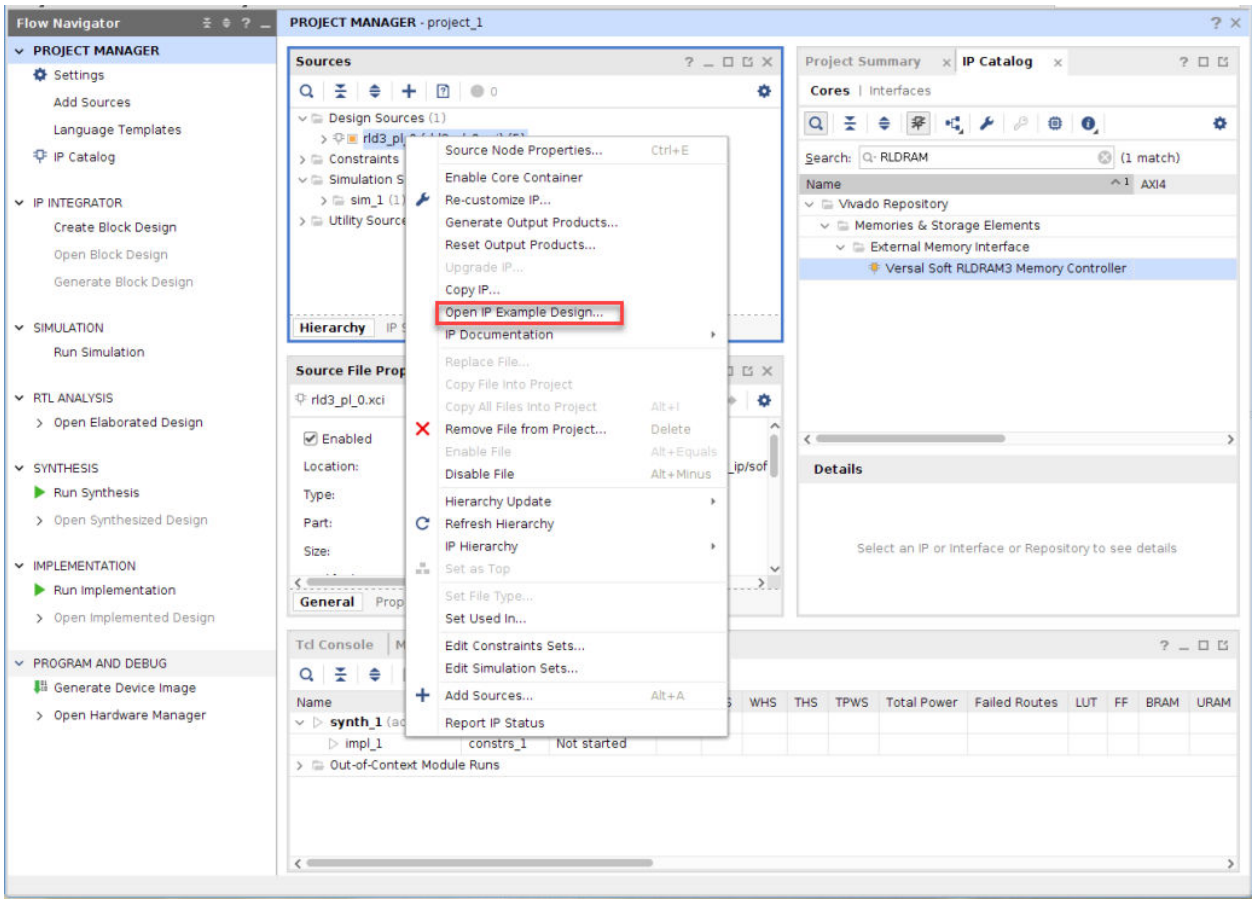
Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Example Design

This section contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in the following figure and select **Open IP Example Design**.

Figure 10: **Open IP Example Design**



This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern to the Memory Controller. This test bench consists of an IP wrapper and an `example_tb` that generates 100 writes and 100 reads.

The example design can be simulated using one of the methods in the following sections.

Project-Based Simulation

This method can be used to simulate the example design using the Vivado Design Suite (IDE). Memory IP delivers IEEE encrypted memory models for RLDRAM 3.

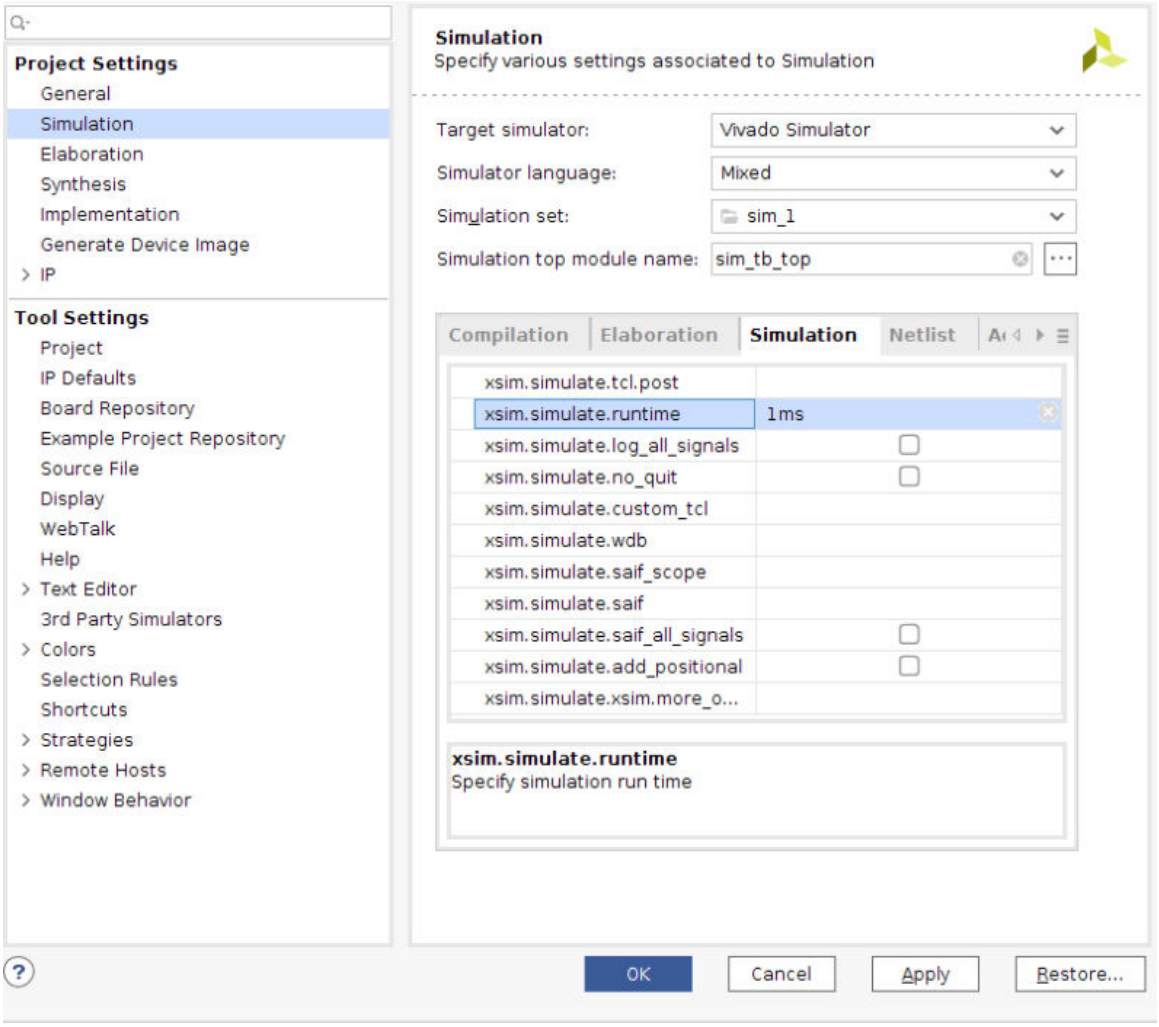
The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for RLDRAM 3 IP verification at each software release. The Vivado simulation tool is used for RLDRAM 3 IP verification from 2020.2 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

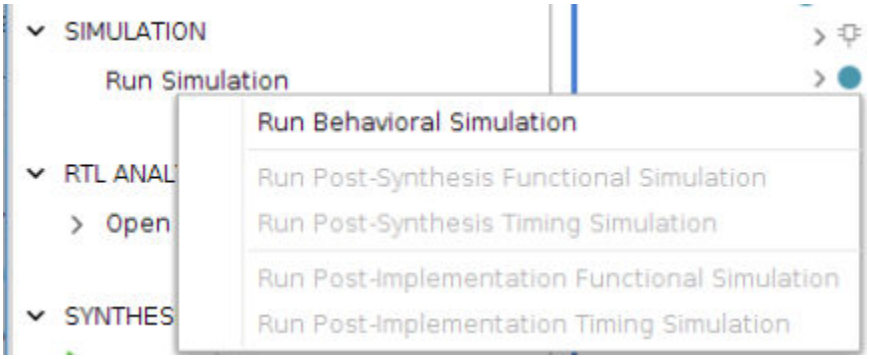
1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the `xsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in the following figure. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Set the **Simulation Language** to **Mixed**.

4. Apply the settings and select **OK**.



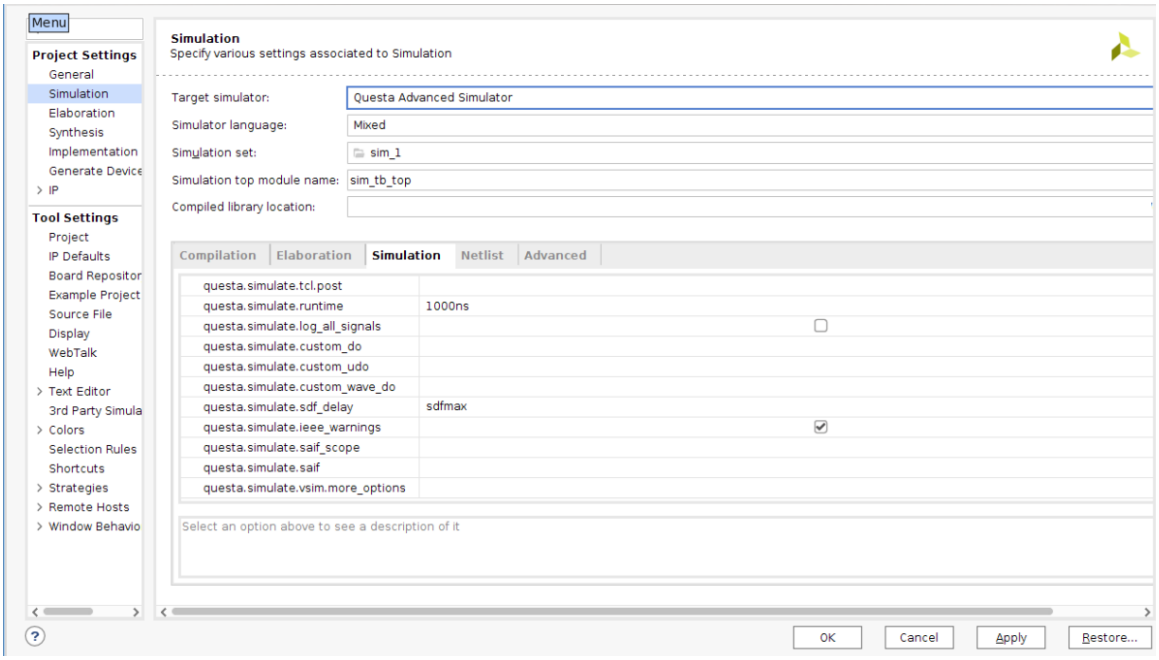
5. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown:



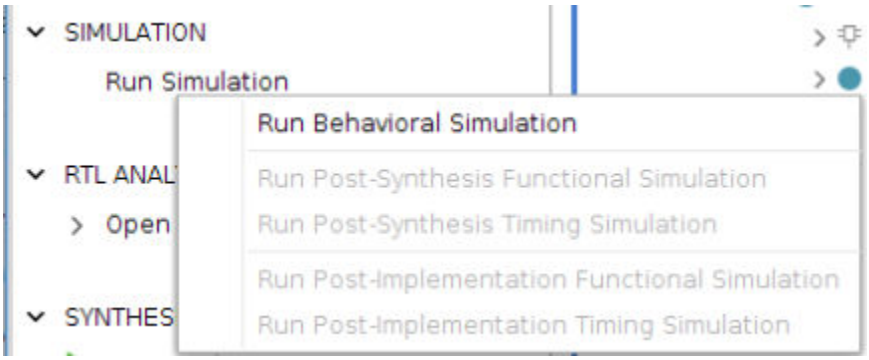
6. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Questa Advanced Simulator**.
   a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
   b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in the following figure. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown:

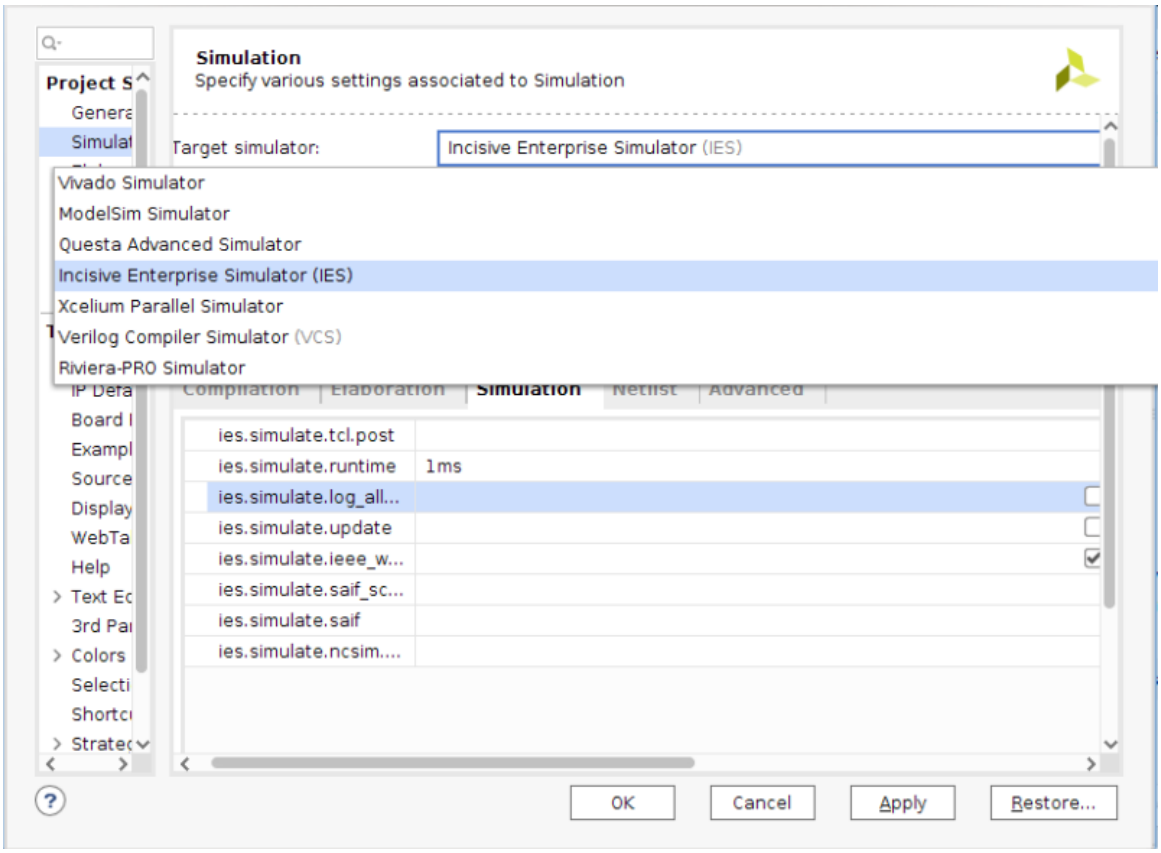


5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

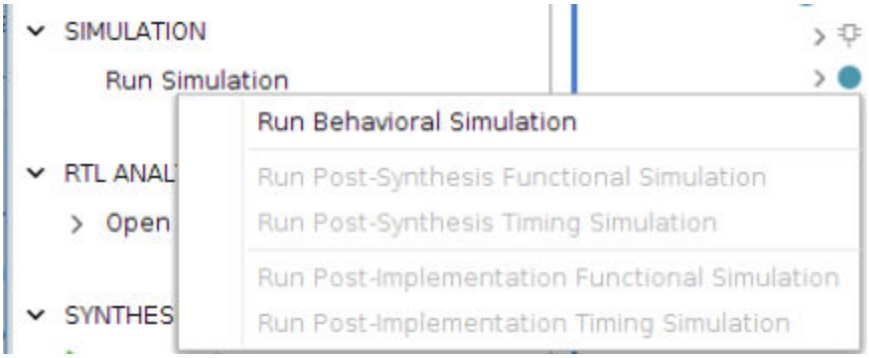
Project-Based Simulation Flow Using IES

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Incisive Enterprise Simulator (IES)**.
   a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
   b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in the following figure. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

3. Apply the settings and select **OK**.



4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown:



5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

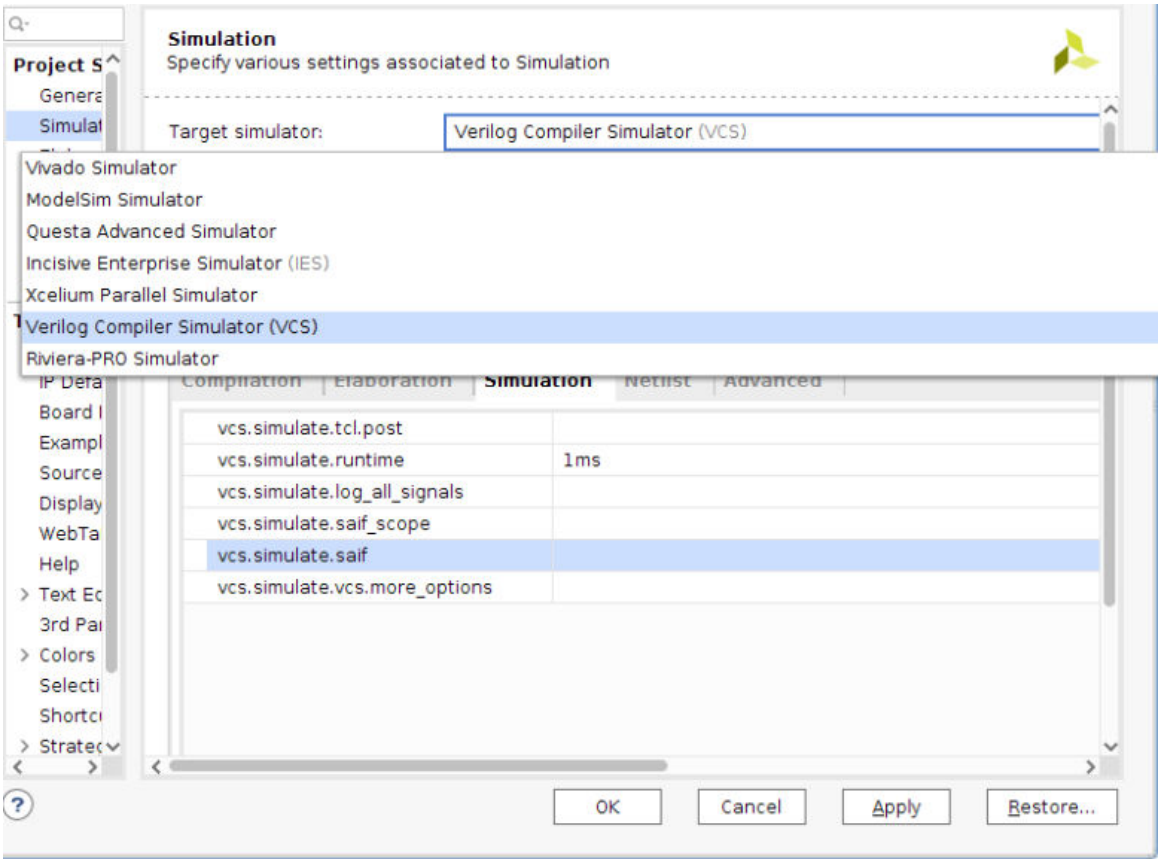
Project-Based Simulation Flow Using VCS

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Verilog Compiler Simulator (VCS)**.

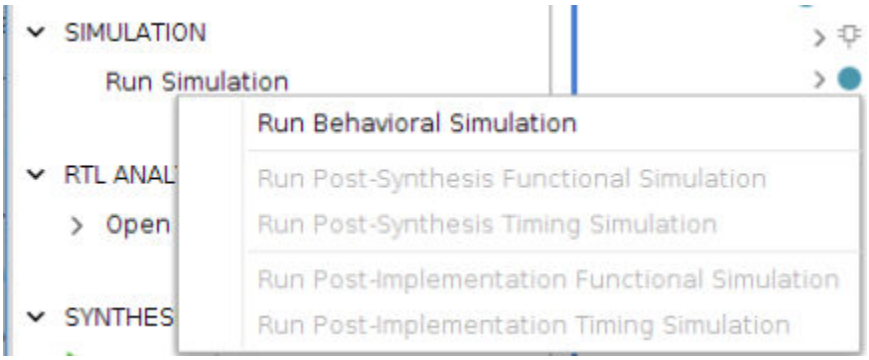
a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.

b. Under the **Simulation** tab, set the `vcs.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in the following figure. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

3. Apply the settings and select **OK**.



4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown:



5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Using Xilinx IP with Third-Party Synthesis Tools

For more information on how to use Xilinx IP with third-party synthesis tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and XPLL are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and XPLL input. RLDRAM 3 manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced** → **Versal ACAP Options** → **Reference Input**). Also, RLDRAM 3 handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced** → **Versal ACAP Options** → **Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and XPLL allocation needs to be handled manually for the IP flow. RLDRAM 3 does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, RLDRAM 3 generates the example design with differential buffer instantiation for system clock pins. RLDRAM 3 generates clock constraints in the `example_design.xdc`. It also generates a CLOCK_DEDICATED_ROUTE constraint as the “BACKBONE” and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and XPLL input if the GCIO and XPLL are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Test Bench

This section contains information about the test bench provided in the Vivado® Design Suite.

The Memory Controller is generated along with a simple test bench to verify the basic read and write operations. The stimulus contains 100 consecutive writes followed by 100 consecutive reads for data integrity check.

XILINX

Appendix A

Upgrading

This appendix is not applicable for the first release of the core.

Appendix B

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Xilinx Community Forums are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address RLDRAM 3 design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx® devices.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956)
2. *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)
3. *Versal ACAP SelectIO Resources Architecture Manual* (AM010)
4. *Versal ACAP PCB Design User Guide* (UG863)
5. *Versal ACAP Clocking Resources Architecture Manual* (AM003)
6. *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187)
7. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide: Getting Started* (UG910)
10. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
11. *Vivado Design Suite User Guide: Implementation* (UG904)
12. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/21/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright